Creating a Guest Book Using WebObjects Builder

WebObjects Builder is an application that helps you create WebObjects applications. To create a WebObjects application, you do the following:

- Launch WebObjects Builder.
- Create a new application directory.
- Enter static text.
- Add elements to the page.
- Set bindings for the dynamic elements.
- Write script to customize the dynamic elements.
- Run the application.

To learn how to use WebObjects Builder (and how to use WebObjects in general), let's create a simple guest book application. The guest book application contains a form that has fields in which guests enter their names, e-mail addresses, and comments. Guests enter all pertinent information and then click Submit. When this button is clicked, the guest is added to a list of guests, the page is redrawn, and the new guest appears at the bottom of the list.

Launch WebObjects Builder

On Windows NT, you typically start WebObjects Builder from the Program Manager. If you're using a different platform or if WebObjects Builder doesn't have a shortcut, you can find it in the directory *NeXT_Root*/**NextDeveloper/Apps/WebObjectsBuilder.app**.

When you start up WebObjects Builder, an untitled document appears.

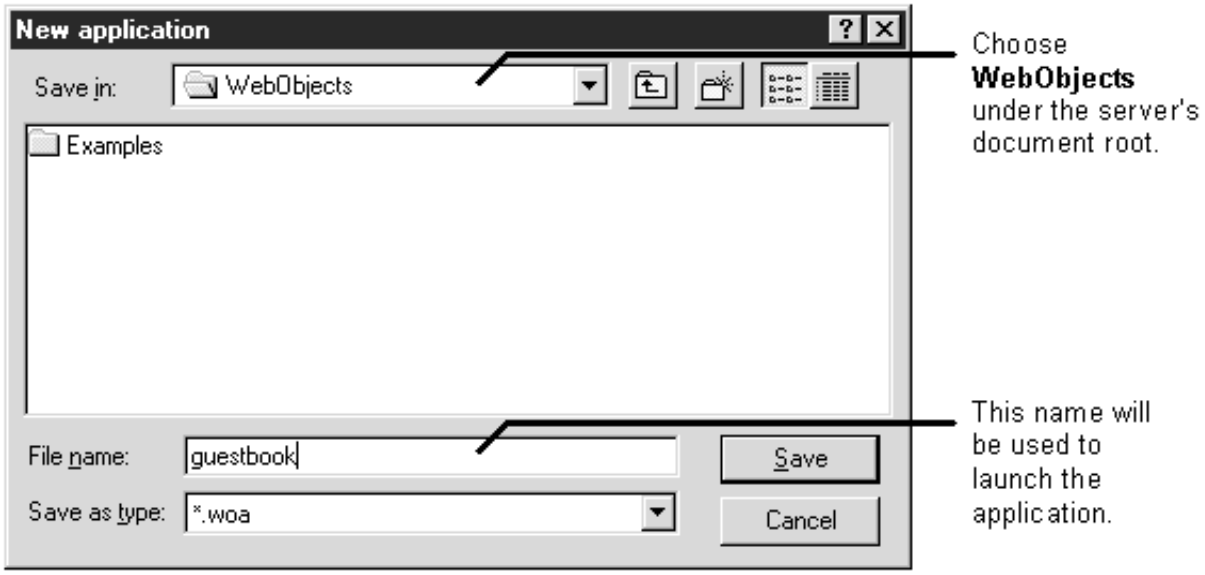
Create a new application directory

The first step to creating the guest book application is to create a directory for it. A WebObjects application is always contained in a single directory that has the extension **.woa**. For the application to run, this directory must be under the your HTTP server's document root (specifically under the **WebObjects** subdirectory) so that your server can access it. WebObjects Builder helps you set the application directory up correctly.

1. Choose File->New Application.

   The panel that appears shows you the contents of the directory <*DocumentRoot*>/**WebObjects**. <*DocumentRoot*> is your HTTP server's document root, which you specified when you installed WebObjects.

2. Type `guestbook` as the file name, and click Save.



WebObjects Builder opens two windows: an application window, which represents the contents of the **guestbook.-woa** directory, and an empty editing window labeled **Main.wo**.

You no longer need the untitled document, so you can close it at any time.

A closer look

The application window shows you what the application directory contains. When first created, all the application directory contains is a directory named **Main.wo**. **Main.wo** is a *component*. Components define dynamic HTML pages and are the basic building blocks of a WebObjects application. The first component in an application is called **Main.wo**.

There are three parts to a component: an HTML template, a script, and bindings between the two. You use WebObjects Builder to create these three parts. "Components" in the introduction to the *WebObjects Developer's Guide* describes other things that can go into a component.

Usually applications contain several components, each representing all or part of a page. The guestbook application, however, only has a single page, which is represented by the Main component. The rest of this tutorial describes how to edit the Main component.

To learn how to set up more complex applications, see "Setting up Applications" in *Using WebObjects Builder*. For more background on WebObjects applications and what they can contain, see "The Ingredients of a WebObjects Application" in the introduction to the *WebObjects Developer's Guide*.

Create the application's input fields

The guestbook application allows users to enter their names, e-mail addresses, and comments in an HTML form. In this first part of the tutorial, you'll create the HTML form for input and provide a way for the component's script to capture the input and add it to the list of guests.

Enter static text

A component can contain both static text and dynamic HTML elements. To add static text to a page, you can type it directly into the component's window. To demonstrate this, add a title for the guestbook's page.

1. Click the **Main.wo** window to bring it to the front.
2. Type `My Guest Book` and press Enter.
3. Select the words `My Guest Book`.
4. In the toolbar, click the largest A and the Center icon to increase the text's font size and to center it.

You just added an HTML element to the page in the simplest way: by typing and formatting. The toolbar allows you to center or justify the text you enter, to select a color, and to select a font size. Usually, you'll want better control over what type of HTML element is created when you enter text. For this reason, it's more common to use the palette window as described in the next section.

