SMART Notation 3.0 Functions

1. Introduction

This document contains definitions of the functions that can be used in function expressions in SN3.

Note: mod, div, sort, first, last, only, .. are not functions but SN3 constructs.

2. Alphabetical list

This chapter contains an alphabetical list of functions, including precise definitions, arguments, outputs, and error situations.
Note that in all cases, the parameters must be given in the order defined here. Parameter names that are given in italics denote *optional* parameters.

Unless stated otherwise, the following situations result in compile-time errors for all functions.

- Wrong number of parameters.
- Parameters of illegal types.

3. Function groups

This chapter contains the definitions of the functions available in function expressions in SN3. The function definitions are grouped into the following 6 themes:

- Type conversion
- Numeric
- Text manipulation
- Temporal
- Collections
- Process functions and miscellaneous

If a function can be seen as belonging to more than one theme, cross-references are provided.

3.1 Type conversion functions

Type conversion functions are functions that convert a value to the type indicated by the function name. In principle, they can take any type of input value that can plausibly be converted to the desired type; in practice, most functions only convert from text-typed values.

to_boolean

Function name	Parameters	Result type
`to_boolean(val, locale)`	`val`: text (`'true'`, `'false'`) or integer (`1`, `0`). `locale`: *optional* locale/language tag, e.g. `'en'` or `'nl'` (default: `'en'`).	boolean

Converts the primary parameter to a Boolean value: `true` if `'true'` or `1`, `false` if `'false'` or `0`. The text values can be given in any combination of upper- and lower-case letters. The locale indicator specifies the language of the first parameter, if it's a text; e.g., if the indicator is `'nl'`, then the input values `'waar'` and `'onwaar'` can be converted. The default language is English. If the input value is an integer the locale indicator, if specified, is ignored.
`to_boolean('true')` ➔ `true`
`to_boolean('FALSE')` ➔ `false`
`to_boolean('onwaar', 'nl')` ➔ `false`
`to_boolean(1)` ➔ `true`		

Error *'Input of function to_boolean() cannot be converted to Boolean.' - if the parameter is of type text or integer but with any other than the allowed values.*		
Availability: GEARS v1.0.0		

to_decimal

Function name	Parameters	Result type
`to_decimal(val, locale)`	`val`: text or integer. `locale`: optional locale indicator (`'EN'`,	number (decimal)

	means periods are interpreted as decimal separators and commas as thousands separators; `'NL'` is the other way around); default: `'EN'`.	

Converts the primary parameter to a decimal value. The parameter has to be a text value or an integer value.
The text value has to represent a number in decimal notation or in scientific notation (a.k.a. exponential or E notation).
For decimal notation, the text must consist only of digits, with at most one decimal separator, with optional thousands separators in the correct positions before the decimal separator, and optionally preceded by a sign (+ or -).
For scientific notation, the text must consist of a number in regular decimal notation as above (with optional sign), followed by the letter E (upper or lower case), followed by an integer number (only digits) optionally preceded by a sign (+ or -).
The result is always a decimal, even if the text value contains no decimal separator.

```
to_decimal('-2.3') ➔ -2.3
to_decimal('2') ➔ 2.0
to_decimal('2.3E-4') ➔ 0.00023
to_decimal(2) ➔ 2.0
to_decimal('123,456', 'NL') ➔ 123.456
```

Error *'Input of function to_decimal() cannot be converted to decimal.' - if the parameter is of type text but does not represent a number as per the rules above. This is also the case if the thousands separators are in the wrong positions.*
Note*: In SN3 expressions, numbers are implicitly converted from integer to decimal if needed. The function to_decimal() can be used in cases where numbers are* not *implicitly converted.*

Availability: GEARS v1.0.0

to_integer

Function name	Parameters	Result type
`to_integer(val, locale)`	`val`: text. `locale`: optional locale indicator (`'EN'`, means commas are allowed as thousands separators; `'NL'` means periods are allowed as thousands separators); default: `'EN'`.	number (integer)

Converts the primary parameter to an integer value. The parameter has to be a text value consisting only of digits, with optional thousands separators in the correct positions before the decimal separator, and optionally preceded by a sign (+ or -).

```
to_integer('+2') ➔ 2
to_integer('1.000.000', 'NL') ➔ 1000000
```

Error *'Input of function to_integer() cannot be converted to integer.' - if the parameter is of type text but does not represent an integer number as per the rules above. This is also the case if the thousands separators are in the wrong positions.*
Note*: This function cannot be used to convert decimal numbers to integer. To convert decimal to integer, one of the rounded() functions should be used (see Numeric functions). If the text value represents a decimal number, an error occurs.*

Availability: GEARS v1.0.0

to_date

Function name	Parameters	Result type
`to_date(datetext, pattern, language)`	`datetext`: text representing a date value. `pattern`: *optional* date pattern indication. `language`: *optional* language tag, e.g. `'en'` or `'nl'` (default: `'en'`).	date

Converts the primary parameter to the corresponding date value. The format of the datetext is assumed to be the standard ISO 8601 “big-endian” order (uuuu-MM-dd) by default; if the datetext has a different format, such as dd-MM-uuuu or d-M-uu, this must be specified in the optional second parameter. See the Java DateTime Formatter definition for details. The language tag can be used when the datetext contains long or medium forms (i.e. full names or abbreviations of days or months); otherwise it is ignored.
Note that the language tag, when present, is always the third parameter. It is not correct to use a language tag if the pattern indication is absent.
`to_date('2020-04-03')` ➔ `2020-04-03`
`to_date('4/3/20','M/d/uu')` ➔ `2020-04-03`
`to_date('21 maart 2020','d MMMM uuuu','nl')` ➔ `2020-03-21`		

Error *‘Input of function to_date() is not a valid date.’ if the parameter does not correspond to a legal date in combination with the applicable date format and/or language tag (e.g. 2020-04-31, 2018-14-10, but also 2017-02-29; or 13-04-2020 with date format MM-dd-yyyy).*		
Availability: GEARS v1.0.0		

to_time

Function name	Parameters	Result type
`to_time(timetext, pattern)`	`timetext`: text representing a time value. `pattern`: *optional* time pattern indication.	time

Converts the primary parameter to the corresponding time value. The format of the timetext is assumed to be the standard HH:mm:ss format by default, where the hour of day is between 0 and 23 inclusive. If the timetext has a different format, e.g. without seconds or with milliseconds, or the 12-hour format with AM/PM indication, this must be specified in the optional second parameter. Seconds and milliseconds are set to 00.000 in the resulting time value if omitted. See the Java DateTime Formatter definition for details.
`to_time('15:41:27')` ➔ `15:41:27.000`
`to_time('9:24 PM','h:mm a')` ➔ `21:24:00.000`		

