

Talking to ActiveForms

by Bob Swart

This is not the first time I've written about ActiveForms in the UK-BUG Developer's Magazine (and it probably won't be the last time either). I can't help it: I find ActiveForms a fascinating technology from Borland. You can design an entire form, and yet use it as a building-block inside an ActiveX host (or container). In this article, I will show you how we can talk to ActiveForms from the most obvious host: Internet Explorer.

Delphi ActiveForms

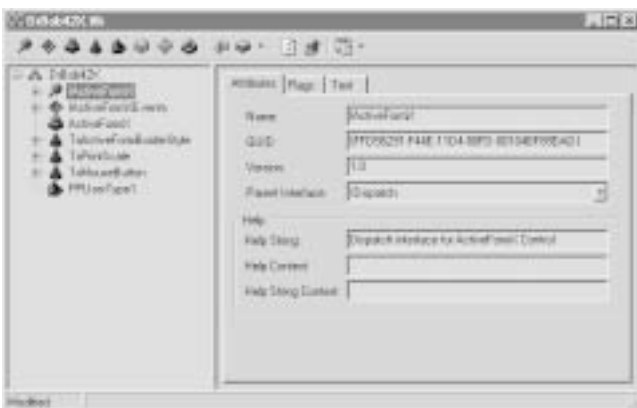
ActiveForms are a feature of Delphi 3 and higher. To start a new ActiveForm, you need to select the corresponding icon in the ActiveX tab of the Object Repository.

In the ActiveForm Wizard, I leave the New ActiveX Name unchanged (this will be an internal name only), but I modify the Project Name to "DrBob42X.dpr", so the resulting ActiveForm will be placed inside DrBob42X.ocx. The most important option to select is the "Include Version Information" checkbox, which will make it easier to deploy new releases of the ActiveForm afterwards (a topic of a previous article in these pages).



Note that although I normally do not even bother with the About Box inside an ActiveForm, we'll select this option today to use as an example in this article. This means that, when we click on the OK button, Delphi will not only generate a new empty ActiveForm, but also an About Box (placed in a normal form, that can be called from the ActiveForm).

View Type Library



Apart from the ActiveForm and the About Box (regular Form, our new project contains a so-called Type Library, containing the interface definitions to communicate to the outside world. You can view and edit these definitions in the Type Library Editor (available in the View menu). As you see from the last screenshot, there are several items available in the treeview, but the most interesting one is the IActiveFormX interface, that defines the interface for our ActiveForm.

If we click on the IActiveFormX to expand it, we see a long list of properties (like HelpFile, Cursor, Enabled) and a method called AboutBox. These are the properties and methods that are pre-defined when you create a new ActiveForm. Each of these properties is defined by a Get and Put (Set) method: for example, for Enabled, we find the following code generated (inside ActiveFormImpl.pas):

```
function TActiveFormX.Get_Enabled: WordBool;
begin
    Result := Enabled;
end;

procedure TActiveFormX.Set_Enabled(Value:
WordBool);
begin
    Enabled := Value;
end;
```

Where Enabled is the actual property of the ActiveForm itself (and the Get_Enabled and Set_Enabled are exported). For the method AboutBox, we find a similar implementation:

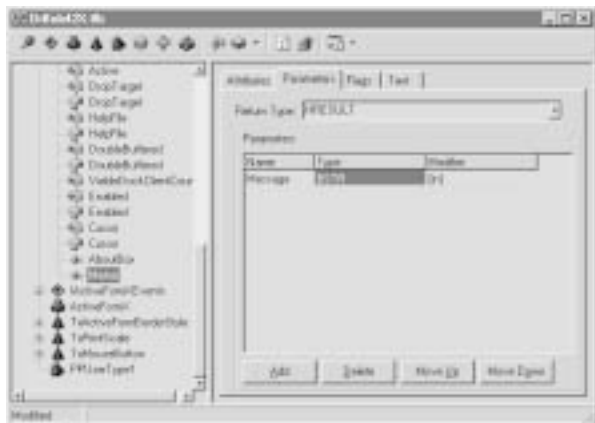
```
procedure TActiveFormX.AboutBox;
begin
    ShowActiveFormXAbout;
end;
```

