Chapter 1

Creating Your First ColdFusion Template

Beginning with this chapter and continuing throughout this book, you will create ColdFusion templates. A ColdFusion template is nothing more than a markup file (typically HTML) that also contains ColdFusion statements and has a `.cfm` file extension.

One of the best learning devices is by doing, so in this chapter you are actually going to write a ColdFusion template. Don't worry, though, you won't be diving off the high board right away; you'll get to wade in a bit at a time. You will be shown how to write a simple template, and then its important points will be deconstructed and explained.

Featured in this chapter:

- Setting up the software
- Understanding ColdFusion templates
- Creating your first template

Setting Up ColdFusion

Before you can begin working with the projects in this book, you will need to install some software first:

- A web server—for example, Apache, Microsoft's Internet Information Server (or its workstation version, Personal Web Server), Netscape, or Deerfield WebSite Pro
- ColdFusion MX
- A web development tool, such as HomeSite+, ColdFusion Studio, or Dreamweaver MX

If you have trouble installing ColdFusion MX, refer to Appendix A, "Installing ColdFusion," for assistance.

Using ColdFusion

Once everything is installed correctly, there are really only two things you need to understand to get started using ColdFusion: how to call a template, and what it takes to write a template.

Calling Templates

Calling a ColdFusion template can be done just like calling an HTML (Hypertext Markup Language) page. Addressing the template from the browser is identical for the user with the exception of the file extension.

By default, when the ColdFusion server is serving its own pages (stand-alone mode), it is not running on port 80 like most other web servers (though it could later be configured to do so). This means that you will have to specify a port address in the page's URL. The port number is 8500 by default—for example,

```
http://localhost:8500/index.cfm
```

ColdFusion Markup Language (CFML)

ColdFusion Markup Language (CFML) is a tag-based programming language. It was designed to have an HTML-like look and feel, making it easier for web designers to become web developers. Most tags have attributes, and many are comprised of start and end tag sets. CFML tags are readily identifiable because their tag names all begin with `cf`.

A common HTML tag is the paragraph tag:

```
<p>some content</p>
```

A common CFML tag looks like this:

```
<cfoutput>some content</cfoutput>
```

As you can see, the two markup languages are very similar. Obviously the major differences are the purposes of the two languages: one is for presentation, and the other is for processing data. As you will see later, this makes for a powerful synergy.

When the CFML interpreter processes a template, it executes the tags that it finds and writes out any resulting content (typically text or HTML tags), but no trace of the original CFML will show in the output. All the CFML tags are processed out of the output on the server side. This means the logic in the template is afforded a degree of protection.

Your First Template

Building your first template isn't difficult at all. The first project in this book will walk you through what you need to do to create your first ColdFusion template.

NOTE *The code here utilizes CFML constructs that you are probably not familiar with yet. Don't panic; learning these things is why you are reading this book! All of the constructs used will be explained in greater detail in the next few chapters.*

Launch your HTML writer and create a new HTML document, complete with a `head` and an empty `body`. Add to the empty `body` the following lines of CFML code:

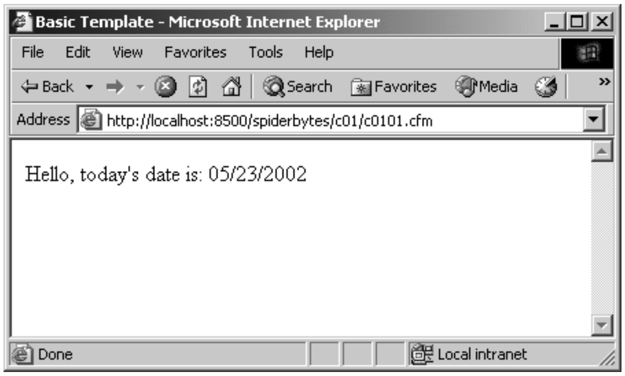
```
<cfset today = dateFormat(now(), "mm/dd/yyyy")>
<cfoutput>
   <p>Hello, today's date is: #today#</p>
</cfoutput>
```

The finished product should look almost exactly like Listing 1.1. Save this document into the web server's document root as `c0101.cfm`. Then load the page in your browser called through the web server.

Listing 1.1: A Basic CFML Template (c0101.cfm)

```
<!---
   Name:      /spiderbytes/c01/c0101.cfm
   Description:   Display current date.
--->
<html>
   <head>
      <title>Basic Template</title>
   </head>
   <body>
      <cfset today = dateFormat(now(),"mm/dd/yyyy")>
      <cfoutput>
         <p>Hello, today's date is: #today#</p>
      </cfoutput>
   </body>
</html>
```

Be certain to load the template by requesting it from your web server (as seen in Figure 1.1) or the ColdFusion server; otherwise, the CFML interpreter will not get invoked to process the template.


Basic Template - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://localhost:8500/spiderbytes/c01/c0101.cfm
Done
Local intranet

```
Hello, today's date is: 05/23/2002
```

Figure 1.1
The output of Listing 1.1

Using Variables

CFML variables are like variables in other development languages: a name given to a place in memory where a value has been saved. There is a variety of ways to construct variables in CFML, but the simplest is `cfset`. Using `cfset` will create a variable if it doesn't exist or replace an existing variable's old value with some new value. It looks like a regular tag but it has a slight twist; the attribute of `cfset` is the name of a variable that will be created or modified when the tag is executed. This means, that unlike other tags, there is not a "standard" attribute for the tag. The attribute value is the value given to that variable. The format is:

```
<cfset variable_name = variable_value>
```

`cfset` will be covered in greater detail in Chapter 4, "Creating and Manipulating Variables," and can also be found in Appendix B, "ColdFusion Tag Reference."

