How to use expressions in Templates to perform Skip logic and calculate Summary values

Expressions

Talea uses an expression language to define skip logic and calculate summary values. This language uses 'functions' (such as OR and DIVIDE) that define an action to be done to 'variables' (such as a number or choices selected for observations, for example 'female' in answer to the question 'dog type'). These functions and parameters must be written in a particular logical order to be understood by the system.

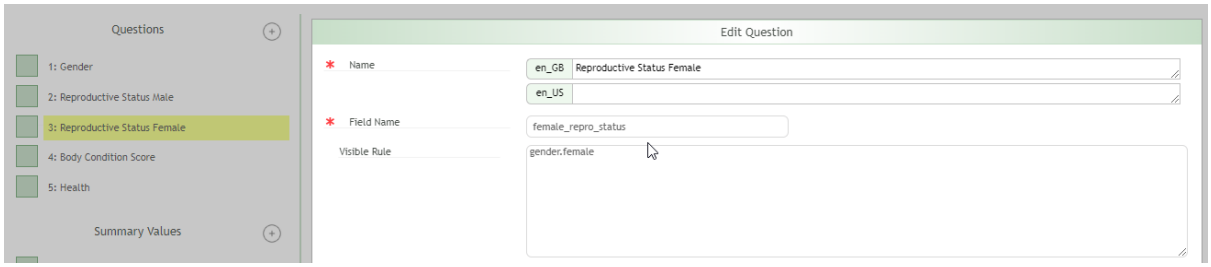
Here we describe how to use this language when writing skip logic and calculating summary values.

Skip logic

Sometimes you will want to change whether a question or choice is visible or required, based on the answer to a previous question. For example, you might not want to the question about lactation to be visible if the animal was previously recorded to be a male. This is called 'skip logic' because you are asking to skip past a question or option according to the logic that you define by an expression.

When used in skip logic, the expression must always return 'true' or 'false' (known as a Boolean value).

When used in skip logic, question.choice variables are 'true' or 'false' (Booleans). For example, a question about the reproductive status of females is only visible when the previous question 'gender' was answered with 'female'. The expression language (gender.female) is entered into the 'Visible Rule' box; so when gender = female is true, the question will be visible. If gender = female is not true, if gender had been answered in another way (e.g. male), this question would not be visible. If this box had been left empty, the question would always visible.



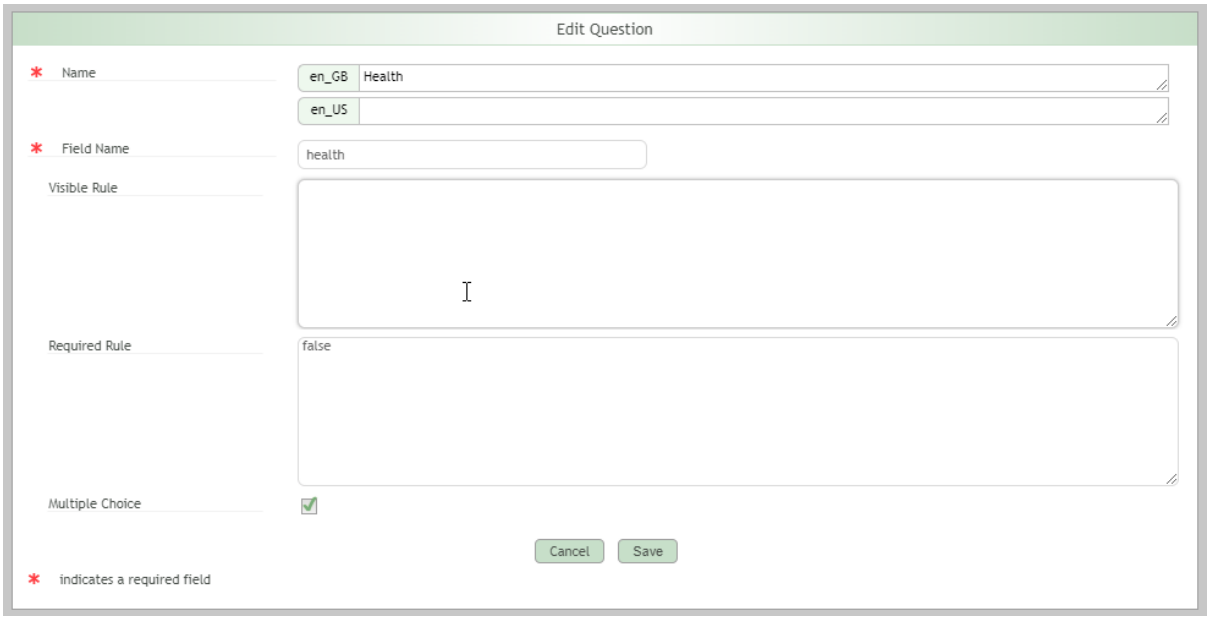
In this example, the expression language is only the variable 'gender.female'. But you can also add functions to this expression. For example NOT(gender.male) would mean the question would be visible if gender.male was false, in other words, if 'male' had NOT been selected in answer to the question of 'gender'.

An expression can be built up to include multiple variables separated with commas. For example, AND(gender.female, age.adult) means the gender must be 'female' AND the age must be 'adult' for the expression to be true. Only 2 variables can be included within a set of brackets, so if you have

more than 2 variables, these will need to be nested. For example AND(AND(gender.female, age.adult), reproduction.intact) means the gender must be ‘female’ AND the age must be ‘adult’ AND the reproductive status must be ‘intact’ for the expression to be true.

An expression can also include multiple functions nested within brackets. For example, NOT(AND(gender.female, age.adult)) means when the gender is ‘female’ AND the age is ‘adult’ the expression would return a false and the question would not be shown.

