Chapter 6

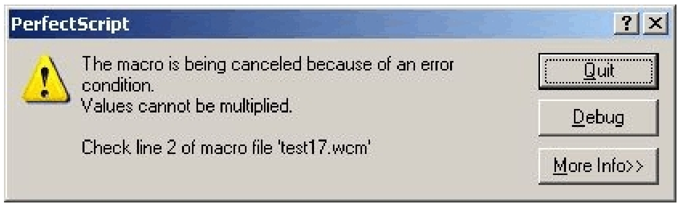
Math Routines

Links: The Chapter Name, above, moves to the next chapter. All page header links go to the contents menu. Within the chapter, Topic Titles returns here. Other Red Links are to other locations in this paper. Blue Links are to web sources.

This chapter covers a few of the many math routines possible in WordPerfect macros. It gives some basics but it is by no means comprehensive. For example, it does not discuss the concepts of "expressions" or "operator precedence" at all, except to say that, for purposes of this manual, an "expression" is a statement containing an assignment of some value to something, usually a variable (which I tend to call a "formula"), e.g., x=1 or var3=var1+var2. Concerning expressions, see http://jdan.com/perfectscript/macros/ch04.htm; for operator precedence, see http://jdan.com/perfectscript/macros/ch04.htm#_VPID_04_24. Even so, this chapter should give you a good working knowledge of math operations you'll likely need to know.

! MATH OPERATORS. * is used for multiplication; / is used for division; - is used for subtraction of numbers and reduction of strings; + is used for addition of numbers and concatenation of strings; % is used for floating point modulus division and returns the floating point division remainder; MOD is used for integer division and returns the integer division remainder; DIV is used for integer division and returns the integer portion; and ** is used for exponentiation or raising a number to a power. Only *, /, +, and - are discussed in this manual. Note that both + and - are string operators, as well.

! GENERAL RULES. Math routines operate upon "numeric" (number), not "string" (text) values. Subject to what's said in Automatic Data Conversion, below, when using math it's extremely important that the values you are using numeric, not string, values. Otherwise, error or an unintended result may occur and, if so, in the absence of some error trapping routine, the macro will fail or work differently that intended. An error dialog similar to the picture below most probably means that one or more of the values or variables containing values are not numeric.


PerfectScript

The macro is being canceled because of an error condition.
Values cannot be multiplied.

Check line 2 of macro file 'test17.wcm'

If var1="$1" and var2=1, var1 contains a string value which cannot convert to a numeric value whereas var2 contains a numeric value.

While WordPerfect will sometimes "know" that a variable containing a value which could be either a string value or a numeric value (e.g., var="1" and var=1 may be interpreted by PerfectScript as numeric, that's not always so in different WordPerfect versions. So, if you're interested in macros working the same way from WordPerfect 6.1 forward, be sure you are using numeric and not string values when doing math.

Since I routinely want my macros to work cross-WordPerfect versions whenever possible, I tend to use routines which reduce math operations to the lowest common denominator in the various WordPerfect versions from 6.1 to the current version. That predisposition affects some of what I say in this Chapter.

! AUTOMATIC DATA CONVERSION. Probably beginning with WordPerfect 7.0 (it may have begun in WordPerfect 6.1 but I don't think so), PerfectScript treats "strings" which would be the equivalent of numbers (but for the fact that they are strings) as numeric, if called upon to do so under certain circumstances. For example, if var="1.5", var2=var*2 returns the numeric value of 3.0 to var2; and if var1="1.5" and var2=50, var3=var2/var1 returns the numeric value of 31.25. Similarly, a numeric value may be treated as a string under certain circumstances. For example, if var=3 (numeric), var2="$"+var returns the string value of "$2". This section describes what will, or may, or cannot happen concerning automatic data conversion.

• The + and - operators. These operators aren't just for math (numeric values), they also work with string values, and, if an expression contains multiple values or variables, the combining of values will result in either concatenation (for the + operator) or reduction (for the - operator). Automatic data conversion isn't involved if all components in an expression are already what is intended (numeric or string), be the intention to work with numeric values (as for math, e.g., var=1+2) or string values (as for strings, e.g., var="1"+"2"). In the former expression, var=3 (numeric), but, in the latter, var="12" (string).

