# Simulation

Generated by Doxygen 1.9.2

# Chapter 1

# Namespace Index

## 1.1 Packages

Here are the packages with brief descriptions (if available):

# Chapter 2

# Hierarchical Index

## 2.1 Class Hierarchy

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

# Chapter 3

# Class Index

## 3.1 Class List

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

# Chapter 4

# File Index

## 4.1 File List

Here is a list of all files with brief descriptions:

# Chapter 5

# Namespace Documentation

## 5.1 Simulation Namespace Reference

### Classes

- class ClosestEncounterWatcher

  *A class that looks for a closest encounter of two objects in a simulation*
- struct ClosestEncounterWatcherSettings

  *Settings that define how a ClosestEncounterWatcher acts*
- class ConstantStepSimulation

  *A simulation, that has a constant delay between each step.*
- class DynamicWorldPoint

  *A point in a world that can move. That means this object is also defined by Velocity, current Acceleration and its Mass*
- struct Encounter

  *Struct that is used to describe encounter of two objects*
- class Program
- class Simulation

  *A base class used to define how world is simulated*
- class SimulationWatcher

  *Abstract class used to define so called watchers. Watchers are objects that can analyze a world every simulation step. For example a watcher can look for a closest encounter of two objects.*
- class StepEventArgs

  *Class that holds data about the current step*
- class SteppedEventArgs

  *Class that holds data about a step that has been taken*
- class World

  *An object used to hold world objects(see World.Object)*

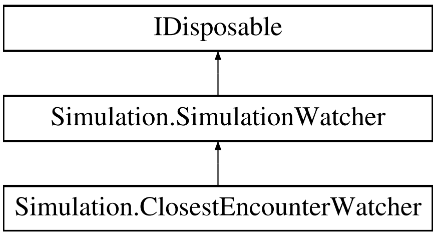
# Chapter 6

# Class Documentation

## 6.1 Simulation.ClosestEncounterWatcher Class Reference

A class that looks for a closest encounter of two objects in a simulation

Inheritance diagram for Simulation.ClosestEncounterWatcher:


IDisposable
Simulation.SimulationWatcher
Simulation.ClosestEncounterWatcher

### Public Member Functions

- ClosestEncounterWatcher (Simulation simulation, World.Object obj1, World.Object obj2)

  *Creates an instance of ClosestEncounterWatcher*
- ClosestEncounterWatcher (Simulation simulation, World.Object obj1, World.Object obj2, ShortestDistanceFinderSettings settings)

  *Creates an instance of ClosestEncounterWatcher*
- void ResetClosestEncounter ()

  *Discards the current closest encounter and starts looking for a new one*

### Static Public Attributes

- static readonly ShortestDistanceFinderSettings DEFAULT_SETTINGS

  *The default settings*

### Protected Member Functions

- override void Simulation_Stepped (object sender, SteppedEventArgs e)

  *Gets invoked after every step in the watched simulation*

## Properties

- Encounter CurrentClosestEncounter `[get]`

  *Latest recorded closest encounter*

## 6.1.1 Detailed Description

A class that looks for a closest encounter of two objects in a simulation

Definition at line 8 of file ClosestEncounterWatcher.cs.

## 6.1.2 Constructor & Destructor Documentation

### 6.1.2.1 ClosestEncounterWatcher() `[1/2]`

```
Simulation.ClosestEncounterWatcher.ClosestEncounterWatcher (
            Simulation simulation,
            World.Object obj1,
            World.Object obj2 )
```

Creates an instance of ClosestEncounterWatcher

**Parameters**

| | |
|---|---|
| *simulation* | The simulation on which the watcher should work |
| *obj1* | The first object which the watcher should watch |
| *obj2* | The second object which the watcher should watch |

Definition at line 24 of file ClosestEncounterWatcher.cs.

### 6.1.2.2 ClosestEncounterWatcher() `[2/2]`

```
Simulation.ClosestEncounterWatcher.ClosestEncounterWatcher (
            Simulation simulation,
            World.Object obj1,
            World.Object obj2,
            ShortestDistanceFinderSettings settings )
```

Creates an instance of ClosestEncounterWatcher

**Parameters**

| | |
|---|---|
| *simulation* | The simulation on which the watcher should work |
| *obj1* | The first object which the watcher should watch |
| *obj2* | The second object which the watcher should watch |
| *settings* | Settings defining how the watcher should act |

Definition at line 35 of file ClosestEncounterWatcher.cs.

## 6.1.3 Member Function Documentation

### 6.1.3.1 ResetClosestEncounter()

```
void Simulation.ClosestEncounterWatcher.ResetClosestEncounter ( )
```

Discards the current closest encounter and starts looking for a new one

### 6.1.3.2 Simulation_Stepped()

```
override void Simulation.ClosestEncounterWatcher.Simulation_Stepped (
             object sender,
             SteppedEventArgs e )  [protected], [virtual]
```

Gets invoked after every step in the watched simulation

**Parameters**

| | |
|---|---|
| *sender* | Instance of the object that invoked the step |
| *e* | Instance of SteppedEventArgs describing the state of the simulated world after the step |

Implements Simulation.SimulationWatcher.

Definition at line 58 of file ClosestEncounterWatcher.cs.

## 6.1.4 Member Data Documentation

### 6.1.4.1 DEFAULT_SETTINGS

```
readonly ShortestDistanceFinderSettings Simulation.ClosestEncounterWatcher.DEFAULT_SETTINGS
[static]
```

**Initial value:**

```
= new ShortestDistanceFinderSettings()
        {
            AutoStopSimulation = false
        }
```

The default settings

Definition at line 13 of file ClosestEncounterWatcher.cs.

### 6.1.5 Property Documentation

#### 6.1.5.1 CurrentClosestEncounter

```
Encounter Simulation.ClosestEncounterWatcher.CurrentClosestEncounter  [get]
```

Latest recorded closest encounter

Definition at line 45 of file ClosestEncounterWatcher.cs.

