HCLSoftware

Detect Expression Language Guide

Contents

Chapter 1. Detect Expression Language

Introduction

The HCL Detect Expression Language is a simple expression language used to specify filter conditions and basic arithmetic manipulations. It can be used as part of configuring triggers. In particular, a trigger's `WHEN` condition can have a free-form expression specified as part of a predicate's right hand-side. Similarly, a trigger's `THEN` action can have an expression specified as an assignment for an output attribute. These expressions are specified in HCL Detect Expression Language.

Types

An expression in HCL Detect Expression Language can have one of the following primitive types: `String` (a string), `Bool` (a Boolean), `Int16` (a 16-bit integer), `Int32` (a 32-bit integer), `Int64` (a 64-bit integer), or `Double` (a double precision floating point number). It can also have a list type, where the element type of the list is one of the primitive types: `List(String)`, `List(Bool)`, `List(Int16)`, `List(Int32)`, `List(Int64)`, `List(Double)`.

Literals

`Bool`, `Double`, `Int32`, `Int64`, and `String` literals can be present in an expression.

- A `Bool` literal can be either `true` or `false`.
- A `Double` literal is a decimal number that either contains the decimal separator (e.g., `3.5`, `.5`, `3.`) or is given in the scientific notation (e.g., `1e-4`, `1.5e3`). A *Double* literal must be between `(2 - 2 ^ -52) * 2 ^ 1023` (`~ 1.79 * 10 ^ 308`) and `-(2 - 2 ^ -52) * 2 ^ 1023` (`~ -1.79 * 10 ^ 308`). Moreover, the magnitude of a literal cannot be less than `2 ^ -1074` (`~ 4.94 * 10 ^ -324`). Furthermore,
    - a `NaN` (not a number) value can be specified through the `Double("nan")` expression,
    - an `Inf` (positive infinity) value can be specified through the `Double("inf")` expression,
    - and a `-Inf` (negative infinity) value can be specified through the `Double("-Inf")` expression.
- Integer literals without a suffix are of the `Int32` type (e.g., `14`). An `Int32` literal must be between `2 ^ 31 - 1` (`= 2147483647`) and `-2 ^ 31` (`= -2147483648`).
- The `L` suffix is used to create `Int64` literals (e.g., `14L`). An `Int64` literal must be between `2 ^ 63 - 1` (`= 9223372036854775807L`) and `-2 ^ 63` (`= -9223372036854775808L`).
- A `String` literal appears within double quotes, as in `"I'm a string literal"`. The escape character `\` can be used to represent new line (`\n`), tab (`\t`), double quote (`\"`) characters, as well as the escape character itself (`\\`).

Arithmetic Operations

An expression in HCL DetectExpression Language can contain the basic arithmetic operations: addition (`+`), subtraction (`-`), multiplication (`*`), division (`/`), and modulo (`%`) with the usual semantics. These are binary operations that expect sub-expressions on each side. An expression in HCL Detect Expression Language can also contain the unary minus (`-`) operation that expects a sub-expression on the right. Finally, parentheses (`()`) are used for adjusting precedence, as usual.

Addition operation corresponds to concatenation for the `String` type and is the only available operation on this type. `Bool` type does not support arithmetic operations. Division applies integer division on integer types and floating point division on `Double` types.

The modulo operation yields the remainder from the division of the first operand by the second. It always yields a result with the same sign as its second operand. The absolute value of the result is strictly smaller than the absolute value of the second operand.

Examples:

- A simple arithmetic expression: `(3+4*5.0)/2`
- A simple expression involving a `String` literal: `"area code\tcountry"`
- A unary minus expression: `-(3+5.0)`

Comparison Operations

An expression in HCL Detect Expression Language can contain the basic comparison operations: greater than (`>`), greater than or equal (`>=`), less than (`<`), less than or equal (`<=`), equals (`==`), and not equals (`!=`) with the usual semantics. These are binary operations. With the exception of equals and not equals, they expect sub-expressions of numerical types or strings on each side. For strings, the comparison is based on lexicographic order. For equals and not equals, the left and the right sub-expressions must be of compatible types with respect to HCL Detect Expression Language coercion rules. The result of the comparison is always of type `Bool`.

Examples:

- An expression using comparisons: `3>5`
- An expression using String comparisons: `"abc"<"def"`
- An expression using inequality comparison: `"abc"!="def"`
- An expression comparing a `Double` literal with a `NaN` (not a number) value: `10.5 != Double("nan")` (Note that in this case, the `math.isNaN` built-in function can also be used.)
- An expression comparing a `Double` literal with an `Inf` (positive infinity) value: `10.5 != Double("inf")` (Note that in this case, the `math.isPositiveInfinity` built-in function can also be used.)
- An expression comparing a `Double` literal with a `-Inf` (negative infinity) value: `10.5 != Double("-inf")` (Note that in this case, the `math.isNegativeInfinity` built-in function can also be used.)

Logical Operations

An expression in HCL Detect Expression Language can contain the basic logical operations: logical and (`&&`) and logical or (`||`) with the usual semantics. These are binary operations that expect sub-expressions of type `Bool` on each side. Shortcutting is used to evaluate the logical operations. For logical *and*, if the sub-expression on the left evaluates to `false`, then the sub-expression on the right is not evaluated and the result is `false`. For logical *or*, if the sub-expression on the left evaluates to `true`, then the sub-expression on the right is not evaluated and the result is `true`. An expression in HCL Detect Expression Language can also contain the *not* (`!`) operator, which is a unary operator that expects a sub-expression of type `Bool` on the right.

Examples:

- A simple logical expression: `3>5||2<4`
- A logical expression that uses not: `!(3>5)`

Ternary Operation

An expression in HCL Detect Expression Language can make use of the the ternary operation: `condition? choice1:choice2`. The condition of the ternary operation is expected to be of type `Bool` and the two choices are expected to be sub-expressions with compatible types. The ternary operation is lazily evaluated. If the condition evaluates to `true`, then the result is the first choice, without the sub- expression for the second choice being evaluated. If the condition evaluates to `false`, then the result is the second choice, without the sub-expression for the first choice being evaluated.

Examples:

- A simple ternary operation: `3<10?"smallerThan10":"notSmallerThan10"`

List Operations

A list is constructed by specifying a sequence of comma (`,`) separated values of the element type, surrounded by brackets (`[]`) . For instance, an example literal for `List(Double)` is `[3.5, 6.7, 8.3]` and an example for `List(String)` is `["HCL", "Detect"]`. An empty list requires casting to define its type. As an example, an empty `List(String)` can be specified as `List(String)([])`.

An expression in HCL Detect Expression Language can contain a few basic list operations: containment (`in`), non-containment (`not in`), indexing (`[i]`), slicing (`[i:j]`), concatenation (`+`), and size inquiring (`size()`).

Containment yields a Boolean answer, identifying whether an element is contained within a list. For instance, `3in[2,5,3]` yields `true`, whereas `8 in [2, 5, 3]` yields `false`. Non-containment is the negated version of the containment. For instance, `3 not in [2, 5, 3]` yields `false`, whereas `8 not in [2, 5, 3]` yields `true`.

Indexing yields the element at the specified index within the list. `0`-based indexing is used and indices are of type `Int32`. For instance, `[2, 5, 3][1]` yields `5`, and `[2, 5, 3][-3]` yields `2`. The index should be between `-size` and `size-1`, inclusive. An index that is out of these bounds will result in an evaluation error at runtime.

Slicing yields a sub-list. The start index is inclusive, whereas the end index is exclusive. If the range is out of bounds, then an empty list is returned. For instance, `[2,5,3,7][1:2]` yields `[5]`, `[2, 5, 3, 7][1:3]` yields `[5, 3]`, `[2, 5, 3, 7][1:1]` yields `[]`, `[2, 5, 3, 7][3:5]` yields `[7]`, `[2, 3, 4][-2:-1]` yields `[3]`, `[2, 3, 4][-5:-1]` yields `[2, 3]`, `[2, 3, 4][1, 6]` yields `[3, 4]`, and `[2, 5, 3, 7][4:6]` yields `[]`.

Concatenation results in a list that contains the elements from the first list followed by the elements from the second list. For instance, `[1,2]+[3]` yields `[1, 2, 3]`.

The size of a list can be retrieved via the builtin function `list.size()`.

Precedence and Associativity of Operations

Operations	Associativity
`()`	
`[]`	
unary `-`, `!`	
`*`, `/`, `%`	left
`+`, `-`	left
`>`, `>=`, `<`, `<=`	left
`==`, `!=`	left
`in`	
`&&`	left
`||`	left
`?:`	

Attributes

Expressions in HCL Detect Expression Language can also contain *attributes*. Attributes are identifiers that correspond to the attributes available in a tuple or in the master profile. Each attribute has a type and can appear in anywhere a sub-expression of that type is expected.

