CHAPTER 16

Adding a Form to Your Site

In This Chapter

- Starting a form
- Adding form fields
- Setting user navigation within a form
- Adding buttons
- Using advanced form fields
- Validating your form data
- Troubleshooting forms
- Using GoLive Actions with forms
- Dealing with nonstandard tags

Chances are good that you've filled out a few forms in your day. When the form is on paper, you complete the relevant information and then someone most likely transfers the information to a database or some sort of list — doing so by reading your information and typing (or retyping) what you've already entered.

The Web introduces a new era for forms, enabling the data you enter to land directly in the database or list without being touched by human hands. (Or, at the very least the data goes into an e-mail and is sent to you to be copied and pasted.)

So what might you do with a form on the Web? You can collect information, as you'd expect. For example, you can ask users for their names, e-mail addresses, shipping addresses, music preferences, favorite Thomas the Tank Engine railroad car, and so on.

GoLive provides an excellent, easy-to-use interface for creating forms on your pages. There's an entire tab in the Objects palette replete with all of the field-building elements you need to create a beautiful form — fields, radio buttons, checkboxes, list boxes, buttons, and form tags.

Understanding How Forms Work

Did you ever imagine using a secret decoder ring? Well, Web page forms aren't exactly rings (unless you count the data's trip to the server and back a circle), but when you use a CGI you're definitely working with a decoder. In fact, you're also working with the encoder.

Gathering data with the form document

What the user sees on a page when he or she submits information is just a part of the form story. Behind the scenes an encoded message is being constructed from commands called *form actions,* from any *hidden fields* you have placed within the form, and from the form's *controls* (which are the various types of fields that collect the user-entered information). It's not being encoded for secrecy, though. It's encoded so it can make the trip across the Internet in one piece. When you choose a ready-made CGI or other agent, the person who created it will tell you what command you need to provide in the form.

Telling the form what to do with the data

The parts of the form that the user fills out are called the *controls.* They contain, for the most part, all the information you desire from the user. In order to tell that information where to land within a database or how to look in an e-mail or elsewhere, you provide a set of *matching information,* such as the name of the field that matches the user's input.

In addition to the information the user provides, you may need to know things like what browser or platform a user is sending from so you can send back a page nicely formatted for each user, or because you're counting how many people from each platform use your site. (You can't collect private information from a user's hard drive — just the browser-information-type stuff. For example, the server's name, path, realm, or IP number. The capability to collect this must be programmed into the CGI [or other agent].)

If a user provided a username and password, you don't want to ask for them each time, so you can hide the information within the page, too — just while dealing with that one user. If you're creating an online order system, the form needs to carry product codes along with the user-entered quantity.

The form needs to know where to send the data, so the form has to be assigned some sort of Action; this is called the Form Action, and you supply that too. The form also needs to know how to carry the information to the Web server, so you have to tell it which Method to use.

As you create your form, the page takes on the look the user will see while the Inspector shows you the behind-the-scenes stuff that you need to provide but not bother the user with. Be sure to keep the Inspector open and handy.

Now you're armed with possibly more than you ever wanted to know about forms, like the one shown in Figure 16-1. In this chapter, I'll show you each of a form's building blocks and how GoLive enables you to add them easily to a page.

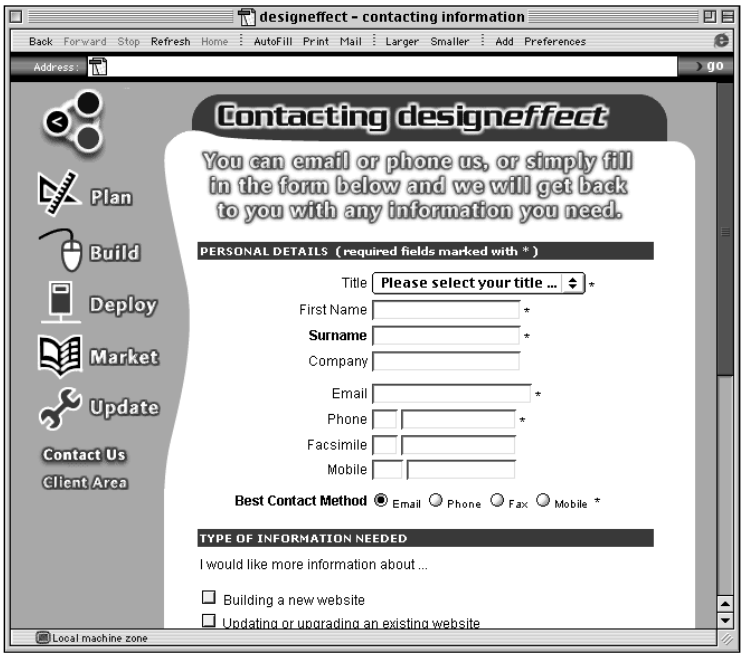


Figure 16-1: Forms are a good way to collect information about your customers.

Starting a Form

A form can exist anywhere on your page, as long as you place it inside the Form container. The form doesn't care where you place it on the page. However, due to the way browsers may render form elements differently (and display them in different sizes), in order to have your form look its best and keep the fields, directions, and labels lined up, place your form items within a table inside the Form Container. You can also place them directly inside the Form container (using the cursor), or you can use a combination of both. Grids are not recommended.

Previous users of GoLive will notice that the way forms are constructed in GoLive 5 is different than in previous versions. Whereas you used the Form Start and End tags before, GoLive now uses a container that writes the end tag for you (the reason for this is so your forms always meet the W3C HTML specifications). This may result in a misaligned tag becoming visible in Layout mode. This is dealt with later in this chapter in the "Form Troubleshooting Issues" section.

Later I mention that another option for setting up your form is to use the new HTML 4 fieldset. See "Placing fieldsets in a form" later in this chapter.

To create a form using a table to contain your form elements, follow these steps:

1. Place the Form container on your page, as shown in Figure 16-2.

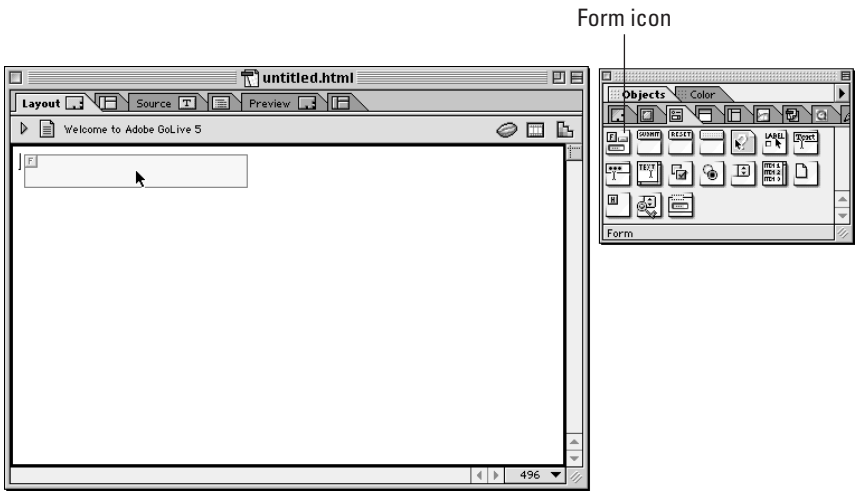
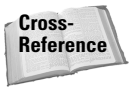


Figure 16-2: Add a form to your page by placing the Form icon from the Objects palette's Form tab.

2. Place the table into your Form container and format it.

   Typically a form has one column at the left for labels — text that serves as a visual clue to tell your viewers what you seek. The fields are usually placed in a column to the right. You may also want to put a narrow column between the label and field columns to provide a bit of space between the two, or you may be able to use cell padding or spacing to separate the columns.



To learn about tables, see Chapter 13.

To create a form directly on your page, drag the Form icon into place, and then add the form elements inside the Form container as needed.

The Form Inspector, which becomes active whenever the form tag is selected, is where you provide the basic necessary directions for your form (see Figure 16-3). The rest of this section explains each of the Form Inspector's form information areas.

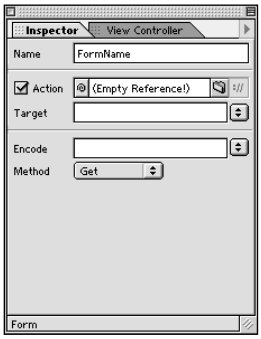


Figure 16-3: The Form Inspector

Naming your form

Naming your form simply helps you keep your sanity by helping you easily identify the form when you need to work with it — if you give your form a descriptive name. Putting more than one form per page makes a name even more important. Your form name should be unique to your site.

For example a form that enables a person to add his name to a list or sign a guest book can be called "AddGuest" or "GuestAdd."

Specifying an Action

Every form must have an Action. The Action tells the user's browser where to go with the information sent when the user presses the Submit button.

The Actions are specific to the CGI or server-side technology you're using. Commonly, the Action is the URL of the CGI document or the name of the page that contains the code for whatever server-side technology you're using.

To enter an Action, follow these steps:

1. Make sure the Form icon on your page is selected so the Inspector window says Form Inspector.
2. Specify the Action in the Actions field.

   If you're using a server-side technology other than a CGI, check with its documentation or with your site host.

   If you're using a CGI, the Action is the address of your CGI script, which is a file that either resides within your site or in a specified directory on the Web server. Check with your site host about this.

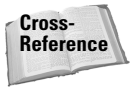
   - If the site host is providing the CGI, it probably provides a list of specific CGIs in a common folder and the address of each of them.

- If you are permitted to use your own, you'll be told where on the server to store the CGI so you'll know the path to it; then add the name of the CGI along with any other specific instructions the CGI provides. CGIs are commonly stored either in a common folder on the server, or within a specifically labeled folder set up for you within your own folder on the server.

The Action sometimes contains more than just the name of the CGI or server-side program. Sometimes it includes a command the program will recognize. The program's documentation will guide you.

If your CGI script is on the Web server and your site host tells you to use a full URL (beginning with `http://`) you can store the address in the External tab of the Site Window, and then use Point and Shoot to designate it. The reverse is also true. If you type the full URL into the Inspector, it will automatically be added to the External tab of the Site Window if you are working with a site open.

If your site host lets you store your CGI script in your own folder on the server, you can store the CGI script in your Site Window. In that case, if the script is already in your Site Window, you can Point and Shoot to it or use the Browse button.



To learn about storing URLs, take a look at Chapter 11.

Targeting form results when using frames

After your form calls upon the CGI, the CGI returns results — presented with another page. You might assume the Target option in the Inspector is where you tell the CGI which page to return, but it's not. The page to be returned is more commonly designated in a hidden tag unless it is permanently embedded in the CGI or uses another scheme.