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

- ClosestEncounterWatcher.cs

## 6.2 Simulation.ClosestEncounterWatcherSettings Struct Reference

Settings that define how a ClosestEncounterWatcher acts

### Properties

- bool AutoStopSimulation `[getset]`
  *True, if the watcher should stop the simulation when the objects start to move away*

### 6.2.1 Detailed Description

Settings that define how a ClosestEncounterWatcher acts

Definition at line 6 of file ClosestEncounterWatcherSettings.cs.

### 6.2.2 Property Documentation

#### 6.2.2.1 AutoStopSimulation

```
bool Simulation.ClosestEncounterWatcherSettings.AutoStopSimulation  [get], [set]
```

True, if the watcher should stop the simulation when the objects start to move away

Definition at line 11 of file ClosestEncounterWatcherSettings.cs.

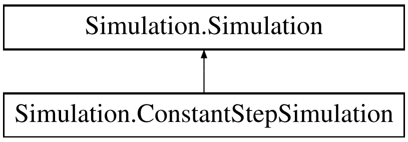
The documentation for this struct was generated from the following file:

- ClosestEncounterWatcherSettings.cs

## 6.3 Simulation.ConstantStepSimulation Class Reference

A simulation, that has a constant delay between each step.

Inheritance diagram for Simulation.ConstantStepSimulation:



### Public Member Functions

- ConstantStepSimulation (World world, TimeSpan stepLength, TimeSpan simulationLength)

  *Creagtes an instance of ConstantStepSimulation*

### Protected Member Functions

- override void DoStep ()

### Additional Inherited Members

### 6.3.1 Detailed Description

A simulation, that has a constant delay between each step.

Definition at line 8 of file StepSimulation.cs.

### 6.3.2 Constructor & Destructor Documentation

#### 6.3.2.1 ConstantStepSimulation()

```
Simulation.ConstantStepSimulation.ConstantStepSimulation (
            World world,
            TimeSpan stepLength,
            TimeSpan simulationLength )
```

Creagtes an instance of ConstantStepSimulation

**Parameters**

| | |
|---|---|
| *world* | A world that will be simulated |
| *stepLength* | A time between each step (smaller value = more precision) |
| *simulationLength* | A total length of simulation |

Definition at line 16 of file StepSimulation.cs.

### 6.3.3 Member Function Documentation

#### 6.3.3.1 DoStep()

```
override void Simulation.ConstantStepSimulation.DoStep ( )  [protected], [virtual]
```

Implements Simulation.Simulation.

Definition at line 25 of file StepSimulation.cs.

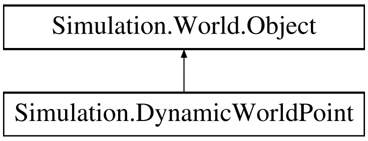
The documentation for this class was generated from the following file:

- StepSimulation.cs

## 6.4 Simulation.DynamicWorldPoint Class Reference

A point in a world that can move. That means this object is also defined by Velocity, current Acceleration and its Mass

Inheritance diagram for Simulation.DynamicWorldPoint:



### Public Member Functions

- DynamicWorldPoint (World world, Vector2 location, float mass)

  *Creates an instance of DynamicWorldPoint and binds it with a specific World*
- DynamicWorldPoint (World world, Vector2 location, float mass, Vector2 velocity)

  *Creates an instance of DynamicWorldPoint with a specififc starting velocity and binds it with a specific World*
- void StartApplyingForce (Vector2 force)

  *Starts applying a force to the point*
- void StopApplyingForce ()

  *Stops all forces, that are currently being applied on the point*
- override World.Object CloneToWorld (World world)

  *Clones this world object to a different instance of World and binds it to that world*

### Protected Member Functions

- override void Step (object sender, StepEventArgs e)

  *A method which is called when a step is being taken*

## Properties

- Vector2 Acceleration `[get]`

  *A vector defining current acceleration and direction, at which the point is accelerating*
- Vector2 Velocity `[getset]`

  *A vector defining the points speed and direction in which it is travelling*
- float Mass `[get]`

  *A mass of the point*
- float Speed `[get]`

  *A speed at which the point is travelling*
- double Direction `[get]`

  *A direction at wihch the point is travelling*

## 6.4.1 Detailed Description

A point in a world that can move. That means this object is also defined by Velocity, current Acceleration and its Mass

Definition at line 9 of file DynamicWorldObject.cs.

## 6.4.2 Constructor & Destructor Documentation

### 6.4.2.1 DynamicWorldPoint() `[1/2]`

```
Simulation.DynamicWorldPoint.DynamicWorldPoint (
            World world,
            Vector2 location,
            float mass )
```

Creates an instance of DynamicWorldPoint and binds it with a specific World

**Parameters**

| | |
|---|---|
| *world* | An instance of World in which the point exists and is bound to |
| *location* | The location in relation to the origin in the World that the point is bound to |
| *mass* | The mass of the point |

Definition at line 17 of file DynamicWorldObject.cs.

### 6.4.2.2 DynamicWorldPoint() `[2/2]`

```
Simulation.DynamicWorldPoint.DynamicWorldPoint (
            World world,
            Vector2 location,
            float mass,
            Vector2 velocity )
```

Creates an instance of DynamicWorldPoint with a specififc starting velocity and binds it with a specific World

**Parameters**

| | |
|---|---|
| *world* | An instance of World in which the point exists and is bound to |
| *location* | The location in relation to the origin in the World that the point is bound to |
| *mass* | The mass of the point |
| *velocity* | A vector defining the points speed and direction in which it is travelling |

Definition at line 28 of file DynamicWorldObject.cs.

## 6.4.3 Member Function Documentation

### 6.4.3.1 CloneToWorld()

```
override World.Object Simulation.DynamicWorldPoint.CloneToWorld (
            World world )  [virtual]
```

Clones this world object to a different instance of World and binds it to that world

**Parameters**

| | |
|---|---|
| *world* | An instance of World the object is being cloned to |

**Returns**

A cloned instance of World.Object

Implements Simulation.World.Object.

