



Number Formatting 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

Number Formatting in Dynamic Text

StyleADVISOR tries to do a good job in formatting numbers displayed by Dynamic Text (DT) by default. Decimals are limited to two places, but trailing zeros are not used by default. Numbers over 1000 show commas by default. Fractions less than 1 but greater than zero do not show a leading zero before the decimal point. There are default date formats for both dates and time periods.

You can override these defaults for an individual number, for all numbers in a block of code, or for all DT in a workbook.

:Format{}

To set an individual number's format in DT, use the `:Format{}` command. You can place one or more keywords inside the squiggly brackets which tell DT how to display your number. The syntax for `:Format{}` is:

```
Display(0.4:Format{Keyword1 = value, Keyword2 = value, ...})
```

For example (by default the code below will display “.4”):

```
<%  
MyNum = 0.4  
Display(MyNum)  
%>
```

If you prefer to see a leading zero and trailing zeros to pad out to a fixed number of places, “0.40”, you would add `:Format{}` with the appropriate keywords:

```
<%  
MyNum = 0.4  
Display(MyNum:Format{minLeadingDigits = 1, trailingDigits =  
2})  
%>
```

Note that the capitalization in the keywords for `:Format{}` is optional. “MinLeadingDigits”, “minleadingdigits”, “minLeadingDigits”, and “MINLEADINGDIGITS” are interchangeable. Spelling is strictly enforced, but `:Format{}` is insensitive to case. As a convention I’m going to use a rather eccentric capitalization scheme, that I’ll explain later.

“minLeadingDigits” is the minimum number of digits acceptable before the decimal. DT will insert zeros to the left of your number to pad out to the value of “minLeadingDigits”. For example:

```
<%  
MyNum = 1.4  
Display(MyNum:Format{minLeadingDigits = 4, trailingDigits =  
2})  
%>
```

will result in “0,001.40” but

```
<%  
MyNum = 12345.4  
Display(MyNum:Format{minLeadingDigits = 4, trailingDigits =  
2})  
%>
```

will not bother to pad with zeros because the number 12345.4 is higher than the minimum you set. You’ll get “12,345.40” in this case. Here is a list of all of the keywords understood by :Format{} with an example of each:

```
{decimal = "."}  
{exponent = "Uppercase"}  
{maxTrailingDigits = 2}  
{minLeadingDigits = 0}  
{minusSign = "Leading"}  
{percent = false}  
{thousand = ","}  
{trailingDigits = ""}
```

I’ve included the default values above. In each example below I’ll list the options.

“decimal”

```
<%  
MyNum = 1234.5678  
Display(MyNum:Format{decimal = ","})  
%>
```

The “decimal” keyword allows you to change the character used for decimal from the default period character. The example will return the confusing number “1,234,57”. “decimal” is often used in conjunction with the “thousand” keyword which lets you replace the default comma with another character like the period.

“exponent”

```
<%  
MyNum = 1234.5678  
Display(MyNum:Format{exponent = "lowercase"})  
%>
```

This will show the result in exponential notation with the “e” shown in lower case like this: “1.23e+3”. The options for “exponent” are “uppercase” and “lowercase”. This

type of notation is rare in financial reports, so you may safely forget this one until it comes up.

“maxTrailingDigits”

```
<%  
ShortNum = 1234.56  
MyNum = 1234.5678  
Display(ShortNum:Format{maxTrailingDigits = 3} .. "|")  
Display(MyNum:Format{maxTrailingDigits = 3})  
%>
```

This sets the limit for how many decimal places will be displayed. If the number has fewer decimal places it will not be affected (no trailing zeros for padding). If the number has greater than the maximum specified it will be rounded to the maximum. The example will show “1,234.56” then on a new line “1,234.568”.

“minLeadingDigits”

```
<%  
MyNum = 1234.5678  
ShortNum = 0.5678  
Display(MyNum:Format{minLeadingDigits = 1} .. "|")  
Display(ShortNum:Format{minLeadingDigits = 1})  
%>
```

This tells DT to pad the number to the left of the decimal with zeros if there are fewer places than the minimum specified. The example will show “1,234.57” then on a new line “0.57”. The first number is greater than the minimum so no padding was required. The second number had a leading zero which it would have lost by default (“minLeadingDigits” defaults to zero).

“minusSign”

```
<%  
MyNum = -1234.5678  
Display(MyNum:Format{minusSign = "Parentheses"})  
%>
```

The options for “minusSign” are “Leading”, “Trailing”, and “Parentheses”. The example will display “(1,234.57)”. “Trailing” places the hyphen after the negative number.

“percent”

```
<%  
MyNum = 1234.5678  
Display(MyNum:Format{percent = true})  
%>
```

This displays a percent sign following the number if true and doesn't if false. The example will display “1,234.57%”.