• String Values. As described in the Glossary (Chapter 10), concatenation is the combining of two or more string values, and reduction is the (non-math) subtraction of one string value from the value of another. *Reduction became available in WordPerfect 7.0,* and is not available in WordPerfect 6.1.

• Concatenation. If var1="1,000", var1="$"+var1, the result is that var1="$1,000" (string), or if var1="1,000" and var2=".05", var1=var1+var2, the result is that var1= "1,000.05" (string).

• Reduction. If var1="1,000.05" (string) and var2=",", in var1=var1-var2 the result is that var1=1000.05" (var1 no longer contains the comma). Without more, this technique removes the 1st occurrence of var2 (the comma) in var1, but not any subsequent occurrences. For that, you'd need to use a loop statement of some kind, such as, where var1="1,000,000,000.00" (or any other string variable), and var2=",":

```
x=StrPos(var1;var2)
If(x>0)
    Repeat
      x=StrPos(var1;var2)
      var1=var1-var2
    Until(x=0)
EndIf
```

• Numeric Values. If var1=1000 and var2=.05, var1=var1+var2 results in var1 equaling 1000.05 (numeric). With the same variables and values, var3=var1-var2 results in var3 equaling 999.5 (numeric).

• When Automatic Data Conversion Comes Into Play. What's said above used true string or numeric values in each example, so automatic data conversion was never involved. But, if one (or more) values are numeric and one (or more) values are string, then automatic data conversion comes into play.

Initially, PerfectScript assumes that the 1st value encountered in an expression is indicative of the value type (string or numeric) that you intend. For example, if var1=1 (numeric):

> var3=var1+var2 assumes that a numeric computation is desired, since var1 is numeric. var2 will automatically convert to a numeric value, if that is possible. So, if var2=".05", var1=var1+var2 results in var1 having a value of 1.05 (numeric) since var2 can convert to a numeric value via automatic data conversion.

But, if var2 cannot be converted to a numeric value, e.g., var2="ABC", even though the initial intention was interpreted to be that a numeric value should be attributed to var2, since that cannot happen (var2 cannot convert to a numeric value), a string value results instead in var3=var1+var2, and var3 is assigned the result of "1ABC".

• Forcing A Presumption. To force PerfectScript to presume that a numeric expression is intended, you can begin the expression with a numeric value which does not affect the computation. To force a string presumption, you could begin the expression with a pair of quotation marks, e.g., var3=""+var1+var2.

For math addition and subtraction, the expression could begin:

var3=0+var1+var2 or var3=0+var1-var2

For math multiplication or division, use a 1 multiplied by the 1st "real" value:

var3=1*var1*var2 or var3=1*var1/var2

• Shifting Presumptions. If you haven't forced a numeric presumption, above, unless all values in an expression are string values, automatic data conversion may nonetheless occur. The presumption may shift at least once, but perhaps many times in long expressions.

• Two Values. If only 2 values are involved in an expression, even if the 1st value is a string capable of converting to numeric and the 2nd value is numeric, it is then assumed that a numeric result was intended. So, where var1="1" (string) and var2=0.05 (numeric), in var1=var1+var2, the result returned to var1 is 1.05 (numeric). If the 1st value (string) cannot be converted to a numeric value via automatic data conversion, the 2nd variable will be treated as a string and the expression will return a string value, e.g., where var1="$" and var2=1000, in the expression var1=var1+var2, var1 will equal the string value "$1000", meaning, among other things, that the result of the formula/expression is a string, and not a numeric, value.

• More Than Two Values. If an expression contains more than two values, it is helpful to understand that:

• If The 1st Value Is Numeric. The remaining values or variables will be attempted to be interpreted as numeric. If they are, even if subsequent values are consecutive string values, those values will be attempted to be treated as numeric. So, if var1=1 and var2=".05 and var3=".001", in var1=var1+var2+var3, var1's value will be 1.051 (numeric).

