January, 2005

Advisor Answers

Restoring original data values

VFP 9/8/7/6

Q: I'm using VFP 6 to build an application that so far spans more than 50 forms. At a meeting with users, they requested a "Reset" ability; they want to click a button on a form and reset all fields to their original values. This would be easy with database containers and TABLEREVERT() but we're not using database containers.

I'm looking for a less painful and tedious way of implementing this than doing something manually with each appropriate control, whether it is a textbox, edit control or option group.

–Henry Hayden (via Advisor.COM)

A: As you point out, the easiest way to do this is using VFP's buffering abilities. You don't actually have to use a DBC to take advantage of buffering (though, prior to VFP 9, a DBC is required for transactions). Free tables can use both row buffering and table buffering. So one solution to your problem is to apply table buffering to your tables and use the TableRevert() function when a user clicks the Reset button.

However, if your application isn't designed to take advantage of buffering, retro-fitting it may be quite painful. It's possible you'd have to modify every form to turn buffering on, and change the way you save form data. However, if all the forms in your application are based on a form class that centralizes data management, the changes actually could be quite simple, though of course, you'd have to do extensive testing to ensure a major change like this doesn't break anything.

I'll assume that switching to buffering is too difficult in your case and offer you an alternative. What you need is a way to store the initial value for each control on a form and a way to restore that value when the user clicks Reset. This is a case where object-orientation makes what appears to be a difficult task pretty straightforward. The basic technique is useful for a wide variety of tasks, not just providing Reset capability.

The secret is to ask each control to store its own original value. The place to do this is in your base classes for this project. Add two custom properties, lSaveValue and uOriginalValue, and two custom methods, SaveValue and RestoreValue, to each class that handles data. The lSaveValue property lets you determine whether any instance of the control saves its value; set it to .T. in the Property Sheet, so that saving the value is the default. The uOriginalValue property stores the original value.

For regular controls (like spinners, textboxes, etc.), the code for SaveValue looks like this:

```
IF This.lSaveValue
   THIS.uOriginalValue = This.Value
ENDIF
```

For those controls, the code for RestoreValue reverses the process:

```
IF This.lSaveValue
   IF PEMSTATUS(THIS, "uOriginalValue", 5)
      This.Value = This.uOriginalValue
   ENDIF
ENDIF
```

Container controls, like pageframes and grids, require special handling. You have to drill down into them to store the values of their contents. Add two methods, SaveValue and RestoreValue. No custom properties are needed for the container, but for the functionality to work, the controls in the container must have the SaveValue and RestoreValue methods. For a pageframe, the SaveValue method looks like this:

```
LOCAL nPage, oControl

* Drill down to each page and to each control on the page
FOR nPage = 1 TO This.PageCount
   FOR EACH oControl IN This.Pages[nPage].Objects
      IF PEMSTATUS(oControl, "SaveValue", 5)
         oControl.SaveValue()
      ENDIF
   ENDFOR
ENDFOR
```

The RestoreValue method for a pageframe is almost identical:

```
LOCAL nPage, oControl

* Drill down to each page and to each control on the page
FOR nPage = 1 TO This.PageCount
   FOR EACH oControl IN This.Pages[nPage].Objects
```

```
      IF PEMSTATUS(oControl, "RestoreValue", 5)
         oControl.RestoreValue()
      ENDIF
   ENDFOR
ENDFOR
```

At the form level (that is, in your base form class), add methods SaveAll and RestoreAll. The code for SaveAll calls each control's SaveValue method, like this:

```
FOR EACH oControl IN This.Controls
   IF PEMSTATUS(oControl, "SaveValue", 5)
      oControl.SaveValue()
   ENDIF
ENDFOR
```

The code for RestoreAll is analogous—check each control to make sure it has the method, then call the method:

```
FOR EACH oControl IN This.Controls
   IF PEMSTATUS(oControl, "RestoreValue", 5)
      oControl.RestoreValue()
   ENDIF
ENDFOR
```

The calls to PEMSTATUS() in the SaveAll and RestoreAll methods mean that any controls on the form that don't offer save and restore capability are simply skipped and don't crash the form.

Finally, you need to call SaveAll and RestoreAll in the right places. What are the right places? For RestoreAll, that's easy. Call it from the Click method of the Reset button. SaveAll needs to be called anytime you retrieve new data into the form. A likely place to make the call is in the Refresh method; if you put the call there, issue DODEFAULT() before calling SaveAll to ensure that the values in the controls are updated before saving them. You may also need to call SaveAll on the way into the form.

This month's Professional Resource CD includes BaseReset.VCX, a class library containing subclasses of CheckBox, ComboBox, EditBox, ListBox, OptionGroup, PageFrame, Spinner and TextBox that incorporate the SaveValue and RestoreValue functionality, and a subclass of Form that includes the SaveAll and RestoreAll methods. There's also an example form, ResetDemo.SCX, that demonstrates some of the controls. Be aware that the form edits data in the Northwind Employees table directly, so changes you make and don't reset will be saved.

The solution shown here requires you to use subclasses of the VFP base classes and may force you to change the control classes you're using in some forms. Another solution is to build the save and restore capability into a separate class and use BindEvents() to connect that ability to existing forms. This is especially useful if your forms use the VFP base classes (which is a bad idea, but you may be stuck with it). The BaseReset class library on this month's PRD also includes cusResetter and cusControlReset, a pair of classes that set up save and restore functionality to make it available for binding, plus ResetDemo2.SCX, a form that uses those classes, and RunForm, a program that runs the form and performs the binding.

While the code here replaces a built-in ability, the strategy of asking a control to store information about itself is one that has wide applicability. Anytime you can store data in a control itself, you avoid the need to set up an outside storage mechanism.

–Tamar