The target GoLive provides for in the Form Inspector is only applicable when you are using a frameset. When your CGI returns a page, the target determines which frame the page will land in. If you are using frames, choose a frame target from the pop-up menu. If you are not using frames, leave this field blank. To learn about frames see Chapter 15.

Encrypting form results

If your site is on a secure server, choose the server's encryption method from the pop-up menu. Check with your site host if you're not sure whether this is the case, or, if it is, what encryption method to select. Encryption is not commonly used and requires that the server be running specialized encryption software.

Of course, it is preferable to encrypt sensitive information. Another, simpler way to encrypt information, such as a credit card field, is to use a simple Perl script or JavaScript that can work on a field-by-field level. Many such scripts are floating around on the Web. I point you to a few script archives at the end of this chapter. This does not require use of the Inspector's Encryption field.

Choosing a method

The method your form uses determines the way that information is carried from the user's browser to the server. Two methods are available, as follows:

- **Get** adds the user's information to the URL that is sent back to the server. This is a very limited method, as URLs can only contain a specific number of characters and then cut off the rest of what you may need sent. It also limits you to using only ASCII characters. Additionally, the sent information appears in the URL — not cool if your form contains hidden data. One reason you might want to use the GET method is if you would like visitors to your site to be able to bookmark specific queries (because the values sent are stored in the URL).
- **Post** is the preferred method. It can carry as much data as desired, and the information carried isn't visible in the URL.

Some hosts only permit the use of GET, although this limitation is rare these days. Check with your host before creating your form.

After you have entered this basic form information, you are ready to begin building your form. Information or search requests are submitted via the various fields on you form.

Using hidden tags

Hidden tags are sort of secret codes you place in your form to communicate with your CGI. No one set of such codes exists. Instead, each CGI requires its own specific tags and may call them commands, variables, or perhaps something else.

For example, say you're using a database that has many layouts. A hidden tag would tell the database which layout to access. Another tag can tell your CGI script which page to return to the user upon completion of a specific exchange of data. A hidden tag can also pass information from one page to another.

Passing information within hidden tags makes it easier for you, the page creator, to update the information without having to learn how to open the CGI and hard-code these values into the CGI. Support for the hidden tags must be built into the CGI in the first place, though.

Hidden tags can be placed anywhere within your form tags. It is common, and recommended, that you place your hidden tags at the beginning of the Form container. That gets the work out of the way so you can go on to focus on the fields and the aesthetics of your page. No limit exists to the number of hidden tags you can place in a form. In fact it's common to call upon several.

To use a hidden tag, follow these steps:

1. Drag a Hidden tag from the Forms tab of the Objects palette, placing it inside the Form container (outside of the table) or within a cell. Placement of the hidden tag is not actually critical, but it's most logical to provide all such information up front or in logical groups.

   A hidden tag appears as a small square with a capitol H on it (as shown in Figure 16-4).

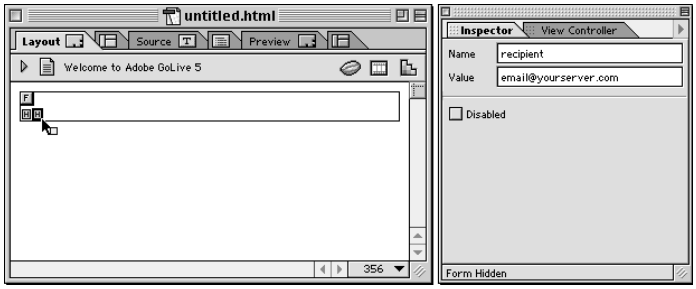


Figure 16-4: A selected hidden tag and the information it contains

2. Open the Inspector, now called the Form Hidden Inspector.
3. In the Name field, enter the name of the variable (command) you are passing.

   Your CGI script or middleware may call for several hidden tags. You can place them one after another. Their order is of little consequence unless you're instructed otherwise. Choose the variable you need to use and enter it here. For example, a variable that tells a database which layout to receive information into may be called "layout," so you would enter the word "layout." A variable can also be a customer ID number or the quantity of an item ordered. In Figure 16-4 the Hidden field carries the e-mail address where the information will be sent after the form has been processed.
4. In the Value field, enter the value that you wish to pass along with the chosen command.

   This value must be one that is appropriate for the variable you enter in Step 3. In Figure 16-4, because the Name is "recipient," the corresponding information is the e-mail address.

To learn about the Disabled option, see "Setting User Navigation Within a Form" later in this chapter.

All hidden tags look the same on a form. To see what each contains, select the tag while in Layout Mode and look at the Inspector, which will be the Form Hidden Inspector. You can also learn a tag's contents by keeping the Source code window

open as you work, or by using Outline mode or Source mode. The code for a hidden tag, by the way, is as follows:

```
<input type="hidden" value="hiddenValue" name="hiddenName">
```

Adding Form Fields

Form fields, or *controls* as they are generically called, are the areas where the user enters data. If your form collects names and addresses, the fields are where that information is placed. If the form requests a search, the fields are where the search criteria are entered.

GoLive fully supports all standard field types (available form fields are determined by the HTML specification, not by GoLive). You can choose a type of field for each piece of information you want collected on your form. Two field types (text field and text area) enable freely typed text; an additional field (password) enables text but hides it for passwords. The rest of the field types enable you to provide predetermined responses (radio buttons, for example). Your choice of which to use determines whether the user must pick only one response, or may pick several.

The only set rule about placing fields is that you have to place them inside the Form container. Any values that are set in fields placed outside of the Form container are ignored and do not get sent to the Form processor. Other than that you can place your fields anywhere you want, using the same formatting and alignment controls for working directly on the page or working within a table. As mentioned previously, placing fields in a table helps keep them well aligned. After you create the form you can set up the *tab order,* the order in which users will jump from field to field when they press the Tab key.

For users to know what is expected of them in each field, you can provide directions by adding text labels as you do on any page or in any table. For example, to let users know you want their first names entered in a field, you would use a Label from the Form tab of the Objects palette to label that field "First Name."

All form fields are placed on the page the same way as just about everything in GoLive: Drag them from the Objects palette into place on your page, or double-click the icon in the Objects palette to have it land by the cursor. All settings for the fields are then entered in the Inspector. The settings vary depending upon the function of the field.

Remember that you need to tell your user what is expected for any form field or data input element. You have two ways to do this. You can simply type instructions and field labels for the user or you can use the label tag. The Label tag adds excellent functionally to all fields, especially radio button and checkbox fields. To find out all about Labels, see the "Adding labels to form fields" section.

Because the most efficient way to organize a form is within a table, you can put field labels (whether hand-typed or placed using the label icon) into the left column of a table, and then put the actual form fields in the right. To place a space between the label and field you can have a third column between the label and field columns. Set the middle column to a few pixels set width. (See Chapter 13.)

One field setting to pay close attention to is the name. Your CGI may require certain naming conventions. For example, if your form is feeding information into a database or requesting a search within a database, the form fields probably have to *exactly match* the names in the database. Be careful not to add extra spaces to the name as that can throw off the CGI and the data will not arrive at the database. The same may hold true for other destinations as well. (If you're using a prewritten CGI, see its instructions. If you've asked someone to write a script for you, ask your provider for specific directions when you receive the script. If you've written the script yourself, you should know what to do.)

In the next sections, I'll go over all available form fields and provide instructions for using each one.

You can use JavaScript to validate a form when it loads into the user's browser, or to validate that a user has completed all form fields properly prior to submitting the form. (Some simple examples of this appear later in this chapter using both GoLive Actions and custom JavaScript.) You can also use it to create a cookie to help keep track of returning visitors, and to give people the convenience of not having to fill out the same forms over and over.

Text field and password field

A *text field* is a one-line area in which a user can type or paste text. A *password field* is exactly like a regular text field except that all characters typed into it immediately appear as bullets or asterisks so the information can't be viewed by someone looking over the user's shoulder. Figure 16-5 shows the results of using each field.

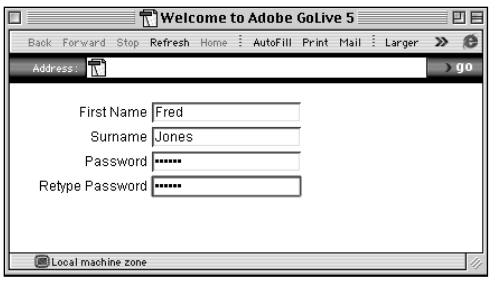


Figure 16-5: A few text fields and a password field as they might look in a browser (the labels to the left of each field were added into the left column of the table)

The setup for each field is identical. In fact, although an icon exists for each on the Objects palette, you can change one to the other and even back again by clicking a

checkbox in the Inspector. The only thing that changes with each field type is the Inspector's name.

Follow these steps to add a text or password field to your form:

1. Drag the text or password field icon from the Objects palette into place on your form. The Inspector palette will now look similar to Figure 16-6.

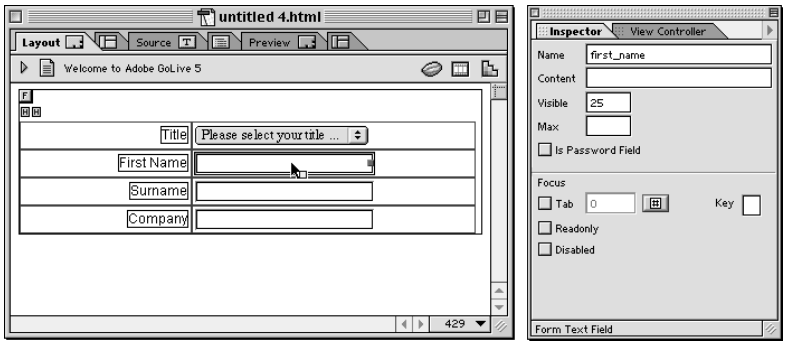


Figure 16-6: A text field being set up in the Form Text Field Inspector

2. By default, the field Name area of the Form Text Field (or Password) Inspector says textfieldName. Select this temporary name and enter your own unique field name. (Each field name must be unique within its form.)

   Remember to check about naming your fields. Often they have to match names in a database or CGI. Capitalization can often also count, as can extra spaces before or after the name.

3. *(Optional)* If you want the field to contain content by default, enter that content in the Inspector field called Content. Some people like to have a field say things like "Please enter your first name" or just "First Name." Most of the time this is left empty so users can see that they have not entered anything there yet.

4. In the Visible field of the Form Text Field Inspector, enter the width — in characters — that you wish the text field to be.

   This sets the width of the field and therefore determines the number of characters that can be seen at any one time in the field. This does not affect the total number of characters that may be entered; it is possible for text to be in the field but not appear, or for users to use their arrow keys to scroll through their entry.

