

Document of ClassLib OscilloscopeKernel

Index

- [Foreword](#)
- [OscilloscopeKernel](#)
 - [MultiThreadOscilloscope](#)
 - [Wave](#)
 - [Drawing](#)
 - [Tools](#)
 - [Producer](#)

Foreword

- if the method or attribute of a certain class that behave the same as the super-class or behave just as the implemented interface requires, it will not be listed again in the document of this certain class.
- `private` attribute, field, or method will not be listed. `protected` attribute and method will be special marked at the class's attribute-list or method-list. So, the attributes and methods that are listed without special mark are all `public`.
- `protected` and `public` has no difference when it comes to the constructor of a abstract class, so `protected` will not be special marked on this occasion.
- the time unit is defined with [Waves.UNIT_NUMBER_PRO_SECOND](#). the defaute time unit is μs but most of System functions use *ms* as the time unit, be careful!.

OscilloscopeKernel

1 | namespace OscilloscopeKernel

type	name	description
abstract class	SingleThreadOscilloscope	an abstract class that describe an oscilloscope that cannot start a new draw-task while the old one has not finish
class	SimpleOscilloscope	a SingleThreadOscilloscope with public Draw() .
class	TimeCountedOscilloscope	a SingleThreadOscilloscope with public Draw() and a built-in watch, which means it doesn't need to delta_time as input.
abstract class	MultiThreadOscilloscope	an abstract class that describe an oscilloscope that can start a new draw-task while the old one has not finish
class	UndrivedOscilloscope	a MultiThreadOscilloscope with public Draw() .
class	DrivedOscilloscope	a MultiThreadOscilloscope that can produce graphs periodically.
namespace	Wave	

SingleThreadOscilloscope

```
1 public abstract class SingleThreadOscilloscope<T>;
```

- namespace: [OscilloscopeKernel](#)
- supers: none
- interfaces: none
- summary:
 - an oscilloscope that cannot start a new draw-task while the old one has not finished.
 - T is the output type of this oscilloscope.
- remarks
 - this is a abstract class, if you want to use it, please try [SimpleOscilloscope](#) or [TimeCountedOscilloscope](#).
 - calling [Draw\(\)](#) to produce and get a new graph.
 - no attribute or method will be provided to get the panel that this oscilloscope is using, so you need to handle the reference of it by yourself.
- constructors:

name	description
SingleThreadOscilloscope (ICanvas canvas, IPointDrawer point_drawer, IGraphProducer graph_producer, IControlPanel control_panel)	

- methods:

name	description
protected T Draw (double delta_time)	produce and get a new graph.

constructors:

```
1 protected SingleThreadOscilloscope(  
2     ICanvas<T> canvas,  
3     IPointDrawer point_drawer,  
4     IGraphProducer graph_producer,  
5     IControlPanel control_panel)
```

- Summary:
 - create a new oscilloscope.
- Remarks:
 - every input objects should not be used by other oscilloscope at the same time.
 - no attribute or method will be provided to get the panel that this oscilloscope is using, so you need to handle the reference of it by yourself.
- Params:
 - [ICanvas](#)<T> canvas: the canvas that produce the graph.
 - [IPointDrawer](#) point_drawer: the point-drawer the producer will produce the graph with.
 - [IGraphProducer](#) graph_producer: a certain GraphProducer, MultiThreadOscilloscope requires a concurrent producer, which means producer.[Produce\(\)](#) can be called by

- different thread.
 - [IControlPanel](#) control_panel: the user-interface of this oscilloscope.
 - ConcurrentQueue<T> buffer: the buffer of this oscilloscope, if null, a new ConcurrentQueue will be created as the buffer, and then you could get it with attribute [Buffer](#).
 - Normal-Behaviour:
 - Pre-Condition:
 - canvas.GraphSize == point_drawer.GraphSize
 - !graph_producer.RequireConcurrentDrawer || point_drawer.IsConcurrent
 - Exception-Behaviour:
 - Exception: OscilloscopeBuildException with inner-exception: DifferentGraphSizeException
 - canvas.GraphSize != point_drawer.GraphSize
 - Exception: OscilloscopeBuildException
 - graph_producer.RequireConcurrentDrawer && !point_drawer.IsConcurrent
-

methods:

```
1 | protected T Draw(double delta_time);
```

- Summary:
 - get the current state of the panel and produce a new graph according to this. then return the graph while finish.
 - Params:
 - double delta_time: the time during which the point will be drawn on the graph. in short you'd better delivery the time span from the latest call of this method.
 - Normal-Behaviour:
 - Post-Condition:
 - a new graph with type T will be produced and return.
-

