CommCare Functions

Functions (also known as "xpath functions") are used when writing advanced logic inside a form, display conditions for a form or a case list filter. Functions can take some information, process it and return a value.

- Name: What type when you want to use that function
- Result: Each function will return some result. This could be a boolean (true/false), a string, a date, etc. Its important to understand what a function returns for when you use it.
- Arguments: When using a function, you can pass it additional information. For example, when you want to convert something to a date, you need to pass the original non-date value as an argument. A function can have zero, one, or more arguments.

Function names are case-sensitive. I.e. if you want to use the function today(), you cannot type Today() or TODAY().

Direct Values

These functions directly return a value with no arguments.

true

- **Return**: Returns the boolean value True
- **Arguments:** None
- **Syntax**: true()
- **Example:** You may want to use true() when you have some advanced logic for a display or validation condition. You could also use them if you want to calculate true for a hidden value. For example, if(/data/question1 = "yes" and /data/question2 > 30, true(), false())

false

- **Return**: Returns the boolean value False
- **Arguments:** None
- **Usage**: false()
- **Example Usage:** You may want to use false() when you have some advanced logic for a display or validation condition. You could also use them if you want to calculate false for a hidden value. For example, if(/data/question1 = "yes" and /data/question2 > 30, true(), false())

today

- **Return**: Returns the current date.
- **Arguments:** None
- **Usage**: today()
- **Example Usage:** You may want to use this when comparing against a user entered date. For example, you may want to check that entered EDD is in the future (/data/edd > today()).

now

- **Return**: Returns the current date.and time
- **Arguments:** None
- **Usage**: now()
- **Example Usage:** You may want to use this if you want to calculate the current date and time in a hidden value. When saved to a case, will only save the date portion without the time. If the time portion is important, convert to a number before saving to a case: double(now()).

here

- **Return**: Returns the current GPS position (Android only)
- **Arguments:** None
- **Usage**: here()
- **Example Usage:** Use this to get the current global position as a space separated string of [latitude longitude altitude accuracy] where the last two fields are measured in meters. This can be used in the case list with the distance() function below to, for example, sort your case list by cases that are nearest to your current position.
- Note: This function can be used **only** in case list or case detail calculations. It **cannot** be used within a form.

Formatting/Conversion Functions

These functions help convert one value into a value of another type. (ex. converting a string value into a date value).

boolean

- **Behavior:** When passed a number, will return true if the number is not zero. Otherwise it will return false. When passed a string, will return true if the string is non-empty.

- **Return**: Returns true or false based on the argument.
- **Arguments:** The value to be converted
- **Syntax**: boolean(value_to_convert)
- **Example:** You may have stored a value that is 1 or 0 into a boolean for other logic. boolean(/data/my_question) or boolean(23)

boolean-from-string

- **Behavior:** Will convert a string value of "1" or "true" to true. Otherwise will return false.
- **Return**: Returns true or false based on the argument.
- **Arguments:** The value to be converted
- **Syntax**: boolean-from-string(value_to_convert)
- **Example:** boolean(/data/my_question) or boolean("1")

number

- **Return**: Returns a number based on the passed in argument.
- **Behavior**: Will convert a string (ex. "34.3") into a number. Will cause an error if the passed in argument is not a number (ex. "two").
- **Arguments:** The value to be converted
- **Syntax**: number(value_to_convert)
- **Example:** If your application has a string value that needs to be stored as number. number(/data/my_string_number) or number("453")

int

- **Return**: Returns a whole number based on the passed in argument.
- **Behavior**: Will convert a string (ex. "34.3") or a decimal value into an integer. It will round down (ex. 34.8 will be evaluated to 34).
- **Arguments:** The value to be converted
- **Syntax**: int(value_to_convert)
- **Example:** int(45.6) or int("45.6") will return 45. You can also directly reference another question - int(/data/my_question).

double

- **Return**: Returns a double number based on the passed in argument.
- **Behavior**: Will convert a string (ex. "34.3") or a integer value into a double.
- **Arguments:** The value to be converted
- **Syntax**: double(value_to_convert)
- **Example:** double(45) or double("45") will return 45.0. You can also directly reference another question - double(/data/my_question).

string

- **Behavior:** Will convert a value into an equivalent string.
- **Return**: Returns a string based on the passed in argument.
- **Arguments:** The value to be converted
- **Syntax**: string(value_to_convert)
- **Example:** If you need to combine some information into a single string (using concatenate for example), you may need to convert some of those values into a string first. concat("You are ", **string(/data/age_question)**, " years old").