If an attribute comes from the master profile, it can be referenced by prefixing it with `profile.`. Otherwise, only the attribute name is used.

Examples:

- A string formed by concatenating a `String` literal and an attribute named `code` from the current tuple: `"AREA_" + code`
- A string formed by concatenating a `String` literal and a profile attribute named `name`: `"My name is " + profile.name`
- An arithmetic expression involving an attribute and `Int32` literals: `numSeconds / (24 * 60 * 60)`
- An arithmetic expression involving a profile attribute and `Int32` literals: `profile.ageInSeconds / (24 * 60 * 60)`
- A floating-point arithmetic expression, where the floating point literal is of type `Double`: `cost / 1000.0`
- Another expression involving a profile attribute, where the floating point literal is of type `Double`: `profile.revenue / 1000.0`
- A Boolean expression checking if a `String` literal is found in an attribute named `places` of type `List(String)`: `"airport" in places`
- A Boolean expression checking if a `String` literal is found in a profile attribute named `favoritePlaces` of type `List(String)`: `"airport" in profile.favoritePlaces`

Conversions

HCL Detect Expression Language supports explicit conversions via casts using a function call syntax. The name of the function is the name of the type we want to cast to.

Examples:

- Casting an `Int32` to a `String`: `String(14)` yields `"14"`
- Casting a `String` to a `Double`: `Double("4.5")` yields `4.5`
- Casting a `String` to a `Double`: `Double("nan")` yields `NaN` (not a number)

HCL Detect Expression Language supports implicit conversions (aka coercions) as well. When two integers of different types are involved in an operation, the one that has the smaller number of bits is coerced into the wider type. When an integer is involved in an operation with a `Double` it is coerced into a `Double`. Also note that, in such operations, `Int64` values that cannot be represented exactly will be rounded to the closest `Double` value.

Examples:

- Coercion with integers of different bit-lengths: `count/2` has type `Int64`, assuming `count` is of type `Int64` (this has the same semantics as the expression `count / Int64(2)`)
- Coercion involving an `Int32` and a `Double`: `timestamp / 1000` has type `Double`, assuming `timestamp` is of type `Double` (this has the same semantics as the expression `timestamp / Double(1000)`)

Aggregates

Expressions in HCL Detect Expression Language can also contain *aggregates*. Aggregates are summary statistics maintained at different temporal granularities. They are specified using their names, zero or more group by attributes (coming from the tuple being processed), period and the window unit.

The available periods are `Current`, `Last`, and `AllTime`. When the period is `Current`, the available window units are: `Day`, `Hour`, `Month` and `Year`. If the period is `Last`, the `Minute` can also be used as the window unit. The computed aggregates are always of type `Double`.

The following examples illustrate the use of aggregates as part of expressions:

- This aggregate, named `numCallsMade`, returns the number of calls made for a given number within the last hour, where `callingNumber` is an attribute available from the current tuple: `aggregate(numCallsMade[callingNumber], Last, 1, Hour)`.
- Aggregates can be involved in arithmetic operations as usual: `aggregate(numCallsMade[callingNumber],Last,1,Hour)+aggregate(numCallsMade[calledNumber], Last, 1, Hour)`
- Some aggregates may have no group by attributes: `aggregate(totalCalls,Current,Month)`
- Some aggregates may have multiple group by attributes: `aggregate(numCallsMade[callingNumber,callingCellTower],Last,1,Hour)`

Aggregates can also specify the number of most recent time units to be used for the aggregation. By default an hourly aggregate is computed from 6 10-minute aggregates, a daily aggregate is computed from 24 hourly aggregates, a monthly aggregate is computed from daily aggregates within a month, and a yearly aggregate is computed from 12 monthly aggregates. One can specify a second argument as part of the aggregate's temporal access function, which represents the number of time units to be used for the aggregation. The time units used are always the most recent ones. For instance:

- The following aggregate gets the number of calls made during the last week (7 days):
  `aggregate(numCallsMade[callingNumber],Last,7,Day)`

The aggregates with the `Current` and `AllTime` periods provide exact results:

- `aggregate(<aggregate>,Current,Hour)`: an exact aggregate value over all the activities within the current hour. E.g.: If the current time is 14:20pm, then the activities within the last 20 minutes are included.
- `aggregate(<aggregate>,Current,Day)`: an exact aggregate value over all the activities within the current day. E.g.: If the current time is 14:20pm, then the activities since midnight are included.
- `aggregate(<aggregate>,Current,Month)`: an exact aggregate value over all the activities with the current month. E.g.: If the current time is 14 May 14:20pm, then the activities since the beginning of May up to 14:20pm on May 14th are included.
- `aggregate(<aggregate>,Current,Year)`: an exact aggregate value over all the activities with the current year. E.g.: If the current time is 14 May 2017 14:20pm, then the activities since the beginning of 2017 up to 14:20pm on May 14th are included.
- `aggregate(<aggregate>,AllTime)`: an exact aggregate value over all the activities, irrespective of time.

There are 4 possible ways of computing aggregates with the `Last` period:

- **Aggregate over the last hour**: an approximate aggregate value over the activities within the last 60 minutes, that is the last 6 10-minute periods. It is an approximate value in the sense that if the current 10-minute interval is at least half past, then the aggregate is over the activities within the current 10-minute interval plus the last 5 10-minute intervals. If the current 10 minute interval is less than half past, then the aggregate is over the activities within the current 10-minute interval plus the last 6 10-minute intervals. E.g.: If the current time is 14:29pm, then the activities within the interval [13:30pm - 14:29pm] are included. If the current time is 14:21pm, then the activities within the interval [13:20pm - 14:21pm] are included.
- **Aggregate over the last day**: an approximate aggregate value over the activities within the last 24 hours. It is an approximate value in the sense that if the current hour is at least half past, then the aggregate is over the activities within the current hour plus the last 23 calendar hours. If the current hour is less than half past, then the aggregate is over the activities within the current hour plus the last 24 calendar hours. E.g.: If the current time is Tuesday 14:50pm, then the activities within the interval [Monday 15:00pm - Tuesday 14:50pm] are included. If the current time is Tuesday 14:10pm, then the activities within the interval [Monday 14:00pm - Tuesday 14:10pm] are included.
- **Aggregate over the last month**: an approximate aggregate value over the activities within the last 30 days. It is an approximate value in the sense that if the current day is at least half past, then the aggregate is over the activities within the current day plus the last 29 calendar days. If the current day is less than half past, then the aggregate is over the activities within the current day plus the last 30 calendar days. E.g.: If the current time is 14 May 22:00pm,

then the activities within the interval [15 April 00:00am - 14 May 22:00pm] are included. If the current time is 14 May 02:00am, then the activities within the interval [14 April 00:00am - 14 May 02:00am] are included.
- **Aggregate over the last year**: an approximate aggregate value over the activities within the last 12 months. It is an approximate value in the sense that if the current month is at least half past, then the aggregate is over the activities within the current month plus the last 11 calendar months. If the current month is less than half past, then the aggregate is over the activities within the current month plus the last 12 calendar months. E.g.: If the current time is 28 May 2017 14:00pm, then the activities within the interval [1 June 00:00am - 28 May 14:00pm] are included. If the current time is 2 May 14:00pm, then the activities within the interval [1 May 00:00am - 2 May 14:00pm] are included.

The same kind of approximation applies if the number of most recent time units are specified while accessing an aggregate. For instance, if the last 4 days are requested from the last month, then the current day plus the last 3 or 4 calendar days are included in the result, depending on whether the current day is at least half past or not, respectively.

These computations are done by using the following aggregate expressions:

- `aggregate(<aggregate>,Last,<windowLength>,Minute)`: this computes the aggregation by using *windowLength* minutes from the last hour.
- `aggregate(<aggregate>,Last,<windowLength>,Hour)`: if *windowLength* is greater than 1, this computes the aggregation over the last *windowLength* hours from the last day. Otherwise it computes the aggregation using the last hour.
- `aggregate(<aggregate>,Last,<windowLength>,Day)`: if *windowLength* is greater than 1, this computes the aggregation over the last *windowLength* days from the last month. Otherwise it computes the aggregation over the last day.
- `aggregate(<aggregate>,Last,<windowLength>,Month)`: if *windowLength* is greater than 1, this computes the aggregation over the last *windowLength* months from the last year. Otherwise it computes the aggregation over the last month.
- `aggregate(<aggregate>,Last,<windowLength>,Year)`: this computes the aggregation over the last year and the *windowLength* must be equal to 1.

Null Values

Attributes in HCL Detect Expression Language are nullable, that is, attributes can take null values. To denote a null value, the `null` keyword is used.