Where the ShowActiveFormXAbout is defined in unit About1.pas that contains the About Box, and is implemented as follows:

```
procedure ShowActiveFormXAbout;
begin
    with TActiveFormXAbout.Create(nil) do
        try
            ShowModal;
        finally
            Free;
        end;
end;
```

We'll see how to call the exported AboutBox method from outside in a moment. But first, let's use the Type Library Editor to define new properties or methods. For this example, I want to add a method called Memo that takes one Message input argument of type BSTR (equivalent to

WideString in Pascal). The purpose of this method is to be called from the ActiveX host, and take a message that we can show in a component on the ActiveForm itself (the obvious choice is to use a TMemo for that, of course).



After we've added the method Memo in the Type Library, we need to click on the "Refresh Implementation" button (the one with two green arrows), and then we'll see an empty method inside the ActiveFormImpl1.pas unit. This method can be implemented as follows:

```
procedure TForm1.ActiveFormX.Memo(const Message:
 WideString);
begin
    Memo1.Lines.Add(Message)
end;
```

Now it's time to deploy our ActiveForm (using the Web Deployment Options and then the Web Deploy menu item). If everything went OK, you can load the generated HTML file and see the ActiveForm inside Internet Explorer. But no About Box, yet, and the memo will still be empty.

In order to call the exported methods of an ActiveForm in Internet Explorer, we need to add a little VBScript to modify the HTML file as follows:

```
<HTML>
<H1> Delphi 5 ActiveX Test Page </H1><p>
You should see your Delphi 5 forms or controls
embedded in the form below.
<HR><center><P>

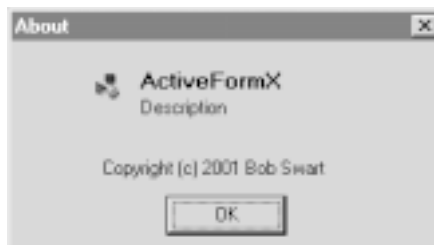
<OBJECT NAME="DrBob42X"
    classid="clsid:FFD56295-F44E-11D4-80FD-
        00104BF89DAD"
    codebase="./
DrBob42X.cab#version=1,0,0,0"
    width=451
    height=253
    align=center
    hspace=0
    vspace=0
>
</OBJECT>

<SCRIPT VBSCRIPT>
    DrBob42X.Memo("This is the HTML page talking
to the ActiveForm");
    DrBob42X.AboutBox();
</SCRIPT>

</HTML>
```


The first and most important change is that we give a NAME to our ActiveForm (in this case DrBob42X), so we can refer to it from the VBScript that follows. Inside the VBScript, we can call the methods, such as DrBob42X.AboutBox(); Note that the () are necessary, otherwise it will look like a property (and you will get an error message from Internet Explorer that this item wasn't found).

The call to Memo takes a string, which is converted to a WideString when it ends up calling the Memo method inside the ActiveForm, showing the text inside the memo field on the ActiveForm. And the AboutBox method calls the generated and built-in AboutBox, which looks as follows:



When using ActiveForm from other environments, such as C++Builder or even Visual Basic (or the upcoming C#), we can call methods and get/set properties in the same way. This allows us to put a lot of independent functionality inside an ActiveForm (turning it into a "black box") and be able to use it as a building block - a bit like lego.

Bob Swart (aka *Dr.Bob* - www.drbob42.com) is an @-Consultant for Everest Delphi OplossingsCentrum and has spoken at the Inprise/Borland Conferences since 1993. Bob is a trainer who has presented all over the world. He's also a free-lance technical author and has written chapters for *The Revolutionary Guide to Delphi 2*, *Delphi 4 Unleashed*, *C++Builder 4 Unleashed* and *C++Builder 5 Developer's Guide*.



Angus' Nifty Tips

Another way to add a background image to a form

Add the following to your public declarations:

```
bmpBackground : TBitmap;
```

To the form's OnCreate event handler add:

```
bmpBackground := TBitmap.Create;
bmpBackground.LoadFromFile(
    'c:\windows\setup.bmp' );
```

To the form's OnPaint event handler add:

```
Canvas.Draw( 0, 0, bmpBackground );
```

To the form's OnDestroy event handler add:

```
bmpBackground.Free;
```