Using Variables

If tags and functions are the building blocks of ColdFusion, variables are the building blocks of applications. In Step 1, “The Basics,” we briefly touched on the concept of using variables. In this step, we will take a closer look at variables, how they are created, the types of information they can contain, and how we can begin to harness their power in our web applications.

Understanding Variables

According to `www.whatis.com`, a *variable* is "a quantity capable of assuming any of a set of values." However, as mentioned in Step 1, you can think of a variable as a container that holds information. You create variables by giving them a name and assigning them some value. ColdFusion Server uses server memory to store these values until either you call for them by using the variable name or they are no longer needed, in which case they are deleted.

Two broad categories of variables are used in ColdFusion: simple variables and complex variables. In this step, we will focus mainly on simple variables. Complex variables will be discussed in Step 10, "Using Lists, Arrays, and Structures."

Data Types

ColdFusion variables can hold many different data types. A data type indicates what kind of information is being stored. ColdFusion variables can store data types, such as numbers, text strings, dates, times, and Boolean values, just to name a few. See Table 2.1 for further information on various ColdFusion data types.

TABLE 2.1 ColdFusion Data Types

Data Type	Description
Integers	These are whole numbers (numbers without anything to the right of the decimal point), such as 0, –17, and 105.
Real numbers	Real numbers, also referred to as floating-point numbers, are numbers that might include a decimal value, such as 3.17, –0.175, and 25.387.
Strings	Strings are a sequence of symbols, such as letters, numbers, and special characters. Strings are enclosed by either single or double quotes. For example, `"ColdFusion Mentor"`, `'user@anysite.com'`, and `"10"`.
Booleans	A Boolean value is either `True` or `False`. For example, `<CFSET UserIsLoggedIn = True>`. Boolean values can be expressed in a number of ways. Negative values can be `False`, `No`, or `0`. Positive vales can be `True`, `Yes`, or `1` (or any nonzero number).
Date-time	Date-time values can be date only, time only, or a combination of both the date and the time, such as the result of the `Now()` function.

Data Type	Description
Lists	A list is a string that consists of multiple entries separated by some type of delimiter. The comma is the default delimiter, but others can be specified. Lists will be explained in Step 10. An example might be `<CFSET Colors="Red,White,Blue">`.
Arrays	This complex data type stores information in a table-like structure of rows and columns. Arrays will be explained in Step 10.
Structures	This complex data type stores information in a series of key-value pairs. Structures will be explained in Step 10.
Queries	This complex data type holds the results of a database query.
Binary	Binary data is raw data, such as the contents of a file or an executable program.
Object	These are complex object types that are created using the `<CFOBJECT>` tag. They can include things like COM, Java, and CORBA objects.

You create most ColdFusion variables simply by giving them a name and assigning them a value. We have already seen one way to accomplish this by using the `<CFSET>` tag. For example, the line of code

```
<CFSET Age="Thirty">
```

would automatically create a variable called `Age` and assign it a string value of `"Thirty"`.

Many other programming languages require you to set the type of value that a variable will hold before you assign it a value. For example, you would have to declare the `Age` variable to be a string variable before you could assign a string value, such as `"Thirty"` to it. However, ColdFusion variables are *typeless*, meaning you are not required to assign a specific data type to a variable name. ColdFusion automatically evaluates variable values when they are used in operations to determine how the variable should be used. For a further explanation of this, see the following sidebar.

Is It a Number or Not?

Because ColdFusion variables are typeless, there can be some confusion about the value of some strings. For example, the code

```
<CFSET Age="30">
```

would set the value of the `Age` *variable to the string value of* `"30"` *(note the quotation marks), not the integer value of* `30`*. In stricter programming languages, a simple mathematical operation using this value—such as ("30"–21—would cause an error because you cannot subtract a number*

Continues

from a string. However, ColdFusion is a bit more clever than that. ColdFusion uses what is called operation-driven evaluation*; this means that when ColdFusion sees a mathematical operation, such as subtraction or multiplication, it automatically tries to convert all the operands (elements in the equation) into numbers. So, in ColdFusion, even though the* `Age` *variable contains a string, instead of throwing an error, the code*

```
<CFSET Age="30">
<CFSET YearsOverTheHill= Age - 21>
<CFOUTPUT>#YearsOverTheHill#</CFOUTPUT>
```

would output the numerical value of 9.

Pretty clever, huh?

Conversely, if you use the following code to set the `DaysTillXmas` *variable to the integer value of* `30` *(note that there are no quotation marks), it will be treated as a number.*

```
<CFSET DaysTillXmas=30>
```

If we then try to combine it with text (as seen in the following code), ColdFusion will be smart enough to convert it to the string value of `"30"` *rather than the number. For example, the code*

```
<CFSET DaysTillXmas=30>
<CFSET Message="Only #DaysTillXmas# to go!">
<CFOUTPUT>Hey kids, Santa's coming. #Message# </CFOUTPUT>
```

would output `Hey kids, Santa's coming. Only 30 days to go!`

Although ColdFusion is pretty clever, there might be times when you want to make sure that a variable gets evaluated as a number and not a string or vice versa. A couple of ColdFusion functions can help you do this. The `Val()` *function will evaluate a string into a number (if possible). For example, the following line of code would convert the* `Age` *variable into a number:*

```
<CFOUTPUT>This is the number #Val(Age)#</CFOUTPUT>
```

To convert a number into a string, you can use `ToString()`*. For example, the following line of code converts an integer value of* `30` *into the string* `"30"`*.*

```
<CFOUTPUT>This is the string #ToString(30)#</CFOUTPUT>
```

For more information on various ColdFusion functions, check out the "Language Reference" section of this book's web site at `www.LearnColdFusionMX.com`*.*

Variable Scopes

As we've already seen, variables can contain varying data types. Variables also can come in varying scopes. A *scope* is the context in which a variable exists, and it determines how long its data persists. To put it simply, a variable's scope determines where it lives, how you access it, and how long its data hangs around.