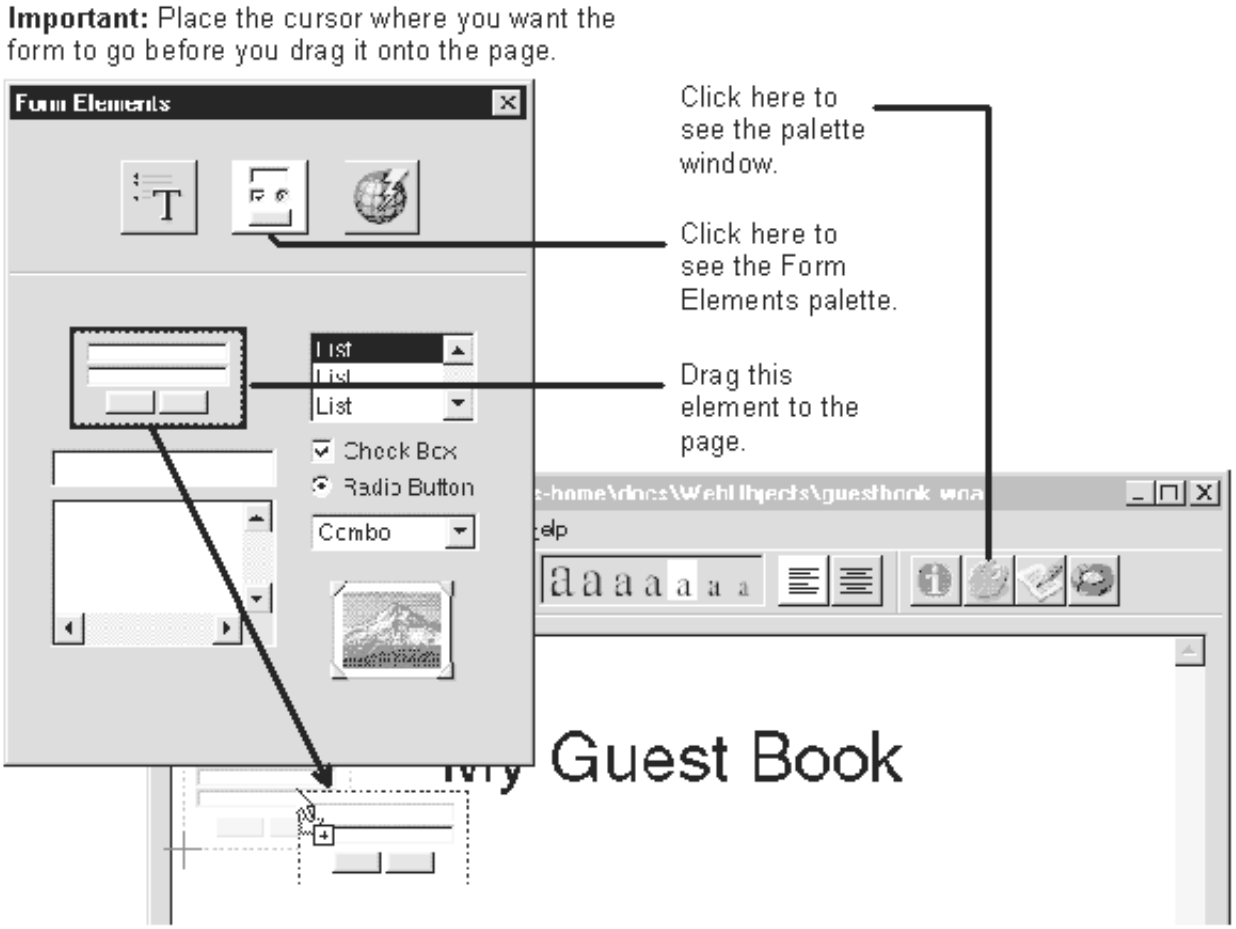
Create form-based dynamic HTML elements

You need to add dynamic HTML elements to capture the guest's input. You'll do this by creating a form.

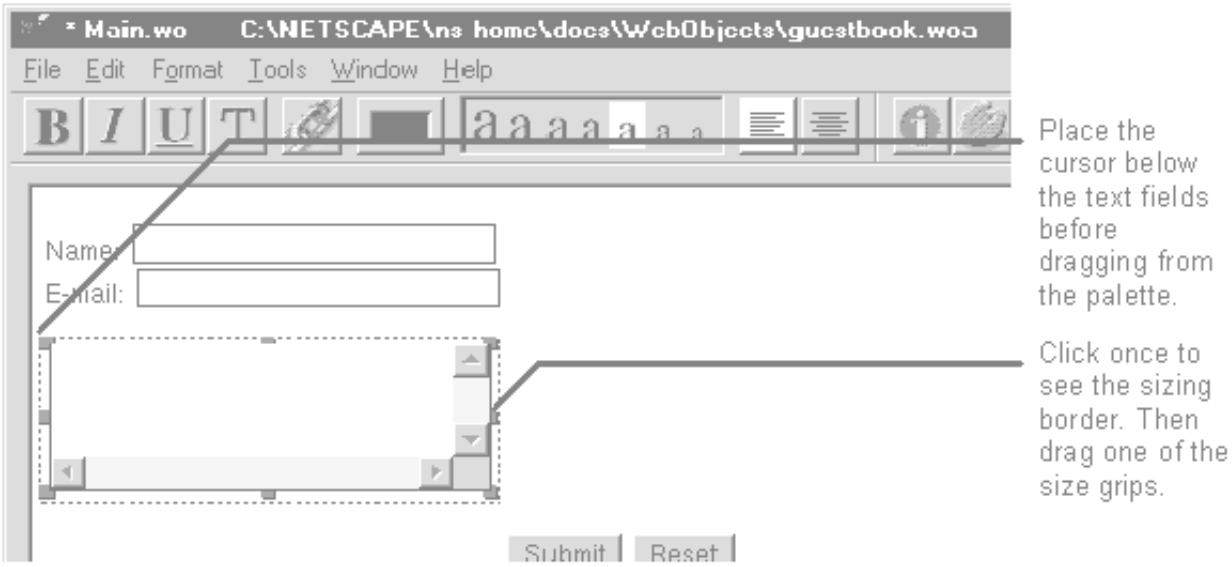
1. Place the cursor on the line after the `My Guest Book` text and press Enter.
2. Click the palette button on the main window to open the palette window.

   The palette window contains three different palettes from which you can choose ready-built elements. The first contains static HTML elements, the second contains form-based dynamic HTML elements, and the third contains abstract dynamic elements (objects that have no true HTML equivalent).
3. In the palette window, click the middle icon to display the Form Elements palette.
4. Drag a form object from the palette onto the page.

   A pre-defined form with two text fields and Submit and Reset buttons appears. This form is just a template that you can modify.



5. Select the word `Field` in front of the first text field and type `Name` in its place.
6. Select the word `Field` in front of the second text field and type `E-mail` in its place.
7. Place the cursor below the text fields, and drag a multi-line text element onto the page.
8. Click the multi-line text element once to select it, and then resize it.



9. Choose File->Save to save the Main component.

A closer look

You just created what look like the form elements defined in the HTML specification. However, what you've really created are several WebObjects dynamic elements: a WOForm, two WOTextFields, a WOText, a WOSubmitButton, and a WOResetButton. These dynamic elements look and act like HTML form elements but have the same programming interface as other WebObjects elements.

Notice that before you dragged the form or the multi-line text element from the palette, you had to set the cursor position. Because an HTML file is a text flow, you can't just drag an element to any point on the window. You must place the cursor where you want the element, using hard returns if necessary, and then add the element. If you have text selected when you drag, the element will replace the selection.

To learn more about adding elements to the page, see "Using Dynamic Elements in WebObjects Builder."

Create variables

Now you have a form in which guests can enter information. You need to create variables that store this information. You need two variables: one to store the current guest's input, and another to store the list of all guests. You'll begin by declaring a class for these two variables.

