"In saying what is obvious, never choose cunning.
Yelling works better."

Cynthia Ozick
"We Are the Crazy Lady and Other Feisty Feminist Fables"
in *The First Ms. Reader*

8 EDITING AND MANIPULATION

Using HTML Input Controls to Accurately Capture Users' Data

In contrast to the Viewing and Navigation layer, the interactions in the Editing and Manipulation layer result in persistent changes to the user's data. Owing to HTML's limited interactive vocabulary and the complex nature of these interactions, the Editing and Manipulation layer contains some of the most challenging and difficult aspects of the overall design. In addition, the technical limitations of Web applications are particularly visible in this layer. In many cases, the ideal level of sophistication and interactivity is too difficult or costly to achieve, resulting in a variety of implementation tradeoffs and compromises. Therefore, the appropriate and practical design solution requires a delicate balance between users' needs and technical feasibility.

This chapter approaches the problem from three different angles: the overall goals and purpose of forms, the proper use of HTML input controls, and how to combine input controls to facilitate complex interactions.

DESIGNING FORMS: THINKING IN TERMS OF THE WHOLE

Remember the chapter about structural models and the discussion about views and forms? When designing the structural model, the focus is on the flow of the site as a whole and how various types of pages contribute to that flow. After determining that a particular workflow requires a form, however, the challenge of the details still remains.

At the highest level, forms are essentially a mechanism for the application to ask the user questions: "When would you like to fly?," "What is your address?," and so forth. By contrast, views are essentially a mechanism for the user to do the asking: "What is the balance of my account?," "What do you have in the housewares department?," and so on.

As we all know, the rules of polite conversation are slightly different, depending on who's doing the talking and who's doing the listening. Therefore, the guidelines and conventions for form design are also slightly different from those for views.

PACE APPROPRIATELY

It's hard to hold a conversation with someone who talks too fast or asks 16 questions before giving you a chance to answer. The same is true whether you're interacting with another person or with a Web application. Therefore, determining the optimal length and density of information are two immediate concerns affecting the design of the overall form.

"We need to collect 67 fields of data. Should we create six really short forms, four medium-length forms, or one honkin' long form?"

The answer is, of course, one of balance. If the user needs to see all the options at the same time, the form will necessarily be long. Similarly, if choices in one part of the form determine the available choices in another part, the form will need to be divided into multiple pages. It's a bit like feeding babies; you want to be sure each bite is big enough to keep them interested, and you also want them to know what's coming for dessert. The balance, however, is to not stuff so much in their mouths at once that they spit it back up all over you.

When you're considering this question, a good place to start is a careful review of the functional requirements. Do you really need all those fields, or could some be eliminated? Are any fields optional? Can those fields be moved to another page or grouped together so as not to distract from the form's main elements? At a minimum, are required fields clearly differentiated from optional fields so that users can quickly identify what information they do and don't have to supply?

Complex forms can have so many input elements that the sheer volume of controls is overwhelming. Although it's acceptable to make users scroll, that doesn't give you a license to create a form of any length. Long forms are as painful and intimidating as a mortgage application and should generally be avoided. Imagine how many potential sellers get scared out of eBay simply because the form for listing auctions is so overwhelming (see Figure 8.1).

Long forms do, however, have two important advantages: efficiency and ease of error reporting. They are efficient in that they require only a single page load and they allow users to view the full set of options without additional navigation. They also enable errors to be detected, reported, and corrected in a single page. These advantages, however, bring with them the high cost of confusion, intimidation, and bewilderment, particularly for users new to the application.

8.1
The overwhelming quantity of options on eBay's selling form must intimidate any but the most determined of potential sellers.

In most cases, the better solution is to identify logical splits in the information so that the form can be presented in multiple pages. What could be presented as a single, lengthy form often turns out to be more manageable as a four-page wizard.

LIMIT NAVIGATION

As I mentioned in the context of structural models, Web applications are built from two fundamental page types: views and forms. Although a major function of views is navigation, such is not the case for forms. For all the reasons pointed out in Chapter 5, "The Structural Model: Understanding the Building Blocks of a Web Interface," the number of navigation options on form pages should be severely restricted. This applies to high-level navigation elements, such as tab bars or trees, as well as low-level navigation. If your form includes a navigational header or other ways to exit the form, at least some of your users are likely to fill out the form and then click one of the navigation paths without clicking the Submit button first. When this happens, it's impossible to know for sure if they meant to submit their changes but simply forgot to click the Submit button, or if they really meant to abandon the transaction. Even if the application is smart enough to trap every possible click out of the form and perform the submit behind the user's back, the basic problem remains: Exiting a form without using a command button creates ambiguity as to whether changes should be accepted or abandoned. Of course, you can't control the user exiting the form by way of the Back button, a bookmark, or typing in a new URL, but treating forms as modal dialog boxes with limited navigation and explicit Cancel and Submit buttons at least helps limit the problem.

INDICATE STATUS AND PROGRESS

If a form is part of a guide or a wizard, it should include clear indications of progress. If users are in step one of a seven-step process, it's only polite to let them know where they are and how many steps are left. Even when some steps are optional—choosing gift wrapping, for example—it is still important to inform users of the number of possible steps. The goal should always be to set appropriate expectations for users so that they feel some level of control over what is happening.

Figure 8.2 from the Opodo travel site is a great example of a progress indicator. Notice how the current step and all the possible steps are indicated in the progress bar.

8.2
A clear indication of where users are in the process sets appropriate expectations and keeps them in control of the experience.

SUPPORT INTELLIGENT FLOW AND KEYBOARD NAVIGATION

Well-designed forms also demonstrate balanced flow and intelligent ordering of elements. The user should be clearly drawn from the top of the form toward the bottom. Although a layout that draws the user across the page is possible, the established Web convention clearly favors vertical layouts.

In addition to enhancing their digestion of the experience, well-designed flow also enables users to quickly navigate through the form from their keyboards. This allows more sophisticated users to quickly progress through the form without having to move their hands back and forth between the keyboard and the mouse. For applications where users

repeatedly interact with a form, this is particularly important. For example, the Compose Message page of an email application is a form users will interact with over and over again. Being able to use keyboard navigation to move through the form's input controls enhances the interaction's efficiency and speed. To maximize the impact, it is important to consciously design and specify the tab order rather than rely on the browser's default behavior.

PROVIDE MULTIPLE CLUES

Another key to form design is to make the use of input controls as clear as possible through the use of clues such as labels, examples, and sizing. One of the easiest things to get right—and wrong—is the label for an input control. Although the subject of labels is actually a component of Layer 9: Text, it is still worth noting here that a major factor in creating clear forms is labeling input fields appropriately.

Providing examples of the expected input when there might be confusion is important, too. For example, a text field for an email address should be labeled "Email" *and* should include a sample of the data, such as "yourname@address.com." Examples are especially important in Web applications because automatically formatting the data for the user is more difficult than in desktop applications. Without some indication of the correct format for the input, users can't be sure of precisely what's being requested. This is particularly true of dates, where so many different standards are in use.

Length is another critical clue you can provide for users. Because text fields can contain any type of text string or number, they offer few clues about the expected input and pose the biggest risk of input errors.

