Dave's Development Blog
Software Development using Borland / Codegear / Embarcadero RAD Studio

Another itch the OTA couldn't scratch…

By David | March 23, 2017 0 Commen

Overview

So in my last post I said there were a few things to fixed and / or enhance. In this post I'll explain how I fixed two issues.

The two issues I've fixed are stoppping the timer which closes the Message View if a re-compilation occurs (very simple and I should have done this in the first place) and only closing the Message View if the RAD Studio Desktop remains the same as when the timer was started.

The Fixes

Stopping the Timer on Recompilation

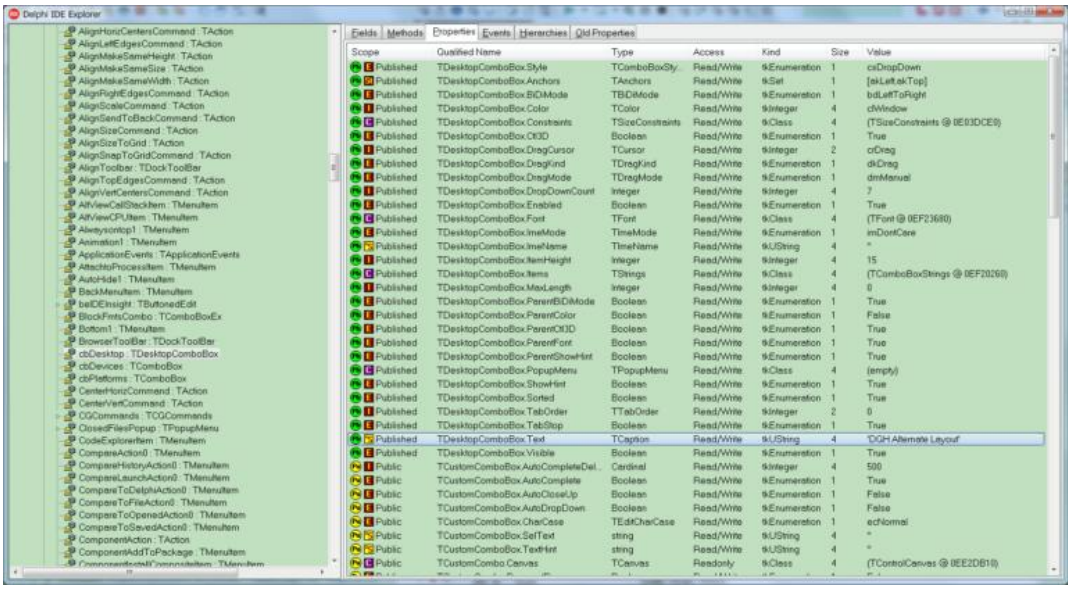
This fix couldn't be simpler and I really should have done this in the first place. To stop the timer on a recompilation I've added a call to disable the timer in the Compile Notifier's `ProjectGroupCompilationStarted` method as below.

```
Procedure TMVHCompileNotifier.ProjectGroupCompileStarted(Mode: TOTACompileMode);

Begin
  FTimer.Enabled := False;
End;
```

Finding the Name of the Active Desktop

The next bug was a little more involved. What was happening before I fixed this is that if you debug an application your RAD Studio Desktop changes (usually) and the Message View in the debugging Desktop was being hidden NOT the one in your main Desktop. The behaviour I wanted is that the main RAD Studio Desktop is hidden but to do that I needed to track which Desktop was active. A search of the Open Tools API files didn't provide any interfaces I could use so I was back to manipulating the IDE directly as before. After a quick look around the IDE I found what I was lookng for.



In the above image (from my Delphi IDE Explorer) I found a custom combo box in the main application form called `cbDesktop` which as

shown above contained my main RAD Studio Desktop name in the `Text` property. So my first thoughts on solving this problem were to access the combo box directly. Now you will notice above that the combo box is a `TDesktopComboBox`, a component we don't have access to therefore I cannot cast or assign the reference in order to access the `Text` property. So I thought I would access the selected item using the `TCustomComboBox` reference and the components `Items` and `ItemIndex` properties.

