maXbox

maXbox Starter 11

Start with Form Programming

1.1 Find the Form

Today we spend time in programming forms on an application. `TApplication`, `TScreen`, and `TForm` are the classes that form (yes I mean form) the backbone of your applications by controlling the behaviour of a project. The `TApplication` class forms the foundation of an application by providing properties and methods that encapsulate the behaviour of a standard script or program.
`TScreen` is used at runtime to keep track of forms and data modules that have been loaded as well as maintaining system-specific information such as screen resolution, cursors and available display fonts.
Instances (objects) of the `TForm` class are the building blocks of your application's graphical user interface (GUI).
You can just ask in maXbox the applications handle or the screen resolution by

```
Writeln(Application.Mainform.Name);
Writeln(IntToStr(Application.Handle));
//Get the Screen Resolution!
Writeln(IntToStr(Screen.Width)+' '+IntToStr(Screen.Height));
```

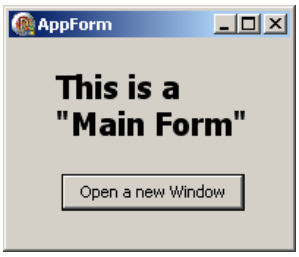
All the windows and dialog boxes in your application are based on the class `TForm`.
A VCL form contains the description of the properties of the form and the components it owns. Each form file or a form code section represents a single form, which usually corresponds to a window or dialog box in an application.
Every Delphi application must have or provide a main form. The main form is the first form created in the body of the application. When the main form closes, the application terminates.


```
73 procedure ButtonCloseClick(Sender: TObject);
74 begin
75   inFrm.Close;
76 end;
```

When a new project or script is created, the first form added to the project automatically becomes the value of the Application's `MainForm` property. Once the application is run, you cannot change the main form of the application.
Our question will be to find out the 4 cases to create a form. All 4 examples discuss a different behaviour in time and resource allocation.

1. Create at design time with a global form var
2. Create at runtime with a global form var

3. Create at design time with a local form var
4. Create at runtime with a local form var



More of this, a form can be modal or not modal (modeless).

☞ `ShowModal` means that the form which opens another form “freezed” until you close the form which has created. So forms can be modal or modeless. Modal forms are forms with which the user must interact before switching to another form (for example, a dialog box requiring user input). In most cases you need this with a user dialog to get a response. The simplest modal dialog to set is the method `Showmessage()`.

☞ `Show` as modeless forms are windows that are displayed until they are either obscured by another window or until they are closed or minimized by the user. Use the method `Show` when you don't want to let your form freeze. `Show` means modeless! You must guarantee that reference variables for modeless forms (such as windows) exist for as long as the form is in use. This means that these variables should have global scope like in our example:

```
14 var
15   myMemo: TMemo;
16   inFrm: TForm;
```

So let’s get the code and I hope you did already work with the Starters 1 to 10 available at:

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

1.2 Build the Form

The first form you create and save in a project becomes, by default, the project's main form, which is the first form created at runtime. As you add forms to your projects, you might decide to designate a different form as your application's main form. Also, specifying a form as the main form is an easy way to test it at runtime, because unless you change the form creation order, the main form is the first form displayed in the running application.
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 with results or messages.

Before this starter code will work you will need to download maXbox from the website. It can be down-loaded from http://sourceforge.net/projects/maxbox site. Once the download has finished, unzip the file, making sure that you preserve the folder structure as it is. If you double-click `maxbox3.exe` the box opens a default program. Make sure the version is at least 3.5 (Now 3.7) because the number cruncher will need it;). Test it with F9 or press **Compile** and you should hear a sound first and a browser will open. So far so good now we’ll open the example:

`245_formapp2.txt`

If you can’t find the file use the link:

http://www.softwareschule.ch/examples/245_formapp2.txt

Or get it as a pdf – document for copy-paste:

http://www.softwareschule.ch/examples/245_formapp2.pdf

Or you use the `Save Page as…` function of your browser[1] and load it from `examples` (or wherever you stored it). One important thing: A form can change the look and feel so it depends on the version of your operation system and the available API (Application Programming Interface). Now let's take a look at the code of this project first with the constant set. Our first line is

```
08 Program Outline_Form_App_Tutorial;
```

