Literals and String Expressions

A common practice in alcode is to inject variables into a string that expose their value on output. For example:

```
$file_lines = load_records();
console(“There were {$file_lines} records loaded”);
// Outputs There were 24 records loaded
```

The form:

```
{$var}
```

is actually more extensive than just simple variables. Formally the {} structure instructs the compiler to delay the evaluation of the expression until run time. So the {} can be considered as this:

```
{<an expression that is evaluated at run time>}
```

Consider these fragments; the first is the common usage:

```
$days_this_month = date('t');
console ("There are {$days_this_month} days this month");
// Outputs There are 31 days this month
```

But this also has the same effect:

```
console ("There are {date('t')} days this month");
```

The expression must evaluate (successfully) to a value, but otherwise there are no restrictions, so this will work:

```
$one_2 = 2;
$two_2 = 2;
console("Add {$one_2} to {$two_2} gives {$one_2 + $two_2}");
// Outputs Add 2 to 2 gives 4
```

The extension of this facility is provided in the **literal** clause:

This is common practice:

```
$c = literal {
   Hello there World
};
console($c);
```

Will output:

Hello there World

That is, the literal content is expected to be a string – that is evaluated at run-time – plus it maintains all white space (blanks, space, tabs, newlines etc).

But as it is evaluated at run time then this is allowed:

```
$a = "Hello";
$b = "World";

$c = literal {
   $a there $b
};

console($c);
```

Will output the same as above; however, the following will not give you what you might expect:

```
$c = literal {
   {$a} there {$b}
};
```

Will actually produce:
{Hello} there {World}

That is, the clause {} **is not recursive**.

Formally, the literal clause is this:

```
literal <string_expression>
```

So this is valid:

```
$c = literal “Hello World”;
```

This will be evaluated at compile-time (the string is not encased in the delay {} clause) and so will output: Hello World. However, this:

```
$c = literal “{$a}+ ‘ ‘ + {$b}”
```

Will produce this: Hello + '' + World
That is, the string will request first level evaluation of string elements (the $a and $b) but will treat the rest of the literal content as a literal.

Consider these examples:

Example 1

```
$c = "{$a + ' there ' + $b}";
console($c);
// Hello there World
```

Example 2

```
$c = literal "{$a + ' there ' + $b}";
console($c);
// Hello there World
```

Example 3

```
$c = literal {$a + ' there ' + $b};
console($c);
// Hello + ' there ' + World
```

The second example says, the literal is followed by a string, the string is evaluated at compile time to be: {$a + ‘there’ + $b}, which says delay the rest of the evaluation until run-time, at run-time this is the string expression: $a + ‘there’ + $b, which would be the same as any other string expression like: $c = $a + ‘there’ + $b;

The third example however, says plant the literal string Hello + ‘there’ + World because the literal strips the outer { } clause to treat the content as a literal string. Therein is the purpose of the literal clause, to form a string that is not over evaluated.

In general usage the literal clause is used like this:

```
$use_db = “MI_CSS”;

$query = literal {
  SELECT * FROM  [$use_db].[dbo].[my_table]
  };
```

That is, the $use_db is evaluated without the { } clause because the whole is treated as a { } clause, but only once – remember no recursion.