Create a class

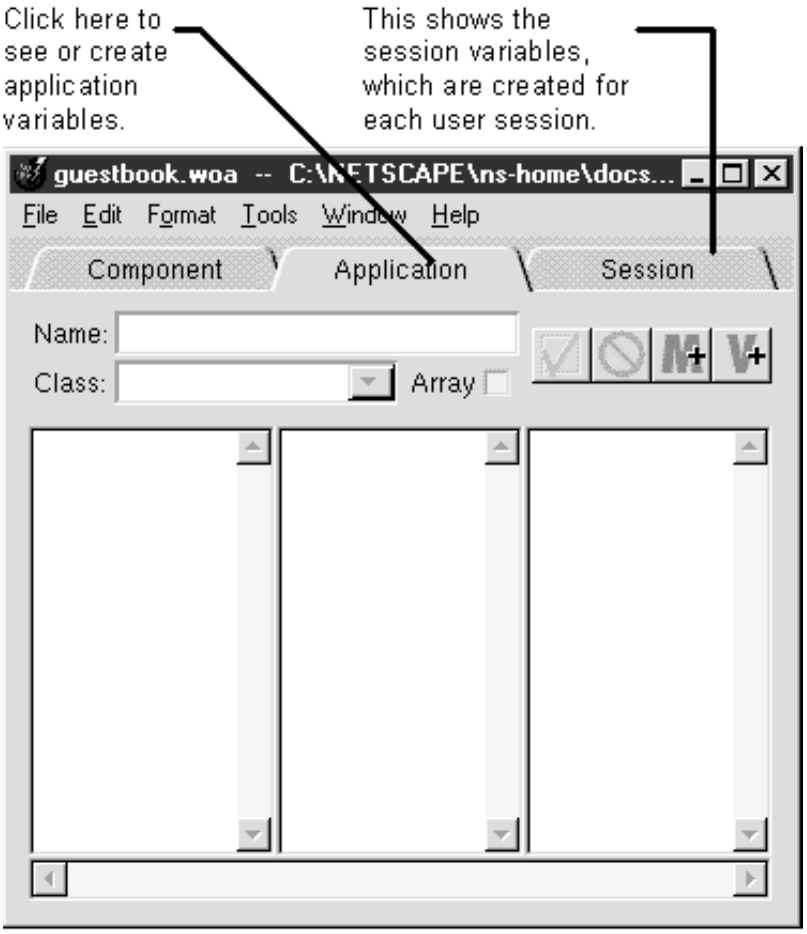
The form you just created contains three pieces of information from a guest—name, e-mail, and comments. You could create three separate variables for these three pieces of information, but as you will see later, creating a class that has three attributes makes writing the application easier by allowing you to treat these three separate pieces as a single entity—a guest.

WebObjects Builder recognizes three types of classes: base classes, such as numbers and strings, composite classes, such as arrays and dictionaries, and custom classes. You don't need to do anything special to create variables of a base class or an array class. But if you want to create a dictionary, you must first go to the Classes window and define the dictionary.

In this task, you will create a dictionary class named Guest. The Guest dictionary class has three keys: name, email, and comments. (You can think of this dictionary as a C structure with three fields.)

1. Go to the application window and click the Application tab.

   The application object browser appears. You use this browser to declare variables that have an application-wide scope. These variables are declared in the application script, which is an optional part of a WebObjects application.



2. Choose Tools->Classes to display the Classes window. This window lists all of the classes that your application recognizes.

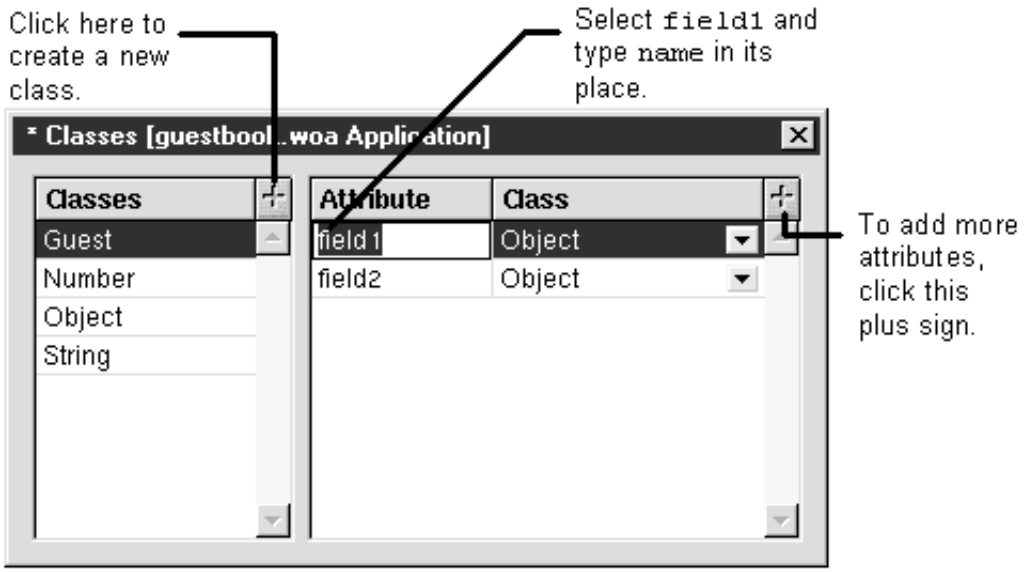
3. Click the plus sign in the Classes table to create a new class.

4. Name the class `Guest`. The Guest class is created with two default attributes.

5. Rename the default attributes `name` and `email`.

6. Click the plus sign in the Attribute table view to add a third attribute named `comments`.

7. Close the Classes window.



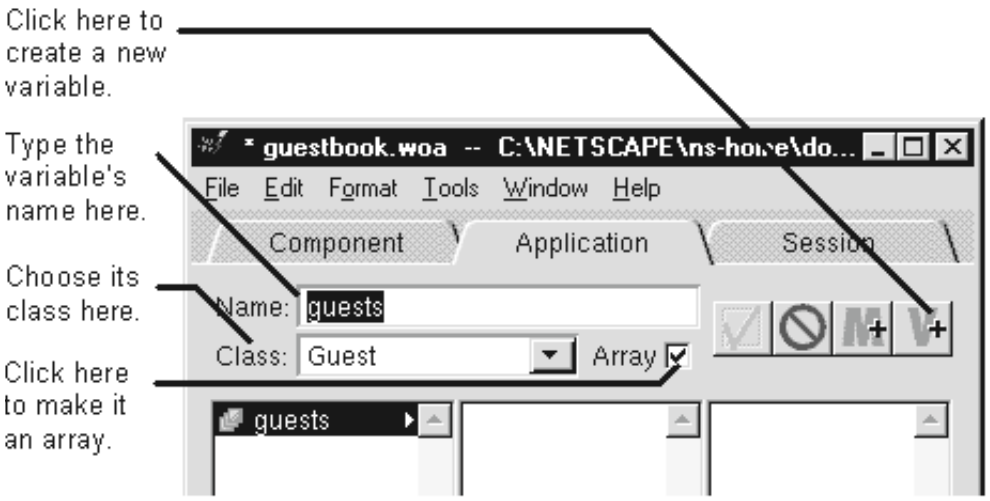
As stated previously, Guest is a dictionary class. A dictionary contains key-value pairs. In the class definition you just created, you specified the keys (or attributes): name, email, and comments. When you create a variable of the Guest class, the variable will store values for each of these three keys.