SimpleOscilloscope

```
1 public class SimpleOscilloscope<T> : SingleThreadOscilloscope<T>;
```

- namespace: [OscilloscopeKernel](#)
- supers: [SingleThreadOscilloscope](#)
- interfaces: none
- summary:
 - the only difference with [SingleThreadOscilloscope](#) is the method [Draw\(\)](#) is public.
- constructors:

name	description
SingleThreadOscilloscope (ICanvas canvas, IPointDrawer point_drawer, IGraphProducer graph_producer, IControlPanel control_panel)	

- methods:

name	description
protected T Draw (double delta_time)	produce and get a new graph.

constructors:

```
1 protected SimpleOscilloscope(  
2     ICanvas<T> canvas,  
3     IPointDrawer point_drawer,  
4     IGraphProducer graph_producer,  
5     IControlPanel control_panel)
```

- Summary:
 - create a new oscilloscope.
- Remarks:
 - every input objects should not be used by other oscilloscope at the same time.
 - no attribute or method will be provided to get the panel that this oscilloscope is using, so you need to handle the reference of it by yourself.
- Params:
 - [ICanvas](#)<T> canvas: the canvas that produce the graph.
 - [IPointDrawer](#) point_drawer: the point-drawer the producer will produce the graph with.
 - [IGraphProducer](#) graph_producer: a certain GraphProducer, MultiThreadOscilloscope requires a concurrent producer, which means producer.[Produce\(\)](#) can be called by different thread.
 - [IControlPanel](#) control_panel: the user-interface of this oscilloscope.
 - ConcurrentQueue<T> buffer: the buffer of this oscilloscope, if null, a new ConcurrentQueue will be created as the buffer, and then you could get it with attribute [Buffer](#).
- Normal-Behaviour:
 - Pre-Condition:

- canvas.GraphSize == point_drawer.GraphSize
 - !graph_producer.RequireConcurrentDrawer || point_drawer.IsConcurrent
 - Exception-Behaviour:
 - Exception: OscilloscopeBuildException with inner-exception: DifferentGraphSizeException
 - canvas.GraphSize != point_drawer.GraphSize
 - Exception: OscilloscopeBuildException
 - graph_producer.RequireConcurrentDrawer && !point_drawer.IsConcurrent
-

methods:

```
1 | public T Draw(double delta_time);
```

- Summary:
 - it will call and return the result of [SingleThreadOscilloscope.Draw](#) directly.
 - get the current state of the panel and produce a new graph according to this. then return the graph while finish.
 - Params:
 - double delta_time: the time during which the point will be drawn on the graph. in short you'd better deliver the time span from the latest call of this method.
 - Normal-Behaviour:
 - Post-Condition:
 - a new graph with type T will be produced and return.
-

TimeCountedOscilloscope

```
1 | public class TimeCountedOscilloscope<T> : SingleThreadOscilloscope<T>;
```

- namespace: [OscilloscopeKernel](#)
- supers: [SingleThreadOscilloscope](#)
- interfaces: none
- summary:
 - the only difference with [SimpleOscilloscope](#) is the method [Draw\(\)](#) will use a built-in watch to get delta-time.
- constructors:

name	description
SingleThreadOscilloscope (ICanvas canvas, IPointDrawer point_drawer, IGraphProducer graph_producer, IControlPanel control_panel)	

- methods:

name	description
protected T Draw ()	produce and get a new graph.

constructors:

```
1 | protected TimeCountedOscilloscope(  
2 |     ICanvas<T> canvas,  
3 |     IPointDrawer point_drawer,  
4 |     IGraphProducer graph_producer,  
5 |     IControlPanel control_panel)
```

- Summary:
 - create a new oscilloscope.
- Remarks:
 - every input objects should not be used by other oscilloscope at the same time.
 - no attribute or method will be provided to get the panel that this oscilloscope is using, so you need to handle the reference of it by yourself.
- Params:
 - [ICanvas](#)<T> canvas: the canvas that produce the graph.
 - [IPointDrawer](#) point_drawer: the point-drawer the producer will produce the graph with.
 - [IGraphProducer](#) graph_producer: a certain GraphProducer, MultiThreadOscilloscope requires a concurrent producer, which means producer.[Produce](#)() can be called by different thread.
 - [IControlPanel](#) control_panel: the user-interface of this oscilloscope.
 - ConcurrentQueue<T> buffer: the buffer of this oscilloscope, if null, a new ConcurrentQueue will be created as the buffer, and then you could get it with attribute [Buffer](#).
- Normal-Behaviour:

