Getting things done at time, every time

This is all about getting things done at the right time in an easy way, using as few resources as possible.

Typically there are two traditional ways to schedule anything in Delphi/C++Builder. Using a TTimer or writing some custom code that extend the TThread class.

Using a TTimer works ok for most parts, but is not terribly exact in its timing, and it relies on Windows messages, which means it will only fire when your GUI is not busy with other stuff, and due to its Windows message requirements, it's not very thread friendly.

By subclassing TThread, you can make your own scheduler solution, which can run at fairly precise intervals (it's up to you to calculate the right timing), and which can run completely independent of Windows messages. So it can handle your events independently of whats running on the foreground GUI thread.

Subclassing TThread unfortunately takes up one whole thread, only for the purpose of your particular interval event, and its slightly cumbersome to use, specially for people who are not used to multithreaded applications.

This is where kbmMW's new TkbmMWScheduler class in kbmMW Enterprise Edition, comes into play.

It basically is the best of both worlds. It is easy to use, and it is selectable precise. In addition it includes many more features for setting start/stop time and interval settings.

The TkbmMWScheduler class is both a container holding any number of scheduled events, and a manager managing the scheduled events, making sure they will be executed at the right time, via its multithreaded subsystems.

kbmMW comes with one standard instance of a TkbmMWScheduler which is called kbmMWScheduler, and which kbmMW itself uses. You can use it too if you want to, but you can also create your own instances of TkbmMWScheduler if you want to have your completely own scheduler.

So how do we schedule something to happen, say every 5 seconds, and start it up?

```
kbmMWScheduler.Schedule(OnScheduledEvent).EverySecond(5).Activate(true);
…
```

OnScheduledEvent is the event handler to call when your code is to run.

```
function TForm1.OnScheduledEvent(const AScheduledEvent:IkbmMWScheduledEvent):boolean;
begin
     .. do something ..
     Result:=true;
end;
```

The event will run every 5 seconds, forever until your application closes. So what if you would like to disable the event at various times?

Well, we need to be able to identify the scheduled event. One way is to remember the returned event object.

```
var
  myevent:IkbmMWScheduledEvent;
…
myevent:=kbmMWScheduler.Schedule(OnScheduledEvent).EverySecond(5).Activate(true);
…
```

Then we can simply deactivate or activate the event when we want:

```
myevent.Active:=false;
```

A different way to get access to the scheduled event, is to remember its ID. The ID is a string GUID value which can be obtained via myevent.ID

You can later refer to the event via the ID using the function GetByID.

```
ev:=kbmMWScheduler.GetByID('someid');
```

If the event ID can't be found, nil is returned.

Finally we can name an event when we schedule it.

```
kbmMWScheduler.Schedule(OnScheduledEvent).NamedAs('5secevent').EverySecond(5).Activate(true);
```

Then you can refer to it via the function GetByName.

```
ev:=kbmMWScheduler.GetByName('5secevent');
```

This 5 second event will default be scheduled to be a so called relaxed event. Scheduled events can either be relaxed events or precise events. Internally the scheduler take advantage of custom threads to run the events. To avoid the problem of each event requiring a thread of its own, kbmMW is able to use thread pooling.

Relaxed events are events which participate in the thread pooling mechanism. That means that the event will not take up any resources, until the event is actually going to run. Then a thread will be assigned to it (picked from a pool of threads to avoid overhead of constantly starting/stopping threads), and it will be run. The moment the event code has finished, the thread will be returned to the thread pool, to be used by another event when its needed.

This means that a limited number of threads will be allocated, regardless of how many events you will defined.

kbmMW will only allocate new threads until the limit of the thread pool, and only when they are needed.

So if your events are never running at the same time, one single thread will do all the work. If events are colliding time wise, which most likely will happen if you have scheduled multiple events, kbmMW will handle that by either allocating new threads (until the thread pool limit), or queuing the scheduled event.