Declare variables

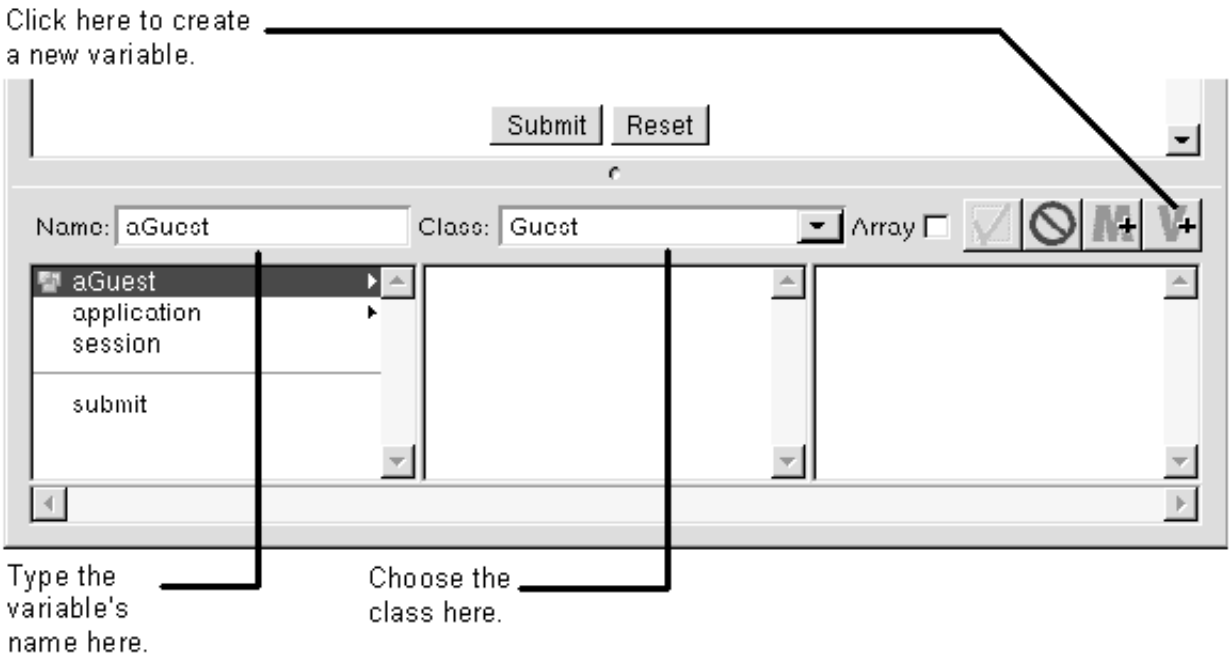
You just defined a Guest class so that you can store and manipulate guest information as a single entity. The next step is to create the array that will store the list of guests and a variable that will store the current guest information. As explained later, these variables are declared in two separate parts of the application: the list is declared in the application script, and the current guest is declared in the Main component.

1. Go back to the application window and click the add variable button.
2. Type `guests` in the Name field and press Enter.
3. Choose **Guest** from the Class pop-up list.
4. Click the Array check box.

   This creates the application variable `guests`, which is an array of Guests.



5. Choose File->Save to save your changes to the application script.
6. In the Main window, click the add variable button (in the bottom half of the screen).
7. Name the variable `aGuest` and select **Guest** from the Class pop-up list.
8. Save the Main component.



A closer look

You just created the variables that represent the current user's input (**aGuest**) and the list of all guests (**guests**).

You declared the **guests** array in the application script (which is named **Application.wos**) instead of the Main component's script. Why? So that it would exist for the life of the application. If you declared **guests** in the Main component (as you did the **aGuest** variable), **guests** would be created each time the Main page was redrawn. The Main page is redrawn every time a user clicks the Submit button, so **guests** would only ever contain one item — the last user who clicked submit.

Note: *Component variables* exist only as long as the component does, which is up until that component's page needs to be redrawn. *Application variables* (defined in **Application.wos**) live as long as the application does. There's a third type of variable, known as *session variables*. Session variables are declared in the session display of the application window and stored in the session script (**Session.wos**). They exist for the lifetime of one user's session. New session variables are created as new sessions are added. For example, if you declared a variable in **Session.wos** and three users were using the application, three instances of that variable would be created, one for each user. If the first user changed the value of that variable, the other two users would not see the change because the session variable is unique to each user's session.

You can read more about application variable, session variables, and component variables in "Using WebScript" in the *WebObjects Developer's Guide*.

Bind the input elements

In the previous task, you created the application's variables. In the next task, you'll write a script that manipulates the variables. For the script to actually use the values of the dynamic HTML elements on the page, you first need to specify which variables represent which elements. This is called "binding" the elements.

There isn't a one-to-one correlation between dynamic elements and variables. Most dynamic elements have several attributes, and you bind each attribute to a different variable or method in your script. For example, WOTextField defines three attributes that define very different things: **value** specifies the value the user enters in the text field, **name** specifies a unique identifier for the text field, and **disabled** specifies if the field is enabled or disabled. If you wanted non-default values for all three attributes, you'd bind them to three different variables.

To bind an element, you must specify three things:

- the variable (or method)
- the dynamic element
- the attribute within the dynamic element

WebObjects Builder provides a shortcut to creating bindings: you can select the variable, select the element, and let WebObjects Builder decide what attribute you want the variable bound to. Most of the time, the shortcut gives you the binding you want. This task shows you how to use the shortcut to bind the elements in the guestbook.

1. In the Main window, select the text field labeled Name in the guestbook's form that you created earlier.
2. In the object browser in the bottom half of the window, navigate to the **name** attribute of **aGuest**.
3. Double-click **name**.

This binds **aGuest.name** to the **value** attribute of the WOTextField labeled Name. After you perform this step, the text `aGuest.name` appears in the text field to show that it has been bound. A status message appears in the upper right corner of the editing display saying the binding has been made to WOTextField's **value** attribute. This means that **aGuest.name** represents the value entered in the Name field.

4. Select the text field labeled E-mail and double-click **aGuest.email** to bind it to the E-mail text field's **value** attribute.

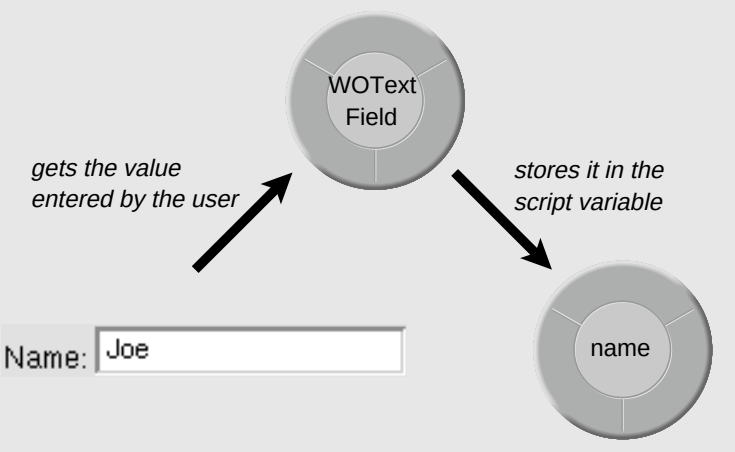
5. Select the multi-line text element and double-click **aGuest.comments** to bind it to the text element's **value** attribute.

   Unlike the text fields, the multi-line text element won't update to show you which variable it is bound to, but you will still receive the status message `bound to value`.

6. Save the Main component.

A closer look

You just bound variables to dynamic elements in the HTML template. In the guestbook application, the WOTextField and WOText elements are input fields—users will enter text in these fields, and the application will read the text. The bindings you made mean that after a user clicks Submit, **aGuest.name** contains the value in the Name field, **aGuest.email** contains the value in the E-mail field, and **aGuest.comments** contains the value in the Comments field.



At the beginning of this section, you learned that WOTextFields have three attributes: **value**, **name**, and **disabled**. Why didn't you have to make a binding to all three? Because **name** and **disabled** are optional attributes. If you don't bind variables to them, WebObjects assigned them default values. It creates a unique name to refer to the WOTextField, and it enables the text field by default. Later, you'll see an example of a dynamic element that has more than one variable bound to it.