Error *‘Input of function to_time() cannot be converted to time.’ - if the parameter is of type text but does not represent a time value as per the rules above.* ***Note****: The to_time() function currently does not support nanoseconds, time zones and offsets.*		
Availability: GEARS v1.0.0		

to_datetime

Function name	Parameters	Result type
to_datetime(datetimetext, *pattern, language*)	datetimetext: text representing a datetime value. pattern: *optional* datetime pattern indication. language: *optional* language tag, e.g. 'en' or 'nl' (default: 'en').	datetime
Converts the primary parameter to the corresponding datetime value. The format of the datetimetext is assumed to be a combination of the default datetext and timetext value (as specified for the to_date() and to_time() functions), separated by a capital T. Other formats may be specified in the optional second parameter, and also consist of a date format and time format indication separated by a T or a space. See the Java DateTime Formatter definition for details. The time value (including the separator T) may be omitted, in which case the time part of the resulting datetime value is set to 00:00:00:000. The language tag can be used when the date part of the datetimetext contains long or medium forms (i.e. full names or abbreviations of days or months); otherwise it is ignored. Note that the language tag, when present, is always the third parameter. It is not correct to use a language tag if the pattern indication is absent. `to_datetime('2019-11-07T15:41:27')` ➔ `2019-11-07T15:41:27.000`		
***Error** 'Input of function to_datetime() cannot be converted to a datetime value.' - if the parameter is of type text but does not represent a datetime value as per the rules above.*		
Availability: GEARS v1.0.0		

Er is ook een variant met de volgende vorm, om een date en time te combineren tot een datetime, vanuit de opgegeven timezone:
to_datetime(<date>, <time>, <zone_name>) -> datetime

to_period

Function name	Parameters	Result type
to_period(periodtext, *language*)	periodtext: text representing a period value. language: *optional* language tag, e.g. 'en' or 'nl' (default: 'en').	period
Converts the primary parameter to the corresponding period value. The parameter is a text value that specifies a number of years, months, weeks, days, hours, minutes, seconds, and/or milliseconds. The quantities in the text must be digits, not words ('3 weeks', not 'three weeks'). Each of the constituents is optional, but those that are present must be provided in the large-to-small order shown here. The periodtext is not parsed for grammatical correctness: '3 week' and '3 weeks' are both accepted. If the periodtext is in another language than English, this should be indicated with the language tag. Note that the internal representation of periods does not use weeks, hours and minutes. Weeks are converted to days, hours and minutes to seconds. When converting a period value back to text, the seconds are automatically converted to minutes and hours; but days are *not* automatically converted to weeks. As a result, conversion of periods to and from text is not symmetrical.		

```
to_period('1 month 3 weeks 2 days 3 minutes 15 seconds') ➔ P0Y1M23D195S
to_period('2 dagen 3 uren 15 minuten, 'nl') ➔ P0Y0M2D11700S
Note the following:
to_text(to_period('1 month 3 weeks 2 days')) ➔ '1 month 23 days'
to_text(to_period('300 seconds')) ➔ '5 minutes'
to_text(to_period('12 months')) ➔ '12 months'
```

Error *'Input of function to_period() cannot be converted to a period value.' - if the parameter is of type text but does not represent a period value as per the rules above.*

Availability: GEARS v1.0.0

to_text

Function name	Parameters	Result type
to_text(value, *pattern*, *language/locale*) *NB: for periods, currently, function call is:* **to_text**(value, *language/locale, pattern*)	`value`: a value of any of the following types: boolean, number, date, time, datetime, period. `pattern`: *optional* specification of desired resulting text pattern. `language/locale`: *optional* language or locale tag, e.g. `'en'/'EN'` or `'nl'/'NL'` (default: `'EN'`).	text

Converts the primary parameter to a text value; details vary depending on the type of the parameter.
Boolean: optional language tag, no pattern. Converts the boolean value to the text `'true'` or `'false'` (or the equivalent in the specified language).

```
to_text(true) ➔ 'true'
to_text(true,'nl') ➔ 'waar'
```

Number (both integer and decimal): optional pattern indicator and locale tag; the locale tag, if present, is always the third parameter. If no pattern indicator is present, the default for integers is # (i.e., just the integer number as a sequence of digits with no leading zeros); the default for decimals is `#.0#` (i.e. no leading zeros in the integer part, no thousands separators, a period as decimal separator, at least one digit but otherwise no trailing zeroes in the decimal part, with locale EN). Currency symbols can be used as prefix, and the percentage sign can be used as a suffix (though not at the same time). See the Java DecimalFormat definition for details.
Note that the to_text() function implicitly rounds decimal numbers under the same rules as the function rounded(). This means that if the number of decimal digits of the primary parameter exceeds the specified pattern (including if the number is a decimal and the pattern specifies only an integer part), then .

```
to_text(1000000) ➔ '1000000'
to_text(21.05) ➔ '21.05'
to_text(3475.3,'00.###E0') ➔ '34.753E2'
to_text(3475.3,'€#.00','NL') ➔ '€3475,30'
to_text(0.43,'0%') ➔ '43%'
```

For all three types date, time, and datetime below, see the Java DateTime Formatter definition for details.
Date: optional pattern indicator and language tag; the language tag, if present, is always the third parameter. The date value is converted to a text value that conforms to the specified pattern; the

default pattern is ISO big-endian (uuuu-MM-dd).5

```
to_text(2020-04-03) ➔ '2020-04-03'
to_text(2020-04-03,'eeee d MMMM uuuu','nl') ➔ 'vrijdag 3 april 2020'
```

Time: optional pattern indicator. The time value is converted to a text value that conforms to the specified pattern; the default pattern is (HH:mm:ss).

```
to_text(15:41:27:000) ➔ '15:41:27'
to_text(15:41:27:000, 'h:mm a') ➔ '3:41 PM'
```

Datetime: optional pattern indicator and language tag; the language tag, if present, is always the third parameter. The datetime value is converted to a text value that conforms to the specified pattern; the default pattern is the concatenation of the default formats for date and time, separated by a T. The pattern may consist of only a date or time part, in which case the part missing from the pattern is not included in the text.

```
to_text(2019-11-07T15:41:27:000) ➔ '2019-11-07T15:41:27'
to_text(2019-11-07T15:41:27:000,'dd/MM/uuuu') ➔ '07/11/2019'
```

Period: optional language tag, no pattern. The period value is converted to a grammatically correct text value in the specified language. Note that the resulting text does not contain weeks; seconds are normalised to minutes and hours, but not to days (see also the function to_period()).

```
to_text(P0Y1M23D195S) ➔ '1 month 23 days 3 minutes 15 seconds'
```

Error *'Input of function to_text() cannot be converted to a text value.' - if the combination of the primary parameter and any of the optional parameters cannot be as per the rules above.*

Availability: GEARS v1.0.0

to_files

Function name	Parameters	Result type
`to_files(documents)`	`documents`: multiple DOCUMENT.	multiple FILE

Converts the given list of documents (type `multiple DOCUMENT`) to a list of file references (type `multiple file`).