Only the following actions are legal on the expressions with null values:

- **Built-in function calls**: A nullable function parameter can take a null value. E.g.: the second parameter in the `list.indicesOf([1, 2, null, 4, 5, null], null)` call
- **Ternary operations**: The values returned from choices can be null. E.g.: `string.startsWith(profile.areaCode, "AREA_") ? profile.areaCode : null` where `areaCode` is a profile attribute of the `String` type
- **List items**: Null values can be list items. E.g.: `[null, 3, 5, 6, null]`
- **List containment check operations**: Null values can be used on the left hand side. E.g.: `null not in [null, 3, 5, 6, null]`

- **Equality check operations**: Null values can be compared for equality and non-equality. E.g.: `MSISDN == null` where `MSISDN` is a tuple attribute (Note: For comparisons with null values, the `isNull` operator can also be used. Examples: `isNull(null)` yields `trueisNull(profile.x)` yields `false` if the `x` profile attribute is not null)
- **Type conversions**: The type conversion rules are similar to the ones for non-null expressions. Some examples that are legal: `List(Int32)(null)`, `Int32(null)`, `String(Int32(null))`, some examples that are illegal: `List(Int32)(List(Int64)(null))`, `Int64(List(Int32)(null))`, `Bool(Int32(null))`, `List(Int16)(Double(null))`

Chapter 2. HCL Detect Expression Language Builtin Functions

Geohash Functions

geohash.covers

Bool geohash.covers(String geohash1, String geohash2)

Checks whether the first geohash covers the second one, where the two geohashes being equal is also considered a positive result.

Parameters:

- geohash1 - the first geohash (non-nullable).
- geohash2 - the second geohash (non-nullable).

Returns:

true if the first geohash covers the second one.

geohash.encode

String geohash.encode(Double latitude, Double longitude, Int32 level)

Encodes a geohash from latitude and longitude information.

Parameters:

- latitude - the latitude (non-nullable).
- longitude - the longitude (non-nullable).
- level - the level (non-nullable).

Returns:

the encoded geohash.

geohash.intersects

Bool geohash.intersects(String geohash1, String geohash2)

Checks whether the two geohashes intersect.

Parameters:

- geohash1 - the first geohash (non-nullable).
- geohash2 - the second geohash (non-nullable).

Returns:

true if the two geohashes intersect, false otherwise.

geohash.intersectsAny

Bool geohash.intersectsAny(List(String) geohashes1, List(String) geohashes2)

Checks whether any pair of geohashes from the two lists intersect.

Parameters:

- geohashes1 - the first geohash list (non-nullable, null elements not allowed).
- geohashes2 - the second geohash list (non-nullable, null elements not allowed).

Returns:

true if any pair of geohashes from the two lists intersect, false otherwise.

List Functions

list.average

<Numeric T> Double list.average(List(T) values)

Returns the average of the input values. If the list of values is empty, the operation yields a runtime error.

Parameters:

- values - the input values (non-nullable, null elements not allowed).

Returns:

the average value.

list.containsAll

<Primitive T> Bool list.containsAll(List(T) list, List(T) values)

Checks whether all of the values are in the input list.

Parameters:

- list - the input list (non-nullable, null elements allowed).
- values - the list of values to check (non-nullable, null elements allowed).

Returns:

true if all of the values are in the input list, false otherwise.

list.containsAny

<Primitive T> Bool list.containsAny(List(T) list, List(T) values)

Checks whether any one of the values are in the input list.

Parameters:

- list - the input list (non-nullable, null elements allowed).
- values - the list of values to check (non-nullable, null elements allowed).

Returns:

true if any one of the values are in the input list, false otherwise.

list.difference

<Primitive T> List(T) list.difference(List(T) list1, List(T) list2)

Returns the difference of the first input list from the second input list by preserving the insertion order of values in the first list and removing duplicates.

Parameters:

- list1 - the first list (non-nullable, null elements allowed).
- list2 - the second list (non-nullable, null elements allowed).

Returns:

the difference of the first list from the second one where the order of values in the first list is preserved and duplicates are removed.

list.disjoint

<Primitive T> Bool list.disjoint(List(T) list1, List(T) list2)

Checks whether the two input lists are disjoint.

Parameters:

- list1 - the first list (non-nullable, null elements allowed).
- list2 - the second list (non-nullable, null elements allowed).

Returns:

true if the input lists are disjoint, false otherwise.

list.indicesOf

<Primitive T> List(Int32) list.indicesOf(List(T) list, T value)

Finds the indices at which the value is present in the input list.

Parameters:

- list - the input list (non-nullable, null elements allowed).
- value - the value (nullable).

Returns:

the list of the indices at which the value is present in the input list.

list.intersection

<Primitive T> List(T) list.intersection(List(T) list1, List(T) list2)

Returns the intersection of the two lists by preserving the order of values in the first input list and removing duplicates.

Parameters:

- list1 - the first list (non-nullable, null elements allowed).
- list2 - the second list (non-nullable, null elements allowed).

Returns:

the intersection of the two lists where the order of values in the first list is preserved and duplicates are removed.

list.lookup

<Primitive T, Primitive R> R list.lookup(T key, List(T) key_list, List(R) value_list)

Performs a dictionary lookup of the given key in the key list then returns the matching value from the value list, produces a runtime error if the key is not found.

Parameters:

- key - the key (non-nullable).
- key_list - the key list (non-nullable, null elements not allowed).
- value_list - the value list (non-nullable, null elements allowed).

Returns:

the result of the dictionary lookup.

list.max

<Primitive T> T list.max(List(T) values)

Finds the maximum element of the input list, results in a runtime error if the list is empty.

Parameters:

- values - the input list (non-nullable, null elements not allowed).

Returns:

the maximum element of the input list.

list.min

<Primitive T> T list.min(List(T) values)

Finds the minimum element of the input list, results in a runtime error if the list is empty.

Parameters:

- values - the input list (non-nullable, null elements not allowed).

Returns:

the minimum element of the input list.

list.populationVariance

<Numeric T> Double list.populationVariance(List(T) values)

Returns the population variance of the input values. If the list of values is empty, the operation yields a runtime error.

Parameters:

- values - the input list (non-nullable, null elements not allowed).

Returns:

the population variance of the elements of the input list.

list.reverse

<Primitive T> List(T) list.reverse(List(T) list)

Returns the reverse of the input list.

Parameters:

- list - the input list (non-nullable, null elements allowed).

Returns:

the reversed list.

list.sampleVariance

<Numeric T> Double list.sampleVariance(List(T) values)

Returns the sample variance of the input values. If the length of the list of values is less than or equal to 1, the operation yields a runtime error.

Parameters:

- values - the input list (non-nullable, null elements not allowed).

Returns:

the sample variance of the elements of the input list.

list.size

<Primitive T> Int32 list.size(List(T) list)

Returns the size of the input list.

Parameters:

- list - the input list (non-nullable, null elements allowed).

Returns:

the size of the input list.

list.sort

<Primitive T> List(T) list.sort(List(T) list)

Returns the sorted version of the input list in ascending order.

Parameters:

- list - the input list (non-nullable, null elements not allowed).

Returns:

the sorted list.

list.subList

<Primitive T> List(T) list.subList(List(T) list, Int32 startPosition, Int32 endPosition)

Returns a slice of the input list. Gives a runtime error if the start or the end position is outside of its valid range.

Parameters:

- list - the input list (non-nullable, null elements allowed).
- startPosition - the start position of the slice (inclusive, in the range [-size,size]). A negative value indicates a position relative to the end of the list (non-nullable).
- endPosition - the end index of the slice (exclusive, in the range [-size,size]). A negative value indicates a position relative to the end of the list (non-nullable).

Returns:

a list representing the specified slice of the input.

list.sum

<Numeric T> Double list.sum(List(T) values)

Returns the sum of the input values. If the list of values is empty, the operation yields a runtime error.

Parameters:

- values - the input list (non-nullable, null elements not allowed).

Returns:

the sum of the input values.

list.union

<Primitive T> List(T) list.union(List(T) list1, List(T) list2)

Returns the union of the two lists by preserving the order of values in the first list and removing duplicates.

Parameters:

- list1 - the first list (non-nullable, null elements allowed).
- list2 - the second list (non-nullable, null elements allowed).

Returns:

the union of the two lists where the order of values in the first list is preserved and duplicates are removed.

Math Functions

math.ceil

Double math.ceil(Double value)

Returns the ceiling of the input value, i.e., the value of the smallest integer greater than or equal to the value.

Parameters:

- value - the input value (non-nullable).

Returns:

the ceiling of the input value.

math.floor

Double math.floor(Double value)

Returns the floor of the input value, i.e., the value of the largest integer greater than or equal to the value.

Parameters:

- value - the input value (non-nullable).

Returns:

the floor of the input value.

math.isInfinity

<Numeric T> Bool math.isInfinity(T value)

Checks whether the input value is infinity.

Parameters:

- value - the input value (non-nullable).

Returns:

true if the input value is infinity, false otherwise.

math.isNaN

<Numeric T> Bool math.isNaN(T value)

Checks whether the input value is NaN (not a number).

Parameters:

- value - the input value (non-nullable).