Definition at line 73 of file DynamicWorldObject.cs.

### 6.4.3.2 StartApplyingForce()

```
void Simulation.DynamicWorldPoint.StartApplyingForce (
            Vector2 force )
```

Starts applying a force to the point

**Parameters**

| | |
|---|---|
| *force* | A vector defining a force and a direction in which it is applied |

Definition at line 60 of file DynamicWorldObject.cs.

### 6.4.3.3 Step()

```
override void Simulation.DynamicWorldPoint.Step (
             object sender,
             StepEventArgs e )  [protected], [virtual]
```

A method which is called when a step is being taken

**Parameters**

| | |
|---|---|
| *sender* | Object that invoked the step |
| *e* | Instance of StepEventArgs describing the step |

Implements Simulation.World.Object.

Definition at line 80 of file DynamicWorldObject.cs.

### 6.4.3.4 StopApplyingForce()

```
void Simulation.DynamicWorldPoint.StopApplyingForce ( )
```

Stops all forces, that are currently being applied on the point

Definition at line 68 of file DynamicWorldObject.cs.

## 6.4.4 Property Documentation

### 6.4.4.1 Acceleration

```
Vector2 Simulation.DynamicWorldPoint.Acceleration  [get]
```

A vector defining current acceleration and direction, at which the point is accelerating

Definition at line 37 of file DynamicWorldObject.cs.

#### 6.4.4.2 Direction

```
double Simulation.DynamicWorldPoint.Direction  [get]
```

A direction at wihch the point is travelling

Definition at line 54 of file DynamicWorldObject.cs.

#### 6.4.4.3 Mass

```
float Simulation.DynamicWorldPoint.Mass  [get]
```

A mass of the point

Definition at line 45 of file DynamicWorldObject.cs.

#### 6.4.4.4 Speed

```
float Simulation.DynamicWorldPoint.Speed  [get]
```

A speed at which the point is travelling

Definition at line 50 of file DynamicWorldObject.cs.

#### 6.4.4.5 Velocity

```
Vector2 Simulation.DynamicWorldPoint.Velocity  [get], [set]
```

A vector defining the points speed and direction in which it is travelling

Definition at line 41 of file DynamicWorldObject.cs.

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

- DynamicWorldObject.cs

## 6.5 Simulation.Encounter Struct Reference

Struct that is used to describe encounter of two objects

## Properties

- static Encounter NULL `[get]`

  *Instance of Encounter used to describe no encounter*
- float Distance `[getset]`

  *Distance between the two objects during the encounter*
- TimeSpan Timestamp `[getset]`

  *Timestamp at which the encounter occured*
- World WorldSnapshot `[getset]`

  *A snapshot of the word at the time the encounter happened*

## 6.5.1 Detailed Description

Struct that is used to describe encounter of two objects

Definition at line 8 of file Encounter.cs.

## 6.5.2 Property Documentation

### 6.5.2.1 Distance

```
float Simulation.Encounter.Distance  [get], [set]
```

Distance between the two objects during the encounter

Definition at line 23 of file Encounter.cs.

### 6.5.2.2 NULL

```
Encounter Simulation.Encounter.NULL  [static], [get]
```

Instance of Encounter used to describe no encounter

Definition at line 13 of file Encounter.cs.

### 6.5.2.3 Timestamp

```
TimeSpan Simulation.Encounter.Timestamp  [get], [set]
```

Timestamp at which the encounter occured

Definition at line 27 of file Encounter.cs.

#### 6.5.2.4 WorldSnapshot

```
World Simulation.Encounter.WorldSnapshot  [get], [set]
```

A snapshot of the word at the time the encounter happened

Definition at line 31 of file Encounter.cs.

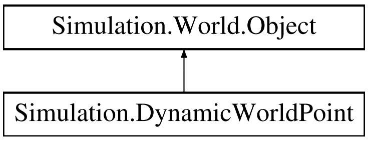
The documentation for this struct was generated from the following file:

- Encounter.cs

## 6.6 Simulation.World.Object Class Reference

Abstract class used to define a world object

Inheritance diagram for Simulation.World.Object:



### Public Member Functions

- Object (World world, Vector2 location)

  *Creates instance of a world object*
- abstract Object CloneToWorld (World world)

  *Clones this world object to a different instance of World and binds it to that world*

### Protected Member Functions

- abstract void Step (object sender, StepEventArgs e)

  *A method which is called when a step is being taken*

### Properties

- Vector2 Location `[getset]`

  *The location in relation to the origin*
- Guid ID `[get]`

  *A unique ID od the world object*

### 6.6.1 Detailed Description

Abstract class used to define a world object

Definition at line 66 of file World.cs.

## 6.6.2 Constructor & Destructor Documentation

### 6.6.2.1 Object()

```
Simulation.World.Object.Object (
            World world,
            Vector2 location )
```

Creates instance of a world object

**Parameters**

| | |
|---|---|
| *world* | An instance of World in which the point exists and is bound to |
| *location* | The location in relation to the origin |

Definition at line 73 of file World.cs.

## 6.6.3 Member Function Documentation

### 6.6.3.1 CloneToWorld()

```
abstract Object Simulation.World.Object.CloneToWorld (
            World world )  [pure virtual]
```

Clones this world object to a different instance of World and binds it to that world

**Parameters**

| | |
|---|---|
| *world* | An instance of World the object is being cloned to |

**Returns**

A cloned instance of World.Object

Implemented in Simulation.DynamicWorldPoint.

### 6.6.3.2 Step()

```
abstract void Simulation.World.Object.Step (
            object sender,
            StepEventArgs e )  [protected], [pure virtual]
```

A method which is called when a step is being taken

**Parameters**

| | |
|---|---|
| *sender* | Object that invoked the step |
| *e* | Instance of StepEventArgs describing the step |

Implemented in Simulation.DynamicWorldPoint.

