maXbox

maXbox Starter 2

Start with OO Programming

1.1 First Step

Today we spend some time in OO Programming (Object Oriented). Hope you did already work with the first Starter and the box at:

http://www.softwareschule.ch/download/maxbox_starter.pdf

This lesson will introduce you to objects, classes and events. So what's an object? An object consists of methods, properties and in many cases events. I'll show you all three of them.
Properties represent the data contained in the object. Methods are the actions the object can perform. Events are conditions or signals the object can react to. All objects descend from an ancestor object (class), in our case from `TObject`.

1.2 Get the Code

As you already know the tool is split up into the toolbar across the top, the editor or code part in the centre and the output window at the bottom.

☝ In maXbox you can't create new classes, because we use the classes from the inbuilt VCL (Visual Component Library).

Before this starter code will work you will need to download maXbox from the website. It can be downloaded from http://sourceforge.net/projects/maxbox or from http://www.softwareschule.ch/maxbox.htm (you'll find the download maxbox2.zip top left on the site). Once the download has finished, unzip the file, making sure that you preserve the folder structure as it is. If you double-click `maxbox2.exe` the box opens a default program. Test it with F9 or press **Compile** and you should hear a sound. So far so good now we'll open our example.

`59_timerobject_starter2.txt`

The **`Load`** button will present you in `/examples` with a list of Programs stored within your directory as well as a list of `/exercises` with defective Pascal code to be fixed for training purposes.
Alternatively you can download the file from:
http://www.softwareschule.ch/download/59_timerobject_starter2.txt
Use the `Save Page as…` function of your browser[1] or load it from `examples` (or wherever you stored it). Now let's take a look at the code of this project. Our first line is

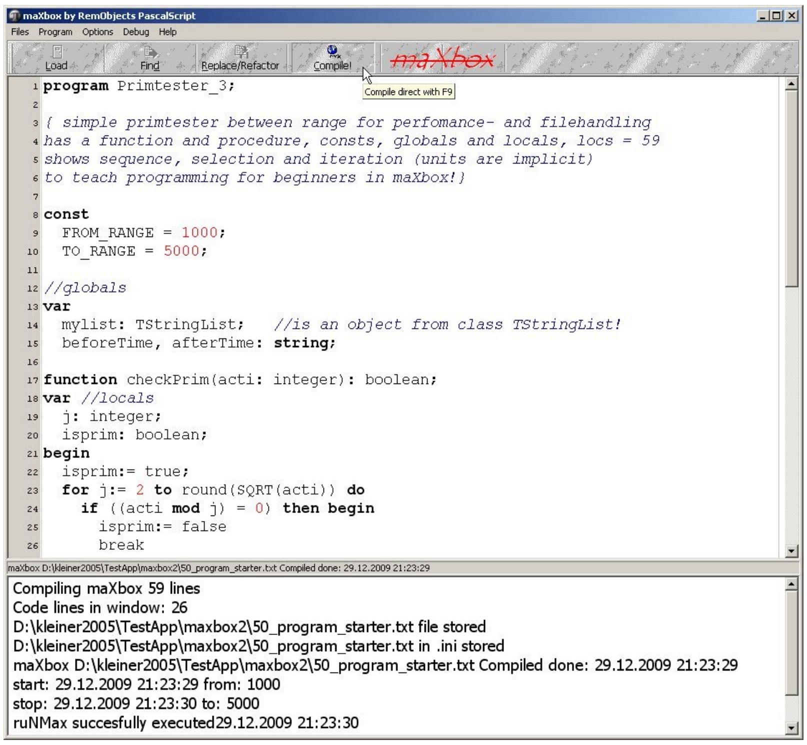
```
1 program TimerEvent_Object;
```

[1] Or copy & paste

We have to name the game, means the program's name is `TimerEvent_Object`.

☞ This example requires a `TTimer` object and a `TObject` too. The program makes a series of calls to an event handler named `timer1Event()`, which is triggered as many times you wish (invoked) from the `onTimer` event.