For example:

```
"invoice is sent" =
    one INVOICE in MAILS is created with:
        from_address    = 'noreply@littlespider.com'
        ...
        attachments     = to_files([INVOICE_DOCUMENT, TERMS_OF_AGREEMENT])

"invoice document is created" =
    one INVOICE_DOCUMENT in DOCUMENTS is created with:
       type       = 'pdf'
       ...
```

Error *'Cannot convert to file' - if an argument cannot be converted to a file reference*
Availability: GEARS v1.9.2

filename

Function name	Parameters	Result type
`filename(val)`	`val`: FILE.	text
Gives the name of the given FILE parameter.		
Availability: GEARS v1.0.0		

NOTES

`to_binary`(x) - to be considered later, when binary types are implemented.
`to_json`(x) - to JSON, argument is a tuple; to be considered later.
`to_html`(x) - to HTML - rendered x as actual HTML on a page.

3.2 Numeric functions

Numeric functions are functions that perform calculations or transformations with numbers, both integers and decimals.

abs

Function name	Parameters	Result type
`abs(number)`	`number:` number (integer or decimal).	number (integer or decimal)

Returns the absolute value of the parameter. The result is the same type as the input.
`abs(-2.41)` ➔ `2.41`
`abs(4)` ➔ `4`		

-		
Availability: GEARS v1.0.0		

rounded

Function name	Parameters	Result type
`rounded(decimal, integer)`	`decimal:` decimal value. `integer`: *optional* integer value.	number (integer or decimal)

Rounds the primary parameter. If no integer value is provided, the decimal is rounded to 0 decimals and the resulting value is an integer. If the integer is a positive value >= 0, then the decimal value is rounded to the number of decimal positions specified by the integer value and the resulting value is a decimal. If the integer value is 0, then the decimal value is rounded to 0 decimals, but the resulting value is not of integer but of decimal type. Numbers are rounded using the “half up” rule, i.e. < 0.5 gets rounded down, ≥ 0.5 gets rounded up.
`rounded(3.78)` ➔ `4`
`rounded(3.78, 0)` ➔ `4.0`
`rounded(3.78, 1)` ➔ `3.8`		

Error *‘Input of function rounded() cannot be converted to decimal.’ if the second parameter is a negative integer.*		
Availability: GEARS v1.0.0		

rounded

Function name	Parameters	Result type
`rounded(period, unit)`	`decimal:` period value. `unit: unit of time to round to.`	period
Rounds the period parameter down to whole *unit*s. Possible values for unit are: nanos, seconds, minutes, hours, days, weeks, months, years.		

```
rounded('1 year 15 days', 'years') ➔ '1 year'
rounded('1 year 15 days', 'months') ➔ '1 year'
rounded('1 year 15 days', 'weeks') ➔ '1 year 2 weeks'
```

Error *'Invalid unit: …' if the second parameter is not one of the allowed units.*

Availability: GEARS v1.0.0

rounded_down

Function name	Parameters	Result type
`rounded_down(decimal, integer)`	`decimal`: decimal value. `integer`: *optional* integer value.	number (integer or decimal)

Rounds the primary parameter down. If no integer value is provided, the decimal is rounded to 0 decimals and the resulting value is an integer. If the integer is a positive value >= 0, then the decimal value is rounded to the number of decimal positions specified by the integer value and the resulting value is a decimal. If the integer value is 0, then the decimal value is rounded to 0 decimals, but the resulting value is not of integer but of decimal type. Numbers are rounded down, i.e. the decimals that are not needed are discarded.

```
rounded_down(3.78) ➔ 3
rounded_down(3.78, 1) ➔ 3.7
```

Error *'Input of function rounded_down() cannot be converted to decimal.' if the second parameter is a negative integer.*

Availability: GEARS v1.0.0

rounded_up

Function name	Parameters	Result type
`rounded_up(decimal, integer)`	`decimal`: decimal value. `integer`: *optional* integer value.	number (integer or decimal)

Rounds the primary parameter up. If no integer value is provided, the decimal is rounded to 0 decimals and the resulting value is an integer. If the integer is a positive value >= 0, then the decimal value is rounded to the number of decimal positions specified by the integer value and the resulting value is a decimal. If the integer value is 0, then the decimal value is rounded to 0 decimals, but the resulting value is not of integer but of decimal type. Numbers are rounded up, i.e. the integer part or the last decimal is incremented by 1 if the decimal part that is not needed > 0.

```
rounded_up(3.78) ➔ 4
rounded_up(3.78, 1) ➔ 3.8
```

Error *'Input of function rounded_up() cannot be converted to decimal.' if the second parameter is a negative integer.*

Availability: GEARS v1.0.0

sequence

Function name	Parameters	Result type
`sequence(sequence_name)`	`sequence_name:` text value.	number (integer)

result in a guaranteed unique sequence number. Currently this is implemented via the database. Because of that it requires that the sequence with the specified `sequence_name` is defined in the database. This can be done with the SQL command

```
CREATE SEQUENCE seq_order_nr
```

Example usage in SMART notation assuming this SQL command was issued on the database:

```
sequence('seq_order_nr') ➔ 1
sequence('seq_order_nr') ➔ 2
sequence('seq_order_nr') ➔ 3
... etc.
```

-

Availability: GEARS v1.0.0

sqrt

Function name	Parameters	Result type
`sqrt(number)`	`number:` number (integer or decimal).	number (decimal)

Computes the square root of the parameter. The result is always a decimal value. This number has the maximum available precision, and may need to be rounded before it can be stored or displayed.

```
sqrt(2) ➔ 1.41421356[..]
sqrt(4) ➔ 2.0
```

-

Availability: GEARS v1.0.0

sum

Function name	Parameters	Result type
`sum(numbers)`	`numbers:` collection of numbers (integers or decimals).	number (integer or decimal)

Computes the sum of the values that make up the collection. If these values are decimals, the resulting sum is also a decimal; if the values are integers, the resulting sum is also an integer.
The sum of an empty collection is defined as 0.

```
sum(CITIES.inhabitants) ➔ (sum of the numbers of inhabitants of all CITIES)
sum(ORDER.LINES.total_price) ➔ (sum of the total price of all the order lines of an order)
```

-

Availability: GEARS v1.0.0

random_number

Function name	Parameters	Result type
`random_number(lowerbound, upperbound)`	`lowerbound, upperbound`: number(integer)	number (integer)
results in a randomly generated number between and including lowerbound and upperbound. `random_number(1, 6)` ➔ 2, and any next time it could a different value ranging from 1, … , 6		
-		
Availability: GEARS v1.0		

NOTES

e(), **pi**() (constants) - These will not be implemented as functions. We consider these unnecessary, they are better represented as a sentence (with the desired precision) when needed.
average() - This function could be implemented someday when the need arises. Would presumably be more efficient (computationally) as a function than as a sentence.
xth_root() - Function for the (x)th root may be implemented someday, if the need arises. Low priority, never needed it yet.
log(), **ln**() - These functions may also be implemented someday, if the need arises. Low priority, never needed them yet.

