Document of ClassLib OscilloscopeKernel

Index

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`.
- the time unit is defined with Waves.UNIT_NUMBER_PRO_SECOND. the defaute time unit is μs but most of Systerm functions use ms as the time unit, be careful!.

OscilloscopeKernel

```
namespace OscilloscopeKernel
```

type	name	description
class	MultiThreadOscilloscope	an abstract class thar describe an oscilloscope that can start a new draw-task while the old one has not finish
class	UndrivedOscilloscope	a newable MultiThreadOscilloscope with public Draw().
namespace	Wave	

type	name	description

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	describtion
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	describtion
ConcurrentQueue<T>	Buffer	G	the productions of this oscilloscope will be put into this buffer.

- methods:

name	describtion
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:

```
public MultiThreadOscilloscope(
    ConstructorTuple<ICanvas<T>> canvas_constructor,
    ConstructorTuple<IPointDrawer> point_drawer_constructor,
    IGraphProducer graph_producer,
    IControlPanel control_panel,
    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<T>> 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 requirs 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: OscillocopeBuildException with inner-exception: DifferentGraphSizeException
    - canvas.GraphSize != point_drawer.GraphSize
  - Exception: OscillocopeBuildException
    - graph_producer.RequireConcurrentDrawer && !point_drawer.IsConcurrent

---

attributes:

```
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:

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

- Summary:
  - get the current state of the panel and produce a new graph accoding 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

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

- methods:

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

constructors:

```
public UndrivedOscilloscope(
    ConstructorTuple<ICanvas<T>> canvas_constructor,
    ConstructorTuple<IPointDrawer> point_drawer_constructor,
    IGraphProducer graph_producer,
    IControlPanel control_panel,
    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.
- Params:
  - ConstructorTuple<ICanvas<T>> 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 requirs 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: OscillocopeBuildException with inner-exception: DifferentGraphSizeException
    - canvas.GraphSize != point_drawer.GraphSize
  - Exception: OscillocopeBuildException
    - graph_producer.RequireConcurrentDrawer && !point_drawer.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 accoding 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

DrivedOscilloscope

```
public class DrivedOscilloscope<T> : MultiThreadOscilloscope<T>;
```

- namespace: OscilloscopeCore
- supers: MultiThreadOscilloscope<T>
- interfaces: none
- summary:
  - a multi-thread oscilloscope that contains a built-in timer.
- remarks
  -
- constructors:

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

- attributes:

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

- methods:

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

constructors:

```
public DrivedOscilloscope(
    ConstructorTuple<ICanvas<T>> canvas_constructor,
    ConstructorTuple<IPointDrawer> point_drawer_constructor,
    IGraphProducer graph_producer,
    IControlPanel control_panel,
    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.
- Params:

    - ConstructorTuple<ICanvas<T>> 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 requirs 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: OscillocopeBuildException with inner-exception: DifferentGraphSizeException
        - canvas.GraphSize != point_drawer.GraphSize
    - Exception: OscillocopeBuildException
        - graph_producer.RequireConcurrentDrawer && !point_drawer.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 difined 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

```
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

```
namespace OscilloscopeKernel.Wave
```

type	name	description