Getting Started

This chapter contains a brief introduction to creating, installing, and running applications with WebObjects. Using the web equivalent of HelloWorld as an example, this chapter answers the following basic questions:

- How do I connect a WebObjects application to the Web?
- What's in a WebObjects application?
- What parts do I write, and how do I write them?
- How do I run a WebObjects application?
- What happens behind the scenes of a running WebObjects application?

Connecting a WebObjects Application to the Web

WebObjects is an environment for building and deploying World Wide Web applications. For development, it provides a scripting language and objects that you use to create web applications. For deployment, it provides a system of interrelated components that connect your WebObjects applications to the Web. WebObjects development topics are discussed in more detail later in this chapter. This section discusses the parts of WebObjects related to deployment.

Connecting a WebObjects application to the Web involves the following:

- *An HTTP server*. You can use any HTTP server that uses the Common Gateway Interface (CGI) or the Netscape Server API (NSAPI).

- *A WebObjects adaptor*. A WebObjects adaptor connects WebObjects applications to the Web by acting as an intermediary between web applications and HTTP servers. Adaptors insulate applications from server interfaces by handling all server communication. Simply by switching adaptors, you use a different HTTP server and a different server interface without modifying application code.

  WebObjects provides an adaptor for servers that use the Common Gateway Interface (CGI) and an adaptor for the Netscape Commerce Server that uses the Netscape Server API (NSAPI).

- *A WebObjects application executable*. The application executable receives incoming requests and responds to them, usually by returning a dynamically generated HTML page. This executable can be a compiled application linked with the WebObjects library, or it can be one of the two default WebObjects application executables: DefaultApp or EOFDefaultApp.

  If the application doesn't contain any compiled code, you use one of the default applications. Instead of building your own executable, you implement

all the application logic in script files. The default applications use these scripts to respond to requests.

Users can use any web browser to connect to a WebObjects application. As shown in Figure 1, when an HTTP server receives a request for a WebObjects application, it forwards the request to the WebObjects adaptor. The adaptor in turn forwards the request to the WebObjects application. Similarly, after an application generates a response—usually an HTML page—the application sends the response to the WebObjects adaptor, and the adaptor sends the response to the HTTP server.

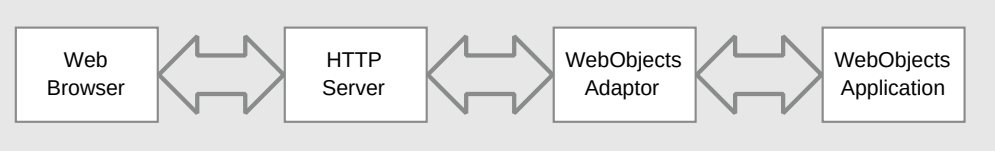

Figure 1. Chain of Communication between Browser and WebObjects

WebObjects Adaptors

When a WebObjects adaptor receives a request from the server, it repackages the request in a standard WebObjects format and forwards it to an appropriate WebObjects application. As shown in Figure 2, all WebObjects adaptors communicate with WebObjects applications in the same way, but they communicate with HTTP servers using whatever interface is provided by a particular server. For example, the WebObjects CGI adaptor uses the Common Gateway Interface, and the Netscape Interface adaptor uses the Netscape Server API. Thus, WebObjects adaptors can take advantage of server-specific interfaces but still provide server-independence.

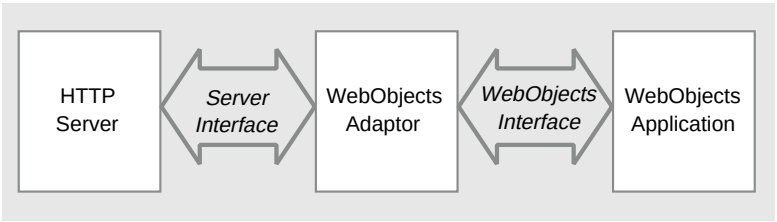

Figure 2. The Role of a WebObjects Adaptor

The Common Gateway Interface is supported by all HTTP servers, so you can use the WebObjects CGI adaptor with any server—including those that are publicly available. As performance demands increase, you can use the Netscape Interface adaptor with a server that supports the Netscape Server API. The

Netscape Server API eliminates the overhead of starting a new process for each request by dynamically loading the Netscape Interface adaptor. As shown in Figure 3, the communication between the Netscape Interface adaptor and the HTTP server occurs inside a single process.

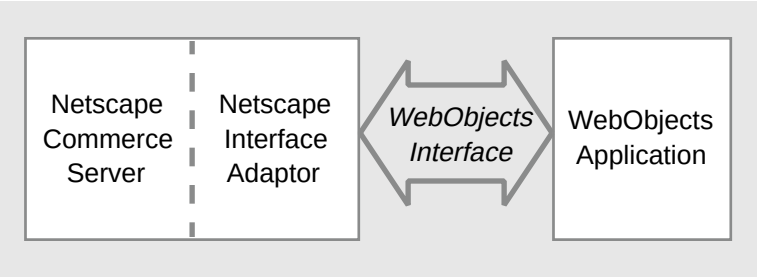

Figure 3. The Netscape Interface Adaptor

By default, WebObjects uses the WebObjects CGI adaptor. For more information on configuring this adaptor or the Netscape Interface adaptor, see the "Serving WebObjects" document.