Returns:

true if the input value is NaN (not a number), false otherwise.

math.isNegativeInfinity

<Numeric T> Bool math.isNegativeInfinity(T value)

Checks whether the input value is negative infinity.

Parameters:

- value - the input value (non-nullable).

Returns:

true if the input value is negative infinity, false otherwise.

math.isPositiveInfinity

<Numeric T> Bool math.isPositiveInfinity(T value)

Checks whether the input value is positive infinity.

Parameters:

- value - the input value (non-nullable).

Returns:

true if the input value is positive infinity, false otherwise.

math.log

Double math.log(Double value)

Returns the natural logarithm of the input value.

Parameters:

- value - the input value (non-nullable).

Returns:

the natural logarithm of the input value.

math.max

<Primitive T> T math.max(T a, T b)

Finds the maximum of the two values.

Parameters:

- a - the first value (non-nullable).
- b - the second value (non-nullable).

Returns:

the maximum of the values.

math.min

<Primitive T> T math.min(T a, T b)

Finds the minimum of the two values.

Parameters:

- a - the first value (non-nullable).
- b - the second value (non-nullable).

Returns:

the minimum of the values.

math.pow

Double math.pow(Double base, Double exponent)

Returns the base value raised to the exponent.

Parameters:

- base - the base (non-nullable).
- exponent - the exponent (non-nullable).

Returns:

the base value raised to the exponent.

String Functions

string.indexOf

Int32 string.indexOf(String string, String substring, Int32 fromPosition)

Finds the index of the first occurrence of the substring within the input string, starting the search at a given start index. Gives a runtime error if the start or the end position is outside of its valid range.

Parameters:

- string - the input string (non-nullable).
- substring - the substring to be searched (non-nullable).
- fromPosition - the start position of the search (inclusive, in the range [-size,size]). A negative value indicates a position relative to the end of the string (non-nullable).

Returns:

the index of the substring, or -1 if not found.

string.join

<Primitive T> String string.join(List(T) values, String delimiter)

Returns a string that is the result of joining the string representations of the input list elements with the specified delimiter.

Parameters:

- values - the input list (non-nullable, null elements not allowed).
- delimiter - the string to be used as a delimiter (non-nullable).

Returns:

the joined string.

string.length

Int32 string.length(String string)

Returns the length of the input string.

Parameters:

- string - the input string (non-nullable).

Returns:

the length of the input string.

string.regexMatch

List(String) string.regexMatch(String string, String regex)

Finds all non-overlapping substrings of the input string that match the regular expression (regex).

Parameters:

- string - the input string (non-nullable).
- regex - the regex to be searched (non-nullable).

Returns:

the list of matching substrings.

string.split

List(String) string.split(String string, String separator)

Splits the list of substrings of the input string resulting from splitting it via the separator.

Parameters:

- string - the input string (non-nullable).
- separator - the separator string (non-nullable).

Returns:

the substrings.

string.startsWith

Bool string.startsWith(String input, String prefix)

Checks whether a string starts with a specified prefix.

Parameters:

- input - the string to check (non-nullable).
- prefix - the prefix (non-nullable).

Returns:

true if the string starts with the specified prefix, false otherwise.

string.substring

String string.substring(String string, Int32 startPosition, Int32 endPosition)

Creates a substring from the input string. Gives a runtime error if the start or the end position is outside of its valid range.

Parameters:

- string - the input string (non-nullable).
- startPosition - the start position of the substring (inclusive, in the range [-size,size]). A negative value indicates a position relative to the end of the string (non-nullable).
- endPosition - the end position of the substring (exclusive, in the range [-size,size]). A negative value indicates a position relative to the end of the string (non-nullable).

Returns:

the substring.

string.toLowerCase

String string.toLowerCase(String string)

Returns the lowercase representation of the string.

Parameters:

- string - the input string (non-nullable).

Returns:

the input string converted to lowercase.

string.toUpperCase

String string.toUpperCase(String string)

Returns the uppercase representation of the string.

Parameters:

- string - the input string (non-nullable).

Returns:

the input string converted to uppercase.

Time Functions

time.currentTimeInSeconds

Double time.currentTimeInSeconds()

Returns the current fractional seconds since the Epoch.

Parameters:

Returns:

the time since the Epoch.

time.daysToHours

<Integral T> Int64 time.daysToHours(T days)

Converts the time duration given in days to hours (as integer).

Parameters:

- days - the time duration given in days (non-nullable).

Returns:

the time duration expressed in terms of hours (days x hours in a day = days x 24).

time.daysToMicros

<Integral T> Int64 time.daysToMicros(T days)

Converts the time duration given in days to microseconds (as integer).

Parameters:

- days - the time duration given in days (non-nullable).

Returns:

the time duration expressed in terms of microseconds (days x (micros in a day) = days x (24 x 60 x 60 x 1000 x 1000)).

time.daysToMillis

<Integral T> Int64 time.daysToMillis(T days)

Converts the time duration given in days to milliseconds (as integer).

Parameters:

- days - the time duration given in days (non-nullable).

Returns:

the time duration expressed in terms of milliseconds (days x (millis in a day) = days x (24 x 60 x 60 x 1000)).

time.daysToMinutes

<Integral T> Int64 time.daysToMinutes(T days)

Converts the time duration given in days to minutes (as integer).

Parameters:

- days - the time duration given in days (non-nullable).

Returns:

the time duration expressed in terms of minutes (days x (minutes in a day) = days x (24 x 60)).

time.daysToNanos

<Integral T> Int64 time.daysToNanos(T days)

Converts the time duration given in days to nanoseconds (as integer).

Parameters:

- days - the time duration given in days (non-nullable).

Returns:

the time duration expressed in terms of nanoseconds (days x (nanos in a day) = days x (24 x 60 x 60 x 1000 x 1000 x 1000)).

time.daysToSeconds

<Integral T> Int64 time.daysToSeconds(T days)

Converts the time duration given in days to seconds (as integer).

Parameters:

- days - the time duration given in days (non-nullable).

Returns:

the time duration expressed in terms of seconds (days x (seconds in a day) = days x (24 x 60 x 60)).

time.formatDateTime

String time.formatDateTime(String formatString, List(Int32) timeComponents)

Returns a formatted string including the given date.

Parameters:

- formatString - format string where the following specifiers are allowed:

%A : Full name of the day of the week (e.g., Tuesday)

%B : Full name of the month of the year (e.g., March)

%H : Two-digit 24-hour clock (in the range [00:23])

%I : Two-digit 12-hour clock (in the range [00:11])

%M : Minute within the hour (two-digits, in the range [00:59])

%S : Second within the minute (two-digits, in the range [00:59])

%Y : Year in four digits (in the range [1:9999])

%a : Short name of the day of the week (e.g., Tue)

%b : Short name of the month of the year (e.g., Mar)

%d : Day of the month (two-digits, in the range [01:31])

%j : Day of the year (three-digits, in the range [001:366])

%m : Month of the year (two-digits, in the range [01:12])

%p : Morning/afternoon marker (AM or PM)

%yLast two digits of the year (in the range [00:99])

(non-nullable).

- timeComponents - the date-time components of the form: [year, month, mday, hour, minute, second, wday, yday, isdst]

year: Year as a decimal number

month: Month, in the range [1:12]

mday: Day of the month, in the range [1:31]

hour: Hours, in the range [0:23]

minute: Minutes, in the range [0:59]

second: Seconds, in the range [0:59]

wday: Day of the week, in the range [0:6], Monday is 0

yday: Day of the year, in the range [1:366]

isdst: 1 if daylight saving time is in effect, 0 otherwise (non-nullable, null elements not allowed).

Returns:

a formatted time string.

time.fractionalDaysToHours

<Numeric T> Double time.fractionalDaysToHours(T days)

Converts the time duration given in days to hours (in fractional form).

Parameters:

- days - the time duration given in days (non-nullable).

Returns:

the time duration expressed in terms of hours (days x hours in a day = days x 24).

time.fractionalDaysToMicros

<Numeric T> Double time.fractionalDaysToMicros(T days)

Converts the time duration given in days to microseconds (in fractional form).

Parameters:

- days - the time duration given in days (non-nullable).

Returns:

the time duration expressed in terms of microseconds (days x (micros in a day) = days x (24 x 60 x 60 x 1000 x 1000)).

time.fractionalDaysToMillis

<Numeric T> Double time.fractionalDaysToMillis(T days)

Converts the time duration given in days to milliseconds (in fractional form).

Parameters:

- days - the time duration given in days (non-nullable).

Returns:

the time duration expressed in terms of milliseconds (days x (millis in a day) = days x (24 x 60 x 60 x 1000)).

time.fractionalDaysToMinutes

<Numeric T> Double time.fractionalDaysToMinutes(T days)

Converts the time duration given in days to minutes (in fractional form).

Parameters:

- days - the time duration given in days (non-nullable).