However, setting the length of text boxes to a dimension appropriate for the input is one clue you can easily embed in your design. For example, if you have a field for a five-digit zip code, don't use a text box 24 characters wide. By setting the text box to a size appropriate for the input length plus a few characters for editing, you give users another valuable clue to what you're asking.

Finally, indicating which fields are required and which are optional is often useful. The current convention on the Web is to add an asterisk to the field label of required fields. Unfortunately, this convention doesn't work very well when most of the fields are required because you end up with indicators at nearly every field.

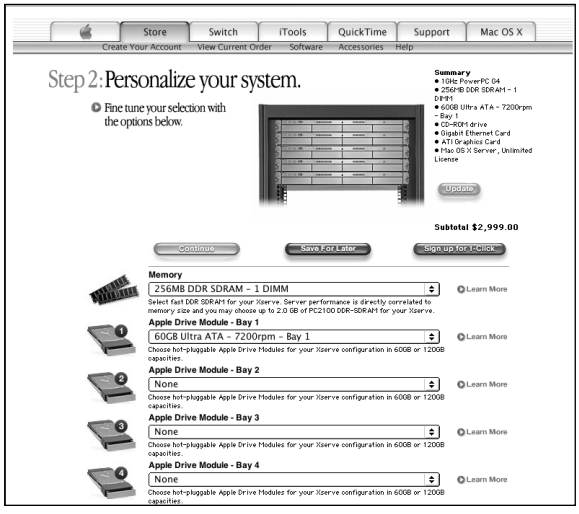
Although it's possible to flip the logic and indicate the optional rather than the required fields, this runs counter to the established convention and can create confusion. Therefore, often the best solution is presenting required and optional fields in different areas of the page. Geography again proves to be an effective indicator of difference.

MAKE CHOICES VISIBLE

One of the major purposes of forms is to give users a way to view and indicate choices. To support this basic goal, it is important to make the choices as visible as possible. Although the demand for screen real estate is always present, radio buttons, check boxes, and list boxes are the best way to communicate choices. Menus offer the promise of visual efficiency, but they do so by hiding choices, increasing the amount of interaction and exploration required for the task. As a result, they should be used sparingly.

Figures 8.3 and 8.4 illustrate the tradeoffs in using menus. These figures compare the system configuration pages from the Dell Computer and Apple Computer online stores. Although the Apple design benefits from the increased visual economy of menus, the Dell site more effectively communicates the available choices, thus reducing the amount of exploration required of the user.

8.3
Apple Computer's system configuration page requires the user to explore the different menus to learn about the available options and prices.



8.4
Dell Computer's online store uses radio buttons instead of menus so that the available configuration options are instantly visible.

Well-designed forms are a delicate balance of many different factors: the length of the form, the density of the layout, the clear use of input controls, and the flow of information.

INPUT CONTROLS: PICKING THE RIGHT TOOL FOR THE JOB

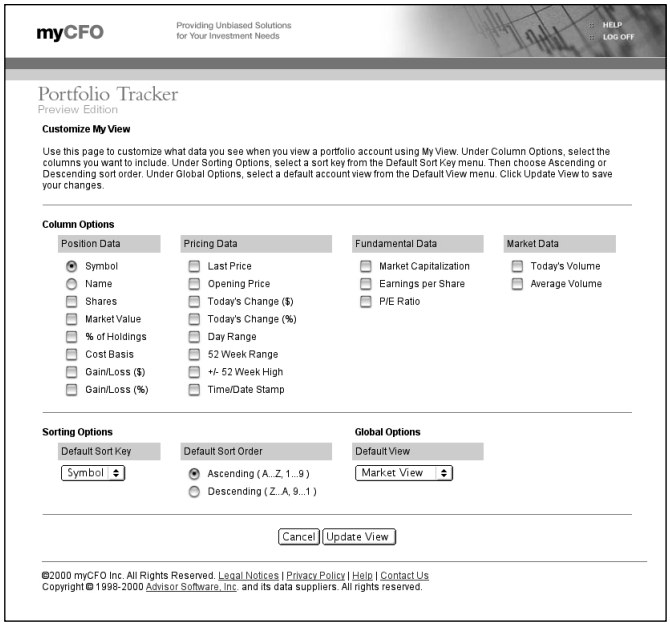
Like any design medium, mastery of Web design in general and form design in particular requires a solid understanding of the available materials. In this case, that means understanding the input controls available in HTML. Although the vocabulary of radio buttons, check boxes, menus, text boxes, and list boxes might seem limited and simple, there are many subtle and not-so-subtle aspects of each control. The following sections explore each control in detail, including examples of their proper use.

CHECK BOXES

As in the printed forms from which they are derived, check boxes indicate whether an item is selected. Although they often appear in groups, check boxes do not necessarily have any relationship to one another. Correctly used, the options that check boxes present are independent choices that do not affect any other options on the form. The one exception is when a check box is nested with other input controls to indicate an explicit master-slave relationship. This style of nesting is used when an option has associated "sub-options." For example, an email application could have an option to check for spelling errors before sending a message. If the user selected the spell check option, there could be additional options to indicate which language to use, whether to ignore Internet addresses, and so forth.

Remember that check boxes, whether alone or in a group, don't require users to select anything at all. Unlike radio buttons or menus, this means check boxes typically allow for a "null" selection. However, in some situations, users can select one or more of the available options but are required to select at least one; in other words, a null selection is not valid. In these cases, check boxes can take on a hybrid radio button behavior by using JavaScript to ensure at least one check box is always selected. Figure 8.5 illustrates a common implementation of check boxes.

8.5
The use of radio buttons for the first two items under Position Data ensures that the user always has at least one of these fields selected, but allows every other field to be optionally selected.



RADIO BUTTONS

Radio buttons derived their name and behavior from the channel selectors on car stereos. In the physical world as well as the virtual one, radio buttons are used to select one, and only one, choice from an exclusive list of options.

Like check boxes, radio buttons are a fundamental element of all graphical user interfaces, including the Web. Although they have the distinct advantage of showing users all the possible options, they also have the disadvantage of scale. In other words, they don't.

Although radio buttons work great with five to seven options, if you go much past that, the required number of radio buttons simply creates too much visual clutter to be effective. In addition, if the number of options varies depending on the user's information or the application's state, radio buttons are a poor design choice because the increase or decrease in the size of the group introduces too much unpredictability into the layout. Because of those two limitations, radio buttons are most appropriate when they represent a fixed set of two to seven options.

Unlike check boxes, there is always an exclusive and dependent relationship between the radio buttons in a single group. Radio buttons are not solitary creatures and should never appear by themselves. Regardless of the size of the radio group, one and only one of the buttons is always selected. That means one of the radio buttons must always be selected by default.

Although some of the best designed applications fail to indicate a default selection within a radio group, doing so places the group in an unnatural state. For example, RedEnvelope's product-ordering pages include a two-button radio group to choose whether or not to wrap the item in its exclusive gift box (see Figure 8.6). However, the application does not indicate a default selection, so users are forced to make an explicit choice between the two radio buttons. This approach succeeds in making users more aware of the gift box option, but it does so by using radio buttons incorrectly and undoubtedly causes input errors.