Where Things Go

WebObjects is installed in a platform-dependent location. The following table shows the location for each platform:

Platform	Installation Directory
NEXTSTEP	/NextLibrary/WebObjects
Solaris	/usr/NextLibrary/WebObjects
Windows NT	<NEXT_ROOT>/NextLibrary/WebObjects

Note: On Windows NT, you specify the location of NEXT_ROOT during the WebObjects installation process.

The **NextLibrary/WebObjects** directory, regardless of its location on your system, contains many of the resources required to run WebObjects applications. Within it, you'll find the following subdirectories:

- *AdaptorSource* contains the source code for the WebObjects adaptors, which you can compile for additional platforms.The source code for the WebObjects adaptors is only available in the WebObjects Pro and WebObjects Enterprise products.

- *Executables* contains **DefaultApp** or **EOFDefaultApp**, generic WebObjects application executables that respond to incoming requests using resources you provide.
- *cgi-bin* contains **WebObjects**, the CGI adaptor that handles communication between your web server and WebObjects applications using the Common Gateway Interface. Your HTTP server does not access the **WebObjects** program in this directory. Instead, it accesses a link or copy of it in the server's **<cgi-bin>** directory as described later in this section.
- *Examples* contains several sample WebObjects applications.

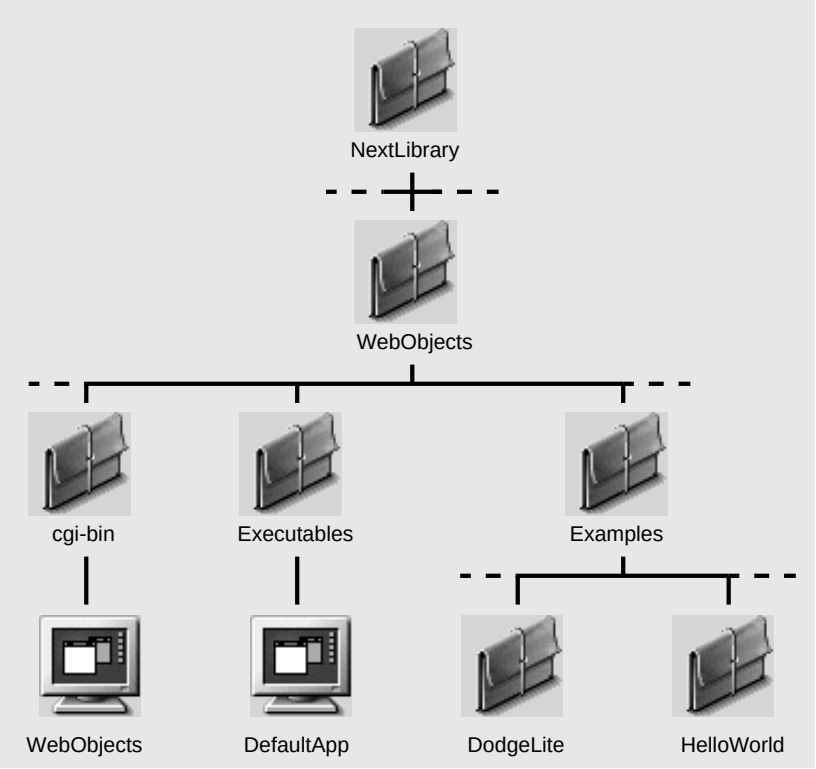

Figure 4. The Contents of **NextLibrary/WebObjects**

Your web server must be able to access the components installed in **NextLibrary/WebObjects** to run the examples and WebObjects applications you write. Therefore, after installing WebObjects, make sure your web server's directories contain the following links or copies:

- *<cgi-bin>/WebObjects* is a copy of or link to **NextLibrary/WebObjects/cgi-bin/WebObjects**.

- *<DocumentRoot>/WebObjects* is a new directory in which WebObjects application resources are installed.
- *<DocumentRoot>/WebObjects/Examples* is a copy of or link to **NextLibrary/WebObjects/Examples**.

Figure 5 shows the resources that should be present in your web server after the WebObjects installation process is completed. If any of these links or copies are missing, check the ReadMe file in **NextLibrary/WebObjects** for any steps in the procedure that you may have missed.

