

Module 8

Making Decisions with ColdFusion

The Goals of This Module

- Introduce conditional logic
- Learn how to create if statements in ColdFusion
- Learn the cfif and cfswitch statements

2 | ColdFusion 5: A Beginner's Guide

Up to this point, all the code written gets executed, no questions asked. But, what if we want to execute only a certain piece of code if certain things are true? For instance, we probably won't want to execute the code on the processing page of a form if the relevant form variables aren't defined. The answer to this is conditional logic. This module explains what conditional logic is and shows you how to implement it in ColdFusion.

An Overview of Conditional Logic

Conditional logic is something we experience every day. We'll talk a little about Boolean logic and operators but, first, let's look at some real-world examples of conditional logic.

What Is Conditional Logic?

Imagine you're in your car, driving down South Street. South Street ends at an intersection with Main Street. You have two options: you can make a right turn onto Main Street or a left turn onto Main Street. If you take a right, you'll be traveling west on Main Street. If you take a left, you'll be traveling east on Main Street. This is an example of conditional logic. You make a choice based on a certain condition and perform an action based on that choice.

Conditional logic is an important concept in computer programming. For example, say you're editing a document in your favorite HTML editor. You try to save it. Is your document a new document, which means you'll have to enter a file name for the document? Or, is it an existing document where the program can save the document without additional input from you? The program will use conditional logic to decide whether this is a new or an old document.

The most common form of conditional logic is the if statement. Later in this module, you'll see how to create if statements in ColdFusion but, first, you can examine some pseudocode to see how if statements can be implemented:

L 8-1

```
If Condition then  
  Perform actions  
Endif
```

If some condition is equal to true, then we go ahead and perform some function. I used an endif identifier to specify where the ending code lies. Let's look back to our driving example:

Module 8: Making Decisions with ColdFusion 3

```
L 8-2  If Take Left Turn onto Main Street then
      You drive west on Main Street
      EndIf

      If take right turn onto Main Street then
      You drive east on Main Street
      EndIf
```

If you take a left turn onto Main Street, then you'll be driving west on Main Street. If you take a right turn onto Main Street, then you will be driving east on Main Street. These are the two separate conditions from the original example.

In the world of coding, though, you would probably want to make steps to ensure both of these conditions don't have the opportunity to happen at once. After all, you probably won't be able to drive both west and east on Main Street at the same time. The way to do this is called an else-if statement. *Else-if statements* enable you to group conditions together. This is the basic structure:

```
L 8-3  If Condition1 then
      Perform action1
      ElseIf Condition2 then
      Perform action2
      Endif
```

If condition1 is true, then we go ahead and perform action1. If condition1 is false, then we check to see if condition2 is true. If condition2 is true, then we go ahead and execute action2. If condition2 is false, we end our statement block and continue executing code.

Hint

In our pseudocode if statements, the actions performed if the condition is true can be either a single action or a group of multiple actions.

Let's Rework the driving example to take advantage of our else-if statement:

```
L 8-4  If Take Left Turn onto Main Street then
      You drive west on Main Street
      ElseIf take right turn onto Main Street then
      You drive east on Main Street
      EndIf
```

4 ColdFusion 5: A Beginner's Guide

This code says if you take a left turn onto Main Street, then you drive west on Main Street, or else, if you take a right turn onto Main Street, then you drive east on Main Street. This is exactly what will happen and it ensures we aren't driving both west and east on Main Street at the same time. Else-if statements can be repeated endlessly until you're out of potential conditions.

One potential problem with the previous code is we don't have a default action. What happens if condition1 is false and condition2 is false? The code will continue to execute at the end of the if statement without executing any commands. Thankfully, else statements exist to achieve this type of code. Instead of an else-if, we simply use an else statement. There is no additional condition, only additional actions if none of the other conditions are true.

```
L 8-5  If Condition1 then
        Perform action1
    ElseIf Condition2 then
        Perform action2
    Else
        Perform Action3
    Endif
```

If condition1 is true, then we perform action1. If condition1 is false, we check condition2. If condition2 is true, we perform action2. Now comes the new concept. If condition2 is false, we perform action3. After that, the execution of our template continues after the end-if statement.

Let's see how the else statement works into your driving example:

```
L 8-6  If Take Left Turn onto Main Street then
        You drive west on Main Street
    ElseIf take right turn onto Main Street then
        You drive east on Main Street
    Else
        You stop driving.
    EndIf
```

If you take a left turn onto Main Street then you will be driving west on Main Street, or else if you take a right turn onto Main Street, then you will be driving east on Main Street, or else you will stop driving. The else statement action could be anything. Perhaps the user turns around and continues to drive, back the way he came. Perhaps the car stalls and he stops driving. The final else action could be anything you want. This is only a way to provide some default logic into your code so something will happen.

Module 8: Making Decisions with ColdFusion 5

Boolean Logic and Operators

In the previous section, we learned about if statements and how they can react to certain conditions. This section teaches you about how to use Boolean logic to create more advanced conditions. As discussed in past modules, a Boolean value is one that can have only one of two values: 0 or 1, true or false, yes or no, on or off. When you're performing Boolean logic, the intent is to get the result down to one of these two values. We're going to review the following Boolean operators: AND, OR, NOT, XOR, EQV, and IMP.

To understand how Boolean logic operates, first we need to examine a truth table. A *truth table* is a table that lists all possible value combinations for the variables in question and the final result of the whole expression. The conditions discussed in the last section of this module only had one condition. This is the simplest of all truth tables:

Condition	Condition
True	True
False	False

If the condition is true, then the result is true. If the condition is false, then the result is false.

The first Boolean operator—the NOT operator—can work on a single condition. The remainder of the operators are ways of comparing two separate conditions. The NOT operator merely reverses the value of a condition. The symbol for NOT is ~, however, writing out the word NOT is also a valid approach and the one ColdFusion uses. Here's the truth table for a not condition:

Condition	Not (Condition)
True	False
False	True

The two most commonly used operators are the AND and OR operators, so let's look at them first.

The AND operator compares two conditions like this:

- If both conditions are true, the result is true.
- If both conditions are false, the result is false.
- If one condition is true and the other is false, the result is false.