We have to name the program called `Outline_Form_App_Tutorial.` Now we jump to line 08:

```
08 Const LEFTBASE = 20;
09       TOPBASE = 30;
10       TEXTPATH = 'examples\outline3.txt';
11       AVIPATH = 'examples\cool.avi';
12       BITMAP = 'examples\citymax.bmp';
```

1.3 First Form

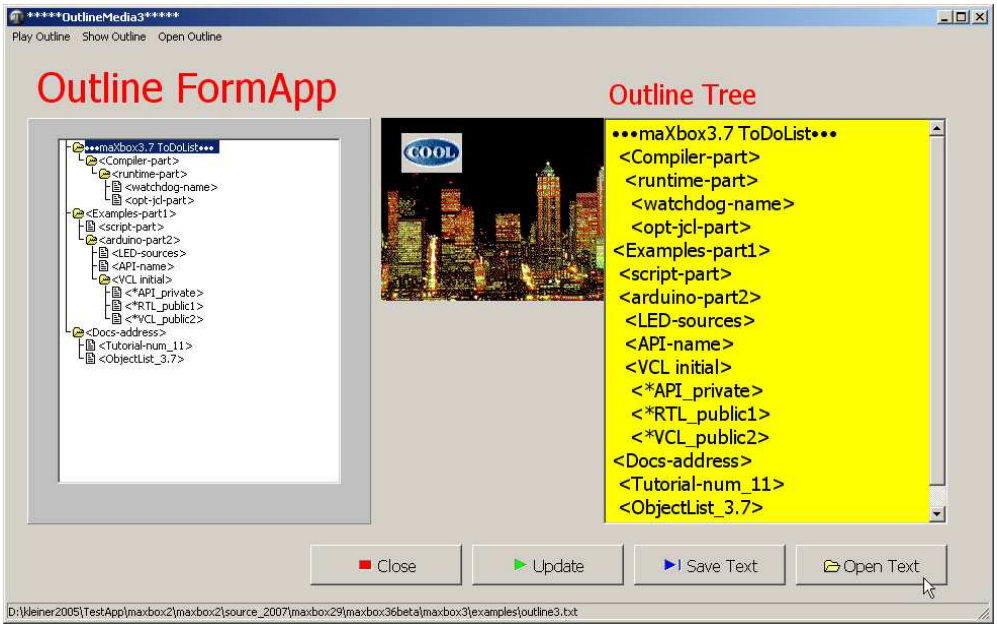
Typically, applications use the global instances of forms. However, if you need a new instance of a modal form, and you use that form in a limited, discrete section of the application, such as a single function, a local instance is usually the safest and most efficient way of working with the form.

```
var
  myMemo: TMemo;
  inFrm: TForm; //Global at designtime
```

The first call to Create creates `inFrm`, an instance of the `TForm` class defined in unit forms with a constructor that takes one argument self as `Owner`, which is the owner of the form being created.

```
87 Begin
88   inFrm:= TForm.Create(self);
```

A complete, executable Delphi application consists of multiple unit modules, all tied together by a single source code module called a project file. In traditional Pascal programming, all source code, including a main program, is stored in .pas files. And in a maXbox script .txt all the units you need are registered and **precompiled** in the engine so you don't need external units in your script file.



[1] Or copy & paste

Our first form looks like a normal form but it has a lot of controls from different components or packages on it. Find out how many controls or objects you get and where the code declaration is.
A **package** or unit is a namespace for organizing classes and interfaces in a logical manner.

1.4 Owner and Event Handler

Typically, you create forms for your application from within the IDE with a form designer. In maXbox we can code directly and native your form with controls. When created this way, the forms have a runtime constructor that takes one argument, `Owner`, which is the owner of the form being created. (The owner is the calling application object or form object.) Owner can be NIL.
To pass additional arguments to a form, create a separate constructor and instantiate the form using this new constructor. The owner of a component is determined by the parameter passed to the constructor when the component is created. But there are three or four possibilities passing one: `Application`, `Self`, `NIL` or a `Owner` by Yourself

By default, a form owns all components that are on it. In turn, the form is owned by the application. Thus when the application shuts down and its memory is freed, the memory for all forms (and all their owned components) is also freed.

