**PGLClock.pas**

**-- Overview --**

The **PGLClock.pas** unit provides functionality for time-keeping and timed-events. It uses the platform's high-resolution timer functions (QueryPerformanceFrequency and QueryPerformanceCounter on Windows).

**Types**

**TPGLClock** - Class

**TPGLEvent**  - Class

**TPGLClockEvent** - Procedure

**TPGLTriggerType** – Enum

**TPGLClock**

**Description**

**TPGLClock** provides the user with a way to keep track of the passage of time and control the interval at which execution of other code happens. Additionally, the **TPGLClock** can "store" instances of **TPGLEvent** , to be executed at designated times and intervals.

**Properties**

**Running Boolean** | **Read Only**  
Returns True if the user has made a successful call to **TPGLClock.Start**(). Otherwise, **Running** returns False.

**Interval Double** | **Read Only**  
A time value in seconds that represents the interval at which the **TPGLClock** instance is to update.  
A returned value of '1' would indicate and update interval of 1 second. A returned value of '0.5' would indicate an update interval of half a second.

**CurrentTime Double** | **Read Only**  
A time value in seconds that represents that last polled time of the **TPGLClock** instance. The last polled time is obtained when **TPGLClock.Wait**() returns.

**LastTime Double** | **Read Only**  
A time value in seconds that represents the time at the start of the **TPGLClock** instance's last cycle.

**TargetTime Double** | **Read Only**  
A time value in seconds that represents the next earliest time that an instance of **TPGLClock** will return from **TPGLClock.Wait**(). **TPGLClock** can return from Wait() later than **TargetTime** if **Wait**() is called after the time that **TargetTime** represents.

**CycleTime Double** | **Read Only**  
The amount of time in seconds of the last completed cycle of a **TPGLClock** instance. This is computed as **TPGLClock.CurrentTime - TPGLClock.LastTime** when **TPGLClock** returns from **Wait**() and values are updated.

**ElapsedTime Double** | **Read Only**  
A time value in seconds that represents how long an instance of **TPGLClock** has been running since the last call to **TPGLClock.Start**().

**FPS Double** | **Read Only**  
The average number of cycles that the instance of **TPGLClock** completed over the last second. **TPGLClock** keeps a private count of the number of cycles and the amount of time since the previous update to **FPS**, and updates it when that time is >= 1 second. **FPS** is calculated as **TPGLClock**.**fFrames** / **TPGLClock**.**fFrameTime**, where fFrames is the number of cycles and fFrameTime is the amount of time elapsed since the last update to **FPS**.

**AverageFPS Double** | **Read Only**  
A time value in seconds that represents the average of **FPS** since a call to any of **TPGLClock.Start**() or **TPGLClock.ResetAverageFPS**().

**Ticks Int64** | **Read Only**  
The number of cycles that an instance of **TPGLClock** has completed since the last call to **TPGLClock.Start**(). Calling **TPGLClock.Stop**() will reset this value.

**ExpectedTicks Int64** | **Read Only**  
The estimated number of cycles that an instance of **TPGLClock** should complete in 1 second, given its current interval.

**CatchUpEnabled Boolean** | **Read, Write**  
Returns if the **TPGLClock** instance has Catch Up enabled. The user can change this value directly with the assignment operator **:=**.  
If Catch Up is enabled, then when the **TPGLClock** instance updates after returning from **Wait**(), it will adjust **TargetTime** to be exactly the time at the last call to **TPGLClock.Start**() + **TPGLClock**.**Ticks** \* **TPGLClock**.**Interval**. Under circumstances where **TPGLClock** rarely updates slower than **Interval** and only by small variations, this will result in **TPGLClock** returning from **Wait**() very slightly before or very slightly after **Interval** seconds since the last update. Under circumstances where **TPGLClock** regularly returns from **Wait**() after **TargetTime** or by large deviations from **Interval**, this can cause **TPGLClock** to execute multiple cycles very quickly, resulting in noticeably uneven update intervals.

**Constructors**

**constructor Create(AFPS: Integer = 60)**   
**constructor Create(AInterval: Double = 0.0166666)**  
 ***AFPS*** - The number of cycles (or frames) **TPGLClock** should complete each second. This sets the **Interval** to double(1 / AFPS).  
 ***AInterval*** - The desired duration in seconds of a **TPGLClock** cycle.

*-- Description --*  
**TPGLClock** .**Create**() returns a **TPGLClock** object. When a new instance of **TPGLClock** is created, it calls a private member function **Init**(), which sets all member fields to 0, save for the **Interval** which is set to the value passed by the user, and the CPU clock frequency is polled and cached.

**Procedures/Functions**

**procedure** **Start()**

*-- Description --*  
Sets the **TPGLClock** instance's **Running** property to true, assigns **CurrentTime** the current CPU time in seconds, calculates the next **TargetTime**, and sets **InitTime** to **CurrentTime**.

**procedure** **Stop()**

*-- Description --*  
Sets the **TPGLClock** instance's **Running** property to false and calls the private member function **Init**() to reset all member fields.

**procedure** **Wait()**

*-- Description --*  
Stalls execution of the thread by entering a loop until **TPGLClock**.**GetTime**() returns a value that greater than or equal to **TargetTime**.