3.3 Text manipulation functions

Text manipulation functions provide functionality to convert, combine or split (collections of) text values into other (collections of) text values.

capitalized

Function name	Parameters	Result type
`capitalized(text)`	`text:` text value.	text
Returns the parameter text, but if the first character in the text was a letter, it is converted to uppercase in the result. If the first character was already an uppercase letter, then the result is identical to the parameter. If the first character is not a letter, then the result is also identical to the parameter. `capitalized('this is a sentence.')` ➔ `'This is a sentence.'`		
-		
Availability: GEARS v1.0.0		

capitalized_words

Function name	Parameters	Result type
`capitalized_words(text)`	`text:` text value.	text
Returns the parameter text, but with the first character of each word converted to uppercase if it was a letter. If the first character of a word was already an uppercase letter, then the word is unchanged. If the first character is not a letter, then the word is also unchanged. A word is defined as a sequence of characters delimited on either end by either a space character, a tab character, a newline, or the beginning or end of the text value. `capitalized_words('This is a 4-word Sentence.')` ➔ `'This Is A 4-word Sentence.'`		
-		
Availability: GEARS v1.0.0		

lowercase

Function name	Parameters	Result type
`lowercase(text)`	`text:` text value.	text
Returns the parameter text, but with all uppercase letters in the text, if any, converted to lowercase. `lowercase('This is a Sentence.')` ➔ `'this is a sentence.'`		
-		
Availability: GEARS v1.0.0		

uppercase

Function name	Parameters	Result type
`uppercase(text)`	`text:` text value.	text
Returns the parameter text, but with all lowercase letters in the text, if any, converted to uppercase. `uppercase('This is a Sentence.')` ➔ `'THIS IS A SENTENCE.'`		
-		
Availability: GEARS v1.0.0		

trimmed

Function name	Parameters	Result type
`trimmed(text)`	`text:` text value.	text
Returns the parameter text, but with all leading and trailing spaces, tabs, and newlines stripped off. `trimmed('This is a sentence.  ')` ➔ `'This is a sentence.'`		
-		
Availability: GEARS v1.0.0		

normalized_spaces

Function name	Parameters	Result type
`normalized_spaces(text)`	`text:` text value.	text

Returns the parameter text, but with all tabs and newlines first replaced by spaces, and then all duplicate space characters replaced by a single space character.
`normalized_spaces('This  is a`
`  sentence.')` ➔ `'This is a sentence.'`		

-		
Availability: GEARS v1.0.0		

substituted

Function name	Parameters	Result type
`substituted(basetext, matchtext, replacetext)`	`basetext:` text value. `matchtext:` text value. `replacetext:` text value.	text
Returns the basetext parameter, but with all occurrences of the matchtext, if any, replaced by the		

replacetext. If there are no matches, then the basetext is returned unchanged. If the basetext is an empty string, the result is also an empty string. `substituted('To be, or not to be','be','do') ➔ 'To do, or not to do'`
-
Availability: GEARS v1.0.0

substituted_regex

Function name	Parameters	Result type
`substituted_regex(basetext, pattern, replacetext)`	`basetext:` text value. `pattern:` text value that denotes a Java regular expression (regex). `replacetext:` text value.	text
Returns the basetext parameter, but with all substrings that match the regex pattern, if any, replaced by the replacetext. The pattern must evaluate to a Java regular expression (see Java Patterns). If there are no matches, then the basetext is returned unchanged. If the basetext is an empty string, the result is also an empty string. `substituted_regex('Masking #MyPW','#[a-zA-Z]+','***') ➔ 'Masking ***'`		
Error *'Pattern parameter of function substituted_regex() is not a valid regular expression.' if the second parameter is a text but cannot be analysed to a valid regular expression.*		
Availability: GEARS v1.0.0		

concat

Function name	Parameters	Result type
`concat(coll_texts, separator)`	`coll_texts:` collection of text values. `separator:` *optional* text value.	text

Returns a text value that consists of a concatenation of all the text values in the collection that is the primary parameter, in the same order as in the collection. If the optional separator is not present, the text values in the collection are directly concatenated. If a separator is given, then the separator is inserted between the individual text values.
If the primary parameter is an empty collection (of type text), then the result is an empty string.
`concat(['This','is','a','sentence.']) ➔ 'Thisisasentence.'`
`concat(['This','is','a','sentence.'],' ') ➔ 'This is a sentence.'`
`concat(BEATLES.first_name,', ') ➔ 'John, Paul, George, Ringo'`
`concat([]) ➔ ''`		

-		
Availability: GEARS v1.0.0		

split

Function name	Parameters	Result type
split(text, separator)	text: text value. separator: text value.	multiple text
Splits the text value of the first parameter into a collection of text values, which are delimited and extracted from the input string based on the separator. If the separator is not present in the input, then the result is a collection with the input string as its sole element. If the separator is present at the beginning (and/or end) of the input string, the first (and/or last) of the elements of the resulting collection is an empty string. If the separator occurs twice (or more) in a row, the resulting collection contains one (or more) empty strings at this location. `split('This is a sentence.',' ') ➔ ['This','is','a','sentence.']`		
-		
Availability: GEARS v1.0.0		

position

Function name	Parameters	Result type
position(basetext, matchtext)	basetext: text value. matchtext: text value.	number (integer)
Returns the index in the basetext where the matchtext starts (first occurrence), if the matchtext is present in the basetext. If the matchtext is not present in the basetext, the result is 0. The indices start at 1: if the matchtext matches the start of the basetext, then the result is 1. If the basetext is an empty string, the result is always 0. `position('This is a sentence.','is') ➔ 3`		
-		
Availability: GEARS v1.0.0		

position_regex

Function name	Parameters	Result type
position_regex(basetext, pattern)	basetext: text value. pattern: text value that denotes a Java regular expression (regex).	number (integer)
Returns the index in the basetext where the regex pattern first matches, if it does. If the pattern does not match the basetext, the result is 0. The indices start at 1: if the pattern matches the start of the basetext, then the result is 1. If the basetext is an empty string, the result is always 0. `position_regex('This is a sentence.',' [a-z] ') ➔ 8`		
Error *'Pattern parameter of function position_regex() is not a valid regular expression.' if the second parameter is a text but cannot be analysed to a valid regular expression.*		

Availability: GEARS v1.0.0

substring

Function name	Parameters	Result type
`substring(basetext, position1, position2)`	`basetext`: text value. `position1`, `position2`: integer values denoting two positions in the basetext.	text

Returns the substring of the basetext between position1 and position2 (inclusive). Both numbers must be between 0 and the length of the basetext (inclusive), and position2 must be larger than or equal to position1. If position1 equals position2, then the result is a string of length 1, consisting of the character at that position.
`substring('This is a sentence.',4,7)` ➔ `'s is'`
`substring('This is a sentence.',2,2)` ➔ `'h'`		