• If The 1st Value Is String. If you haven't forced the numeric assumption (above), if the 1st value is a string, all consecutive string values thereafter will be treated as strings and concatenation will occur for all such values, unless and until a numeric value is encountered. If a numeric value is encountered, the string value which preceded it will be treated as numeric and a numeric result will obtain. So, if var1="100", var2="5", and var3=8, the formula var4=var1+var2+var3 returns a value of 1013 (numeric) to var4. In the formula sequence, var4 1st became the value of var1 ("100"); 2nd, concatenation occurred when added to var2 ("5"), causing var4 to become "1005"; 3rd, encountering the numeric var3 (8), "1005" was treated as numeric and var3 (8) was added to it, resulting in var4 equaling 1013 (numeric).

• Examples:

(1) w="12345" x="3" (both are string values which can convert to numeric)

```
z=w+x          //  z="123453" (string concatenation)
z=0+w+x        //  z=12348 (numeric - see Forcing A Presumption, above)
z=w-x          //  z="1245" (string reduction)
z=0+w-x        //  z=12342 (numeric - see Forcing A Presumption, above)
z=x*w or z=x/w  //  error; string values cannot be multiplied or divided
z=1*x*w        //  z=37035 (numeric) (numeric - see Forcing A Presumption, above)
z=1*x/w        //  z=4115 (numeric) (numeric - see Forcing A Presumption, above)
```

(2) w="12345" x=3 (w is a string value which can be converted to numeric, x is numeric)

```
z=w+x          //  z=12348 (numeric)
z=w-x          //  z=12342 (numeric)
z=w*x          //  z=37035 (numeric)
z=w/x          //  z= 0.000243013365735115
```

(3) w="1" x="9" y=5 (w and x are string values which can be converted to numeric; y is a numeric value)

```
z=w+x+y     //  z=24 (numeric)
z=x+w+y     //  z=96 (numeric)
z=w+y+x     //  z=15 (numeric)
z=y+w+x     //  z=15 (numeric)
z=y+w-x     //  z=14 (numeric)
z=x+w-y     //  z=86 (numeric)
z=x-w+y     //  z=14 (numeric)
```

(4) w="2" x="ABC" y=5 (w is a string value which can be converted to numeric; x is a string value which cannot; and y is a numeric value)

```
z=w+x+y     //  z="2ABC5" (string)
z=y+w+x     //  z="7ABC" (string)
z=w-y+x     //  z="-3ABC" (string)
z=w*y+x     //  z="10ABC" (string)
z=y/w+x     //  z="2.5ABC" (string)
```

• It's Your Choice. I'll close this discussion of automatic data conversion by saying that it is not difficult to include error trapping routines which test the value of any variable to make sure that it is the type of variable (string or numeric) that you intend and not rely upon automatic data conversion at all. That's my preference – I guess that it makes me "feel" better. But, do know that you have a choice in the matter, and that you may be very content to use the methods described above.

! ADDITION. The plus sign, "+", adds numeric values. So, var=1+3 results in var having a value of 4. Numeric values in the formula can be replaced by variables containing numeric values, so that var=var1+var2 would result in var having a value of 49, if var1=30 and var2=19. For this operator's string usage, see concatenation, above.

! SUBTRACTION. The minus sign, "-", subtracts one numeric value from another. So, var=3-1 results in var having a value of 2. Numeric values in the formula can be replaced by variables containing numeric values, so that var=var1-var2 would result in var having a value of 11, if var1=30 and var2=19. But, see Floating Cell Problem, below. For this operator's string usage, see reduction, above.

! MULTIPLICATION. The asterisk, "*", multiples one numeric value with another. So var=2*4 results in var having a value of 8. Numeric values in the formula can be replaced by variables containing numeric values, so that var=var1*var2 would result in var having a value of 570, if var1=30 and var2=19.

! DIVISION. The leaning-to-right slash, "/", divides one numeric value by another. So var=4/2 results in var having a value of 2. Numeric values in the formula can be replaced by variables containing numeric values, so that var=var1/var2 would result in var having a value of 1.5789473..., if var1=30 and var2=19.

