Table of Contents

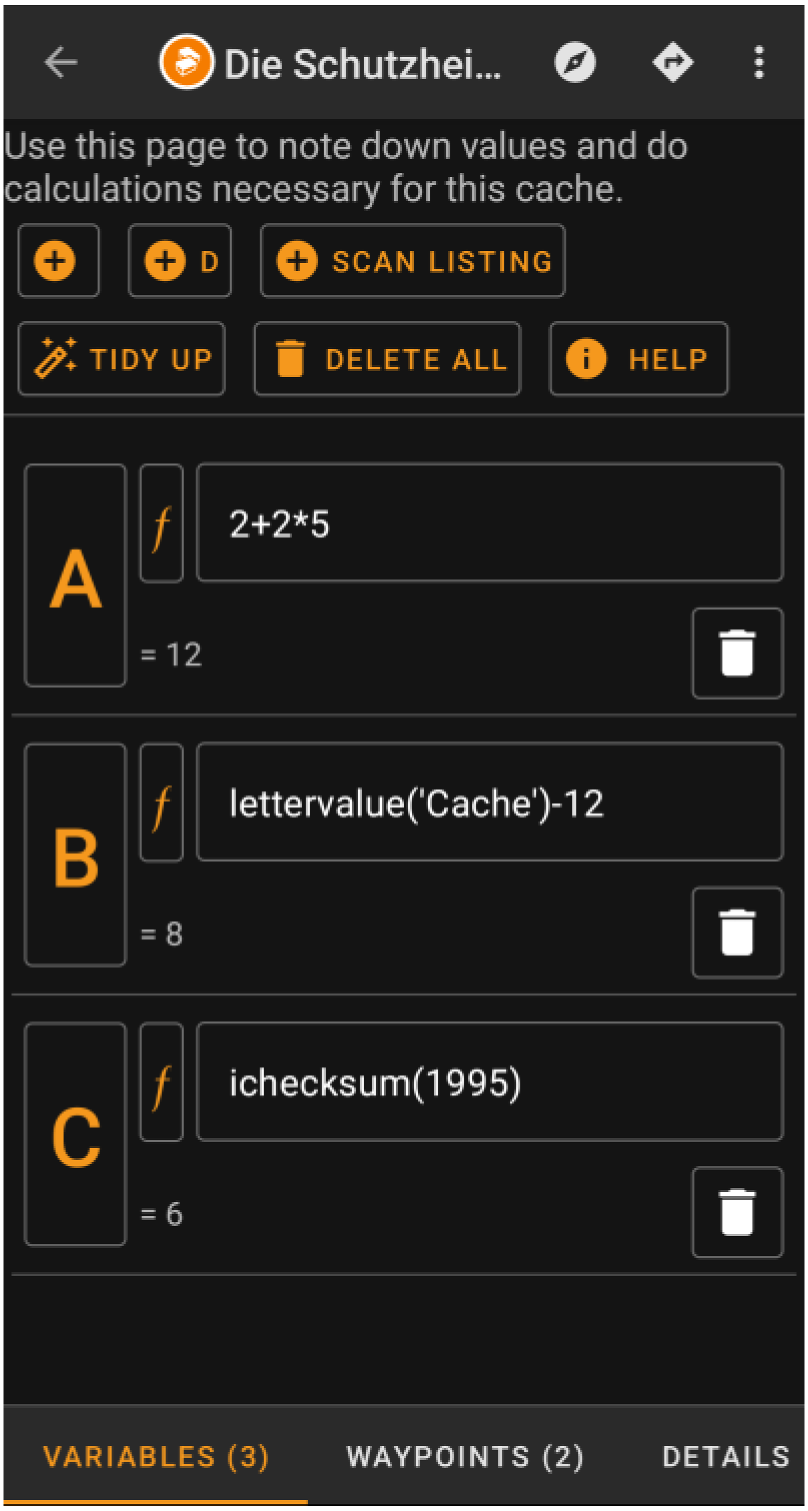
Fix Me! This will be used to describe the new tab on cache details used for cache variables.