So far, we have just been using simple local variables. A *local variable* lives only in the template in which it was created. For example, if we use `<CFSET Message="I am a local variable">` to set the value of a variable called `Message` in a ColdFusion template called `page1.cfm`, we can display the value of that variable anywhere in that template simply by using `<CFOUTPUT>#Message#</CFOUTPUT>`. However, if we were to attempt to use `<CFOUTPUT>#Message#</CFOUTPUT>` in another template called `page2.cfm`, ColdFusion would return an error because the `Message` variable does not exist in the `page2.cfm` template. If you want the value of a variable to persist from one template/page to another, you must use a different scope.

NOTE

The value of a local variable will also be available to any included templates that are called using `<CFINCLUDE>`. Because using `<CFINCLUDE>` is essentially the equivalent of copying and pasting code into the calling template, local variable values set on the calling template will be available for use in the code of the included page code as well.

Table 2.2 lists some of the different types of scopes we will be using throughout this book. For a complete list of scopes and their descriptions, see the "Language Reference" section of this book's web site at `www.LearnColdFusionMX.com`.

TABLE 2.2 ColdFusion Variable Scopes

Scope	Description
Variables (local)	As previously described, a local variable is only available on the page in which it was created and any included pages.
Form	This scope is for variables passed via HTML or ColdFusion forms using the `post` method of the form. See the section "Passing Values with Forms" later in this step.
URL	This scope contains variables passed via parameters added to the end of a URL, such as `www.anysite.com?id=1175`. See the section "Passing Values Via the URL" later in this step.
CGI	This scope is for environment variables that automatically accompany each page request and server response, such as browser type and server name.
Cookie	This scope is for variables used to read and write browser cookies.
Client	This scope contains variables associated with a particular client (user). They are maintained as a user moves from page to page and are available over multiple browser sessions, or visits. For more information on client variables, see Step 8, "ColdFusion Application Framework."

Continues

TABLE 2.2 ColdFusion Variable Scopes (Continued)

Scope	Description
Session	This scope contains variables associated with one client for one session or visit. These variables time out when a user closes their browser or after periods of inactivity. For more information on session variables, see Step 8.
Application	This scope contains variables that are available throughout one entire application (web site) on a server. For more information on application variables, see Step 8.
Server	This scope contains variables that are set for a particular ColdFusion server and that are available to all applications (web sites) on that server. For more information on server variables, see Step 8.

Naming Variables

When it comes time to create your variables, there are some naming rules you must follow:

- Variable names must start with a letter, an underscore character, or a Unicode currency symbol. Examples of valid variable names are `FirstName`, `_department`, or `$userID`.
- After the starting character, variable names can contain any number of letters, underscores, numbers, or Unicode currency symbols.
- Variable names must not contain spaces.
- Simple variable names must not contain the dot (.) character. For example, setting a variable name to `Sales.Manager` will actually create a structure. (For more information on structures, see Step 10.)

NOTE

A database query result is treated as a variable and must not have the same name as any local variables in the same ColdFusion template. See Step 3, "Databases and SQL," for more information on queries.

Provided you follow the preceding rules, you can call your variables just about anything you like. However, most developers follow some sort of naming convention for their variables. Naming conventions make your code more readable and easier to work with, for both yourself and other team members.

For example, ColdFusion variables are not case sensitive. This means that as far as ColdFusion is concerned, `firstname`, `firstName`, and `FirstName` all refer to the same variable. However, most developers use a standard convention with regard to case usage. Java developers, for instance, usually start variable names with a lowercase character and use an uppercase character to mark the start of each new word in the variable name. The variable name `employeeStartDate` would be an example of this convention. Many ColdFusion developers use a similar convention but also start variable names with an uppercase character, as in `EmployeeStartDate`. We will be following the latter convention throughout this book.

Here are some additional guidelines for working with variable names:

- Avoid using abbreviations. The variable name `EmployeeStartDate` will make sense when you revisit your code in six months time; the variable name `ESD` probably will not.
- Use a consistent naming convention and capitalization scheme for variable names.
- When using forms to update database fields, make the form field names match the database field names. (For more information, see Step 6, "Updating, Inserting, and Deleting Database Records.")
- Always use the variable's scope prefix for any scope other than the Variables/Local scope.

Many ColdFusion developers also like to use variable name prefixes to describe the type of data that a variable contains. For more information on data type prefixes, see the following sidebar.

"What's in a Name?"

Many developers like to use prefixes with their variable names to describe the type of information that variable is meant to contain. This way, when any developer comes across a variable somewhere in code, they can instantly tell what type of data it is meant to contain.

Table 2.3 lists some common prefixes.

TABLE 2.3 Data Type Prefixes

Data Type	Prefix	Example
Array	a	`aAnArrayVariable`
Boolean	b	`bChecked`, `bFlag`
Date	d	`dEmployeeStartDate`
Date-time	dt	`dtLastVisit`

Continues

TABLE 2.3 Data Type Prefixes (Continued)

Data Type	Prefix	Example
Integer	`i`	`iAge`, `iDaysTillXmas`
List	`l`	`lFavoriteColors`
Number	`n`	`nValueOfPi`
Query	`q`, `qry`	`qProducts`, `qryDepartments`
String	`s`	`sFirstName`, `sLastName`
Time	`t`	`tStartTime`, `tFinishTime`
Structure	`st`	`stMyStructure`

Setting Values

In ColdFusion, you create variables by giving them a name and assigning them a value. In this step, we will concentrate on the most commonly used ways to assign a value to a variable.