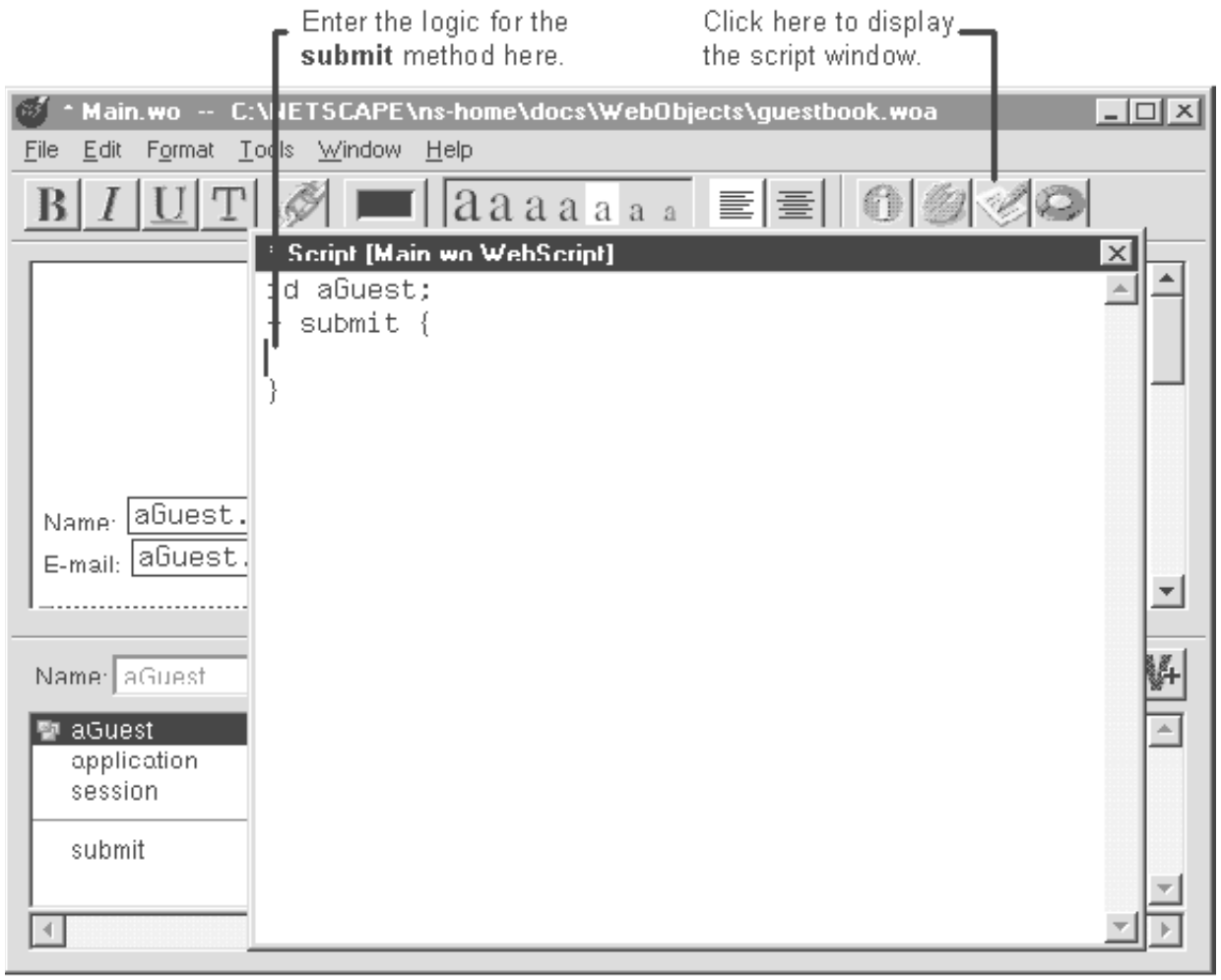
Implement a method

Now that you have declared some variables and you know which elements the variables represent, you need to write a script that uses this information. Specifically, the script needs to take the input from the form elements and add it to the end of the **guests** array. You do this in the **submit** method.

You may have noticed that when you dragged the form onto the page, the word `submit` appeared in the object browser at the bottom of the window. This represents the **submit** method. When you create a form with a Submit button, WebObjects Builder creates a **submit** method for you and binds the button to the method.

1. In the Main window, click the script button to display a window containing Main's script file.

   Main's script file appears in a separate window. It contains declarations for the variables and methods listed in Main's object browser, namely **aGuest** and the **submit** method. (You don't see the **guests** array because it's declared in the application script, not the Main component.)



2. Enter the following line inside the declaration for the **submit** method:

```
[self.application.guests addObject:aGuest];
```

3. Close the script window and save the Main component.

A closer look

You just implemented the **submit** method. When the user clicks the Submit button, the variable **aGuest** receives the values that the user entered in the form, and the **submit** method is invoked. This method adds the information contained in **aGuest** to the end of the application script's **guests** array. The **guests** array contains a list of everybody who ever used the guestbook.

To implement this method, you used a language called WebScript. WebScript is the WebObjects scripting language. You can read more about it in "Using WebScript" in the *WebObjects Developer's Guide*.