8.6
No default selection is indicated for the two-item radio group—an improper use of radio buttons as input controls.

When there are only two options, using a single check box in place of two radio buttons is sometimes possible. For example, in a stock portfolio application that can include or exclude closed accounts, it is more compact to have a single check box labeled "Include Closed Accounts" rather than one radio button for include and another for exclude. However, this solution works only when the two options are obvious opposites of one another, such as include/exclude, on/off, or hide/show.

LIST BOXES

As you might guess from their name, list boxes are used to display lists of items in a fixed amount of space. List boxes can operate in a single or multi-select mode, essentially mimicking a list of radio buttons or check boxes, respectively.

Because they require less screen real estate than a comparable group of radio button or check boxes, list boxes are an effective solution when five or more options are being presented. In addition, because they take up a fixed amount of space, they are particularly effective when the number of options is unknown or variable.

The height of a list box is expressed in lines and is one of the few formatting options available. A reasonable question is "What is the optimal number of lines to display in a list box?" A slightly vague but accurate answer is "Tall enough to reveal a useful number of items but short enough to be visually manageable"—in other words, not too tall and not too short.

More precisely, you should aim for between five and seven lines, preferably five *or* seven lines. Lists with an odd number of items are generally easier to digest visually because one of the list items lies at the exact middle. As a result, it's easier to break the list into two smaller units that are quicker to scan.

The height of the list box not only affects the form's visual layout, but also gives the user an important clue about the list box contents. This is similar to how the length of a text field helps communicate the field's intended contents. A list box that displays three lines implies that the total number of items in the list is relatively short, whereas a list box with a height of seven items implies that the total list is much longer. In addition, the height conveys an impression of the list's importance, with bigger lists being perceived as more important choices than shorter lists.

In addition to the height of the list box, HTML also gives you control over the list box's behavior—specifically, whether the user can select multiple items or only a single item in the list. Put another way, does the list behave like a group of radio buttons or a group of check boxes?

The unfortunate answer is that multi-select list boxes cause a number of usability problems and are an unacceptable option for the majority of Web users. When the list box is in multi-select mode, users can make a multiple selection by holding down a modifier key (for example, the Ctrl key for Windows) while clicking additional items in the list box. However, not all users understand or are even aware of this behavior.

It is also common for a selection to include items spread out across the list. As a result, users have to scroll through the list to view the full selection, often losing track of what is and isn't selected. Depending on the selection and the scroll position, the entire selection might be displayed outside the list's visible area, giving the erroneous impression that nothing is selected.

Suffice it to say that multi-select list boxes are riddled with opportunity for error and misunderstanding. If the situation calls for a multiple selection from a long list, there are better solutions than a multi-select list box.

A final point is that list boxes, like a group of radio buttons, should always offer a meaningful default. Similarly, if a null selection is allowed, the list box should contain an option labeled "None" instead of being displayed without a selection.

MENUS

If you compressed a list box so that it displayed only one item at a time, you would end up with a menu. Despite the differences in behavior and appearance, menus provide the same function as list boxes and radio buttons: making an exclusive choice from a list of multiple options.

The primary advantage of menus is visual efficiency. In addition to their ability to present a large number of options in a small space, their "now you see it, now you don't" behavior makes them particularly appropriate when the number of options is variable.

Unfortunately, menus have one major weakness: They're hard to use. Picking an item from a menu is one of the most complicated mouse operations there is. Selecting an option from a menu requires two clicks, a complex click-and-drag movement, or a combination of keys from the keyboard. To make matters worse, the click target for an item is fairly small, and the error recovery is zilch. If you pick the wrong item, you're all the way back to square one. Finally, the menu behavior hides options from users and gives them few clues about the menus contents. As a result, menus are not only an exercise in physical dexterity, but also a challenge to understanding and recall.

Despite these caveats, however, menus still occupy an important position in the interactive landscape. The challenge is knowing when and where to use them so as to minimize their disadvantages.

In addition to the ever-present concern for how all the form's elements work together, there are two other considerations specific to menus: how many items it contains and the default selection. Because menus can be difficult to navigate, the number of options should generally be kept short. Although there's no hard-and-fast rule, a list of more than 15 items is hard to look at, and anything north of 21 is difficult to use. These limits can be expanded, however, if the contents of the menu are sorted and well understood by the target user group. For example, a menu containing the 50 U.S. states might be a reasonable solution for a site catering to users in the United States, particularly if the state choice was optional. A list of 50 hotel locations, on the other hand, would not be appropriate because users would be unfamiliar with the list's content and would have to read through each option one by one.

The second consideration when using a menu is indicating a default selection. In many cases, an application won't have the necessary information to provide an appropriate default, so the real question becomes "What is the most effective way to communicate that the user needs to make a choice?" Two methods are commonly used in such situations: Don't indicate a default so the menu is displayed without a selection, or select a dummy item labeled "Select…."

Because the fundamental behavior of a menu is to indicate a selection, not having a selection is an invalid state for the control. Therefore, the optimal solution is to select a dummy item labeled "Select…". Even better is to reference the menu's contents in the selection—for example, "Select State…."

Menus are perhaps the most overused and inappropriately used control on the Web. Of course, they're also one of the most useful. Before you place them all over your forms, however, be sure you understand and accept the usability problems they're likely to introduce.

TEXT BOXES

Text boxes are used to capture strings of text, numbers, or both. They are the most straightforward input control from a behavior perspective, but because there is no real control over what users type in them, they are also a significant source of input errors. To help alleviate this risk, giving users multiple clues about the correct input is essential. As discussed earlier, these clues include clear labels, sample text, useful defaults, and appropriate sizing.

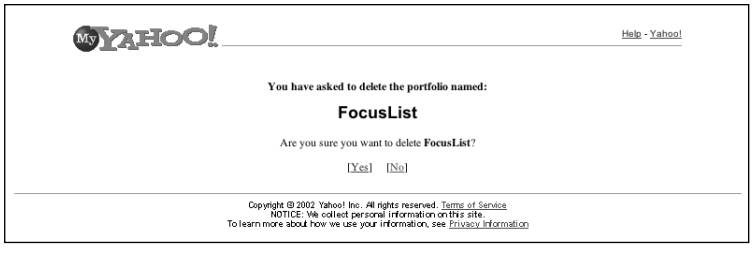
In addition to width, HTML can also specify whether a text box should display one or more lines of text. Although most text boxes display a single line of text, in some situations—the body of an email, for example—a multiline text box is useful. Whatever the case, it still comes down to setting the text box to a size appropriate for the input.

BUTTONS

The final commonly used input control is the button. From a technical perspective, buttons come in two basic flavors: Submit and Cancel. Submit buttons send a form's contents to the server for processing, and Cancel buttons throw out a form's contents and return users to the page from whence they came. From a user's perspective, however, a button's function is known by its name.

Because there is no control over a button's appearance, most sites use images in place of standard HTML buttons. Unfortunately, some sites also use links in place of standard buttons. The two objects have different uses, however, and should not be used interchangeably. Although there are rare exceptions, in general, links should be used solely as navigational devices, not to submit or save information. Similarly, buttons should be used only to initiate commands, not as navigation. Figure 8.7 illustrates mixing up the use of links and buttons.

Although HTML provides only a small set of input controls (summarized in Table 8.1), don't let the limited vocabulary dissuade you from creating sophisticated interactions. It's true that a more dynamic set of controls would make some operations easier to design and use, but the set that *is* available can generally get the job done.