## 6.6.4 Property Documentation

### 6.6.4.1 ID

```
Guid Simulation.World.Object.ID  [get]
```

A unique ID od the world object

Definition at line 91 of file World.cs.

### 6.6.4.2 Location

```
Vector2 Simulation.World.Object.Location  [get], [set]
```

The location in relation to the origin

Definition at line 84 of file World.cs.

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

- World.cs

## 6.7 Simulation.Program Class Reference

### Static Public Member Functions

- static Encounter FindClosestEncounter (Simulation simulation, DynamicWorldPoint obj1, DynamicWorldPoint obj2, bool reportTime=false)

  *Runs the simulation and records closest encounter of two objects*

## Static Public Attributes

- const float s1 = 15

  *Distance of Point1 from the origin of the right angle*
- const float s2 = 10

  *Distance of Point2 from the origin of the right angle*
- const float v1 = 15

  *The size of velocity vector used by Point1 that is heading to the origin of the right angle*
- const float v2 = 20

  *The size of velocity vector used by Point2 that is heading to the origin of the right angle*
- const float m1 = 10

  *The mass of Point1*
- const float m2 = 20

  *The mass of Point2*
- const float a1 = 2

  *The acceleration used by Point1 (used only by the acceleration simulation)*
- const float a2 = 1

  *The acceleration used by Point2 (used only by the acceleration simulation)*

## 6.7.1 Detailed Description

Definition at line 9 of file Program.cs.

## 6.7.2 Member Function Documentation

### 6.7.2.1 FindClosestEncounter()

```
static Encounter Simulation.Program.FindClosestEncounter (
            Simulation simulation,
            DynamicWorldPoint obj1,
            DynamicWorldPoint obj2,
            bool reportTime = false )  [static]
```

Runs the simulation and records closest encounter of two objects

**Parameters**

| | |
|---|---|
| *simulation* | The simulation in which the objects are simulated |
| *obj1* | First object that should be watched |
| *obj2* | Second object that should be watched |

**Returns**

Instance of Encounter describing the closest encounter

Definition at line 95 of file Program.cs.

## 6.7.3 Member Data Documentation

### 6.7.3.1 a1

```
const float Simulation.Program.a1 = 2  [static]
```

The acceleration used by Point1 (used only by the acceleration simulation)

Definition at line 52 of file Program.cs.

### 6.7.3.2 a2

```
const float Simulation.Program.a2 = 1  [static]
```

The acceleration used by Point2 (used only by the acceleration simulation)

Definition at line 56 of file Program.cs.

### 6.7.3.3 m1

```
const float Simulation.Program.m1 = 10  [static]
```

The mass of Point1

Definition at line 43 of file Program.cs.

### 6.7.3.4 m2

```
const float Simulation.Program.m2 = 20  [static]
```

The mass of Point2

Definition at line 47 of file Program.cs.

### 6.7.3.5 s1

```
const float Simulation.Program.s1 = 15  [static]
```

Distance of Point1 from the origin of the right angle

Definition at line 25 of file Program.cs.

#### 6.7.3.6 s2

```
const float Simulation.Program.s2 = 10  [static]
```

Distance of Point2 from the origin of the right angle

Definition at line 29 of file Program.cs.

#### 6.7.3.7 v1

```
const float Simulation.Program.v1 = 15  [static]
```

The size of velocity vector used by Point1 that is heading to the origin of the right angle

Definition at line 34 of file Program.cs.

#### 6.7.3.8 v2

```
const float Simulation.Program.v2 = 20  [static]
```

The size of velocity vector used by Point2 that is heading to the origin of the right angle

Definition at line 38 of file Program.cs.

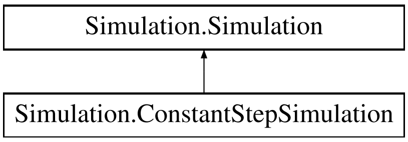
The documentation for this class was generated from the following file:

- Program.cs

## 6.8 Simulation.Simulation Class Reference

A base class used to define how world is simulated

Inheritance diagram for Simulation.Simulation:



### Public Member Functions

- delegate void SteppedEventHandler (object sender, SteppedEventArgs e)

  *Handler used to handle a stepped event*
- Simulation (World world)
- void StopSimulation ()
- void Simulate ()

## Protected Member Functions

- abstract void DoStep ()

## Protected Attributes

- World world

## Properties

- TimeSpan TimeSinceEpoch `[getprotected set]`

## Events

- SteppedEventHandler Stepped

## 6.8.1 Detailed Description

A base class used to define how world is simulated

Definition at line 8 of file Simulation.cs.

## 6.8.2 Constructor & Destructor Documentation

### 6.8.2.1 Simulation()

```
Simulation.Simulation.Simulation (
            World world )
```

Definition at line 17 of file Simulation.cs.

## 6.8.3 Member Function Documentation

### 6.8.3.1 DoStep()

```
abstract void Simulation.Simulation.DoStep ( )  [protected], [pure virtual]
```

Implemented in Simulation.ConstantStepSimulation.

#### 6.8.3.2 Simulate()

```
void Simulation.Simulation.Simulate ( )
```

Definition at line 32 of file Simulation.cs.

#### 6.8.3.3 SteppedEventHandler()

```
delegate void Simulation.Simulation.SteppedEventHandler (
             object sender,
             SteppedEventArgs e )
```

Handler used to handle a stepped event

**Parameters**

| | |
|---|---|
| *sender* | Object that invoked the step |
| *e* | Instance of SteppedEventArgs describing the state of the simulated world after a step |

#### 6.8.3.4 StopSimulation()

```
void Simulation.Simulation.StopSimulation ( )
```

### 6.8.4 Member Data Documentation

#### 6.8.4.1 world

```
World Simulation.Simulation.world  [protected]
```

Definition at line 24 of file Simulation.cs.

