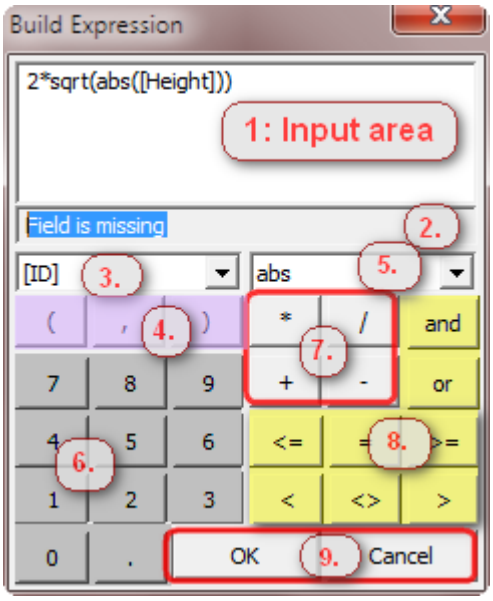
Expression editor manual

Basics

Expressions are similar to Excel functions: by using them you can perform mathematical or logical calculations, like checking a field's value and performing an action upon a certain condition. Expressions can be used for automatic data modification.

The expression builder is a built in tool to edit expressions in a simple way. To reach this module select one field, click its name, and set the Default value to Expression.



1. Input area
2. Real time validation
3. Field list
4. Function helper controls
5. Function list
6. Numeric inputs
7. Mathematical operators
8. Logical operators
9. Window buttons

The basic syntax is **function**(*param1*, *param2*, ..., *paramN*). The spaces between commas and values (paramN) are not necessary.

- The function names are CAsE-SenSitiVe, and have to be written with lower case: you have to use „func" instead of „Func" or „FUNC"!
- Param: if the param is string, you have to define it with quotation marks ("string").
- The length of expression depends on the length of the expression field length, maximum is 256 characters.

Basic operators:

Mathematical:

- **+** : add
- **-** : extract
- ***** : multiple
- **/** : divide

Logical:

- **=** : equal
- **<>** : not equal:
- **=<** : smaller or equal
- **<** : smaller
- **>** : bigger
- **>=** : bigger or equal
- **and** : and (A AND B statements are have to TRUE to gives back TRUE the whole expression)
- **or** : and (A OR B statements are have to TRUE to gives back TRUE the whole expression)

Character concatenation: „**+**”

To link a field, use this syntax: [FieldName] (or [FieldAlias] if setted). If the field uses a Code dictionary, the same order is required! (After that linking, please do not modify the general field type (i.e.: text to numeric or vice versa)!)

“@” field prefix

When you’re using a code dictionary for a field, you can decide if you want to use the codes in the fields or the name connected to a code.

[Fieldname] – the code stored in the field – if you used # in the code dictionary, it will be a number, in case of using $ it will be a string.

@[Fieldname] – the name connected to the code, for example you’ve got a “1 Oak” line in the cdt file and the field value is 1, you’ll get “Oak” when using @[Fieldname]

Other usage: when you put the @ prefix for a numeric value field, it will convert it to a string with the given display options (width and decimal places) and you can concatenate it to another text field or a string.

Real time evaluation

Messages in the evaluation line (run in real time):

- *Value of the expression* – if the expression is valid
- **Field is missing**: Valid field(s) is(are) cannot be found – mistyped field name(s)
- **Bad operator**: Not a valid syntax, because of operator
- **Bad operand**: Not a valid syntax, because of operandus
- **, is missing**: at least one param is missing
- **-1.#IND** – internal mathematical error (i.e: square root of negative numbers)

Functions by type

- Mathematical: **abs(), exp(), floor(), sqrt()**

- Trigonometrical: **acos(), asin(), atan(), cos(), sin(), tan()**

- Text manipulation:
  - Character: **asc(), chr(),**
  - Text: **find(), left(), len(), lower(), mid(), replace(), right(), upper()**