- Pre-Condition:
 - `canvas.GraphSize == point_drawer.GraphSize`
 - `!graph_producer.RequireConcurrentDrawer || point_drawer.IsConcurrent`
 - Exception-Behaviour:
 - Exception: `OscilloscopeBuildException` with inner-exception: `DifferentGraphSizeException`
 - `canvas.GraphSize != point_drawer.GraphSize`
 - Exception: `OscilloscopeBuildException`
 - `graph_producer.RequireConcurrentDrawer && !point_drawer.IsConcurrent`
-

methods:

```
1 | public T Draw();
```

- Summary:
 - it will get `delta_time` with built-in watch.
 - it will call and return the result of [SingleThreadOscilloscope.Draw](#) directly.
 - get the current state of the panel and produce a new graph according to this. then return the graph when finish.
 - Params:
 - `double delta_time`: the time during which the point will be drawn on the graph. in short you'd better deliver the time span from the latest call of this method.
 - Normal-Behaviour:
 - Post-Condition:
 - a new graph with type `T` will be produced and return.
-

MultiThreadOscilloscope

```
1 public abstract class MultiThreadOscilloscope<T>;
```

- namespace: [OscilloscopeKernel](#)
- supers: none
- interfaces: none
- summary:
 - an oscilloscope that can start a new draw-task while the old one has not finished.
 - T is the output type of this oscilloscope.
- remarks
 - this is a abstract class, if you want to use it, please try [UndrivedOscilloscope](#) or [DrivedOscilloscope](#).
 - calling [Draw\(\)](#) to start a draw-task, and after the draw-task is complete, a new graph will be put into [Buffer](#).
 - no attribute or method will be provided to get the panel that this oscilloscope is using, so you need to handle the reference of it by yourself.
- constructors:

name	description
MultiThreadOscilloscope (ConstructorTuple<ICanvas<T>> canvas_constructor,ConstructorTuple<IPointDrawer> point_drawer_constructor,IRulerDrawer ruler_drawer,IGraphProducer graph_producer,IControlPanel control_panel,ConcurrentQueue<T> buffer = null)	

- attributes:

type	name	accessor	description
ConcurrentQueue<T>	Buffer	G	the productions of this oscilloscope will be put into this buffer.

- methods:

name	description
protected void Draw (double)	get the current state of the panel and produce a new graph accoding to this.then put the new graph into Buffer

constructors:

```
1 public MultiThreadOscilloscope(  
2     ConstructorTuple<ICanvas<T>> canvas_constructor,  
3     ConstructorTuple<IPointDrawer> point_drawer_constructor,  
4     IGraphProducer graph_producer,  
5     IControlPanel control_panel,  
6     ConcurrentQueue<T> buffer = null)
```

- Summary:
 - create a new Oscilloscope.
- Remarks:
 - the control_panel and graph_producer should not be used by other oscilloscope at the same time.
 - no attribute or method will be provided to get the panel that this oscilloscope is using, so you need to handle the reference of it by yourself.
- Params:
 - [ConstructorTuple<ICanvas<T>>](#) canvas_constructor: a ConstructorTuple that can create new ICanvas.
 - [ConstructorTuple<IPointDrawer>](#) point_drawer_constructor: a ConstructorTuple that can create new IPointDrawer.
 - [IGraphProducer](#) graph_producer: a certain GraphProducer, MultiThreadOscilloscope requires a concurrent producer, which means producer.[Produce\(\)](#) can be called by different thread.
 - [IControlPanel](#) control_panel: the user-interface of this oscilloscope.
 - ConcurrentQueue<T> buffer: the buffer of this oscilloscope, if null, a new ConcurrentQueue will be created as the buffer, and then you could get it with attribute [Buffer](#).
- Normal-Behaviour:
 - Pre-Condition:
 - canvas_constructor.NewInstance().GraphSize == point_drawer_constructor.NewInstance().GraphSize
 - !graph_producer.RequireConcurrentDrawer || point_drawer_constructor.NewInstance().IsConcurrent
- Exception-Behaviour:
 - Exception: OscilloscopeBuildException with inner-exception: DifferentGraphSizeException
 - canvas_constructor.NewInstance().GraphSize != point_drawer_constructor.NewInstance().GraphSize
 - Exception: OscilloscopeBuildException
 - graph_producer.RequireConcurrentDrawer && !point_drawer_constructor.NewInstance().IsConcurrent

attributes:

```
1 | public ConcurrentQueue<T> Buffer { get; }
```

- Summary:
 - the productions of this oscilloscope will be put into this buffer.
 - the reference of buffer will never change.