   As with any other part of your page, if you make the field width longer than a browser window, the browser will show horizontal scrollbars.

   A text field's size can also be set visually on the page by dragging its right (and only) handle. As you drag, the number of visible characters automatically adjusts.

5. *(Optional)* If you wish to limit the number of characters a user may enter in this text field, enter that number, again in characters, in the Max field of the Form Text Field Inspector.

When you're collecting information that contains a set number of characters, such as an area code, phone number, zip code, or social security number, limiting the number of characters permitted for a field can help prevent user mistakes.

If you want to change a text field into a password field, check the Is Password Field box. To turn a password field into a text field, uncheck the same option.

Each of the other options for this field is discussed later in the chapter.

Adding labels to form fields

Labels are another new-to-HTML-4 feature that make forms easier for the user. When you create the form you link each radio button or checkbox's option to the actual button or checkbox. Then when the user clicks the label it has the same effect as clicking inside the radio button or checkbox. Aptly enough, this tag is called `<label>`. In source mode it begins with `<label for . . .>` and ends with the closing tag `</label>`.

Because of a label's close association with a field's choice, text readers for people with disabilities have an easier time with forms when labels are used.

Labels work staring with Internet Explorer 4 (Mac and Windows) and in iCab (Mac). They don't exhibit the added functionality in the Netscape 4.*x* browsers on the Mac or in Windows. In any browser that doesn't enable clicking the label, labels just appear like normal text, so you can use them in your forms without worrying about compatibility problems with older browsers.

To add a label to your form, follow these steps:

1. Drag the label icon from the Objects palette into place by the choice or option it will identify.
2. Edit the label by clicking the default label (which says Label), select that text, and then type your own words.

   The label is a text area so you can press Enter/Return to move text to a new line or press Shift along with Enter/Return to create a line break that doesn't add space between the lines of the label. Format as desired.
3. Set the label to activate the form's choice in one of these two ways:
   - Point and Shoot from the Inspector to the label's form element as shown in Figure 16-7.

- Press ⌘ (Mac) or Alt (Windows) as you click the Label's border and drag to the element you are labeling. GoLive will assign a cryptic-looking reference to the label. The reason for this is to guarantee the uniqueness of the reference when two labels might be linked to different form fields of the same name (radio buttons, for example).

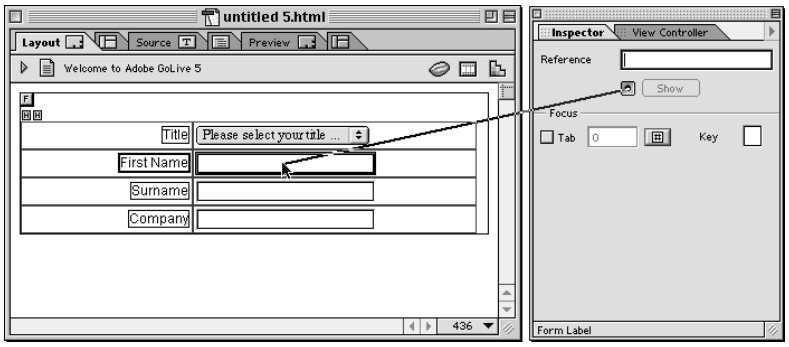


Figure 16-7: You can associate a Label and choice by Pointing and Shooting from the Label's border to the choice.

4. To verify your link between label and option, click the Show button in the Inspector. While this button is pressed, the link appears as shown in Figure 16-8.

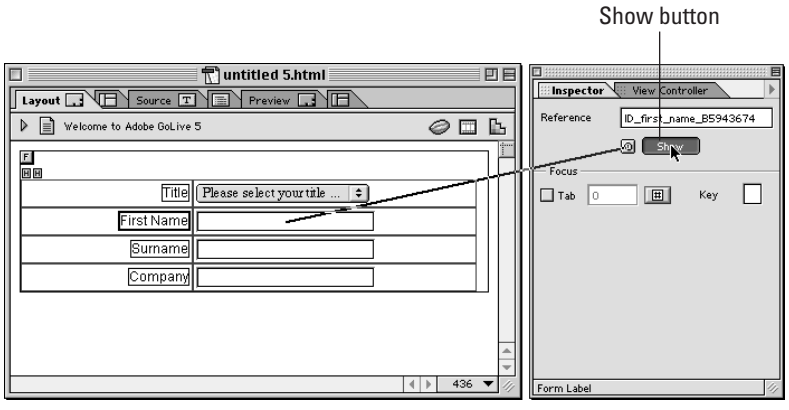


Figure 16-8: Press Show to verify an association between a label and the option it will activate.

Adding a text area field to your form

A *text area* is similar to a text field except that it provides extra lines to hold more text. A Text area also has a vertical scrollbar in order to accommodate plenty of text. Users type within a text area, as they type anywhere else on their computers. The Return or Enter key (on the letter part of the keyboard) adds a normal return, creating a new paragraph. However, the Enter key on the number pad may not have the same effect. This can differ depending on the browser (Internet Explorer may type a character, whereas Netscape usually ignores it). If this is important to the functioning of your form then you may need to tell viewers of the site that this can occur.

To add a text area field to your form, follow these steps:

1. Drag the Text Area field icon from the Objects palette into place on your form, as shown in Figure 16-9. It will then appear as shown in Figure 16-10.

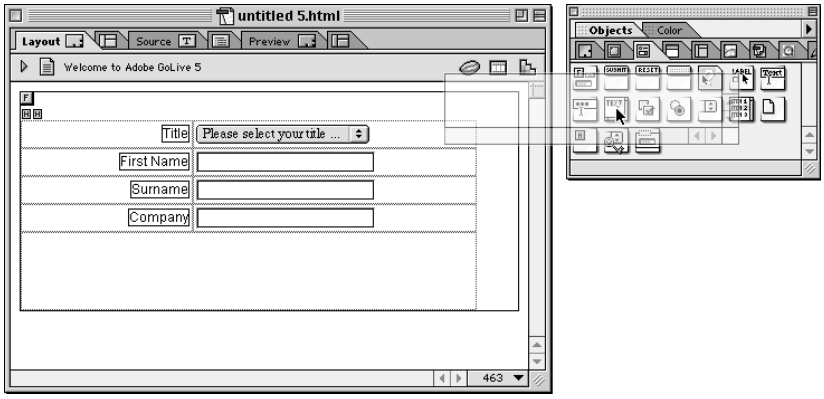


Figure 16-9: The Text Area icon as it is just starting to be dragged from the Objects palette to the page

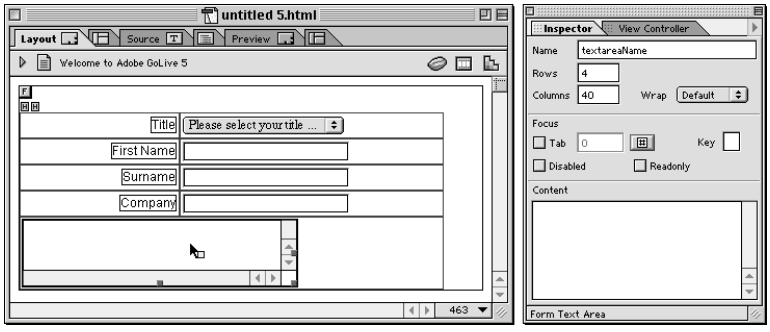


Figure 16-10: A new Text Area as it appears in a borderless cell, and its corresponding Inspector

2. By default the field name area of the Form Text Area Inspector says textareaName. Select this temporary name and enter your own field name unique to the current form.

   Remember, field names often have to exactly match names in a database or CGI.

   If you have two field names (regardless of kind), they'll both feed their contents or data into the same field within the database or CGI.

3. *(Optional)* If you want the field to contain content by default, enter that content in the Content field in the Inspector. This text will appear in the browser but can be deleted and typed over by the user. When the user tabs into the field the pre-entered text will be selected, and if the user clicks into this field, the cursor appears where he or she clicks. (See Figure 16-11.)

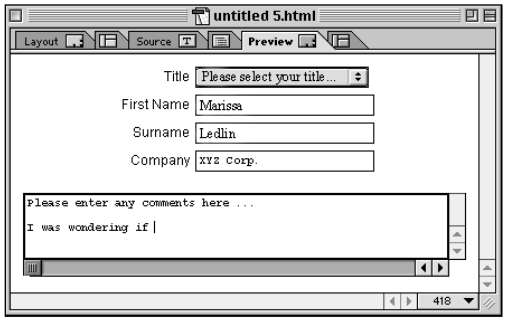


Figure 16-11: A Text Area in Preview mode demonstrates text being entered. (The user kept the original content, but it can also be selected and typed over.)

4. The text area is four rows high by default, providing four rows of visible text for the user to see without scrolling. To add or delete visible rows enter a new number in the Rows field. This has no effect on the amount of text the user may enter.

5. The text area is 40 columns wide by default, which is actually 40 characters. You can make this area wider or narrower by entering a new number in the Columns field. This width has no effect on the amount of text the user may enter.

6. In the Wrap (Mac) pop-up menu, choose an option for how (or if) text will wrap when a user enters text:

   - *Default* adopts the browser's behavior.
   - *Off* turns text wrap off, which means as the user types the text just keeps going and going and going. The user needs to scroll horizontally to see what he or she wrote. The user has to press Return to see the text appear at the visible part of the text area and may press Return often to read what he or she is writing. However, that actually creates a new paragraph.
   - *Virtual* provides the user with the effect of text wrap, so the words are visible between the left and right edges of the Text Area field. When the field's content is sent to the server, though, it is sent as one continuous line. That way the database, e-mail, or other document that receives the text can treat it as appropriate for that application or document.

- *Physical* not only shows the user the effect of text wrap, but also adds a line break to physically cut each line at the end of the column. The words entered are sent to the server wrapping exactly the way they appear in the form. (Depending on the platform the user is on, the text you receive may have end of line characters such as "=" that you would have to clean up.)

Regardless of which wrapping behavior you choose, the user must also press Return to create a new paragraph.

Adding checkboxes to your form

A checkbox enables you to have the user choose from preselected information. When the user leaves the checkbox empty, no information is sent upon submission. When the box is checked, the value that you assign to the checkbox is transferred via the browser. Checkboxes are toggles; each time the user clicks the checkbox it toggles between on and off.

Checkboxes are your best choice when you want the user to be able to choose more than one option (for example, when a form says, "select as many as desired"). Another reason to use a checkbox is when you want the user to feel in control of a choice, such as when electing to have you send him (or her) something. In this case, a yes or no set of radio buttons, or a pop-up menu of choices also works, but the checkbox is clearer.