date

- **Behavior:** Will convert a string or a number value into an equivalent date. Will throw an error if the format of the string is wrong or an invalid date is passed.
- **Return**: Returns a date
- **Arguments:** The value to be converted (either a string in the format YYYY-MM-DD or a number).
- **Syntax**: date(value_to_convert)
- **Example:** If you have stored any date values in a case, they are actually stored as a string in the format YYYY-MM-DD. You will need to convert them into a date prior to using that for date math or when formatting for display. (ex. date(/data/case_edd)).
- **Notes**:
    - When working with dates prior to 1970 you should use date(floor(value_to_convert)) in order to avoid an issue where negative numbers are rounded incorrectly.

format-date

- **Behavior:** Will change the format of a date for display
- **Return**: Returns a string conversion of the provided date.
- **Arguments:** the date to be converted, and a string describing how it should be formatted. The syntax for the display format string is below
    - '%Y' = year
    - '%y' = 2 digit year
    - '%m' = 0-padded month
    - '%n' = numeric month
    - '%B' = full text month (English) (January, February, etc.)
    - '%b' = short text month (Jan, Feb, etc.)
    - '%d' = 0-padded day of month
    - '%e' = day of month

        - '%H' = 0-padded hour (24 hour time)
        - '%h' = hour (24 hour time)
        - '%M' = 0-padded minutes
        - '%S' = 0-padded second
        - '%3' = 0-padded milliseconds
        - '%a' = three-letter short text day (Sun, Mon, etc.)
        - '%A' = full text day (English) (Sunday, Monday, etc.) **Since:** This format is available on CommCare 2.32 and later
        - '%w' = numeric day of the week (0 through 6, 0 is Sunday)
    - **Syntax**: format-date(date_to_convert, display_format_string)
    - **Example:** When you are displaying a date in the display text, its useful to format it in a manner that is readable for your end users (the default is YYYY-MM-DD). Some examples
        - format-date(date(/data/my_date), "%e/%n/%y") will return a date that looks like D/M/YY
        - format-date(date(/data/my_date), "%a, %e %b %Y") will return a date that looks like Sun, 7 Apr 2012.
        - format-date(now(), '%y/%n/%e - %H:%M:%S') will return the current date and time in the format YY/M/D - HH:MM:SS
        - format-date(now(), '%H:%M:%S') will return the current time in the format HH:MM:SS

There are also uses of format-date() which can format dates to be appropriate for alternate calendars, such as the Nepali or Amharic calendars. More detail can be found at Alternate Calendar Formats for CommCare Android.

Logic and Math

+

- **Behavior:** Simple addition between two values
- **Return**: Returns the sum of the two values
- **Arguments:** Two numbers
- **Syntax**: x + y.
- **Example:** '5 + 7' returns 12. Many common uses.

-

- **Behavior:** Simple subtraction between two values
- **Return**: Returns the difference of the two values
- **Arguments:** Two numbers
- **Syntax**: x - y.
- **Example:** '3 - 12' returns -9. Many common uses.

*

- **Behavior:** Simple multiplication between two values
- **Return**: Returns the product of the two values
- **Arguments:** Two numbers
- **Syntax**: x * y.
- **Example:** '6 * 6' returns 36. Many common uses.

div

- **Behavior:** Simple division between two values
- **Return**: Returns the quotient of the two values
- **Arguments:** Two numbers
- **Syntax**: x div y.
- **Example:** '15 div 2' returns 7.5. Many common uses.

not

- **Behavior:** Will evaluate to true if the argument is false. Otherwise will return false.
- **Return**: Returns a boolean value (true or false)
- **Arguments:** The value to be converted
- **Syntax**: not(value_to_convert)
- **Example:** In some situations its easier to write the display or validation condition for when something *shouldn't* be shown. You can then pass this to the not function which will reverse it, allowing it to be used as a display condition. For example, not(/data/is_pregnant = "yes" and /data /has_young_children = "yes")

mod

- **Behavior:** x mod y, gives remainder when x is divided by y
- **Return**: Returns a number which is the remainder on dividing the two numbers
- **Arguments:** two numbers (divisor and divident)
- **Syntax**: x mod y.
- **Example:** '15 mod 4' returns 3; '6 mod 2' returns 0, and so on. Common use case, checking if a number is exactly divisible by another or not. eg, even number check, x mod 2 = 0
- **Limitation:** Note that mod does not work correctly for negative arguments. '-1 mod 12' should return 11, but returns -1. To avoid this limitation, add multiples of the second argument to the first to ensure it's never negative 'x + ay mod y' (11 mod 12 = 11).