Most of the functions and objects we use like `showMessage()` or `TTimer` are implicit in a library (or unit). A library is a collection of code, which you can include in your program. By storing your commonly used code in a library, you can reuse code for many times in different projects and also hide difficult sections of code from the developer. Once a unit or an object is tested it's stable to use.


maXbox by RemObjects PascalScript
Files Program Options Debug Help
Load
Find
Replace/Refactor
Compile!
maXbox
Compile direct with F9

```
program Primtester_3;

{ simple primtester between range for perfomance- and filehandling
has a function and procedure, consts, globals and locals, locs = 59
shows sequence, selection and iteration (units are implicit)
to teach programming for beginners in maXbox!}

const
  FROM_RANGE = 1000;
  TO_RANGE = 5000;

//globals
var
  mylist: TStringList;    //is an object from class TStringList!
  beforeTime, afterTime: string;

function checkPrim(acti: integer): boolean;
var //locals
  j: integer;
  isprim: boolean;
begin
  isprim:= true;
  for j:= 2 to round(SQRT(acti)) do
    if ((acti mod j) = 0) then begin
      isprim:= false
      break
```

```
maXbox D:\kleiner2005\TestApp\maxbox2\50_program_starter.txt Compiled done: 29.12.2009 21:23:29
```

```
Compiling maXbox 59 lines
Code lines in window: 26
D:\kleiner2005\TestApp\maxbox2\50_program_starter.txt file stored
D:\kleiner2005\TestApp\maxbox2\50_program_starter.txt in .ini stored
maXbox D:\kleiner2005\TestApp\maxbox2\50_program_starter.txt Compiled done: 29.12.2009 21:23:29
start: 29.12.2009 21:23:29 from: 1000
stop: 29.12.2009 21:23:30 to: 5000
ruNMax succesfully executed29.12.2009 21:23:30
```

1: The maXbox

Next we learn how a constant works. Constants are fixed numeric or character values represented by a name. Constants can't be changed[2] while a program runs. A section starts with the word const:

```
06 Const  MILLISECONDS = 1000; //one second
```

The lines 8 to 9 of the program contain the declaration of 2 variables `myTimer` and `glob_count`. So line 8 is our object variable, namely an object of type `TTimer` (as its name suggests an object with clock or watch functions). So each variable has a type.

☞ A type is essentially a name for a kind of data. When you declare a variable you must specify its type, which determines the set, range and storage of values the variable can hold and the operations (functions, procedures or methods) that can be performed on it. A method is just a procedure or function associated with a class!

```
08 var myTimer: TTimer; //is an object of class TTimer!
09     glob_count: byte;
```

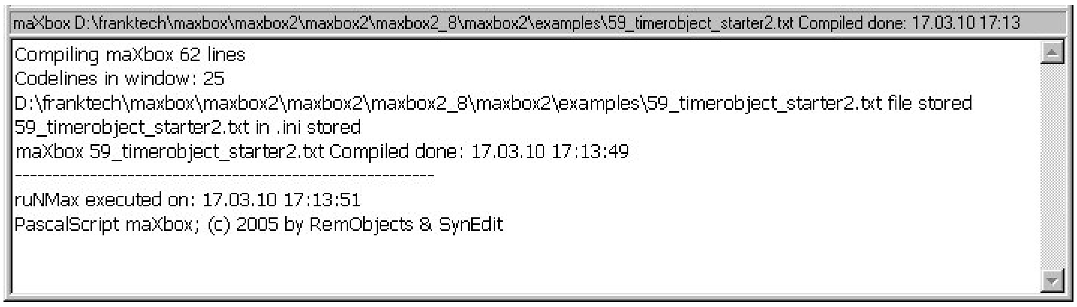
A variable is a place to store data. In this case you are setting up a variable of type `TTimer` and `byte`. Imagine a variable as a small box (in maXbox;)) where you can keep things. A variable is called a variable because its value can change during the programs execution. But watch out not every name

[2] You can only change before

can hold a variable because you can't use any keywords like object, class, set, while, case, if then etc as variable names.
Now comes the time to explain the difference between an object and a class, for many of my students a bit dark chapter ;). An **Object** is a software bundle of related state and behaviour (methods). So `myTimer` is a special variable, it's an object variable of a special type called a class:

A Class is a blueprint or prototype from which individual objects are created (later on we create an object). Methods appear in a class declaration as function or procedure headings, with no body. Defining declarations for each method occur elsewhere in the program.

