

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`.

OscilloscopeKernel

```
1 namespace OscilloscopeKernel
```

[illegible]

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 finish.
 - 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 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 according 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.
- Params:
 - [ConstructorTuple<ICanvas>](#) canvas_constructor: a ConstructorTuple that can create new ICanvas.
 - [ConstructorTuple<IPointDrawer>](#) canvas_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.IsConcurrent
- Exception-Behaviour:
 - Exception: OscilloscopeBuildException with inner-exception: DifferentGraphSizeException
 - canvas.GraphSize != point_drawer.GraphSize
 - Exception: OscilloscopeBuildException
 - graph_producer.RequireConcurrentDrawer && !point_drawer.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 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 put into [Buffer](#)

UndrivedOscilloscope

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

- namespace: [OscilloscopeCore](#).
- supers: MultiThreadOscilloscope<T>
- interfaces: none
- summary:
 - the only difference between [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
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<T>](#).
- Remarks:
 - the control_panel and graph_producer should not be used by other oscilloscope at the same time.
- Params:

- [ConstructorTuple<ICanvas>](#) canvas_constructor: a ConstructorTuple that can create new ICanvas.
- [ConstructorTuple<IPointDrawer>](#) canvas_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.IsConcurrent
- Exception-Behaviour:
 - Exception: OscilloscopeBuildException with inner-exception: DifferentGraphSizeException
 - canvas.GraphSize != point_drawer.GraphSize
 - Exception: OscilloscopeBuildException
 - graph_producer.RequireConcurrentDrawer && !point_drawer.IsConcurrent

attributes:

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](#)

Wave

1 | namespace OscilloscopeKernel.wave

type	name	description