Using `<CFSET>`

As we have already seen, the `<CFSET>` tag can be used to assign a value to a variable. The `<CFSET>` tag is quick and easy to use and has no attributes other than the variable name and its assigned value. For example, the line of code

```
<CFSET Qty=2>
```

would create a variable named `Qty` and would set the value of that variable to the numerical value of `2`. If the variable already exists, you can use `<CFSET>` to reassign the variable a different value. For example, the code

```
<CFSET Qty=2>
<CFSET Qty=3>
<CFOUTPUT>You have ordered #Qty# jars.</CFOUTPUT>
```

would first set the value of the `Qty` variable to `2`. Because the `Qty` variable already exists, the second line of code would then reassign the variable's value to `3`. The third line of code would then output `You have ordered 3 jars.`

You can also perform calculations inside the `<CFSET>` tag when assigning a value. For example, the code

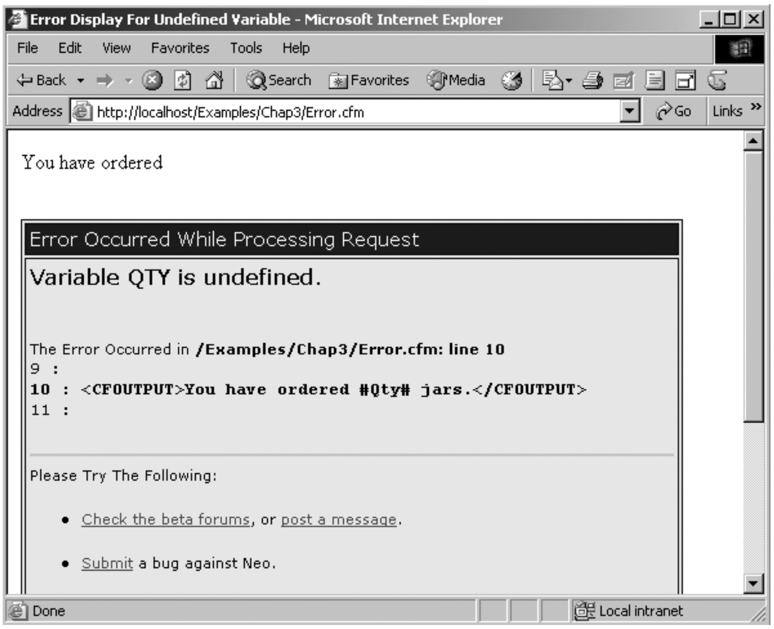
```
<CFSET Qty=2>
<CFSET Qty=Qty + 1>
<CFOUTPUT>You have ordered #Qty# jars.</CFOUTPUT>
```

would produce the same result as the previous code. The second line would take the current value of the `Qty` variable (in this case `2`), add `1` to it, and assign that value (`3`) back to the `Qty` variable. This is a common way to increment the value of a variable.

Using <CFPARAM>

Trying to use or output a variable that has not yet been created will cause an error. Figure 2.1 shows you the error output that ColdFusion will return if you try to use the following line of code without first creating the `Qty` variable.

```
<CFOUTPUT>You have ordered #Qty# jars.<CFOUTPUT>
```


Error Display For Undefined Variable - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://localhost/Examples/Chap3/Error.cfm
Go Links
Done
Local intranet

```
You have ordered

Error Occurred While Processing Request
Variable QTY is undefined.

The Error Occurred in /Examples/Chap3/Error.cfm: line 10
9 :
10 : <CFOUTPUT>You have ordered #Qty# jars.</CFOUTPUT>
11 :

Please Try The Following:

  • Check the beta forums, or post a message.
  • Submit a bug against Neo.
```

FIGURE 2.1

The error message that results from failing to create variable.

Because we want to avoid errors, it is good practice to check whether a variable exists before you try to use it. Normally, the code would flow something like this.

1. Check to see if a variable currently exists.
2. If it does, use the variable's current value.
3. If the variable does not exist, assign it some default value.
4. Display or use the variable.

The `<CFPARAM>` tag encapsulates all this logic into one line of code. The `<CFPARAM>` tag takes three attributes, as outlined in Table 2.4.

TABLE 2.4 <CFPARAM> Tag Attributes

Attribute	Description
`NAME` (required)	The name of the variable to be checked, including scope prefix.
`DEFAULT` (optional)	The default value you would like to assign to the variable if it does not already exist.
`TYPE` (optional)	The type of data that the variable should contain.*

**(See the "Language Reference" section of* `www.LearnColdFusionMX.com` *for possible values.)*

For example, in the code

```
<CFPARAM NAME="Qty" DEFAULT=1>
<CFOUTPUT>Quantity ordered is #Qty#.</CFOUTPUT>
```

the first line would check for the existence of a variable named `Qty`. Because this variable has not yet been created, it will be created by the `<CFPARAM>` tag and will be assigned the default value of `1`. The second line of code then outputs `Quantity ordered is 1.`

If a variable with the same name already exists, the `<CFPARAM>` tag is ignored. For example, in the code

```
<CFSET Qty=2>
<CFPARAM NAME="Qty" DEFAULT=1>
<CFOUTPUT>Quantity ordered is #Qty#.</CFOUTPUT>
```

the first line of code sets the value of the `Qty` variable to the numeric value of `2`. On the second line, the `<CFPARAM>` tag checks for the existence of a variable named `Qty`. Because this variable already exists, the `<CFPARAM>` tag is ignored, and the program flow continues on to the third line, which outputs `Quantity ordered is 2.`

The `<CFPARAM>` tag is a quick and simple way to check for the existence of variables before you attempt to use them. It will come in very handy when passing variables from one page to the next, such as when processing form data.