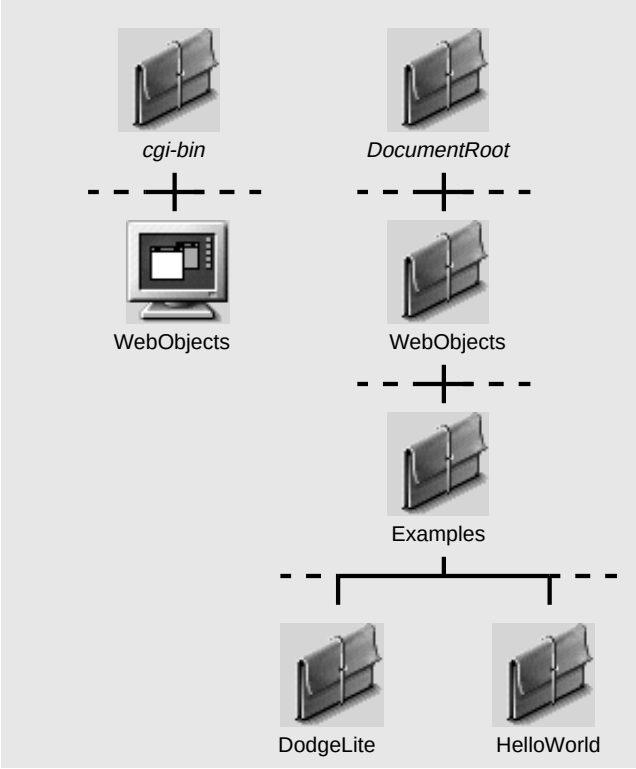

Figure 5. The Contents of Your Web Server's Directories After Installing WebObjects

The Ingredients of a WebObjects Application

To create a WebObjects application, you use the WebObjects library of classes that define an infrastructure for web applications. A WebObjects application uses instances of these classes to respond to requests received from a web browser. For instance, every WebObjects application contains an application object that receives requests and responds to them using application resources you provide.

A typical WebObjects application contains the following ingredients:

- Components that specify the content, presentation, and behavior of the application's pages
- An optional application script that creates and manages application-wide resources
- Optional compiled code that implements custom data and logic
- WebObjects classes that provide an infrastructure for the web application

To write a WebObjects application, you provide components and, optionally, compiled code.

Note: You can incorporate compiled code in a WebObjects application only if you have the WebObjects Pro or WebObjects Enterprise product.

If you write an application that includes compiled custom code, you must also provide an application executable by compiling and linking your application with the WebObjects library. See the "Compiling and Debugging WebObjects Applications" chapter for more information. Applications that don't include compiled code use the DefaultApp or EOFDefaultApp executable. In either case, the application executable receives incoming requests and responds to them using the components you provide.

Components

Components are a part of each WebObjects application that you write. Each component defines the content, presentation, and behavior of a page or portion of a page. You can write *scripted components* in WebScript (the WebObjects scripting language) or *compiled components* in Objective-C.

Scripted components generally consist of three files:

- An *HTML template* that specifies how the corresponding page looks. HTML templates contain markup elements that define the format for both static and dynamic page content.
- A *script file* that implements application behavior specific to the component. Script files declare variables for managing dynamic page content and actions that define responses to user requests.
- A *declarations file* that defines a mapping between the HTML template and script variables and actions.

Scripted and compiled components play the same role, and are created in essentially the same way. However, instead of using a script file, compiled components use an Objective-C class.

The files of a component are organized in a *component directory*. The name of the directory has the same base name as the name of the component, but the extension **.wo**. For example, the HelloWorld example has a WebScript component named **Main** and a corresponding component directory named **Main.wo**.

The template, script, and declarations files in the component directory also have the same base name, and each file type has its own extension. Template files have the extension **.html**, declarations have the extension **.wod**, and scripts have the extension **.wos**. Thus, the **Main** component has the files **Main.html**, **Main.wos**, and **Main.wod**, as shown in Figure 6. In addition to these three files, a component directory may also contain images and other resources used by the component.

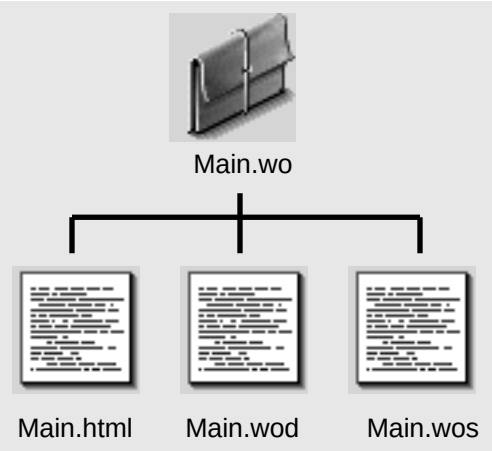

Figure 6. The Contents of a Component Directory

Note: Not all components represent an entire page. Some components represent only a portion of a page. You can extract common data and functionality from components for whole pages into smaller, reusable components. These smaller components can be nested as subcomponents inside a component representing a whole page. For more information, see the "Creating Reusable Components" chapter.

The Main Component

The component for the first page of a WebObjects application is generally named **Main**. When a user starts a session with a WebObjects application, he or she can specify the name of the first page, but it is uncommon to have to do so. If no page is specified, WebObjects applications look for a component named **Main** to represent the first page.

Where Components Go

Generally, you put the components of a WebObjects application in a directory with the same name as the application. For example, the HelloWorld example has a corresponding directory named **HelloWorld** that contains its component directories.