8.7
In this example from Yahoo!, the designers have inappropriately used links in place of buttons.

Table 8.1 Standard HTML Interface Controls

Name	Purpose
Check box	Select none, one, or many from a fixed list of options, preferably seven or fewer.
Radio button	Select one and only one from a fixed list of options, preferably 7 or fewer.
List box (single select)	Select one and only one from a list of any size. Also used for lists containing an unknown number of options.
Menu	Select one and only one from a list, preferably fewer than 25 items. Also used for lists containing an unknown number of options.
Text box	Input a string of any length.
Button	Perform an action or a command.
File	Upload a file.

COMMON INTERACTION PROBLEMS AND SOLUTIONS

Lest you conclude there's nothing more to forms than a series of independent, isolated input controls, this section focuses on common interaction problems requiring multiple input controls working together. These problems include the following interactions:

- Selecting a single item from a small, medium, or large number of options
- Selecting multiple items from a large set of options
- Selecting a date

Although it would be impossible to anticipate or explore every interaction problem, the analysis of these problems should give you an understanding of how to approach and solve complex interaction problems in general.

PICKING A SINGLE ITEM FROM A LIST

One of the most common input operations in a Web application requires the user to select a single option from a list. Depending on the situation, the user might need to select an item from a list of 2, 20, 200, or even 2,000 possible options. Regardless, the interaction requirement remains the same: an exclusive choice from an exhaustive list.

There are three basic solutions in these situations, depending on the number of options and whether the list of options is fixed or variable. The simplest solution relies on a group of radio buttons and is appropriate for a small, static number of options. The second solution, appropriate for a large set of options, uses a list box. The last solution uses a menu and is appropriate for situations somewhere in between.

Small Set of Choices

The least complex incarnation of the single-selection problem requires the user to select an item from a fixed small set of options, generally between two and seven. There are two reasonable solutions in this situation: a group of radio buttons or a menu.

Although many designers faced with this problem use menus in place of radio buttons, the advantage of visual efficiency that menus offer comes at the price of hidden options and usability concerns. Although this may be an acceptable tradeoff for seldom used features or very lengthy forms, as a general rule, options should not be hidden from users. Menus hide options; radio buttons don't. Call it a guideline, call it a rule, call it a suggestion: Whenever possible, use radio buttons instead of menus.

When the number of choices depends on the user's data or the application's state, however, this choice isn't possible. For example, a stock portfolio application typically allows users to create custom views of their portfolios, displaying the list of views as a menu in the main portfolio page. The menu is an appropriate solution in this case because it enables the page layout to remain consistent regardless of the number of views the user creates.

Online clothing stores are an instructive example of a design that uses menus when a group of radio buttons would be more appropriate. For most clothing articles, users have to indicate both a size and a color. This is typically handled as two independent selections and presented as two different menus. However, size and color are not independent selections. Rather, they are two distinct characteristics that combine to describe a single item. In addition, the number of options in either control is typically fewer than seven.

Figure 8.8 shows a typical solution that uses menus for selecting clothing size and color. Figure 8.9 shows an alternative design, with text links in place of the menus. Compared to the standard menu solution, the text link solution has these key advantages:

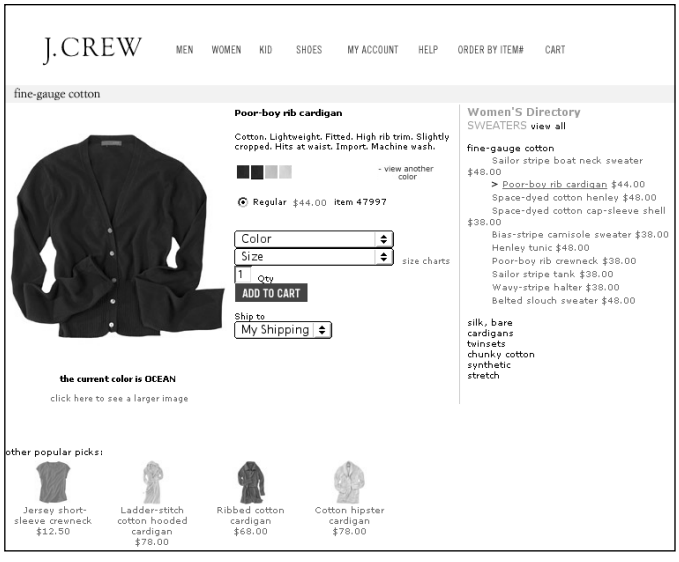
- **Visibility of choices.** The full set of choices is instantly visible without the user having to explore the contents of one or more menus.
- **Reduced click count.** The click count has been reduced by substituting a single link for the two menu choices and the Add to Cart button. An action that took three clicks—select size, select color, and add to cart—has been reduced to one.

- **Simple mechanism for communicating unavailable size/color combinations.** This solution easily handles unavailable size/color combinations. By contrast, the menu required the user to first select the size/color combination before the interface reported whether it was available.
- **Ease of use.** As an interface control, text links are physically easier to operate than menus, so this solution is easier to use than a solution relying on menus.

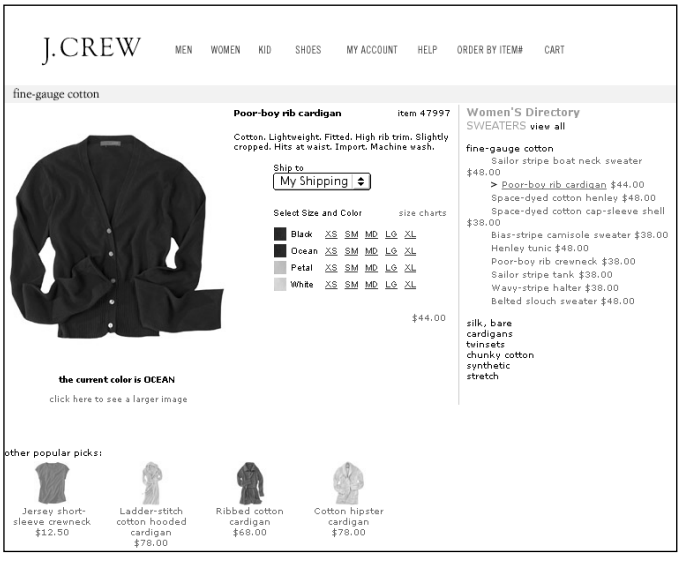
As is typically the case, however, this solution also comes with some clear disadvantages. Determining whether the following disadvantages are a reasonable tradeoff compared to the advantages is no simple question:

- **Limited scalability.** Although this solution works well in the illustrated situation, if the number of sizes or colors grew past seven, the size of the matrix would be so large that it would overwhelm the page.
- **No clear Add to Cart action.** The most suspicious part of this design is removing the Add to Cart button. Although the text link is a reasonable way to add an item, the interaction goes against convention and could easily confuse users.
- **Difficult error recovery.** By equating a link with the Add to Cart action, the solution does not have a simple method for users to review their selection before adding it to their Shopping Carts. This problem is compounded by the solution's inability to give users a way to recover from an incorrect selection, other than deleting the incorrect item from their Shopping Carts.

One way around these disadvantages would be to replace the text links with radio pictures, which would enable the Add to Cart button to be added, thus eliminating the two major problems. However, the visual weight of a radio picture would likely make the design even less scalable.