**procedure** **WaitForStableFrame()**

*-- Description --*  
Continuously calls **TPGLClock**.**Wait**() until the instance caches an **FPS** greater than or equal to 99% of (1 / Interval). In effect, this blocks execution until the **TPGLClock**’s cycles-per-second/frames-per-second is approaching the rate desired by the user.

**procedure** **SetIntervalInSeconds(AInterval: Double)**  
**AInterval** - The value to set the **TPGLClock**'s update interval to.

*-- Description --*  
Immediately changes the value of **Interval**. **Interval** is set to abs(AInterval) so as to disallow negative values. Does not affect execution if called while **TPGLClock** is running.

**procedure** **SetIntervalInFPS(AInterval: Double)**  
**AInterval** - The desired frames-per-second/cycles-per-second.

*-- Description --*  
Immediately changes the value of **Interval**. **Interval** is calculated as abs(1 / AInterval) so as to disallow negative values. Does not affect execution if called while **TPGLClock** is running.

**function** **GetTime(): Double**

*-- Description --*  
Returns the current CPU time in seconds. Does not affect **CurrentTime** or any other stored time values. Internally, **TPGLClock** calls **GetTime**() continuously during **Wait**() in order to block execution of code until **GetTime**() >= **TargetTime**, using the last value returned from **GetTime**() as the new **CurrentTime**.

**TPGLEvent**

**Description**

**TPGLEvent**  is an object that describes an "event" that the user wishes to happen at a pre-determined time or at an interval. **TPGLEvent**  must be used in conjunction with **TPGLClock**. **TPGLEvent**  is assigned a **TPGLClock** "owner" either at the time of or after creation. The **TPGLClock** owner caches a list of "owned" instances of **TPGLEvent**  , and checks conditions during updates to decide whether or not a **TPGLEvent**  should execute it's **EventProc**. A **TPGLEvent**  is either "trigger on time" or "trigger on interval". In the former case, the **EventProc** should be executed once at the designated trigger time. In the latter, the **EventProc** should execute at interval after the time that the **TPGLEvent**  was made active. "Trigger on Interval" events can execute once, or be set to repeating.

**Properties**

**Owner TPGLClock** | **Read Only**  
Returns the instance of **TPGLClock** that the **TPGLEvent**  has been assigned to. If **TPGLEvent**  has not been assigned an **Owner**, returns nil.

**Active Boolean** | **Read and Write**  
Returns the **Active** status of a **TPGLEvent**  . If a **TPGLEvent**  is owned by a **TPGLClock** and is **Active**, then the **TPGLClock** will allow it to execute its **EventProc** when time conditions are met. Otherwise, if the **TPGLEvent**  is not **Active**, it is not evaluated by its **owner** and its **EventProc** will not be executed.

The user can attempt to set **Active** to True or False at any point. If the **TPGLEvent**  does not have an **Owner**, its **TriggerType** is pgl\_trigger\_on\_interval and **TriggerInterval** = 0, or its **TriggerType** is pgl\_trigger\_on\_time and the **Owner's** **CurrentTime** > **TriggerTime**, an attempt to set **Active** to True will fail. An attempt to set **Active** to False will always succeed in the case that **Active** is True.

**Repeating Boolean** | **Read, Write**  
Returns if a **TPGLEvent**  with **TriggerType** pgl\_trigger\_on\_interval is set to repeat execution of its **EventProc**. If a **TPGLEvent** 's **TriggerType** is pgl\_trigger\_on\_time, Repeating will return False, even if **Repeating** was set to True while the **TPGLEvent**  had a **TriggerType** of pgl\_trigger\_on\_time.

The user can attempt to set **Repeating** to True or False at any point. If the **TPGLEvent** 's **TriggerType** is pgl\_trigger\_on\_time, the attempt will fail. If **Repeating** is set to true and then the **TriggerType** is set to pgl\_trigger\_on\_time, **Repeating** will not automatically be set to False, although it will return False until the **TriggerType** is set back to pgl\_trigger\_on\_interval.

**EventProc TPGLClockEvent** | **Read, Write**  
A procedure that is to be executed when the **TPGLEvent** 's time conditions have been met. **TPGLClockEvent** is defined as 'procedure()', or a procedure that takes no parameters.

The user can assign or set **EventProc** to nil at any point. **TPGLEvent**  does not require an assigned **EventProc** in order to be set as **Active**. Simply, nothing will happen when its time conditions are met.

**TriggerType TPGLTriggerType** | **Read, Write**  
A value that determines what kind of trigger the **TPGLEvent**  has and how it is handled by its **TPGLClock** owner. Must be one of either pgl\_trigger\_on\_time or pgl\_trigger\_on\_interval.

A value of pgl\_trigger\_on\_time means that the **TPGLEvent** 's **EventProc** will only execute once, and then the **TPGLEvent** 's **Active** status will be set to false, though the status is set to false *before* the **EventProc** is executed, so the user can update the **TriggerTime** and reset the **Active** status in the **EVentProc**, thereby effectively having a repeating event.

A value of pgl\_trigger\_on\_interval means that the **TPGLEvent** 's **EventProc** will execute every time an **Interval** amount of time has passed since being assigned to a **TPGLClock** and being set to **active**, or since the last time the **EventProc** was executed. If **Repeating** is False, then the **Active** status will be set to False and the **EventProc** will not continue to be executed.