WebObjects application directories such as **HelloWorld** can go anywhere under the **<DocumentRoot>/WebObjects** directory. You can create application directories immediately under **<DocumentRoot>/WebObjects**, or you can create hierarchies of directories within **<DocumentRoot>/WebObjects** to organize application directories however you wish.

As shown in Figure 7, the HelloWorld application directory is located in **<DocumentRoot>/WebObjects/Examples**.

Reusable WebObjects components used by multiple applications aren't usually kept in application directories. Rather, they are located directly under **<DocumentRoot>/WebObjects** so they can be accessed by all the applications that use them.

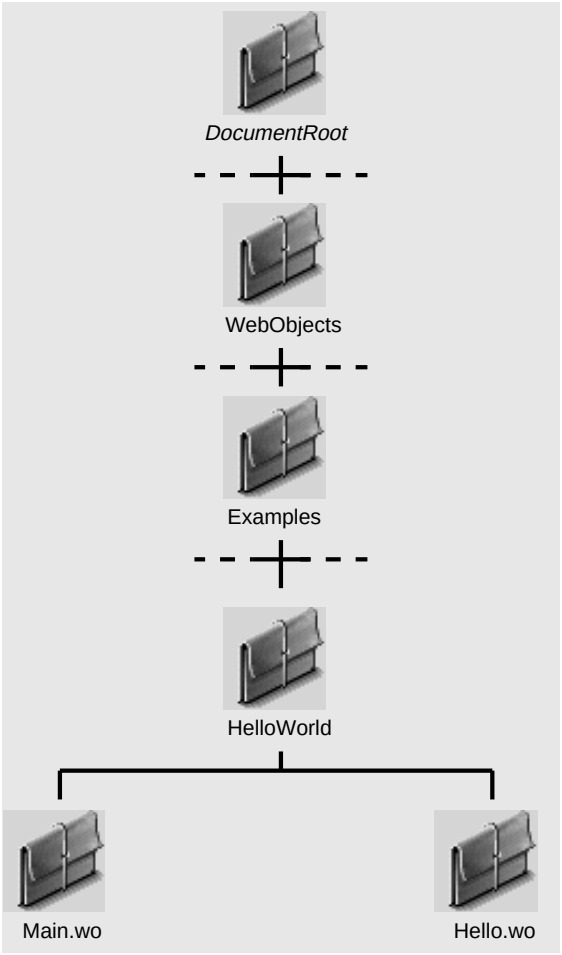

Figure 7. HelloWorld Application Directory

Optional Application Script

In addition to components, some applications have an application script that creates and manages application-wide resources. Application scripts are installed directly under the application directory, and they have the name **Application.wos**. For example, if HelloWorld had an application script, it would go in the directory **<DocumentRoot>/WebObjects/Examples/HelloWorld**. For more information on application scripts, see the section "The Role of Scripts in a WebObjects Application" in the "Using WebScript" chapter.

Optional Compiled Code

If you have WebObjects Pro or WebObjects Enterprise, you can incorporate custom compiled code into your WebObjects applications. Compiled code can play many different roles in a WebObjects application. For example, compiled code is commonly used to achieve the following objectives:

- *Encapsulate business data and logic.* You can design classes or modules that encapsulate behavior that is specific to the domain of your application. For example, an application that reserves conference rooms could use ConferenceRoom objects to manage room data like location and maximum occupancy. In addition, the ConferenceRoom class could implement business logic that denies requests for rooms that are already reserved. See the "Compiling and Debugging WebObjects Applications" chapter for more information on providing custom business logic using compiled code.

- *Improve performance.* By converting scripted components to Objective-C components, you can improve the performance of a WebObjects application. Script length, script complexity, and the number of nested subcomponents all affect a component's performance. As a component becomes more sophisticated, the performance advantage of writing it in Objective-C increases.

- *Provide custom dynamic elements.* A dynamic element is an object that uses an HTML template to generate HTML elements containing dynamic content. WebObjects provides elements such as buttons, hyperlinks, forms, text fields, and so on, which not only generate dynamic HTML but also bind custom logic to user actions. You can add to the collection of dynamic elements that come with WebObjects, or you can modify their behavior by replacing them with one of your own.

- *Interface with other software.* For example, you can use Enterprise Objects Framework to interface with a database server, and you can use NeXT's Distributed Objects system to implement communication between WebObjects applications and other software. See "The Enterprise Objects Framework Developer's Guide" for more information on Enterprise Objects Framework.

Application Executables

Applications that don't contain compiled code use DefaultApp or EOFDefaultApp. EOFDefaultApp is provided with WebObjects Enterprise, while DefaultApp is provided with the other products. Both applications play the same role, which is to use the resources you provide to respond to user requests. The difference between the two executables is that EOFDefaultApp

can use Enterprise Objects Framework to access industry-standard relational databases using an object-oriented interface. For more information on Enterprise Objects Framework, see "Enterprise Objects Framework Developer's Guide."

If you incorporate compiled code into your WebObjects application, you must also provide the application executable. You must write a **main()** function, compile the source code, and link it with the WebObjects library. See the "Compiling and Debugging WebObjects Applications" chapter for more information.