8.8
To purchase an item, the user has to navigate both the size and color menus, optionally indicate a quantity and ship-to address, and click the Add to Cart button.



8.9
In this alternative design, size and color menus have been replaced with a two-dimensional matrix of links.

In addition to their standard representation as circles, radio buttons can be represented as more meaningful icons. An iconic presentation of radio buttons is known as *radio pictures*. Although the two presentations are functionally identical (an exclusive choice from an exhaustive list of options), radio pictures are more visually efficient. A common example of a solution using radio pictures is the left, right, center, and justified text alignment icons in many desktop applications.

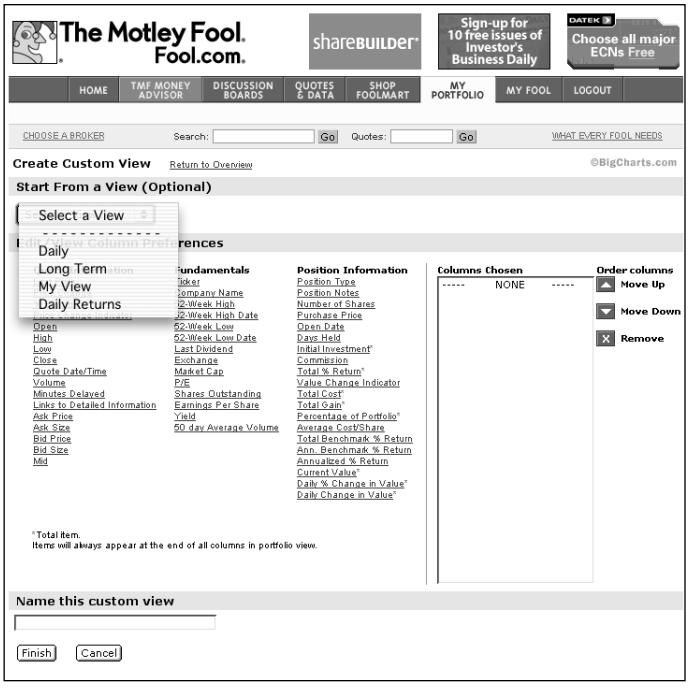
Other changes to the original design include eliminating the Quantity text box and placing the Ship To menu at the top of the shopping area. Because users rarely order multiple quantities of the exact same article of clothing—socks, T-shirts, and underwear being exceptions—the quantity option has been moved from the main product page to the Shopping Cart. In addition, because the Ship to option is less frequently used, it has been placed in a less visually prominent position.

Whether this solution represents a definitive, meaningful improvement over the original is a question best left to a usability study. Clearly the design represents a departure from convention—a move that should always be viewed with skepticism. In the perpetual search for improvement, however, examining the conventional solution in light of alternatives is always useful and often educational.

Medium or Unknown Number of Choices

With more choices—7 to 21, give or take—a different approach is required. The use of radio buttons is ruled out, thanks to the sheer number of options. The remaining options are a menu or a scrolling list box, although the former is suspect owing to its ever-present usability problems.

Still, menus do have their place, and Figure 8.10 is one of them. This page from The Motley Fool's portfolio tracker allows users to create a customized view of their portfolio. Instead of starting each new view from scratch, users can base a view on an existing one by selecting it from a menu.



8.10
The "Start From a View" option is an appropriate use of a menu because the control is not required and because the number of choices is variable but unlikely to be more than 20.

Three main factors justify the use of a menu in this situation:

- **Unknown number of items.** Depending on how many views the user has set up, the list could have 3 to 33+ different options. Because of this variability, radio buttons are inappropriate.
- **Small to medium range of options.** The number of options in the list, although variable, is unlikely to be more than 20. After all, how many different portfolio views is one user likely to create? With a range of 2 to 20, the number of options is reasonably manageable with a menu.

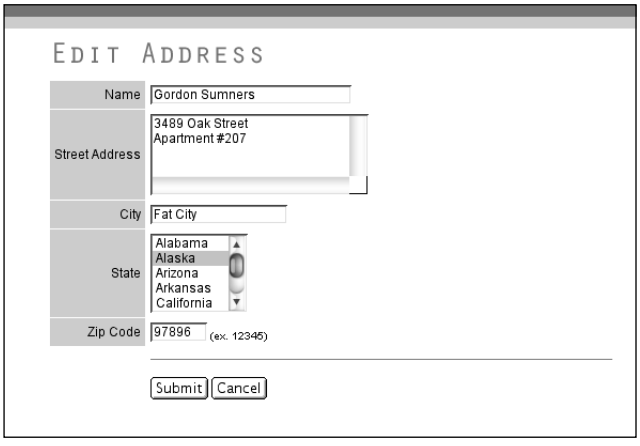
- **The control is optional.** Users don't have to select an existing view as a starting point. Therefore, the control doesn't warrant the visual weight and prominence of a scrolling list box.

In situations calling for the user to select a single item from a medium or an unknown number of choices, the challenge is ultimately one of balance. Although a menu or scrolling list box can be used, the optimal solution is the one that best allocates screen real estate given the situation.

Large Number of Choices

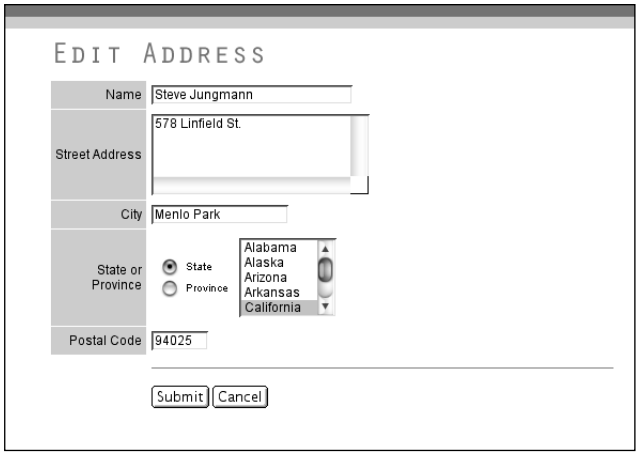
A third type of design challenge involves selecting a single item from a large list. What constitutes "large" is open to interpretation, but generally speaking, it's a number north of 21. In these cases, radio buttons and menus are ruled out by the volume of options. This leaves scrolling list boxes as the only viable alternative.

An educational point is found on almost every address entry form on the Web. The conventional design for address forms incorrectly uses a menu as the input control for U.S. states and Canadian provinces. The solution requires the user to navigate a menu of 50+ items, even though the same input could be captured with a scrolling list box (see Figure 8.11). Granted, the list box requires more screen real estate, but that cost is outweighed by the usability benefits of revealing all the options and providing an easier-to-use control.



8.11
This figure illustrates the use of a scrolling list box as the input control for U.S. states and Canadian provinces.

The use of a scrolling list box has been extended in Figure 8.12 to demonstrate how the contents of an especially long set of choices can be filtered by using an additional control. In this example, the standard list of U.S. states and Canadian provinces has been divided into two smaller groups. This style of interaction works well when the list can be filtered in two to four different ways. With a larger set of filter choices, a menu can be substituted for the radio buttons.