Error *'Position parameter for function substring() out of bounds' if either of the integer parameters is too small or too large; error should specify which of the parameters is out of bounds.* *'Position parameters for function substring() out of order' if both of the integer parameters are within bounds, but position2 is smaller than position1.*		
Availability: GEARS v1.0.0		

length

Function name	Parameters	Result type
`length(text)`	`text`: text value.	number (integer)
Returns the length of the text parameter. If the parameter is an empty string, its length is 0. `length('This is a sentence.')` ➔ `19`		
-		
Availability: GEARS v1.0.0		

encode_password

Function name	Parameters	Result type
`encode_password(raw_password)`	`raw_password`: the unencrypted password to encode.	text
Encodes the specified raw, unencrypted password. `encode_password('Welkom123')` ➔ `'{bcrypt}$2a$10$g5slr...8vmAAGAVARElHAYsvym'`		
-		
Availability: GEARS v1.9.2		

generate_password

Function name	Parameters	Result type
`generate_password(length, categories)`	`length`: the length of the password to generate. `categories`: indicates which character categories to include in the generated password.	text
Generates a random password of the specified length. The parameter `categories` is a special indicator for which of the four character classes to include in the password: A: uppercase letters (A-Z); a: lowercase letters (a-z); 0: digits (0-9); !: symbols (!@#$%^&*()_-+=<>?/{}~|). `generate_password(12, 'Aa0!')` ➔ `'g)4$4T2jGHzY'`		
-		
Availability: GEARS v1.16.5		

3.4 Temporal functions

Temporal functions are functions that perform calculations and comparisons, or extract information from date, time, and datetime values.
Note: determining the name of a weekday or month from a date is done using the function **to_text**(), see type conversion functions.

current_date

Function name	Parameters	Result type
`current_date()`	any one parameter	date
Returns the date of the moment the function is evaluated. While this function needs no parameters, it can accept one parameter, of any type, which may create dependencies that affect the timing of its evaluation. `current_date()` ➔ *(date the function is evaluated, e.g. `2019-12-23`)* `current_date(ORDER.price)` ➔ *(current date, where the dependency ensures that the function will not be evaluated before ORDER.price is set)*		
-		
Availability: GEARS v1.0.0		

current_time

Function name	Parameters	Result type
`current_time()`	any one parameter	time
Returns the time (precision in milliseconds) of the moment the function is evaluated. While this function needs no parameters, it can accept one parameter, of any type, which may create dependencies that affect the timing of its evaluation. See `current_date()` function for an example. `current_time()` ➔ *(time the function is evaluated, e.g. `14:56:12.746`)*		
-		
Availability: GEARS v1.0.0		

current_datetime

Function name	Parameters	Result type
`current_datetime()`	any one parameter	datetime
Returns the datetime or timestamp (precision in milliseconds) of the moment the function is evaluated. While this function needs no parameters, it can accept one parameter, of any type, which may create dependencies that affect the timing of its evaluation. See `current_date()` function for an example. `current_datetime()` ➔ *(datetime the function is evaluated, e.g. `2019-12-23T14:56:12.746`)*		

-
Availability: GEARS v1.0.0

date_part

Function name	Parameters	Result type
`date_part(datetime)`	`datetime`: a datetime value; date values are also allowed.	date
Returns the date part of the datetime value. When the parameter is a date value, this value is returned unchanged. `date_part(2019-12-23T14:56:12.746) ➔ 2019-12-23`		
-		
Availability: GEARS v1.0.0		

time_part

Function name	Parameters	Result type
`time_part(datetime, zone_name)`	`datetime`: a datetime value; time values are also allowed. `zone_name`: the name of a time zone, e.g. 'Europe/Amsterdam'.	time
Returns the time part of the datetime value. When the parameter is a time value, this value is returned unchanged. `time_part(2019-12-23T14:56:12.746Z, 'Europe/Amsterdam') ➔ 13:56:12.746`		
-		
Availability: GEARS v1.15.7		

day_of_month

Function name	Parameters	Result type
`day_of_month(date/dt)`	`date/dt:` date or datetime value.	number (integer)
Returns the day number (in the month) for the provided date or datetime value. `day_of_month(2020-02-03) ➔ 3`		
-		
Availability: GEARS v1.0.0		

day_of_year

Function name	Parameters	Result type
`day_of_year(date/dt)`	`date/dt:` date or datetime value.	number (integer)
Returns the day number (in the year) for the provided date or datetime value. `day_of_year(2020-02-03)` ➔ `34`		
-		
Availability: GEARS v1.0.0		

week_of_year

Function name	Parameters	Result type
`week_of_year(date/dt)`	`date/dt:` date or datetime value.	number (integer)
Returns the week number for the provided date or datetime value (as per ISO 8601). `week_of_year(2019-12-23)` ➔ `52`		
-		
Availability: GEARS v1.0.0		

month_of_year

Function name	Parameters	Result type
`month_of_year(date/dt)`	`date/dt:` date or datetime value.	number (integer)
Returns the month number for the provided date or datetime value. `month_of_year(2019-12-23)` ➔ `12`		
-		
Availability: GEARS v1.0.0		

quarter_of_year

Function name	Parameters	Result type
`quarter_of_year(date/dt)`	`date/dt:` date or datetime value.	number (integer)
Returns the quarter number for the provided date or datetime value. `quarter_of_year(2019-12-23)` ➔ `4`		
-		
Availability: GEARS v1.0.0		

day_in_week

Function name	Parameters	Result type
day_in_week(week_nr, year_nr, day_name)	week_nr: the week number. year_nr: the year number. day_name: the name of the weekday.	date
day_in_week(date, day_name)	date: the year number. day_name: the name of the weekday.	

Returns the date of the week day in the week specified by either the combination of a week number and a year number or a date.

```
day_in_week(13, 2024, 'tuesday') ➔ 2024-03-26
day_in_week(2024-05-27, 'sunday') ➔ 2024-06-02
```

-

Availability: GEARS v1.16.5

days_between

Function name	Parameters	Result type
days_between(date/dt1, date/dt2)	date/dt1,2: date or datetime value.	number (integer)

Returns the number of days between date/datetime1 and date/datetime2. See below for the definition of “between”.
If date/datetime2 is before date/datetime1, then the result is a negative number. If either of the values is a date while the other is a datetime, then the date is extended with the time part 00:00:00.000 and the result is evaluated as if both values were datetimes.

```
days_between(2019-12-13, 2019-12-23) ➔ 10
days_between(2019-12-13T12:00:00.746, 2019-12-23T12:00:00.000) ➔ 9
days_between(2019-12-13T12:00:00.746, 2019-12-23T12:10:00.000) ➔ 10
```

About “between”
Since the term “between” is imprecise (especially in the sense of, are the boundaries included or not?), we define “between” in the context of the temporal [..]_between() - functions as follows.
For dates, “between” u1 and u2 is defined as the number of temporal units between u1 and u2, including one boundary (conceptually, the smaller one). In other words,
days_between(2019-12-13, 2019-12-15) = 2.
days_between(2019-12-13, 2019-12-13) = 0.
For times (including datetimes), “between” u1 and u2 is defined as the *whole* number of temporal units between u1 and u2, also including one boundary. In other words,
days_between(2019-12-13T12:00:00.000, 2019-12-15T11:00:00.000) = 1.
If either of the values is a date while the other is a datetime, then the date is extended with the time part 00:00:00.000, so both are regarded as datetimes.
See the individual [..]_between() - functions for examples.