Where Application Executables Go

If you build an executable for a WebObjects application containing compiled code, you can put it in one of two places:

- The application directory under **<DocumentRoot>/WebObjects**. For example, if you created a compiled application called CheckFund and installed its components in **<DocumentRoot>/WebObjects/FinancialApps/CheckFund**, you would also put the executable in **<DocumentRoot>/WebObjects/FinancialApps/CheckFund**.
- The application executable directory under **NextLibrary/WebObjects/Executables**. The path of the application executable directory relative to **<NextLibrary>/WebObjects/Executables** must be the same as the path of the application directory relative to **<DocumentRoot>/WebObjects**. For example, instead of installing the CheckFund executable in the application directory as described above, you could put it in **NextLibrary/WebObjects/Executables/FinancialApps**.

Note: The default application executable—either DefaultApp or EOFDefaultApp—is located in **NextLibrary/WebObjects/Executables**.

Running WebObjects Applications

To run a WebObjects application, you start the application and then connect to it with a web browser.

Starting a WebObjects Application

Whether you're using the default application executable or your own WebObjects application executable, there are two ways to start a WebObjects application:

- Start it from the command prompt.
- Let a WebObjects adaptor autostart it.

The autostart mechanism has the advantage of convenience, but it's easier to debug an application that you start explicitly. In addition, you have more deployment options if you start WebObjects applications yourself. For more information on deployment options, see the "Serving WebObjects" document.

Starting WebObjects Applications from the Command Prompt

To start a WebObjects application:

1. Locate the application executable. If you don't have compiled code and haven't built a custom executable, use the default application executable located in **NextLibrary/WebObjects/Executables**.

2. Change directories to the directory in which the application executable is located.

3. Start the application by invoking the executable as follows:

   ```
   ApplicationExecutable RelativeApplicationDirectory
   ```

   You must provide at least one argument to the executable: the application directory relative to **<DocumentRoot>/WebObjects**. WebObjects applications use this argument to find application resources such as components and images. For example, the HelloWorld application directory is located in **<DocumentRoot>/WebObjects/Examples/HelloWorld**. Therefore, the relative application directory is **Examples/HelloWorld**, and the following command starts HelloWorld:

   ```
   DefaultApp Examples/HelloWorld
   ```

   You start a custom executable the same way. For example, the following command starts an application named Registration whose application directory is **<DocumentRoot>/WebObjects/Promos/Registration**:

   ```
   Registration Promos/Registration
   ```

Note: Additional configuration is required if the WebObjects application is not installed on the HTTP server. See the "Serving WebObjects" document for additional instructions.

Autostarting WebObjects Applications

When a WebObjects adaptor receives a request from a web server, it checks to see if the requested WebObjects application is running. If the application is not

running, the adaptor starts it automatically. Thus, if you don't start your application from the command prompt, WebObjects starts it for you.

When an adaptor receives a request for an application that isn't running, it looks in **NextLibrary/WebObjects/Executables** for the application executable. If it doesn't find an executable there, it looks in **<DocumentRoot>/WebObjects**. If the adaptor doesn't find an application in either location, it starts the default application executable instead. For more information, see the "Where Application Executables Go" section.

Note: To use the autostart feature of WebObjects, you must install WebObjects on your web server host.

Connecting to a WebObjects Application

To connect to a WebObjects application from a web browser, you open a URL with the following form:

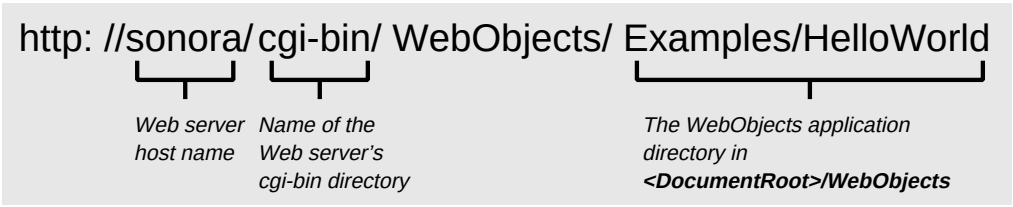

Figure 8. URL to Start a WebObjects Application

Writing HelloWorld

Writing a WebObjects application involves creating a component for each page in the application and installing the components in a directory that's accessible to WebObjects and your web server. Using HelloWorld as an example, this section explains how to perform these tasks. As an additional reference, the source code for HelloWorld is located in **NextLibrary/WebObjects/Examples**.

HelloWorld

The HelloWorld application consists of two pages. Figure 9 shows the first page.

