January, 1997

Advisor Answers

Visual FoxPro

Q: My VFP ComboBox gets its items from a field in a table. How can I tell it to display a particular record as the default? In my test case form, the initial value is always the first record in the table. In any case, I really want the initial value to be a particular record from the table. I did a LOCATE and Refresh, but it doesn't make any difference.

–Garth Groft (via the Internet)

A: The key to solving your problem is to understand that, even when a combo box or list box is based on a table, it maintains its own list of items internally. Movement in the table does *not* move the pointer in the combo or list. To change the current record in the control, you need to set the Value property of the control (or, for a combo, the DisplayValue property - either one does the trick).

To keep the combo or list in synch with the table, you need to set Value every time you move the record pointer. Since there are several times you might need to do this (when the form starts or when you click any button that moves to a new record), it seems that a custom method to re-synch the combo is your best bet.

In order to add a method to a control, we have to subclass the control. In the Form Designer, we can add methods only to the form (or formset, if we have one). For a quick-and-dirty solution, it's okay to add a control-specific method like this to the form, but for production code, it's a lot better to subclass the control for the functionality you want. It makes the code clearer and, since you're likely to want to use the same technique for many combos, gives you a reusable tool.

I already have subclasses of all the controls that contain no changes at all from VFP's base classes - this gives me a starting place to make changes. We'll subclass the combo class in that class library.

Let's call the new method Resynch; the simplest version of the code looks like:

```
THIS.Value = <your table>.<the field in BoundColumn>
```

For example, Figure 1 shows a combo based on the TasTrade Employee table. The combo contains the last name and first name of each employee, with the first column containing Last_Name as the bound column. So, the code in Resynch would be:

```
THIS.Value = Employee.Last_Name
```

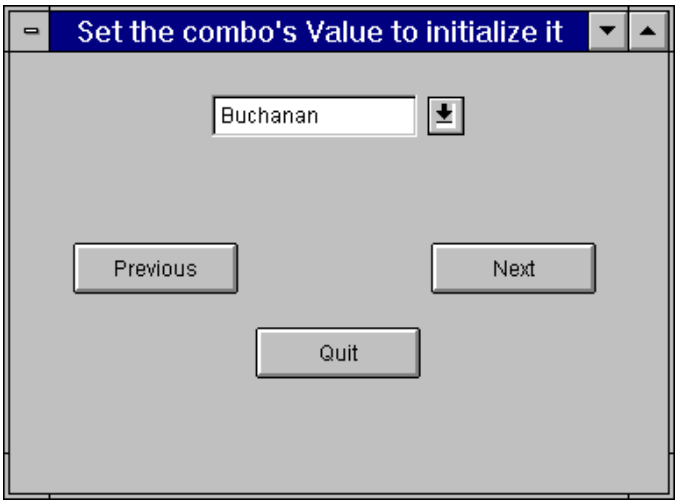

Figure 1: Initializing a combo box - The combo has a custom method called Resynch that sets its Value after the record pointer has been moved.

In either the combo's Init or the form's Init, we call Resynch to set the initial value (assuming something has moved the record pointer to the desired record). We call Resynch again in the Click methods of any buttons that move the record pointer.

The advantage of using a custom method, even one as simple as this, is that changes are centralized. If we change our minds and bind a different field in the combo, we only have to change code in one place.

However, I still find this code much too specific. I'd like to write this code once and forget about it. The code above has to know the name of the table and field the combo is based on. A better solution is to add custom properties to the combo that tell it which alias and field to use. We could derive this information from existing properties (RowSource and BoundColumn), but it would be an ugly job of parsing and the parsing technique would need to be aware of RowSourceType, so we'll simply add two properties:

cAlias - the alias of the table, cursor or view to which the combo is bound.

cField - the name of the field of the table, cursor or view corresponding to bound column.

These properties can be set in the property sheet as part of the design stage.

Now the code in Resynch can be generic:

```
THIS.Value = EVAL(THIS.cAlias+"."+THIS.cField)
```

It's a little harder to read, but now we don't have to touch it every time we change the RowSource or BoundColumn. Just set the appropriate properties which is much easier. Be sure to include a description of the custom properties and information about how they must be set in the documentation for your subclass.

This month's Companion Resource Disk contains three class libraries: Controls, the base Controls library, Sample which contains the Employee-specific value-setting combo class, and Combos which contains the generic value-setting combo class. There are two

forms: EmpCombo demonstrates the Employee-specific combo, while SetCVal demonstrates the generic combo. (The files are in VFP 3 format - use Compile ClassLib to convert the libraries for VFP 5.)

–Tamar