Returns:

the time duration expressed in terms of minutes (days x (minutes in a day) = days x (24 x 60)).

time.fractionalDaysToNanos

<Numeric T> Double time.fractionalDaysToNanos(T days)

Converts the time duration given in days to nanoseconds (in fractional form).

Parameters:

- days - the time duration given in days (non-nullable).

Returns:

the time duration expressed in terms of nanoseconds (days x (nanos in a day) = days x (24 x 60 x 60 x 1000 x 1000 x 1000)).

time.fractionalDaysToSeconds

<Numeric T> Double time.fractionalDaysToSeconds(T days)

Converts the time duration given in days to seconds (in fractional form).

Parameters:

- days - the time duration given in days (non-nullable).

Returns:

the time duration expressed in terms of seconds (days x (seconds in a day) = days x (24 x 60 x 60)).

time.fractionalHoursToDays

<Numeric T> Double time.fractionalHoursToDays(T hours)

Converts the time duration given in hours to days (in fractional form).

Parameters:

- hours - the time duration given in hours (non-nullable).

Returns:

the time duration expressed in terms of days (one 24th of hours).

time.fractionalHoursToMicros

<Numeric T> Double time.fractionalHoursToMicros(T hours)

Converts the time duration given in hours to microseconds (in fractional form).

Parameters:

- hours - the time duration given in hours (non-nullable).

Returns:

the time duration expressed in terms of microseconds (hours x (micros in an hour) = hours x (60 x 60 x 1000 x 1000)).

time.fractionalHoursToMillis

<Numeric T> Double time.fractionalHoursToMillis(T hours)

Converts the time duration given in hours to milliseconds (in fractional form).

Parameters:

- hours - the time duration given in hours (non-nullable).

Returns:

the time duration expressed in terms of milliseconds (hours x millis in an hour = hours x (60 x 60 x 1000)).

time.fractionalHoursToMinutes

<Numeric T> Double time.fractionalHoursToMinutes(T hours)

Converts the time duration given in hours to minutes (in fractional form).

Parameters:

- hours - the time duration given in hours (non-nullable).

Returns:

the time duration expressed in terms of minutes (hours x minutes in an hour = hours x 60).

time.fractionalHoursToNanos

<Numeric T> Double time.fractionalHoursToNanos(T hours)

Converts the time duration given in hours to nanoseconds (in fractional form).

Parameters:

- hours - the time duration given in hours (non-nullable).

Returns:

the time duration expressed in terms of nanoseconds (hours x (nanos in an hour) = hours x (60 x 60 x 1000 x 1000 x 1000)).

time.fractionalHoursToSeconds

<Numeric T> Double time.fractionalHoursToSeconds(T hours)

Converts the time duration given in hours to seconds (in fractional form).

Parameters:

- hours - the time duration given in hours (non-nullable).

Returns:

the time duration expressed in terms of seconds (hours x (seconds in an hour) = hours x (60 x 60)).

time.fractionalMicrosToDays

<Numeric T> Double time.fractionalMicrosToDays(T micros)

Converts the time duration given in microseconds to days (in fractional form).

Parameters:

- micros - the time duration given in microseconds (non-nullable).

Returns:

the time duration expressed in terms of days (micros / (micros in a day) = micros / (24 x 60 x 60 x 1000 x 1000)).

time.fractionalMicrosToHours

<Numeric T> Double time.fractionalMicrosToHours(T micros)

Converts the time duration given in microseconds to hours (in fractional form).

Parameters:

- micros - the time duration given in microseconds (non-nullable).

Returns:

the time duration expressed in terms of hours (micros / (micros in an hour) = micros / (60 x 60 x 1000 x 1000)).

time.fractionalMicrosToMillis

<Numeric T> Double time.fractionalMicrosToMillis(T micros)

Converts the time duration given in microseconds to milliseconds (in fractional form).

Parameters:

- micros - the time duration given in microseconds (non-nullable).

Returns:

the time duration expressed in terms of milliseconds (micros / micros in a millisecond = micros / 1000).

time.fractionalMicrosToMinutes

<Numeric T> Double time.fractionalMicrosToMinutes(T micros)

Converts the time duration given in microseconds to minutes (in fractional form).

Parameters:

- micros - the time duration given in microseconds (non-nullable).

Returns:

the time duration expressed in terms of minutes (micros / (micros in a minute) = micros / (60 x 1000 x 1000)).

time.fractionalMicrosToNanos

<Numeric T> Double time.fractionalMicrosToNanos(T micros)

Converts the time duration given in microseconds to nanoseconds (in fractional form).

Parameters:

- micros - the time duration given in microseconds (non-nullable).

Returns:

the time duration expressed in terms of nanoseconds (micros x nanos in a microsecond = micros x 1000).

time.fractionalMicrosToSeconds

<Numeric T> Double time.fractionalMicrosToSeconds(T micros)

Converts the time duration given in microseconds to seconds (in fractional form).

Parameters:

- micros - the time duration given in microseconds (non-nullable).

Returns:

the time duration expressed in terms of seconds (micros / (micros in a second) = micros / (1000 x 1000)).

time.fractionalMillisToDays

<Numeric T> Double time.fractionalMillisToDays(T millis)

Converts the time duration given in milliseconds to days (in fractional form).

Parameters:

- millis - the time duration given in milliseconds (non-nullable).

Returns:

the time duration expressed in terms of days (millis / (millis in a day) = millis / (24 x 60 x 60 x 1000)).

time.fractionalMillisToHours

<Numeric T> Double time.fractionalMillisToHours(T millis)

Converts the time duration given in milliseconds to hours (in fractional form).

Parameters:

- millis - the time duration given in milliseconds (non-nullable).

Returns:

the time duration expressed in terms of hours (millis / (millis in an hour) = millis / (60 x 60 x 1000)).

time.fractionalMillisToMicros

<Numeric T> Double time.fractionalMillisToMicros(T millis)

Converts the time duration given in milliseconds to microseconds (in fractional form).

Parameters:

- millis - the time duration given in milliseconds (non-nullable).

Returns:

the time duration expressed in terms of microseconds (millis x micros in a millisecond = millis x 1000).

time.fractionalMillisToMinutes

<Numeric T> Double time.fractionalMillisToMinutes(T millis)

Converts the time duration given in milliseconds to minutes (in fractional form).

Parameters:

- millis - the time duration given in milliseconds (non-nullable).

Returns:

the time duration expressed in terms of minutes (millis / (millis in a minute) = millis / (60 x 1000)).

time.fractionalMillisToNanos

<Numeric T> Double time.fractionalMillisToNanos(T millis)

Converts the time duration given in milliseconds to nanoseconds (in fractional form).

Parameters:

- millis - the time duration given in milliseconds (non-nullable).

Returns:

the time duration expressed in terms of nanoseconds (millis x (nanos in a millisecond) = millis x (1000 x 1000)).

time.fractionalMillisToSeconds

<Numeric T> Double time.fractionalMillisToSeconds(T millis)

Converts the time duration given in milliseconds to seconds (in fractional form).

Parameters:

- millis - the time duration given in milliseconds (non-nullable).

Returns:

the time duration expressed in terms of seconds (millis / millis in a second = millis / 1000).

time.fractionalMinutesToDays

<Numeric T> Double time.fractionalMinutesToDays(T minutes)

Converts the time duration given in minutes to days (in fractional form).

Parameters:

- minutes - the time duration given in minutes (non-nullable).

Returns:

the time duration expressed in terms of days (minutes / (minutes in a day) = minutes / (24 x 60)).

time.fractionalMinutesToHours

<Numeric T> Double time.fractionalMinutesToHours(T minutes)

Converts the time duration given in minutes to hours (in fractional form).

Parameters:

- minutes - the time duration given in minutes (non-nullable).

Returns:

the time duration expressed in terms of hours (minutes / minutes in an hour = minutes / 60).

time.fractionalMinutesToMicros

<Numeric T> Double time.fractionalMinutesToMicros(T minutes)

Converts the time duration given in minutes to microseconds (in fractional form).

Parameters:

- minutes - the time duration given in minutes (non-nullable).

Returns:

the time duration expressed in terms of microseconds (minutes x (micros in a minute) = minutes x (60 x 1000 x 1000)).

time.fractionalMinutesToMillis

<Numeric T> Double time.fractionalMinutesToMillis(T minutes)

Converts the time duration given in minutes to milliseconds (in fractional form).

Parameters:

- minutes - the time duration given in minutes (non-nullable).

Returns:

the time duration expressed in terms of milliseconds (minutes x (millis in a minute) = minutes x (60 x 1000)).

time.fractionalMinutesToNanos

<Numeric T> Double time.fractionalMinutesToNanos(T minutes)

Converts the time duration given in minutes to nanoseconds (in fractional form).

Parameters:

- minutes - the time duration given in minutes (non-nullable).