“thousand”

```
<%  
MyNum = 1234.5678  
Display(MyNum:Format{thousand = "."})  
%>
```

This sets the symbol used as the thousands separator. The above example will result in this difficult-to-figure-out number: “1.234.57”. Normally, if you change the thousands separator to period, you'd also change the decimal symbol to a comma at the same time like this:

```
<%  
MyNum = 1234.5678  
Display(MyNum:Format{thousand = ".", decimal = ","})  
%>
```

So that you'd get “1.234,57”.

“trailingDigits”

```
<%  
MyNum = 1234.5678  
ShortNum = 1234.56  
Display(MyNum:Format{trailingDigits = 3} .. "|")  
Display(ShortNum:Format{trailingDigits = 3})  
%>
```

This is similar to “maxTrailingDigits” but in this case it doesn't mean, “Up to max” it means “Exactly this many”. The above example will round the first number to “1,234.568” then on a new line pad the second number to “1,234.560”. If “trailingDigits” is set it will take priority over “maxTrailingDigits”.

Note that you can sometimes get away with just adding :Format{} to a number while you're displaying it, or even when you're assigning it to a variable (var = 1234.5678:Format{trailingDigits = 3} for example), but there are exceptions, so it's easier to just always use :Format{} on variables when you display them the way I've done in the examples.

By now you've probably noticed that some of these would be more convenient if they could apply to more than one number at a time. A block of code that displays several percentages would be a little tiring if you had to add `:Format{percent = true}` to each number. A block of code that displays several dollar amounts would be a lot simpler if you could set the number of trailing digits for every number rather than once for each.

The next section is about how DT addresses this.

Format Variables

We've provided a set of variables with which you can set format values for an entire block of code rather than an individual item. After the variable is set, any number displayed by the block will conform to that formatting without using `:Format{}`. These variables have the same names as the keywords for `:Format{}` so that you don't have to remember a different set. We'll give examples of using these variables later, but here's the list:

```
decimal = "."
exponent = "Uppercase"
maxTrailingDigits = 2
minLeadingDigits = 0
minusSign = "Leading"
nilDisplay = "N/A"
percent = false
thousand = ","
trailingDigits = ""
```

Important note: Format variables are *case sensitive*. The spelling shown is required, including the lack of an initial cap. Since this version works okay in `:Format{}` it's easiest to just memorize the way these look and always spell them this way whether you're using `:Format{}` or the variables.

Did you notice that there's an extra one in there called "nilDisplay"? This is to make it easy to modify how invalid numbers or missing values will display in your DT. By default it's set to "N/A". If you'd rather have your DT say "Missing Value" or "--" or whatever, just set this variable. It's called that because a missing value or a variable that hasn't been assigned has a type of "Nil". Whenever DT tries to display something with a type of "Nil" it will replace it with whatever string you assign to "nilDisplay".

The options and defaults for these variables are identical to the keywords in `:Format{}` so we won't go over those again, just how and when to use them.

```
<%
FirstNum = 1234.456
SecondNum = 7890.123
thousand = "."
decimal = ","
```

```
Display(FirstNum .. "|")
Display(SecondNum)
%>
```

This block is functionally equivalent to:

```
<%
FirstNum = 1234.456
SecondNum = 7890.123

Display(FirstNum:Format{ decimal = ",", thousand = "."} ..
"|")
Display(SecondNum:Format{decimal = ",", thousand = "."})
%>
```

in that it will display the same result, but it's a lot less complicated, and if you add another number to display, the first block will automatically use the comma for decimals and the period for thousands because you've changed the default for this entire block. Both blocks should display "1.234,46" then on the next line "7.890,12". In the second example, the new formatting only applies to the numbers with the :Format{} keywords set. If you added another Display() there and you didn't do a :Format{} to it, it would use the system defaults and the number would be formatted differently.

Note that you can change the default within a block of code as many times as you'd like, so in theory, you could change individual numbers using these variables instead of :Format{}, but that would make your code look ugly and we wouldn't like that, would we? No, you're better off using :Format{} for formatting single values and the variables only if you want to display several values all using the same formatting.

Workbook Format Variables

What if you want all of the DT in your workbook to use the same number formatting? If you define these variables as Workbook Variables, they'll apply to every block of code in your workbook. You will have, in effect, changed the defaults for this workbook.