Hint: Owner is a property of `TComponent` so you don't have to declare it when passing to the Constructor.
In our case when dynamically creating one or more components as part of a form, the owner has to be the form itself, so you refer to the form via the self pointer in line 88 and a pointer by yourself (`inFrm`).

```
87 Begin
88  inFrm:= TForm.Create(self);
89  stat:= TStatusbar.Create(inFrm);
90  mPanel:= TPanel.Create(inFrm);
```

Next we step to the concept of event handlers. Once you determine when the event occurs (later in the main program), you must define how you want the event handled.

```
094 with inFrm do begin
095   caption:='*****OutlineMedia3*****';
096   height:= 610;
097   width:= 980;
098   Position:= poScreenCenter;
099   onClose:= @FormCloseClick;  //uncomment for debug
100  Show;
101 end;
```

`TForm` declares and defines a method `FormCloseClick` which will be invoked at runtime whenever the user presses close button **x** on the window or <Alt><F4>. This procedure is called an event handler (line 99) because it responds to events that occur while the program is running. The `onClose` is called the event itself.

```
65 procedure FormCloseClick(Sender: TObject; var Action: TCloseAction);
66 begin
67  //myImage.Free;
68  Writeln('Outline Form Closed at: '+ TimeToStr(Time));
69  //inFrm.Free;
70  Action:= caFree;
71 end;
```

Free (Destroy) deactivates the form object by setting Enabled to False before freeing the resources required by the form. The event handler FormCloseClick in the example deletes the form after it is closed, so the form would need to be recreated if you needed to use a form elsewhere in the application. If the form were displayed using Show you could not close the form within the event handler because Show returns while the form is still open. So it's better to work with another event handler ButtonCloseClick which calls with a method inFrm.Close also the centralized FormCloseClick:

```
73 procedure ButtonCloseClick(Sender: TObject);
74 begin
75   inFrm.Close;
76 end;
```

At its simplest, you control the layout of your user interface by where you place controls in your forms. The placement choices you make are reflected in the control's Top, Left, Width, and Height properties. You can change these values also with SetBounds in line 105 at runtime to change the position and size of the controls in your forms.
Controls have a number of other properties, however, that allow them to automatically adjust to their contents or containers. This allows you to lay out your forms so that the pieces fit together into a unified whole.

```
102 with mPanel do begin
103    caption:= '***Outline***';
104    Parent:= inFrm;
105    SetBounds(LEFTBASE,TOPBASE+40,340,400)
```

☞ This form example requires 6 local variables in the method Procedure SetForm from 4 different classes TMainMenu, TMenuItem, TPanel and TBitmap.

1.5 The Parent

Parent in line 104, 111 and so on refers to the component that another component is contained in, such as TForm, TMemo or a TPanel. If one control (parent) contains others, the contained controls are child controls of the parent. Parent is the property of the TControl class.

```
  with mPanel do begin
    caption:= '***Outline***';
    Parent:= inFrm;
    SetBounds(LEFTBASE,TOPBASE+40,340,400);
    color:= clsilver;
    font.color:= clyellow;
    font.size:= 30;
  end
  with mOutln do begin
    //AddChild
    Parent:= inFrm;
    LoadFromFile(selectFile);
    FullExpand;
    width:= 280; height:= 340;
    top:= TOPBASE+60; left:= LEFTBASE+30;
    onclick:= @showEntry;
  end;
```

You can test this by changing the parent of e.g. the outline. All components that share the same Parent are available as part of the Controls property of that Parent. For example, Controls may be used to iterate over all the children of the windowed control.

```
for i:= 0 to Panel1.ControlCount - 1 do
        Panel1.Controls[i].Visible:= false;
```

1.6 Some Animation“

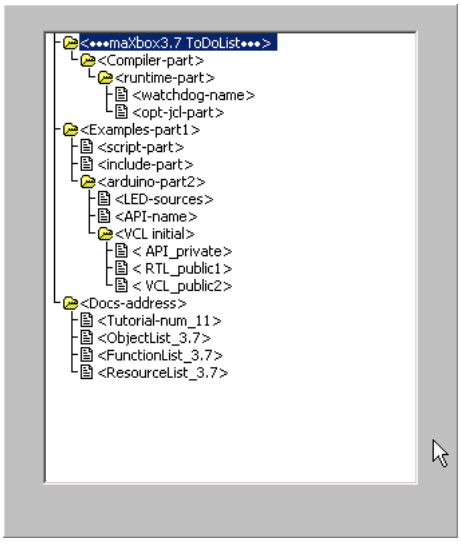
`TAnimate` in line 169 is a rather nice component. However if you don't want to use the built in AVI files and want to create your own AVI files from BMP files, this is also possible. O.k., we come with some goodies to the end of this tutorial.
We Load the animation from the AVI file in the startup directory.

```
11 AVIPATH = 'examples\cool.avi';
```

An alternative to this would be to create a .RES file including the cool.avi as an AVI resource and use the `ResName` or `ResId` properties of Animate. I was particularly interested in this in which I found a utility that takes a list of BMP files that creates an AVI file which can be used by the `TAnimate` component. In our case we play the AVI from the first frame (see below) indefinitely.