6 ColdFusion 5: A Beginner's Guide

Both conditions must be true for the AND operator to return a true value. Our truth table with two conditions must be expanded from our original single condition truth table. We have more options. Let's call these conditions X and Y:

X	Y	X AND Y
True	True	True
True	False	False
False	True	False
False	False	False

For example, say X represents "The sky is blue" and Y represents "The ocean is blue." Because both X and Y are true, the final result is true. If we change Y to represent "The sky is black" then X is true, while Y is false. The final result will be false.

The OR operator compares two values like this:

- If both conditions are true, then the result is true.
- If both conditions are false, then the result is false.
- If one conditions is true and the other value is false, then the result is true.

The OR operator is similar to the AND operator when both values are equal, but it's the exact opposite when the values are different. Only one condition must be true for an or statement to return true.

X	Y	X OR Y
True	True	True
True	False	True
False	True	True
False	False	False

If X is "The sky is blue" and Y is "The ocean is blue.," our result value for the OR operator will be true because both X and Y are true. If we change Y to "The sky is black," then X is true, while Y is false. The final result will be true because at least one condition is true.

The next operator is the XOR operator. The XOR operator is known as the exclusive or, which means either or, but not both. The exclusive or is governed by these facts:

Module 8: Making Decisions with ColdFusion 7

- If both conditions are true or both values are false, then the result is false.
- If one conditions is true and the other value is false, then the result is true.

Let's look at the truth table:

X	Y	X XOR Y
True	True	False
True	False	True
False	True	True
False	False	False

Based on the rules of the exclusive or, if *X* says "The sky is blue" and *Y* states "The ocean is blue," your result would be false. Both of the values are true. If we change *Y* to "The sky is black," then *X* is true, while *Y* is false, and the final result would be true. If we then change *X* to "The ground is orange," that is also false, so false exclusive or false will return true.

The opposite of the exclusive or is known as equivalent. The ColdFusion symbol for equivalent is `EQV`. Equivalent computes values like this:

- If both values are true, then the result is true.
- If both values are false, then the result is true
- If one value is true and the other value is false, then the result is false

8

The truth table for the equivalent operator can be seen here:

X	Y	X EQV Y
True	True	True
True	False	False
False	True	False
False	False	True

Following along with the example we've been using for equivalence, if *X* is "The sky is blue" and *Y* is "The ocean is blue," the result would be true because both conditions are true. If we change *Y* to be "The sky is black," then *X* is true, while *Y* is false, so the final result would be false. If we then change *X* to "The ground is orange," then both *X* and *Y* are false, making the result value true.

8 ColdFusion 5: A Beginner's Guide

The final operator to examine is the implication operator. ColdFusion uses `IMP` to refer to implication. To say $X \text{ IMP } Y$ is the equivalent of "IF X , then Y ." *Implication* is the only operator where the order of the conditions will make a difference. $X \text{ IMP } Y$ might not be the same as $Y \text{ IMP } X$. Implication follows these rules:

- If condition1 is true and condition2 is false, then the result is false.
- If condition1 is true and condition2 is true, then the result is true.
- If condition1 is false, then the result is true.

The implication truth table is this:

X	Y	X IMP Y
True	True	True
True	False	False
False	True	True
False	False	True

If X states "The sky is blue" and Y says "The ocean is blue," then your result would be true because $X \text{ IMP } Y$. If we change Y to "The sky is black," then X is true, while Y is false, then the final result would be false. If we then change X to "The ground is orange," then both X and Y are false, making the result value true. False implies false.

Truth tables can easily be expanded to include more than two conditions. Let's look at a more complex condition that includes three variables:

L 8-7

`((X and Y) or Z)`

I used parentheses to make the order of operators more apparent. Because different operators are in this expression, we have a larger truth table than we had before. To simplify the process of creating the truth table, we'll figure out the first condition— X and Y —in its own column. Then we have the values there for comparison as we find our final result. This is the truth table:

X	Y	Z	(X and Y)	((X and Y) or Z)
True	True	True	True	True
True	True	False	True	True

Module 8: Making Decisions with ColdFusion 9

X	Y	Z	(X and Y)	((X and Y) or Z)
True	False	True	False	True
True	False	False	False	False
False	True	True	False	True
False	True	False	False	False
False	False	True	False	True
False	False	False	False	False

Expanding on the example from earlier in this module, let's say *X* states "The sky is blue," *Y* is "The ocean is blue," and *Z* is "The sky is orange." We have ((true and true) or false), making the final result true. If we change *Z* to "Clouds are made up of water," then the final result will be true no matter what the values *X* and *Y* contain. Next, we look into the tag that ColdFusion uses to implement a conditional if statement.



1-Minute Drill

- What is a truth table?
- What does a conditional statement do?

8

ColdFusion's cfif Tag

We've examined the concept behind conditional programming logic and looked at how Boolean logic is used to make a decision. The next step is to see how to implement these features in ColdFusion. ColdFusion provides a set of tags to perform an if statement.

The cfif Tag

The tag ColdFusion uses to create a conditional statement is called *cfif*. As discussed earlier in this module, ColdFusion also has *cfelse* and *cfelseif* statements to allow for multiple choices or a default condition. This section reviews the *cfif* tags and some uses for them in ColdFusion development.

-
- A truth table is a way to get all possible outcomes of a condition.
 - A conditional statement enables you to make decisions in your code

10 ColdFusion 5: A Beginner's Guide

The format of the cfif statement is this:

L 8-8

```
<cfif Expression>  
  Actions  
</cfif>
```

The cfif is a tag that has an open and close tag. There are no attributes to discuss in this tag, only an expression after the cfif. The expression must result in a Boolean value.

Hint

If your expression doesn't result in a Boolean value, but is an Integer value, ColdFusion will accept a 0 as false, and anything else as true.

ColdFusion also provides tags to do else-if and else statements. The tags, as you might guess, are cfelseif and cfelse, respectively. This is the format:

L 8-9

```
<cfif Expression1>  
  Action1  
<cfelseif Expression2>  
  Action2  
<cfelse>  
  Action3  
</cfif>
```

The cfelseif tag is similar to a cfif tag. After the tag name comes an expression. These expressions can be simple or complex expression as long as they result in a Boolean value.

