



Conditional Statements in Dynamic Text

Zephyr Associates, Inc.
P.O. Box 12368
Zephyr Cove, NV 89448

775-588-0654
800-789-5323
Fax 775-588-8423

www.styleadvisor.com

Getting Iffy

What if you want something to happen in your script only in certain circumstances? You need code to make it happen, but you also need a way to test to see if the circumstances are right. Well, Dynamic Text (DT) has a way to do that. For consistency, I'm going to call conditional statements in DT "If/Then" statements. Here's the simplest form of an If/Then statement:

```
if true then
  Execute this code
end
```

That's fake code to explain the format of If/Then statements. You start with the keyword "if" (note the capitalization...it's required to be lower case) followed by a test for some condition (usually a comparison, like checking to see if a variable equals some important value) followed by the keyword "then", followed by some code, then concluding with the keyword "end". Everything between "then" and "end" will be executed if the test passes and will be skipped over if the test fails.

A real If/Then statement looks more like this:

```
<%
var = 13

if var > 10 then
  Display("The variable passed my test")
end
%>
```

This will only display the sentence "The variable passed my test" if "var > 10" is true. In the example, 13 is greater than 10 so the sentence will be displayed. If we modified the variable to be equal to 5 then this block of code wouldn't display anything.

The shortest test you can do in an If/Then statement is to check to see if a variable has a value. A variable which has not yet been assigned a value or which has been assigned a missing value evaluates to false in an If/Then statement. One that has been assigned a value, any value other than nil, evaluates to true in an If/Then statement.

```
<%
if var then
  Display("The variable var has been assigned")
end
%>
```

Since this block doesn't set a value for "var" the sentence will not be displayed.

```
<%  
var = Value(1, "Style Drift", "Page 2|Scan #1")  
  
if var then  
  Display("The variable var has been assigned")  
end  
%>
```

This block will display the sentence only if the first manager actually has a Style Drift score in the Scan View on Page 2. If the first manager doesn't have enough history to compute a Style Drift score the sentence won't be shown. This is a handy way to check for missing values in a Scan.

What if you have a list of conditions to check and a different set of instructions to execute if each is true? Well, it's perfectly legal to just write a series of If/Then statements in sequence.

```
<%  
StyleDrift = Value(1, "Style Drift", "Page 2|Scan #1")  
  
if StyleDrift then  
  Display("The variable StyleDrift has been assigned")  
end  
  
if StyleDrift < 10 then  
  Display("Very good Style Drift score")  
end  
  
if StyleDrift > 30 then  
  Display("Getting driftier")  
end  
  
if StyleDrift > 50 then  
  Display("Terrible Style Drift score")  
end  
%>
```

You may have noticed that values for "StyleDrift" could easily match more than one of these tests, so you might end up with several of these sentences displayed. Well, you're allowed to nest If/Then statements inside each other, so that each condition would only be tested if the previous test passed. There's another way to do it though. If you only have two mutually exclusive results in mind you can use the keyword "else" to define another block of code to execute if the test fails. If you have several mutually exclusive tests in mind you can use the keyword "elseif" to define more tests.

```
<%
StyleDrift = Value(1, "Style Drift", "Page 2|Scan #1")
if StyleDrift then --Only check further if StyleDrift has a value
  if StyleDrift < 10 then
    Display("Very good Style Drift score")
  else
    Display("Not as good a score")
  end
end
end
%>
```

In this example we're nesting tests for good and not-so-good scores inside a test for whether or not our manager has a score at all. First we assign the manager's Style Drift score to the variable "StyleDrift". Then we check to see if this manager has a score. If he does, we check to see if he has a score lower than 10 and compliment him if that test passes. Otherwise we chastise him a little. Values of 10 or higher result in the string "Not as good a score". Values below 10 result in the string "Very good Style Drift score".

If we have a list of escalating comments depending on values for the manager's style drift score we'll need to do a set of tests using "elseif".

```
<%
StyleDrift = Value(1, "Style Drift", "Page 2|Scan #1")
if StyleDrift then --Only check further if StyleDrift has a value
  if StyleDrift > 50 then
    Display("Terrible Style Drift score")
  elseif StyleDrift > 30 then
    Display("Style Drift is getting driftier")
  elseif StyleDrift > 10 then
    Display("Not a great Style Drift score")
  else
    Display("Very good Style Drift score")
  end
end
end
%>
```

This example does what the first multiple test example was trying to do, without ever displaying more than one sentence. First it assigns the manager's Style Drift score to the variable "StyleDrift", then it checks to see if there's any point in proceeding because if there's no value the other tests are a waste of time. Once we know the manager has a score we check for a score greater than fifty. If it passes that test we don't pull any punches and tell the manager the score is terrible. If it fails that test we check to see if the score is greater than 30. If it is we warn that the score is pushing the limits we'll tolerate. If it fails that test we do another test to see if it's greater than 10. If it's greater than ten we point out that it could be better. If it's anything else (that is, if it's below ten) we compliment the score.

Complicated tests

What if you want to see if a value falls within an acceptable range? For example, what if you want to see if a manager falls in the Mid Cap style? That would mean that you need to see if the value is between the lower bound for Mid Cap and the upper bound for Mid Cap. You could do this with two tests, one for greater than or equal to the lower bound and another for less than or equal to the upper bound, but you can combine the two tests into one with the “and” operator. This lets you put multiple tests between “if” and “then” and only execute the following code if all tests pass.

```
<%  
SmallLarge = Value(1, "Small Large", "Page 2|Scan #1")  
  
if SmallLarge >= -0.33 and SmallLarge <= 0.33 then  
  Display("Mid Cap")  
else  
  Display("Not Mid Cap")  
end  
  
%>
```

This example gets the “Small Large” statistic for the first manager from the scan then in a single test checks to see if the value falls between -0.33 and 0.33. It compares the variable “SmallLarge” to the lower bound of -0.33 then it compares the same variable to the upper bound of 0.33. If and only if *both* tests pass it displays the string “Mid Cap”. Otherwise it displays “Not Mid Cap”.

You can also execute code if either of a pair of tests passes by using the “or” operator. For example, if you were trying to find managers who are not Mid Cap, you could test for values below or above the boundaries like this:

```
<%  
SmallLarge = Value(1, "Small Large", "Page 2|Scan #1")  
  
if SmallLarge <= -0.33 or SmallLarge >= 0.33 then  
  Display("Not Mid Cap")  
else  
  Display("Mid Cap")  
end  
  
%>
```

This version checks to see if the value of SmallLarge is below the lower bound of -0.33 or above the upper bound of 0.33 and if *either* is true it displays “Not Mid Cap”.

You can string together as many tests as you can stand in one line combined with “and” and “or”. Just remember that “and” means the tests on either side must both pass. The “or” operator means that only one of the tests around the operator must pass.

You can also invert the result of a test by preceding it with the “not” operator. In earlier examples we checked to see if a variable had a value and only did more checking if it did. You can also check to see if it doesn’t have a value by saying something like this:

```
if not StyleDrift then
  Display("N/A")
end
```