if

- **Behavior:** Can be used to test a condition and return one value if it is true and another if that condition is false. Behaves like the Excel if function.
- **Return**: Will return either the value of the true or false branch.
- **Arguments:** The condition, the true value and the false value.
- **Syntax**: if(condition_to_test, value_if_true, value_if_false)
- **Example:** This function is very useful for complex logic. Ex. if(/data/mother_is_pregnant = "yes" and /data/mother_age > 40, "High Risk Mother", "Normal Mother"). If you need more complex logic (if a, do this, otherwise if b, do this, otherwise do c), you can nest if statements. Ex. if(data /mother_is_pregnant = "yes", "Is Pregnant", if(/data/mother_has_young_children = "yes", "Newborn Child Care", "Not Tracked"))

cond

- **Behavior:** Takes a set of test/expression pairs along with a default expression. The test conditions are evaluated in sequence and once one returns to true, 'cond' evaluates and returns the value of the corresponding expression and doesn't evaluate any of the other tests or expressions. If none of the test conditions evaluate to true, the default expression is returned.
- **Return**: Will return the value corresponding to one of the expression or the default expression.
- **Arguments:** Any number of test condition & expression pairs along with a default expression.
- **Syntax**: cond(first_condition, value_if_first_true, second_condition, value_if_second_true, ..., default_value)
- **Example:** This function is useful for avoiding nested if-statements. Instead of writing if(data/mother_is_pregnant = "yes", "Is Pregnant", if(/data /mother_has_young_children = "yes", "Newborn Child Care", "Not Tracked")) you can write cond(data/mother_is_pregnant = "yes", "Is Pregnant", /data/mother_has_young_children = "yes", "Newborn Child Care", "Not Tracked")
- **Since:** This function is available on CommCare 2.31 and later

random

- **Return**: Returns a random number between 0.0 (inclusive) and 1.0 (exclusive). For instance: 0.738
- **Arguments:** None
- **Usage**: random()
- **Example Usage:** When you need to generate a random number. For example, to generate a number between 5 and 23, you can use (random()* (23 - 5)) + 5. This will be something like 12.43334. You can convert that to a whole number by using int((random()*(23 - 5)) + 5). You can also reference questions instead of directly typing numbers. Ex. int(random()*(/data/high_num - /data/low_num) + /data/low_num).

sum

- **Behavior:** Sum the items in a group (ex. a question in a repeat group)
- **Return**: Will return the sum of all items.
- **Arguments:** The group of questions to be summed.
- **Syntax**: sum(question_group_to_be_summed)
- **Example:** This is useful if you have a repeat and need to add up the values entered for one of the questions. Ex. sum(/data/my_repeat_group /some_number_question).

min

- **Behavior:** Return the minimum value of the passed in values. These can either be a reference to a group of questions or a direct set of values.
- **Return**: Number that is the minimum.
- **Arguments:** There are two potential ways this function will work
    - Single argument that is the group of questions in which to find the minimum
    - Multiple arguments (an unlimited number) in which to find the minimum.
- **Syntax**: min(question_group) or min(value_1, value_2, value_3, ...)
    - NB: all values must exist within the min function. If a value is skipped or blank in the form, the function will not work correctly!
- **Example:** You can use this when you want to find the minimum number entered in a repeat group. Ex. min(/data/repeat_group /my_number_question). Or when you have multiple questions. Ex. min(/data/question_1, /data/question_2, /data/question_3, /data/question_4).

max

- **Behavior:** Return the maximum value of the passed in values. These can either be a reference to a group of questions or a direct set of values.
- **Return**: Number that is the maximum.
- **Arguments:** There are two potential ways this function will work
    - Single argument that is the group of questions in which to find the maximum
    - Multiple arguments (an unlimited number) in which to find the maximum.
- **Syntax**: max(question_group) or max(value_1, value_2, value_3, ...)
    - NB: all values must exist within the max function. If a value is skipped or blank in the form, the function will not work correctly!
- **Example:** You can use this when you want to find the maximum number entered in a repeat group. Ex. max(/data/repeat_group /my_number_question). Or when you have multiple questions. Ex. max(/data/question_1, /data/question_2, /data/question_3, /data/question_4).
- **Edge Cases:**
    - When providing a single argument which is a group of questions (IE: max(/data/repeat_group/my_number_question)), if there are *no matching questions* in the group of questions (for example, the group is a repeat where no items were added), max() produces an argument which is not a number.