-
Availability: GEARS v1.0.0

weeks_between

Function name	Parameters	Result type
`weeks_between(date/dt1, date/dt2)`	`date/dt1,2:` date or datetime value.	number (integer)

Returns the number of weeks between date/datetime1 and date/datetime2. See the function definition of **days_between**() for the definition of "between". This is equivalent to **days_between**(d1,d2) `div 7`. If date/datetime2 is before date/datetime1, then the result is a negative number. If either of the values is a date while the other is a datetime, then the date is extended with the time part 00:00:00.000 and the result is evaluated as if both values were datetimes.
`weeks_between(2019-12-13, 2019-12-20) ➔ 1`
`weeks_between(2019-12-13, 2019-12-19) ➔ 0`
`weeks_between(2019-12-13T12:00:00.746, 2019-12-27T12:00:00.000) ➔ 1`
`weeks_between(2019-12-13T12:00:00.746, 2019-12-27T12:11:00.000) ➔ 2`		

-		
Availability: GEARS v1.0.0		

months_between

Function name	Parameters	Result type
`months_between(date/dt1, date/dt2)`	`date/dt1,2:` date or datetime value.	number (integer)

Returns the number of months between date/datetime1 and date/datetime2. See the function definition of **days_between**() for the definition of "between".
If date/datetime2 is before date/datetime1, then the result is a negative number. If either of the values is a date while the other is a datetime, then the date is extended with the time part 00:00:00.000 and the result is evaluated as if both values were datetimes.
`months_between(2019-10-13, 2019-12-23) ➔ 2`
`months_between(2019-10-01, 2019-10-31) ➔ 0`
`months_between(2019-10-13T12:00:00.746, 2019-12-13T12:00:00.000) ➔ 1`
`months_between(2019-10-13T12:00:00.746, 2019-12-13T12:11:00.000) ➔ 2`		

-		
Availability: GEARS v1.0.0		

years_between

Function name	Parameters	Result type

years_between(date/dt1, date/dt2)	date/dt1,2: date or datetime value.	number (integer)

Returns the number of years between date/datetime1 and date/datetime2. See the function definition of **days_between**() for the definition of "between".
If date/datetime2 is before date/datetime1, then the result is a negative number. If either of the values is a date while the other is a datetime, then the date is extended with the time part 00:00:00.000 and the result is evaluated as if both values were datetimes.
`years_between(2018-10-13, 2019-12-23) ➔ 1`
`years_between(2018-10-13, 2019-10-12) ➔ 0`
`years_between(2018-10-13T12:00:00.746, 2019-12-13T12:00:00.000) ➔ 1`
`years_between(2018-10-13T12:00:00.746, 2019-12-13T12:10:00.000) ➔ 1`		

-		
Availability: GEARS v1.0.0		

milliseconds_between

Function name	Parameters	Result type
milliseconds_between(date/dt/t1, *date/dt/t2*)	date/dt/t1,2: date, datetime, or time value.	number (integer)

Returns the number of milliseconds between date/datetime/time1 and date/datetime/time2. See the function definition of **days_between**() for the definition of "between".
If date/datetime/time2 is before date/datetime/time1, then the result is a negative number. If either of the values is a date, then it is extended with the time part 00:00:00.000. If either of the values is a time, then the other one must be a time also, otherwise an error will result.
`milliseconds_between(2018-10-13, 2018-10-14) ➔ 86400000`
`milliseconds_between(12:00:00.746, 12:00:00.000) ➔ -746`
`milliseconds_between(2018-10-13T12:00:00.001, 2018-10-14) ➔ 43199999`		

***Error** 'Function milliseconds_between() cannot determine number of milliseconds between a time and a date or datetime' if one parameter is a time value and the other one is not.*		
Availability: GEARS v1.0.0		

seconds_between

Function name	Parameters	Result type
seconds_between(date/dt/t1, *date/dt/t2*)	date/dt/t1,2: date, datetime, or time value.	number (integer)
Returns the number of whole seconds between date/datetime/time1 and date/datetime/time2. See the function definition of **days_between**() for the definition of "between". If date/datetime/time2 is before date/datetime/time1, then the result is a negative number. If either of the values is a date, then it is extended with the time part 00:00:00.000. If either of the values is a time, then the other one must be a time also, otherwise an error will result. `seconds_between(2018-10-13, 2018-10-14) ➔ 86400`		

```
seconds_between(12:05:00, 12:00:00) ➔ -300
seconds_between(2018-10-13T12:00:01, 2018-10-14) ➔ 43199
```

Error *'Function seconds_between() cannot determine number of seconds between a time and a date or datetime' if one parameter is a time value and the other one is not.*

Availability: GEARS v1.0.0

minutes_between

Function name	Parameters	Result type
minutes_between(date/dt/t1, *date/dt/t2*)	date/dt/t1,2: date, datetime, or time value.	number (integer)

Returns the number of whole minutes between date/datetime/time1 and date/datetime/time2. See the function definition of **days_between**() for the definition of "between".
If date/datetime/time2 is before date/datetime/time1, then the result is a negative number. If either of the values is a date, then it is extended with the time part 00:00:00.000. If either of the values is a time, then the other one must be a time also, otherwise an error will result.

```
minutes_between(2018-10-13, 2018-10-14) ➔ 1440
minutes_between(12:05:00, 12:00:00) ➔ -5
minutes_between(2018-10-13T12:00:01, 2018-10-14) ➔ 719
```

Error *'Function minutes_between() cannot determine number of minutes between a time and a date or datetime' if one parameter is a time value and the other one is not.*

Availability: GEARS v1.0.0

hours_between

Function name	Parameters	Result type
hours_between(date/dt/t1, *date/dt/t2*)	date/dt/t1,2: date, datetime, or time value.	number (integer)

Returns the number of whole hours between date/datetime/time1 and date/datetime/time2. See the function definition of **days_between**() for the definition of "between".
If date/datetime/time2 is before date/datetime/time1, then the result is a negative number. If either of the values is a date, then it is extended with the time part 00:00:00.000. If either of the values is a time, then the other one must be a time also, otherwise an error will result.

```
hours_between(2018-10-13, 2018-10-14) ➔ 24
hours_between(17:05:00, 12:00:00) ➔ -5
hours_between(2018-10-13T12:00:01, 2018-10-14) ➔ 11
```

Error *'Function hours_between() cannot determine number of hours between a time and a date or datetime' if one parameter is a time value and the other one is not.*

Availability: GEARS v1.0.0

milliseconds_from_epoch

Function name	Parameters	Result type
milliseconds_from_epoch(date/dt)	date/dt: date or datetime value.	number (integer)