! THE FLOATING CELL PROBLEM. Due to "floating cell" issues inherent in processor chips, SOMETIMES simple math routines, in my experience subtraction, particularly, using num3=num1-num2 DOES NOT produce an exactly correct result. See this thread at WordPerfect Universe: http://www.wpuniverse.com/vb/showthread.php?s= &threadid=1775. This is true in Excel, Quattro Pro, and it is true in WordPerfect macros. Try this yourself in your spreadsheet program: 692 - 685.08, and be sure your column width is overly wide. I think you'll find the result to be 6.91999999999996, but, even if slightly different, I doubt that your result will be 6.92, the obviously correct result. So, something is amiss and the problem can manifest itself in WordPerfect macros when using simple numeric subtraction, num3 = num1 - num2.To test this anomaly yourself, copy the following code to a macro, name it as you will, e.g., mathtest.wcm, and then run the macro.

```
Application (WordPerfect; "WordPerfect"; Default!; "EN")
GetNumber(n1;"Enter a real number... try 692";"Testing Macro Subtraction")
GetNumber(n2;"Enter a 2nd number ... try 685.08";"Testing Macro Subtraction")
n3=n1-n2
Messagebox(;"Computation Result of n1-n2";n3)
```

Unless I am badly mistaken, n3 will equal 6.91999999999996 using the above macro code. The correct result should be exactly 6.92. As far as I'm able to tell, this problem relates only to subtraction, not addition, multiplication or division. So, when doing subtraction, it makes sense to massage the numbers to be sure that results are exactly correct. An example is shown in Math.wcm, below.

J. Dan Broadhead offers this explanation:

This problem is not really Windows related, but is related to the computer processor. And it has to do with the fact that the computer internally uses base 2, and we (humans) use base 10. Most fractional base 10 numbers cannot be 100% precisely represented in base 2 as the exact same value. When the base 10 number is to be displayed, the internal base 2 number is converted back to base 10, and displayed, sometimes showing the imprecision problem. It is more apparent with certain values that others.

This is usually solved by rounding off the number after it is converted to base 10 for display. The problems lies in the "rounding off for display" operation.

This is not limited to just subtraction, but could potentially occur for any mathematical operations.

The internal numbers are not affected, they are still as precise as possible, but the display appears to be imprecise due to the rounding not working entirely correctly in all cases.

So the best advice is not to worry about it when performing the math operations, but to only worry about it when the values are going to be displayed someplace. Continual rounding after every math operation could eventually have an affect on the precision of the values.

! MASSAGING NUMBERS. This discussion assumes that "real" numeric values are used in the processes described herein ... or that automatic data conversion has occurred if possible, or that some other method has been used to insure and/or convert values to be numeric and not string.

! Integers (Whole Numbers). By definition, an integer (a whole number) has no "points" at all – it is 100, it is not 100.5 or 100.0. Results are not "rounded" to reach a whole number. If a non-numeric value is used with the command, macro operations will stop and an error message will be generated. If you want a math result to be an integer, use the Integer expression, like this (num1 and num2 can be numbers or variables containing numeric values):

```
num3 = Integer(num1+num2)     num3 = Integer(num1-num2)     num3 = Integer(num1/num2)
    Integer(50.6+50)              Integer(50.6-50)              Integer(50.6/50)
       num3=100                      num3=0                        num3=1

num3 = Integer(num1*num2)       num2 = Integer(num1)
    Integer(50.6*50)                Integer(50.6)
      num3=2530                       num=50
```

Note that the numeric range of the Integer command is -2,147,483,648 to 2,147,483,647, inclusive. If the integer is beyond that range, x=Integer(num) will produce zero (0) as the result.

! Working With Decimals. On the occasions that you need to do math which involve decimal values, it is essential that you have a working knowledge of NumStr, Strnum, Substr, StrLen, Concatenation, and Reduction. Click the links just shown for the Glossary (Chapter 10) descriptions, and also see StrTransform, StrTrim and StrToChars in the Glossary for other useful tools. Links in the next paragraph are to links within this chapter. Briefly stated, these commands or procedures to this:

NumStr converts numeric values to strings and only works with numeric values. StrNum only works with string values and converts them to numeric. Concatenation joins two or more strings together; and Reduction reduces the content of a string by the content of another. These work with either numeric or string values: SubStr extracts part of the value; StrPos determines the position of particular character(s) in a value; StrLen determines the character length (including spaces) of a value. Each of these commands and techniques is valuable in working with decimal (including percentage) values.

Practical examples follow these preliminary notes. First, here's a bit more detail.