Before I can do that I needed another function in my application to find a component on a form by name. Below is a simple function that takes a form and a name and returns the component reference if the component with the given name is found on the given form.

```
Function  FindComponent(Form : TForm; strComponentName : String) : TComponent;

Var
  iComponent: Integer;

Begin
  Result := Nil;
  For iComponent := 0 To Form.ComponentCount - 1 Do
    If CompareText(Form.Components[iComponent].Name, strComponentName) = 0 Then
      Begin
        Result := Form.Components[iComponent];
        Break;
      End;
End;
```

Using the above function I tried the following code to get the current desktop name.

```
Function  CurrentDesktopName : String;

Var
  F: TForm;
  C: TComponent;
  CB: TCustomComboBox;

Begin
  Result := '(not found)';
  F := FindForm('AppBuilder');
  If Assigned(F) Then
    Begin
      C := FindComponent(F, 'cbDesktop');
      If Assigned(C) And (C Is TCustomComboBox) Then
        Begin
          CB := C As TCustomComboBox;
          Result := CB.Items[CB.ItemIndex];
        End;
    End;
End;
```

You would think that all is well? Well no. I found during testing (outputting the desktop name to the CodeSite Live Viewer) that in some instances the `ItemIndex` of the combo box was `-1` yet on checking the properties of the desktop dropdown with my Delphi IDE Explorer the `Text` property was correct. So I needed another solution.

Since I was targetting RAD Studio 2010 and above and my Delphi IDE Explorer had already found the information I needed I though that I would use the new RTTI capabilities in RAD Studio 2010 and above. If you've not used RTTI before then I would suggested you create a test project and have a little play as its very powerful and really quite easy to use.

So I redefined my function as below (I'll walk you thought it below):

```
Function  CurrentDesktopName : String;

Var
  F: TForm;
  C: TComponent;
  CB: TCustomComboBox;
  Ctx: TRttiContext;
  T: TRttiType;
```

```
  P: TRttiProperty;
  V: TValue;

Begin
  Result := '(not found)';
  F := FindForm('AppBuilder');
  If Assigned(F) Then
    Begin
      C := FindComponent(F, 'cbDesktop');
      If Assigned(C) And (C Is TCustomComboBox) Then
        Begin
          CB := C As TCustomComboBox;
          Ctx := TRTTIContext.Create;       // Create context
          Try
            T := Ctx.GetType(CB.ClassType); // Get a type
            P := T.GetProperty('Text');     // Get the Text Property
            If Assigned(P) Then
              Begin
                V := P.GetValue(CB);        // Get the value of the Text property
                Result := V.AsString;       // Return the text value a a string
              End;
          Finally
            Ctx.Free;                       // Free context
          End;
        End;
    End;
End;
```

The first thing to do if you want to use RTTI is you must create a `TRTTIContext` in which you can access the RTTI attributes of an object. Next we need to get the type of the object we are referring to with a call to the context's `GetType` method which takes the objects class type. Once we have this we can access the object's fields, methods and properties. In this case we just need a specific property so I call the type's `GetProperty` method with the name of the property I want. Now that I have the property reference I can call the property's `GetValue` method passing the instance of the object to get the value of the property in a `TValue` record. All we need to do now is call the value's `AsString` method to get the property's string information and return it from the function. You need to make sure that you free the context you've created else you will have a memory leak.

So with the above function in place my compile notifier method `ProjectGroupCompileFinished` gets an additional line of code to store the name of the RAD Studio Desktop in a field at the time of compilation as follows:

```
Procedure TMVHCompileNotifier.ProjectGroupCompileFinished(Result: TOTACompileResult);

Begin
  If (Result = crOTASucceeded) And (mvhoEnabled In TMVHOptions.MVHOptions.Options) Then
    Begin
      FTimer.Interval := TMVHOptions.MVHOptions.HideMessageViewDelay;
      FDesktopName := CurrentDesktopName;
      FTimer.Enabled := True;
    End;
End;
```

Also the timer event changes to ensure the Message View is only closed when the desktop is the same as when the compilation started.

```
Procedure TMVHCompileNotifier.CloseMessageView(Sender: TObject);

Begin
  If CompareText(FDesktopName, CurrentDesktopName) = 0 Then
    Begin
      FTimer.Enabled := False;
      HideMessageView;
    End;
End;
```

I hope the above gives people further ideas on how to achieve interesting stuff in the RAD Studio IDE but just remember that you're

mucking about with the actual IDE and the names and locations of element of the IDE might change in different releases so check all the versions of RAD Studio you are going to support.

Downloads

The compiled BPLs and source code for this plug-in can be found on the web page Message View Helper which contains links to a downloadable ZIP file with the BPLs and source code or a GitHub link to the source code.

Related posts:

1. I had an itch the OTA couldn't scratch… (16)
2. Chapter 5: Useful Open Tools Utility Functions (11.9)
3. Chapter 8: Editor Notifiers (11.9)
4. Chapter 6: Open Tools API Custom messages (11.8)
5. Notify me of everything… – Part 1 (9)

Category: Open Tools API RAD Studio Tags: Borland, CodeGear, Delphi, Embarcadero, IOTACompileNotifier, OTA, RAD Studio