In Listing 1.1, we created a variable called *`today`* and assigned it the result of a function called `dateFormat`.

There are some general rules to using CFML variables in your templates:

- A variable cannot be a reserved word, cannot contain spaces or special characters and cannot begin with a number.
- In order to display or use the value of a variable as part of an HTML tag (such as the value of a tag's attribute), it must be processed within the body of a `cfoutput` block. Variables are identified to `cfoutput` by enclosing them within pound signs (`#`) as you did in the paragraph body of Listing 1.1.

Using Functions

CFML is not only composed of tags; it also contains nearly 300 functions. Functions will be discussed in greater detail in Chapter 5, "Functions," but suffice it to say for now that a function is a language construct that can be called to perform a specific task. There are groups of functions for handling date and time conversions, string parsing, number processing and formatting, and various data type handling. Most of the functions return values that can be saved to variables. This is most commonly done with `cfset`, as you did in Listing 1.1.

Data can be passed into a function (comma-delimited within parentheses following the function name) as well as returned. A piece of data passed to a function is called an *argument.* The kind of data and meaning that an argument has are specific to each individual function. An argument can be a literal value, a variable, or even the value returned from another function.

There are actually two functions used in Listing 1.1, `dateFormat` and `now`. The first and more complicated of these takes two arguments, in the order `dateFormat(`*`date`*`, `*`mask`*`)`. In our example, the *`date`* argument is actually the value returned from the `now` function, and is a date/time value between 100 and 9999 AD. The second argument, *`mask`*, is a string that describes how `dateFormat` should display the first argument. Table 1.1 lists the various character strings used in the *mask* argument to indicate the desired format.

TABLE 1.1: FORMAT CHARACTER CODES FOR THE DATEFORMAT FUNCTION

CHARACTER	FORMAT EFFECT
`d`	Day of the month as digits; no leading zero.
`dd`	Day of the month as digits; with leading zero.
`ddd`	Day of the week as three-letter abbreviation.
`dddd`	Full name of the day of the week.
`m`	Month as digits; no leading zero.
`mm`	Month as digits; with leading zero.
`mmm`	Month as three-letter abbreviation.
`mmmm`	Full name of the month.
`y`	Year as last two digits; years less than 10 no leading zeros.
`yy`	Year as last two digits; years less than 10 with leading zeros.
`yyyy`	Year in four-digit representation.
`gg`	Period/era indicator. Currently ignored.

Using cfoutput

The `cfoutput` tag will not be discussed in detail in this chapter, as it will be covered over the next few chapters and can also be found in Appendix B. `cfoutput` is a vital CFML tag, and you will use it often; it is important to get comfortable with it. This tag is used to block off some content (typically HTML) that contains CFML variables that need to be displayed. CFML variables can be used as attribute values to other CFML tags or can be assigned to other CFML variables without being processed by `cfoutput`. However, if you are using a variable or function results to format content for display or to feed as an attribute to an HTML tag, then you will need `cfoutput`.

Commenting Your Code

Commenting code is a much-maligned developer task, but the importance of commenting your code cannot be stressed strongly enough. Who knows when you'll look at your code again once you're done? Or more likely, who knows who *else* will have to look at your code?! If you don't have much experience writing code, this can seem less important early in your evolution, but one day it will be critical. It's best to get into the practice early, so it won't seem like so much of a chore later.

Most web designers are familiar with HTML comments:

```
<!-- something important to remember -->
```

Anything between the comment's start and end punctuation will be ignored by the browser and will not have an impact on the display of the content in any way.

CFML comments are very similar, but they use three dashes instead of just two:

```
<!--- something important to remember --->
```

The interpreter will ignore anything, including other CFML statements, within a CFML comment block. Further still, like other CFML statements, the comment block does not become part of the generated output that goes to the client.

This being the case, a developer should feel comfortable liberally adding CFML comments to templates without fear of exposing dangerous information to the client. At a minimum, each template should carry a block that identifies the various things about it:

```
<!---
   Filename: template name
   Author: developer's name
   Date: template's creation date
   Application: application's name
   Type: standard template
   Description:
      Describe the template in some detail.
   Changes:
      1/1/00   GR   did something to the template logic
--->
```

NOTE *This header is different than the header that is used with the samples throughout this book. The reason for this is that the one used in the sample code is meant for identification purposes and brevity, and it does not attempt to be as encompassing as a real header ought to be. This should not be taken to mean that the other fields listed in the longer header are superfluous—quite the contrary. Well-documented code, even if only a complete header, is vital to providing quality production code.*

This block contains seven very important pieces of information: the filename, the original author, the date of creation, the application, the kind of template this file represents, the description, and the changes log. Some of these things have obvious importance, such as Filename and Author. Others, like Type, may not be instantly obvious, though. Type describes the kind of functionality the template encapsulates. This is something that should be tailored to the development team's jargon. Not all templates are for displaying content; some are libraries of functions that are meant to be included into other templates.

Of all of the fields in the block, the Changes information is, arguably, the most important. It tracks what, when, and by whom alterations have been made to a template. This will be vital during the lifecycle of an application as new or altered business requirements force changes in the code.

Where Do We Go from Here?

This chapter showed you how to create your first ColdFusion template. Certainly there wasn't anything terribly complex about this template, but it introduced you to some very important CFML concepts, such as using CFML variables and functions and commenting your ColdFusion code.

The next chapter will build on these concepts, and you will learn how to pass data between templates via HTML forms and hyperlinks. These are standard practices that you will use time and again when building ColdFusion applications.