Choosing Options

Peter Vogel

Access provides three different ways for users to select among multiple choices: check boxes, option buttons, and toggles—and then there are option groups. Peter Vogel looks at what you can (and can't) do with these tools.

WORKING with my clients, I've occasionally wandered into "user interface wars." In these UI conflicts, different users are following different conventions for implementing their forms. In many cases, the user is the primary casualty, especially when the various controls are used inappropriately. One primary area of conflict and unfortunate choices is that of the three "choice controls": option buttons, check boxes, and toggle buttons.

All three of the choice controls allow the user to select among choices (see Figure 1). That doesn't stop developers from using other tools to select among choices, as I'll mention later. However, option buttons, check boxes, and toggle buttons have the advantage of being instantly recognized by the experienced Windows user as indicating a set of choices.

When should you use each control? The simplest answer is to use option buttons when the user must select one choice among many, and check boxes when the user can select many choices. While an option button can be used by itself (indicating that the user is either accepting or rejecting an option), a check box is a better choice for this kind of yes/no selection. Option buttons are more useful when used in an option group, which ensures that only one button in the group will be selected.

Check boxes can also be put in an option group, which will also ensure that the user can only check one box—but this contradicts the standard use of a check box. If your user is allowed only one selection, use option buttons. If you want to enclose a set of check boxes in a box on the screen to distinguish them visually from the rest of the form, use a rectangle, which won't force the user to select only one choice.

Toggle buttons

So where do toggle buttons fit in? My short answer is, "Mostly, nowhere." Toggle buttons have the unfortunate feature of looking like a command button. It's not until the user clicks a toggle button and it stays down that they discover that it isn't really a command button. Where a toggle button is used in a group, it's more obvious to the user that the button is *not* being used to make something happen but, instead, to choose among options. Figure 2, demonstrates this visual ambiguity. On the left, I have a single button—is it a toggle or a command button? Just looking at them it's hard to tell the difference. On the right, I have a group of toggle buttons, which the user is more likely to read as "a set of toggle buttons," especially because some are in the selected state.

The toggle button tends to be a niche solution, applying only in a few specialized cases. For instance, the overlap in appearance between command and toggle buttons supports using the toggle button for a special kind of selection: When selecting an option that causes something to change on the screen, a toggle button may be your best choice. For instance, a user interface design that I saw implemented by Servant Systems (www.servantsystems.com) used toggle buttons this way. A set of toggle buttons in an option group down the left-hand side of the screen allowed the user to select between different functions in the system. As each toggle was clicked, a different form was loaded into the subform control on the right-hand side of the screen. It was obvious to the users that they were choosing among options, and just as obvious that something significant was going to happen when they did.

Graphically, the toggle button offers some benefits over option buttons and check boxes. You can, for instance, put an image on a toggle button and Access will automatically "dither" the image when the toggle button is selected. If this graphical feature is important to you, a toggle button is your best choice. Since toggle buttons can be sized and a check box can't, you can use toggle buttons to make a stronger visual statement than is possible with the other two controls. Obviously, toggle buttons can be made larger than check boxes or option buttons can. Less obviously, toggle buttons can also be of different sizes to

Figure 1. Option buttons, check boxes, and toggle buttons.

Figure 2. Toggle buttons and command buttons.

indicate relative importance.

Alternatives

However, none of the choice controls can be databound in the same way that, for instance, a list box can. You can bind which field is updated by the control, but you can't generate the list of choices by databinding to a table. If you want to generate your options at runtime based on the data in your table, you're going to be forced to write code that either:

- puts your form in design mode while you add new controls; or
- makes visible a set of invisible controls added at design time.

Both choices will create numerous problems in laying out the controls on the page without overlaying other controls. If the controls are in an option group, you'll also have to resize the group. Where you want to use table-driven data, your best choice is a list box or (where screen real estate is limited or the list is long) a combo box.

This limits the choice controls to selecting among options that are constrained by the system's code and architecture, rather than the current state of the data. For instance, if you're allowing the user to choose between landscape and portrait formats when printing, an option group is your best choice. It's difficult to imagine more ways to print on a piece of paper. However, just because you currently only provide four different backgrounds for a report, it doesn't mean that you should use an option group to let users select among them. Unless there's some physical reason why more backgrounds can't be added, I'd put the choices in a table and allow the user to select from a list box.

Tree views are becoming more popular as a way of selecting among choices. However, the tree view merely provides a way of organizing multiple groups of choices in an expanding and collapsing framework. Most users are still unfamiliar with using tree views in this way, so you shouldn't rush to this solution just because your favorite development tool uses it. Where you have what would be an overwhelming number of choices, a set of tab controls is a better way to organize the options for your users.

These guidelines should allow you to recognize when you should use a choice control and when to use them most effectively. In addition to saving you from getting involved in the crossfire of a UI war, you'll also be supporting your users' expectations of what these controls will do. ▲

Peter Vogel (MBA, MCSD) is the editor of *Smart Access*. He's a principal in PH&V Information Services, which specializes in the design and development for systems that use Microsoft tools. Peter has designed, built, and installed intranet and component-based systems for Bayer AG, Exxon, Christie Digital, and the Canadian Imperial Bank of Commerce. He also wrote *The Visual Basic Object and Component Handbook* (Prentice Hall, currently being revised for .NET). In addition to teaching for Learning Tree International, Peter wrote its Web application development, ASP.NET, and technical writing courses. His articles have appeared in every major magazine devoted to VB-based development, can be found in the Microsoft Developer Network libraries, and are included in *Visual Studio .NET*. Peter also presents at conferences around the world.