Retrieving Values

To use variables in a particular scope, you add the name of the scope to the beginning of the variable's name and separate the two with a dot. For example, to output a form variable called `LastName`, you would use the following line of code:

```
<CFOUTPUT>Last name submitted: #FORM.LastName#</CFOUTPUT>
```

This is referred to as *dot notation*. If you leave off the scope name, your code will probably still work (most of the time), but it will be less efficient. By adding the scope prefix to the variable name, you are telling ColdFusion where to look for that variable. If there is no scope prefix, ColdFusion must search through all of the scopes until it finds the variable. If ColdFusion comes across a variable without a scope prefix, it will look through the various scopes in the following order of precedence.

1. Variables (local)
2. CGI
3. URL
4. Form
5. Cookie
6. Client

As you can see, you can save processing time by telling ColdFusion that a particular variable lives in the Form scope rather than letting ColdFusion search through all scopes.

There is a pitfall to be aware of when using variables. It is possible to create two variables with the same name but different scopes in the same template. For example, imagine you have a template that processes data submitted by a form and you inadvertently create a local variable in that template with the same name. In the code

```
<!--- check for value from form --->
<CFPARAM Name="FORM.Qty" Default=10>
<!--- set local variable --->
<CFSET Qty=2>
<!--- output variable --->
<CFOUTPUT>Quantity ordered is #Qty#.</CFOUTPUT>
```

we use `<CFPARAM>` to check for the existence of an incoming Form variable called `Qty`. If it does not exist, we set the value to `10`. Next, we create (by mistake) a local variable called `Qty` and assign it the value of `2`. We have now ended up with two variables with the same name. We then use `<CFOUTPUT>` to display the value of the variable. However, because we have not specified a scope prefix in the `<CFOUTPUT>` block, ColdFusion starts to search for the variable using the order of precedence previously listed. It first searches the Variables (local) scope, finds a `Qty` variable there, and outputs the value `2`.

If we wanted to make sure we used the value from the submitted form, we would change the last line of code to read as follows:

```
<CFOUTPUT>Quantity ordered is #FORM.Qty#.</CFOUTPUT>
```

ColdFusion would then know exactly in which scope to look for the `Qty` variable. Our template would then display the value from the submitted form, or the default value of `10`.

The moral of the story is to always use prefix scopes on all variables other than local variables.

NOTE

You might have noticed that the first scope listed in the search order is the Variables scope, which is the scope for local variables. So, if we wanted to be as correct as possible when using local variables, our code would read something like this:

```
<CFSET Variables.Age="30">
<CFSET Variables.YearsOverTheHill= Age - 21>
<CFOUTPUT>#Variables.YearsOverTheHill#</CFOUTPUT>
```

However, because the Variables scope is the first one searched in the order of precedence, there is no real savings in processing time, and most developers leave off the scope prefix for local variables. We are a lazy bunch when we can get away with it.

Accessing CGI Variables

Table 2.2 mentioned a Variable scope called CGI, which is an abbreviation for *common gateway interface*. Variables in this scope are read-only and are created every time a template is requested from, or returned to, the server. These variables can be very useful and can include things like the user's browser type, his IP address, the web page he has just come from, as well as server information, such as the server's name and what type of software it is running. To access CGI variables, you simply use the CGI scope prefix and the variable's name. For a complete list of CGI variable names, see the "Language Reference" section of `www.LearnColdFusionMX.com.`

Example 2.1: Using CGI Variables

In this example, we will create a ColdFusion template to display some commonly used CGI variables.

1. Open your editor and type the code shown in Listing 2.1, or you can open the `CGIVariables.cfm` file from the `CompletedFiles\Examples\Step02` folder.

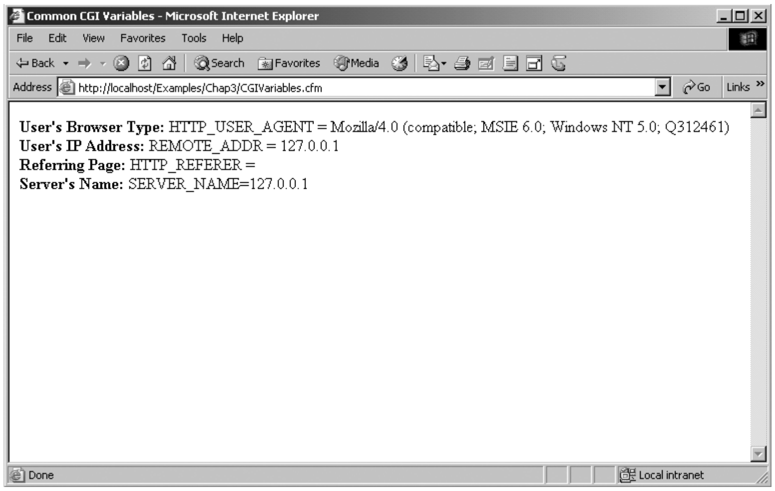
LISTING 2.1 Commonly Used CGI Variables

```
<!---
File:        CGIVariables.cfm
Description: Displays some common CGI variables
Author:
Created:
--->

<HTML>
<HEAD>
      <TITLE>Common CGI Variables</TITLE>
</HEAD>

<BODY>
<CFOUTPUT>
<!--- display some commonly used CGI variables --->
<B>User's Browser Type:</B> HTTP_USER_AGENT = #CGI.HTTP_USER_AGENT#<BR>
<B>User's IP Address:</B> REMOTE_ADDR = #CGI.REMOTE_ADDR#<BR>
<B>Referring Page:</B> HTTP_REFERER = #CGI.HTTP_REFERER#<BR>
<B>Server's Name:</B> SERVER_NAME=#CGI.SERVER_NAME#<BR>
<B>Server's Port:</B> SERVER_NAME=#CGI.SERVER_PORT#<BR>
</CFOUTPUT>

</BODY>
</HTML>
```

2. Save this file into your `Examples\Step02` directory as `CGIVariables.cfm`.
3. Use a URL to browse to the page. For example, `http://localhost/Examples/Step02/CGIVariables.cfm`.
4. You should see something similar to the page displayed in Figure 2.2.
5. To see `HTTP_REFER` information, set up a hyperlink to `CGIVariables.cfm` on another page and use the link to jump to `CGIVariables.cfm`.