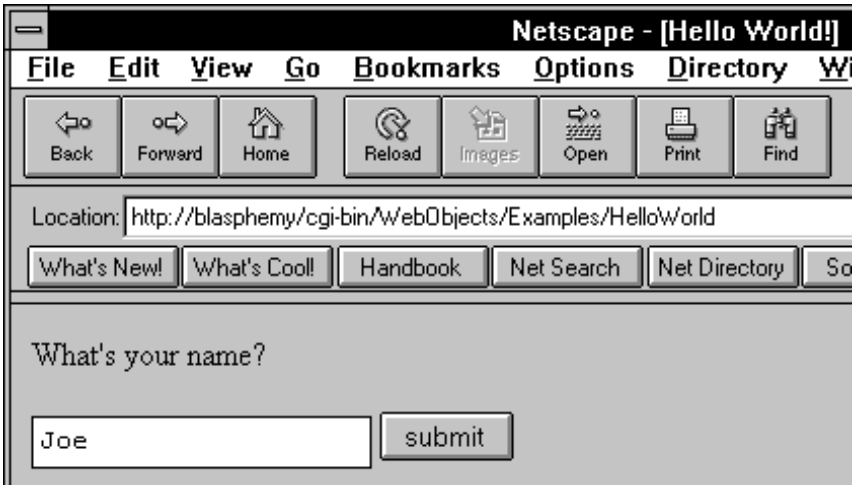

Figure 9. The First Page of HelloWorld

The presentation may vary slightly from browser to browser, but the page elements are the same regardless. The first page contains a single input field in which you type your name. Clicking Submit opens a new page with a personalized greeting. For instance, typing "Joe" and clicking Submit opens a page similar to the one in Figure 10.

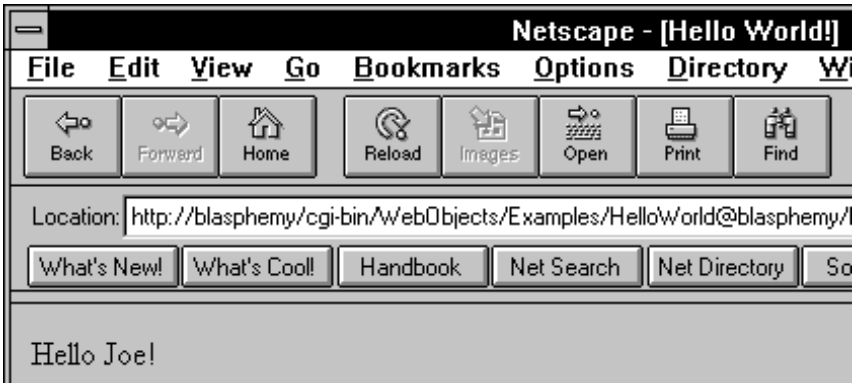

Figure 10. The Second Page of HelloWorld

The HelloWorld application has three simple requirements: get the name that's entered in the input field, dynamically generate the HTML required to display the message in the second page, and open the second page when Submit is clicked. More generally:

- Get user input.
- Determine dynamic page content and generate corresponding HTML.
- Specify page-to-page transitions.

HelloWorld has two components—**Main** and **Hello**. The following sections describe the files for the Main and Hello components of the HelloWorld application.

Main Component Files

The template for the Main page contains the following HTML elements:

```
<HTML>
<HEAD>
    <TITLE>Hello World!</TITLE>
</HEAD>
<BODY>
    <FORM>
    What's your name?
    <P>
    <WEBOBJECT NAME = "NAME_FIELD"><INPUT TYPE = "TEXT"></WEBOBJECT>
    <WEBOBJECT NAME = "SUBMIT_BUTTON"><INPUT TYPE = "SUBMIT"></WEBOBJECT>
    </P>
    </FORM>
</BODY>
</HTML>
```

The WEBOBJECT elements—a new kind of markup element introduced by WebObjects—are replaced with dynamically generated HTML when the HelloWorld application returns the Main page. The declarations file specifies the kind of objects that perform the HTML substitutions.

The declarations file for the Main page contains the following two declarations:

```
NAME_FIELD: WOTextField {value = nameString};
SUBMIT_BUTTON: WOSubmitButton {action = sayHello};
```

Each declaration corresponds to a WEBOBJECT element in the template. Each declaration declares an object—a WODynamicElement—to represent its corresponding WEBOBJECT element. The declaration specifies what kind of object to create and how to configure it. Specifically, a declaration associates a WODynamicElement with variables and methods defined in the corresponding script file. For more information on dynamic elements, see the chapter "Using Dynamic Elements."

The script for the Main page contains the following lines:

```
id nameString;
- sayHello
{
    id nextPage;
    nextPage = [WOApp pageWithName:@"Hello"];
    [nextPage setNameString:nameString];
    return nextPage;
}
```

Together, these three files (template, declarations, and script) establish what action to take when a user clicks Submit, which is to return the second page with a customized greeting. The files do two things:

- Store the name entered by the user.
- Return the Hello page.

Storing the Name

The declaration for the NAME_FIELD WEBOBJECT element:

```
NAME_FIELD: WOTextField {value = nameString};
```

specifies how to store the name entered by the user. It associates the NAME_FIELD element with the **nameString** variable declared in **Main.wos**.

The declaration specifies that a WOTextField object generates HTML for the NAME_FIELD element, and that the variable assigned to the **value** attribute of the WOTextField object**—nameString**— stores user input.