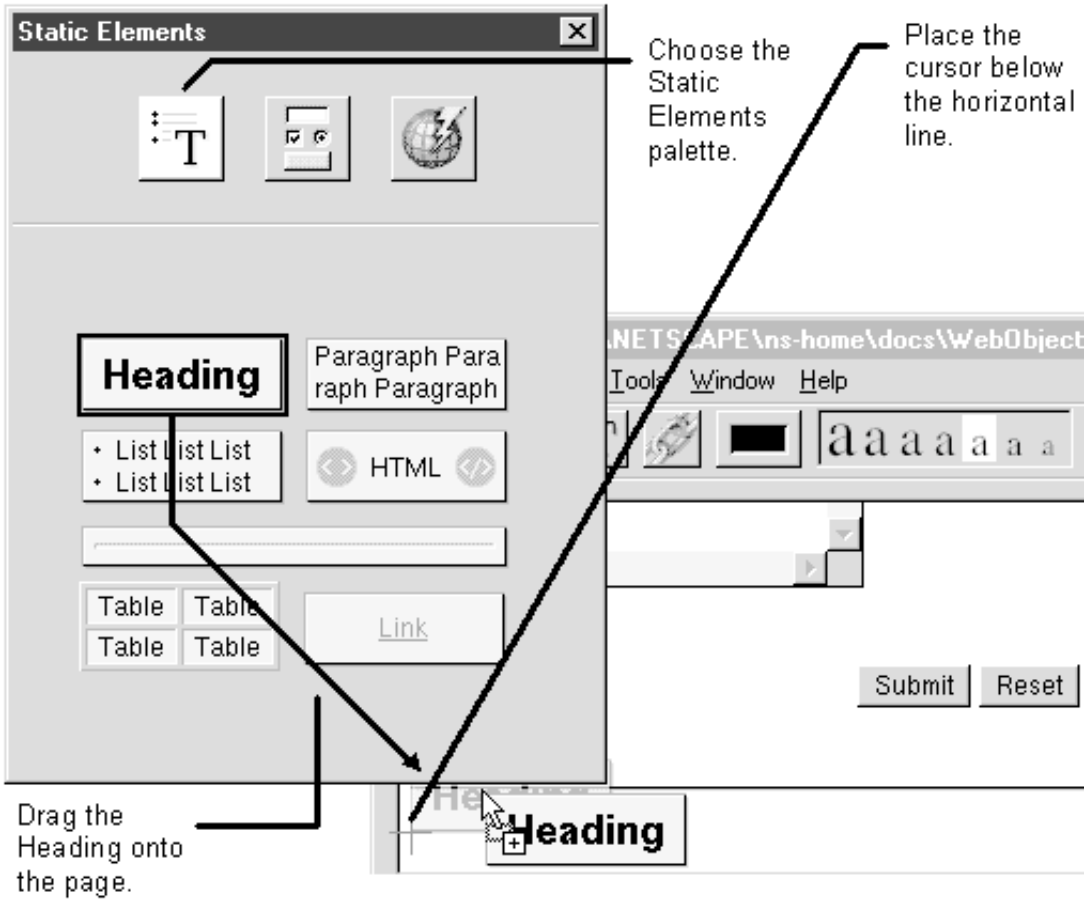
Create the application's output

So far, you have a way for the guest to enter information, a way for the application to add that information to the guest list, and a way to store the guest list. Now, you need a way for the application to display the list of guests. To do this, you'll add more dynamic elements to the main page. But first, create a heading for the output.

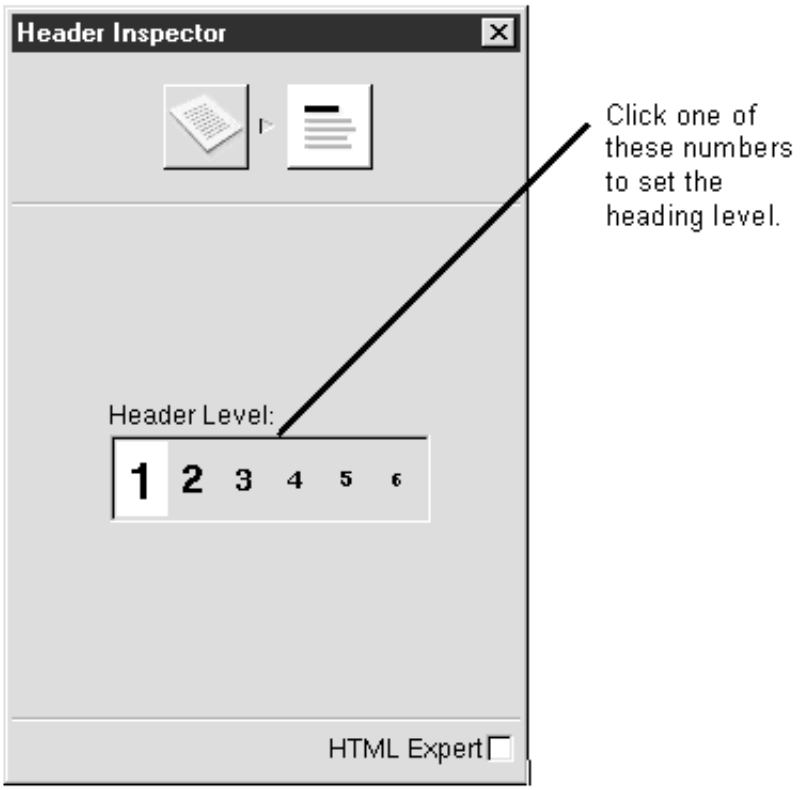
Create static HTML elements

When you created the heading at the top of the page, you just typed directly into the Main component's window. To create the heading for the page's output section, you could just do the same thing, but it's more common to drag elements from the palette so that you can better control what HTML tags are used.

1. Place the cursor at the bottom of the Main page, below the horizontal line.
2. In the palette window, click the first icon to display the Static Elements palette.
3. Drag a heading element to the page.
4. Select the text inside the heading and type `Guests`.



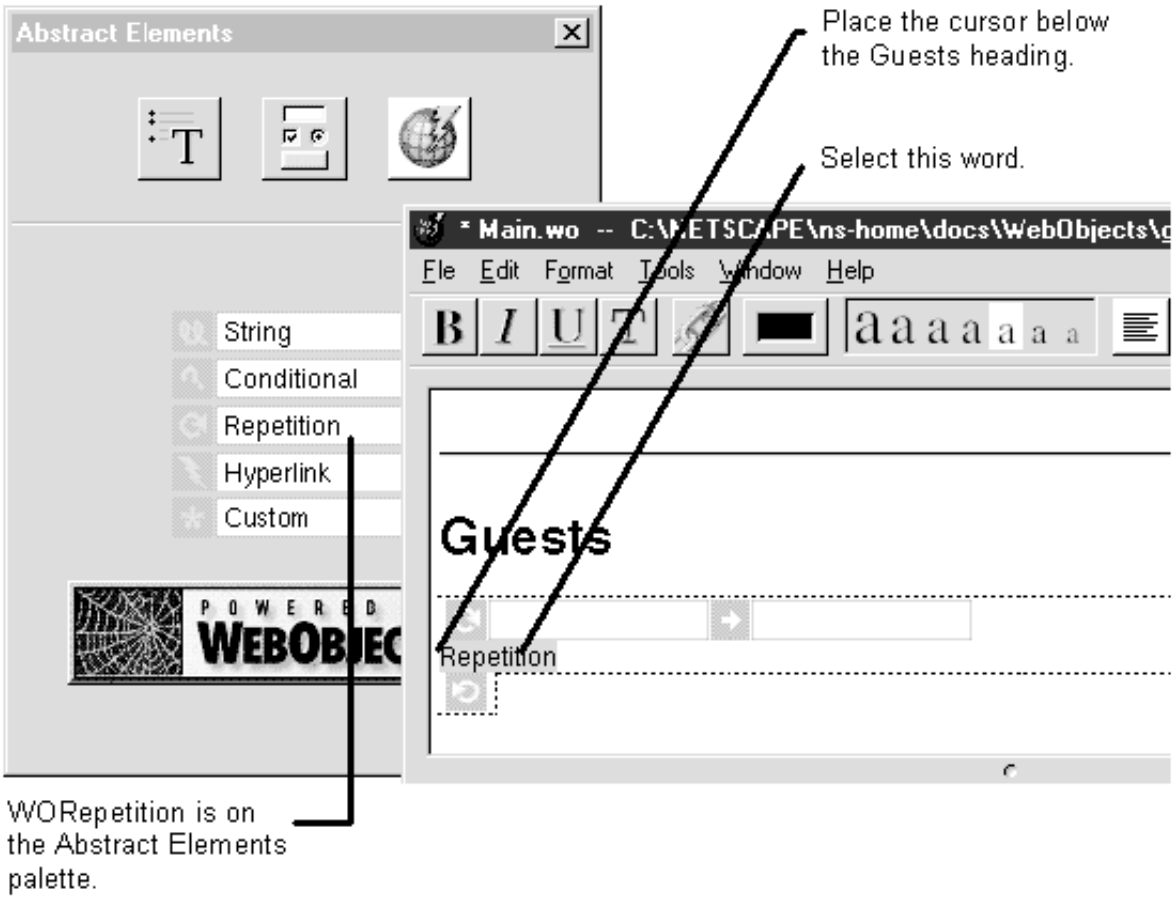
You just added a static HTML element, a heading, to the page. As an optional step, you can modify the heading's appearance using the Inspector. Click the inspector button (the button labeled with an "i") to display the Inspector. Select the heading element, and you'll see the properties you can set for the heading. You can change the heading to any one of the six levels supported in HTML (H1 through H6). To learn more about adding elements to the page, see "HTML Editing in WebObjects Builder" in *Using WebObjects Builder*.