### 6.8.5 Property Documentation

#### 6.8.5.1 TimeSinceEpoch

```
TimeSpan Simulation.Simulation.TimeSinceEpoch  [get], [protected set]
```

Definition at line 30 of file Simulation.cs.

### 6.8.6 Event Documentation

#### 6.8.6.1 Stepped

`SteppedEventHandler Simulation.Simulation.Stepped`

Definition at line 22 of file Simulation.cs.

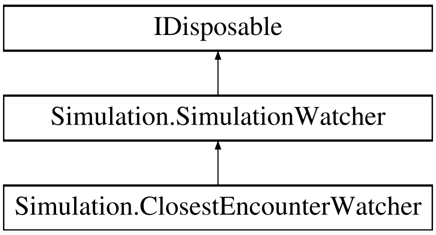
The documentation for this class was generated from the following file:

- Simulation.cs

## 6.9 Simulation.SimulationWatcher Class Reference

Abstract class used to define so called watchers. Watchers are objects that can analyze a world every simulation step. For example a watcher can look for a closest encounter of two objects.

Inheritance diagram for Simulation.SimulationWatcher:


IDisposable
Simulation.SimulationWatcher
Simulation.ClosestEncounterWatcher

### Public Member Functions

- SimulationWatcher (Simulation simulation)

    *Creates an instance of watcher, that analyzes a specific simulation*
- void Dispose ()

    *Releases the watcher from the simulation*

### Protected Member Functions

- abstract void Simulation_Stepped (object sender, SteppedEventArgs e)

    *Gets invoked after every step in the watched simulation*

### 6.9.1 Detailed Description

Abstract class used to define so called watchers. Watchers are objects that can analyze a world every simulation step. For example a watcher can look for a closest encounter of two objects.

Definition at line 9 of file SimulationWatcher.cs.

## 6.9.2 Constructor & Destructor Documentation

### 6.9.2.1 SimulationWatcher()

```
Simulation.SimulationWatcher.SimulationWatcher (
             Simulation simulation )
```

Creates an instance of watcher, that analyzes a specific simulation

**Parameters**

| | |
|---|---|
| *simulation* | Instance of Simulation, that the watcher watches |

Definition at line 15 of file SimulationWatcher.cs.

## 6.9.3 Member Function Documentation

### 6.9.3.1 Dispose()

```
void Simulation.SimulationWatcher.Dispose ( )
```

Releases the watcher from the simulation

### 6.9.3.2 Simulation_Stepped()

```
abstract void Simulation.SimulationWatcher.Simulation_Stepped (
             object sender,
             SteppedEventArgs e )  [protected], [pure virtual]
```

Gets invoked after every step in the watched simulation

**Parameters**

| | |
|---|---|
| *sender* | Instance of the object that invoked the step |
| *e* | Instance of SteppedEventArgs describing the state of the simulated world after the step |

Implemented in Simulation.ClosestEncounterWatcher.

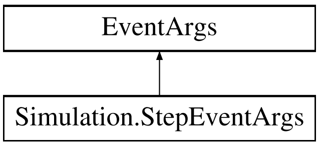
The documentation for this class was generated from the following file:

- SimulationWatcher.cs

## 6.10 Simulation.StepEventArgs Class Reference

Class that holds data about the current step

Inheritance diagram for Simulation.StepEventArgs:



### Properties

- TimeSpan DeltaTime `[getset]`

  *Time, between the current step and the previous step*

### 6.10.1 Detailed Description

Class that holds data about the current step

Definition at line 8 of file StepEventArgs.cs.

### 6.10.2 Property Documentation

#### 6.10.2.1 DeltaTime

```
TimeSpan Simulation.StepEventArgs.DeltaTime  [get], [set]
```

Time, between the current step and the previous step

Definition at line 13 of file StepEventArgs.cs.

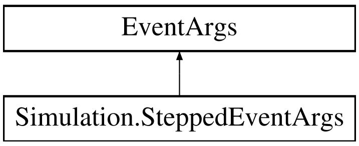
The documentation for this class was generated from the following file:

- StepEventArgs.cs

## 6.11 Simulation.SteppedEventArgs Class Reference

Class that holds data about a step that has been taken

Inheritance diagram for Simulation.SteppedEventArgs:



## Properties

- World WorldSnapshot `[getset]`

  *A clone of the world after the step was taken*
- TimeSpan TimeSinceEpoch `[getset]`

  *A since the world has been created in the world time*

### 6.11.1 Detailed Description

Class that holds data about a step that has been taken

Definition at line 8 of file SteppedEventArgs.cs.

### 6.11.2 Property Documentation

#### 6.11.2.1 TimeSinceEpoch

```
TimeSpan Simulation.SteppedEventArgs.TimeSinceEpoch  [get], [set]
```

A since the world has been created in the world time

Definition at line 17 of file SteppedEventArgs.cs.

#### 6.11.2.2 WorldSnapshot

```
World Simulation.SteppedEventArgs.WorldSnapshot  [get], [set]
```

A clone of the world after the step was taken

Definition at line 13 of file SteppedEventArgs.cs.

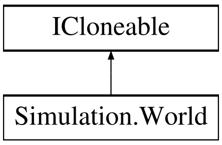
The documentation for this class was generated from the following file:

- SteppedEventArgs.cs

## 6.12 Simulation.World Class Reference

An object used to hold world objects(see World.Object)

Inheritance diagram for Simulation.World:



## Classes

- class Object

  *Abstract class used to define a world object*

## Public Member Functions

- delegate void StepEventHandler (object sender, StepEventArgs e)

  *Handler used to handle a step event*
- World ()

  *Creates a new world*
- void DoStep (TimeSpan deltaTime)

  *Takes a simulation step*
- object Clone ()
- Object FindObject (Guid id)

  *Find a objects by its ID*
- Object GetObjectByIndex (int index)

  *Gets object by its index. Indexes are being asigned by order the objects were binded to the world.*

## Events

- StepEventHandler Step

  *Event, which gets invoked when a new step is being taken*

## 6.12.1 Detailed Description

An object used to hold world objects(see World.Object)

Definition at line 10 of file World.cs.