8.12
This example shows how radio buttons can be used as a filtering mechanism for a long list of choices.

When users are required to select a single item from an exhaustive list of options, there are two key considerations: the number of options and whether the number of options varies. Although radio buttons, menus, and scrolling list boxes are all acceptable solutions, the ultimate challenge is to find a balance between the need to make all the options visible, the ease of use of the input controls, and the visual weight of the different interface elements.

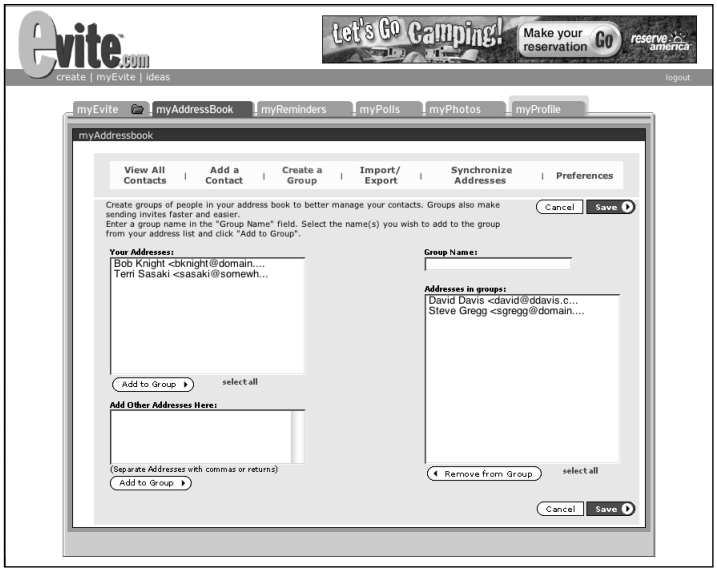
Picking Multiple Items from a List

A second common interaction problem is the ever-present "picking multiple items from an exhaustive list." Although multiple selection and single selection pose similar design challenges, the two are far from identical. In both cases, the elements supporting the interaction have to indicate the list of options as well as the current selection. For single–select controls, such as menus and list boxes, one interface element serves to simultaneously indicate the selection and the available options. With multiple selections, however, the interface has to communicate a selection involving any combination of options and, as a result, a different set of interaction mechanisms is required.

In situations calling for a relatively small number of choices, the issues parallel single-selection problems. Check boxes are substituted for radio buttons, and everybody moves on. As you might expect, however, that simplicity evaporates if the number of choices is variable or expands much beyond seven choices.

One potential solution for multiple selections is a scrolling list box in multiple-select mode. In this mode, users select multiple items by holding down a modifier key while simultaneously clicking items in the list. Unfortunately, multi-select list boxes have a host of usability and feedback problems, not the least of which is that many users don't even realize they exist.

An alternative solution uses two single-select list boxes working in concert with two or more command buttons. In this solution, one control contains the possible options, and the other contains the currently selected items. In addition, a group of command buttons is used to move items back and forth between the lists. Figure 8.13 from Evite shows a good example of this solution.

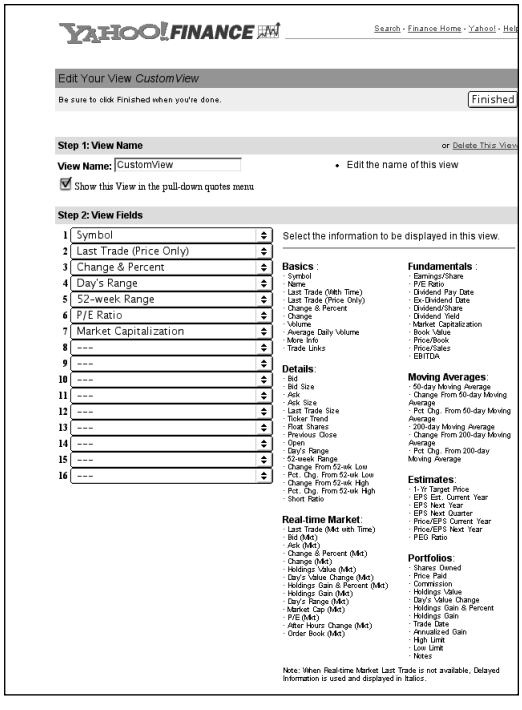


8.13
Evite's address book supports selecting multiple items through the use of a list box containing possible options and another containing the selected items. Items are moved between the lists by clicking a command button.

Harkening back to the days of Macintosh O/S 1.0, this motif is sometimes called "font/DA mover" or, in the more modern vernacular, "source/target list." Although this solution requires JavaScript, it is almost always the preferred alternative, certainly warranting the extra code and effort.

A redesign of the Edit View form from Yahoo!'s portfolio tracker illustrates how source/target lists can clean up a complicated multiple-selection problem. Figure 8.14 from Yahoo! is used to customize users' views of their stock portfolio. In this implementation, the user can customize up to 16 columns, each containing one of 76 different options.

8.14
Yahoo! portfolio tracker allows users to customize the view of their portfolio. The design relies on 16 menus, each containing 76 elements.



The most conspicuous problem with this design is the use of a menu containing 76 items—a poor showing in the usability race when there's only one such menu, much less 16 of them.

Aside from their function as selection mechanisms, the menus are also used to control the order of the columns. Unfortunately, although the menus are stacked vertically, the top-to-bottom ordering is translated to a left-to-right ordering when the view is being used. In addition to the oddity of this vertical to horizontal mapping, the use of menus as an ordering mechanism has another problem. Because the design doesn't provide a way to insert a column, users are required to change multiple menus if they want to place a data element between two existing elements. For example, if a user wanted to insert a new data element in column 2 but wanted to retain the order of the next 12 columns, he or she would have to reselect the elements for columns 2 through 13—a tedious task, even without 76 elements in each menu.

Finally, because there is no interactivity between the menus, this design does nothing to prevent users from displaying the same data element in more than one column.

On the positive side, the page includes a categorized list of all 76 possible data elements. By including this reference, the design mitigates some of the problems created by hiding the options.

Working backward from the implemented design, you can infer the following requirements:

- Limit the number of columns to 16.
- Do not use JavaScript or DHTML.
- Any element can appear in any column.
- Data elements can be custom sorted.

With these requirements in hand, there are a few alternatives worth exploring.