0	1	0	**1**	0	1	**1**	**1**	0	0	1	1	0	**1**	0	0	1	1	0	0

Each time `FillBuffer`() is called, the bitmap's data is copied over into the sample, and the sample is time stamped with a start time (frame number * frame length) and duration (frame length). The frame rate is set to 10 frames per second in the filter.



☞ The result of the speed depends on your GPU (graphical processing unit).

```
  with TAnimate.Create(inFrm) do begin
    Parent:= inFrm;
    //Transparent:= True;
    SetBounds(LEFTBASE+400,TOPBASE+55,0,0) //315,-5
    FileName:= ExePath + AVIPATH;
    Active:= True;
    //Hide;
  end;
```

You may not always want all your application's forms in memory at once. To reduce the amount of memory required at load time, you may want to create some forms or components only when you

need to use them. For example, a bitmap needs to be in memory only during the time we pass the bitmap to the canvas of the form!

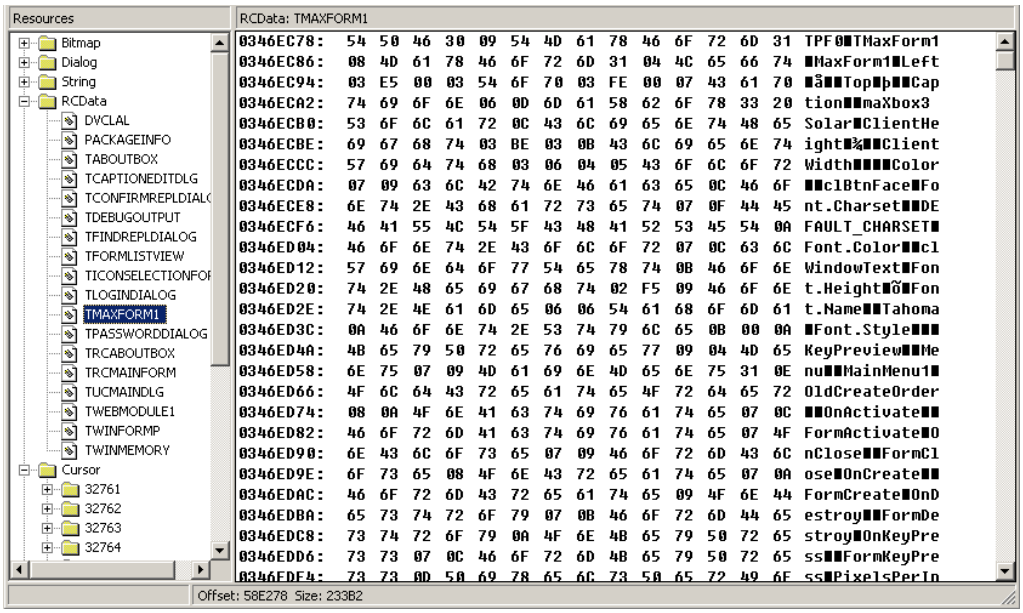
And that's how we create a bitmap with a local var at runtime in line 177:

```
177 bmp:= TBitmap.Create;
178  try
179    bmp.LoadFromFile(Exepath+BITMAP);
180    inFrm.Canvas.Draw(370,70,Bmp);
181  finally
182    bmp.Free;
183 end;
```

