How to remember a window's size, state and position

Why do it?

It's often useful to remember the size and state of your program's windows between executions. This article discusses how.

How it's done

The method we're going to use is to save the position in the registry. I won't discuss details of how to access the registry from code in depth here. First of all, decide where in the registry you're going to keep the information. It's customary for applications to place information that varies between users in:

```
HKEY_CURRENT_USER\Software\MyCompany\MyProgram\X.X
```

(where `X.X` is the version number of the program). We'll use such a key in this article.

You can save the window's current position and size when the program is closing down – the *OnDestroy* form event handler is a good place for this. The program then restores it's position from the registry (if it's been written yet) when opening – we use the form's *OnCreate* handler for that code.

There are complications when saving and restoring the window state. When the window is minimised, Delphi doesn't minimise the form – it hides it and displays the Application window in the taskbar. The method I've used causes a previously minimised window to flash on-screen briefly. I'd welcome ideas on any alternative approaches.

Another complication is that when a window is maximised Delphi updates the *Width*, *Height*, *Left* and *Top* properties of the form to the window's maximised size and position. This means that closing a maximised window stores the maximised size in the registry. When the program is run again it appears maximised, but when the user restores it they expect it to go to the previous normal size and position, but if we reloaded the *Left*, *Top*, *Height* and *Width* properties, the form won't shrink when restored. We get round this by using the Windows API to get the non-maximised size.

Here's the code – the comments should explain what's happening.

```
const
  CRegKey = 'Software\Demos\WdwStateDemo\1.0';

// Helper function to read registry values, and
// deal with cases where no values exist
function ReadIntFromReg(Reg: TRegistry;
  Name: string; Def: Integer): Integer;
  {Reads integer with given name from registry
  and returns it. If no such value exists, returns
  Def default value}
begin
 if Reg.ValueExists(Name) then
    Result := Reg.ReadInteger(Name)
  else
    Result := Def;
end;

procedure TForm1.FormDestroy(Sender: TObject);
var
  Reg: TRegistry;          // the registry
  State: Integer;          // state of window
  Pl : TWindowPlacement;   // used for API call
  R: TRect;                // used for window pos
begin
  {Calculate window's normal size and position
  using Windows API call - the form's Width, Height,
  Top and Left properties will give maximised window
```

```
  size if form is maximised, which is not what we
  want here}
  Pl.Length := SizeOf(TWindowPlacement);
  GetWindowPlacement(Self.Handle, @Pl);
  R := Pl.rcNormalPosition;
  Reg := TRegistry.Create;
  try
    // Open required key - create if doesn't exist
    Reg.RootKey := HKEY_CURRENT_USER;
    Reg.OpenKey(CRegKey, True);
    // Write window size and position
    Reg.WriteInteger('Width', R.Right-R.Left);
    Reg.WriteInteger('Height', R.Bottom-R.Top);
    Reg.WriteInteger('Left', R.Left);
    Reg.WriteInteger('Top', R.Top);
    // Write out state of window
    {Record window state (maximised, minimised or
    normal) - special case when minimised since form
    window is simply hidden when minimised, and
    application window is actually the one
    minimised - so we check to see if application
    window *is* minimised and act accordingly}
    if IsIconic(Application.Handle) then
      {minimised - write that state}
      State := Ord(wsMinimized)
    else
      {not minimised - we can rely on window state
      of form}
      State := Ord(Self.WindowState);
    Reg.WriteInteger('State', State);
  finally
    Reg.Free;
  end;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  Reg: TRegistry;   // the registry
  State: Integer;   // state of window
begin
  Reg := TRegistry.Create;
  try
    // Open required key - exit if it doesn't exist
    Reg.RootKey := HKEY_CURRENT_USER;
    if not Reg.OpenKey(CRegKey, False) then Exit;
    // Read the window size and position
    // - designed form sizes are defaults
    Self.Width := ReadIntFromReg(Reg, 'Width', Self.Width);
    Self.Height := ReadIntFromReg(Reg, 'Height', Self.Height);
    Self.Left := ReadIntFromReg(Reg, 'Left', Self.Left);
    Self.Top := ReadIntFromReg(Reg, 'Top', Self.Top);
    // Now get window state and restore
    State := ReadIntFromReg(Reg, 'State', Ord(wsNormal));
    {check if window was minimised - we have special
    processing for minimised state since Delphi
    doesn't minimise windows - it uses application
    window instead}
    if State = Ord(wsMinimized) then
    begin
      {we need to set visible true else form won't
      restore properly - but this causes a brief
      display of form.
      Any ideas on how to stop this?}
      Self.Visible := True;
      Application.Minimize;
    end
    else
      Self.WindowState := TWindowState(State);
  finally
    Reg.Free;
  end;
end;
```

Listing 1

Worked Example

You can download a *worked example* that includes a Delphi project that demonstrates what we've described here.

Component

My *TPJRegWdwState component* encapsulates all this functionality on behalf of the form that owns it. A sister component, *TPJWdwState*, is included that works with ini files rather than the registry.