- Datum/Time related: **date(), days()**

- Logical: **if()**

Function list

abs - Absolute value

- *Syntax:* **abs**(*param*) – Absolute value of param (as number)
- *Samples:*
  - abs(-100) = 100
  - abs([Filed]) = return with positive numbers of the numeric Filed value

acos - Arcus cosinus

- *Syntax:* **acos**(*param*) – Arcus cosinus of degree in radian (param as number)
- *Samples:*
  - acos(-1/2) = 2.0943951023932 (rad)
  - acos(1) = 0 (rad)

asc - ASCII code of the letter

- *Syntax:* **asc**(*param*) – ASCII code of character (param)
- // if you define more than one character, we use the first)
- *Inverse function: chr()*
- *Samples:*
  - asc(”é”) = 233
  - asc(”@”) = 64

asin – Arcus sinus

- *Syntax:* **asin**(*param*) – Arcus sinus of degree in radian (param as number)
- *Samples:*
  - asin(-1/2) = -0.5235987755983 (rad)
  - asin(1) = 1.5707963267949 (rad)

atan – Arcus tangent

- *Syntax:* **atan**(*param*) – Arcus tangent of degree in radian (param as number)
- *Samples:*
  - atan(-1/2) = -0.46364760900081 (rad)
  - atan(1) = 0.78539816339745 (rad)

chr – Letter by ASCII code

- *Syntax:* **chr**(*param*) – Character by ASCII code (param as number)
- *Inverse function: asc()*
- *Samples:*
  - chr(64) = @
  - chr(100) = d

cos – Cosinus

- *Syntax:* **cos**(*param*) – Cosinus of degree in radian (param as number)
- *Samples:*
  - cos(-1/2) = 0.87758256189037 (rad)
  - cos(0) = 1 (rad)

date – Date transformation: from seconds to date

- *Syntax:* **date**(*param*) – date from AD 0/03/Jan. in seconds (param as number)
- // 1 year=365.2425sec
- *Inverse function: days()*
- *Samples:*
  - date(1956*365.2425*24*3600-2*24*3600) = 1956.01.01 7:55:11
  - date(0) = AD 03/Jan 0000 0:00:00
  - date(365.2425*2014*86400+90*86400) = 2014.04.02 9:28:48

days – Date transformation: from seconds to date

- *Syntax:* **date**(*param*) – date without delimitator chars (param as number)
- // 1 year=365.2425sec, seconds from AD 0/03/Jan.
- *Inverse function: date()*
- *Samples:*
  - days (1956010175511) = 61725196800 (sec)
  - days(20140101000000) = 63555580800 (sec)

exp – Calculate exponential logarithm

- *Syntax:* **exp**(*param*) – calculate exp. logarithm of the number (param as number)
- *Samples:*
  - exp (0.5) = 1.6487212707001
  - exp(-1) =0.36787944117144

find – Search exact sting

- *Syntax:* **find**(*what,where*) – Possible to finding "what" string in "where", TRUE(1) or FALSE(0)
- // not case sensitive, eg: „London", „LONDON" and „london" are equal
- *Samples:*
  - find("London",[address]) = Gives back with 1, if address field contains „London" string
  - find("1",values) = 1 (true), if „values" field contains „1", but False if contains „2" or „11"

floor – Gives back the integer part of number

- *Syntax:* **floor**(*param*) – calculate the integer part of the given number (param as number)
- // not case sensitive, eg: „London", „LONDON" and „london" are equal
- *Samples:*
  - floor(2014.023) = 2014
  - floor([area]) = If area is 201.24, gives back 201

if – Conditional statement

- *Syntax:* **if**(*condition, true, false*) – the result depends on the logical investigation
- // the maximum of nested ifs is not set, but the expression length is max 256 char!
- *Samples:*
  - if(1=1, "true", "false") = Gives back „true" in every case