Tip

Remember, the expression in a cfif or cfelseif statement can be a complex expression.

In web development, there are many reasons to use conditional statements. In secure applications, checking to see if users are logged in before showing them the page they're trying to load is common. On a form processing page, we want to make sure the form variables are defined before processing the page. Or, perhaps you'll want to verify the data type of a form or a URL variable before processing the data. All these things can be done with the cfif statement. ColdFusion provides some specific functions for a variable's value.

Ask the Expert

Question While reviewing the truth tables in this section, I realized if we're comparing two expressions using the AND operator, if the first condition is false, there's no need to look at the second condition because the answer will definitely be false. Am I correct in my assessment? How does ColdFusion handle situations like this?

Answer You're correct in your assessment of the AND operator. You can say something similar about the OR operator. Once you find a single true condition, the whole expression will return a true result. You'll find you can make similar assumptions with all the Boolean operators. Earlier versions of ColdFusion would inefficiently calculate all expressions before deciding what course to take. Starting in ColdFusion 4.01, ColdFusion started to use something called *short-circuit evaluation*, which takes note of these specific Boolean conditions. If short-circuit evaluation finds something that will give it a concrete result for the expression, it won't evaluate the remainder of its expressions.

8

The first function we should look at is to check whether a variable exists. ColdFusion will produce errors if you try to access variables that don't exist. We can use the `IsDefined` function to check for a variable's existence. The one attribute this function takes is the name of the variable, in a string. Its use is like this:

L 8-10

```
<cfif IsDefined("variables.MyVar1")>  
  <cfoutput>#variables.MyVar1#</cfoutput>  
<cfelse>  
  Warning, the variable was undefined.  
</cfif>
```

This `cfif` statement checks for the existence of the `MyVar1` variable in the local variable scope. If this is found, it outputs the variable's value. If the `MyVar1` variable isn't found, it outputs an undefined variable warning. This is a common implementation for which you'll find many uses throughout your application.

12 ColdFusion 5: A Beginner's Guide

ColdFusion also provides a handful of functions for checking the data type of a variable. These functions are used much in the same way the `IsDefined` function is used. All of them accept some value. You'll usually be testing against a variable name. Here's a partial list of the relevant data-checking functions:

- **IsBoolean** The *IsBoolean* function tests whether the value you give it as an argument can convert into a Boolean value.
- **IsDate** The *IsDate* function tests whether the argument is a valid date format.
- **IsNumeric** The *IsNumeric* function is used to test if a value is a valid number value.
- **IsSimpleValue** The *IsSimpleValue* function checks whether the variable you give it is a complex or a simple data type.

ColdFusion also provides functions for checking if a variable is a complex data type. It's important to know functions like these exist, so you can use them when the need arises during your development.

Comparison Operators

We learned about arithmetic operators and string operators in Module 6 and explored ColdFusion's Boolean operators earlier in this module, so the only operators we haven't examined yet are the comparison operators available for use in ColdFusion expressions. You can see the list of comparison operators in Table 8-1.

Operator	Shorthand/Alternates	Description
IS	EQUAL, EQ	Compares two values and returns true if the values are identical.
IS NOT	NOT EQUAL, NEQ	Compare two values and returns true if the values aren't identical.

Table 8-1 ColdFusion's Decision Operators

Module 8: Making Decisions with ColdFusion 13

Operator	Shorthand/Alternates	Description
CONTAINS	N/A	Checks to see if the value on the left is contained in the value on the right. If it is, it returns true.
DOES NOT CONTAIN	N/A	Checks to see if the value on the left is contained in the value on the right. If it is, it returns false.
GREATER THAN	GT	Checks to see if the value on the left is larger than the value on the right. If it is, it returns true.
LESS THAN	LT	Checks to see if the value on the left is smaller than the value on the right. If it is, it returns true.
GREATER THAN OR EQUAL TO	GTE, GE	Checks to see if the value on the left is greater than or equal to the value on the right. If it is, it returns true.
LESS THAN OR EQUAL TO	LTE, LE	Checks to see if the value on the left is less than or equal to the value on the right. If it is, it returns true.

8

Table 8-1 ColdFusion's Decision Operators (*continued*)

You can test for equality using the IS operators. IS NOT can test for two values not being equal. You can compare numbers using GREATER THAN, LESS THAN GREATER THAN OR EQUAL TO, or LESS THAN OR EQUAL TO. CONTAINS is used to see if one value is contained in the other. DOES NOT CONTAIN is used to see if one value is not contained in the other. As shown in Table 8-1, some shorthand versions exist of many of these operators.



1-Minute Drill

- What tag does ColdFusion use to perform conditional logic?
- What is ColdFusion's equality operator?

-
- ColdFusion uses the `cfif` tag to perform conditional logic.
 - IS, EQ, or EQUAL are all equality operators.

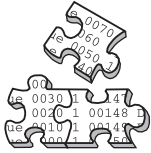
14 ColdFusion 5: A Beginner's Guide

At this point you've learned about all four types of operators in ColdFusion: arithmetic, string, Boolean, and comparison. The order of operations was discussed in the section on math functions, but we need to revisit that order of operations list now that we have more operators in the mix. In ColdFusion, we can mix and match operators of the different types.

The arithmetic operators come first, and then the string operator, followed by the comparison operators, followed by the Boolean operators. This is the order:

- **Unary +, Unary -** To show the sign of a number
- **^** To perform exponents
- ***, /** To multiple or divide, respectively
- **** To perform a div function
- **MOD** To perform modulus operation
- **+, -** To perform addition or subtraction
- **&** to perform string concatenation
- **EQ, NEQ, LT, LTE, GT, GTE, CONTAINS, DOES NOT CONTAIN** To perform decision operators
- **NOT** To reverse Boolean conditions
- **AND** To perform the Boolean and operator
- **OR** To perform the Boolean or operator
- **XOR** To compare using an exclusive or
- **EQV** To compare using equivalence
- **IMP** To compare using the implication operator

Remember, parentheses can be used in your expressions and can be used to have an effect on the order of operations. The order of operations is important to understand when we're creating conditions to command the flow of logic in your ColdFusion application.