FIGURE 2.2

`CGIVariables.cfm` *browser display.*


Common CGI Variables - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://localhost/Examples/Chap3/CGIVariables.cfm
Go Links
Done
Local intranet

User's Browser Type: HTTP_USER_AGENT = Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.0; Q312461)
User's IP Address: REMOTE_ADDR = 127.0.0.1
Referring Page: HTTP_REFERER =
Server's Name: SERVER_NAME=127.0.0.1

Passing Values

Hypertext Transfer Protocol (HTTP), the underlying protocol used by the World Wide Web, is a stateless protocol. This means that each page request made to a web server is independent from any others and has no information about anything that has occurred previously. Consequently, no information is kept about what data has been entered or what events have occurred prior to the current page request.

The capability to track *state*, which includes such things as user actions or entered information, is the cornerstone of any modern application. For us to be able to transform our old, static web site into a responsive, data-driven web application, we have to be able to track and pass information from one page to the next.

One of the easiest ways to track information is to store it in a variable and then pass that variable between pages. This can be done in a number of ways. In this step, however, we will focus on using forms and web-page URLs to pass variables.

Passing Values with Forms

If you have been using the Internet for any length of time, you have probably filled out a web form or two. This is one of the most common ways to pass information between web pages. Information entered into a form is packaged up and sent to the web server. The web server then uses a script of some sort to process that information and do something with it, such as email it to the webmaster, pass it to another page, or enter it into a database.

Using forms to pass information will be covered in detail in Step 5, "Using Forms with ColdFusion MX."

Passing Values Via the URL

Another simple way to pass information between web pages is to append it to the end of the URL being requested. This is done by adding a question mark after the filename and then adding a variable name and value. For example, with the URL

```
http://www.anysite.com/Details.cfm?Name=John
```

we are requesting a page called `Details.cfm` from `www.anysite.com`, and we are also passing a variable called `Name` that has been assigned a value of `John`. Variables passed using this method become variables in the URL scope.

To retrieve this variable and use its value in the `Details.cfm` template, simply call the variable using its name and the URL scope prefix. For example, the line of code

```
<CFOUTPUT>Welcome #URL.Name#</CFOUTPUT>
```

would look in the URL string for a variable called `Name` and retrieve its value, in this case `John`.

If you want to send more than one variable via the URL, you can append additional variable/value pairs to the end of the URL using the ampersand (`&`) symbol. For example, the URL

```
http://www.anysite.com/Details.cfm?Name=John&Dept=Sales
```

would pass our `Name` variable again as well as another variable called `Dept`. As you might have guessed, you would access that variable's value in the same way we used our `Name` variable, by using the variable's name (`Dept`) and the URL scope prefix as illustrated here:

```
<CFOUTPUT>
      Name: #URL.Name# <BR>
      Dept: #URL.Dept#
</CFOUTPUT>
```

No Space Allowed

When passing information via URL variables, we have to be careful that those variables do not contain spaces. URLs are not allowed to contain spaces, so consequently, if we tack on a URL variable that has spaces in it, we will break that rule. For example, if we use a URL variable, such as `TemplateName.cfm?Name=#UsersName#` *and the* `#UsersName#` *variable happens to contain the value* `John Smith`*, we could be in trouble. The resulting URL would be* `TemplateName.cfm?Name=John Smith`*. This URL could cause us problems, depending on the browser type and version being used.*

Continues

To get around this problem, we can use the `URLEncodedFormat()` *function. This function encodes URLs and converts non-URL-friendly characters, such as spaces, into URL-safe hexadecimal escape characters. For example, if we use* `TemplateName.cfm?Name=#URLEncodedFormat(UsersName)#` *instead of the previous example, we would end up the a URL that looks like* `TemplateName.cfm?Name=John%20Smith`.

To turn that variable back into its original form, you use the `URLDecode()` *function. So, on the* `TemplateName.cfm` *template, we could use code like*

```
<CFOUTPUT>Hello, #URLDecode(URL.Name)# </CFOUTPUT>
```

to display `Hello, John Smith`.

NOTE

Using URL variables is a very common technique for passing information from one template to the next. However, this method should never be used for sensitive information, such as usernames or passwords. Because this information is attached to the URL, it is there for anyone to see. In addition, this information is also recorded if the user happens to take a bookmark of the URL or add it to her favorites list.

Example 2.2: Passing Variables Via the URL

In this example, we are going to use two pages to demonstrate how to pass variables via a URL.

1. Open your editor and type the code shown in Listing 2.2, or you can open the `Page1.cfm` file from the `CompletedFiles\Examples\Step02` folder.

LISTING 2.2 `Page1.cfm`

```
<!---
File:        Page1.htm
Description: Demonstrates passing URL variables
Author:
Created:
--->

<HTML>
<HEAD>
      <TITLE>Pick a color any color</TITLE>
</HEAD>
<BODY>
      <H2>What is your favorite color?</H2>
      <A HREF="Page2.cfm?Color=Blue">Blue</A><BR>
      <A HREF="Page2.cfm?Color=Red">Red</A><BR>
      <A HREF="Page2.cfm?Color=Green">Green</A><BR>
      <A HREF="Page2.cfm?Color=Yellow">Yellow</A>
</BODY>
</HTML>
```

2. Save the file as `Page1.htm` into your `Examples\Step02` folder.
3. In your text editor create a new file and type the code shown in Listing 2.3, or you can open the `Page2.cfm` file from the `CompletedFiles\Examples\Step02` folder.