Add abstract dynamic elements to the page

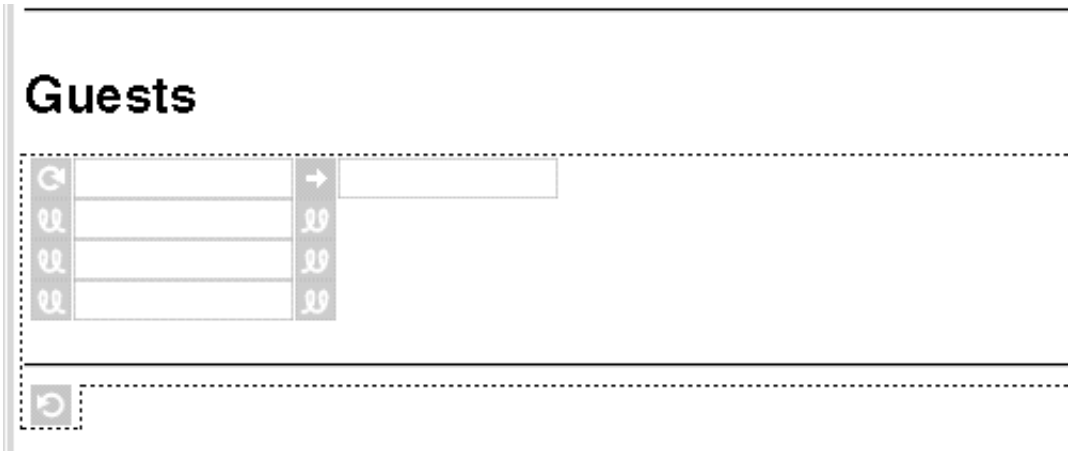
To display the list of guests that have visited the page, you'll use abstract dynamic elements. Unlike the form elements you added earlier, abstract elements have no true equivalent in HTML. Instead, your application determines what these elements look like.

1. Place the cursor at the bottom of the page.
2. In the palette window, click the third icon to display the Abstract Elements palette.
3. Drag a Repetition element onto the page.



4. Select the word `Repetition` inside the Repetition object, and then drag three String elements from the Abstract Elements palette. After each String element, enter a carriage return so that each string is on a separate line.

5. Place the cursor below the third String element inside the Repetition, choose the Static Elements palette, and drag a horizontal line onto the page.



You're done adding elements to the page now, so you can close the palette window and save the **Main.wo** component.

A closer look

You just added several abstract dynamic elements: a WORepetition and three WOStrings. The WOStrings are simply dynamic strings. The application decides what text the strings should display at run time. WORepetition is an object with two parts: contents and a list. You just defined the contents. They are the three WOStrings and the horizontal rule. The list you define when you bind the WORepetition, which you'll do next. WORepetition is a complex element. You'll read about how it works after you make its bindings.

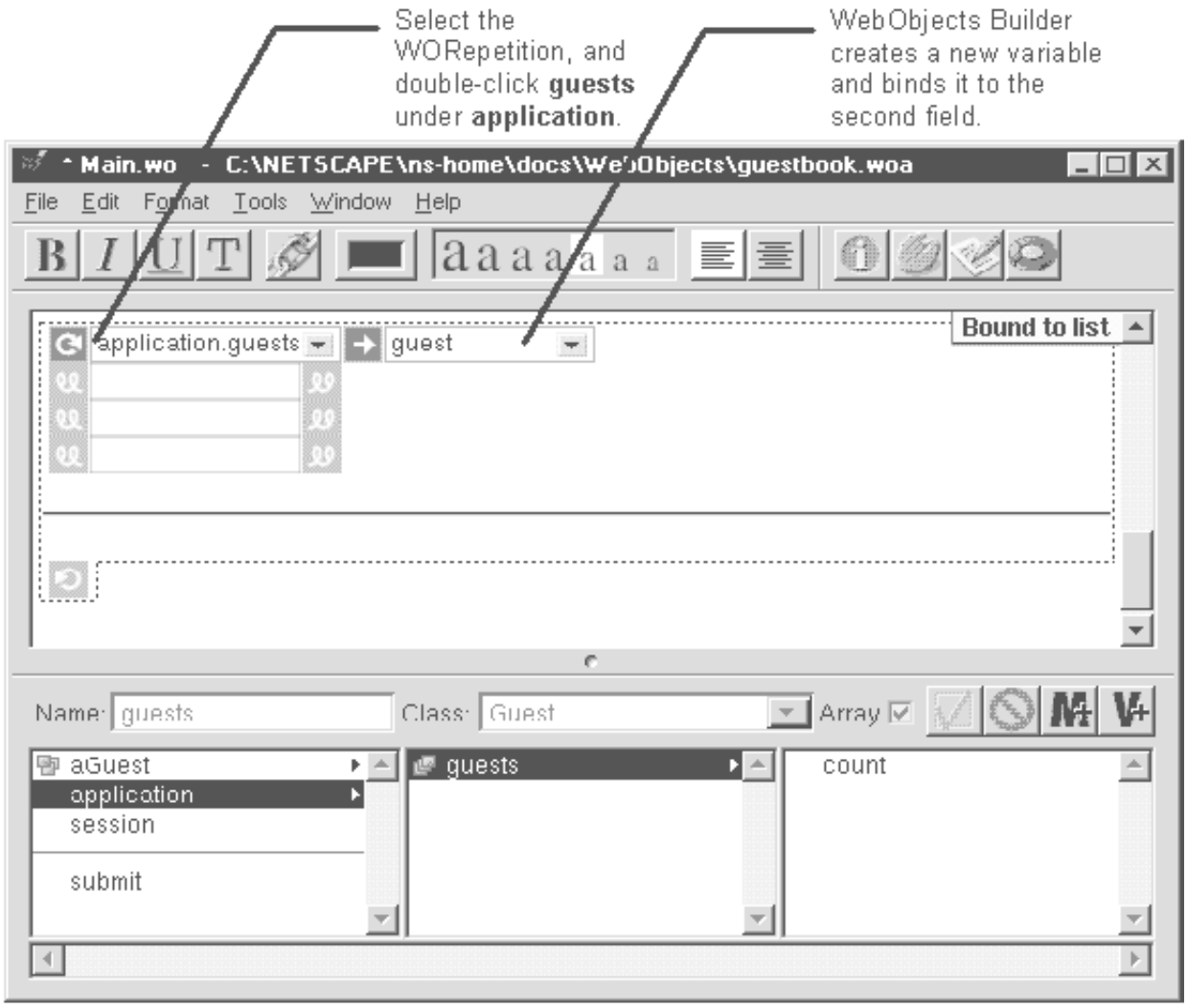
Although this example uses WORepetition and WOString only to display output, you could also use them to display input. To learn more about these elements, see the Dynamic Elements section of the *WebObjects Reference* .

Bind the WORepetition element

Just as you had to bind the input elements to variables so that the input values could be captured in the script, you now have to bind the output elements to script variables so that the page displays the correct information.

1. Click the first field inside the WORepetition, and double-click **guests** under **application** in the object browser.

   This binds the **guests** array to the **list** attribute of WORepetition. WebObjects Builder automatically creates a variable called **guest** and binds it to the **item** attribute (the second field in the repetition).