To add checkboxes to your form, follow these steps:

1. Drag the Checkbox icon from the Objects palette into your form.

   You'll drag one checkbox for each option you want to offer.

2. Enter a name for the checkbox.

   As usual with a form, this is not an arbitrary name; it commonly must match the name of its corresponding field in the database or CGI from which it feeds or receives data.

   The name of the checkbox must be unique from other fields in the same form but not unique from the other checkboxes in its set that provide the other options. If several checkboxes provide value options for the same field within a database, you will create several checkboxes all with the same Name.

3. Enter a value. If the checkbox is selected, the value will be sent when the user submits the form.

   If the form is working in conjunction with a database, the corresponding field within the database already has values assigned to it. The values on your form need to match the database. Checkbox data can go into a corresponding set of checkboxes in the database or into text fields, not into other types of fields. For instance, if your database field only accepts entries eight characters in length, make sure that the corresponding field on your form only accepts eight characters too.

4. *(Optional)* If you want this checkbox to be checked by default when the form first appears to the user, check the Selected option. See the example of a completed checkbox in Figure 16-12.

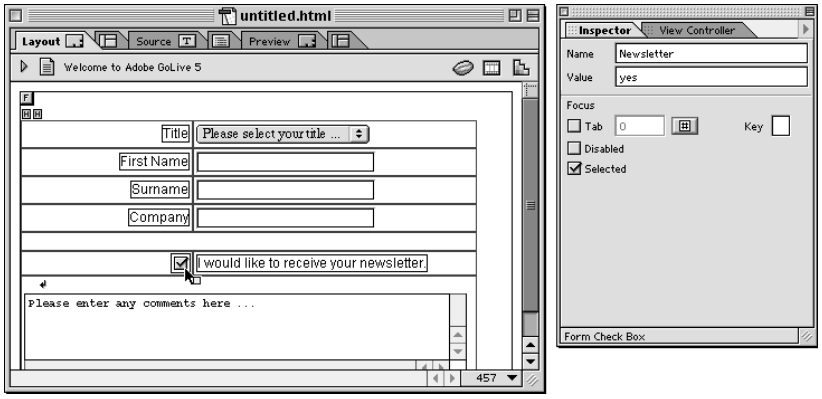


Figure 16-12: A checkbox for which *yes* will be sent to a field called Newsletter. It is preselected when the page loads.

If several checkboxes will provide value options for the same field within a Form, repeat the preceding steps to create the other checkboxes, all with the same Name. You can save time by copying the button instead of starting with a new button. Select the first button, which is already set up, copy it, and then paste as many new buttons as are needed. This way you ensure that the group name remains identical. (On the Mac you can Option-drag the original button to copy it.)

As with all form elements, you need to add a label by each option to let users know what they're selecting. For example, back in Figure 16-12, text tells the user that by checking this checkbox they're requesting to receive a newsletter. You have two ways to add labels to your form. You can simply type descriptive text, or you can use a feature new to HTML — the label tag. See the "Adding labels to form fields" section.

Adding radio buttons to your form

Radio buttons present the user with multiple choices for a specific field but force the user to *choose just one.* For example, you might ask "Do you have a credit card?" and have two radio buttons — Yes and No — that both apply to the same question field. The user can select either yes or no; when one is selected the other is automatically deselected. When radio buttons report to the same field, they are in a "group."

An empty radio button sends nothing to the server. A selected radio button sends the value that you assign to it.

Checkboxes versus Radio Buttons

Checkboxes differ from radio buttons in that more than one can be selected simultaneously by the user, meaning that the data sent from the form for this field can have more than one value. This is dealt with by sending the values as a comma-delimited list. For example, if you have a form field called "Hobbies" and someone selects the checkboxes for Bushwalking, Cycling, and Reading, the values would be sent as something similar to this:

```
Hobbies=Bushwalking,Cycling,Reading
```

You don't have to worry about the code that sends this information; the browser does it for you. The values are set using the value associated with the checkbox.

Follow these steps to add a radio button to a form:

1. Drag the Radio Button icon from the Objects palette into your form.
2. Enter a name for the radio button group; in the Form Radio Button Inspector the name goes into the Group field. If the user is asked to choose between two radio buttons, then both belong to the same field within the database so both have the same group name (as shown in Figure 16-13).

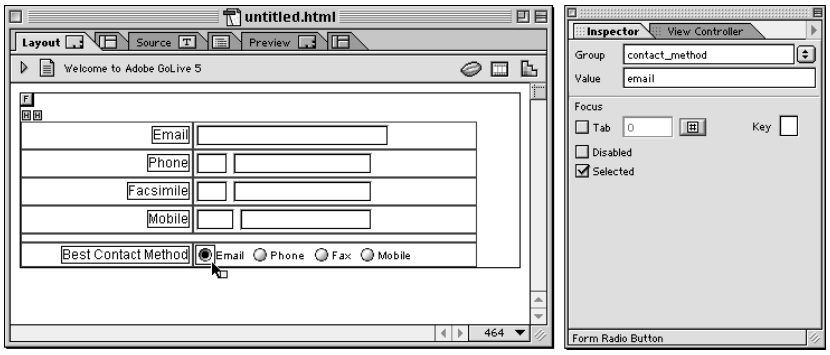


Figure 16-13: Four radio buttons, all with the same name, determine that the users will have to choose one response from four possibilities.

As usual with a form, this is not an arbitrary name; it commonly must match the name of its corresponding field in the database or CGI that it feeds or received data from.

3. Enter a value. The value is the actual data that is sent when the user submits the form. This is the same as with a checkbox.

   If the form is working in conjunction with a database, the corresponding field within the database may require a particular type of value (such as an integer, a ten-letter word, a Boolean, or a date) assigned to it, so these values

must match the database. The field into which the radio button value goes can take the form of a corresponding radio button or a text field, but not other types of fields.

4. *(Optional)* To have this radio button selected by default when the form first appears to the user, check the Selected option. (Only one button in the set can be selected, so only one can be preselected.)

5. Repeat these steps to add other radio buttons to the group. Each one will have the same group name but different values. You must have at least two radio buttons to offer any option. Otherwise, you have no way to deselect a radio button.

   You can save time by copying the button instead of starting with a new button. Select the first button, which is already set up, copy it, and then paste as many new buttons as are needed. This way you ensure that the group name remains identical. (On the Mac you can Option-drag the original button to copy it.)

You need to add a label by each option so users know what they're selecting. For example, you need to tell users that one button is for choosing Yes and one is for No. You have two ways to add labels to your form. You can simply type descriptive text, or you can use a feature new to HTML — the label tag. See "Adding labels to forms" earlier in this chapter for details.

Creating a pop-up menu

A pop-up menu is an alternative to radio buttons. The user must choose only one of the choices you present. However, depending on how many choices exist, a pop-up menu may be a great space-saver. A large group of radio buttons can not only take up a lot of room, but also look confusing. A pop-up presents a simple, small, clean look as shown in Figure 16-14. Beware of creating too long a list, though.

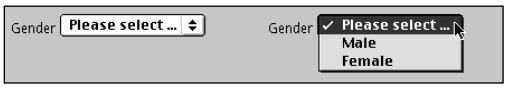


Figure 16-14: A pop-up menu as it looks normally and the same pop-up being clicked

With the other field types you have to type a label such as the Yes or No for each option offered by a radio button and place it beside the option. With a pop-up list, that option label is built into the pop-up. The choice that pops up in the list is the label that guides the user's choice. As with checkboxes and radio buttons, the actual value passed from the form does not have to be the same exact text as the label. The actual data the form sends can be something totally different. This enables you to have a friendly looking list. For example, the label can say Monday, Tuesday, and so on, while the values can be Mon, Tue, and so on — because in a database it's best to keep values short.

To create a Pop-up menu, follow these steps:

1. Drag the Pop-up icon from the Objects palette into place on your form.
2. In the Form Pop-up Inspector's Name field, enter the name for the field as defined by your CGI.
3. Click the first row under Label so the first label and value appear in the entry fields below the choice list.

   This is the first step to entering the label and value for the first choice in your menu.
4. In the choice entry area type a new label for your first choice. If the default label is preselected, just begin typing. If it's not, then select and type over this default text. This label is the first item that will appear in the pop-up. Click the Return button or press Enter/Return to confirm the change or press Tab to confirm the change automatically and also move to the Value field.
5. In the field next to the label entry area, replace the default value with your own value, as in Figure 16-15. As with the other form fields, this value must match those that exist in the corresponding field in the database. Click the Return button or press Enter/Return to confirm the change.

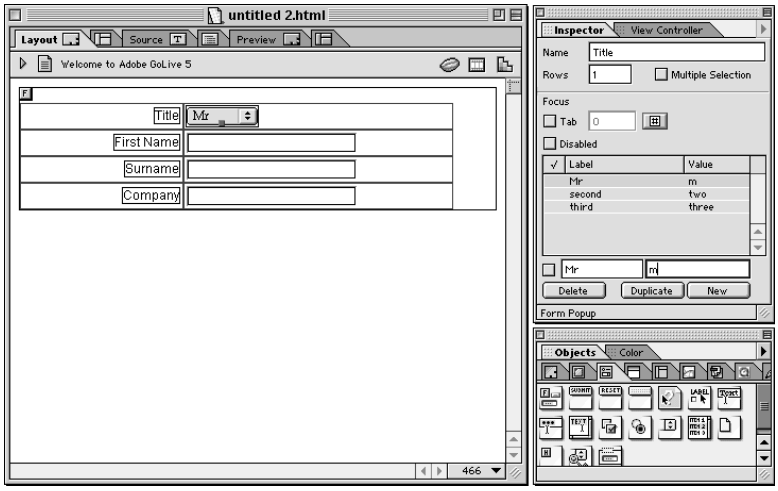


Figure 16-15: A choice's value being set after the Label is entered

When you click another item on the choice list or add a new item, the change in the value field is automatically confirmed.

6. Do the same for the next two default choices.

   If you have fewer than three choices, chances are you wouldn't use a menu, but in case you want to remove one of the defaults, click it in the list and then click Delete.

7. To add another choice click New, and then replace the default text that comes into the Label and Value fields. (The defaults are the words *label* and *value*.)

To remove any choice, select the choice in the list and then click the Delete button.

By default the first choice in the list appears in the Pop-up. However, you can set any other choice as the preselection. For example, if you have an alphabetical list of many countries, you may want one particular country to appear as the default. To preselect a choice, double-click the box to the left of the label. A checkmark then appears under the checkmark column in the choice list. (Alternately, you can check the box to the left of the entry boxes when you enter the label and value for any choice.)