Returns the number of milliseconds between the so-called *epoch* or *UNIX epoch* (i.e., 1 Jan 1970, 00:00:00.000) and the given date/datetime. See the function definition of **days_between**() for the definition of "between".
If the given date/datetime is before the epoch, then the result is a negative number. If the given value is a date, then it is extended with the time part 00:00:00.000.
`milliseconds_from_epoch(1970-01-02) ➔ 82800000`
`milliseconds_from_epoch(1970-01-02T12:00:00) ➔ 126000000`		

-		
Availability: GEARS v1.0.0		

days_in_period

Function name	Parameters	Result type
days_in_period(period)	period: either a period value, or a text value that denotes a period (see function definition **to_period**()).	number (integer)

If the parameter is a period value, this returns the value of the day-part of the period value. If the parameter is a text value, the number of days consists of the number of weeks, if present, times 7, the number of days, and the number of days included in smaller time units (e.g. multiples of 24 hours).
`days_in_period(P0Y1M23D195S) ➔ 23`
`days_in_period('1 month 3 weeks 2 days 30 hours') ➔ 24`		

Error *'Input of function days_in_period() cannot be interpreted as a period value.' - if the parameter is of type text but does not represent a period value as per the rules for period values.*		
Availability: GEARS v1.0.0		

weeks_in_period

Function name	Parameters	Result type
weeks_in_period(period)	period: either a period value, or a text value that denotes a period (see function definition **to_period**()).	number (integer)
If the parameter is a period value, this returns the value of the day-part of the period value, divided by 7, rounded down. If the parameter is a text value, the number of weeks consists of the number of weeks, if present, the number of days divided by 7 rounded down, and the number of weeks included in smaller time units (e.g. multiples of 168 hours). `weeks_in_period(P0Y1M23DT195S) ➔ 3`		

`weeks_in_period('1 month 3 weeks 12 days')` ➔ `4`
Error *'Input of function weeks_in_period() cannot be interpreted as a period value.' - if the parameter is of type text but does not represent a period value as per the rules for period values.*
Availability: GEARS v1.0.0

months_in_period

Function name	Parameters	Result type
`months_in_period(period)`	`period:` either a period value, or a text value that denotes a period (see function definition **to_period**()).	number (integer)

If the parameter is a period value, this returns the value of the year-part of the period value times 12, plus the value of the month-part of the period value. The same calculation applies for text values.
`months_in_period(P1Y1M23DT195S)` ➔ `13`
`months_in_period('1 month 5 weeks 12 days')` ➔ `1`		

Error *'Input of function months_in_period() cannot be interpreted as a period value.' - if the parameter is of type text but does not represent a period value as per the rules for period values.*		
Availability: GEARS v1.0.0		

years_in_period

Function name	Parameters	Result type
`years_in_period(period)`	`period:` either a period value, or a text value that denotes a period (see function definition **to_period**()).	number (integer)

If the parameter is a period value, this returns the value of the year-part of the period value. For text values, this is the number given for the years in the text, plus the number of months, divided by 12 rounded down.
`years_in_period(P1Y1M23DT195S)` ➔ `1`
`years_in_period('1 year 23 months 5 weeks 12 days')` ➔ `2`		

Error *'Input of function years_in_period() cannot be interpreted as a period value.' - if the parameter is of type text but does not represent a period value as per the rules for period values.*		
Availability: GEARS v1.0.0		

NOTES

workdays_later(d, i) - From a date and an integer, return a date i workdays later (i.e, not counting Saturdays and Sundays)
workdays_between(d1, d2) - Number of workdays between two dates.
Function to combine a date and a time into a datetime?

3.5 Collection functions

Collection functions are functions that manipulate collections and return other collections, or features of the collection.
For functions **concat**() and **split**(), which concern collections of text values, see the Text manipulation functions. Note also that **sorting** of collections is not done using a function, but with a SMART Notation expression.

count

Function name	Parameters	Result type
`count(collection)`	`collection:` a collection of elements of any type.	number (integer)
Returns the size (number of elements) of the collection. Note: if the parameter is an empty collection, the returned result is the value 0. `count(BEATLES) ➔ 4`		
-		
Availability: GEARS v1.0.0		

distinct

Function name	Parameters	Result type
`distinct(collection)`	`collection:` a collection of elements of any type.	collection

Returns the distinct (a.k.a. unique, a.k.a. deduplicated) elements of the collection. For simple collections (e.g. lists of values) this is done based on the value. For complex collections of entities this is done with the unique identifier of the element in that collection. If a table is used to store this entity collection, the unique technical identifier of that table is used.
`distinct([1,1,5,2,3,5]) ➔ [1,5,2,3]`
`distinct(BEATLES + RINGO) ➔ BEATLES`		

-		
Availability: GEARS v1.11.7		

flattened

Function name	Parameters	Result type
`flattened([collection, collection, ...])`	a list of `collection:` each collection must contain elements of any type but they must have the same type.	collection
Returns one collection containing all of the elements in each of the `collections`.		

```
flattened([[1,1,5],[2],[3,5]]) ➔ [1,1,5,2,3,5]
flattened([[JOHN, PAUL], [GEORGE], [RINGO]]) ➔ [JOHN, PAUL, GEORGE, RINGO]
== BEATLES
```

Default error *'Elements must all have the same type, but they were …' if the collections don't all have the same type. A build error with message 'Functions cannot be applied to given types' occurs if the collection is not a nested collection.*

Availability: GEARS v1.11.7

without_undefined

Function name	Parameters	Result type
`without_undefined(collection)`	`collection:` a collection of elements of any type.	collection

Returns a collection with all the elements of the given collection, except any undefined values are omitted.

```
without_undefined([1,1,undefined,2,undefined,5]) ➔ [1,1,2,5]
```

-

Availability: GEARS v1.11.7

element_at

Function name	Parameters	Result type
`element_at(collection, position, message)`	`collection:` a collection of elements of any type. `position:` integer number indicating a position in the (ordered) collection. `message:` an *optional* stringLiteral (text value, optionally with placeholders) used as an error message in the system log.	(same type as elements)

Returns the element in the collection at the indicated position. The position of the first element in the collection is 1. If the indicated position is out of bounds for the collection, the function results in an error.
In case of error, if the optional *message* is specified, then this is inserted in the log. Since out-of-bounds errors are generally hard to pin down, it is recommended to always include a meaningful error message.

```
element_at([1,1,2,3,5,8,13,21],4) ➔ 3
element_at(BEATLES,6,'There are only 4 Beatles (officially).') ➔ (error)
```

Default error *'Position for element_at() out of range' if the position is <=0, or larger than the size of the collection. However, if the message argument is specified, the default error is overruled by the specified message.*

Availability: GEARS v1.0.0

min

Function name	Parameters	Result type
`min(collection)`	`collection:` a collection of elements of type integer, decimal, text, date, datetime, or time.	(same type as elements)