This is the reason why this is called relaxed scheduled events, because there is a slight chance that the event will be a little bit delayed, if there are lots of other relaxed events running, and no more vacant threads available in the thread pool.

For absolutely most purposes, it doesn't matter if an event occasionally is delayed slightly, but if you need to schedule an event which must be run at expected intervals, you can declare the event to be a precise event. This results in the scheduled event being assigned its own private thread.

Notice though that the OS might still delay your event slightly depending on the load of your system. This is out of kbmMW's control.

```
kbmMWScheduler.Schedule(OnScheduledEvent).NamedAs('5secevent').EverySecond(5).Precise.Activate(true);
```

All events scheduled to be run at less than 2 sec. intervals will default be declared precise by the scheduler. If you want to let them run in relaxed mode, you can also specify that, and thus save system resources.

```
kbmMWScheduler.Schedule(OnScheduledEvent).NamedAs('1secevent').EverySecond(1).Relaxed.Activate(true);
```

If your event code is going to update some visual stuff on screen via the VCL components, then you must define the event as synchronized. The VCL components require all updates in them to be run within the main application thread (the GUI thread). By specifying the Synchronized flag, kbmMW will make that happen for you. Please notice that a synchronized scheduled event may not appear to run exactly at time, due to the required internal synchronization of threads, and thus it doesn't make much sense in declaring such events as precise.

```
kbmMWScheduler.Schedule(...)...EverySecond(5).Synchronized.Activate(true);
```

For now we have only scheduled events on full second intervals. There are many more interval options available.

EveryMSecond(n) – Schedules for every n milliseconds (1/1000 second)
EveryMinute(n) – Schedules for every n minutes
EveryHour(n) – Schedules for every n hours
EveryDay(n) – Schedules for every n days
EveryMonth(n) – Schedules for every n months
EveryYear(n) – Schedules for every n years

EverySecond, EveryMinute, EveryHour, EveryDay, EveryMonth and EveryYear takes fractional values, so to schedule something to run every half hour you can use EveryHour(0.5).

You can also use a combination of them all as you wish. The scheduler will automatically calculate the combined interval. Let's say you want to have something to run every 30 hours. You can do that in multiple ways, for example:

```
.EveryDay(1).EveryHour(6).
```

or

```
.EveryHour(30).
```

It may seem more practical and readable to use EveryHour(30) in this case. Another example could be that you want the event to run every 2 ½ months and 3 hours. Then you could use

```
.EveryMonth(2.5).EveryHour(3).
```

You can also schedule something to happen at specific weekdays

```
.AtWeekDays([mwwdSunday,mwwdWednesday]).
```

Further you may want the events only to run after a specific date, or until a specific date

```
.StartingAt(TkbmMWDateTime.Create('2015-11-22T15:00:00GMT')).
.EndingAt(TkbmMWDateTime.Create('2015-12-22T15:00:00GMT')).
```

And you may want to delay the initial run (in this case by 10 seconds)

```
.DelayInitial(TkbmMWDuration.Create(10)).
```

In addition to calling an event method, kbmMW's scheduler supports calling any class instance that implements the interface IkbmMWOnScheduledEvent. The interface contains a single function

```
function OnScheduledEvent(const AScheduledEvent:IkbmMWScheduledEvent):boolean;
```

which will be called when the scheduled event is to be triggered.

Further you can in Delphi also provide an anonymous function, like this

```
FScheduler.Schedule(
  function(const AScheduledEvent:IkbmMWScheduledEvent):boolean
  begin
       mData.Lines.Add('anonymous:'+AScheduledEvent.Name);
       Result:=true;
  End
).NamedAs('Relaxed event Every 2 secs').EverySecond(2).Synchronized.Activate(true);
```

You can define that the event is only to be run once or repeatedly (default).

```
.Occurs(mwsoOnce).
```

When set to once, the event will be automatically deactivated after first run.

As you may have seen, the new kbmMW scheduler is quite versatile while still very easy to use.

Happy scheduling

Kim/C4D