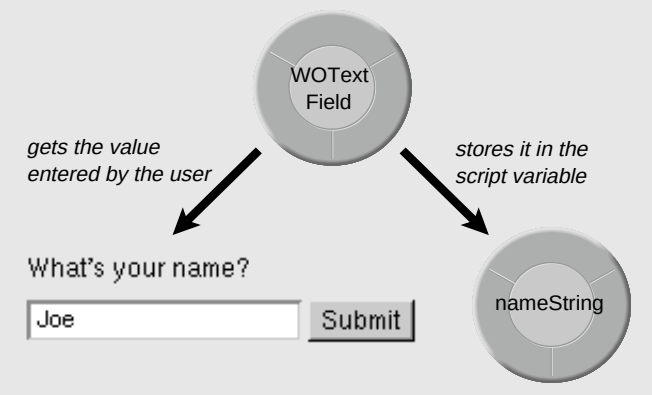

Figure 11. Getting and Storing a Value

For more information on WOTextFields, see the "WOTextField" section in the "Using Dynamic Elements" chapter.

Returning the Hello Page

The declaration for the SUBMIT_BUTTON WEBOBJECT element:

```
SUBMIT_BUTTON: WOSubmitButton {action = sayHello};
```

specifies how to return the Hello page. It associates the SUBMIT_BUTTON element with the **sayHello** method defined in **Main.wos**.

The declaration specifies that a WOSubmitButton generates HTML for the SUBMIT_BUTTON element, and that the action assigned to the WOSubmitButton object, **sayHello**, is invoked when a user submits the form.

The **sayHello** method:

```
id nameString;
- sayHello
{
    id nextPage;
    nextPage = [WOApp pageWithName:@"Hello"];
    [nextPage setNameString:nameString];
    return nextPage;
}
```

finds or creates a component object to represent the Hello page by sending a **pageWithName:** message to WOApp—the global variable representing HelloWorld's application object. If an object representing the Hello page doesn't exist, **pageWithName:** finds the **Hello.wo** component directory and creates a component object from it.

Next, **sayHello** sets the Hello component's **nameString** variable by sending a message to the Hello component. To access the variables declared in another script file, you use *accessor* methods. There are two kinds of accessor methods: *set* methods that set the value of a variable and *get* methods that return the value of a variable.

Set methods have the form **setVariableName:**, where **variableName** is the name of the script variable. For example, the Hello page declares the variable **nameString**, so the method **setNameString:** is automatically available to set its value. Notice that the "n" in the variable name is lowercase, but in the set method name, it's uppercase. The method name for a set method capitalizes the first letter of the variable name if it's not an uppercase letter, and then prepends the word "set" to it.

Get methods have the form **variableName**, where **variableName** is the name of the variable. For example, to get the value of the Hello component's **nameString** variable, you invoke a method of the same name. In WebScript, both set and get accessor methods are automatically available for all script variables.

Note: It is customary to start all variable names with lowercase letters.

After setting the Hello component's **nameString** variable, **sayHello** returns the Hello component.

Hello Component Files

The files for the Hello component establish how to generate the personalized greeting. The template for the Hello page contains the following HTML elements:

```
<HTML>
<HEAD>
    <TITLE>Hello World!</TITLE>
</HEAD>
<BODY>
    Hello <WEBOBJECT NAME = "NAME_STRING"></WEBOBJECT>!
</BODY>
</HTML>
```

The declarations file contains the following declaration of a WOString object to substitute the user's name for the NAME_STRING WEBOBJECT element:

```
NAME_STRING: WOString {value = nameString};
```

Like WOTextField objects, WOString objects have a **value** attribute. The WOString is responsible for getting the value in **nameString** and putting it in the corresponding page.

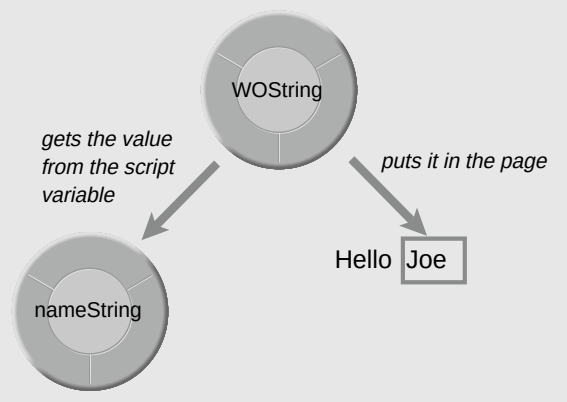

Figure 12. Getting a Value and Displaying it in a Page

The script for the second page contains only one line:

```
id nameString;
```

The **nameString** variable must be declared so it can be associated with the NAME_STRING element in the template file. Recall that **nameString** is set from the Main component in the **sayHello** method.

Behind the Scenes of a WebObjects Application

When a WebObjects application receives a request from a WebObjects adaptor, it processes the request in three phases. As shown in Figure 13, the application uses the page-to-component mappings defined in the declarations files to:

- Prepare for the request.
- Invoke an action.
- Generate a response page.