NumStr() converts a numeric value to a string value. If the value attempted to be converted is not numeric, error occurs and the macro will stop. NumStr has 2 realistic parameters but only 1 parameter is required – the numeric value to convert, which can be a raw numeric value or can be a variable containing one. If var=0.005, both the following produce the same result: x=NumStr(0.005) x=NumStr(var). The optional parameter deals with decimal values ... those following a decimal, if any. The parameter can be 0 to 16. Rounding should occur for points beyond than the parameter value, breaking a 5 (up). So, NumStr(var;0) rounds the value to what amounts to a whole number string value, NumStr(var;2) rounds the value to a string value with 2 decimal points, if in fact there are any decimals to round. It doesn't "add" decimal strings (e.g., ".00") if none are present in the value being affected.

> *Windows 2000 and XP users should note:* Apparently, changes made beginning with Windows 2000 sometimes cause NumStr to not round correctly, e.g., if var=12345.555, x=NumStr(var;2) should result in x having a value of 12345.56, but, instead, the result is 12345.55, even though the correct result obtains in earlier Windows versions. See the following link at WordPerfect Universe if you want to know more: www.wpuniverse.com/vb/showthread.php?s=&threadid=6961. So, you may want to consider using alternative means than NumStr in Windows2000 and higher. A correct result can always be obtained by breaking the whole and decimal portions of string values and then recombine them, as is shown in examples in this chapter and in Chapter 9, particularly in Math.wcm.

StrNum() is the opposite of NumStr() – it converts string values to numeric values. If the 1st string character is not a numeric value, error results and the macro will stop. The single parameter is the string to convert. The parameter can be a literal string, or it can be a variable containing a string. So, if var="50.00", var=StrNum("50.00") produces the same result as var=StrNum(var), the numeric value of 50.0. If a decimal is not present in the string, a decimal will not be presented in the converted numeric value. So, if var="50", then the converted numeric value would have no decimals but would be the numeric value 50. StrNum can only work with numeric values – characters, including commas, are not numeric values. The only valid characters are +-1234567890 and "period" (decimal). If a non-numeric value is encountered in the string, it and everything following will be disregarded when using StrNum, rather like Reduction, above, by analogy. So, if var="1234g", x=StrNum(var) will return a numeric value of 1234 to x.

StrPos() is used to determine the position of a particular character (or characters) in a string or numeric value. In working with decimals, the character will typically be a decimal, a "period". Two parameters are used: the string, which can be a variable, and the character to locate. The parameters are separated by a semicolon and the searched for character is between a pair of regular (not curly) quotation marks. If the character is not present, a numeric value of zero is returned, but, otherwise, the exact location of the 1st matching character in the string is returned. So, if var="500.25", x=StrPos(var;"."), the value of x is 4, the 4th character. If var="500", x=StrPos(var;"."), the value of x is 0 (zero) since the decimal is not present.

SubStr() is used to extract "substrings" from numeric or string values. It has 3 parameters: the value (commonly a variable); the numeric beginning point of the extraction within the value; and the numeric ending extraction point within the value. The beginning and ending points can both be variables, or can be formulas based upon variables, such as by using StrLen(), described below, and/or by using StrPos() described above. The variable "type" (numeric or string) is unaffected by the command and the return value type will be the same as the original value type.

Here are 2 examples, where var="500.25" (string value), *or* where var=500.25 (numeric value):

> x=SubStr(var;1;3) returns a value of "500" (or 500 if var is numeric) to variable x
> x=SubStr(var;4;2) and x=SubStr(var;4;Strlen(var)) returns a value of "25" (or 25) to variable x

Let's throw in the StrPos() command, using the same var value ("500.25" or 500.25):

> y=StrPos(var;".") x=SubStr(var;1;y-1) returns a value of "500" (or 500) to variable x
> z=Substr(var;x+1;StrLen(var)) returns a value of "25" (or 25) to variable z

So, now we have the elements of the original value broken into 2 parts, neither of which include the decimal, making it possible to perform operations on these 2 components of the original variable, whether numeric or text. The length of each element can be determined using StrLen(), further testing can occur as for the existence of leading zeros, or whatever. The "whatever" will be largely shown in the remainder of this chapter.