It's often helpful to the user to make the selected choice a helpful hint that they need to select something from the pop-up list. Using a label like "Please select . . ." or "Which size would you like?" helps give the user a visual reminder that they need to do something with the field. This becomes especially important now that browsers are letting users auto-fill forms using preselected values. It makes it very easy for a visitor to your site to miss a field that they are supposed to enter a value into.

In providing these directions I left out the Rows and Multiple Selection options. That's because adding more rows to a pop-up turns it into a *list.* Enabling Multiple Selections for a one-row pop-up prevents the pop-up from functioning at all. When working directly in HTML, adding the Rows attribute creates the change, and so it does here. By using the same Inspector interface for both a pop-up and a list, GoLive enables you to switch your field from one to another without having to re-enter the values.

Adding a list box to your form

The list is actually a variant of the pop-up menu. Unlike the pop-up menu, though, users can choose multiple responses from a list. You might say it's a cross between a pop-up menu and a group of checkboxes.

To create a list, follow the same directions as for creating a pop-up menu, with these two additions that can be performed at any point.

- In the Rows field, enter the number of rows that you want the user to see on the field. The more that are in view, the longer your list is. If you have six objects and three rows, scrollbars appear. Figure 16-16 demonstrates.

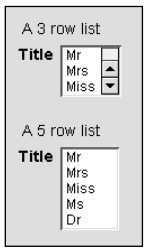


Figure 16-16: Both lists here contain five choices. The top list has fewer rows than choices, however, so scrollbars enable the user to see the extra choices.

- If you want the user to be able to select more than one choice, check the box by the Multiple Selection option.

  If you enable Multiple Selection, be sure to include written instructions telling users that they can select as many as they want — and tell them how. To select more than one noncontiguous choice they need to press ⌘ (Mac) or Ctrl (Windows) as they click each choice. Or, they can press Shift as they click to select multiple items that appear one after another on the list. Shift-clicking one option and then another selects all options in between the two.

  If you enable multiple selections you can have more than one choice selected by default.

HTML doesn't provide the capability to limit the number of items that can be selected in the list, or "multiple-select," select box. However, you can use a JavaScript to provide that control. An example of code, written by David Shadovitz for you to adapt for yourself, is located in the JavaScript folder on the book's CD-ROM. Additionally, OUTactions has developed a GoLive Action just for this book that provides this function for you. You can find this Action in the GoLive Stuff folder on the CD-ROM.

If you decide you prefer a list to be a pop-up menu instead, just change the number in the Rows field to 1 and turn off Multiple Selection.

Setting User Navigation Within a Form

HTML 4 introduced some new ways to enable users to navigate through forms. One of the key additions was the concept of focus. When a visitor to your form clicks or tabs to a particular field, that field is said to have the focus. This information can be used in a number of ways, including setting the tab order of your form (which field should have focus next), setting key commands for particular fields, and other useful behaviors.

The focus area of the Inspector enables you to determine how your users will navigate through your form. Simple things like setting a logical tab order can help make it easier for your visitor to provide you with information or make requests.

The functions described in this section are new to HTML 4, so they won't work with pre-4.0 browsers, though they can be used without causing problems for the older browsers.

Setting the tab order of your fields

All browsers that read forms enable the user to tab through the forms, but they don't control the order of the tabbing and only provide for tabbing into text fields. HTML 4 enhances tab control, letting the designer set the tab order and providing

for tabbing to select nontext fields. The Tab control in the Form Inspector is for setting the tabs as recognized by HTML 4 browsers that incorporate this capability. According GoLive's Web database, this feature, called `tabindex`, is Internet Explorer only.

When the standard is fully recognized and implemented by browsers, users will be able to tab into fields such as radio buttons and then press Enter/Return to select that choice. Tabbing in the order you specify will also be possible. For now, even though many browsers can't see your tab order (and the older browsers never will, of course), no harm or conflict is caused by setting a custom tab order.

In case you're wondering how tabs work in noncompliant browsers: In Netscape 4.6 browsers for the Macintosh, you can tab between text fields, but once you land in a text area you end up tabbing through the text area, jumping about five spaces per tab as if you are in a text document. In many browsers, the Tab key also takes you to the URL entry bar. After it cycles through the URL entry it doesn't always find its way to all of your other text fields again.

To set a custom tab order for your form, follow these steps:

1. Select the field that you want to be first in the tab order.
2. In this field's Inspector, press the # button next to the Tab field. Alternatively you can choose Special ➪ Start Tabulator Indexing.

   Little numbered yellow boxes appear on each field that can be tabbed into. The cursor also gains a #.
3. Click each field in the order you wish the tabbing to proceed.

   As you click each field, the tab box shows its tab number. The same number is also recorded in the tab field in the item's Inspector. This number is the `tabindex`.
4. After you've assigned a tab order to all of the fields you want included, press the # button in the Inspector again or choose Special ➪ Stop Tabulator Indexing.

If you have multiple forms on your page, the other forms are also included in the tab numbering. Each form is not numbered separately. However, once a user clicks in a cell, the tabs proceed in order, so the user can still tab through each form. The focus will travel to the next form after the last cell of the current form.

After you've assigned a tab order, when the # button is selected, new tab numbers are assigned in sequence. If you click the same field multiple times, its number increments.

To edit the tab order, deselect the # button to turn off numbering, and then select the cell you want to pick up the numbering from. Turn tabulator indexing on again and start clicking. The numbering starts incrementing from the number of the currently selected field.

To redo the entire tab order, select the field you want as number one and enter 1 into the Inspector. After that you can press the # button and click each field to assign new numbers.

You can also click each field and enter the tab order number for each one in the tab field in the Inspector. This method has more steps, so you probably won't choose it initially. However, it may come in handy for editing the tab order.

Assigning keyboard keys

The key capability is designed by HTML 4.0 to enable you, the designer, to assign a keyboard key to a field so the user can jump directly to a specific field. This is a new idea and not widely implemented yet, though. Currently only newer versions of Internet Explorer support this function. The key command uses the Alt modifier in Windows browsers and the Control modifier in Mac browsers, meaning that if you set the Key to G, then users would press Control-G (on a Mac) or Alt+G (on Windows) to reach that field.

Be careful when using this feature. So many keys are already assigned to computers that the key you choose may already do something else on the user's computer, and therefore may perform an unexpected procedure when the user follows your instructions.

To assign a keyboard key to a field, select the field that will have the key action. Then, in the Key field, type the key combination.

Disabling form elements

When you disable a form element it appears in the browser, but it is grayed out and unusable. The power of this capability, new to HTML 4, is to use it along with a script that runs a quick verification of your form and only enables the field when certain criteria are met. For example, you can have a button that only becomes available after all the fields are properly filled in, or you can have certain questions come alive depending on the way another question is answered.

To disable a form element you simply select the element and check the Disabled option. That option then appears grayed out on the form. To enable the field again, you need to write a script to do the verification and set the field as enabled. (This would perhaps be a JavaScript that is placed in the Header or Body of the page, not any special script that attaches to the actual element.)

This feature only works in Internet Explorer, though, not Netscape browsers (at least as of the time I write this). However, JavaScript provides a function that is even more useful and elegant; with JavaScript you can actually hide a field until certain criteria are met, revealing only fields (or buttons) that are pertinent to the responses a user provides or to the user's browser. (See Chapter 22 for more on JavaScript.)

Setting a field to read-only

You can set a field to read-only so the user can see the information but not change it. This can be handy when reporting information back to a user. To make a field read-only, simply select the field and check the option in the field's Inspector. A password or username is a good candidate for reading only; that way it serves as a reminder but can't be changed.

Placing fieldsets in a form

Fieldsets, literally sets of fields, are one of HTML 4's new accessibility features. The fieldset groups a number of fields within its boundary. It doesn't do anything special to contain or align the fields. It just makes it easier for speech synthesizers and text readers.

The legend that goes along with the fieldset helps make the group of fields understandable.

In the browser, a user sees a light gray line border around the cells that you place within the set's area.

However, fieldsets don't work in current Netscape browsers (they do work in Netscape 6 and Mozilla) or in Opera. They work in Internet Explorer 4 (except on a grid) and in iCab (an alternative Mac browser).

Follow these steps to place a fieldset on your form:

1. Drag the Fieldset icon to the form.

   It comes into the page as wide as your window and resizes as you resize the page.

2. By default a fieldset displays a legend — a textual description within the set's box.

   - To customize your legend with your own words, click the word Legend on the page and then type as you please.
   - If you decide you don't want a legend, uncheck Use Legend.

3. Choose a location for the Legend from the Alignment pop-up. The default alignment, called Default, is at the top left unless the browser happens to specify something different.

   Left, Right, and Center refer to the top as well as their respective horizontal alignment. An option also exists to set the alignment to the Bottom of the fieldset (you cannot set the horizontal alignment of a Bottom-aligned legend).

4. Move your fields into the set's boundary area and arrange as desired.

Adding Buttons to a Form

A form isn't much of a form until users can submit it to someone someplace. HTML provides for three types of buttons.

A Submit button submits your form, a Reset button removes all user input and returns the form to the way it looked when first sent to the browser, and a Push button attaches to a script. Each of these button types can be dragged from the Objects palette. If you place the wrong button type, you can change the type within the Inspector.

A button can optionally have an image associated with it. Each of the following button types is discussed further in the following sections:

- ✦ Submit buttons
- ✦ Normal buttons
- ✦ Reset buttons
- ✦ Universal buttons

Submit buttons

This button enables your visitors to send the entered information or request to the server. Here's how to use it:

1. Drag the button into place on your form, as in Figure 16-17.

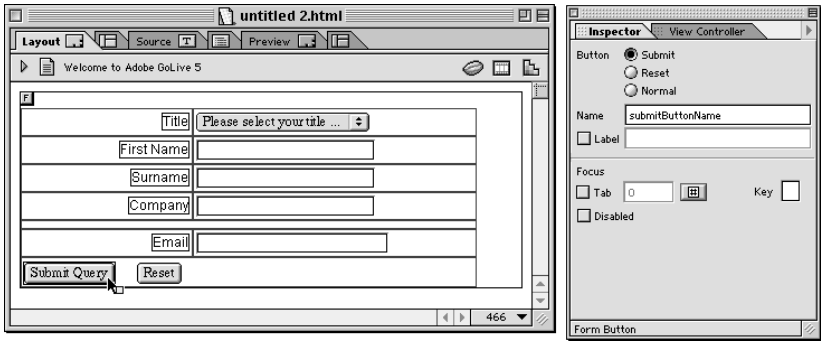


Figure 16-17: The Submit button and its Inspector

Typically a button is placed at the end of the form so users won't push it until after the form's contents are complete.

