COM Corner: Top 10 ActiveForm Questions

by Steve Teixeira

ActiveForms provide a quick and nifty means of embedding small-scale Delphi applications into ActiveX control containers such as Internet Explorer or Visual Basic applications. When used in conjunction with a web browser, ActiveForms also provide a convenient means for delivering and versioning these mini applications, since the web browser is able to handle a lot of the dirty work. However, since their inception in Delphi 3, ActiveForms have been one of the most mysterious features in the product. It's no wonder: ActiveForms can be complex, difficult to use, and they are not particularly well documented. Even two years after their introduction, I still see many of the same questions cropping up in the borland.public.delphi.activex.controls.writing newsgroup that I saw in the Delphi 3 days.

In this article, I hope to shed some light into the dark corners of ActiveForm technology in order to help you better understand where and how to properly apply this very useful technology, by presenting the answers to ten of the most common ActiveForm questions.

ActiveForm Versus ActiveX

Q What is an ActiveForm, how is it different to an ActiveX?

A Looking at this question from a strictly COM-oriented perspective, ActiveForms and ActiveX controls created by Delphi are the same thing: they are both inproc COM servers that expose a control coclass and implement the interfaces necessary to be considered an ActiveX control. From a Delphi perspective, Delphi ActiveX controls and ActiveForms embody the same abstract idea: a *COM thing* that wraps a *VCL thing*, thereby enabling a VCL control to function as an ActiveX control in ActiveX containers.

On a more technical level, however, there is a subtle, yet important, architectural difference between ActiveX controls and ActiveForms in the Delphi world. An ActiveX control is a fully written VCL control for which a COM wrapper is generated by the wizard. After the wizard does its, uh, magic, your project essentially comprises the COM wrapper, a descendant of `TActiveXControl`, which you can edit and tweak to meet your needs. In a nutshell, the control source code remains rather static in nature, while the COM wrapper's source may undergo many additions or changes.

An ActiveForm is the converse of this situation. The COM wrapper, the `TActiveFormControl` class, is defined by the VCL and remains static, while the VCL control, a `TActiveForm`, is what you will spend most of your time developing.

Finally, another distinction between Delphi ActiveX controls and ActiveForms is in their use. Because ActiveX controls encapsulate VCL controls, a typical example of their use involves having the ActiveX control as a part of a larger application, perhaps in a Visual Basic application or as a widget that serves some specific function on a web page. Because you can stuff application-like functionality into ActiveForms, they are typically used to provide mini application-like functionality to a web page. As such, there are a host of problems associated with ActiveForms that you won't run across as often with ActiveX controls, such as keyboard handling and mouse focus issues.

Design Considerations

Q I'm considering an ActiveForm to add application-like functionality to a web page. Are there any design considerations I need to be aware of?

A You bet there are! Before anything else, you must determine whether your ActiveForm will be available on the public internet or only privately on an intranet. Because of their relatively large download size and the security and stability problems inherent in web-delivered ActiveXs, ActiveForms were really intended as an intranet solution as opposed to something you would throw on any old public web page. 'Wait a minute,' you're saying to yourself, 'what security and stability problems?' Web-delivered ActiveX presents a security risk because it enables native code to run unchecked on your machine. While code signing works towards reducing this security risk, it does not reduce the stability risk. That is, no matter how well intentioned the author was, there is no protection against bugs that could crash your browser, crash your system, or do even worse damage. And as anyone who has spent their fair share of time on the web knows, crashing someone's machine is not the way to go about increasing traffic to your site. Because of all this, my recommended use of ActiveForms is within a controlled corporate intranet environment, where people are actually responsible for the code that executes on the systems of others.

Another important design consideration for ActiveForms running within the context of a web browser is to remember your place! An ActiveForm is a guest of the HTML page in which it resides,

```
type
  TActiveFormX = class(TActiveForm, IActiveFormX)
    ...
  private
    function GetButtonCaption: string;
    procedure SetButtonCaption(const Value: string);
    Button1: TButton;
  TMyActiveForm
  published
    property ButtonCaption: string read GetButtonCaption write SetButtonCaption;
  end;
function TActiveFormX.GetButtonCaption: string;
begin
  Result := Button1.Caption;
end;
procedure TActiveFormX.SetButtonCaption(const Value: string);
begin
  Button1.Caption := Value;
end;
```

➤ *Above: Listing 1*

➤ *Below: Listing 2*

```
function TActiveFormX.Get_ButtonCaption: WideString;
begin
  Result := ButtonCaption;
end;
procedure TActiveFormX.Set_ButtonCaption(const Value: WideString);
begin
  ButtonCaption := Value;
end;
```

which is in turn a guest in the web browser. If your ActiveForm is too intrusive, or at odds metaphorically with the browser UI, then it is a rude guest. For example, it doesn't make sense to want to put a menu on an ActiveForm, and you should think twice before bringing up other modal or modeless windows from your ActiveForm.