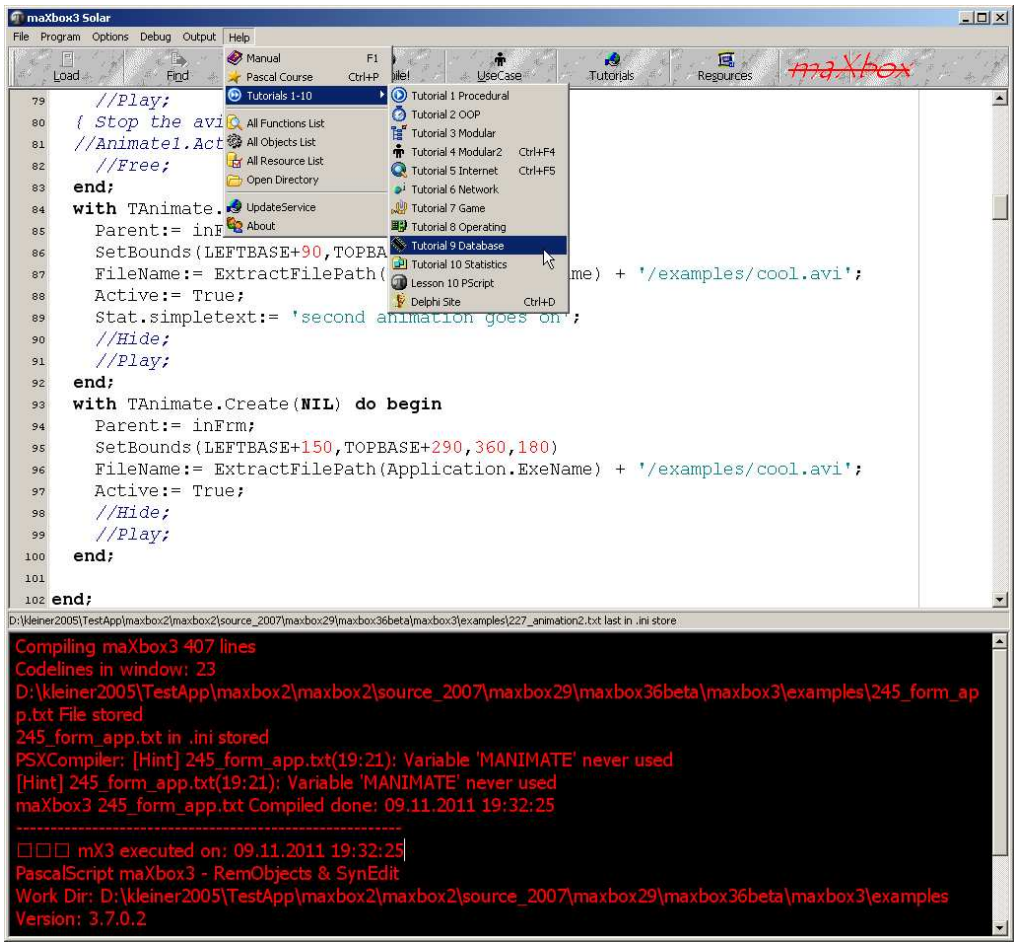
The `try..finally` block ensures the object created is freed when it's no longer needed. If you don't do this, you will get memory problems with large bitmaps, pictures or images, and windows will probably crash in the long run or maybe in short time.

A safer way to create a unique instance of a modal form is to use a local variable in the event handler as a reference to a new instance. If a local variable is used, it does not matter whether is auto-created or not.
Of course, you cannot use local variables in event handlers for modeless forms because they must have global scope to ensure that the forms exist for as long as the form is in use.
If you're interested in creating dialogs at runtime, here's the script:
`224_dialogs.txt`

At last I want to show you how a form or a bitmap is stored as a resource in the exe file. You can really see a bitmap pattern in the output too;):



You can even store sound waves or pictures and play it without the need to create a file, play it just from memory. It is possible to embed any kind of file (also executables) in an executable using the Resource Compiler or a Hexer: http://www.delphi3000.com/articles/article_4337.asp



1: The Animation Component in Code

Task: Try to change the Animation or Bitmap on the Form.

Change the Outliner component with a ListBox component.

1.7 Conclusion

A global variable of type `TScreen` called Screen is created when you create a project. Screen encapsulates the state of the screen on which your application is running.
The global variable Application, of type `TApplication`, is in every VCL- or CLX-based application. Application encapsulates your application as well as providing many functions that occur in the background of the program. For instance, Application handles how you call a Help file from the menu of your program.
Every component we create, at design- or runtime, must be owned by another component like `TForm`. Parent determines how the component is displayed. For example, the Left and Top properties are all relative to the Parent.

max@kleiner.com

Links of maXbox and an Article of Application Owner:

http://www.softwareschule.ch/maxbox.htm
http://sourceforge.net/projects/maxbox
http://sourceforge.net/apps/mediawiki/maxbox/
http://www.delphi3000.com/articles/article_3281.asp