2. In the Name field enter a unique name so you can identify the button later.

   A name is not required unless your CGI specifically uses a set of names as part of its instructions. However, if you have more than one Submit button on a form it will help you tell which is which. The name is included in the information passed to the server, so keep the name short.

3. *(Optional)* The Submit button says Submit Query by default. This label is merely cosmetic; it's a guide for users so they know what the button does. You can have your button say anything you want. For example, "Submit Application" or "Send your idea now." To change the words on the button, check the Label option in the Inspector and enter a new label. The button will grow to the label you give it.

It will help if you understand the HTML behind Submit buttons and know how GoLive's Inspector relates to the HTML.

The HTML for a Submit button looks like this by default:

```
<input type="submit" name="submitButtonName">
```

When you customize the label within the Inspector, you add another attribute, the *value,* as shown here:

```
<input type="submit" name="submitButtonName" value="Submit
Application">
```

The elements in this line of HTML are as follows:

- The *input type* is determined by the type of button it is. As long as the Submit option is active in the Inspector, the input type is submit. If you are instructed to change this type, choose the Normal button.
- The *name* part of the HTML is obtained directly from the Name field in the Inspector.
- The *value* in the HTML is entered and changed through the Label field of the Inspector. Note that unlike other form objects where the value is the data that gets passed to the server, with a button the value is merely the name that appears on the button to guide users. The value is optional.

Normal buttons

If instead of a button of input type Submit you need a button that is of input type Button, add a regular Submit button to your page and then change it to normal in the Inspector (by choosing the radio button for Normal).

This is what the HTML looks like with the normal option selected for a button:

```
<input type="button" name="submitButtonName">
```

Notice that the input type is button instead of submit. The button also says "Button." If you add a label to change the face of the button, you see something like the following code, which creates a button that says "I'm normal" on it.

```
<input type="button" name="submitButtonName" value="I'm normal">
```

Reset buttons

A reset button isn't necessary, but it can be a nice convenience for the user. When the user presses it, all of the information the user may have entered on your form is cleared. It can be helpful if a user is entering information for more than one person. It also clears a form quickly in case the user is entering a contest when his or her boss passes by. Reset doesn't send anything to the server; it does its thing completely within the browser.

Follow these steps to add a Reset button to your form:

1. Drag the Reset button into place on your form from the Forms tab of the Objects palette. Typically this button is placed at the end of the form along with the Submit button.
2. *(Optional)* By default the Reset button says Reset, but you can make it say anything you'd like. To change the words on the button, check the Label option in the Inspector and enter a new Label. The button grows to accommodate whatever you type.

You don't have anything to adjust name-wise. The Inspector is the same, but the Name field is inactive. The HTML for a Reset button is `<input type="reset">`.

Universal buttons

As this button is new to HTML 4, it is only recognized by HTML-4-compatible browsers. (GoLive's Web Settings lists it as Internet Explorer only.) It works just like any other button but is more customizable and the Inspector makes it easy to attach a script to it.

To add a universal button to your form, follow these steps:

1. Drag the Button icon from the Forms tab of the Objects palette.
2. The button says Button by default. Click the word Button on the face of the button and replace this text by typing the words you would like to have appear on the face of the button, as shown in Figure 16-18. You can adjust the text attributes too (such as font, size, color, and so on). This button face also accepts images and HTML, so you can go to town customizing its look.

3. Name the button for your own reference. The names Submit and Reset cannot be used as they are already assigned.
4. In the value field, enter the Action or value to be passed to the agent (CGI) via the server.

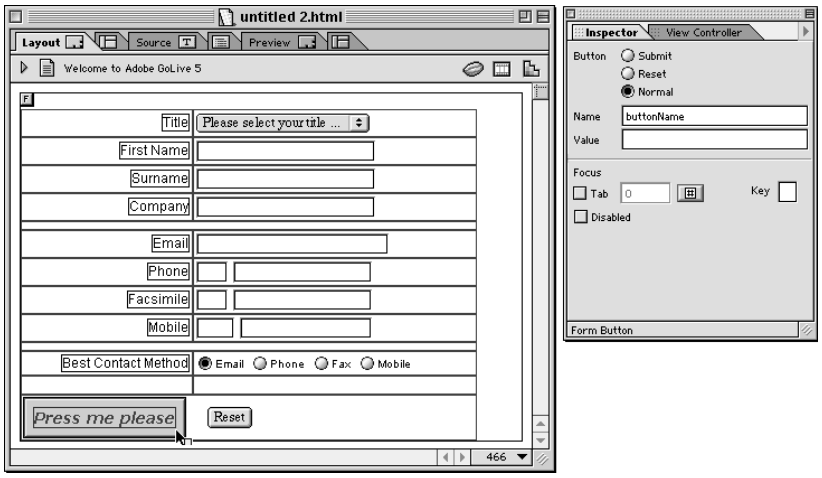


Figure 16-18: A Custom button that now says "Press me please." The name and value are unedited.

Image placeholders

Images can be used as buttons. Sometimes it's a nice touch to have the user click an icon instead of reading words.

To use an image as a button, follow these steps:

1. Drag the Input Image icon from the Forms tab of the Objects palette.

   When an image is used as a button, the Form Input Image Inspector is similar to the Inspector for other images.
2. Assign the desired image to the button.
   - Point and Shoot from the Basic tab of the Inspector to the image in the Files tab of the Site Window.
   - Browse to the image.
   - Drag an image directly from the hard drive, but remember that if the image is not a GIF or JPEG, GoLive will bring up the Save for Web window. You will then be able to set all of the attributes for the imported image and select where you would like to save the image. (See Chapter 10 for more on GoLive's Save for Web feature.)

The location of this image, as in the Source field of the Basic tab, is included in the HTML as the input source, written as follows:

```
<INPUT type="image" src="path/to/image">
```

3. Switch to the Special tab of the Inspector and enter a name to identify the image. As with the name of a plain button, this is not required unless your CGI specifically uses the name attribute as part of its communication. However, it will help you identity the button later, in case you have several buttons on your page.

   Keep the IsForm box checked, as it is by default. This tells the image it is part of a form.

4. As with all images on the Web, entering Alt text to briefly describe the image helps people whose browser reads for them or prints out your page in Braille.

You have no need to set the border image to zero (the way you might for other graphics used as buttons), as no border exists around a button image.

The HTML for a button image is `<input type="image" src="images/yourgif.gif">`. After you add a name it looks like this: `<input type="image" src="media/yourgif.gif" name="buttonname">`.

Parameters will be sent using the name to contain the coordinates of the point clicked in the image: buttoname.x and buttonname.y.

Advanced Form Fields

The form fields covered in this section are not standard elements. Consider their use carefully, as they will not be supported in every browser.

Placing a key generator on a form

If your site supports encryption, your user can choose from any encryption algorithm you list on your page. This is not a standard of HTML; it is a Netscape-only feature. Ask your site host about it before you try to set it up.

To place a key generator on your form, follow these steps:

1. Drag the Key icon to the form.
2. In the Name field, enter what your site host tells you to enter.
3. In the Challenge field enter the security level, as shown in Figure 16-19.

To facilitate discussion between you and your site host, this is the HTML that your host will be instructing you about. The items in the quotes are the words you change within the Inspector. The words before the equals sign are the codes the site host will recognize.

```
<keygen name="keygenName" challenge="publickey">
```

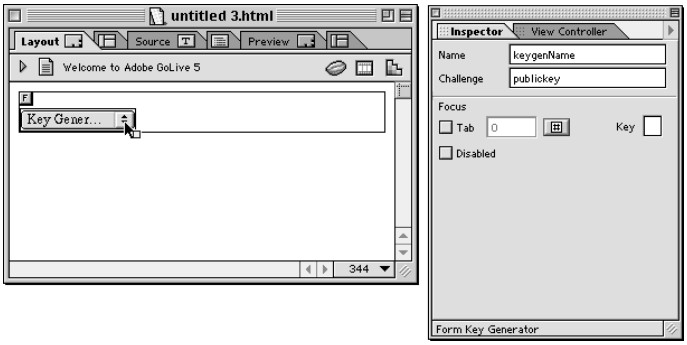


Figure 16-19: The encryption key selection field as it is being set up

Placing a file browser on a form

The File Browser is a visual interface that, along with a CGI, enables a user to browse his or her hard drive and locate a file to upload. This requires a CGI located on your server and an upload destination.

Follow these steps to place a File Browser on your form:

1. Drag the File Browser icon into place on the form.
2. In the Name field, select the default fileGetterName and place it with the path to the CGI that will do the uploading. If the CGI is within your site, you know its path. If the CGI is on the Web server, you may have to check with your site host to learn this.
3. In the Visible field enter the desired number of characters you'd like the enter field to contain.

   This field is where the name of the file to be uploaded appears after the user browses to the file and selects it. The longer this field is, the more of the file's name the user will be able to see and verify.

Validating Your Form Data

Now that you have a form set up to gather the information that you need to collect, you need to make sure that people are actually entering the right things into the right fields. It isn't going to be very useful to receive an e-mail where necessary fields are missing or wrong (as shown in Figure 16-20). This may seem like an over-the-top example, but believe me, it's not!

You have a number of different ways to validate the entries that people make in your forms before the information gets to you or gets put into a database for storage. You can check things such as if a user has entered a value in a field, whether e-mail addresses or credit card numbers are in the correct format, or if a user has entered something of a specific length or type (like a four-digit number or six-letter word).

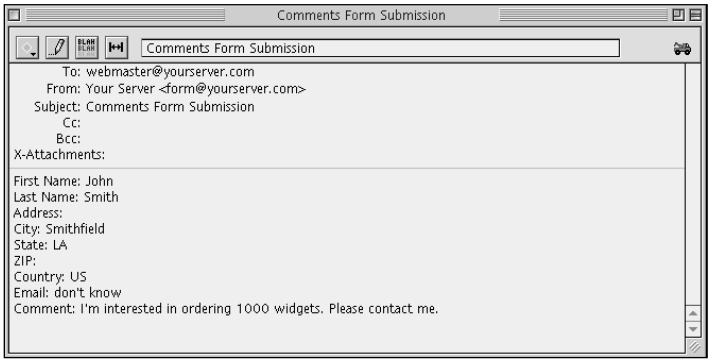


Figure 16-20: The information from this form doesn't help you because the entries weren't validated before the form was sent.