Properties

Q I would like to add properties to my ActiveForm and get them to participate in the streaming of the control. How can I do this?

A A classic question. A typical scenario might be that the ActiveForm has a button and you wish to make the caption of that button editable by users of your ActiveForm. Since an ActiveForm's representation in the type library consists of 'flat' interfaces rather than the nested components you're familiar with in VCL. This means that if you have a form with several buttons, they cannot easily be addressed in the VCL manner of `ActiveForm.Button.ButtonProperty` as an ActiveForm. Instead, the easiest way to accomplish this is to surface the button properties in question as properties of the ActiveForm itself. The DAX framework makes adding properties to ActiveForms a pretty painless process: you just need to follow a couple of steps. Here's what's required to publish the `Caption` property of a button on an ActiveForm.

First, add a new published property to the ActiveForm declaration in the implementation file. This property will be called `ButtonCaption`, and it will have reader and writer methods that modify the Caption of the button.

Then, add a new property of the same name to the ActiveForm's interface in the type library. Delphi will automatically write the skeletons for the reader and writer methods of this property, and you must implement them by reading and writing the ActiveForm's `ButtonCaption` property.

For example, I first add a new property to my ActiveForm along with a reader and writer, as in Listing 1. I then go to the type library editor and add a property called `ButtonCaption` to the coclass for this ActiveForm. I implement the skeletons as shown in Listing 2. That's all there is to it. Now the `ButtonCaption` property will be saved and loaded along with the ActiveForm's stream.

Web Deployment

Q I don't seem to be able to get web deployment to work for me. Can you help me figure how all the pieces fit together?

A Bob Swart provided an excellent overview of web deployment in his article on ActiveForms in the June 1997 issue (#22), so I recommend that article as a resource for this topic. Additionally I will say that 90% of the problems I have seen with web deployment are cause by one of a handful of problems.

First, the browser's web security is set to *high*, so ActiveX controls are not downloaded. The solution to this problem is to downgrade the browser security to *medium*, which means you will be warned before ActiveX content is downloaded.

Second, Netscape Navigator doesn't display the ActiveForm. This is because Navigator doesn't provide built-in support for ActiveX controls. The solution is to download a third party plugin that enables ActiveX support, such as the ScriptActive plugin from NCompassLabs (who are at www.ncompasslabs.com).

Third, the URL that references your control, as specified in the HTML or INF file, is incorrect. Scrutinize the control URL and ensure it points to the correct location.

Finally, your ActiveForm needs additional files that have not been deployed to the remote machine.

➤ *Table 1*

File	Required When...
StdVclxx.dll	Your ActiveForm exposes properties of type IStrings, uses any of the standard Delphi property pages, or serves as a MIDAS client.
ShareMem.dll	Your ActiveForm exports AnsiString variables.
DbClient.dll	Your ActiveForm is a MIDAS client.
BDECab.dll	You wish to deploy the BDE with your ActiveForm.

The IDE automatically detects which packages are used by your project, but how do you know what other files might be additionally required? Table 1 gives a brief list of some files that might be required by your ActiveForm:

Initializing Properties

Q How can I initialize ActiveForm properties in my HTML code?

A You can set the properties of your ActiveForm using the PARAM tag with a NAME/VALUE pair in your HTML code. For example, the HTML in Listing 3 is built from the example in the earlier properties question and demonstrates how to initialize the button caption to hello.

Note that the control must support IPersistPropertyBag in order for this technique to work. Delphi 4's version of TActiveXControl provides support for IPersistPropertyBag, whereas the Delphi 3 version does not.

```
<HTML>
<H1> Delphi 4 ActiveX Test Page </H1><p>
You should see your Delphi 4 forms or controls embedded in the form below.
<HR><center><P>
<OBJECT
  classid="clsid:831B0597-6C40-43A2-8E61-13CB3E2F7D18"
  codebase="file://c:/temp/ActiveFormProj1.ocx"
  width=374
  height=215
  align=center
  hspace=0
  vspace=0
>
<PARAM NAME="ButtonCaption" VALUE="hello">
</OBJECT>
</HTML>
```

➤ *Listing 3*

Debugging

Q How do you debug an ActiveForm?