Please don't get the idea that above shows the only way to accomplish the same thing. I'd be surprised if there aren't better ways. Here's another method that accomplishes essentially the same thing, where var="500.25" or var=500.25:

```
y=strpos(var;".")
x=Integer(var)
z=substr(var;y+1; Strlen(var))
```

StrLen() determines the number of characters, including spaces, in string or numeric values. If var="0.05" (string) or var=0.05 (numeric), then x=StrLen(var) results in x having a numeric value of 4: the 1st zero, the decimal, and the 2nd zero and the 5. StrLen can be used in combination with other commands, such as in the example immediately above.

! SPECIFIC TASKS. Very often the examples which follow are taken from Math.wcm in Chapter 9. The illustrations may be more tangible if you have an idea of what the main dialog in Math.wcm looks like. In Chapter 8, those illustrations here: Image showing control names; and clean images.

! Getting Rid of Commas. Commas aren't numeric values. So, to work with string values which appear to be numeric values (i.e., they include commas), here's the way I originally did that in Math.wcm, when I wasn't aware of the StrTransform command. This routine assumes that var is a string value, which may contain commas, and that Label(vComma) is called from elsewhere in the macro:

```
Label(vComma) // this strips commas in strings which, if used with NumStr, produces
    erroneous values
x=strpos(var;",")
If(x>0)
    While (x>0)
        y=substr(var;1;x-1)+substr(var;x+1;strlen(var)) var=y x=strpos(var;",")
    EndWhile
EndIf
Return
```

Explanation: x=strpos(var;",") determines the location of the 1st comma in var, if a comma is present. If not, x will equal zero (0). If so, x will be greater than zero. The While - EndWhile routine strips away each comma using SubStr and rejoins the substrings until no commas are present. So, if the initial value of var is the string value "1,222,333,444.50", the resulting value will be "1222333444.50". With this routine, var=StrNum(var) results in var equaling the numeric value of 1222333444.5. Without this routine, var=StrNum(var) results in var having the numeric value of 1 (since commas and what follows them are disregarded when using StrNum, described above.

As you can see, the above process relies upon concatenation. But, a simpler (but not the simplest – see below) is to use reduction, as follows:

```
Label(vComma) // this strips commas in strings which, if used with NumStr, produces
    erroneous values
x=strpos(var;",")
If(x>0)
    Repeat
        x=strpos(var;",")
        var=var-","
    Until (x=0)
EndIf
Return
```

Explanation: similar to the concatenation example, except that commas are removed from variable var using reduction until no commas are left.

But, *the simplest way* to accomplish the same thing is to use StrTransform, as is done in the revised Math.wcm, like this:

```
var=StrTransform(var;",";"")
```

All commas are replaced by "nothing" in one simple phrase of code! Why didn't I use the above in the first (or second) place? I wasn't aware of the StrTransform command until recently!

! Converting a Numeric Value to a String to "Look" Right in a Document.

! Adding Text, Generally. If, after working with numeric values in a macro, numeric values will need to be stuck into a "real" document you are working on (or, into a dialog the user will see), then it may be desirable to modify the numeric values so that they "look right". If working with regular numbers, you will likely want commas inserted where they belong, and, if you want decimal values shown, you may only want to show 2 decimal values. If you want money values, you will want some "money" sign, e.g., a $ (dollar), £ (pound), or ¡ (Euro) sign, of if you want a percentage value displayed, a % (percent), sign added to the value before it is written into a document.

To add a character, simply do so: xvar="$"+var or xvar=var+"%"

Note that I've not changed the value of variable "var" in either of the above, just in case I need to use such values in later computations. Instead, I've used "xvar" as the variable which can be used for display purposes in a dialog or when writing the variable to text. Of course, do what makes sense in the context of your macro.