## 6.12.2 Constructor & Destructor Documentation

### 6.12.2.1 World()

```
Simulation.World.World ( )
```

Creates a new world

Definition at line 22 of file World.cs.

## 6.12.3 Member Function Documentation

#### 6.12.3.1 Clone()

```
object Simulation.World.Clone ( )
```

Definition at line 43 of file World.cs.

#### 6.12.3.2 DoStep()

```
void Simulation.World.DoStep (
             TimeSpan deltaTime )
```

Takes a simulation step

**Parameters**

| | |
|---|---|
| *deltaTime* | A time between this step and a previous step |

Definition at line 38 of file World.cs.

#### 6.12.3.3 FindObject()

```
Object Simulation.World.FindObject (
             Guid id )
```

Find a objects by its ID

**Parameters**

| | |
|---|---|
| *id* | ID of the object |

**Returns**

An instance of the object from this world

#### 6.12.3.4 GetObjectByIndex()

```
Object Simulation.World.GetObjectByIndex (
             int index )
```

Gets object by its index. Indexes are being asigned by order the objects were binded to the world.

**Parameters**

| | |
|---|---|
| *index* | |

**Returns**

#### 6.12.3.5 StepEventHandler()

```
delegate void Simulation.World.StepEventHandler (
            object sender,
            StepEventArgs e )
```

Handler used to handle a step event

**Parameters**

| | |
|---|---|
| *sender* | Object that invoked the step |
| *e* | Instance of StepEventArgs describing the step |

### 6.12.4 Event Documentation

#### 6.12.4.1 Step

```
StepEventHandler Simulation.World.Step
```

Event, which gets invoked when a new step is being taken

Definition at line 30 of file World.cs.

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

- World.cs

# Chapter 7

# File Documentation

## 7.1 ClosestEncounterWatcher.cs File Reference

### Classes

- class Simulation.ClosestEncounterWatcher

  *A class that looks for a closest encounter of two objects in a simulation*

### Namespaces

- namespace Simulation

## 7.2 ClosestEncounterWatcher.cs

Go to the documentation of this file.

```
using System;

namespace Simulation
{
    public class ClosestEncounterWatcher : SimulationWatcher
    {
        public static readonly ShortestDistanceFinderSettings DEFAULT_SETTINGS = new ShortestDistanceFinderSettings()
        {
            AutoStopSimulation = false
        };

        public ClosestEncounterWatcher(Simulation simulation, World.Object obj1, World.Object obj2) : this(simulation, obj1, obj2, DEFAULT_SETTINGS)
        {
        }

        public ClosestEncounterWatcher(Simulation simulation, World.Object obj1, World.Object obj2, ShortestDistanceFinderSettings settings) : base(simulation)
        {
            this.settings = settings;
            this.obj1 = obj1;
            this.obj2 = obj2;
        }

        public Encounter CurrentClosestEncounter => closestEncounter;
        private Encounter closestEncounter = Encounter.NULL;

        private ShortestDistanceFinderSettings settings;
        private float lastDistance = float.PositiveInfinity;
        private readonly World.Object obj1;
        private readonly World.Object obj2;

        public void ResetClosestEncounter() => closestEncounter = Encounter.NULL;

        protected override void Simulation_Stepped(object sender, SteppedEventArgs e)
        {
            float distance = (obj1.Location - obj2.Location).Length();

            // If the distance between objects is closer than anytime before -> record the encounter
            if (distance < closestEncounter.Distance)
                closestEncounter = new Encounter()
                {
                    Distance = distance,
                    Timestamp = e.TimeSinceEpoch,
                    WorldSnapshot = e.WorldSnapshot
                };

            // When objects are starting to move away and settings allow it -> stop the simulation
            if (settings.AutoStopSimulation && distance > lastDistance)
                simulation.StopSimulation();

            lastDistance = distance;
        }
    }
}
```

## 7.3 ClosestEncounterWatcherSettings.cs File Reference

### Classes

- struct Simulation.ClosestEncounterWatcherSettings

  *Settings that define how a ClosestEncounterWatcher acts*

### Namespaces

- namespace Simulation

## 7.4 ClosestEncounterWatcherSettings.cs

Go to the documentation of this file.

```
namespace Simulation
{
    public struct ClosestEncounterWatcherSettings
    {
        public bool AutoStopSimulation { get; set; }
    }
}
```

## 7.5 DynamicWorldObject.cs File Reference

### Classes

- class Simulation.DynamicWorldPoint

  *A point in a world that can move. That means this object is also defined by Velocity, current Acceleration and its Mass*

### Namespaces

- namespace Simulation

## 7.6 DynamicWorldObject.cs

Go to the documentation of this file.

```
using System;
using System.Numerics;

namespace Simulation
{
    public class DynamicWorldPoint : World.Object
    {
        public DynamicWorldPoint(World world, Vector2 location, float mass) : this(world, location, mass, new Vector2(0))
        {
        }

        public DynamicWorldPoint(World world, Vector2 location, float mass, Vector2 velocity) : base(world, location)
        {
            Velocity = velocity;
            Mass = mass;
        }

        public Vector2 Acceleration { get; private set; }
        public Vector2 Velocity { get; set; }
        public float Mass { get; }

        public float Speed => Velocity.Length();
        public double Direction => Math.Atan2(Velocity.Y, Velocity.X);

        public void StartApplyingForce(Vector2 force)
        {
            Acceleration += force / Mass;
        }

        public void StopApplyingForce()
        {
            Acceleration = new Vector2(0);
        }

        public override World.Object CloneToWorld(World world)
        {
            DynamicWorldPoint clone = new DynamicWorldPoint(world, Location, Mass, Velocity);
            clone.Acceleration = Acceleration;
            return clone;
        }

        protected override void Step(object sender, StepEventArgs e)
        {
            Location += Velocity * (float)e.DeltaTime.TotalSeconds;
            Velocity += Acceleration * (float)e.DeltaTime.TotalSeconds;
        }
    }
}
```