Project 8-1: Revisit Application.cfm

In Module 7, we created a sample Application.cfm. This project revisits that Application.cfm to see how we can improve the efficiency of our code by using the cfif statement. Most noticeably, our application and session variables needn't be reset if they already exist. This is where you use your conditional logic.

Step-by-Step

1. We want to look at our existing Application.cfm from Project 7-1 in Module 7. You can download this file and our updated Application.cfm from the web site.

```
<!---
Description: A Sample Application.cfm

Entering: N/A
Exiting: N/A

Dependencies: N/A
Expecting: N/A

Modification History
Date      Modifier      Modification
*****
11/09/2001 Jeff Houser, DotComIt Created
--->

<cfapplication name="SampApp"
  applicationtimeout=#CreateTimeSpan(0,1,0,0)#
  setclientcookies="Yes" clientmanagement="Yes"
  sessiontimeout=#CreateTimeSpan(0, 1, 0, 0)#
  sessionmanagement="Yes">

<!-- define the application variables -->
<cflock scope="application" type="exclusive"
  timeout="30">
  <cfset application.WebServerRoot =
    "C:\Projects\htdocs">
  <cfset application.ImageDir = "/image/">
</cflock>

<!-- define the session variables -->
<cflock scope="session" type="exclusive"
```

16 ColdFusion 5: A Beginner's Guide

```
        timeout="30">
    <cfset session.IsLoggedIn = "False">
</cflock>

<!-- copy the session scope into the request scope -->
<cflock scope="session" type="readonly" timeout="30">
    <cfset request.TempSession = Duplicate(session)>
</cflock>
```

Notice we set the session and application variables every time the Application.cfm is executed, which is every time for every page. Let's look at the Application variables, and then we can look at the session variables.

2. The first, and perhaps most important, change we want to make is to add some modification history to your documentation header. The following file shows the results:

```
<!---
Description: A Sample Application.cfm

Entering: N/A
Exiting: N/A

Dependencies: N/A
Expecting: N/A

Modification History
Date      Modifier      Modification
*****
11/09/2001 Jeff Houser, DotComIt Created
11/13/2001 Jeff Houser, DotComIt Added checking for
                                   setting of app and session
                                   variables _____
--->
Modification history

<cfapplication name="SampApp"
    applicationtimeout="#CreateTimeSpan(0,1,0,0)#"
    setclientcookies="Yes" clientmanagement="Yes"
    sessiontimeout="#CreateTimeSpan(0, 1, 0, 0)#"
    sessionmanagement="Yes">

<!-- define the application variables -->
<cflock scope="application" type="exclusive"
```

Module 8: Making Decisions with ColdFusion 17

```
        timeout="30">
        <cfif not IsDefined("application.WebServerRoot")>
        <cfset application.WebServerRoot =
            "C:\Projects\htdocs">
        <cfset application.ImageDir = "/image/">
        </cfif>
    </cflock>

    <!-- define the session variables -->
    <cflock scope="session" type="exclusive"
        timeout="30">
        <cfset session.IsLoggedIn = "False">
    </cflock>

    <!-- copy the session scope into the request scope -->
    <cflock scope="session" type="readonly" timeout="30">
        <cfset request.TempSession = Duplicate(session)>
    </cflock>
```

cfif around application variables

The previous code segment adds in our modification history. Let's move down deeper into the document to examine the lock around the application variables. This code makes the assumption that if one application variable isn't defined, then all of them must not be defined. I checked the WebServerRoot variable. If it isn't defined, we go ahead and define all application variables. If it is defined, we do nothing.

The cfif and cfset all reside in an exclusive cflock block. Even though the cfif is only read access to the application scope, if we have to write the variables, we'll need an exclusive lock.

3. The last step in our code is to add our cfif statement around the session variables.

```
<!---
Description: A Sample Application.cfm

Entering: N/A
Exiting: N/A

Dependencies: N/A
Expecting: N/A

Modification History
Date      Modifier      Modification
```

18 ColdFusion 5: A Beginner's Guide

```
*****
11/09/2001 Jeff Houser, DotComIt Created
11/13/2001 Jeff Houser, DotComIt Added checking for
        setting of app and session
        variables
--->

<cfapplication name="SampApp"
  applicationtimeout="#CreateTimeSpan(0,1,0,0)#"
  setclientcookies="Yes" clientmanagement="Yes"
  sessiontimeout="#CreateTimeSpan(0, 1, 0, 0)#"
  sessionmanagement="Yes">

<!-- define the application variables -->
<cflock scope="application" type="exclusive"
  timeout="30">
  <cfif not IsDefined("application.WebServerRoot")>
  <cfset application.WebServerRoot =
    "C:\Projects\htdocs">
  <cfset application.ImageDir = "/image/">
  </cfif>
</cflock>

<!-- define the session variables -->
<cflock scope="session" type="exclusive"
  timeout="30">
  <cfif not IsDefined("session.IsLoggedIn")> ←
  <cfset session.IsLoggedIn = "False">
  </cfif>
</cflock>

<!-- copy the session scope into the request scope -->
<cflock scope="session" type="readonly" timeout="30">
  <cfset request.TempSession = Duplicate(session)>
</cflock>
```

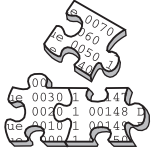
cfif around session variables

Similar to what we did with our application variables, we check for the definition of a session variable. If it doesn't exist, we create it. If it does exist, we do nothing.

The reason we don't use a cfparam around the session and application variables is this: the cfparam performs a check on all variables while we examine a single variable and the cfparam uses that check to make the decision for all variables.

Project Summary

Session and application variables are persistent variables that exist between page requests, but there's no reason to try to set the variables if they already exist. The techniques discussed here are a common solution to the problem.



Project 8-2: Revisit Form Submission

If you remember back to Module 6, we created a form for creating a new genre in our product database. We used some string processing functions to truncate a user's entry if it was too long. This project reexamines that code and provides a better way to create a new genre. Instead of automatically truncating the information without warning the user, we use conditional logic to warn him that his entry was too long and he should modify it.