So far we have learned something about constants and variables and the difference between an object and a class. We say an object is an instance of a class. Now it's time to run our program at first with F9 (if you haven't done yet). The program generates 6 message boxes, the first to start and the next five messages to show the time in an interval of a second.



```
maXbox D:\franktech\maxbox\maxbox2\maxbox2\maxbox2_8\maxbox2\examples\59_timerobject_starter2.txt Compiled done: 17.03.10 17:13
Compiling maXbox 62 lines
Codelines in window: 25
D:\franktech\maxbox\maxbox2\maxbox2\maxbox2_8\maxbox2\examples\59_timerobject_starter2.txt file stored
59_timerobject_starter2.txt in .ini stored
maXbox 59_timerobject_starter2.txt Compiled done: 17.03.10 17:13:49
------------------------------------------------------------
ruNMax executed on: 17.03.10 17:13:51
PascalScript maXbox; (c) 2005 by RemObjects & SynEdit
```

2: The Output Window

The **`Compile`** button is also used to check that your code is correct, by verifying the syntax before the program starts. When you run this code you will see that we catch 6 message boxes in a set count up from 1 to 5.

So let's jump to line 12. This line is our **procedure** called `timer1Event()`.[3]

```
12 procedure timer1Event(Sender: TObject);
```

A procedure (call) consists of the name of a procedure (with or without qualifiers), followed by a parameter list like `(Sender: TObject)`. Functions return a value where procedures must not! Most functions and procedures require parameters of specific types.

```
12 procedure timer1Event(Sender: TObject);
13 begin
14  incb(glob_count)
15  //ShowMessage('this '+intToStr(glob_count)+' time') //never get
16  if glob_count >= 5 then begin
17    myTimer.enabled:= false;
18    myTimer.Free;
19  end;
20  ShowMessage('this is from timer event '+intToStr(glob_count)+' time: '
21                     +timeToStr(time))
22 end;
```

[3] In this case an event handler

In line 12 we have a parameter named sender of type `TObject`. In a Delphi event handler, the sender parameter indicates which component has produced the event and therefore is called the sender. I know you need some time or experience to understand this; anyway when the timer starts, the event handler `timer1Event` gets the event from the `myTimer.onTimer` event.

It is also possible to get information back from the handler.
Once you determine when the event occurs (later on in the main program), you must define how you want the event handled. And this goes like this:

In our first `if` Statement we check how often we want to receive the event:

```
16  if glob_count >= 5 then begin
```

If the condition `(glob_count >=5)` is true, then we stop and close the timer but as long it's not true we show in each call the time in a message box.

The `else` keyword of the `if` statement is optional;
We could run this code as many times as we like simply by changing the global count condition for example from 5 to 10. After each event of the loop, the counter in line 14 is incremented. Consequently, `glob_count` is called a loop counter.

```
14 incb (glob_count);
```

After we got called 5 times we close the event

```
18 myTimer.Free;
```

Free (Destroy) deactivates the timer object by setting Enabled to False before freeing the resources required by the timer object.

If we don't' close the timer event by a condition the showmessage would go on by infinitive, as long your machine is running, so you can use it like a clock or heart beat.

Try to change the const declaration for example from 1000 to 2000 in line 6! How many messages you get and how long it'll last?

A **package** or unit is a namespace for organizing classes and interfaces in a logical manner.