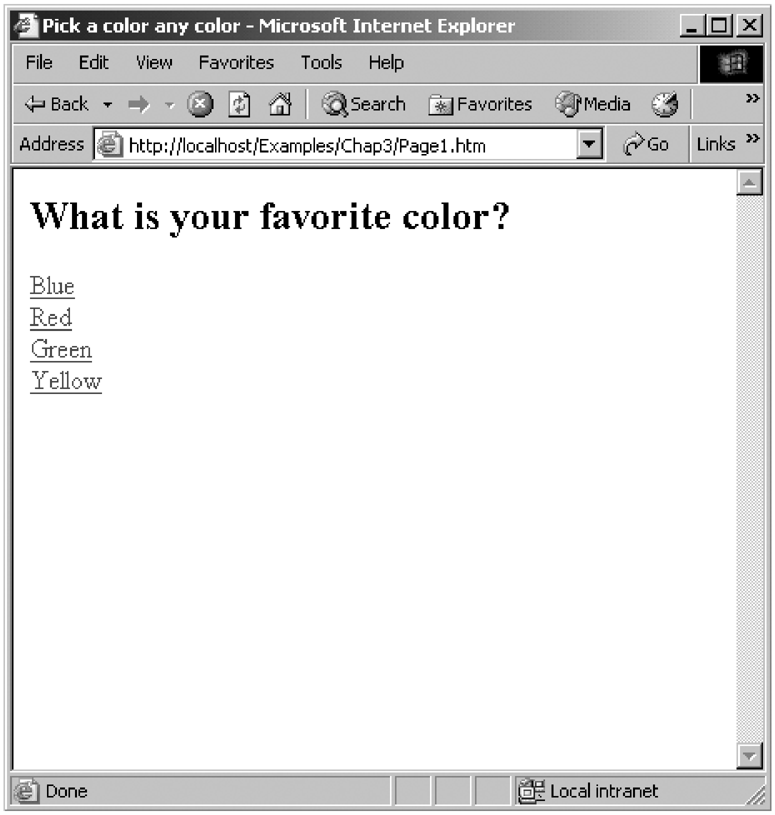
LISTING 2.3 `Page2.cfm`

```
<!---
File:        Page2.cfm
Description: Demonstrates the use of URL scope variables
Author:
Created:
--->
<HTML>
<HEAD>
      <TITLE>That is so cool</TITLE>
</HEAD>
<BODY>
      <H2>That is so cool!</H2>
      <H2><CFOUTPUT>#URL.Color# is my favorite too.</CFOUTPUT></H2>
      <A HREF="Page1.htm">&lt;&lt;back</A>
</BODY>
</HTML>
```

4. Save the file as `Page2.cfm` into your `Examples\Step02` folder.
5. Open a browser and browse to `Page1.htm`. Your URL should be something similar to `http://localhost/Examples/Step02/Page1.htm`. You should see a page similar to the one shown in Figure 2.3.

FIGURE 2.3

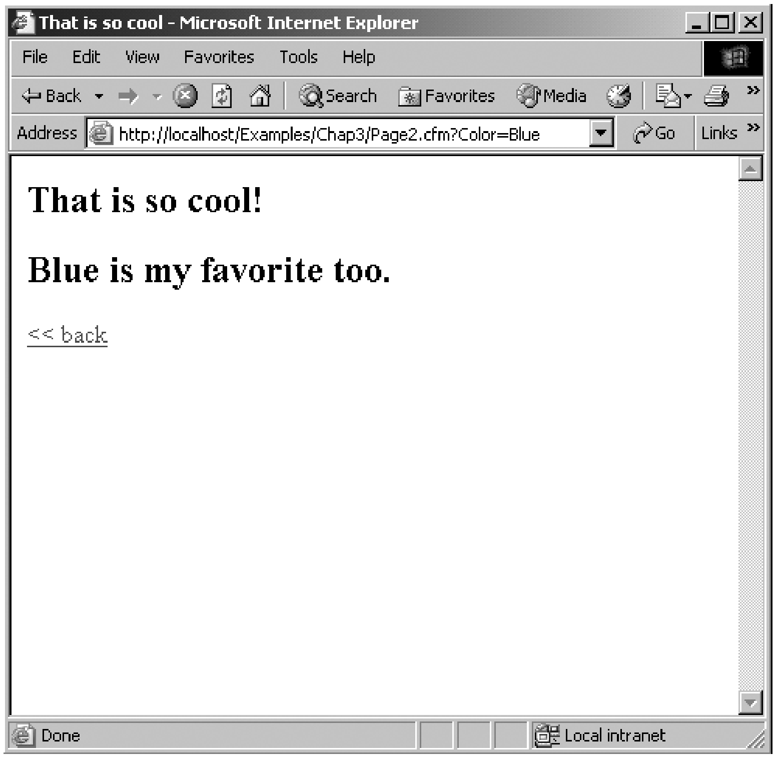
`Page1.htm` *browser display.*


Pick a color any color - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://localhost/Examples/Chap3/Page1.htm
Go Links
Done
Local intranet

What is your favorite color?

Blue
Red
Green
Yellow

6. Follow any link on `Page1.htm`. Notice the URL variable present in the browser's address bar. You should see a page similar to the one shown in Figure 2.4.


That is so cool - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://localhost/Examples/Chap3/Page2.cfm?Color=Blue
Go Links
Done
Local intranet

That is so cool!