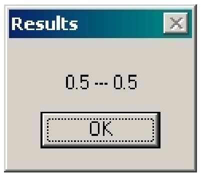
! Adding commas. Chances are that you won't want your document to display a numeric value such as 1234565000. Instead, you'd want it to be 1,234,565,000. While other ways are possible, here's one that works for me – Label(NewNum) is extracted here from Math.wcm, below. The macro code below assumes that variable "var" contains a numeric value, and that LabelNewNum) is called from elsewhere in the macro:

```
    Label(NewNum) // this routine massages numeric values into new string values
If(var=0)
  Return
EndIf
x=StrPos(var;"-")
If(x=1)
  var=Substr(var;2;strlen(var)) NegNum=1
EndIf
x=StrPos(var;".")
Switch(x)
  Caseof 0:  tNum1=var Call(mainNum) tNum2=""
  Default:  tNum1=Substr(var;1;x-1) Call(mainNum) tnum2=substr(var;x;strlen(var))
     x=strlen(tnum2)-1
     If(x<varD and vFinal=1)
        Repeat
           tnum2=tnum2+"0" x=strlen(tnum2)-1
        Until(x=varD)
     EndIf
  EndSwitch
var=tNum1+tNum2
If(NegNum=1)
  var="-"+var
EndIf
Discard(NegNum;tNum1;tNum2)
Return
  Label(mainNum) // this routine creates commas in the main number from Label(NewNum)
If(StrLen(tNum1)<=3)
  Return
EndIf
z=Integer(StrLen(tNum1)/3)
```

```
newvar=""
Repeat
  y=SubStr(tNum1;StrLen(tNum1)-2;3)
  tNum1=SubStr(tNum1;1;StrLen(tNum1)-3)
  If(StrLen(tNum1)>3)
     newvar=","+y+newvar Else newvar=tNum1+","+y+newvar
  EndIf
  If(StrLen(tNum1)>3)
     z=Integer(StrLen(tNum1)/3) Else z=0
  EndIf
Until(z=0)
tNum1=newvar
Return
```

! Percentages. You've used your macro to produce a percentage, or, at least, which will become a percentage, by division. Your macro code might be:

```
Application (WordPerfect; "WordPerfect"; Default!; "EN")
var1=50 var2=100 var=var1/var2 var3=NumStr(var)
Messagebox(;"Results";var+" --- "+var3)
```



```
0.5 --- 0.5
```

If multiplication, division, addition or subtraction produces a result less than 1, a leading "zero" and decimal will always be part of the numeric value returned, and so will the result of using NumStr(var) to convert the value to a string, both of which values (numeric "var" is at the left, string "var3" is at the right).

But, in your document, you want the 0.5 value to be 50% or 50.0% or 50.00%. Various means are available to accomplish that, but here's one way:

Massage the string value to determine its parts and then put them all back together again. Here's the previous code in Label(MakePer) in the Math.wcm macro below – the label is called from some other location in the macro and notice that it also calls Label(NewNum), shown above. Note that the variable being tested and worked over is a variable named "var" which contains a string value:

```
  Label(MakePer) // this routine creates percentage string values; revised 6/4/2004 for scientific notation values
e=0
var=StrTransform(var;",";"")
var=strnum(var)
var=var*100 // a % value is always 100 times the non-percentage value
x=strpos(var;"e") // scientific notation routine
If(x>0)
  e=1
  y=substr(var;x+2;3) y=strnum(y) y=y+1
  var=substr(var;1;x-1)
  var=var-"."
  varq=strlen(var)
  If(varq<y)
     Repeat
         var=var+"0"
         Until (strlen(var)=y)
  EndIf
  x=strpos(var;".")
  If(x=0)
     var=var-"."
  EndIf
EndIf
If(e=0)
  var=numstr(var) // want to determine if that following the decimal is greater than 0, but not if scientific notation
EndIf
```

```
x=strpos(var;".") testn=substr(var;x+1;strlen(var)) testn=strnum(testn) // testn is rem w/o the "."
If(testn=0)
  var=substr(var;1;x-1)  // if the remainder is 0, drops the ".0" from var
EndIf
If(e=0) // not if scientific notation
  var=strnum(var)
EndIf
Call(NewNum)
var=var+"%"
If(negNum=1)
  var="-"+var
EndIf
Return
```