Step-by-Step

1. Open the original page, `genrep.cfm`, from Project 6-1. The original code follows.

```
<!---
Description: The processing page for creating a new
            genre

Entering: Genrei.cfm
Exiting: N/A

Dependencies: N/A
Expecting: form.Genre

Modification History
Date      Modifier      Modification
*****
11/06/2001 Jeff Houser, DotComIt Created
---->

<!-- insert query -->
<cfquery datasource="Chapter6" name="InsertGenre">
  INSERT INTO Genre (Genre)
  VALUES ('#Left(Genre,25)#' )
</cfquery>

<!-- output the info -->
<cfoutput>
  Your new genre <b>#Left(Genre,25)#</b> was created. <br>
</cfoutput>
```



20 ColdFusion 5: A Beginner's Guide

This is a form processing page and we get here by submitting the Genrei.cfm template. The code first inputs the user entry into the database, and then outputs a thank you message. We use the Left function to trim unnecessary characters automatically from the user input. We want to remove the Trim function.

2. The first two things we do in this project are remove the Left function and update the modification history of our documentation.

```
<!---
Description: The processing page for creating a new genre

Entering: Genrei.cfm
Exiting: N/A

Dependencies: N/A
Expecting: form.Genre

Modification History
Date      Modifier      Modification
*****
11/06/2001 Jeff Houser, DotComIt Created

11/13/2001 Jeff Houser, DotComIt Added conditional
          check to check the length
          of the form.Genre variable
--->
```

Added modification history

```
<!-- insert query -->
<cfquery datasource="Chapter6" name="InsertGenre">
  INSERT INTO Genre (Genre)
  VALUES ('#Genre#' )
</cfquery>
```

Removal of left function

```
<!-- output the info -->
<cfoutput>
  Your new genre <b>#Genre#</b> was created. <br>
</cfoutput>
```

Making these two modifications is the first step in your change. Next, we want to add our conditional.

Module 8: Making Decisions with ColdFusion 21

- The final step in our process is to add a conditional statement to check to see if the variable meets our specified length. The following shows the code in our updated Genrep.cfm template:

```
<!---
Description: The processing page for creating a new genre

Entering: Genrei.cfm
Exiting: N/A

Dependencies: N/A
Expecting: form.Genre

Modification History
Date      Modifier      Modification
*****
11/06/2001 Jeff Houser, DotComIt Created
11/13/2001 Jeff Houser, DotComIt Added conditional
                                     check to check the length
                                     of the form.Genre variable
--->
<cfif Len(form.Genre) GT 25>
Warning, the name for the genre that you were trying
to create was too long. Please shorten it. <br>
<A HREF="Genrei.cfm">Back to the Input Page</a>
<CFELSE>
<!-- insert query -->
<cfquery datasource="Chapter6" name="InsertGenre">
INSERT INTO Genre (Genre)
VALUES ('#Genre#' )
</cfquery>

<!-- output the info -->
<cfoutput>
Your new genre <b>#Genre#</b> was created. <br>
</cfoutput>
</cfif>
```

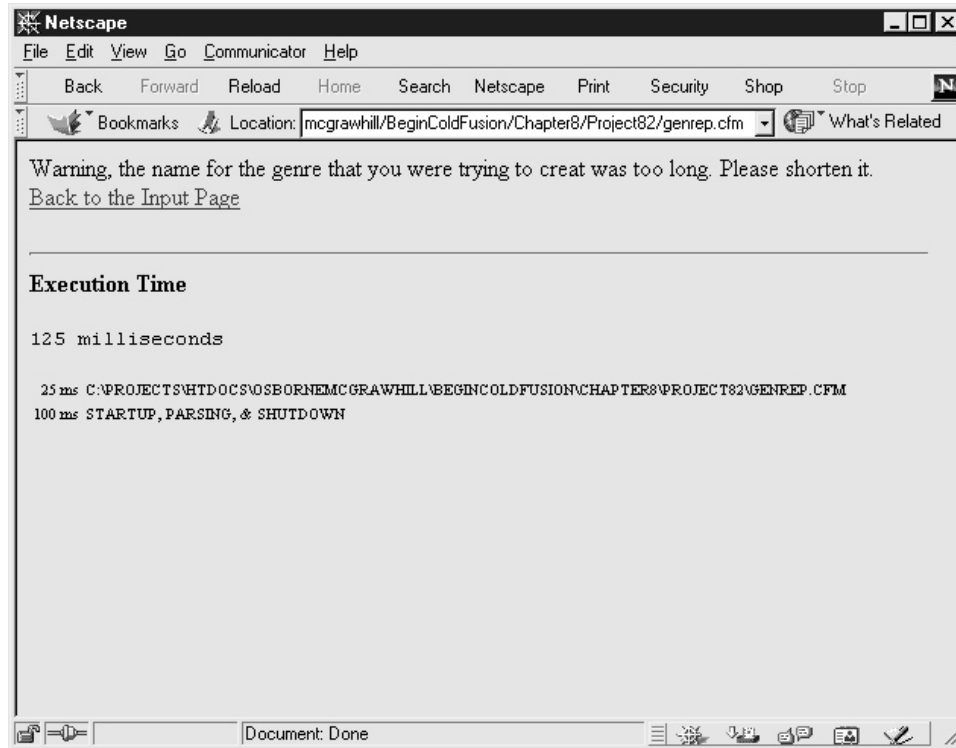
Addition of conditional statement

Check length of variable

22 ColdFusion 5: A Beginner's Guide

In our cfif tag, we use the Len function against our form.Genre variable. The len function will return the length of the string. If the value is greater than 25, then we will give the user a warning message, like so:

Ill 8-1



If the length of our new genre string is less than 25, we will continue to process the code normally, by inserting it into the database and displaying a thank you message to the user.

Project Summary

This project stepped you through some simple form validation you can use when you're out in the web development world. Incorrect data could cause database errors or unexpected results in your application. Verifying the data's type and value or other attributes is an important concept.

When You Have More Options

The `cfif` statement is an important statement to make decisions in ColdFusion. If you have many options, though, the code can become inefficient and hard to maintain because it will have to cycle through, for example, five different paths before coming to the correct path. ColdFusion provides a tag, called `cfswitch`, to help optimize performance and readability when you're working with code that necessitates numerous conditions.