The two main different methods are client-side validation and server-side validation. Client-side validation occurs in the user's browser, either when they are entering the information (field-level validation) or when they press the Submit button (form-level validation). You can use GoLive Actions or custom JavaScript to achieve this within GoLive. Server-side validation occurs after the form has been sent to the server for processing. This is done by using a server-side language (such as ASP, ColdFusion, or PHP) to analyze the form entries and then take action from there. For instance, if the fields are correct, an e-mail should be sent or the values should be put in a database, and an error message should be returned to the user if a problem exists.

Field-level validation using the FieldValidator Action

An Action is now available for GoLive 5 called the FieldValidator Action, which enables you to do field-level validation on a form. (See Figure 16-21.) It doesn't ship with GoLive itself, but it is available through Adobe Online in the Help menu (choose Help ➪ Adobe Online and then click the Extras button — you need to be connected to the Internet to do this).

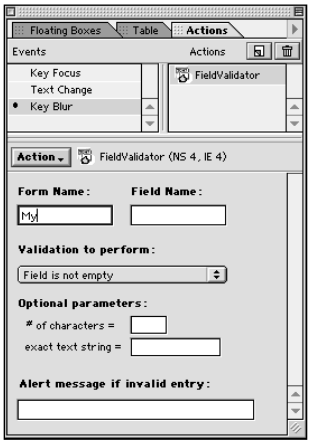


Figure 16-21: The FieldValidator Action as shown in the Actions palette

To set a field to use this Action, follow these steps:

1. Select the field you wish to validate.
2. Open the Actions palette, select Key Blur in the Events listing, and click the New Action button.
3. Choose Action ➪ ActionsPlus ➪ FieldValidator as your Action.
4. Set the Form Name to the name of the form that this field is in and the Field Name to the name of the selected field.
5. In the Validation to Perform pop-up menu, choose the type of validation you need to perform.
6. Set any optional parameters for the type you have chosen. For instance, if you have chosen "Field has this many characters," type in the number of characters the entry has to be, and if you chose "Field = exact text string," enter the text that the field must contain.
7. For cases when an error occurs, you can optionally enter the message you would like the user to see in the "Alert message if invalid entry" field.

Because this Action uses the Key Blur event to trigger the validation, you cannot use this Action on adjacent fields in your form. (This means there must be a field to tab into between fields using FieldValidator.) The problem has to do with the way that the Action is executed. It checks the form whenever you tab out of a field (which immediately puts the cursor in the next field), and then automatically pops you back to the same field if the entry is wrong. Unfortunately, if the next field has a similar Action applied, when it pops back to the first field, the Action executes on the second field also and you fall into a deadly loop. You can apply the Action as a "Text Change" event, which will solve this problem, but then it doesn't prevent the user from leaving the field blank.

Form-level validation using the VerifyForm Action

This Action by Walter Blady can be used to verify that there has been an entry in up to five text fields and one e-mail field on a standard HTML form. Each field can be set up to check for an exact match or any entered string. The E-mail field is used to check an e-mail address for proper formatting. If a mismatch exists, the form is not submitted and a dialog box warns the users to check their entries. It is available for purchase from Walter Blady's Web site at `www.wblady.com/actions.html`.

To add Walter Blady's field verification to your form, follow these steps:

1. Select the Actions tab and then the Mouse Click Event. Click the New Action button and select the VerifyForm.Action from the Action selection menu. The Actions palette will now look like Figure 16-22.

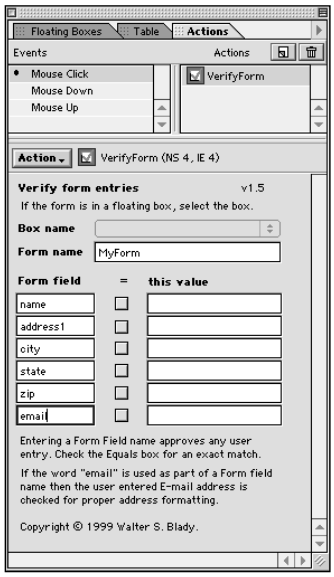


Figure 16-22: The VerifyForm Action as shown in the Actions palette

2. If your form is in a floating box, select the box name from the Box name pull-down menu.
3. Enter the name of your form in the Form name field.
4. Enter the field names that you want to verify in the Form field boxes. These names must be exactly the same as the ones you used in your form. All names are case sensitive.
5. The corresponding fields on your form will be checked for an entry, and any entry will be considered valid. If you want the user to enter an exact value, check the Equals (=) box opposite the Form field and then enter the value that the user must match in the "this value" box.

Confirming a Submission

One of the silliest things you can do is have the user submit a form and then leave the user hanging. Don't forget to confirm the submission and thank your user. While you're at it, don't miss the opportunity to send the user someplace useful.

Any good CGI will have a command that enables you to have a confirmation page returned to the user after successful submission (and also enable you to return a page noting an error). You typically send this instruction within a hidden tag. The name would be the command and the variable would be the name of the page to be returned.

If your form is simply sending an e-mail and no CGI is used, you can still send a confirmation page. To do this, you add a "redirect" JavaScript Action to the form. However, this is one case where you cannot use one of GoLive's prewritten Actions. You will have to do this manually in Source mode.

In Layout mode, select the Form tag and then switch to the Source tab. Because the form tag was selected in Layout mode, the form Action is selected so you can easily locate it. The form Action will look something like this, with the real e-mail address, of course:

```
<form name="mailform" action="mailto:mail@server.com"
method="post">
```

Place your cursor at the end of this code, immediately inside the closing bracket. Then type this:

```
onSubmit="window.location='yourcofirmationpage.html'"
```

Your entire tag will look like this (with the real name of the confirmation page, of course):

```
<form name="mailform" action="mailto:mail@server.com"
method="post"
onSubmit="window.location='yourcofirmationpage.html'">
```

If you enter a field name that includes the word "e-mail" as part of the field name, VerifyForm will perform a basic format check of the e-mail address entered by the user. If the format doesn't conform to the format necessary for a correct e-mail address, the "Form entry error" dialog box will appear.

Server-side validation

Sometimes doing validation in the client's browser doesn't provide the necessary failsafes that you need for either data integrity or security reasons. One situation that could cause this is if a user of your site has JavaScript turned off — none of your validation scripts would be triggered, and whatever they entered, or didn't enter, would be sent to the form processor to be put into a database or sent to you.

In cases where this is critical, it may be best to use server-side validation using something like ASP, PHP, or ColdFusion to check the results of the form after it has

been submitted, but before the data is processed. You could then either process the data if it is correct, or represent the form (or an error page) to the user if problems occur. This approach is also very important where security or the origin of the data is critical, because you can check information like the visitor's IP address, whether the data came from your form, or whether someone had already filled out the form (important in the case of applications like polls).

Form Troubleshooting Issues

If you find a green bug symbol next to a page in the Site Window that contains a form, it may be the address of the CGI — the Form Action. This isn't a problematic bug, as you know the Action is valid. (You know a form Action is valid when the form behaves properly.) The bug appears because GoLive simply can't find this file. When you export the site, the server will be able to locate the CGI file and that's all that counts. However, having this cause a bug prevents you from being alerted to other bugs that may occur.

You can put an end to this bug by telling GoLive not to report this file (whenever it is used). All you need to do is enter a part of the Form Action's name in the URL filter. Then, any time GoLive sees a URL that contains those characters, it will know not to flag that URL as an error. For example, if you're storing your CGIs in a folder called cgi-bin, as is common, you can enter the part of the URL that contains that part of the path so all your CGIs are covered at once.

Follow these steps to set GoLive to ignore the unknown URL it is flagging:

1. Choose Preferences ➪ Edit, or use the shortcut ⌘-Y (on Mac) or Ctrl+Y (on Windows).
2. Open the General Preferences by clicking the triangle by General.
3. Select URL Handling.
4. Under URL Filter, click the New button.

   The words New Filter appear preselected in the field above the button.
5. Type the name of the file/database your CGI accesses (the Action address) that is causing the bug, as follows:
   - If you are storing all of your CGIs in a folder called cgi-bin, as is common, you can enter **cgi-bin** as the filter. That way all files stored within that folder will not be marked as creating a bug.
   - You may not need to enter the entire address. In the case of Figure 16-23, all I typed is the common part that ends with the dollar sign. That way other pages in the site that also access this CGI won't cause bugs, even though they use an ending such as $update or $delete.
6. Click OK.

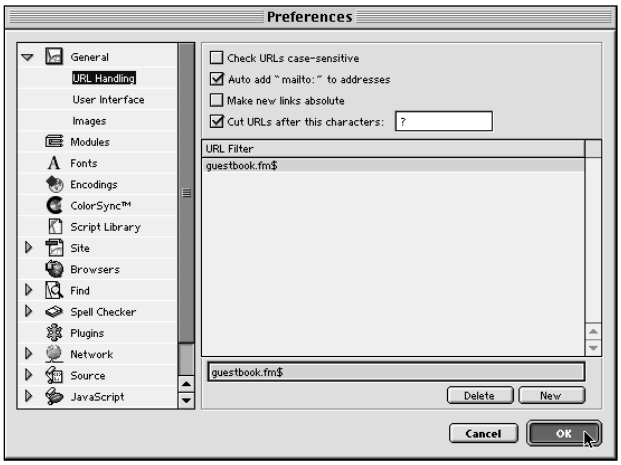


Figure 16-23: Entering a new URL filter

7. GoLive asks you if you want to update all open sites based on this new rule. Click OK unless you have multiple sites open and you have some reason not to change one of them. In that case, close the excluded site, and then repeat this procedure, accepting the change next time.

You may find that other parts of your form cause GoLive to report errors. For example, if you're using PHP, you'll find that GoLive thinks that any link containing "`<? . . . ?>`" is bad. This type of error report can also be avoided with a URL filter. However, if you add a filter for "`<?`" or "`?>`" you also won't see errors in XML that contains that same sequence. Instead, set the filter to include more of the code that is common to the problematic PHP code. In this case, filter out `<?php` and your XML will not be included in the filter. (You will need to be using PHP's long style tags for this to work, as PHP can also use `<?`, `<%`, and `<script type="PHP">` tags.)

Migration to GoLive 5's Form Container

In GoLive 5, Adobe has changed the form to a "container" paradigm in an effort to always write the form tags correctly according to the HTML specifications. Whereas in GoLive 4 you placed a start `<form>` tag and an end `</form>` tag, GoLive 5 takes care of this process for you. This can cause an inconvenience when first opening a form created in GoLive 4 or another HTML editor. Usually this takes the form of misaligned HTML tags showing up in Layout mode (as shown in Figure 16-24). This problem, while a nuisance, can be easily fixed.

