Peaksimple User Calculations

A UserCalc formula defines a numeric value, which is then displayed as any other numeric field available in Results. This can be made up of references to other fixed fields, specified by name with square brackets around them, for instance:

```
[Area]
[External]
```

Or can be made up of functions, which take up to 3 parameters, in round brackets with commas in between, and return a value. For instance, to calculate the square root of the peak area, the formula would use the SQRT function, which takes one parameter:

```
SQRT([Area])
```

Names of fields and functions are case-insensitive, so the following are equivalent to the above:

```
Sqrt([area])
sqrt([AREA])
```

Functions with two parameters can also be expressed as "infix operators", which means they are special symbols of one or more characters with their parameters provided to the left and right on either side. A familiar example of this is the addition symbol "+", which in this example can be used to return the sum of the [Area] and [External] value for a peak:

```
[Area]+[External]
```

Infix operators are available for other common mathematical operations - subtraction, multiplication, division, and power:

```
[Area]-[External]
[Area]*[External]
[Area]/[External]
[Area]^[External]
```

Spaces can be added anywhere in a formula as long as they don't interrupt the characters that make up the name of a field or function. So the following are equivalent to examples above:

```
[ Area ] + [ External]
Sqrt ( [area] )
```

Numerical constants can be included wherever a numeric value is expected. For example to multiply the area of a peak by 3 and a half would be:

```
[Area] * 3.5
```

Functions, operators, fields and constants can be mixed to build up more complex mathematical calculations. For instance, to take the square root of half the area and add it to the external would be:

```
Sqrt([Area]/2) + [External]
```

When infix operators are used alongside each other, standard mathematical rules determine which take priority. For instance:

```
2 * 3 + 4
```

Returns 10, not 14, as the multiplication is done first. If the addition should come first, then this can be forced by using round brackets, like this:

```
2 * (3 + 4)
```

Which would return 14, as the addition is done first. Comparison infix operators will compare their two parameters and return either 1 or 0, depending on whether the comparison is true or false. For instance the greater than symbol ">" is used in this example:

```
[Area]>[External]
```

Which will return 1 if area is greater than external, otherwise 0. Hence 1 (actually anything not 0) is taken to mean "true", and 0 is taken to mean "false". Similar comparison operators are available for less-than, greater-than-or-equal-to, less-than-or-equal-to, equal-to, and not--

```
[Area]>=[External]
[Area]<=[External]
[Area]=[External]
[Area]<>[External]
```

These comparison operators can be used together with "IF" function, which will return one of two different values depending on whether or not the expression given in the first parameter is 1 or 0. The second parameter determines what is returned if it's 1, and the third parameter determines what is returned if it's 0. For instance, the following will return 33 if area is less than external, otherwise 44:

```
If ( [Area] < [External] , 33 , 44 )
```

Further logical operators can be used to combine together the results of the comparison operators. For instance the following uses the "AND" operator to only return 1, if area is greater than 100 and also external is greater than 10:

```
[Area]>100 and [External]>10
```

The & symbol can be used instead of "and", so the following is equivalent:

```
[Area]>100 & [External]>10
```

An "or" logical operator is also provided, so the following will return 1 (true) if either area is greater than 100 or external is greater than 10:

```
[Area]>100 or [External]>10
```

Text values can also be included in a formula, either by reference to text fields, or by including fixed string constants in double-quotes. However the final result returned by the entire formula must be of numeric type. So the following is not allowed on its own:

```
[Name]
```

But text can still be used in conjunction with a function which expects a text parameter, and returns a numeric. An example is the "LENGTH" function, which takes one text parameter, and returns the length of that text. So the following will return 5:

```
Length("Hello")
```

And the following will return the number of characters in the name of a peak:

```
Length( [Name] )
```

The "Contains" function compares two text parameters and returns true if the second is found somewhere within the first (ignoring case). This example returns 100 if the name of a peak includes the word "Carbon", otherwise zero:

```
IF ( Contains([Name],"carbon") , 100, 0 )
```

The function "IFS" is a text (string) version of the "IF" function, and expects it's second and third parameter to be text. It also returns a text type. The following example returns 1 if the Client field contains the text "EPA", unless the Client field is empty, in which case it will check the Lab name field instead:

```
Contains ( IFS( Length([Client])>0 , [Client] , [Lab name] ) , "carbon" )
```

Here is a full list of available functions:

```
PI      <no parameters>         numeric type, returns constant PI (3.141592...)
Sin ( num1 )                    numeric type, returns sine of num1, in radians
Cos ( num1 )                    numeric type, returns cosine of num1, in radians
Tan ( num1 )                    numeric type, returns tangent of num1, in radians
Min ( num1 , num2 )             numeric type, returns the smallest of num1 and num2
Max ( num1 , num2 )             numeric type, returns the largest of num1 and num2
Sqrt ( num1 )                   numeric type, returns the positive square root of num1, or zero if num1 is
negative
Logn ( num1 )                   numeric type, returns the natural (base e) logarithm of num1, or zero if
num1 is negative
Log ( num1 )                    numeric type, returns the base-10 logarithm of num1, or zero if num1 is
negative
Exp ( num1 )                    numeric type, returns e to the power (i.e. natural antilog) of num1
```

If (num1 , num2 , num3)	numeric type, returns num2 if num1 is not zero, or num3 if num1 is zero
Ifs (num1 , text2, text3)	text type, returns text2 if num1 is not zero, or text3 if num1 is zero
Length (text1)	numeric type, returns the length of text1
Contains (text1, text2)	numeric type, returns 1 if text1 contains text2 (ignoring case), otherwise 0