Returns:

the time duration expressed in terms of nanoseconds (minutes x (nanos in a minute) = minutes x (60 x 1000 x 1000 x 1000)).

time.fractionalMinutesToSeconds

<Numeric T> Double time.fractionalMinutesToSeconds(T minutes)

Converts the time duration given in minutes to seconds (in fractional form).

Parameters:

- minutes - the time duration given in minutes (non-nullable).

Returns:

the time duration expressed in terms of seconds (minutes x seconds in a minute = minutes x 60).

time.fractionalNanosToDays

<Numeric T> Double time.fractionalNanosToDays(T nanos)

Converts the time duration given in nanoseconds to days (in fractional form).

Parameters:

- nanos - the time duration given in nanoseconds (non-nullable).

Returns:

the time duration expressed in terms of days (nanos / (nanos in a day) = nanos / (24 x 60 x 60 x 1000 x 1000 x 1000)).

time.fractionalNanosToHours

<Numeric T> Double time.fractionalNanosToHours(T nanos)

Converts the time duration given in nanoseconds to hours (in fractional form).

Parameters:

- nanos - the time duration given in nanoseconds (non-nullable).

Returns:

the time duration expressed in terms of hours (nanos / (nanos in an hour) = nanos / (60 x 60 x 1000 x 1000 x 1000)).

time.fractionalNanosToMicros

<Numeric T> Double time.fractionalNanosToMicros(T nanos)

Converts the time duration given in nanoseconds to microseconds (in fractional form).

Parameters:

- nanos - the time duration given in nanoseconds (non-nullable).

Returns:

the time duration expressed in terms of microseconds (nanos / nanos in a microsecond = nanos / 1000).

time.fractionalNanosToMillis

<Numeric T> Double time.fractionalNanosToMillis(T nanos)

Converts the time duration given in nanoseconds to milliseconds (in fractional form).

Parameters:

- nanos - the time duration given in nanoseconds (non-nullable).

Returns:

the time duration expressed in terms of milliseconds (nanos / (nanos in an millisecond) = nanos / (1000 x 1000)).

time.fractionalNanosToMinutes

<Numeric T> Double time.fractionalNanosToMinutes(T nanos)

Converts the time duration given in nanoseconds to minutes (in fractional form).

Parameters:

- nanos - the time duration given in nanoseconds (non-nullable).

Returns:

the time duration expressed in terms of minutes (nanos / (nanos in a minute) = nanos / (60 x 1000 x 1000 x 1000)).

time.fractionalNanosToSeconds

<Numeric T> Double time.fractionalNanosToSeconds(T nanos)

Converts the time duration given in nanoseconds to seconds (in fractional form).

Parameters:

- nanos - the time duration given in nanoseconds (non-nullable).

Returns:

the time duration expressed in terms of seconds (nanos / (nanos in a second) = nanos / (1000 x 1000 x 1000)).

time.fractionalSecondsToDays

<Numeric T> Double time.fractionalSecondsToDays(T seconds)

Converts the time duration given in seconds to days (in fractional form).

Parameters:

- seconds - the time duration given in seconds (non-nullable).

Returns:

the time duration expressed in terms of days (seconds / (seconds in a day) = seconds / (24 x 60 x 60)).

time.fractionalSecondsToHours

<Numeric T> Double time.fractionalSecondsToHours(T seconds)

Converts the time duration given in seconds to hours (in fractional form).

Parameters:

- seconds - the time duration given in seconds (non-nullable).

Returns:

the time duration expressed in terms of hours (seconds / (seconds in an hour) = seconds / (60 x 60)).

time.fractionalSecondsToMicros

<Numeric T> Double time.fractionalSecondsToMicros(T seconds)

Converts the time duration given in seconds to microseconds (in fractional form).

Parameters:

- seconds - the time duration given in seconds (non-nullable).

Returns:

the time duration expressed in terms of microseconds (seconds x (micros in a second) = seconds x (1000 x 1000)).

time.fractionalSecondsToMillis

<Numeric T> Double time.fractionalSecondsToMillis(T seconds)

Converts the time duration given in seconds to milliseconds (in fractional form).

Parameters:

- seconds - the time duration given in seconds (non-nullable).

Returns:

the time duration expressed in terms of milliseconds (seconds x millis in a second = seconds x 1000).

time.fractionalSecondsToMinutes

<Numeric T> Double time.fractionalSecondsToMinutes(T seconds)

Converts the time duration given in seconds to minutes (in fractional form).

Parameters:

- seconds - the time duration given in seconds (non-nullable).

Returns:

the time duration expressed in terms of minutes (seconds / seconds in a minute = seconds / 60).

time.fractionalSecondsToNanos

<Numeric T> Double time.fractionalSecondsToNanos(T seconds)

Converts the time duration given in seconds to nanoseconds (in fractional form).

Parameters:

- seconds - the time duration given in seconds (non-nullable).

Returns:

the time duration expressed in terms of nanoseconds (seconds x (nanos in a second) = seconds x (1000 x 1000 x 1000)).

time.hoursToDays

<Integral T> Int64 time.hoursToDays(T hours)

Converts the time duration given in hours to days (as integer).

Parameters:

- hours - the time duration given in hours (non-nullable).

Returns:

the time duration expressed in terms of days (one 24th of hours - integer division).

time.hoursToMicros

<Integral T> Int64 time.hoursToMicros(T hours)

Converts the time duration given in hours to microseconds (as integer).

Parameters:

- hours - the time duration given in hours (non-nullable).

Returns:

the time duration expressed in terms of microseconds (hours x (micros in an hour) = hours x (60 x 60 x 1000 x 1000)).

time.hoursToMillis

<Integral T> Int64 time.hoursToMillis(T hours)

Converts the time duration given in hours to milliseconds (as integer).

Parameters:

- hours - the time duration given in hours (non-nullable).

Returns:

the time duration expressed in terms of milliseconds (hours x millis in an hour = hours x (60 x 60 x 1000)).

time.hoursToMinutes

<Integral T> Int64 time.hoursToMinutes(T hours)

Converts the time duration given in hours to minutes (as integer).

Parameters:

- hours - the time duration given in hours (non-nullable).

Returns:

the time duration expressed in terms of minutes (hours x minutes in an hour = hours x 60).

time.hoursToNanos

<Integral T> Int64 time.hoursToNanos(T hours)

Converts the time duration given in hours to nanoseconds (as integer).

Parameters:

- hours - the time duration given in hours (non-nullable).

Returns:

the time duration expressed in terms of nanoseconds (hours x (nanos in an hour) = hours x (60 x 60 x 1000 x 1000 x 1000)).

time.hoursToSeconds

<Integral T> Int64 time.hoursToSeconds(T hours)

Converts the time duration given in hours to seconds (as integer).

Parameters:

- hours - the time duration given in hours (non-nullable).

Returns:

the time duration expressed in terms of seconds (hours x (seconds in an hour) = hours x (60 x 60)).

time.localDateTimeFromDate

List(Int32) time.localDateTimeFromDate(Int32 year, Int32 month, Int32 day)

Produces the local date-time components ([year, month, mday, hour, minute, second, wday, yday, isdst]) according to the given date information (year, month and day)

year: Year as a decimal number

month: Month, in the range [1:12]

mday: Day of the month, in the range [1:31]

hour: Hours, in the range [0:23]

minute: Minutes, in the range [0:59]

second: Seconds, in the range [0:59]

wday: Day of the week, in the range [0:6], Monday is 0

yday: Day of the year, in the range [1:366]

isdst: 1 if daylight saving time is in effect, 0 otherwise.

Parameters:

- year - year (in the range [1:9999]) (non-nullable).
- month - month (in the range [1:12]) (non-nullable).
- day - the day of the month (in the range [1:31]) (non-nullable).

Returns:

the local date-time components in the format: [year, month, mday, hour, minute, second, wday, yday, isdst].

time.localDateTimeFromDateTime

List(Int32) time.localDateTimeFromDateTime(Int32 year, Int32 month, Int32 day, Int32 hour, Int32 minute, Int32 second)

Produces the local date-time components ([year, month, mday, hour, minute, second, wday, yday, isdst]) according to the given date and time information (year, month, day, hour, minute and second)

year: Year as a decimal number

month: Month, in the range [1:12]

mday: Day of the month, in the range [1:31]

hour: Hours, in the range [0:23]

minute: Minutes, in the range [0:59]

second: Seconds, in the range [0:59]

wday: Day of the week, in the range [0:6], Monday is 0

yday: Day of the year, in the range [1:366]

isdst: 1 if daylight saving time is in effect, 0 otherwise.

Parameters:

- year - the year (in the range [1:9999]) (non-nullable).
- month - the month within the year (in the range [1:12]) (non-nullable).
- day - the day within the month (in the range [1:31]) (non-nullable).
- hour - the hour within the day (in the range [0:23]) (non-nullable).