If you are not familiar with HTML, by far the easiest thing to do is to simply follow these steps:

1. Make a backup of your page.

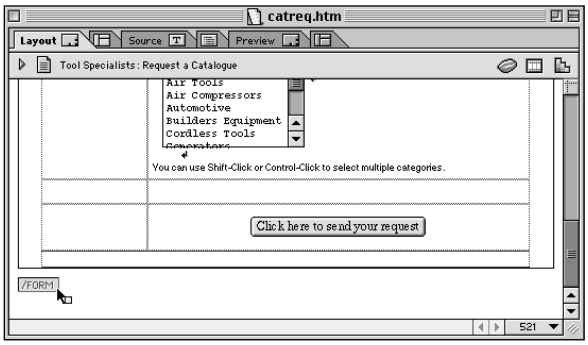


Figure 16-24: A misaligned </form> tag showing up in Layout mode after opening a form created in GoLive 4

2. Drag a new Form container from the Forms tab of the Objects palette to the beginning of your page.
3. Copy the settings from your old Form (such as Name, Action, and other attributes in the Form Inspector) to your new form.
4. Move the contents of the old form into the new one (either by copying and pasting or drag and drop). Be careful that you move any Hidden fields too.
5. Once you are sure that nothing exists in the old form container, delete it. If a misaligned `</form>` tag shows up on your page, delete that too.

If you do know HTML, another way to correct this problem with pages that use a single form (see the note on pages with multiple forms later in this chapter) is to move the misaligned end tag into the correct position in the HTML source (either in source mode or in the Source Code palette. You need to change to Source mode or use the Source Code palette to locate the misaligned tag pair and correct it (usually by moving the end tag that is showing in Layout mode after the `</form>` end tag in the source).

The layout of pages that contain multiple forms may need to be rethought slightly to accommodate the new Form container. For instance, try splitting the page into multiple tables, each one containing a form (which will usually help with page-rendering times anyway).

Using GoLive Actions with Forms

As well as validating your form data, GoLive Actions can be used to do other things with forms and form data. Sometimes you need to store or retrieve information that someone has input in a form separately from other information they have input. For instance, customers place orders on your Web site, and when they return they are welcomed with a personalized greeting or brought back to the section they were last viewing. You don't need all of the information they filled in, just the relevant fields. This is where you can use GoLive Actions such as Get Form Value and Set Cookie.

The Get Form Value Action is used to read the user's input from the text fields in a form. Typically, you'll need to do several form field value reads in order to gather all the entries (such as first name, last name, address, and so on). Each field you want to read requires a separate Get Form Value Action. Normally you would use this Action with an OnCall event triggered by a Set Variable Action attached to the form's Submit button, but unfortunately the version of Set Variable that ships with GoLive 5 doesn't work properly with the Get Form Value Action. A replacement for the Set Variable Action that does work has been developed by Robert McDaniels and is available at `www.golivebible.com`, and should also be available from the Adobe Web site by the time you read this.

To demonstrate the replacement Set Variable Action written by Robert McDaniels and to give you an idea of how you could use form fields, variables, and cookies in your sites, Rob Keniger of Big Bang Solutions has prepared a simple, three-page example site showcasing the Action. Make sure you install the replacement Set Variable Action before opening the site and remember that cookies don't work on files opened from your hard disk, so you'll have to upload the pages to a server to test them.

To set an Action to get the value of a form's text field and store it in a variable, follow these steps:

1. Install the Replacement Variable Action into your GoLive ➪ Modules ➪ Jscripts ➪ Actions folder.
2. Drag a Head Action from the Smart tab of the Objects palette into the header of the page, select OnLoad from the Exec menu, and then choose Actions ➪ Variables ➪ Declare Variable. Enter a name for the variable in the Inspector and give it a type of String.
3. Drag another Head Action into the header of the page, select OnLoad from the Exec menu, and then choose Actions ➪ Variables ➪ Init Variable. Choose the variable you just created in the previous step from the pop-up and give it a value of zero.
4. Drag another Head Action into the header of the page, select OnCall from the Exec menu, and then choose Actions ➪ Getters ➪ Get Form Value. Name the Action in the Inspector so you can use it later and enter the name of the form and the field you want to check in the appropriate inspector fields.
5. Set up a trigger. You can attach the trigger for the Action to a form, a link, or a form button. Do one of the following to attach the trigger:
   - To attach it to a form, select the form placeholder, open the Actions palette, choose "Form Submit" from the Events list, click the New Action button to add an event, and then choose Actions ➪ SetVariable (Form).
   - To attach it to a button, select the button, open the Actions palette, choose "Mouse Click" from the Events list, click the New Action button to add an event, and then choose Actions ➪ SetVariable (Form).

- To attach it to a link, select the link and change its URL to #, open the Actions palette, choose "Mouse Click" from the Events list, click the New Action button to add an event, and then choose Actions ➪ SetVariable (Form).

The Action now appears in the lower half of the Inspector along with a pair of text fields into which you enter your data (see Figure 16-25).

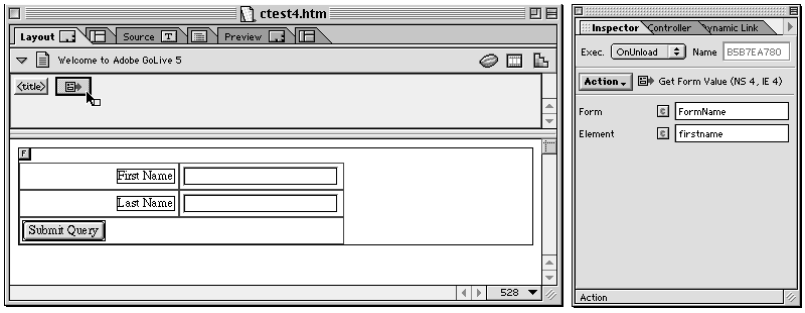


Figure 16-25: Getting a Form Value

6. Enter the name of the form in the first text box.
7. Enter the name of the text field in the second text box.

To avoid possible typographical errors which would introduce bugs into your Actions, copy and paste the object names from the appropriate Inspectors into these fields.

Now that the field is stored in a variable, you could use a Write Cookie Action to store the value so that it can be displayed on other pages. This is discussed in Chapter 18.

Dealing with Nonstandard Tags

In addition to all the HTML tags, the CGI or other server-side processing program you use is bound to have its own commands. One of the new features of GoLive 5 is 360 Code. GoLive should leave all of your custom code alone, and shouldn't break it, change it, or alter its formatting in the HTML source. If you find GoLive is having trouble with your code or making changes to it that aren't correct, you can use these tips.

The Noedit tag

You can tell GoLive not to touch any part of your code by surrounding that code with the noedit tag. Anything surrounded by this tag doesn't appear in Layout, so you don't want to surround any of your page's visual elements.

Because the code you're protecting is not standard HTML and therefore has no visual GoLive interface, you'll be entering the code in Source mode. While you're at it you might as well add the <noedit> tag set. While in Source mode, simply add <noedit> in front of the text you're protecting and then add </noedit> at the end of the code you're protecting.

The 360 Code feature of GoLive 5 may seem to make the <noedit> tag unnecessary, and in the sense of it protecting the code from GoLive, it is (in *almost* all situations). It is still useful though—for protecting code from yourself or other people working with your page in GoLive. It separates the code visually within your page and stops you from changing things within that specific block of code in Layout mode that may affect how it is displayed in the browser. Think of it as insurance against human error rather than insurance against GoLive.—*Richard McLean, Web developer, designeffect.com*

When you switch back to Layout mode you'll see a pinkish rectangle that says "<noedit>...</>." The protected code is between those tags, but not visible in Layout. The result of this is something like: <noedit><?php code ...; ?></noedit>

If you're going to reuse this code frequently, consider storing it in the Objects palette's Custom tab. To do this, switch back to Layout mode and drag the <noedit> ... </> icon to the Custom tab. In the future, whenever you drag that saved tag back from the Custom tab to a page, you'll be adding the pair of tags, complete with the protected content.

I don't recommend adding the <noedit> tag in Layout view because it's often hard to tell where the tags belong. However, in case you really, really want to try it, to add the <noedit> tag from Layout mode, drag the Tag icon from the Basic tab of the Objects palette into place in front of the code you wish to protect. This tag doesn't have any attributes, so leave Attribute and Value blank. Finally, switch to the Content tab and type (or paste) your code.

For the rare occasions when you've created your page for a server-side application or CGI, and GoLive is having problems with it, you can use Find & Replace to add the <noedit> tag.

For example, if you are using PHP and are using the long form of PHP tags, you could search for <?php and replace it with <noedit><?php and then find all occurrences of ?> and change them to ?></noedit>.

Working in Source view

If you work *only* in Source view you can avoid the problem. You can switch to Layout (or Outline) to look at your page, but you can't make any changes to the page in these views. The act of *changing* your code in Layout or Outline causes the rewrite that is problematic.

Locking pages into Source only

If you find that when you try to work solely in Source mode, you forget and make changes while in Layout (or Outline) mode, you can set GoLive to open your page only in plain HTML source. The good news is that this trick eliminates the temptation to make changes in Layout, but the bad news is that you lose all of the great GoLive interface. This trick puts you in a plain text mode, like being in SimpleText (Mac) or Notepad (Windows).

If the pages in your site are typically using the `.html` extension, you will take advantage of the also-available `.htm` extension.

The trick is to set up GoLive's File Mapping so that pages with a specific extension only open as plain text. Here's how:

1. Choose Edit ➪ Web Settings.
2. Switch to the File Mappings tab.
3. Click the arrow next to the text/group to reveal the individual file types.
4. Scroll down to htm (listed in the Suffix column) and click that row to select it.

   The htm suffix's preferences appear in the Inspector (now the File Info Extension Inspector).
5. In the Mime Type area, change the word "text/html" to "text/plain" and then click OK.

This causes any pages that end in `.htm` to open as a plain text document showing pure HTML code. Then change the extension on any pages that you want to force into source mode to `.htm` instead of `.html` and work on that page in source mode. When you open this page you won't see the usual GoLive interface. You'll just see text. If you change your mind about working this way, you can change the extension of the page back to `.html`. File Mapping only sets the way GoLive treats your page. It doesn't change the code in the pages or the way the pages will be handled on the Web server.

If you are using `.htm` as your normal page extension, reverse these instructions and set `.html` as the plain preference. However, if your server absolutely doesn't recognize `.html`, then you will have to set this extension back to `.htm` prior to uploading the site (or page).

Alternatively, you can set any other pages of a specific file type to open as plain text. For example, you can set all of your ColdFusion pages to open this way by adding the extension `.cfm` in the File Mapping preferences and then setting its Mime Type as "text/plain."

✦ ✦ ✦