In the above examples we have used expressions to change the visibility of a question or choice. This language can also be used to change whether a question is required or not. In the following example, a multiple choice question about health has been set to false, by default (and when this box is empty) this would be ‘true’, meaning this question must be answered. An expression could also be used here, for example ‘NOT(age.puppy)’, this would mean this question about health would NOT have to be answered if the dog’s age had previously been answered as ‘puppy’.



Other functions that can be used in skip logic include:

(Note function names are not case sensitive)

And(<boolean>, <boolean>) : <boolean>

Return true if both the first and second parameters are true.

Blank(<variable>) : <Boolean>

Return true if the variable has no value. The variable can be of any type.

Not(<boolean>) : <boolean>

Return true if the parameter is false, else return true.

NotBlank(<variable>) : <boolean>

Return true if the variable has a value. The variable can be of any type.

NotEqual(<expression>, <expression>) : <boolean>

Return true if the first and second parameters are different.

Or(<boolean>, <boolean>) : <boolean>

Return true if either the first or second parameters is true.

Summary values

In your Template you can add 'summary values', these are calculations that will be performed on all the available data when you ask to access your data via Analysis. These calculations are defined by expressions.

When used in summary values, the expression always returns a number.

When used in summary values, question.choice variables are numbers, calculated by aggregating all the observations. For example, where the question is 'gender' and the choices are 'male' or 'female', and 7 observations were made with 4 males and 3 females, gender.male would be assigned the value 4 and gender.female would assigned the value of 3.

Four summary values are calculated automatically and can be used as part of your summary value expressions:

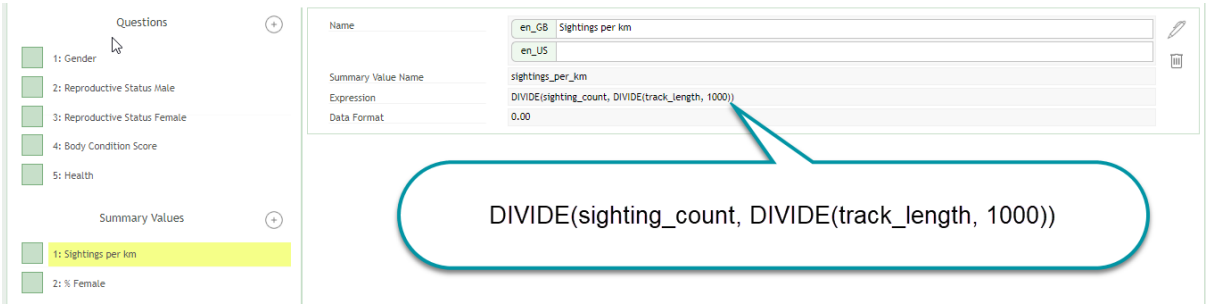
sighting_count = A simple count of all observations, excluding those that were deleted.

on_route_count = Count of undeleted observations that were recorded where the observer was within the buffer of the route.

track_length = The aggregate length of recorded track segments, in meters.

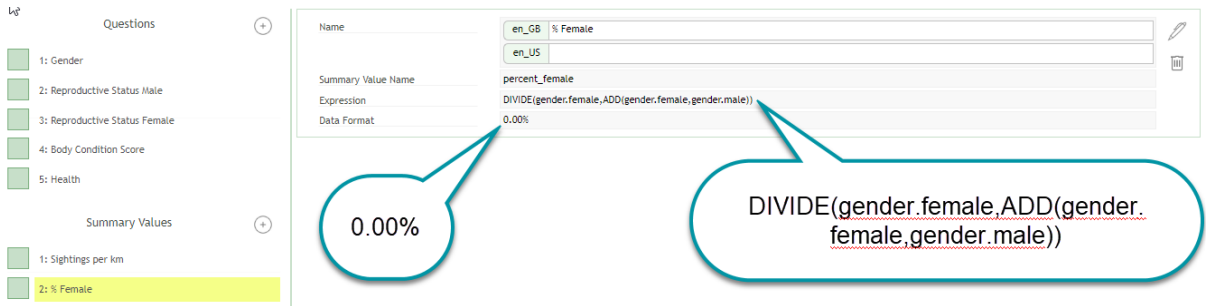
track_time = The total time of recorded track segments, in seconds (this excludes any time when the survey was paused).

For example, in the following example, the number of sightings per km of track surveyed is calculated using both the automatic summary values **sighting_count** and **track_length**.



You can also use summary values you have created previously as part of new summary value expressions.

The data format is also defined here, including the number of decimal places and any symbols such as "%". For example, the following expression returns the percentage of animals that are female, in the data file this value will be expressed to 2 decimal places and followed by the "%" sign.



Other functions that can be used in summary values include:

(Note function names are not case sensitive)

Add(<number>, <number>) : <number>
Add the first and second parameters and return result.

BetweenExclusive(<number>, <number>, <number>) : <boolean>

- Return true if first parameter is greater than the second parameter and less than the third parameter.

BetweenInclusive(<number>, <number>, <number>) : <boolean>

- Return true if first parameter is greater than or equal to the second parameter and less than or equal to the third parameter.

Divide(<number>, <number>) : <number>

- Divide the first parameter by the second parameter and return result.

Multiply(<number>, <number>) : <number>

- Multiply the first parameter by the second parameter and return result.

Random() : <number>

- Return a random number between 0 and 1.

Subtract(<number>, <number>) : <number>

- Subtract the second parameter from the first parameter and return result.

Try(<expression>, <expression>) : <expression>

- Return the first parameter. If that causes an exception then return the second expression.