Blue is my favorite too.

<< back

FIGURE 2.4

`Page2.cfm` *browser display.*

7. The variable is pulled from the URL and is used in the `<CFOUTPUT>` statement.

Makeover Exercise

During this portion of our web site makeover, we will do the following:

- Use CGI variables to ensure that all the path information for our included files is always correct, no matter where in our directory structure the included page ends up.
- Prepare for future exercises by adding URL variables to our product categories.

First we will fix some of the file path problems that we might have encountered with our included files. When you use a `<CFINCLUDE>` to include code in a page (as we did in Step 1), you will remember that it is essentially the same as copying the code into the calling template. This might cause problems if the templates are in different directories and you are using relative paths for image and file links. However, if you use absolute paths, you have to change that information every time you move your code from a development or testing server to a production server.

In the previous step, we setup a directory structure called `NewSite` to hold our makeover web site. You should have a virtual mapping to the directory set up. For information on setting up virtual mappings see the Appendix, "System Setup".

In the first part of this exercise, we will add CGI variables to the path information in our three included files: `Header.cfm`, `Footer.cfm`, and `LeftNav.cfm`. By adding these CGI variables, we will create "automatic" absolute path information that will work with whatever server the files are stored on. Follow these steps to get this accomplished:

1. Open `Header.cfm` in your editor.
2. Change the path information for the image tag for our logo to read as follows:

   ```
   <IMG SRC="http://#CGI.Server_Name#/NewSite/Images/logo.gif" WIDTH="250"
   HEIGHT="75" VSPACE="2">
   ```

   When ColdFusion processes the template, it will use the values for the server name for the web server on which the files are currently stored.
3. Repeat the procedure for the other two graphics.
4. In the `Header.cfm` file, you will find a link to the `FeedbackForm.cfm` file. Add similar code to this link:

   ```
   <A HREF="http://#CGI.Server_Name#/NewSite/Feedback/FeedbackForm.cfm">
   ```
5. Put `<CFOUTPUT> </CFOUTPUT>` tags around the entire table.
6. If you were to try to use the template now, it would produce an error. Because we have surrounded the table with `<CFOUTPUT>` tags, ColdFusion will become confused by the `#` symbols in the hexadecimal color values. You must escape the `#` symbol by putting an additional `#` symbol in front of it. For more information, see the Note on escaping pound signs in Step 1.
7. You should end up with code very similar to that displayed in Listing 2.4.

LISTING 2.4 `Header.cfm`

```
<!---
File:        Header.cfm
Description: File header table for all files
Author:
Created:
--->

<!--- start header table --->
```

```
<!---       add CGI Variables to make sure the path info
            is correct no mater where the template is used as a
            CFINCLUDE.  Don't forget to 'escape' the # symbols in the
            Hex color codes that appear between CFOUTPUT tags--->
<CFOUTPUT>
 <TABLE WIDTH="735" BORDER="0" CELLPADDING="0" CELLSPACING="0" ALIGN="center">
<!--- escape the # symbol in the color codes --->
   <TR BGCOLOR="##CC0000">
     <TD WIDTH="559" HEIGHT="79" VALIGN="middle" BGCOLOR="##FFFFFF">
        <IMG SRC="http://#CGI.Server_Name#/Images/logo.gif" WIDTH="266"
        ➥HEIGHT="100" VSPACE="2">
     </TD>
     <TD VALIGN="bottom" WIDTH="202" ALIGN="right" BGCOLOR="##FFFFFF">
        <A HREF="mailto:info@beelzebubba.com">
          <IMG SRC="http://#CGI.Server_Name#/NewSite/Images/email.gif" WIDTH="150"
          ➥HEIGHT="27" BORDER="0" ALT="send us some email"></A>

<!--- don't forget to add path info to the HREF attribute of the link to the form
➥--->
         <A HREF="http://#CGI.Server_Name#/NewSite/Feedback/FeedbackForm.cfm">
          <IMG SRC="http://#CGI.Server_Name#/NewSite/Images/feedback.gif"
          ➥WIDTH="150" HEIGHT="24" VSPACE="3" BORDER="0" ALT="please let us know
          ➥what you think"></A>
      </TD>
    </TR>

    <TR BGCOLOR="##333333">
      <TD HEIGHT="8" VALIGN="top" BGCOLOR="##FF6600"
      ➥background="http://#CGI.Server_Name#/NewSite/images/bar.gif">
        &nbsp;
      </TD>
      <TD VALIGN="top" BGCOLOR="##FF6600" ALIGN="right" CLASS="date"
      ➥background="http://#CGI.Server_Name#/NewSite/images/bar.gif" height="8">
      ➥#DateFormat(Now(),"MMMM DD, YYYY")# &nbsp;
      </TD>
  </TR>
  <TR BGCOLOR="##333333">
    <TD HEIGHT="16" VALIGN="top" BGCOLOR="##FFFFFF">
    ➥&nbsp;
    </TD>
    <TD VALIGN="top" BGCOLOR="##FFFFFF">
    ➥&nbsp;
    </TD>
  </TR>
 </TABLE>
</CFOUTPUT>
<!--- end header table --->
```

8. Save your changes.
9. Open `Footer.cfm` in your editor. Use CGI variables to add path information to the links for the navigation bar. Your code should look similar to Listing 2.5.