`cfswitch` and `cfcase`

ColdFusion's `cfswitch` tag is used in situations where you have many conditions. It offers increased performance and readability over a group of `cfif/cfelseif` tags. The switch statements have two parts: the first part is the *switch*, which is an expression that defines how to choose the result. The second part is the *case*, which displays a list of specific values the switch might result in. Based on the switch, the case's actions are performed.

The `cfswitch` tag contains a single attribute: `expression`. It accepts the expression that will result in our condition. The `cfcase` tag has two attributes: `value` and `delimiters`. The `value` attribute is the value of our expression. If the result of the expression in the `cfswitch` is equal to the value of the `value` attribute in the `cfcase` statement, then that `cfcase` code is executed. Multiple values can be attributed to a single `cfcase`. The second attribute to the `cfcase` statement is the `delimiter` attribute. If you are applying multiple values to a `cfcase`, the `delimiter` attribute will specify the separator. The default value is a comma.

Both `cfswitch` and `cfcase` contain a start and end tag. Here's the format for the `cfswitch` and `cfcase` tags:

L 8-11

```
<cfswitch expression="Expression">
  <cfcase value="Value1">
    Actions1
  </cfcase>
  <cfcase value="Value2">
    Actions2
  </cfcase>
</cfswitch>
```

24 ColdFusion 5: A Beginner's Guide

The `cfcase` tags are placed inside the `cfswitch` block. When ColdFusion comes up to this code, it evaluates the expression in the `cfswitch`. Then it checks through each `cfcase` tag and examines the value attribute. If the value is equal to the result of the expression, it executes the specific action in the `cfcase` block, and then continues template execution after the end of the `cfswitch`. Otherwise, it checks the next `cfcase` tag and checks its value. The corresponding `cfif` statement would be the following:

L 8-12

```
<cfif Expression is Value1>  
  Actions1  
<cfelseif Expression is Value2>  
  Actions2  
</cfif>
```

As your actions become more complex, the `cfcase` is much easier to read and more efficient.



1-Minute Drill

- What ColdFusion tag is used for advanced conditional logic?
- What is the attribute for a `cfswitch` tag?

Switches are great in cases where you compare the value of many variables. Perhaps you're in a secure sight. Whenever people's login's time out, you redirect them back to a login page. After they successfully login, you might want to give users a link back to the place they were. You can do this by passing a URL variable to the login page. On the processing page of the login, you can use the switch statement to compare the URL variable and send the users back to the page they were viewing.

`cfdefaultcase`

As with the `cfif` group of tags, the `cfswitch` provides a way to create a default action if none of the cases are valid values. The name of the tag used to accomplish this is `cfdefaultcase` and it accepts no attributes. Let's look at the format of a `cfswitch` that uses `cfdefaultcase`.

-
- The `cfswitch` and `cfcase` ColdFusion tags are used for advanced conditional logic.
 - Expression is the attribute for a `cfswitch` tag.

Module 8: Making Decisions with ColdFusion 25

Hint

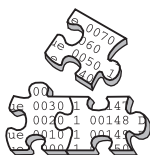
Cfcase tags after a cfdefaultcase tag will be ignored. Cfdefaultcase must be the last command in your switch statement.

L 8-13

```
<cfswitch expression="Expression">  
  
  <cfcase value="Value1">  
    Actions1  
  </cfcase>  
  
  <cfdefaultcase>  
    ActionsDefault  
  </cfdefaultcase>  
</cfswitch>
```

This code performs the first set of actions if the expression is equal to Value1. If the expression isn't equal to Value1, then it performs the default actions defined in the cfdefaultcase tag. Good to note is that our examples of the cfswitch tag have been simplified. In the real world, you usually won't want to use a cfswitch if you only have one or two options.

8



Project 8-3: Choosing Different Product Layouts

In Module 5, we used Project 5-1 to create a drill-down interface for displaying products. We started out by listing the genres, and then all the songs in that genre. Finally, based on the song selected, we displayed all the songs on a disc. This project revisits the past project. We use the cfswitch tag to create separate page layouts for the discs, based on the genre.

Step-by-Step

1. Our original drill-down interface started with a list of all genres. Selecting the genre, we displayed a list of all songs. Selecting the song, we displayed all the info from the disc on which the song was located. We want to modify two templates from Project 5-1 to accomplish our new task: change the SongList.cfm and DiscList.cfm. You can download both the original and new

26 ColdFusion 5: A Beginner's Guide

versions of these templates from the InstantColdFusion.com web site. First, look at SongList.cfm:

```
<!---
Description: A page to list all the songs in a genre

Entering: GenreList.cfm
Exiting: DisclList.cfm

Dependencies: N/A
Expecting: GenreID

Modification History
Date      Modifier      Modification
*****
10/23/2001 Jeff Houser, DotComIt Created
--->

<!-- a query to get all the songs for the
specific genre -->
<cfquery datasource="Chapter5" name="GetSongs">
  SELECT Song.*, Artist.Artist
  FROM Song, Artist, SongGenre
  WHERE SongGenre.GenreID = #url.GenreID# AND
        SongGenre.SongID = Song.SongID AND
        Song.ArtistID = Artist.ArtistID
</cfquery>

<!-- html header -->
<html>
<head>
  <title>Song List</title>
</head>
<!-- end HTML header -->

<!-- Start Main Content-->
<body>

<table>
  <!-- cfoutput over the query -->
  <cfoutput query="getSongs">
  <tr>
  <td>
```

Module 8: Making Decisions with ColdFusion 27

```
<a href="DiscList.cfm?DiscID=#getSongs.discID#">
  #GetSongs.Song#
</a>
</td>
<td>
  #GetSongs.artist#
</td>
</tr>
</cfoutput>
<!-- end cfoutput -->
</table>
<!-- end the content table -->

</body>
</html>
```



Because we're planning on displaying the disc information based on the genre it's located in, we need the GenreID once we get to the disclist.cfm page. As noted in the code, our disclist.cfm page doesn't currently get the GenreID passed on to it. We need to add it.

2. We made two code changes to our SongList.cfm page. First, we added an entry into the modification history, and then we added the GenreID to the DiscList.cfm URL.