A Take a look at Bob Swart's article on *Creative Debugging Techniques* in the July 1999 issue (#47), in which he explains ActiveForm debugging techniques in detail .

Keystrokes

Q What is the deal with keystrokes in ActiveForms?

A If you've had to do any keystroke handling in ActiveForms, you may have noticed some flaky behavior. This reason for this is because every container has its own slight variation on how keystrokes work. For example, Internet Explorer does things differently to Visual Basic, which does things differently to Delphi. If you need to do any special keystroke handling, the best advice I can give you is to code for your target container. In general, there are a couple of places to go if you need to do special keystroke handling. One is the VCL control's WndProc method. The second is TActiveXControl's implementation of

```
IOleInPlaceActiveObject.
  TranslateAccelerator
```

which should catch special keystrokes such as tabs, cursor keys and carriage returns.

Database Connections

Q How do I connect to a database within an ActiveForm?

A Believe it or not, there is no trick to displaying and manipulating data in an ActiveForm. It works in a manner identical to normal forms. You need only ensure that the middleware you are using to connect to the database is either a part of your control's deployment or is guaranteed to reside on the client machines. For example, if your application is accessing Paradox tables via the BDE, you should probably deploy BdeCab.dll with your ActiveForm so that the BDE will be installed on the client machines when they attempt to use the ActiveForm. Likewise, if you are accessing an Oracle database through non-BDE components, then you will need to somehow ensure that the proper Oracle middleware is deployed or installed on client machines in order for the ActiveForm to access data.

Working The Browser

Q How do you make the browser do stuff from within an ActiveForm?

A Since ActiveX controls can run within the context of a web browser, it makes sense that web browsers expose functions and interfaces that allow ActiveX controls to manipulate them. Most of these functions and interfaces are located in the UrlMon unit. Among the simplest of these functions are the HlinkXXX functions, which cause the browser to hyperlink to different locations. For example, the HlinkGoForward and HlinkGoBack functions cause the browser to travel forward or back in its location stack. The HlinkNavigateString function causes the browser to travel to a specified URL. These functions are defined in UrlMon as shown in Listing 4.The pUnk parameter for each of these functions is the IUnknown for the ActiveForm. Since the 'COM part' of the ActiveForm can be accessed via its VclComObject property, you can pass IUnknown(VclComObject) in this parameter (just pass Control as IUnknown in the case of an ActiveX control). The szTarget parameter of HlinkNavigateString() represents the URL you want to use.

These methods provide a convenient means for controlling the browser from within the context of your ActiveForm.

➤ *Listing 4*

```
function HlinkGoBack(pUnk: IUnknown): HResult; stdcall;
function HlinkGoForward(pUnk: IUnknown): HResult; stdcall;
function HlinkNavigateString(pUnk: IUnknown; szTarget: PWideChar): HResult;
  stdcall;
```

Data Files

Q How can I get data files from my server to my ActiveForm?

A This is actually a follow-up to the previous question, since the real question is, 'How can I make the browser download files I need from the web server for me?' Going back to the UrlMon unit, it has a method called URLDownloadToFile, which, given a URL, downloads a file to the client machine. This method is defined in UrlMon as:

```
function URLDownloadToFile(
  p1: IUnknown; p2: PChar;
  p3: PChar; p4: DWORD;
  p5: IBindStatusCallback):
  HResult; stdcall;
```

p1 represents the IUnknown for the ActiveX control, similar to the pUnk parameter of the HlinkXXX functions; p2 holds the URL of the file to be downloaded; p3 is the name of the local file that will be filled with the data of the file specified by p2; p4 must be set to 0, and p5 holds an optional IBindStatusCallback interface pointer. This interface can be used to obtain incremental information on the file as it downloads.

For example, let's say my web server has a file called fun.avi located in its web root directory. I could then retrieve this file to c:\temp\fun.avi using the following line of code:

```
URLDownloadToFile(
  IUnknown(VCLComObject),
  PChar(Format(
  'http://MyServer/fun.avi',
  'c:\temp\fun.avi', 0, nil);
```

I could also have optionally provided an IBindStatusCallback pointer to get information on the download as it occurs. The disk accompanying this issue contains a sample application which does just that.

Steve Teixeira is Vice President of Software Development at DeVries Data Systems, located in Silicon Valley. Send those Delphi COM questions to steve@dvdata.com