Returns the lowest value in the collection: i.e., for elements of type number, the smallest value; for texts, the first value if the collection is ordered alphabetically; for temporal values, the earliest in time.
Note: Alphabetical ordering is described in the SN3 Language Reference Manual, § 4.21.1.
Note: if the parameter is an empty collection, the function returns the value *undefined*.
`min([4,2,9,14,3,6,2,4]) ➔ 2`
`min(BEATLES.first_name) ➔ 'George'`		

-		
Availability: GEARS v1.0.0		

max

Function name	Parameters	Result type
`max(collection)`	`collection:` a collection of elements of type integer, decimal, text, date, datetime, or time.	(same type as elements)

Returns the highest value in the collection: i.e., for elements of type number, the largest value; for texts, the last value if the collection is ordered alphabetically; for temporal values, the latest in time.
Note: Alphabetical ordering is described in the SN3 Language Reference Manual, § 4.21.1.
Note: if the parameter is an empty collection, the function returns the value *undefined*.
`max([4,2,9,14,3,6,2,4]) ➔ 14`
`max(BEATLES.last_name) ➔ 'Starr'`		

-		
Availability: GEARS v1.0.0		

range

Function name	Parameters	Result type
`range(integer1, integer2)`	`integer1, integer2:` integer boundary values.	collection of integers

Returns a collection of integer values between integer1 and integer2. If integer1 is bigger than integer2, then the result is still a collection of integer values, but in descending order. If integer1 and integer2 are the same, the result is a collection of 1 value.
`range(4,9) ➔ [4,5,6,7,8,9]`
`range(20,15) ➔ [20,19,18,17,16,15]`		

`range(3,3) ➔ [3]`
-
Availability: GEARS v1.0.0

cumulative_plus

Function name	Parameters	Result type
`cumulative_plus(start, collection)`	`start`: number or temporal. `collection`: a collection of numbers or period	collection of numbers if `start` is a number and `collection` is a multiple number collection of temporals if `start` is a temporal and `collection` is a multiple period

Returns a collection of values that is the result of cumulatively adding the values in `collection` to `start`.
`cumulative_plus(4,[2, 6, -4]) ➔ [6, 12, 8]`
`cumulative_plus(2019-01-01,[to_period('3 days'), to_period('1 week')]) ➔ [2019-01-04, 2019-01-11]`		

-		
Availability: GEARS v1.15.6		

cumulative_minus

Function name	Parameters	Result type
`cumulative_minus(start, collection)`	`start`: number or temporal. `collection`: a collection of numbers or period	collection of numbers if `start` is a number and `collection` is a multiple number collection of temporals if `start` is a temporal and `collection` is a multiple period

Returns a collection of values that is the result of cumulatively subtracting the values in `collection` to `start`.
`cumulative_minus(4,[2, 6, -4]) ➔ [2, -4, 0]`
`cumulative_minus(2019-01-01,[to_period('3 days'), to_period('1 week')]) ➔ [2018-12-29, 2018-12-22]`		

-		

Availability: GEARS v1.15.6

distribute_capacity

Function name	Parameters	Result type
distribute_capacity(max, time_unit, starts, quantities)	max: an integer or decimal value describing the maximum of this resource per time unit time_unit: a period value indicating the size of the time_unit. starts: a list of datetime values indicating when work on a quantity should start. quantities: a list of those quantities. The size of this list should match the size of starts.	A collection of integer or decimal values describing the distributed quantities.

This function distributes work to be done given maximum per time_unit. This function is used as a simple way to do resource levelling.
In the 1st example below a person has a maximum capacity of 40 hours per week. In w(eek)1 the equivalent of 80 hours has to be produced, in w2 85 hours and in w3 32 hours.
The second example shows that this function will not try to move planned work to an earlier time unit if the maximum is not yet reached and as a result this function will add empty time units if needed. For instance in w1 only 20 hours need to be produced and in w3 60 hours. This could fit into the first 2 weeks. Instead only 60 hours will be distributed over the last 2 of in total 4 weeks:

```
distribute_capacity(40, to_period('1 week'), [w1, w2, w3], [80, 85, 32])
➔ [40, 40, 40, 40, 37]
distribute_capacity(40, to_period('1 week'), [w1, w3], [20, 60])
➔ [20, 0, 40, 20]
```

Default errors:

- *'*`the length of list `starts` and `quantities` should be the same.`*' if they are not.*

Note that it is allowed to use negative values for both max and quantities values. It is also allowed to have empty lists for both starts and quantities. This will simply result in an empty list [ ].

To shorten the examples we used `w1`, `w2`, *... Of course real datetimes should be used. E.g.* `to_datetime('2019-11-07T15:41:27')`.

Availability: GEARS 1.17.8

NOTES

subset_at(x,i1,i2[,mess]) - returns a collection of elts from (ordered) collection, between indices i1 and i2 (inclusive); mess is error message

KLANT.KINDEREN[2:4] = collection_at(KLANT.KINDEREN, 2, 4)

was get_collection()

replaced_collection(X,x1, x2) - 3 arguments, collection and two compatible elements, element 2 is added to collection replacing element 1, if present.

collection_head() - returns the collection minus the last element

collection_tail() - returns the collection minus the first element
inverted() - returns the collection in inverted order

3.6 Process functions and Miscellaneous

This last section groups functions that retrieve information about the process that is currently being executed, with miscellaneous other functions that do not fit into any of the above categories.

process_initiator

Function name	Parameters	Result type
`process_initiator()`	No parameters.	USER
Returns the user who initiated the current process instance. `process_initiator()` ➔ *(USER who initiated the current process instance)*		
-		
Availability: GEARS v1.0.0		

process_instance_id

Function name	Parameters	Result type
`process_instance_id()`	No parameters.	text
Returns the current process instance ID. `process_instance_id()` ➔ *(the UUID of the current process instance)*		
-		
Availability: NOT		

current_user

Function name	Parameters	Result type
`current_user()`	No parameters.	USER
Returns the current USER; in practice, this is the user who executed the last task (or, if evaluated before the first task, the user who initiated the process). However, in the context of information belonging to a task (such as based-on or chosen-from information), this is the user who will execute that task. While this function needs no parameters, it can accept one parameter, of any type, which may create dependencies that affect the timing of its evaluation. `current_user()` ➔ *(USER who is currently handling the process)*		
-		
Availability: only in SNEL		

unique_id

Function name	Parameters	Result type
`unique_id()`	No parameters.	text
Returns a unique ULID, such as `'01ARZ3NDEKTSV4RRFFQ69G5FAV'`. ULIDs consist of 26 characters of the alphabet `'0123456789ABCDEFGHJKMNPQRSTVWXYZ'`. `unique_id()` ➔ *(a new ULID)*		
-		
Availability: GEARS v1.15.6		

NOTES

Additional potential functions in this category, to be considered.

`process_name`() - no argument, returns the name of the current process (text).
`process_key`() - no argument, returns the key of the current process (text).
`process_subject`() - no argument, returns the subject of the current process instance (tuple).