March, 1996

Advisor Answers

Visual FoxPro for Windows

Q: Is there a way to set a default font on a form?

–Roger Parker (via CompuServe)

A: Setting defaults is one of the strongest arguments for subclassing in Visual FoxPro. Your custom classes can include the defaults you want with no extra work at design time.

Let me back up a minute and answer your question directly. In fact, there's no way to set a default font for a form and have it apply to every control you put on that form. (Actually, even in FoxPro 2.x for Windows, this capability didn't work very well since push buttons used 8-point MS Sans Serif regardless of the default font.)

There are two approaches to a solution, depending exactly what result you want. Both involve subclassing Visual FoxPro's base classes.

The first level answer is that you should create your own custom class for each of the visually sub-classable base classes (a few of VFP's classes can only be subclassed in code, not in the Class Designer). In your subclasses, specify the font characteristics you normally want to use for that type of control. Then, use your custom classes in your forms (which should be based on a custom subclass of the form base class).

This is probably the solution you're looking for. However, there's another issue here. You may want to be able to change the font of a form and have all the controls on it change, too. This isn't as simple as setting the form's FontName and FontSize properties (or even the whole group of font properties).

The font properties on a form actually affect very little. Only text placed on the form with the form's Print method uses the form's font properties. The controls on a form use their own font properties. Measurements of the size and location of objects on the form depend on the font characteristics only if the form's ScaleMode property is set to 0 (foxels).

So how can you change a form's font properties and have that trickle down to the form's controls? If you only need to do it once, when the form first appears, put code like the following in the form's Init method:

```
THISFORM.SetAll("FontName",THISFORM.FontName)
THISFORM.SetAll("FontSize",THISFORM.FontSize)
THISFORM.SetAll("FontBold",THISFORM.FontBold)
THISFORM.SetAll("FontItalic",THISFORM.FontItalic)
THISFORM.SetAll("FontStrikeThru",THISFORM.FontStrikeThru)
THISFORM.SetAll("FontUnderline",THISFORM.FontUnderline)
```

The code uses the form's SetAll method, which allows bulk setting of properties. All container objects have SetAll which changes the specified property for every contained object. SetAll drills down to the lowest level contained. So, the code above not only changes the font for controls directly on the form, but for any controls inside those controls. For example, if the form contains a pageframe which contains a button group, the font of the individual buttons in the group is changed by SetAll. (SetAll takes an optional third parameter which lets you limit it to objects of a certain class.) You can use SetAll at design-time, too, either from the Command Window or from a Builder - there are several builders around designed specifically to change groups of properties at once.

Of course, what the code above doesn't do is adjust the size of the objects to make sure their text still fits in the new font. Making such an adjustment automatically is pretty complex - you'll need to play with FONTMETRIC().

The final case is where you want to change the form's font characteristics while it's running and have all the controls change, too. I can't think of too many situations where you'd want to do this, but it is possible. Use the same code as above. This time, though, put it in a custom method of the form and call that method whenever you change one of the form's font characteristics. You'll probably want to set the form's LockScreen property to .T. before the calls to SetAll and set it back to .F. afterwards, so all the changes happen visually at once, rather than redrawing each item as you go.

–Tamar