regex

- **Behavior:** Evaluates a value against a regular expression and returns true if the value matches that regular expression.
- **Return**: true or false
- **Arguments:** There are two arguments, the value to be validated and the regular expression as a string.
- **Syntax**: regex(value, regular_expression)

- **Example:** This is useful when doing complex validation against some value. For example, to validate that a string contains only numbers, you can use regex(/data/my_question, "[0-9]+"). You can test and develop other regular expressions using http://www.regexr.com/. Also see the Advanced Validation Conditions page.

uuid()

- **Behavior:** Calculates a ***random*** hexadecimal (0-9, a-f) identifier of 32 characters long. There are 16^32 = 3.4*10^38 possibilities for this kind of identifier. The statistical chance that these values are not unique is extremely low, so we call it a **unique identifier**. https://en.wikipedia.org/wiki/Universally_unique_identifier
- **Return**: The unique id.
- **Arguments: none!**
- **Syntax**: uuid(). Omitting the argument results in a 32 digit unique identifier.
- **Examples:** uuid() could return 24235c71-09bf-40e8-87d2-00767efd7a14 or 4498b8a9-0af4-4413-994a-ad44e166f073

uuid(argument)

- **Behavior:** Calculates a ***random*** alphanumeric (0-9, A-Z) identifier of a particular length. The longer the length, this more likely it is that this random number is unique across the project. If the argument is 3, then there are 36^3 = 4.7*10^4 possibilities. This is a large number of possibilities, but not large enough to ensure that there is not a duplicate. If the argument is 32, there will be 36^32 = 6.3*10^49 possibilities, which is statistically very unlikely to produce a duplicate. Note that uuid() with an argument is not a traditionally-defined UUID as described here; https://en.wikipedia.org/wiki/Universally_unique_identifier.
- **Return**: The random id.
- **Arguments:** The length of the random id
- **Syntax**: uuid(length).
- **Examples:** uuid(4) could return WZV4 or 5J43. uuid(32) could return 4KV5JRAUM48YS9SP2SWX2G94UEAJBHXQ or N9HTXSZPJI0H8GQS2SBW88V881CJEN1I.

coalesce

- **Behavior:** Useful for choosing which of two values to return. Will return the non-null/empty value. If both are not null, will return the first argument.
- **Return**: One of the values
- **Arguments:** The two values to be coalesced
- **Syntax**: coalesce(value_1, value_2).
- **Example:** This is useful if you want to use a default value when referring to a question which may or may not have been answered. Ex. coalesce(/data/my_question, "my default value").
- **Note:** Since CommCare version 2.31, coalesce accepts more than 2 arguments, returning the first non-null argument.

depend

- **Behavior:** Used to force the engine to re-calculate the first argument when any of the other arguments change
- **Return**: The first argument passed in
- **Arguments:** 1 or more arguments
- **Syntax**: depend(expression, ..., expression)
- **Example:** depend(/data/some_variable, /data/count, /data/dob)

checklist

- **Behavior:** Performs a checklist computation, calculating if at least some number or a maximum number of items are answered a particular way.
- **Return**: True or false depending on the checklist (if number of true items is between the minimum and maximum specified).
- **Arguments:**
  - The first argument is a numeric value expressing the minimum number of factors required. If -1, no minimum is applicable
  - The second argument is a numeric value expressing the maximum number of allowed factors. If -1, no maximum is applicable
  - arguments 3 through the end are the individual factors, each treated as a boolean.
- **Syntax**: checklist(min_num, max_num, checklist_item_1, checklist_item_2, ...)
- **Example:** You may want to check that the mother has at least 2 out of 4 high risk symptoms. Ex. checklist(2, -1, /data/high_risk_condition_1 = "yes", /data/high_risk_condition_2 = "yes", /data/high_risk_condition_3 = "yes", /data/high_risk_condition_4 = "yes")

weighted-checklist

- **Behavior:** Similar to a checklist but each item is assigned a weight. Will return true if the total weight of the true items is between the range specified.
- **Return**: True or false depending on the weighted-checklist (if value of the weighting is within the specified range).
- **Arguments:**
  - The first argument is a numeric value expressing the minimum value. If -1, no minimum is applicable
  - The second argument is a numeric value expressing the maximum value. If -1, no maximum is applicable
  - arguments 3 through the end come in pairs. The first is the value to be checked and the second is the weight of that value.
- **Syntax**: weighted-checklist(min_num, max_num, checklist_item_1, checklist_item_weight_1, checklist_item_2, checklist_item_weight_2, ...)
- **Example:** weighted-checklist(-1, 2, /data/high_risk_condition_1 = "yes", 0.5, /data/high_risk_condition_2 = "yes", 2.5, /data/high_risk_condition_3 = "yes", 0.75)

pow

- **Behavior:** Raises a number to an exponent, b^n
- **Return**: The value of the the first argument, raised to the power of the second argument
- **Arguments:**