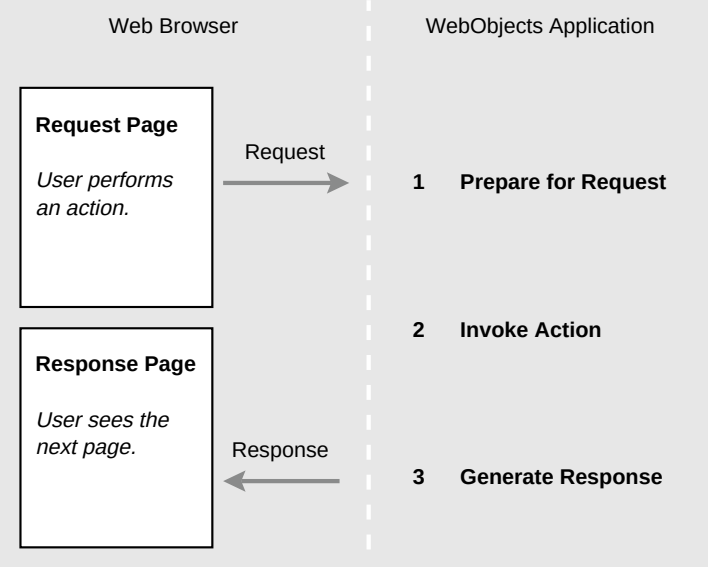

Figure 13. Request-Response Loop

The following sections describes what happens during each phase.

Prepare for Request

The application prepares for the request by updating variables in the request page—the page from which the request was made. If a user has provided any input that maps to a component variable, the application assigns the new value to the variable. For example, recall what happens when a user clicks Submit in the first page of HelloWorld: the application gets the value from the NAME_FIELD text field and assigns it to the **nameString** variable defined in the Main component.

Invoke Action

After preparing for the request, the application determines whether or not the user has triggered an action. If an action has been triggered—for example, if the user clicked a button or a hyperlink—the application invokes the action method that corresponds to what the user did. For example, clicking Submit in HelloWorld has the effect of invoking the **sayHello** action method. An action method returns an object to represent the *response page*—the page that is sent back to the web server. **sayHello** returns an object to represent the Hello page. If the user does not trigger an action, the object representing the request page also represents the response page.

Generate Response

It is the responsibility of the response page object to generate the HTML for the response. Using the HTML template and declarations file, a component generates the HTML that is eventually displayed in the user's web browser. For example, after Submit is clicked in HelloWorld and **sayHello** returns an object to represent the Hello page, the Hello page object generates the resulting personalized greeting.

Summary

How Do I Connect a WebObjects Application to the Web?

To connect a WebObjects application to the Web, you need the following:

- *An HTTP server*. You can use any HTTP server that uses the Common Gateway Interface (CGI) or the Netscape Server API (NSAPI).
- *A WebObjects adaptor*. A WebObjects adaptor connects WebObjects applications to the Web by acting as an intermediary between web applications and HTTP servers. Adaptors insulate applications from server interfaces by handling all server communication. Simply by switching adaptors, you use a different HTTP server and a different server interface without modifying application code.

  WebObjects provides an adaptor for servers that use the Common Gateway Interface (CGI) and an adaptor for the Netscape Commerce Server that uses the Netscape Server API (NSAPI).
- *A WebObjects application executable.* An application executable receives incoming requests and responds to them, usually by returning a dynamically generated HTML page. An executable can be a compiled application linked with the WebObjects library, or it can be one of the two default WebObjects application executables: DefaultApp or EOFDefaultApp.

  If the application doesn't contain any compiled code, you use a default application. Instead of building your own executable, you implement all the application logic in script files. The default applications use these scripts to respond to requests.

What's in a WebObjects Application?

A typical WebObjects application contains the following ingredients:

- Components that specify the content, presentation, and behavior of the application's pages
- An optional application script that creates and manages application-wide resources
- Optional compiled code that implements custom data and logic
- WebObjects classes that provide an infrastructure for the web application

What Parts Do I Write?

You write the following parts of a WebObjects application:

- Components consisting of HTML templates, script files, and declarations files
- An optional application script
- Optional compiled code

How Do I Run a WebObjects Application?

To run a WebObjects application, you open a URL with the following form:

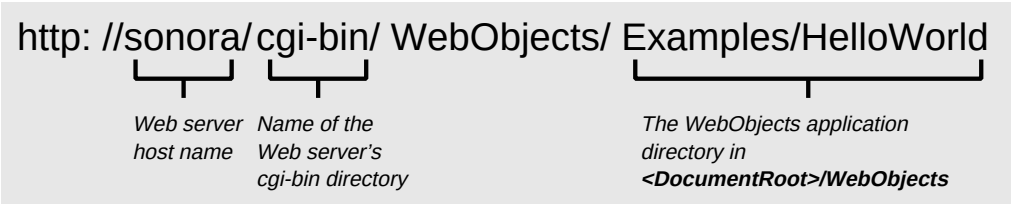

Figure 14. URL to Start a WebObjects Application

What Happens Behind the Scenes?

Behind the scenes of a running WebObjects application, the application enters a request-response loop each time it receives a request. In the request-response loop, a WebObjects application uses the page-to-component mappings defined in declarations files to:

- Prepare for the request.
- Invoke an action.
- Generate a response page.