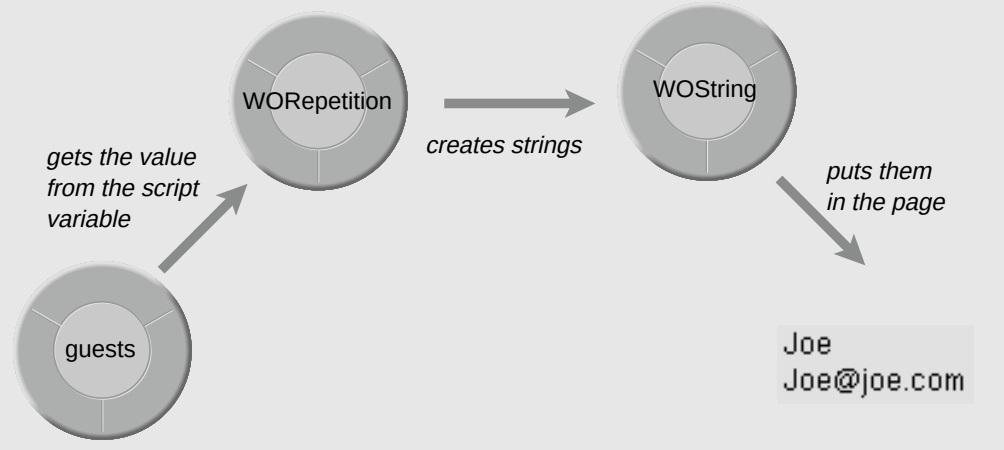


2. Bind each item in the variable **guest** (**name**, **email**, and **comments**) to the three WOStrings inside the WORepetition by selecting the WOString and double-clicking the variable.

3. Save the Main component.

A closer look

You just bound the WORepetition's **list** attribute to the application script's **guests** array, which you created earlier. A WORepetition displays its contents for each item contained in its **list** attribute. When you created the WORepetition, you defined its contents as three WOStrings. Then, you bound those WOStrings to the information in the newly created variable **guest**, which is updated to contain the current item in the list as WORepetition moves through the list.



Creating a WORepetition and binding it in this manner is the equivalent of saying "for each guest item in the guests array, display the name, e-mail, and comments."

WORepetition is an example of a dynamic element that requires bindings to two different variables. The **list** attribute is bound to the **guests** array, and the **item** attribute is bound to the variable **guest**. WOString, on the other hand, is like the WOTextFields and WOText that you bound earlier: WOString defines multiple attributes, but you only need to bind to one attribute, the **value** attribute, for the string to work properly. The other attributes are assigned default values.

You can learn more about WORepetitions, WOStrings, and the other dynamic elements in this example by looking them up in the Dynamic Elements section of the *WebObjects Reference*. To learn more about how to use WebObjects Builder to create dynamic elements, see "Using Dynamic Elements in WebObjects Builder."

Implement awake and sleep

The final step to creating the guestbook application is to implement the methods **awake** and **sleep**. These two methods are standard methods that can be implemented in any component that needs them. **awake** is a method that sets up the component at the beginning of every transaction. **sleep** resets the component at the end of every transaction.

1. Click the script button to display Main's script file.
2. Add two more methods in the script window:

```
- awake {
   aGuest = [NSMutableDictionary dictionary];
}
```

```
- sleep {
   aGuest = nil;
}
```

3. Choose File->Save All to save all of the files in the application.

You're done writing the guestbook application, so you can exit WebObjects Builder.

A closer look

The job of **awake** is to prepare the page for the current transaction. In Main's case, whenever a new transaction begins, there's a new user to add to the guest list. Therefore, Main needs to allocate a new, empty **aGuest** variable before the transaction begins. **aGuest** (and all other variables of type Guest) are really NSMutableDictionary objects. NSMutableDictionary is a class defined in the Foundation Framework. NSMutableDictionary objects have key-value pairs. (The class is called "mutable" because you can change the contents of the dictionary after you create it.)

After the **awake** method, WebObjects code takes the values from the form and assigns them to the attributes in the **aGuest** variable. By the time the **submit** method begins executing, **aGuest** contains the current guest's information.

The **sleep** method performs the inverse operation of the **awake** method. It essentially erases any temporary state the page had for this transaction. Because you created an empty **aGuest** variable in **awake**, you reset it to **nil** in **sleep**.

The sequence of events for a transaction in the guestbook goes like this:

1. The user clicks the Submit button.
2. Main's **awake** method creates a new **aGuest**.
3. WebObjects code populates **aGuest**'s attributes (name, e-mail, and comments) with information the user entered on the page.
4. The **submit** method adds **aGuest** to the end of the guest list.
5. The WORepetition iterates through the guest list and displays the name, e-mail, and comments of each guest.
6. Main's **sleep** method sets the **aGuest** to **nil**.

To learn more about **awake**, **sleep**, and WebScript in general, see "Using WebScript" in the *WebObjects Developer's Guide*. To learn more about the Foundation Framework (where NSMutableDictionary is defined), see the Foundation chapter of the *WebObjects Developer's Guide*.

Run the application

If you look at your application in the file system, you can see the files WebObjects Builder created for you. The **Application.wos** script is directly under the **guestbook.woa** directory, and there are three files under **Main.wo**: an HTML file, a script file, and a *declarations file*, which holds the bindings between the script and the HTML file. You might also see files with a **.woo** extension or files named **API.table**. These are files used internally by WebObjects Builder. You'll also see a **Session.wos** file (for the session script) directly under **guestbook.woa**, but it is an empty file because guestbook does not need a session script.

1. Launch your web browser (for example, Netscape Navigator).

2. Load a URL with the following form:

   `http://`*web_server_host*`/`*cgi-bin_directory*`/`*adaptor*`/`*application_directory*

For example, with a web server named Gandhi, a cgi-bin directory named **cgi-bin**, a WebObjects adaptor named **WebObjects**, and an application directory named **guestbook**, use this URL:

```
http://Gandhi/cgi-bin/WebObjects/guestbook
```

It's common to store all of your application in one directory under **WebObjects**. If you do this, you must give the path to the application. For example, if you stored the **guestbook.woa** directory in the directory *<DocumentRoot>***/WebObjects/MyApps**, the URL would be:

```
http://Gandhi/cgi-bin/WebObjects/MyApps/guestbook
```

To learn what happens when you run a WebObjects application, see "Connecting to a WebObjects Application" in the introduction to the *WebObjects Developer's Guide*.

Troubleshooting

If you have trouble running the application, try running it manually. To do so, open a DOS command window and enter this command:

`%NEXT_ROOT%\NextLibrary\Executables\WODefaultApp -d` *server/DocumentRoot application_directory*

where **NEXT_ROOT** is usually **c:\NeXT**, *server/DocumentRoot* is the full path of the server's document root directory, and *application_directory* is the application's directory (relative to the **WebObjects** directory). For example:

```
c:\NeXT\NextLibrary\Executables\WODefaultApp -d c:/netscape/ns-home/docs guestbook
```

Note: Be sure to use forward slashes in the arguments to **WODefaultApp**.

This command starts up the WebObjects default executable, which runs the guestbook application and connects it to the WebObjects Framework. Once you have started this executable, go back to your Web browser and reload the URL. Make sure that the URL actually reloads. If necessary, quit the browser and start it up again.