        - The first argument is a numeric value
        - The second argument is the numeric exponent to which the first argument should be raised. It can be a negative value.
            - **NOTE:** Due to technical restrictions the Exponent can only be an **integer** (non-decimal) value on Java Phones (Nokia, etc.). Decimal values can be used elsewhere.
- **Syntax**: pow(value, exponent)
- **Example:** pow(2.5, 2)
- **Since:** The pow function is available on CommCare 2.9 and later.

exp

- **Behavior:** Raises Euler's constant to the power of the provided number
- **Return**: A number representing e^x, where e is Euler's constant and x is the argument.
- **Arguments:** A expression that evaluates to a number, used as the exponent
- **Syntax**: exp(value)
- **Example:** exp(0) -> 1

pi

- **Behavior:** Returns Pi
- **Return**: Pi
- **Arguments:** None
- **Syntax**: pi()

sqrt

- **Behavior:** Calculates the square root of a number
- **Return**: A double value that represents the square root of the provided argument.
- **Arguments:** An expression that evaluates to a number
- **Syntax**: sqrt(value)
- **Example:** sqrt(4) -> 2.0

log

- **Behavior:** Takes the natural logarithm of a number
- **Return:** The natural logarithm of the argument passed to the function
- **Arguments:** The only argument is the number whose natural logarithm you want to take
    - **NOTE:** A negative argument will return a blank value.
- **Syntax:** log(number)
- **Example:** log(2.49)
- **Since:** The log function is available on CommCare 2.18 and later

log10

- **Behavior:** Takes the base-10 logarithm of a number
- **Return:** The base-10 logarithm of the argument passed to the function
- **Arguments:** The only argument is the number whose base-10 logarithm you want to take
    - **NOTE:** A negative argument will return a blank value
- **Syntax:** log10(number)
- **Example:** log10(2.49)
- **Since:** The log function is available on CommCare 2.18 and later

abs

- **Behavior:** Finds the absolute value of a number.
- **Return:** The absolute value of the argument passed to the function
- **Arguments:** The only argument is the number whose absolute value you want
- **Syntax:** abs(number)
- **Example:** abs(-2.49)
- **Since:** This function is available on CommCare 2.19 and later

ceiling

- **Behavior:** Finds the smallest integer that is greater than or equal to a number
- **Return:** The smallest integer that is greater than or equal to the given number
- **Arguments:** The only argument is the number whose ceiling you want
- **Syntax:** ceiling(number)
- **Example:** ceiling(2.49)
- **Since:** This function is available on CommCare 2.19 and later

floor

- **Behavior:** Finds the largest integer that is less than or equal to a number
- **Return:** The largest integer that is less than or equal to the given number

- **Arguments:** The only argument is the number whose floor you want
- **Syntax:** floor(number)
- **Example:** floor(2.49)
- **Since:** This function is available on CommCare 2.19 and later

round

- **Behavior:** Rounds a number to the nearest integer
- **Return:** The argument passed to the function, rounded to the nearest integer.
    - **NOTE**: Rounding negative numbers can be counter-intuitive. round(1.5) returns 2, while round(-1.5) returns -1.
- **Arguments:** The only argument is the number you want to round
- **Syntax:** round(number)
- **Example:** round(2.49)
- **Since:** This function is available on CommCare 2.19 and later

distance

- **Behavior:** Calculates the distance between two locations ***in meters***
    - **NOTE:** Although this function makes use of trig functions, it works on both Android and J2ME, using our custom implementation.
- **Return:** The distance between two locations, -1 if one of the locations is an empty string. Note that this returns the distance in meters. Make sure to convert to other units if needed (miles, kilometers, etc.)
- **Arguments:** The two locations. The locations may be passed in as strings consisting of four space-separated numbers denoting latitude, longitude, altitude, and accuracy. However, altitude and accuracy are optional, and are ignored by the distance function.
- **Syntax:** if(location1 = '', '', if(location2 = '', '', distance(location1, location2)))
- **Example:** distance("42 -71", "40 116")
- **Since:** This function is available on CommCare 2.26 and later