Fix Me! We could add a chapter with some practical How-To examples (e.g. typical lettervalue case)

Cache Variables

Introduction

For each geocache detail view c:geo offers a tab called “Variables” to note down and perform calculations with formulas and variables you need for this cache.



Use this page to note down values and do calculations necessary for this cache.

A: 2+2*5 = 12

B: lettervalue('Cache')-12 = 8

C: ichecksum(1995) = 6

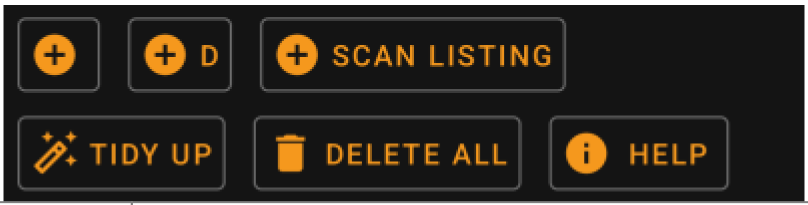
This can come in handy if you e.g. perform a multi cache which requires to collect some values while out in the field and perform mathematical calculations with it to get to the next stage or the final.

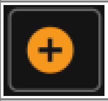
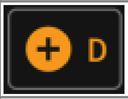
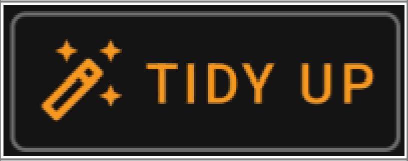
You can use this variable tab as a standalone helper to do some calculations or you can also reuse any variable defined here for generating a new calculated waypoint for this cache.

The following sections on this page describe the content and functionality of the variable tab.

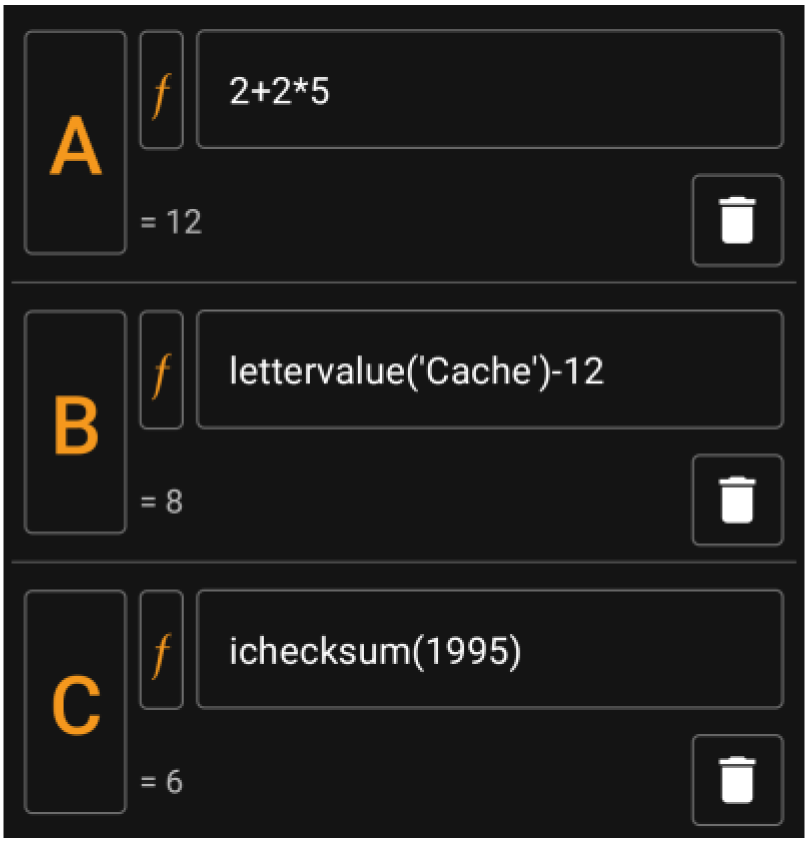
Control section

On the top of the variable tab you will see a set of buttons offering functions to populate the variable section below:



Button	Description
	Manually add a variable to the view, by providing its name.
	Automatically add next free variable in alphabetical sequence to the view.
	This function will scan the geocache description for potential formulas contained and offer them to be taken over into the variable tab. Each selected found formula will be added as content of a new variable.
	This will remove all variables without value or formula and can be used in case you accidentally created to many variables or have cleared some of them and no longer need them.
	This will delete all defined variables and their values.
	Opens this page in your browser

Variable section



This section allows you the enter the value or formula for the generated variables. You can perform the following actions here:

Button	Action
A	Click on the variable name to change it.
f	Click on the “function” button to prefill the value field with the desired supported function.
ichecksum(1995)	Fill in the value field manually with a value or some formula using the formula syntax.
🗑	Use the trash can icon to delete the variable.

The text below the value field will show a result preview. This can either be the concrete result of the formula or hints regarding syntax errors or missing values.

Formula syntax

The value field of each variable can hold different types of values and also other variables. It supports numerous mathematical operations as well as several (partly geocaching related) numerical and string related functions as described below.

Don't be afraid of the syntax. While it supports rather complex operations, it can also be used for simple and plain calculations as you are used to from any calculator. Some supported functions are probably for advanced users only.

The syntax will be explained in detail in the following sub-chapters. However as a quick overview about what is supported, you will find a list of examples here:

- `2*2+3` will evaluate to `7`
- `2*(2+3)` will evaluate to `10`
- `3*sin(90)` will evaluate to `3`

- `4+length('test')` will evaluate to `8`
- `rot13('abc')` will evaluate to `nop`
- `lettervalue('cache')` will evaluate to `20`
- `checksum(lettervalue('cache'))` will evaluate to `2`

- `A + A*2` with A=3 will evaluate to `9`
- `AA(A+1)` with A=3 will evaluate to `334`
- `$hello + 1` with variable `hello=24` will evaluate to `25`
- `$hello(A+1)` with A=3 and `hello=24` will evaluate to `244`
- `${hello}8A` with A=3 and `hello=24` will evaluate to `2483`

Value types

The formula syntax supports three types of values. Typing is loose, in general formula evaluation will try to fit given values as best as possible.

Type	Description	Literal syntax	Examples
Integer	Number without decimal places	Use digits	`1234`, `-3`
Decimal	Number with decimal places	Use digits with decimal point or comma	`3.14`, `-3.14`, `3,14`
String	Text	Surround value with ' or “ To use the `'` or ” their selves, type Fix Me! or “”	`'test'`, “`test`” “`He said ””“yes”“!`”

Fix Me! using monotext does not work for ' in this table as its also part of the monotext syntax in Dokuwiki. :(

Numeric operators

The following numeric operators are supported:

Operator	Function	Example
+	Addition	2+4 evaluates to 6
-	Subtraction (or negating a number)	`6-4` evaluates to 2 `-(5-2)` evaluates to `-3`
*	Multiplication	3*4 evaluates to 12
/	Division	12/3 evaluates to 4
%	Modulo	12%5 evaluates to 2
^	Potentiate	3^3 evaluates to 9

Operator	Function	Example
!	Factorize	`4!` evaluates to 24

Relational operators and conditions

Relational operators like < or == can be used to compare two values with each other. In general, such an operation will return the value `1` if the comparison yields true and the value `0` if it yields false.

For example, the expression `3 < 4` will compute to the value `1`.

Relational operators are especially used in the `if` function. This function evaluates its first parameter. If this parameter is true (means: has a value > 0 or is a non-empty string) then it returns its second parameter. Else, and if it has a third parameter, the third parameter is returned.