! ERROR TRAPPING ROUTINES. Unless you build in routines to trap errors, when commands dealing with strings are used as though they were numbers, and vice versa, macro error may result causing the macro to stop or unintended results may occur. The 1st step is to insure that a user's intended "numbers" are numeric. An "O" (oh) is not a zero (0), as you know. A comma is not a numeric entry, either. So, the 1st thing to do is to test the user's data (such as from an edit box in a dialog) to insure the characters are "legal" if they are converted to a numeric value (as by using StrNum or in performing mathematical computations). The following are Label(chkNum) and Function IsNumberString(InString) contained in Math.wcm, below. It also tests for a "minus" sign in the 1st character position in the variable. It assumes that variable "var" contains a string value, and that Label(chkNum) is called from elsewhere in the macro:

```
  Label(chkNum) // this routine determines if all characters are valid for numeric purposes; revised 6/4/2004
x=StrToChars(var;Keep!;".") x=StrLen(x) //this routine insures that only one decimal is in the string
If(x>1)
  q=1
  Return
EndIf
x=substr(var;1;1)
If(x="-")
  NegNum=1 var=substr(var;2;strlen(var)) Else NegNum=0
EndIf
Ret=IsNumberString(var)
If(Ret)
  Else q=1
EndIf
Return // if the variable contains non-allowed characters, q=1 says error is present

Function IsNumberString (InString)
RefString="1234567890.,"
ForNext (Count;1;StrLen(InString);1)
  If (StrPos(RefString;SubStr(InString;Count;1))=0)
     Return (False)
  EndIf
EndFor
Return (True)
EndFunc
```

The 1st routine in Label(chkNum) makes sure that only one decimal is present in the string by using StrToChars. Since a decimal is "legal" in Function IsNumberString(InString), we need to be sure that not more than 1 decimal is present. So, variable "x" initially becomes the string length of all characters other than "." If the string length is 1 or 0, fine, but if more than 1, that's not a "legal" number (in string form at the moment).

The 2nd routine determines whether var is a "negative" number by extracting the 1st character of the string and assigning the 1st character's value to variable "x". If x = a minus sign, variable "NegNum" is assigned a value of 1, and variable "var" is reassigned the portion of the string following the minus sign. Otherwise, variable "NegNum" is assigned a value of 0. Sometimes you may care, or not, whether a variable is "negative", and, if so, the "NegNum" value may be used following the point of the Call(chkNum) depending on whether you care if the value is negative.

Following that routine, the remainder of the code determines whether variable "var" contains any characters other than 1 2 3 4 5 6 7 8 9 0 or a period or a comma. This is done by calling the Function IsNumberString by use of the code, Ret=IsNumberString(var). If the variable contains any other characters other than those just mentioned, variable "Q" is assigned a value of 1. Note that, here, a comma has been treated as "legal" even though it is not, strictly speaking, for numeric purposes. A subsequent routine will eventually need to address the presence of commas. But, not now.

After this operation, macro flow returns to the point following the Call(chkNum). For example, in Label(getV), the call "shorthand" chkNum is used in place of Call(chkNum):

```
Label(getV) var=RegionGetWindowText("vMath.B2") If(var="") var="0" EndIf

chkNum If(exists(q)) RegionSetWindowText("vMath.S6";"2nd value, "+var+", is not legal")
RegionSetFocus ("vMath.B2") RegionSetEditSelection("vMath.B2") Return EndIf

If(NegNum=1) q=1 RegionSetWindowText("vMath.S6";"2nd value, "+var+", is not legal. Use
Subtract in regular date list instead.") Discard(NegNum) RegionSetFocus ("vMath.B2")
RegionSetEditSelection("vMath.B2") setResult Return EndIf

var=StrNum(var) If(vMethod="D") tempd2=var Else tempL2=var EndIf SecondN=var Return
```

Immediately following the chkNum call, if variable "Q" exists, a message is sent to the "vMath" dialog (particularly, to the "vMath.S6" control that the value entered in the dialog is not "legal"), and the macro does not continue beyond that point but instead the Return command ends the call. If variable "Q" does not exist, the macro proceeds.

Since, in the above example, since I DO care whether a negative entry was made in the dialog, variable "Q" is assigned a value of 1 and a message is again sent to the same "vMath.S6 " control, and, once again, the macro does not proceed past the point of the Return command which ends the call.

Otherwise, all tests being passed, the last line of the above code converts the string value to a numeric value. Some other stuff is done, too, not being relevant to this discussion.

Other error-trapping routines are certainly possible, but the above is one which I use again and again, and it works every time.

Top of Chapter 6 Chapter 7

NOTES

NOTES