1.3 The Main Routine (TimeTime)

An OP program must have a main routine between begin and end. The main routine is run once and once only at the start of the program (after you compiled) and is where you will do the general instruct-ions and the main control of the program.
Now main part: Use the Timer component to trigger an event, either one time or repeatedly, after a measured interval. Write the code that you want to occur at the specified time inside the timer component's `OnTimer` event. I would say line 30 is the core of your program:

```
30  myTimer.onTimer:= @timer1Event;
```

Of course the main program starts in line 25. In line 26 we initialise the global counter with 0. What's the meaning of initialising? It has the purpose to set `glob_count` to a known value, to 0 in this case. In line 29 we create our object. Call create to instance a timer at runtime. Given the declaration of line 8 (`var myTimer: TTimer;`) you can NOW create an instance of `TTimer` as follows:

```
29  myTimer:= TTimer.Create(self);
```

The identifier self references the object in which the method is called.

A class type must be declared and given a name before it can be instantiated.
For example, here is the declaration of the `TTimer` class from the `ExtCtrls` unit.

`TTimer` is used to simplify the calling of the Windows API timer functions `SetTimer` and `KillTimer`, and to simplify the processing of WM_TIMER messages. Use one timer component for each timer in the application. `TTimer` descends from `TComponent` (in the Classes unit), inheriting most of its members. Each member is declared as private, protected, or public (this class has no published members); for explanations of these terms, see 1.4 Appendix declaration of class members.

```
24 //main program
25 begin
26   glob_count:= 0;
27   ShowMessage('Press OK to start with timer at: '
28                  +dateToStr(date)+': '+timeToStr(time))
29   myTimer:= TTimer.Create(self);
30   myTimer.onTimer:= @timer1Event;
31   myTimer.interval:= MILLISECONDS;
32 end.
```

In line 27 we are telling the compiler to call a procedure named `ShowMessage`; It simply returns the current system date and time. Date and time are also functions ? We pass no parameters to it (empty parenthesis or no parenthesis).
In line 31 we specify the amount of elapsed time in milliseconds before the timer event is triggered, we use the interval property. To discontinue a timed event, set the timer component's enabled property to false, which we set in line 17.

```
30   myTimer.onTimer:= @timer1Event;
31   myTimer.interval:= MILLISECONDS;
```

The interval determines how frequently the `OnTimer` event occurs. Each time the specified interval passes, the `OnTimer` event occurs and calls our event handler in line 12.

Use Interval to specify any cardinal value as the interval between `OnTimer` events. The default value is 1000 (one second). Interesting point: A 0 value is valid, however the timer will not call an `OnTimer` event for a value of 0 ;).
So an event handler is called a handler because it responds to events that occur while the program is running. Event handlers are assigned to specific events like we do in line 30.

How can you change the situation that every minute an event occurs? Yes, you define a constant with sum of 60000:

```
Const
  const MILLISECONDS = 1000 * 60;
```

And then you get every minute a message box five times.

In conclusion the `TTimer` is used to simplify calling the system timer functions. Use one timer component for each timer in the application. The execution of the timer occurs through its `OnTimer` event. `TTimer` has an Interval property that determines how often the timer event occurs.

We can state that for educational reasons its better to begin in a procedural way of programming followed by OOP. In procedural programming (which predates OOP), you create constants, variables, functions, and statements. Those have to be learned before OOP. I would say that even OOP is tougher to learn and much harder to master than procedural coding, you learn it better after procedural thinking. The truth is that almost anything that can be done procedurally can be done using objects and vice versa.

With a right mouse click on the pop up menu in the editor you can set so called breakpoints and study the source from which point you want or use different steps to execute and stop from. As a last exercise you may reuse the program as a game:

Try to shoot down the messagebox before the next event occurs. Set the interval back to one second or try it faster with half second.