- minute - the minute within the hour (in the range [0:59]) (non-nullable).
- second - the second within the minute (in the range [0:59]) (non-nullable).

Returns:

the local date-time components in the format: [year, month, mday, hour, minute, second, wday, yday, isdst].

time.localTime

List(Int32) time.localTime(Double value)

Converts the time in seconds since the Epoch to the local date-time components in the format: [year, month, mday, hour, minute, second, wday, yday, isdst]

year: Year as a decimal

month: Month, in the range [1:12]

mday: Day of the month, in the range [1:31]

hour: Hours, in the range [0:23]

minute: Minutes, in the range [0:59]

second: Seconds, in the range [0:59]

wday: Day of the week, in the range [0:6], Monday is 0

yday: Day of the year, in the range [1:366]

isdst: 1 if daylight saving time is in effect, 0 otherwise.

Parameters:

- value - the time (fractional seconds since the Epoch) (non-nullable).

Returns:

the local date-time components in the format: [year, month, mday, hour, minute, second, wday, yday, isdst].

time.microsToDays

<Integral T> Int64 time.microsToDays(T micros)

Converts the time duration given in microseconds to days (as integer).

Parameters:

- micros - the time duration given in microseconds (non-nullable).

Returns:

the time duration expressed in terms of days (micros / (micros in a day) = micros / (24 x 60 x 60 x 1000 x 1000) - integer division).

time.microsToHours

<Integral T> Int64 time.microsToHours(T micros)

Converts the time duration given in microseconds to hours (as integer).

Parameters:

- micros - the time duration given in microseconds (non-nullable).

Returns:

the time duration expressed in terms of hours (micros / (micros in an hour) = micros / (60 x 60 x 1000 x 1000) - integer division).

time.microsToMillis

<Integral T> Int64 time.microsToMillis(T micros)

Converts the time duration given in microseconds to milliseconds (as integer).

Parameters:

- micros - the time duration given in microseconds (non-nullable).

Returns:

the time duration expressed in terms of milliseconds (micros / micros in a millisecond = micros / 1000 - integer division).

time.microsToMinutes

<Integral T> Int64 time.microsToMinutes(T micros)

Converts the time duration given in microseconds to minutes (as integer).

Parameters:

- micros - the time duration given in microseconds (non-nullable).

Returns:

the time duration expressed in terms of minutes (micros / (micros in a minute) = micros / (60 x 1000 x 1000) - integer division).

time.microsToNanos

<Integral T> Int64 time.microsToNanos(T micros)

Converts the time duration given in microseconds to nanoseconds (as integer).

Parameters:

- micros - the time duration given in microseconds (non-nullable).

Returns:

the time duration expressed in terms of nanoseconds (micros x nanos in a microsecond = micros x 1000).

time.microsToSeconds

<Integral T> Int64 time.microsToSeconds(T micros)

Converts the time duration given in microseconds to seconds (as integer).

Parameters:

- micros - the time duration given in microseconds (non-nullable).

Returns:

the time duration expressed in terms of seconds (micros / (micros in a second) = micros / (1000 x 1000) - integer division).

time.millisToDays

<Integral T> Int64 time.millisToDays(T millis)

Converts the time duration given in milliseconds to days (as integer).

Parameters:

- millis - the time duration given in milliseconds (non-nullable).

Returns:

the time duration expressed in terms of days (millis / (millis in a day) = millis / (24 x 60 x 60 x 1000) - integer division).

time.millisToHours

<Integral T> Int64 time.millisToHours(T millis)

Converts the time duration given in milliseconds to hours (as integer).

Parameters:

- millis - the time duration given in milliseconds (non-nullable).

Returns:

the time duration expressed in terms of hours (millis / (millis in an hour) = millis / (60 x 60 x 1000) - integer division).

time.millisToMicros

<Integral T> Int64 time.millisToMicros(T millis)

Converts the time duration given in milliseconds to microseconds (as integer).

Parameters:

- millis - the time duration given in milliseconds (non-nullable).

Returns:

the time duration expressed in terms of microseconds (millis x micros in a millisecond = millis x 1000).

time.millisToMinutes

<Integral T> Int64 time.millisToMinutes(T millis)

Converts the time duration given in milliseconds to minutes (as integer).

Parameters:

- millis - the time duration given in milliseconds (non-nullable).

Returns:

the time duration expressed in terms of minutes (millis / (millis in a minute) = millis / (60 x 1000) - integer division).

time.millisToNanos

<Integral T> Int64 time.millisToNanos(T millis)

Converts the time duration given in milliseconds to nanoseconds (as integer).

Parameters:

- millis - the time duration given in milliseconds (non-nullable).

Returns:

the time duration expressed in terms of nanoseconds (millis x (nanos in a millisecond) = millis x (1000 x 1000)).

time.millisToSeconds

<Integral T> Int64 time.millisToSeconds(T millis)

Converts the time duration given in milliseconds to seconds (as integer).

Parameters:

- millis - the time duration given in milliseconds (non-nullable).

Returns:

the time duration expressed in terms of seconds (millis / millis in a second = millis / 1000 - integer division).

time.minutesToDays

<Integral T> Int64 time.minutesToDays(T minutes)

Converts the time duration given in minutes to days (as integer).

Parameters:

- minutes - the time duration given in minutes (non-nullable).

Returns:

the time duration expressed in terms of days (minutes / (minutes in a day) = minutes / (24 x 60) - integer division).

time.minutesToHours

<Integral T> Int64 time.minutesToHours(T minutes)

Converts the time duration given in minutes to hours (as integer).

Parameters:

- minutes - the time duration given in minutes (non-nullable).

Returns:

the time duration expressed in terms of hours (minutes / minutes in an hour = minutes / 60 - integer division).

time.minutesToMicros

<Integral T> Int64 time.minutesToMicros(T minutes)

Converts the time duration given in minutes to microseconds (as integer).

Parameters:

- minutes - the time duration given in minutes (non-nullable).

Returns:

the time duration expressed in terms of microseconds (minutes x (micros in a minute) = minutes x (60 x 1000 x 1000)).

time.minutesToMillis

<Integral T> Int64 time.minutesToMillis(T minutes)

Converts the time duration given in minutes to milliseconds (as integer).

Parameters:

- minutes - the time duration given in minutes (non-nullable).

Returns:

the time duration expressed in terms of milliseconds (minutes x (millis in a minute) = minutes x (60 x 1000)).

time.minutesToNanos

<Integral T> Int64 time.minutesToNanos(T minutes)

Converts the time duration given in minutes to nanoseconds (as integer).

Parameters:

- minutes - the time duration given in minutes (non-nullable).

Returns:

the time duration expressed in terms of nanoseconds (minutes x (nanos in a minute) = minutes x (60 x 1000 x 1000 x 1000)).

time.minutesToSeconds

<Integral T> Int64 time.minutesToSeconds(T minutes)

Converts the time duration given in minutes to seconds (as integer).

Parameters:

- minutes - the time duration given in minutes (non-nullable).

Returns:

the time duration expressed in terms of seconds (minutes x seconds in a minute = minutes x 60).

time.nanosToDays

<Integral T> Int64 time.nanosToDays(T nanos)

Converts the time duration given in nanoseconds to days (as integer).

Parameters:

- nanos - the time duration given in nanoseconds (non-nullable).

Returns:

the time duration expressed in terms of days (nanos / (nanos in a day) = nanos / (24 x 60 x 60 x 1000 x 1000 x 1000) - integer division).

time.nanosToHours

<Integral T> Int64 time.nanosToHours(T nanos)

Converts the time duration given in nanoseconds to hours (as integer).

Parameters:

- nanos - the time duration given in nanoseconds (non-nullable).

Returns:

the time duration expressed in terms of hours (nanos / (nanos in an hour) = nanos / (60 x 60 x 1000 x 1000 x 1000) - integer division).

time.nanosToMicros

<Integral T> Int64 time.nanosToMicros(T nanos)

Converts the time duration given in nanoseconds to microseconds (as integer).

Parameters:

- nanos - the time duration given in nanoseconds (non-nullable).

Returns:

the time duration expressed in terms of microseconds (nanos / nanos in a microsecond = nanos / 1000 - integer division).

time.nanosToMillis

<Integral T> Int64 time.nanosToMillis(T nanos)

Converts the time duration given in nanoseconds to milliseconds (as integer).

Parameters:

- nanos - the time duration given in nanoseconds (non-nullable).

Returns:

the time duration expressed in terms of milliseconds (nanos / (nanos in an millisecond) = nanos / (1000 x 1000) - integer division).

time.nanosToMinutes

<Integral T> Int64 time.nanosToMinutes(T nanos)

Converts the time duration given in nanoseconds to minutes (as integer).

Parameters:

- nanos - the time duration given in nanoseconds (non-nullable).