8

```
<!---
Description: A page to list all the songs in a genre

Entering: GenreList.cfm
Exiting: DiscList.cfm

Dependencies: N/A
Expecting: GenreID

Modification History
Date      Modifier      Modification
*****
10/23/2001 Jeff Houser, DotComIt Created
11/13/2001 Jeff Houser, DotComIT Added cfswitch
statement for disc display
---->

<!-- a query to get all the songs for the
specific genre -->
```

28 ColdFusion 5: A Beginner's Guide

```
<cfquery datasource="Chapter5" name="getSongs">
  SELECT Song.*, Artist.Artist
  FROM Song, Artist, SongGenre
  WHERE SongGenre.GenreID = #url.GenreID# AND
         SongGenre.SongID = Song.SongID AND
         Song.ArtistID = Artist.ArtistID
</cfquery>

<!-- html header -->
<html>
<head>
  <title>Song List</title>
</head>
<!-- end HTML header -->

<!-- Start Main Content-->
<body>

<table>
  <!-- cfoutput over the query -->
  <cfoutput query="GetSongs">
    <tr>
      <td>
        <a href="DiscList.cfm?DiscID=#GetSongs.DiscID#
              &GenreID=#url.GenreID#">
          #GetSongs.Song#
        </a>
      </td>
      <td>
        #GetSongs.Artist#
      </td>
    </tr>
  </cfoutput>
  <!-- end cfoutput -->
</table>
<!-- end the content table -->

</body>
</html>
```

↑
Added GENREID to link

We added a modification history entry to say we were modifying the URL to pass the GenreID into the DiscList.cfm page. And when we create our links to the DistList.cfm page, we noted that the GenreID is now listed in the query string next to the DiscID.

Module 8: Making Decisions with ColdFusion 29

3. The next template we want to modify is the DisList.cfm template. You can see our original code, as follows:

```
<!---
Description: A page to list all songs on a disc
          based on the DiscID

Entering: SongList.cfm
Exiting: N/A

Dependencies: N/A
Expecting: DiscID

Modification History
Date      Modifier      Modification
*****
10/23/2001 Jeff Houser, DotComIt Created
--->

<!-- a query to get all the songs for the
      specific genre -->
<cfquery datasource="Chapter5" name="GetDisc">
  SELECT Discs.*, Song.*, Artist.Artist
  FROM Song, Artist, Discs
  WHERE Song.DiscID = #url.DiscID# AND
        Discs.DiscID = Song.DiscID AND
        Song.ArtistID = Artist.ArtistID
</cfquery>

<!-- html header -->
<html>
<head>
  <title>Disc List</title>
</head>
<!-- end HTML header -->

<!-- Start Main Content-->
<body>

<table>
  <!-- cfoutput over the query -->
  <cfoutput query="GetDisc" group="Disc">
  <tr>
    <td colspan="2">Album: #GetDisc.Disc#</td>
```

30 ColdFusion 5: A Beginner's Guide

```
</tr>
<tr>
  <td>Song</td>
  <td>Artist</td>
</tr>

<cfoutput>
<tr>
  <td>#GetDisc.Song#</td>
  <td>#GetDisc.Artist# </td>
</tr>
</cfoutput>
</cfoutput>
<!-- end cfoutput -->
</table>
<!-- end the content table -->

</body>
</html>
```

We want to make one main change to this page. We want to take the display portion of the template and modify it to display the data differently, based on the GenreID.

4. As always, we want to add an entry in the modification history:

```
<!---
Description: A page to list all songs on a disc
           based on the DiscID

Entering: SongList.cfm
Exiting: N/A

Dependencies: N/A
Expecting: DiscID

Modification History
Date      Modifier      Modification
*****
10/23/2001 Jeff Houser, DotComIt Created
11/13/2001 Jeff Houser, DotComIT Added cfswitch
           statement for disc display
--->
```

Module 8: Making Decisions with ColdFusion 31

```
<!-- a query to get all the songs for the
specific genre -->
<cfquery datasource="Chapter5" name="GetDisc">
SELECT Discs.*, Song.*, Artist.Artist
FROM Song, Artist, Discs
WHERE Song.DiscID = #url.DiscID# AND
Discs.DiscID = Song.DiscID AND
Song.ArtistID = Artist.ArtistID
</cfquery>

<!-- html header -->
<html>
<head>
<title>Disc List</title>
</head>
<!-- end HTML header -->

<!-- Start Main Content-->
<body>

<table>
<!-- cfoutput over the query -->
<cfoutput query="GetDisc" group="Disc">
<tr>
<td colspan="2">Album: #GetDisc.Disc#</td>
</tr>
<tr>
<td>Song</td>
<td>Artist</td>
</tr>

<cfoutput>
<tr>
<td>#GetDisc.Song#</td>
<td>#GetDisc.Artist# </td>
</tr>
</cfoutput>
</cfoutput>
<!-- end cfoutput -->
</table>
<!-- end the content table -->

</body>
</html>
```

32 ColdFusion 5: A Beginner's Guide

The entry simply refers to the modifications we're making, but the main bulk of our changes will reside in the cfswitch tag.

1. The next step is to change the display portion of the template to use a cfswitch statement. We'll make the current listing our default listing using the cfdefaultcase tag. Your updated code looks like this:

```
<!---
Description: A page to list all songs on a disc
            based on the DiscID

Entering: SongList.cfm
Exiting: N/A

Dependencies: N/A
Expecting: DiscID

Modification History
Date      Modifier      Modification
*****
10/23/2001 Jeff Houser, DotComIt Created
11/13/2001 Jeff Houser, DotComIT Added cfswitch
            statement for disc display
-->

<!-- a query to get all the songs for the
      specific genre -->
<cfquery datasource="Chapter5" name="GetDisc">
  SELECT Discs.*, Song.*, Artist.Artist
  FROM Song, Artist, Discs
  WHERE Song.DiscID = #url.DiscID# AND
         Discs.DiscID = Song.DiscID AND
         Song.ArtistID = Artist.ArtistID
</cfquery>

<!-- html header -->
<html>
<head>
  <title>Disc List</title>
</head>
<!-- end HTML header -->

<!-- Start Main Content-->
<body>
```

Module 8: Making Decisions with ColdFusion 33

```
<table>
<!-- cfoutput over the query -->
<cfoutput query="GetDisc" group="Disc">
<cfswitch expression="#url.GenreID#">

  <cfdefaultcase>
  <tr>
    <td colspan="2">Album: #getDisc.disc#</td>
  </tr>
  <tr>
    <td>Song</td>
    <td>Artist</td>
  </tr>

  <cfoutput>
  <tr>
    <td>#GetDisc.Song#</td>
    <td>#GetDisc.Artist# </td>
  </tr>
  </cfoutput>
  </cfdefaultcase>
</cfswitch>

</cfoutput>
<!-- end cfoutput -->
</table><!-- end the content table -->

</body>
</html>
```

8

We put the GenreID variable as the expression in our cfcase statement. ColdFusion takes the value of that variable and compares it to all the cfcase statements. Because we have only set up cfdefaultcase at the moment, your cfswitch isn't very useful at this point.