methods:

```
1 | protected void Draw(double delta_time);
```

- Summary:
 - get the current state of the panel and produce a new graph according to this. then put the new graph into [Buffer](#)
- Params:
 - double delta_time: the time during which the point will be drawn on the graph. in short you'd better deliver the time span from the latest call of this method.
- Normal-Behaviour:
 - Post-Condition:
 - a new graph with type T will be produced and put into [Buffer](#)

UndrivedOscilloscope

```
1 public class UndrivedOscilloscope<T> : MultiThreadOscilloscope<T>;
```

- namespace: [OscilloscopeKernel](#)
- supers: MultiThreadOscilloscope<T>
- interfaces: none
- summary:
 - the only difference with [MultiThreadOscilloscope](#) is that the [Draw\(\)](#) of [UndrivedOscilloscope](#) is public.
- constructors:

name	description
UndrivedOscilloscope (ConstructorTuple<ICanvas<T>> canvas_constructor, ConstructorTuple<IPointDrawer> point_drawer_constructor, IGraphProducer graph_producer, IControlPanel control_panel, ConcurrentQueue<T> buffer = null)	

- methods:

name	description
void Draw (double)	call MultiThreadOscilloscope.Draw() directly.

constructors:

```
1 public UndrivedOscilloscope(  
2     ConstructorTuple<ICanvas<T>> canvas_constructor,  
3     ConstructorTuple<IPointDrawer> point_drawer_constructor,  
4     IGraphProducer graph_producer,  
5     IControlPanel control_panel,  
6     ConcurrentQueue<T> buffer = null)
```

- Summary:
 - create a new Oscilloscope.
 - the same as [MultiThreadOscilloscope](#).
- Remarks:
 - the control_panel and graph_producer should not be used by other oscilloscope at the same time.
 - no attribute or method will be provided to get the panel that this oscilloscope is using, so you need to handle the reference of it by yourself.
- Params:
 - [ConstructorTuple](#)<[ICanvas](#)<T>> canvas_constructor: a ConstructorTuple that can create new ICanvas.
 - [ConstructorTuple](#)<[IPointDrawer](#)> point_drawer_constructor: a ConstructorTuple that can create new IPointDrawer.

- [IGraphProducer](#) graph_producer: a certain GraphProducer, MultiThreadOscilloscope requires a concurrent producer, which means producer.[Produce\(\)](#) can be called by different thread.
 - [IControlPanel](#) control_panel: the user-interface of this oscilloscope.
 - ConcurrentQueue<T> buffer: the buffer of this oscilloscope, if null, a new ConcurrentQueue will be created as the buffer, and then you could get it with attribute [Buffer](#).
 - Normal-Behaviour:
 - Pre-Condition:
 - canvas_constructor.NewInstance().GraphSize == point_drawer_constructor.NewInstance().GraphSize
 - !graph_producer.RequireConcurrentDrawer || point_drawer_constructor.NewInstance().IsConcurrent
 - Exception-Behaviour:
 - Exception: OscilloscopeBuildException with inner-exception: DifferentGraphSizeException
 - canvas_constructor.NewInstance().GraphSize != point_drawer_constructor.NewInstance().GraphSize
 - Exception: OscilloscopeBuildException
 - graph_producer.RequireConcurrentDrawer && !point_drawer_constructor.NewInstance().IsConcurrent
-

methods:

```
1 | public void Draw(double delta_time);
```

- Summary:
 - it will call [MultiThreadOscilloscope.Draw\(\)](#) directly.
 - get the current state of the panel and produce a new graph according to this. then put the new graph into [Buffer](#)
- Params:
 - double delta_time: the time during which the point will be drawn on the graph. in short you'd better deliver the time span from the latest call of this method.
- Normal-Behaviour:
 - Post-Condition:
 - a new graph with type T will be produced and put into [Buffer](#)

DrivenOscilloscope

```
1 public class DrivenOscilloscope<T> : MultiThreadOscilloscope<T>;
```