acos

- **Behavior:** Performs the arccosine operation
- **Return:** The inverse cosine of the value
- **Arguments:** The value whose inverse cosine you want to calculate
- **Syntax:** acos(number)
- **Example:** acos(.5)

asin

- **Behavior:** Performs the arcsine operation
- **Return:** The inverse sine of the value
- **Arguments:** The value whose inverse sine you want to calculate
- **Syntax:** asin(number)
- **Example:** asin(.5)

atan

- **Behavior:** Performs the arctangent operation
- **Return:** The inverse tangent of the value
- **Arguments:** The value whose inverse tangent you want to calculate
- **Syntax:** atan(number)
- **Example:** atan(.5)

cos

- **Behavior:** Performs the cosine operation
- **Return:** The cosine of the value
- **Arguments:** The value whose cosine you want to calculate
- **Syntax:** cos(number)
- **Example:** cos(.5)

sin

- **Behavior:** Performs the sine operation
- **Return:** The sine of the value
- **Arguments:** The value whose sine you want to calculate
- **Syntax:** sin(number)
- **Example:** sin(.5)

tan

- **Behavior:** Performs the tangent operation
- **Return:** The tangent of the value
- **Arguments:** The value whose tangent you want to calculate

- **Syntax:** tan(number)
- **Example:** tan(.5)

String Functions

concat

- **Behavior:** Joins multiple strings together. The arguments can either reference a question or be a directly type string.
- **Return**: Joined string
- **Arguments:** Multiple arguments (as many as needed) that represent the strings to be joined in order.
- **Syntax**: concat(text1, text2, text3, ...)
- **Example:** concat("Full name: ", /data/first_name, " ", /data/last_name)

contains

- **Behavior:** Tests if one string is contained within another string.
- **Return**: True or false.
- **Arguments:** The string to search in, followed by the string to search for.
- **Syntax**: contains(haystack, needle)
- **Example:** contains(/data/start_date, "2014")
- **Since:** This function is available on CommCare 2.19 and later

ends-with

- **Behavior:** Tests if one string ends with another string.
- **Return**: True or false.
- **Arguments:** The string to search in, followed by the string to search for.
- **Syntax**: ends-with(text, suffix)
- **Example:** ends-with(/data/word, "ita")
- **Since:** This function is available on CommCare 2.19 and later

join

- **Behavior**: Joins the values of all nodes in a given nodeset, or an arbitrarily long list of strings, with a given string. This can be used to get all the values of a node in a repeat group.
- **Return:** Joined string
- **Arguments:**
    - If joining a nodeset: A string to join the values of the nodeset with, and a nodeset.
    - If joining a list of strings:
        - First argument: A string to join the values in the list with
        - Subsequent arguments: List of strings to join
- **Syntax**:
    - Nodeset: join(text, my_nodeset)
    - List of strings: join(text, string_1, string_2, ..., string_n)
- **Examples**:
    - Nodeset: join(", ", /data/my_repeat/child_name)
    - List of strings: join(", ", /data/question1, /data/question2, /data/question3)

lower-case

- **Behavior:** Transforms all letters in a string to their lowercase equivalents.
- **Return**: Updated string
- **Arguments:** The string you want to transform.
- **Syntax**: lower-case(text)
- **Example:** lower-case("i AM a Test") -> "i am a test"
- **Since:** This function is available on CommCare 2.19 and later

replace

- **Behavior:** Searches a string for a pattern and replaces any occurrences of that pattern with another string.
- **Return**: String with any pattern matches replaced
- **Arguments:** Three arguments
    - The string to search in
    - A regular expression pattern to search for
    - The text with which to replace any matched patterns
    - **NOTE**: Unlike the XPath spec, this function does not support backreferences (e.g., using $1 in the replacement string to represent a matched group).
- **Syntax**: replace(text, pattern, replacement)
- **Examples**
    - replace("aaabbbaa", "a+", "a") returns "aba"
    - replace("abbbccd", "a.*c", "") returns "cd"
- **Since:** This function is available on CommCare 2.19 and later

starts-with

- **Behavior:** Tests if one string begins with another string.
- **Return**: True or false.
- **Arguments:** The string to search in, followed by the string to search for.
- **Syntax**: starts-with(text, prefix)
- **Example:** starts-with(/data/last_name, "Mc")
- **Since:** This function is available on CommCare 2.19 and later