## 7.7 Encounter.cs File Reference

### Classes

- struct Simulation.Encounter

  *Struct that is used to describe encounter of two objects*

### Namespaces

- namespace Simulation

## 7.8 Encounter.cs

Go to the documentation of this file.

```
using System;

namespace Simulation
{
    public struct Encounter
    {
        public static Encounter NULL => new Encounter()
        {
            Distance = float.PositiveInfinity,
            Timestamp = TimeSpan.FromSeconds(-1),
            WorldSnapshot = null
        };

        public float Distance { get; set; }
        public TimeSpan Timestamp { get; set; }
        public World WorldSnapshot { get; set; }
    }
}
```

## 7.9 obj/Debug/net5.0/.NETCoreApp,Version=v5.0.AssemblyAttributes.cs File Reference

## 7.10 .NETCoreApp,Version=v5.0.AssemblyAttributes.cs

Go to the documentation of this file.

```
// <autogenerated />
using System;
using System.Reflection;
[assembly: global::System.Runtime.Versioning.TargetFrameworkAttribute(".NETCoreApp,Version=v5.0",
 FrameworkDisplayName = "")]
```

## 7.11 obj/Debug/net5.0/Simulation.AssemblyInfo.cs File Reference

## 7.12 Simulation.AssemblyInfo.cs

Go to the documentation of this file.

```
//------------------------------------------------------------------------------
// <auto-generated>
//     This code was generated by a tool.
//     Runtime Version:4.0.30319.42000
//
//     Changes to this file may cause incorrect behavior and will be lost if
//     the code is regenerated.
// </auto-generated>
//------------------------------------------------------------------------------

using System;
using System.Reflection;

[assembly: System.Reflection.AssemblyCompanyAttribute("Simulation")]
[assembly: System.Reflection.AssemblyConfigurationAttribute("Debug")]
[assembly: System.Reflection.AssemblyFileVersionAttribute("1.0.0.0")]
[assembly: System.Reflection.AssemblyInformationalVersionAttribute("1.0.0")]
[assembly: System.Reflection.AssemblyProductAttribute("Simulation")]
[assembly: System.Reflection.AssemblyTitleAttribute("Simulation")]
[assembly: System.Reflection.AssemblyVersionAttribute("1.0.0.0")]

// Generated by the MSBuild WriteCodeFragment class.

```

# 7.13 Program.cs File Reference

## Classes

- class Simulation.Program

## Namespaces

- namespace Simulation

# 7.14 Program.cs

Go to the documentation of this file.

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Numerics;
using System.Threading.Tasks;

namespace Simulation
{
    class Program
    {
        /*
         * POINT1
         *   v
         *   |
         *   |
         *   |
         *   |
         *   |
         *   \----------- < POINT2
         */

        public const float s1 = 15;
        public const float s2 = 10;

        public const float v1 = 15;
        public const float v2 = 20;

        public const float m1 = 10;
        public const float m2 = 20;

        public const float a1 = 2;
        public const float a2 = 1;

        static void Main(string[] args)
        {
            Console.WriteLine("Creating worlds...");
            // Create seperate worlds for each simulation
            World constVelWorld = new World();
            var constPoint1 = new DynamicWorldPoint(constVelWorld, new Vector2(0, s1), m1, new Vector2(0, -v1));
            var constPoint2 = new DynamicWorldPoint(constVelWorld, new Vector2(s2, 0), m2, new Vector2(-v2, 0));

            World accelVelWorld = new World();
            var accelPoint1 = new DynamicWorldPoint(accelVelWorld, new Vector2(0, s1), m1, new Vector2(0, -v1));
            accelPoint1.StartApplyingForce(new Vector2(0, -(a1 * m1)));
            var accelPoint2 = new DynamicWorldPoint(accelVelWorld, new Vector2(s2, 0), m2, new Vector2(-v2, 0));
            accelPoint2.StartApplyingForce(new Vector2(-(a2 * m2), 0));

            Console.WriteLine("Creating simulations...");
            Simulation constSimulation = new StepSimulation(constVelWorld, TimeSpan.FromMilliseconds(1), TimeSpan.FromSeconds(10));
            Simulation accelSimulation = new StepSimulation(accelVelWorld, TimeSpan.FromMilliseconds(1), TimeSpan.FromSeconds(10));

            // Run the simulations
            Console.WriteLine("Constant speed simulation:");
            var closestConstEncounter = FindClosestEncounter(constSimulation, constPoint1, constPoint2, true);
            Console.WriteLine("Constant speed simulation:");
            var closestAccelEncounter = FindClosestEncounter(accelSimulation, accelPoint1, accelPoint2, true);

            // Print the results
            Console.WriteLine($"Closest encounter with constnant speed happend { closestConstEncounter.Timestamp.TotalSeconds.ToString("0.000") } seconds after the epoch. Points were { closestConstEncounter.Distance }m apart from eachother.");
            Console.WriteLine($"Closest encounter when accelerating happend { closestAccelEncounter.Timestamp.TotalSeconds.ToString("0.000") } seconds after the epoch. Points were { closestAccelEncounter.Distance }m apart from eachother.");
            Console.ReadKey(true);
        }

        public static Encounter FindClosestEncounter(Simulation simulation, DynamicWorldPoint obj1, DynamicWorldPoint obj2, bool reportTime = false)
        {
            Encounter closest;
            using (var watcher = new ClosestEncounterWatcher(simulation, obj1, obj2))
            {
                Console.WriteLine("Simulating...");
                Console.Write("Seconds Processed: ");
                if(reportTime)
                    simulation.Stepped += Simulation_ReportTime;

                simulation.Simulate();
                Console.WriteLine();
                closest = watcher.CurrentClosestEncounter;
            }

            if (reportTime)
                simulation.Stepped -= Simulation_ReportTime;

            return closest;

            static void Simulation_ReportTime(object sender, SteppedEventArgs e)
            {
                int x = Console.CursorLeft, y = Console.CursorTop;
                Console.Write(e.TimeSinceEpoch.TotalSeconds.ToString("0.000"));
                Console.SetCursorPosition(x, y);
            }
        }
    }
}
```

## 7.15 Simulation.cs File Reference

### Classes

- class Simulation.Simulation

  *A base class used to define how world is simulated*

### Namespaces

- namespace Simulation

## 7.16 Simulation.cs

Go to the documentation of this file.