The `if` function accepts any number of parameters and interprets them in an “if-then-if-then-if-then-…-else” cascade.

This means, that if the function was given 5 parameters then: * If the first parameter is true, then the second is returned * Else if the third parameter is true, then the fourth parameter is returned * Else the fifth parameter is returned

For example `if(A==5;50;A==4;40;30)` will evaluate to `50` if `A=50`, to `40` if `A=4` and to `30` for any other value of A.

Operator	Meaning	Example
==	Checks for equality	2==2 evaluates to `1`(=true)
<>	Checks for inequality.	3<>2 evaluates to `1`(=true)
<	Is less than	3<4 evaluates to `1`(=true)
⇐	Is less or equal than	3⇐3 evaluates to `1`(=true)
>	Is greater than	3>4 evaluates to `0`(=false)
>=	Is greater or equal than	5>=5 evaluates to `1`(=true)

Functions

Functions all start with a letter, contain only letters and digits and have a directly attached parameter list in parenthesis. Multiple parameters are separated using `;`.

An example for a one-parameter function call is `sin(90)`. An example for a two-parameter function call is `rot('test'; 13)`.

The following functions are defined:

Function	Synonyms	Description	Parameter1	Parameter 2	Example
`sqrt`	-	calculates square root of given parameter	numeric parameter	-	`sqrt(9)` evaluates to 3
`sin/cos/tan`	-	calculates sinus/cosinus/tangens of given parameter	numeric parameter in degree(!)	-	`sin(90)` evaluates to 1
`abs`	-	calculates absolute value	numeric parameter	-	`abs(-34)` evaluates to 34
`round`	-	rounds decimal values mathematically	value to round	optional: number of places to round to	`round(4.65)` evaluates to 5, `round(4.65; 1)` evaluates to `4.7`

Function	**Synonyms**	**Description**	**Parameter1**	**Parameter 2**	**Example**
`if`	-	evaluates conditions and returns conditional values	list of if-then-else-values. See previous section for details	-	`if(3<4;5;6)` evaluates to 5
`checksum`	`cs`	calculates checksum of given numeric value. Calculates lettervalue if given paramter is of type text	positive integer or text	-	`checksum(345)` evaluates to 12
`ichecksum`	`ics`	calculates iterative checksum of given numeric value. Starts from lettervalue if given parameter is of type text	positive integer or text	-	`ichecksum(345)` evaluates to 3
`lettervalue`	`lv`, `wordvalue`, `wv`	calculates lettervalue of given string value	string	-	`lettervalue('test')` evaluates to 64
`rot`	-	calculates rotated string of given string value	string	count to rotate by	`rot('abc'; 1)` evaluates to `'bcd'`
`rot13`	-	calculates rotated-13 of given string value	string	-	`rot13('abc')` evaluates to `'nop'`
`roman`	-	scans a given string value as a roman number and returns its decimal value	string	-	`roman('VI')` evaluates to 6.
`vanity`	`vanitycode`, `vc`	returns the vanity code of a string	string	-	`vanity('cgeo')` evaluates to 2436.

Variables

Variables are used in a formula as placeholders for values. When a formula containing a variable is evaluated, it needs to be passed a value for each of the contained variables in order to be correctly evaluated.

Variable names are case sensitive and have to start with an alphanumeric char. Remaining chars can be alphanumeric or digits. Examples for legal variable names are: `Test`, `T1`, `t`, `Tt123`. Examples for non-legal variable names are: `1a`, `2`

One-letter-variables can just be typed into the formula and will be evaluated along. For example, the formula `A + 2` is valid. If A has the value 5, the formula will evaluate to 7.

If multiple chars are concatenated within a formula, they will be interpreted as individual one-letter-variables. For example, the formula `AA + 2` will be interpreted as variable A concatenated two times and adding 2 afterwards. If A=4, this formula will evaluate to `44 + 2 = 46`. See following section for more details wrt concatenation.