Returns:

the time duration expressed in terms of minutes (nanos / (nanos in a minute) = nanos / (60 x 1000 x 1000 x 1000) - integer division).

time.nanosToSeconds

<Integral T> Int64 time.nanosToSeconds(T nanos)

Converts the time duration given in nanoseconds to seconds (as integer).

Parameters:

- nanos - the time duration given in nanoseconds (non-nullable).

Returns:

the time duration expressed in terms of seconds (nanos / (nanos in a second) = nanos / (1000 x 1000 x 1000) - integer division).

time.parseLocalDateTime

List(Int32) time.parseLocalDateTime(String dateTimeString, String formatString)

Produces the date-time components of a local date-time string according to the given format.

Parameters:

- dateTimeString - a local date-time string (non-nullable).
- formatString - a format string where the following specifiers are allowed:

%A : Full name of the day of the week (e.g., Tuesday)

%B : Full name of the month of the year (e.g., March)

%H : Two-digit 24-hour clock (in the range [00:23])

%I : Two-digit 12-hour clock (in the range [00:11])

%M : Minute within the hour (two-digits, in the range [00:59])

%S : Second within the minute (two-digits, in the range [00:59])

%Y : Year in four digits (in the range [1:9999])

%a : Short name of the day of the week (e.g., Tue)

%b : Short name of the month of the year (e.g., Mar)

%d : Day of the month (two-digits, in the range [01:31])

%j : Day of the year (three-digits, in the range [001:366])

%m : Month of the year (two-digits, in the range [01:12])

%p : Morning/afternoon marker (AM or PM)

%yLast two digits of the year (in the range [00:99])

(non-nullable).

Returns:

the date-time components in the format: [year, month, mday, hour, minute, second, wday, yday, isdst]

year: Year as a decimal

month: Month, in the range [1:12]

mday: Day of the month, in the range [1:31]

hour: Hours, in the range [0:23]

minute: Minutes, in the range [0:59]

second: Seconds, in the range [0:59]

wday: Day of the week, in the range [0:6], Monday is 0

yday: Day of the year, in the range [1:366]

isdst: 1 if daylight saving time is in effect, 0 otherwise.

time.parseUtcDateTime

List(Int32) time.parseUtcDateTime(String dateTimeString, String formatString)

Produces the date-time components of a UTC date-time string according to the given format.

Parameters:

- dateTimeString - a UTC date-time string (non-nullable).
- formatString - a format string where the following specifiers are allowed:

%A : Full name of the day of the week (e.g., Tuesday)

%B : Full name of the month of the year (e.g., March)

%H : Two-digit 24-hour clock (in the range [00:23])

%I : Two-digit 12-hour clock (in the range [00:11])

%M : Minute within the hour (two-digits, in the range [00:59])

%S : Second within the minute (two-digits, in the range [00:59])

%Y : Year in four digits (in the range [1:9999])

%a : Short name of the day of the week (e.g., Tue)

%b : Short name of the month of the year (e.g., Mar)

%d : Day of the month (two-digits, in the range [01:31])

%j : Day of the year (three-digits, in the range [001:366])

%m : Month of the year (two-digits, in the range [01:12])

%p : Morning/afternoon marker (AM or PM)

%yLast two digits of the year (in the range [00:99])

(non-nullable).

Returns:

the date-time components in the format: [year, month, mday, hour, minute, second, wday, yday, isdst]

year: Year as a decimal

month: Month, in the range [1:12]

mday: Day of the month, in the range [1:31]

hour: Hours, in the range [0:23]

minute: Minutes, in the range [0:59]

second: Seconds, in the range [0:59]

wday: Day of the week, in the range [0:6], Monday is 0

yday: Day of the year, in the range [1:366]

isdst: 1 if daylight saving time is in effect, 0 otherwise.

time.secondsToDays

<Integral T> Int64 time.secondsToDays(T seconds)

Converts the time duration given in seconds to days (as integer).

Parameters:

- seconds - the time duration given in seconds (non-nullable).

Returns:

the time duration expressed in terms of days (seconds / (seconds in a day) = seconds / (24 x 60 x 60) - integer division).

time.secondsToHours

<Integral T> Int64 time.secondsToHours(T seconds)

Converts the time duration given in seconds to hours (as integer).

Parameters:

- seconds - the time duration given in seconds (non-nullable).

Returns:

the time duration expressed in terms of hours (seconds / (seconds in an hour) = seconds / (60 x 60) - integer division).

time.secondsToMicros

<Integral T> Int64 time.secondsToMicros(T seconds)

Converts the time duration given in seconds to microseconds (as integer).

Parameters:

- seconds - the time duration given in seconds (non-nullable).

Returns:

the time duration expressed in terms of microseconds (seconds x (micros in a second) = seconds x (1000 x 1000)).

time.secondsToMillis

<Integral T> Int64 time.secondsToMillis(T seconds)

Converts the time duration given in seconds to milliseconds (as integer).

Parameters:

- seconds - the time duration given in seconds (non-nullable).

Returns:

the time duration expressed in terms of milliseconds (seconds x millis in a second = seconds x 1000).

time.secondsToMinutes

<Integral T> Int64 time.secondsToMinutes(T seconds)

Converts the time duration given in seconds to minutes (as integer).

Parameters:

- seconds - the time duration given in seconds (non-nullable).

Returns:

the time duration expressed in terms of minutes (seconds / seconds in a minute = seconds / 60 - integer division).

time.secondsToNanos

<Integral T> Int64 time.secondsToNanos(T seconds)

Converts the time duration given in seconds to nanoseconds (as integer).

Parameters:

- seconds - the time duration given in seconds (non-nullable).

Returns:

the time duration expressed in terms of nanoseconds (seconds x (nanos in a second) = seconds x (1000 x 1000 x 1000)).

time.utcDateTimeFromDate

List(Int32) time.utcDateTimeFromDate(Int32 year, Int32 month, Int32 day)

Produces the UTC date-time components ([year, month, mday, hour, minute, second, wday, yday, isdst]) according to the given date information (year, month and day)

year: Year as a decimal number

month: Month, in the range [1:12]

mday: Day of the month, in the range [1:31]

hour: Hours, in the range [0:23]

minute: Minutes, in the range [0:59]

second: Seconds, in the range [0:59]

wday: Day of the week, in the range [0:6], Monday is 0

yday: Day of the year, in the range [1:366]

isdst: 1 if daylight saving time is in effect, 0 otherwise.

Parameters:

- year - year (in the range [1:9999]) (non-nullable).
- month - month (in the range [1:12]) (non-nullable).
- day - the day of the month (in the range [1:31]) (non-nullable).

Returns:

the UTC date-time components in the format: [year, month, mday, hour, minute, second, wday, yday, isdst].

time.utcDateTimeFromDateTime

List(Int32) time.utcDateTimeFromDateTime(Int32 year, Int32 month, Int32 day, Int32 hour, Int32 minute, Int32 second)

Produces the UTC date-time components ([year, month, mday, hour, minute, second, wday, yday, isdst]) according to the given date and time information (year, month, day, hour, minute and second)

year: Year as a decimal number

month: Month, in the range [1:12]

mday: Day of the month, in the range [1:31]

hour: Hours, in the range [0:23]

minute: Minutes, in the range [0:59]

second: Seconds, in the range [0:59]

wday: Day of the week, in the range [0:6], Monday is 0

yday: Day of the year, in the range [1:366]

isdst: 1 if daylight saving time is in effect, 0 otherwise.

Parameters:

- year - the year (in the range [1:9999]) (non-nullable).
- month - the month within the year (in the range [1:12]) (non-nullable).
- day - the day within the month (in the range [1:31]) (non-nullable).
- hour - the hour within the day (in the range [0:23]) (non-nullable).
- minute - the minute within the hour (in the range [0:59]) (non-nullable).
- second - the second within the minute (in the range [0:59]) (non-nullable).

Returns:

the UTC date-time components in the format: [year, month, mday, hour, minute, second, wday, yday, isdst].

time.utcTime

List(Int32) time.utcTime(Double value)

Converts the time in seconds since the Epoch to the UTC date-time components in the format: [year, month, mday, hour, minute, second, wday, yday, isdst]

year: Year as a decimal number

month: Month, in the range [1:12]

mday: Day of the month, in the range [1:31]

hour: Hours, in the range [0:23]

minute: Minutes, in the range [0:59]

second: Seconds, in the range [0:59]

wday: Day of the week, in the range [0:6], Monday is 0

yday: Day of the year, in the range [1:366]

isdst: 1 if daylight saving time is in effect, 0 otherwise.

Parameters:

- value - the time (fractional seconds since the Epoch) (non-nullable).

Returns:

the UTC date-time components in the format: [year, month, mday, hour, minute, second, wday, yday, isdst].

Generic Type Classes

Integral

- Int16
- Int32
- Int64

Numeric

- Double
- Int16
- Int32
- Int64

Primitive

- Bool
- Double
- Int16
- Int32
- Int64
- String

Temporal

- DateTime