Option 1: Check Boxes

As shown in Figure 8.15, one alternative is to replace the menus with check boxes. Although this approach doesn't satisfy all the requirements, it does have some important advantages—as well as disadvantages—to consider:

Advantages

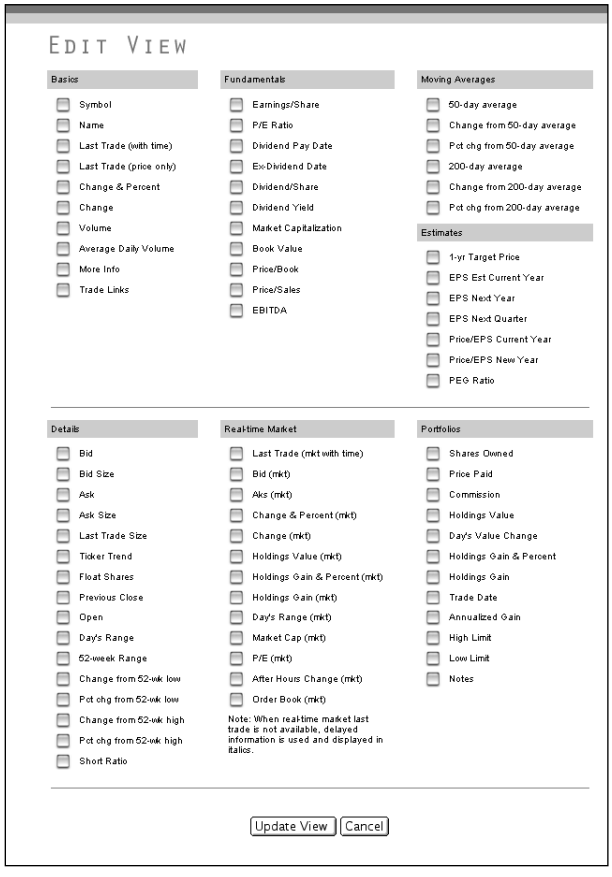
- Obvious presentation of available choices
- Simple selection interaction
- Prevents users from adding the same data element twice

Disadvantages

- No ready mechanism for limiting the selection to 16 data elements, thus requiring an error notification if more than 16 elements are selected
- No support for column ordering

Although the first disadvantage could be handled through an error alert, addressing the second would necessitate a reconsideration of the requirements.

Because the data elements can be grouped together in a meaningful way, the lack of support for column ordering might not be an issue. The nature of the data elements is such that determining a prescribed order that addresses most user's needs is possible. Ultimately, the questions rests on whether column ordering is essential to the primary persona and, if so, at what cost?

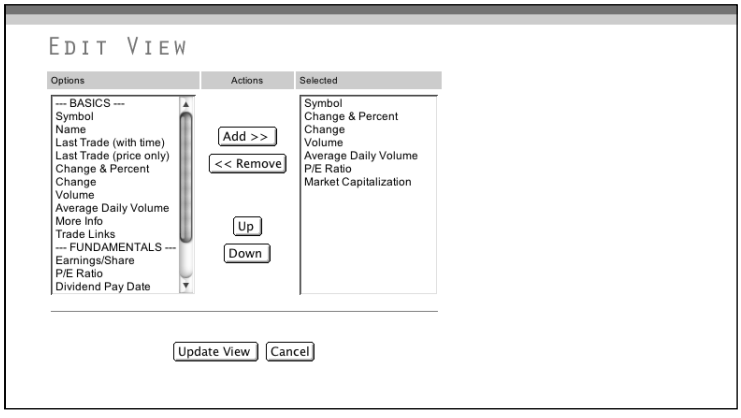


8.15
This redesign of Yahoo!'s Edit View page does not support customizing the order of the columns, but it simplifies the interaction in the original design.

Option 2: Source/Target List

A second alternative, relying on a source/target list, is shown in Figure 8.16. This option includes a source list containing the full complement of data elements and a target list containing the elements currently selected. This design also includes buttons for moving items in and out of the selection and for reordering the selected items. The combination of two list boxes and four buttons provides all the functionality of the original design but without the complexity of 16 menus.

8.16
This design simplifies the interaction by reducing the number of input controls needed. It also enables users to reorder data elements easily.



Compared to the other alternatives, this solution has the following advantages and disadvantages:

Advantages

- Clear communication of the current selection
- Easy reordering of selected items
- Simplified interaction using fewer input controls
- Visually compact and approachable
- Depending on the preferred use, elements can or cannot be added more than once

Disadvantages

- No obvious way to limit the selection to 16 items
- The scrolling list cannot display all options in a single glance
- Requires JavaScript

A variation of this design could integrate a menu as a filtering mechanism for the source list box. This would limit the number of items displayed in the list at any given time and help users locate specific items quickly.

Like the check box alternative, because this design requires JavaScript, it does not fully satisfy all the requirements. However, the improvement in the user experience is again enough to question the requirement.

As these examples show, designing an interface that allows users to select multiple items is no trivial matter. The unique environment of Web applications requires creative use of multiple input controls and a thorough understanding of the product's requirements and target users.

SELECTING DATES

The selection of dates, a problem found in many Web applications, is another complex interaction problem worth exploring. Because HTML doesn't provide any input controls specifically designed to select dates, Web applications are left to use the standard set of input controls to accomplish this rather complex task. Unfortunately, the Web's lack of interface standards is nowhere more apparent than in this particular problem. The following sections analyze the most common solutions to this problem with attention to the unique pros and cons of each.

Interface Requirements for Date Input

Not surprisingly, the purpose of a date input control is to enable users to specify a date—that is, the combination of a month, a day, and a year. In a desktop application, this is typically accomplished with a simple text box or

some sort of custom UI widget. In both cases, the application has special logic allowing it to instantly recognize and report invalid dates.

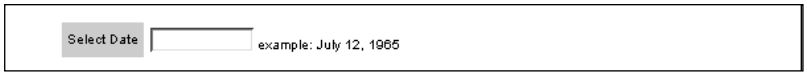
With those controls as a baseline, it is reasonable to assume the following requirements for a proper date input control:

- **Provide context.** Where appropriate, the input control should place a date in context. A travel itinerary or future appointment should be displayed in the context of a calendar, for example, but a user's birth date or credit card expiration should not.
- **Prevent errors.** The input control should prevent users from selecting an invalid, a nonexistent, or an incorrect date.
- **Be efficient.** The input control should be quick, efficient, and simple to operate.

Although Web applications are a different medium from desktop applications, these basic requirements remain the same.

Text Boxes

As shown in Figures 8.17 and 8.18, the option at the low end of the sophistication scale relies on a single text box or a group of text boxes for date input.



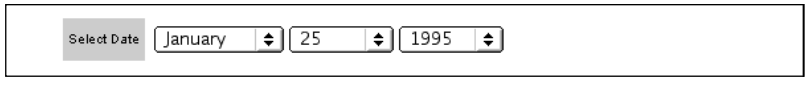
8.17
Although a single text box is a simple approach, it fails to offer any context or do anything to reduce errors.

8.18
Although a group of text boxes does not address all the shortcomings of the single text box, at least it removes the question of the correct order for the three pieces of data.

Regardless of whether the design uses one text box or three, this approach fails to meet any of the requirements. Although the single text box solution offers an example of the correct date format, nothing in the interface forces the user to follow it. By contrast, the solution with multiple text boxes more clearly communicates and enforces the required format. From a technical perspective, a solution using only text boxes represents the least amount of effort, but the obvious usability shortcomings generally make it a poor choice.

Group of Menus

The most common interface for selecting dates relies on a series of menus, one each for the month, the day, and the year (see Figure 8.19). Despite the popularity of this design, however, it is not without substantial usability issues.



8.19
Despite its widespread use, this solution fails to provide context or prevent input errors.

As an improvement on the text box approach, menus help reduce some input errors by ensuring the following:

- Values are entered in a controlled manner.
- The application clearly knows which input value represents the month, day, and year.
- The month, day, and year values are limited to an appropriate range.

Although the design could support logic dependencies between the day and month menus—for example, limiting the options for day based on the selected month to prevent the user from selecting February 30—these dependencies have their own host of usability concerns and engineering complexities. As a result, most applications that use this motif rely on server-side validation to confirm whether the indicated date is valid. Although this approach helps limit input errors, it does not completely eliminate them.