Think about the so called Call Tree: [1]

```
29 myTimer:= TTimer.Create(self);
30 myTimer.onTimer:= @timer1Event
12 timer1Event;
```

That's all folks of primetime, may the source be with you for the second step and I hope to see you from time to time in maXbox. A third Starter "Modular Programming" is also available:

http://www.softwareschule.ch/download/maxbox_starter3.pdf

Feedback @
max@kleiner.com

Many thanks to the Reviewer Team and Beat Strähl for C++ Support

Literature:
Kleiner et al., Patterns konkret, 2003, Software & Support

Links of maXbox:

http://www.softwareschule.ch/maxbox.htm

http://www.softwareschule.ch/

http://sourceforge.net/projects/maxbox

The Document itself:

http://www.softwareschule.ch/download/maxbox_starter2.pdf

1.4 Appendix

1.4.1 TTimer Declaration

```
  TTimer = class(TComponent)
  private
    FInterval: Cardinal;
    FWindowHandle: HWND;
    FOnTimer: TNotifyEvent;
    FEnabled: Boolean;
    procedure UpdateTimer;
    procedure SetEnabled(Value: Boolean);
    procedure SetInterval(Value: Cardinal);
    procedure SetOnTimer(Value: TNotifyEvent);
    procedure WndProc(var Msg: TMessage);
  protected
    procedure Timer; dynamic;
  public
    constructor Create(AOwner: TComponent); override;
    destructor Destroy; override;
  published
    property Enabled: Boolean read FEnabled write SetEnabled default True;
    property Interval: Cardinal read FInterval write SetInterval default
1000;
    property OnTimer: TNotifyEvent read FOnTimer write SetOnTimer;
  end;
```

1.4.2 Code in OP

```
program TimerEvent_Object;
// timerevent with time dialog synchronisation or event triggering, loc's=59
// when messagebox is to early (line 25) the break condition never arrives

const MILLISECONDS = 1000 * 60; //one minute

var myTimer: TTimer; //is an object of class TTimer!
    glob_count: byte;

procedure timer1Event(Sender: TObject);
begin
  incb(glob_count)
  //ShowMessage('this '+intToStr(glob_count)+' time') //never get
  if glob_count >= 5 then begin
    myTimer.enabled:= false;
    myTimer.Free;
  end;
  ShowMessage('this is from timer event '+intToStr(glob_count)+' time: '
                 +timeToStr(time))
end;
```

```
//main program
begin
  glob_count:= 0;
  ShowMessage('Press OK to start with timer at: '
                 +dateToStr(date)+': '+timeToStr(time))
  myTimer:= TTimer.Create(self);
  myTimer.onTimer:= @timer1Event;
  myTimer.interval:= MILLISECONDS;
end.
```

1.4.3 Code in C++ with Qt

```
/////////////// file mainclass.h
/////////////////////////////////////////////////////////////////

#ifndef MAINCLASS_H
#define MAINCLASS_H
#include <QObject>

class QMainClass : public QObject
{
  Q_OBJECT
  public:
    QMainClass();
  private:
    int counter;
  private slots:
    void onTimeout();
};

#endif

/////////////// end of file mainclass.h
//////////////////////////////////////////////////////////

/////////////// file mainclass.cpp
///////////////////////////////////////////////////////////////

#include <QCoreApplication>
#include "mainclass.h"
#include <stdio.h>

QMainClass::QMainClass()
  : QObject(0), counter(0)
{
}

void QMainClass::onTimeout()
{
  printf("counter value %d\n",++counter);
  if (counter == 5)
    QCoreApplication::exit(0);  // ends the Qt event loop "app.exec()"
politely
}
/////////////// end of file mainclass.cpp
/////////////////////////////////////////////////////////////

/////////////// file main.cpp
/////////////////////////////////////////////////////////////////////////

#include <QCoreApplication>
#include <QTimer>
#include "mainclass.h"

int main(int argc, char *argv[])
{
    QCoreApplication app(argc, argv);

    QMainClass mainClassInstance;
    QTimer *timer = new QTimer(0);
    timer->setSingleShot(false);
      mainClassInstance.connect(timer,SIGNAL(timeout()),
      &mainClassInstance,SLOT(onTimeout()));
    timer->start(1000);
    return app.exec();
    delete timer;
}

/////////////// end of file main.cpp
///////////////////////////////////////////////////////////////////
```

🚩	Aim
👉	Introduction to a new topic
☝	Important Information
📄	Code example
👓	Analysis of Code
⌨	Lesson
☑	Test
📖	Summary
👍	Congratulation