substr

- **Behavior:** A substring function. Finds a specific part of the string (based on a start position and end position).
- **Return**: The substring identified.
- **Arguments:** Three arguments
    - The text value in which to find the sub string
    - The start position in the string. This is inclusive (so will include that character). The characters are numbered starting at 0.
    - The end position in the string. This is exclusive (so will not include that character). The characters are numbered starting at 0.
- **Syntax**: substr(text_value, start_position, end_position)
- **Example:** For example, you would like to grab just the year from the string "2012-09-21". You can use substr(/data/string_date, 0, 4)

string-length

- **Behavior:** The number of characters in a string.
- **Return**: A number (characters)
- **Arguments:** The string for which you need the length.
- **Syntax**: string-length(text_value)
- **Example:** You may have users entering some identifier (numbers and letters) and you'd like to validate that is of a specific length. Ex. string-length(/data/my_id_question)

selected-at

- **Behavior:** Extracts the nth word from a space-separated string.
- **Return**: The nth word from the string
- **Arguments:** The space-separated string and the position of the word that is to be returned. The count is zero-indexed so the first word is at position 0.
- **Syntax**: selected-at(text, string_position)
- **Example:** selected-at("I am a sentence to test", 3) -> "sentence"

translate

- **Behavior:** Replace each of a given set of characters in one string with one of another set of characters.
- **Return**: String with replacements made.
- **Arguments:** Three arguments
    - The string to manipulate
    - The set of characters to replace
    - The set of replacement characters. Any occurrences of the first character in the second argument will be replaced by the first character in this argument; any occurrences of the second character in the second argument will be replaced by the second character in this argument; etc. If there are fewer replacement characters than characters to replace, the "extra" characters will be deleted. If there are fewer characters to replace than replacement characters, the "extra" replacement characters will be ignored.
- **Syntax**: translate(text, to-replace, replacements)
- **Examples**
    - translate('aBcdE', 'xyz', 'qrs') returns "aBcdE"
    - translate('bosco', 'bos', 'sfo') returns "sfocf"
    - translate('yellow', 'low', 'or') returns "yeoor"
    - translate('bora bora', 'a', 'bc') returns "borb borb"
- **Since:** This function is available on CommCare 2.19 and later

upper-case

- **Behavior:** Transforms all letters in a string to their uppercase (capitalized) equivalents.
- **Return**: Updated string
- **Arguments:** The string you want to transform.
- **Syntax**: upper-case(text)
- **Example:** upper-case("i AM a Test") -> "I AM A TEST"
- **Since:** This function is available on CommCare 2.19 and later

encrypt-string

- **Behavior**: Takes a message string, a Base64-encoded secret key and an algorithm name and returns a Base64 encoded encrypted message. The only algorithm currently implemented is AES encryption in the GCM mode.
- **Return**: A Base64 encoded sequence of bytes, structured as follows:
    - Byte 0: the length of the initialization vector (IV_LEN)
    - Bytes 1 to IV_LEN: the initialization vector, which is needed for AES decryption
    - Bytes IV_LEN+1 to end: the encrypted message bytes

- **Arguments**: A message string, a Base64-encoded secret key, which should be 256 bits (32 bytes) for AES, and an algorithm name string, which should currently always be 'AES'.
- **Syntax**: encrypt-string(message, key, algorithm)
- **Example**: encrypt-string('abcdef', 'Vx1ZqYO06xSiYRoJNAqPtjXGAbjRS/lKqMukkoD0vEc=', 'AES') -> "DAqlxpaPUrTPsCnQ2yVoVM6xf7YqvN1LNA7jdp34NAb25yI="
- **Since**: This function is available in CommCare 2.51 and later.
- Click here for an example.

substring-before

- **Behavior:** Takes two strings, a base string and a query string and returns the substring of the base string that precedes the first occurrence of the query string, or the empty string if the base string does not contain the query string
- **Return**: A substring of the first argument
- **Arguments:** A base string and a query string.
- **Syntax**: substring-before(full_string, substring)
- **Example:** substring-before('hello_there', '_there'). In conjunction with string-length, this can calculate the index of the 1st occurrence of a query string: string-length(substring-before(base_string, query_string))+1
- **Since:** This function is available on CommCare 2.31 and later

substring-after

- **Behavior:** Takes two strings, a base string and a query string and returns the substring of the base string that follows the first occurrence of the query string, or the empty string if the base string does not contain the query string
- **Return**: A substring of the first argument
- **Arguments:** A base string and a query string.
- **Syntax**: substring-after(full_string, substring)
- **Example:** substring-after('hello_there', 'hello_') -> "there"
- **Since:** This function is available on CommCare 2.31 and later