Another concern is the failure of this solution to provide context for the date. By simply reflecting a disembodied date with no reference to a monthly or weekly calendar, the solution doesn't communicate the date's relationship to past or future time. In some cases, this relationship is not important—credit card expiration dates, for example—but in others, such as travel itineraries, it's critical for the user to quickly understand that the 15th is next Tuesday.

Two factors account for this solution's popularity: conservation of screen space and technical expediency. In some situations, these advantages clearly outweigh any of the disadvantages; in others, however, a more sophisticated solution is necessary.

Calendars

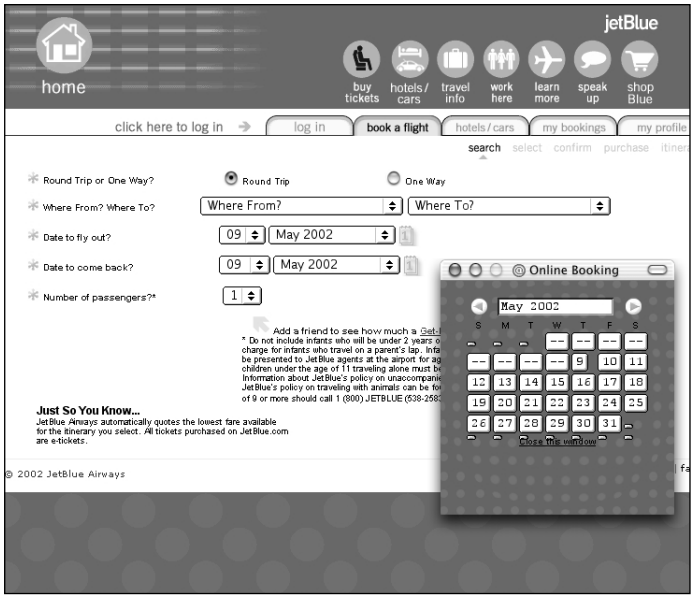
Another solution for handling date input relies on a calendar as the primary interface element. Although it's no great intellectual leap to conclude that calendars offer an easy-to-use input control, many applications don't use them because of the additional technical overhead. Of the solutions described so far, however, the calendar is the only one to satisfy the two fundamental requirements: context and error prevention.

Unfortunately, providing calendar-style input is no simple engineering task in an HTML environment. Although it is possible to create a calendar with nothing but standard HTML, changing the calendar's displayed time frame typically requires a trip to the server and page regeneration. For multiple date entries—travel reservations, for example—this behavior renders calendars disruptive and unwieldy. For a single date input, however, a fully HTML-based calendar, such as the one Evite uses (see Figure 8.20), can be a useful approach.

8.20
This page from Evite uses a calendar built with HTML. Clicking a date creates a new invitation for the indicated date.

Instead of placing the calendar directly in the layout, another common solution is displaying the calendar in a secondary browser window. Figure 8.21 from JetBlue exemplifies this approach. From the flight search page, the user can click the calendar icon, opening the calendar in a new window. When the user clicks a date, the calendar window closes and main page is updated to reflect the selected date.

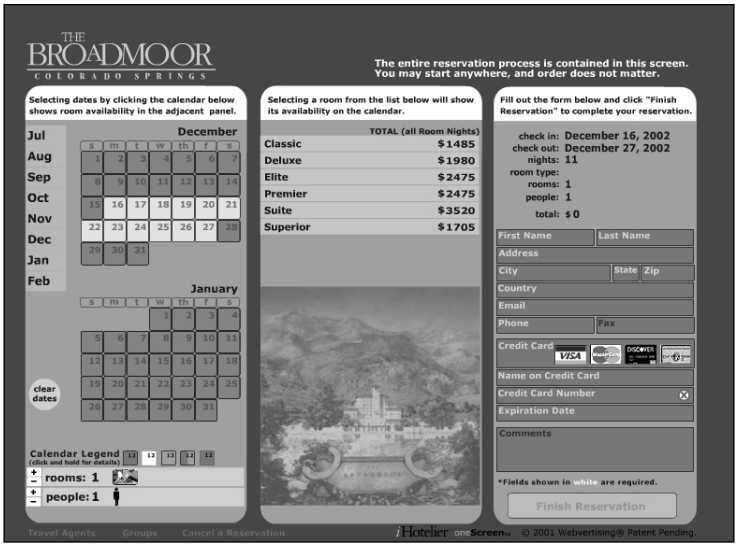
8.21
The flight search page includes typical menus as well as a calendar option.



Of the options discussed so far, this is the only one that satisfies the requirements for providing context and preventing errors. Unfortunately, it also compromises this functionality by introducing a secondary window and thereby violating the basic page model of the Web.

Another solution worth consideration also relies on a calendar motif, but instead of opening the calendar in a secondary window, it is integrated directly into the form. As shown in Figure 8.22, the Broadmoor Hotel's

reservation page includes two one-month calendars as well as links for changing the month being displayed. When the user selects a date, its color changes to white. In addition, after two dates have been indicated, the dates falling between the two are also highlighted, giving the user a clear, obvious way of viewing the date range of his or her stay.



8.22
The Broadmoor's reservation form uses two monthly calendars appropriately integrated into the reservation form.

Although this implementation requires additional engineering effort and image production, it is clearly a superior design from a user's perspective. It not only satisfies the requirements for providing context and preventing errors, but it does so without introducing usability problems.

Owing to the transaction orientation of many Web applications, selecting and entering dates are common interactions. Unfortunately, the requirements for an ideal date interface dictate significant design and engineering efforts. The results of this effort, however, are more control for the user and fewer errors—two benefits worthy of substantial effort.

SUMMARY

In most applications, the number of form pages is dwarfed by the volume of view pages. A lower number, however, should not be assumed to mean less complexity or importance. Creating forms with integrity and elegance is one of the most difficult and important challenges of the interface design process. Well-designed forms are a critical component of an intelligent, enjoyable, and satisfying user experience. By contrast, poorly designed forms inevitably lead to user frustration, confusion, and disappointment.

If resources are limited and time is in short supply, forms are perhaps the single most important area in which to focus your efforts. Here are a few of the key design principles to keep in mind:

- **Pick an appropriate pace.** Do not overwhelm users with long forms that intimidate. Likewise, do not insult them with needlessly simple forms that fail to contain a task of satisfying dimension.
- **Limit navigation.** In general, the navigational paths out of a form should be limited to explicit Submit and Cancel actions. Eliminating navigational elements from a form focuses users on the task at hand and prevents them from exiting the form without definitively saving their changes.
- **Provide multiple clues.** A well-designed form takes advantage of as many different communication channels as possible. Clear labeling of fields, appropriate sizing of text boxes, examples of correct input, and obvious indications of required versus optional fields are all important clues to a form's use.

- **Make choices visible.** Forms should not be an advanced version of hide and seek. Relevant choices should be clearly visible at all times. Users should not be required to explore an interface to accomplish basic tasks.

A theme running throughout this chapter has been the importance of error prevention. Although a perfect world would be free of all such user or application errors, the world of Web applications is far from that ideal. In the next chapter, you'll turn from the subject of form design to the subject of help, status, and alerts.