```
using System;

namespace Simulation
{
    public abstract class Simulation
    {
        public delegate void SteppedEventHandler(object sender, SteppedEventArgs e);

        public Simulation(World world)
        {
            this.world = world;
        }

        public event SteppedEventHandler Stepped;

        protected World world;

        private bool cancellationRequested = false;

        public void StopSimulation() => cancellationRequested = true;

        public TimeSpan TimeSinceEpoch { get; protected set; }

        public void Simulate()
        {
            // First step (before updating any world objects)
            HandleStepped();

            while (!cancellationRequested)
            {
                DoStep();
                HandleStepped();
            }
        }

        private void HandleStepped()
        {
            Stepped?.Invoke(this, new SteppedEventArgs()
            {
                WorldSnapshot = world.Clone() as World,
                TimeSinceEpoch = TimeSinceEpoch
            });
        }

        protected abstract void DoStep();
    }
}
```

# 7.17 SimulationWatcher.cs File Reference

## Classes

- class Simulation.SimulationWatcher

  *Abstract class used to define so called watchers. Watchers are objects that can analyze a world every simulation step. For example a watcher can look for a closest encounter of two objects.*

## Namespaces

- namespace Simulation

# 7.18 SimulationWatcher.cs

Go to the documentation of this file.

```
using System;
namespace Simulation
{
    public abstract class SimulationWatcher : IDisposable
    {
        public SimulationWatcher(Simulation simulation)
        {
            simulation.Stepped += Simulation_Stepped;
            this.simulation = simulation;
        }

        private readonly Simulation simulation;

        protected abstract void Simulation_Stepped(object sender, SteppedEventArgs e);

        public void Dispose() => simulation.Stepped -= Simulation_Stepped;
    }
}
```

## 7.19 StepEventArgs.cs File Reference

### Classes

- class Simulation.StepEventArgs

  *Class that holds data about the current step*

### Namespaces

- namespace Simulation

## 7.20 StepEventArgs.cs

Go to the documentation of this file.

```
using System;

namespace Simulation
{
    public class StepEventArgs : EventArgs
    {
        public TimeSpan DeltaTime { get; set; }
    }
}
```

## 7.21 SteppedEventArgs.cs File Reference

### Classes

- class Simulation.SteppedEventArgs

  *Class that holds data about a step that has been taken*

### Namespaces

- namespace Simulation

## 7.22 SteppedEventArgs.cs

Go to the documentation of this file.

```
using System;

namespace Simulation
{
    public class SteppedEventArgs : EventArgs
    {
        public World WorldSnapshot { get; set; }
        public TimeSpan TimeSinceEpoch { get; set; }
    }
}
```

## 7.23 StepSimulation.cs File Reference

### Classes

- class Simulation.ConstantStepSimulation

  *A simulation, that has a constant delay between each step.*

### Namespaces

- namespace Simulation

## 7.24 StepSimulation.cs

Go to the documentation of this file.

```
using System;

namespace Simulation
{
    public class ConstantStepSimulation : Simulation
    {
        public ConstantStepSimulation(World world, TimeSpan stepLength, TimeSpan simulationLength) : base(world)
        {
            this.stepLength = stepLength;
            this.simulationLength = simulationLength;
        }

        private readonly TimeSpan stepLength;
        private readonly TimeSpan simulationLength;

        protected override void DoStep()
        {
            world.DoStep(stepLength);
            TimeSinceEpoch += stepLength;

            if (TimeSinceEpoch > simulationLength)
                StopSimulation();
        }
    }
}
```

## 7.25 World.cs File Reference

### Classes

- class Simulation.World

  *An object used to hold world objects(see World.Object)*

- class Simulation.World.Object

  *Abstract class used to define a world object*

### Namespaces

- namespace Simulation

## 7.26 World.cs

Go to the documentation of this file.

```
using System;
using System.Collections.Generic;
using System.Numerics;

namespace Simulation
{
    public class World : ICloneable
    {
        public delegate void StepEventHandler(object sender, StepEventArgs e);

        public World()
        {
            WorldObjects = new List<Object>();
        }

        public event StepEventHandler Step;

        private readonly List<Object> WorldObjects;

        public void DoStep(TimeSpan deltaTime)
        {
            Step?.Invoke(this, new StepEventArgs() { DeltaTime = deltaTime });
        }

        public object Clone()
        {
            World clone = new World();
            WorldObjects.ForEach(obj => obj.CloneToWorld(clone));
            return clone;
        }

        public Object FindObject(Guid id) => WorldObjects.Find(x => x.ID.Equals(id));
        public Object GetObjectByIndex(int index) => WorldObjects[index];

        public abstract class Object
        {
            public Object(World world, Vector2 location)
            {
                world.Step += Step;
                world.WorldObjects.Add(this);
                Location = location;
                id = Guid.NewGuid();
            }

            public Vector2 Location { get; set; }

            private Guid id;

            public Guid ID => id;

            protected abstract void Step(object sender, StepEventArgs e);

            public abstract Object CloneToWorld(World world);
        }
    }
}
```