Multi-Select Helper Functions

These functions are useful for working with a multi-select question. Multi-select questions store what is selected as a space-separated list of items. These functions will operate over that space separated list.

selected

- **Behavior:** Checks to see if a value was selected from a multiselect question. You cannot just do /data/my_question = "my_value_1" - this will fail if both "my_value_1" and "my_value_2" were selected.
- **Return**: True if that particular value was selected. Otherwise false.
- **Arguments:** Two arguments, the multi-select question and the value to check.
- **Syntax**: selected(my_question, value_to_check)
- **Example:** selected(/data/my_multi_select_question, "my_value_4").

count-selected

- **Behavior:** Counts the number of items selected in a multi-selected.
- **Return**: Returns the number of items selected.
- **Arguments:** The multi-select question (or a space-separated list of items).
- **Syntax**: count-selected(my_question)
- **Example:** You may want to check that at least three items were chosen in a multi-select question. Ex. count-selected(/data/my_question) >= 3

selected-at

- **Return:** Returns the nth selected item.
- **Arguments:** The multi-select question (or a space-separated list of items) and the number of selected item. **Note**: The number is zero-indexed. This means that to choose the first item, you need to enter 0, the second item, 1, etc.
- **Syntax**: selected-at(my_question, number)
- **Example:** To return the 3rd selected item, you can use selected-at(/data/my_question, 2).

Sequence Functions

Sequence functions provide limited support for manipulating data which are either a space separated list of items (like the output of a checkbox question) or a "nodeset" reference (such as a query into the casedb, or a reference questions inside of a Repeat Group). These functions require a good understanding of the underlying data model and form and should be tested carefully.

count

- **Behavior:** Counts the number of items in a group (ex. a repeat group)
- **Return**: Will return a number of items in the group.

- **Arguments:** The group of items to be counted.
- **Syntax**: count(/data/group)
- **Example:** This is useful if you have repeats in your form that allow the end user to choose the number of items. You may want to calculate how many items were chosen and store them in a hidden value. Ex. count(data/repeat_group)

distinct-values

- **Behavior:** Takes a group of values (ex. a question in a repeat group) and returns a group of values which contains only one instance of each unique value in the input group.
- **Return:** A group of values where no two values are the same. The order of the returned values is not guaranteed
- **Arguments:** Either a group of values, a reference to a group of values, or a string containing a space separated list of values.
- **Syntax**: distinct-values(input_sequence)
- **Examples :**
  - join(" ", distinct-values("a a b b c c c")) "a b c"
  - count(distinct-values(/data/myrepeat/myquestion)) Number of unique responses to 'myquestion'
- **Since:** This function is available on CommCare 2.41 and later

sort

- **Behavior:** Takes 1 string, which should be a space-separated string representing a list of strings, and an optional boolean value indicating the desired sort direction, and returns the sorted list
- **Return**: A string representing a sorted, space-separated list
- **Arguments:** A string representing a space-separated list, and an optional boolean argument indicating direction (true() is ascending, false() is descending, default is ascending)
- **Syntax**: sort("a b c d", true())
- **Examples:**
  - sort("4 2 1 5 3 2") --> "1 2 2 3 4 5"
  - sort("4 2 1 5 3 2", true()) --> "1 2 2 3 4 5"
  - sort("4 2 1 5 3 2", false()) --> "5 4 3 2 2 1"
- **Since:** This function is available on CommCare 2.38 and later

sort-by

- **Behavior:** 2 strings, each of which should be a space-separated string representing a list of strings, and an optional boolean value indicating the desired sort direction, and returns the result of sorting the 1st list by the 2nd list.
- **Return**: A string representing a sorted, space-separated list
- **Arguments:** 2 strings representing space-separated lists, and an optional boolean argument indicating direction (true() is ascending, false() is descending, default is ascending)
- **Syntax**: sort-by("a b c d", "3 2 4 1", true())
- **Examples:**
  - sort-by("apple banana chip dog egg flea", "4 2 1 5 3 2") --> "chip banana flea egg apple dog"
  - sort-by("apple banana chip dog egg flea", "4 2 1 5 3 2", false()) --> "dog apple egg flea banana chip"
- **Since:** This function is available on CommCare 2.38 and later