Variable names longer than one char can be declared in Unix-Bash-Style by prepending their name with $. For example, a variable named `Test` is can be referenced using `$Test`. The formulua `$Test + 2` is valid. If value for variable `Test` is 4 the formula will evaluate to 6.

In situations where variable name conflicts with following alphas/chars, the variable name can be enclosed in `{}` to differentiate it from following text. For example, the following expression will concatenate the value of variable `Test` with the value of variable A: `${Test}A`

Some more complex examples:

- The formula `A + $A * $Test - t` uses three variables named A, `Test` and `t`. The variable A is used in two places. Assuming A=2, `Test`=3 and `t`=1, the formula would evaluate to 7.
- The formula `AA + b + $A1` uses three variables A, b and A1. Assuming A=2, b=3 and A1=4, the formula would evaluate to 29 (= 22 + 3 + 4)
- The formula `AB(B+1)` uses two variables A and B. Assuming A=2 and B=3, the formula would evaluate to 234
- The formula `$AB(B)(B+1)` uses two variables AB and B. Assuming AB=2 and B=5, the formula would evaluate to 256
- Using `{}` syntax, the previous example could also be written like this: `${AB}B(B+1)`

Concatenations

If multiple expressions are concatenated directly after another with no separating operator, values are concatenated to a consecutive expression. This expression evaluates to a number if it forms a valid numeric expression, otherwise it evaluates to a text value.

Expressions, which can be concatenated, include e.g. integer digits, variables, expressions in parenthesis and the Overflow character (see next subsection).

For example, the formula `AA(A+4)55$Test(3)` contains two variables A and `Test`. Assuming A=9 and `Test=70`, it would evaluate to `991355703`.

Overflow character

In concatenated expressions, the character _ can be used as an overflow sign. It is a placeholder for possible spillovers if numeric variables evaluate to a value with more than one digit, otherwise it is filled with 0.

An example should make the usage clear:

- The Formula 1A with A=2 will evaluate to 12
- The Formula 1_A with A=2 will evaluate to 102
- The Formula 1_A with A=23 will evaluate to 123
- The Formula 1A with A=23 will evaluate to 1023 * The Formula 1A with A=234 will evaluate to 1234

Fix Me! Double underscore is interpreted as underline by Dokuwiki

Range expressions

You can specify ranges in formulas using `[]`. This is needed when variables are used in a context where a range of values should be iterated over. A prominent example is the “Generate Waypoints” function.

An example for a range expression is `[0-9]`. This specifies a range with 10 values (the integer values 0 to 9).

You may specify consecutive values using `,` as a delimiter. You may exclude values or value ranges by prepending a ^ to it. Ranges are parsed from left-to-right, giving an order to the elements in the range. For example the following are valid range specifications:

- `[0-2, 4]` evaluates to a range containing `0`, 1, 2 and 4.
- `[0-3, ^1-2]` evaluates to a range containing `0` and 3.

- `[0-3, ^1-2, 5]` evaluates to a range containing `0`, 3 and 5.

When a range is used in a context where only one value is allowed (this is the case in normal calculation), the first range value is used for calculation. For example, the expression `[8, 0-9]` will evalulate to `0` in a normal calculation context, while `[8, 0-9]` will evalulate to 8.

Ranges currently support only positive constant integer values. A range must always be evaluate to at least 1 value and a range may not evaluate to more than 20 values. For example the following ranges are invalid:

- `[]`: empty
- `[5, ^0-9]`: evaluates to empty
- `[0-1000]`: evaluates to more than 20 entries
- `[-5]`: negative int not allowed
- `[A]`: variables not allowed

A formula may include one or more range definitions mixed with normal other formula parts. For example the following formulas are valid:

- `3*[0-2]`: evaluates to values `0`, 3 and 6
- `A*[4, 7]`: for A=3 this evaluates to values 12 and 21
- `[1-2]*[3-4]`: evaluates to 3, 6, 4 and 8.