WarmFusion Reference Manual

Robert Dong (rd2439)

Table of Contents

1. Introduction

WarmFusion is a web application development language based on the earlier language ColdFusion. WarmFusion has syntax similar to HTML/XML and allows users to place its tags inline with existing HTML/XML code. The output generated by a WarmFusion application is often viewed in a web browser.

2. Lexical Conventions

2.1. Comments

The characters `<!---` introduce a comment, which terminates with the characters `--->`.

2.2. Identifiers

An identifier is a sequence of letters and digits. The first character must be a letter. The underscore counts as a letter. Identifiers are case-sensitive.

2.3. Keywords

All identifiers beginning with the letters "`wf`" are reserved keywords. The following identifiers are also reserved keywords: `NOT`, `LT`, `LTE`, `GT`, `GTE`, `EQ`, `NEQ`.

2.4. Numerical Constants

A numerical constant consists of an integer part and/or a decimal part. The integer part consists of one or more digits. The decimal part consists of a "`.`" followed by one or more digits.

2.5. String Constants

A string constant is a sequence of characters surrounded by double quotes.

3. Types

WarmFusion supports two fundamental types: number and string. A numerical constant as described above is considered a number while a string constant is considered a string.

4. Expressions

4.1. Unary Operators

4.1.1. – expression

The result is the negative of the expression. The type of the expression must be number.

4.1.2. *NOT expression*

The result of the logical negation operator NOT is 1 if the value of the expression is 0, or 0 if the value of the expression is non-zero. The type of the result is numeric. The type of the expression must be number.

4.2. Multiplicative Operators

4.2.1. *expression * expression*

The binary * operator indicates multiplication. The type of both expressions must be number.

4.2.2. *expression / expression*

The binary / operator indicates division. The type of both expressions must be number.

4.3. Additive Operators

4.3.1. *expression + expression*

The result is the sum of the expressions. The type of both expressions must be number.

4.3.2. *expression – expression*

The result is the difference of the expressions. The type of both expressions must be number.

4.4. Relational Operators

4.4.1. *expression LT expression*

4.4.2. *expression GT expression*

4.4.3. *expression LTE expression*

4.4.4. *expression GTE expression*

The operators LT (less than), GT (greater than), LTE (less than or equal to) and GTE (greater than or equal to) all yield 0 if the specified relation is false and 1 if it is true. The type of both expressions must be number.

4.5. Equality Operators

4.5.1. *expression EQ expression*

4.5.2. *expression NEQ expression*

The EQ (equal to) and the NEQ (not equal to) operators yield 0 if the specified relation is false and 1 if it is true.

4.6. *expression % expression*

The binary % operator yields the remainder from the division of the first expression by the second. The type of both expressions must be number.

4.7. *expression & expression*

The result is a string of the two concatenated expressions. The type of both expressions must be string.

5. Statements

Except as described, statements are executed in sequence. If a string is in the code that is not categorized as a statement or function declaration, that string is outputted to the user (i.e. HTML code is outputted to the user).

5.1. Assignment Statement

Assigning a value to a variable takes place as follows:

```
<wfset lvalue = expression />
```

5.2. Conditional Statement

The two forms of the conditional statement are:

```
<wfif expression>
     statement
</wfif>
```

and

```
<wfif expression>
     statement
<wfelse>
     statement
</wfif>
```

In both cases, the expression is evaluated and if it is non-zero, the first substatement will be executed. In the second case, the second substatement is executed if the expression is 0. The `wfelse` statement is associated with the most recent `wfif` statement.

5.3. Loop Statements

5.3.1. Conditional Loop

A conditional loop has the form:

```
<wfloop condition="expression">
      statement
</wfloop>
```

The substatement is executed repeatedly as long as the value of the expression remains non-zero. The test takes place before each execution of the statement.

5.3.2. *Loop with counter*

A loop with a counter has the form:

```
<wfloop index=”variable” from=”number1” to=”number2”>
      statement
</wfloop>
```

The variable is initialized to number1. The substatement will be executed repeatedly as long as the value of variable is less than or equal to number2. The test takes place before each execution of the statement. After the substatement execution, variable is incremented by one.

5.4. Argument Statement

A function specifies the names of its arguments in the following form:

```
<wfargument name=”argName” />
```

A `wfargument` tag may only follow the opening `wffunction` tag or another `wfargument` tag.

5.5. Return Statement

A function returns to its caller by means of the `wfreturn` statement, which has the form:

```
<wfreturn expression />
```

5.6. Output Statement

Output statements allow values of expressions to be evaluated and displayed to the user. Output statements have the following form:

```
#expression#
```

Output statements are often used to display the value of a variable to the user.

6. Function Declarations

A function declaration has the form:

```
<wffunction name=”NameOfFunction”>
   <wfargument name=”arg1” />
   <wfargument name=”arg2” />
   …
   <wfargument name=”argN” />

   statement

   <wfreturn 0 />
</wffunction>
```

A simple example of a complete function definition is:

```
<wffunction name="CalculateAreaOfSquare">
   <wfargument name="sideLength" />
   <wfset area = sideLength * sideLength />
   <wfreturn area />
</wffunction>
```

7. Scope Rules

All variables share the same scope. Even if a variable is first set within a function, that variable will be visible after the function has finished executing. Similarly, if a variable is first set within a loop, the variable can still be used after the loop has completed.

8. Example

An example of WarmFusion code is shown below. WarmFusion code is mixed in with the HTML code which allows rapid development of simple web applications. The code declares two functions `CalculateAreaOfSquare` and `CalculateDiagonalOfSquare`, both of which take one argument. An example of a comment in WarmFusion is shown in the second function.

An HTML table is printed out and WarmFusion loops through the numbers 1 through 10 to display each row. In the table, there is a conditional statement which highlights a row gray if it is odd. The two functions are called in the loop to create the table of data. As shown, all WarmFusion statements are HTML/XML tags that begin with "`wf`".

8.1. Example Code

```
<wffunction name="CalculateAreaOfSquare">
      <wfargument name="sideLength" />
      <wfset area = sideLength * sideLength />
      <wfreturn area />
</wffunction>

<wffunction name="CalculateDiagonalOfSquare">
      <wfargument name="sideLength" />
      <!--- Length of a side multiplied by the square root of 2 --->
      <wfset diagonal = sideLength * Sqr(2) />
      <wfreturn diagonal />
</wffunction>

<h1>Square Information:</h1>
<table border="1">
      <tr>
            <th>Side Length</th>
            <th>Area of Square</th>
            <th>Diagonal of Square</th>
      </tr>

      <wfloop index="i" from="1" to="10">
```

```
            <wfif i % 2 EQ 1>
                  <wfset rowBgColor = "lightgray" />
            <wfelse>
                  <wfset rowBgColor = "white" />
            </wfif>

            <tr bgcolor="#rowBgColor#">
                  <td>#i#</td>
                  <td>#CalculateAreaOfSquare(i)#</td>
                  <td>#CalculateDiagonalOfSquare(i)#</td>
            </tr>
      </wfloop>
</table>
```

8.2. Output From Example

HTML code would be generated from the previous example. It would be shown in a web browser as follows:

Square Information:

Side Length	Area of Square	Diagonal of Square
1	1	1.41421356237
2	4	2.82842712475
3	9	4.24264068712
4	16	5.65685424949
5	25	7.07106781187
6	36	8.48528137424
7	49	9.89949493661
8	64	11.313708499
9	81	12.7279220614
10	100	14.1421356237