December, 1997

Advisor Answers

Sharing Code Between Forms

Visual FoxPro

Q: We have an application which has two forms which are different except for one piece of code which is needed by both. To call this function in the first form, a button is pushed, and that control's Click method is executed.

Instead of typing the commands over in the second form, we thought we would just reuse the code from the button of form1, by putting Form1.Command1.Click() in the Click method of the command button in the second form.

When we push the button in form2, VFP calls the code in form1 correctly. However, some of the code refers to objects in the current form. When we are in Form2 and we call that button and the code from Form1 is executed, the routine thinks that we are currently in Form1, when we are clearly in Form2. Thus statements such as

```
if ThisForm.Text4.value = "CQ"
```

don't work correctly because ThisForm refers to Form1 rather than Form2, not the currently active form. Thus, we cannot reuse the code.

Is there a simple way to re-use this code?

–Tom Phelan & Svetlana Roehrs, Lincoln, NE (via Internet)

A: Let me first explain what's going on here, then I'll offer several ways you can write the code only once and use it as needed.

The special object references, THIS and THISFORM (and THISFORMSET), are designed to let you write code that doesn't have to know the name of the object that's executing it. Without these object references, you'd have a hard time writing code that could be reused.

In your situation, as soon as you call the Click method for the button in Form1, THISFORM becomes an object reference to Form1. While this may seem confusing when you're working with forms (because of the additional issue of which form is "active"), if you look at it from the point of view of other kinds of object, especially non-visual objects, it makes more sense.

Suppose you have an application object and a security object. The application object calls a method of the security object (say, LogIn) when the user starts the application. The LogIn method needs to be able to access the properties and methods of the security object using the THIS reference. If THIS referred to the object that called the method, it would be impossible to write generic code.

What makes your situation especially confusing is that you're calling a method of another object, not because you want to use that other object, but in order to share code. This is also the key to finding a good general solution.

There is one approach you can take that would let you keep the basic structure you have, but it's not really a good idea. Rather than using THISFORM in the code, you can use _SCREEN.ActiveForm, which always refers to the currently active form.

However, in the long run, having one form call a method in another as a means of sharing code will get you in trouble. What if you need to use Form2 without Form1? A better solution is to put the code where it should be and where both forms can use it as needed.

Where is that? The answer is "It depends." Is this routine something that's inherently a part of these forms (and conceivably, other forms as well)? Is it something that, in essence, extends the VFP programming language? Or is it something that's really part of another operation entirely? Each of these three ideas calls for a different solution.

If the routine is an inherent part of the forms, the best approach is to add a custom method to a form class and then base both forms on that class. The Click method for each button can call your custom method.

This approach has an added benefit beyond making things work the way you want. By putting the code in a named custom method, you're making it visible rather than hiding it in the user interface. The name should make it clear what the code does and any control that needs to do this thing can call on this named method. In general, anytime you find one method doing nothing but calling another method (especially an event method) as a means of code sharing, consider adding a custom method for the shared code and having it called from each method that needs it.

What kind of code could be viewed as an extension of the programming language? Anything that you think should have been built in, but wasn't. For example, several months ago (August '97) in this column, I showed a routine that converts a number of seconds to the string format "HH:MM:SS". In my view, that function wasn't very different than DTOC() or TTOC(). When you see a routine as part of the base language, you may want to simply create it as a stand-alone function and call it as needed. So you'd simply take the shared code, save it to a .PRG and name it appropriately. Then, your forms can call on that routine as needed.

The final possibility is that the routine is specialized (not a language extension), but is part of some other process. In that case, you should create a class for that process and make your routine a method of that class. If the method needs access to data from the forms that call it, you'll want to pass appropriate parameters. In the example in your question, you should probably pass just the relevant value, like this:

```
oProcessorObject.MyMethod(THISFORM.Text4.Value)
```

The method can accept the value or values it needs like this:

```
PROCEDURE MyMethod
LPARAMETERS cValue
```

If the process needs access to a number of values from the form, you may want to pass a reference to the form itself, like this:

```
oProcessorObject.MyMethod(THIS)
```

In the method, define a parameter to receive the object reference and you can then use it in the code:

```
PROCEDURE MyMethod
LPARAMETERS oCallingObject

* Your code goes here.
* To refer to an item on the form, do something like

IF oCallingObject.SomeProperty ...

ENDPROC
```

However, passing the entire form means that the processing object has to know where to look in the form for the values it needs. (For example, it would have to know that text4.Value is the one to test.) So this approach is more appropriate when you need to be able to process multiple objects with the same structure.

You have to make the processor object available to the forms by instantiating it and keeping a reference around. You could create a processor object as part of the initialization of the form class. Add a custom property to the form class to hold a reference to the processing object, then create it in the form's Init method:

```
THIS.oProcessor = CREATEOBJECT("ProcessorClass")
```

Then, to use the method within the form, you reference the processor method like this:

```
THIS.oProcessor.MyMethod(<parameters>)
```

However, this approach means that each form has its own instance of the processor class. If you only want to create one instance, a simple way to do it is to use CREATEOBJECT() in your main program and save the reference in a variable:

```
oProcessorObject = CREATEOBJECT("ProcessorClass")
```

However, doing things this way means that your forms depend on the existence not only of an instance of the processor class, but an instance with a particular name. One way to avoid the problem is to pass the reference to the processor object as a parameter when you start the form, using:

```
DO FORM Form1 WITH oProcessorObject
```

You need to add a custom property to the form to hold the reference. Then, in the form's Init method, you receive the parameter and save the reference:

```
* Form1.Init
LPARAMETERS oProcessorObject

THIS.oProcessorObject = oProcessorObject
```

Now, the methods in the form can call on the processing method without knowing the name of the processing object:

```
THIS.oProcessorObject.MyMethod(<parameters>)
```

In general, the ability to pass object references from one routine to another makes it much easier to write generic code and to allow objects to communicate with each other.

One of the three situations I've outlined here should apply in your case. Now that you've seen the possibilities, all three will probably come in handy in your development.

–Tamar