- if([area]>100, "big area", "small area") = Gives back „big area” or „small area” depending on the area field value

left - Gives back the left part of text

- *Syntax:* **left**(txt,*param*) – gives back the defined length (param as number) of the text (as txt) from left
- *Inverse function: right()*
- *Samples:*
  - left(123456789,5) = 12345 – because this is the first five letter
  - left("Abies Alba",1) = A – because „A” is the first letter
  - left([TreeSpec],5) = give back the Treespec filed left 5 characters

len – Length of string

- *Syntax:* **len**(*param*) – calculate the length of the string (as param)
- *Samples:*
  - len("Ash") = 3, because „Ash” contains 3 letters
  - len(1024) = 4, because „1024” contains 4 characters
  - len([area]) = Gives back the area field characters length

lg - Calculate logarithm

- *Syntax:* **lg**(*param*) – calculate logarithm of the number (as param)
- // the param need to be >=0
- *Samples:*
  - lg(10) = 1
  - lg(1/2) = -0.30102999566398

ln - Calculate natural logarithm

- *Syntax:* **ln**(*param*) – calculate natural logarithm of the number (as param)
- // the param need to be >=0
- *Samples:*
  - ln(1) = 0
  - ln(1/2) = -0.69314718055995

lower – Transform characters to its lowercase equivalents

- *Syntax:* **lower**(*txt*) –transform to lowercase equivalents of the text (as txt)
- *Inverse function: upper()*
- *Samples:*
  - lower("Abies Alba") = „abies alba”
  - lower([Treespec]) = convert all of the Treespec filed values to lowercase one

mid - Gives back the middle part of text

- *Syntax:* **mid**(txt,*from,length*) – gives back the middle part of the text from left of the position (from as number), to the defined length (length as number) of the text (as txt)
- *Similar functions: right(),left()*
- *Samples:*
  - mid(123456789,2,3) = 234 – because this is the 3 character from 2nd position

- mid("Abies Alba",6,4) = Alba – because „Alba” is the „middle” of this text
- mid("TreeSpecies",0,4) = „Tree” – this is the (0…4) characters

replace – Replace part of text

- *Syntax:* **right**(txt,*which,what*) – replace sting1 (as text) of the string2 (as text) from right
- *Samples:*
  - replace("Aabies Alba","Aa","A") = „Abies Alba”
  - replace([TreeSpec], "Aabies Alba","Abies Alba") = replace all „Aabies Alba” in Treespec field to „Abies Alba”

right - Gives back the left part of text

- *Syntax:* **right**(txt,*param*) – gives back the defined length (param as number) of the text (as txt) from right
- *Inverse function: left()*
- *Samples:*
  - right(123456789,5) = 56789 – because this is the last five letter
  - right("Abies Alba",1) = a – because „a” is the last letter
  - right([TreeSpec],5) = give back the Treespec filed right 5 characters

sin – Sinus

- *Syntax:* **sin**(*param*) – Sinus of degree in radian (param as number)
- *Samples:*
  - cos(-1/2) = -0.4794255386042 (rad)
  - sin(0) = 0 (rad)

sqrt – Square root

- *Syntax:* **abs**(*param*) – Absolute value of param (as number)
- // the param need to be >=0
- *Samples:*
  - sqrt(9) = 3
  - sqrt(2) = 1.4142135623731
  - sqrt([FiledName]) = return the square root value of the numeric filed value

tan – Tangent

- *Syntax:* **tan**(*param*) – Arcus tangent of degree in rad (param as number)
- *Samples:*
  - tan(1/2) = 0.54630248984379 (rad)
  - tan(90) = -1.9952004122082 (rad)

upper – Transform characters to its uppercase equivalents

- *Syntax:* **upper**(*txt*) –transform to the uppercase equivalents of the text (as txt)
- *Inverse function: lower()*
- *Samples:*
  - upper("Abies Alba") = „ABIES ALBA”
  - upper([Treespec]) = convert all of the Treespec filed values to uppercase one

Complex examples

Alert in the data table:

If the input is too short in the Field1 or Field2, we give a message in the other field:

if((len([Field1])<10 or (len([Field2])<10 ,”Please type more detailed description”,””)

Code Dictionary to Tree Species scientific names:

If the same-ordered code dictionary defined for CommonNames and SciNames field, the SciNames can be set up automatically:

[CommonName]

…

Ask more examples at our forum or by mail: **support@digiterra.hu!**