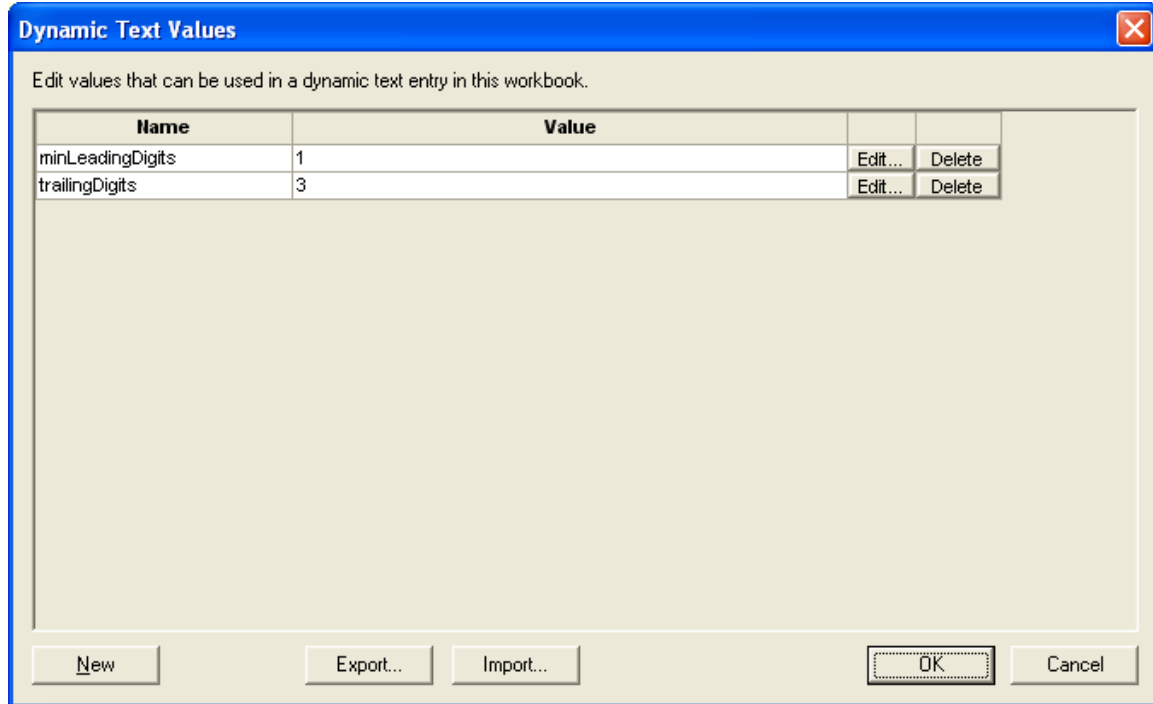
Suppose you want to require all your DT to display a leading zero and to round to three decimal places, rather than just an individual number as we've done in earlier examples.

Try the following block of code in a new workbook:

```
<%
MyNum = 0.1234
Display(MyNum)
%>
```

This will show ".12".

Now Edit|Dynamic Text Values for Workbook so that it looks like this:



When you hit OK, your block should update to show “0.123” Congratulations, you’ve changed the defaults for your workbook. All of your DT will comply with the formatting you’ve set, unless you override them.

Order Of Precedence For Formatting

Here’s the order in which number formatting is done by DT:

1. System defaults
2. Workbook-wide settings done by setting values in Edit|Dynamic Text Values for workbook.
3. Block-wide settings done by setting values within a block of code.
4. Individual settings using `:Format{ }`

The last one executed wins. This is easier to visualize with a concrete example.

Suppose you’re making a new workbook with tables of DT values in it. Most of the values you intend to display are dollar values and you’d like them to align at the decimal point. You decide that workbook-wide you should set “trailingDigits” to 2 and “minLeadingDigits” to 1, overriding the default “maxTrailingDigits” of 2 and “minLeadingDigits” of 0. If you right-align your table cells, your dollar values will now have their decimal points aligned. Pretty slick, huh?

Okay, now you realize that you need to have one block of code that displays a couple of percentages, and you need more decimal places for them. Within that block of code you can set “percent” to true and “trailingDigits” to 5 by doing this:

```
<%
percent = true
trailingDigits = 5

StdDev = Value( 1, 'Standard|Deviation', 'Page 2|Custom Table #1'
)
BMStdDev = Value( BenchmarkName("Market"), 'Standard|Deviation',
'Page 2|Custom Table #1' )

Display(ManagerName(1) .. "'s Std Dev is " .. StdDev .. "|")
Display(BenchmarkName("Market") .. "'s Std Dev is " .. BMStdDev)
%>
```

This will change the formatting for both numbers in this block, but won’t affect any other blocks. The rest will remain looking like dollars and cents, but this block will put out something like this: “First Manager’s Std Dev is 23.39690%” then on the next line something like this: “S&P 500’s Std Dev is 14.66689%”.

In another block of code you notice that you need to display last year. Here’s an attempt to do this:

```
<%
ThisYear = Today():Format("YYYY")
LastYear = ThisYear - 1
Display(LastYear)
%>
```

What’s unfortunate about this is that when you do math (like subtracting 1 from the current year) the output gets the default formatting. In this workbook you’ll end up with something that looks like this: “2,007.00”. Nobody displays years like that. You’d rather see “2007” with no decimal places and no comma. You don’t want to mess with the formatting for the entire workbook, and farther down the line, the block may need to display some dollars and cents values as well. You just want to format the variable “LastYear”. Here’s how you’d do that:

```
<%
ThisYear = Today():Format("YYYY")
LastYear = ThisYear - 1
Display(LastYear:Format{thousand = "", trailingDigits = 0})
%>
```

That will clean up the year for you.