5. The last step in this project is to create our examples for the cfcase. I decided to set up a few examples for the cfcase statements. The code is shown here:

```
<!---
Description: A page to list all songs on a disc
          based on the DiscID

Entering: SongList.cfm
```

34 ColdFusion 5: A Beginner's Guide

```
Exiting: N/A

Dependencies: N/A
Expecting: DiscID

Modification History
Date      Modifier      Modification
*****
10/23/2001 Jeff Houser, DotComIt Created
11/13/2001 Jeff Houser, DotComIT Added cfswitch
statement for disc display
--->

<!-- a query to get all the songs for the
specific genre -->
<cfquery datasource="Chapter5" name="GetDisc">
SELECT Discs.*, Song.*, Artist.Artist
FROM Song, Artist, Discs
WHERE Song.DiscID = #url.DiscID# AND
Discs.DiscID = Song.DiscID AND
Song.ArtistID = Artist.ArtistID
</cfquery>

<!-- html header -->
<html>
<head>
<title>Disc List</title>
</head>
<!-- end HTML header -->

<!-- Start Main Content-->
<body>

<table>
<!-- cfoutput over the query -->
<cfoutput query="GetDisc" group="Disc">
<cfswitch expression="#url.GenreID#">

<cfcase value="18">
<!-- if the genre is TV -->
<tr>
<td>Album: </td>
<td>Song</td>
<td>Artist</td>
```

Module 8: Making Decisions with ColdFusion 35

```
</tr>

<cfoutput>
<tr>
  <td>#GetDisc.disc#</td>
  <td>#GetDisc.Song#</td>
  <td>#GetDisc.Artist# </td>
</tr>
</cfoutput>
</cfcase>

<cfcase value="14">
<!-- if the genre is rap -->
<tr>
  <td colspan="2" bgcolor="##000000">
    <font color="##ffffff">
      Album: #GetDisc.Disc#
    </font>
  </td>
</tr>
<tr>
  <td bgcolor="##000000">
    <font color="##ffffff">
      Song
    </font>
  </td>
  <td bgcolor="##000000">
    <font color="##ffffff">
      Artist
    </font>
  </td>
</tr>

<cfoutput>
<tr>
  <td bgcolor="##000000">
    <font color="##ffffff">
      #GetDisc.Song#
    </font>
  </td>
  <td bgcolor="##000000">
    <font color="##ffffff">
      #GetDisc.Artist#
    </font>
  </td>
</tr>
</cfoutput>
```

36 ColdFusion 5: A Beginner's Guide

```
</td>
</tr>
</cfoutput>
</cfcase>

<cfdefaultcase>
<tr>
  <td colspan="2">Album: #GetDisc.Disc#</td>
</tr>
<tr>
  <td>Song</td>
  <td>Artist</td>
</tr>

<cfoutput>
<tr>
  <td>#GetDisc.Song#</td>
  <td>#GetDisc.Artist# </td>
</tr>
</cfoutput>
</cfdefaultcase>
</cfswitch>

</cfoutput>
<!-- end cfoutput -->
</table><!-- end the content table -->

</body>
</html>
```

For the TV genre, GenreID 18, I set it up so the disc title isn't separated from the song and artist name. The disc title is displayed once for every song and artist.

I set up something different for Genre 14, the rap genre. This layout is distinguished by the colors used to display the information. The background color of the table cells is set to black and the text color is shown as white using the HTML font tag. You can define numerous other layouts to suite your needs.

Project Summary

This project demonstrates how we can use the cfswitch tags as an advanced conditional statement. Examining the code in this project, you might think the final template is getting fairly large and, therefore, might be difficult to manage, especially if you were to create separate layouts for all 20 genres we have in the database. You might be right. In Module 10, we examine code modularization and ways to help simplify complicated templates like this one.

Module Summary

Conditional logic is an important programming concept. Decisions are made, whether with or without user input in all applications. Understanding how to operate with cfifs and cfswitch statements is important. The next chapter explores another important programming concept: looping constructs.

A. Cfswitch

Mastery Check

8

1. A function used to check for the existence of a variable is ____.

2. Create the truth table for this statement: $((\text{NOT } A) \text{ and } ((X \text{ and } Y) \text{ or } Z) \text{ EQV } ((X \text{ or } Y) \text{ and } Z))$.

3. Based on Project 8-3, expand your code to include additional genres layouts. You can design the layouts in any way you choose.

38 ColdFusion 5: A Beginner's Guide

Mastery Check

4. What are the primary tags used in conjunction with cfif statement?
 - A. cfthen
 - B. Cfif
 - C. Cfelse
 - D. Cfelseif
 - E. All of the above
5. A Boolean variable is one that can have only _____ values.
6. What is the tag used to create a default condition in a cfswitch statement?
 - A. Cfswitch
 - B. Cfif
 - C. Cfdefaultcase
 - D. Cfcase
7. Which are Boolean operators and which are decision operators?
 - A. NOT
 - B. EQUAL
 - C. AND
 - D. EQV
 - E. GT
 - F. LT
8. What is used to list all possible answers to a Boolean expression?

Module 8: Making Decisions with ColdFusion 39

Mastery Check

9. The AND operator returns a ____ value if both the conditions are true.
- 10 The OR operator returns a ____ value if the two conditions are opposites of each other.