LISTING 2.5 Footer.cfm

```
<!---
File:        Footer.cfm
Description: Text-based secondary navigation menu
Author:
Created:
--->

<!-- text nav bar -->
<CFOUTPUT>
<P align="center">
  <A href="http://#CGI.Server_Name#/NewSite/Index.cfm">
    Home
  </A>|
  <A href="http://#CGI.Server_Name#/NewSite/News/News.cfm">
    News
  </A>|
  <A href="http://#CGI.Server_Name#/NewSite/Products/Products.cfm">
    Products
  </A>|
  <A href="http://#CGI.Server_Name#/NewSite/About/About.cfm">
    About Us
  </A>|
  <A href="http://#CGI.Server_Name#/NewSite/Contact/Contact.cfm">
    Contact Us
  </A>
</P>
</CFOUTPUT>
```

10. Save your changes.
11. Open `LeftNav.cfm` in your editor. Using CGI variables, add path information to the links for the navigation menu. Your code should look similar to Listing 2.6.

LISTING 2.6 LeftNav.cfm

```
<!---
File:        LeftNav.cfm
Description: Left side main navigation menu
Author:
Created:
--->

<!--- navigation menu --->

<CFOUTPUT>
<!---       add CGI Variables to make sure the path info
            is correct no mater where the template is
            used as a CFINCLUDE   --->
<BR>
<P>
  <A HREF="http://#CGI.Server_Name#/NewSite/Index.cfm"
```

```
CLASS="leftNavMajor">Home</A>
</P>

<P>
  <A HREF="http://#CGI.Server_Name#/NewSite/News/News.cfm"
CLASS="leftNavMajor">News</A>
</P>

<P>
  <A HREF="http://#CGI.Server_Name#/NewSite/Products/Products.cfm"
CLASS="leftNavMajor">Products</A><BR>
  <A HREF="http://#CGI.Server_Name#/NewSite/Products/Chillies.cfm"
CLASS="leftNavMinor">Chillies</A><BR>
  <A HREF="http://#CGI.Server_Name#/NewSite/Products/Sauces.cfm"
CLASS="leftNavMinor">Sauces</A><BR>
  <A HREF="http://#CGI.Server_Name#/NewSite/Products/Clothing.cfm"
CLASS="leftNavMinor">Clothing</A><BR>
  <A HREF="http://#CGI.Server_Name#/NewSite/Products/Gifts.cfm"
CLASS="leftNavMinor">Gifts</A><BR>
</P>

<P>
  <A HREF="http://#CGI.Server_Name#/NewSite/About/About.cfm"
CLASS="leftNavMajor">About Us</A>
</P>

<P>
  <A HREF="http://#CGI.Server_Name#/NewSite/Contact/Contact.cfm"
CLASS="leftNavMajor">Contact Us</A>
</P>
</CFOUTPUT>
```

12. Save your changes.

13. Browse to the home page. Check that the links all work correctly and that the images appear correctly on each page.

    In the next part of the exercise, we are going to prepare some of our links for upcoming exercises. In a future exercise, we are going to create one `ProductCategory.cfm` page to display products from a chosen category. To do this, we will have to pass a category variable to that page via the URL.

14. Open `LeftNav.cfm` in your editor.

15. Find the link for the Chillies product category and add a URL variable to specify the product category. Your link should look something like this:

    ```
    <A HREF="http://#CGI.Server_Name#/NewSite/Products/Chillies.cfm?Category=Chillies"
    CLASS="leftNavMinor">Chillies</A>
    ```

16. Follow the same procedure for the rest of the product categories. Your code should end up looking similar to Listing 2.7.

LISTING 2.7 `LeftNav.cfm` with Category Variables

```
<!---
File:        LeftNav.cfm
Description: Left side main navigation menu
Author:
Created:
--->

<!--- navigation menu --->

<CFOUTPUT>
<!---       add CGI Variables to make sure the path info
            is correct no mater where the template is
            used as a CFINCLUDE   --->
<BR>
<P>
  <A HREF="http://#CGI.Server_Name#/NewSite/Index.cfm"
  ➥CLASS="leftNavMajor">Home</A>
</P>

<P>
  <A HREF="http://#CGI.Server_Name#/NewSite/News/News.cfm"
  ➥CLASS="leftNavMajor">News</A>
</P>

<P>
  <A HREF="http://#CGI.Server_Name#/NewSite/Products/Products.cfm"
  ➥CLASS="leftNavMajor">Products</A><BR>
  <A
  ➥HREF="http://#CGI.Server_Name#/NewSite/Products/Chillies.cfm?Category=Chillies"
  ➥CLASS="leftNavMinor">Chillies</A><BR>
  <A HREF="http://#CGI.Server_Name#/NewSite/Products/Sauces.cfm?Category=Sauces"
  ➥CLASS="leftNavMinor">Sauces</A><BR>
  <A
  ➥HREF="http://#CGI.Server_Name#/NewSite/Products/Clothing.cfm?Category=Clothing"
  ➥CLASS="leftNavMinor">Clothing</A><BR>
  <A HREF="http://#CGI.Server_Name#/NewSite/Products/Gifts.cfm?Category=Gifts"
  ➥CLASS="leftNavMinor">Gifts</A><BR>
</P>

<P>
  <A HREF="http://#CGI.Server_Name#/NewSite/About/About.cfm"
  ➥CLASS="leftNavMajor">About Us</A>
</P>

<P>
  <A HREF="http://#CGI.Server_Name#/NewSite/Contact/Contact.cfm"
  ➥CLASS="leftNavMajor">Contact Us</A>
</P>
</CFOUTPUT>
```

17. Save your file.
18. Browse to the home page and hover your mouse pointer over the category links. Check the URL information displayed in the browser's status bar to confirm that the links are correct. Do not try to follow these links at the moment because they point to a page that does not currently exist; we will create the new page in a future exercise.

Summary

In this step, we looked at variables and discussed the different types of data they can contain as well as the different scopes in which they can live. We also looked at how to create new variables, check for the existence of variables, and pass variables from one template to the next.

In Step 3, we will take a brief look at database basics. We also will discover how to access information stored in a database and use it in our web site.