- namespace: [OscilloscopeKernel](#)
- supers: MultiThreadOscilloscope<T>
- interfaces: none
- summary:
 - a multi-thread oscilloscope that contains a built-in timer.
 - it will produce graphs periodically and put them into the [Buffer](#).
- constructors:

name	description
DrivenOscilloscope (ConstructorTuple<ICanvas<T>> canvas_constructor,ConstructorTuple<IPointDrawer> point_drawer_constructor,IGraphProducer graph_producer,IControlPanel control_panel,ConcurrentQueue<T> buffer = null)	

- attributes:

type	name	accessor	description
bool	IsRunning	G	marks wheather this oscilloscope is running

- methods:

name	description
void Start (int delta_time)	start to produce graphs periodically.
void End ()	stop this oscilloscope.

constructors:

```
1 public DrivenOscilloscope(  
2     ConstructorTuple<ICanvas<T>> canvas_constructor,  
3     ConstructorTuple<IPointDrawer> point_drawer_constructor,  
4     IGraphProducer graph_producer,  
5     IControlPanel control_panel,  
6     ConcurrentQueue<T> buffer = null)
```

- Summary:
 - create a new Oscilloscope.
 - the same as [MultiThreadOscilloscope](#).
- Remarks:
 - the control_panel and graph_producer should not be used by other oscilloscope at the same time.

- no attribute or method will be provided to get the panel that this oscilloscope is using, so you need to handle the reference of it by yourself.
- Params:
 - [ConstructorTuple<ICanvas>](#) canvas_constructor: a ConstructorTuple that can create new ICanvas.
 - [ConstructorTuple<IPointDrawer>](#) point_drawer_constructor: a ConstructorTuple that can create new IPointDrawer.
 - [IGraphProducer](#) graph_producer: a certain GraphProducer, MultiThreadOscilloscope requires a concurrent producer, which means producer.[Produce\(\)](#) can be called by different thread.
 - [IControlPanel](#) control_panel: the user-interface of this oscilloscope.
 - ConcurrentQueue<T> buffer: the buffer of this oscilloscope, if null, a new ConcurrentQueue will be created as the buffer, and then you could get it with attribute [Buffer](#).
- Normal-Behaviour:
 - Pre-Condition:
 - canvas_constructor.NewInstance().GraphSize == point_drawer_constructor.NewInstance().GraphSize
 - !graph_producer.RequireConcurrentDrawer || point_drawer_constructor.NewInstance().IsConcurrent
- Exception-Behaviour:
 - Exception: OscilloscopeBuildException with inner-exception: DifferentGraphSizeException
 - canvas_constructor.NewInstance().GraphSize != point_drawer_constructor.NewInstance().GraphSize
 - Exception: OscilloscopeBuildException
 - graph_producer.RequireConcurrentDrawer && !point_drawer_constructor.NewInstance().IsConcurrent

attributes:

```
1 | public bool IsRunning { get; }
```

- Summary:
 - marks wheather this oscilloscope is running
- Remarks
 - while IsRunning is true, ths oscilloscope will produce a new graph and put it into the [Buffer](#) periodically.
- Getter

methods:

```
1 | public void Start(int delta_time);
```

- Summary:
 - the oscilloscope start to run, which means it will put a new graph into the [Buffer](#) every `delta_time`.

- Remarks:
 - be careful about the time unit of delta_time. the time unit is still defined with [Waves.UNIT_NUMBER_PRO_SECOND](#).
 - Params:
 - int delta_time: the period that this oscilloscope produce a new graph and put into the [Buffer](#).
 - Normal-Behaviour:
 - Pre-Condition:
 - IsRunning == true
 - Post-Condition:
 - stop and then restart to run.
 - IsRunning == true
 - Normal-Behaviour:
 - Pre-Condition:
 - IsRunning == false
 - Post-Condition:
 - start to run.
 - IsRunning == true
-

```
1 | public void End()
```

- Summary:
 - stop this oscilloscope.
 - Remarks:
 - if the oscilloscope is not running, nothing will happen.
 - Normal-Behaviour:
 - Pre-Condition:
 - IsRunning == true
 - Post-Condition:
 - the oscilloscope will stop producing graphs periodically
 - IsRunning == false
 - Normal-Behaviour:
 - Pre-Condition:
 - IsRunning == false
 - Post-Condition:
 - nothing will happen
-

Wave

1 | namespace OscilloscopeKernel1.wave

type	name	description