Microsoft Power Query Preview For Excel

Formula Library Specification

February, 2015

1. Number

The following functions relate to Number values.

1.1 Constants

1.1.1 Number.NaN

Represents `0/0`.

1.1.2 Number.NegativeInfinity

Represents -1/0.

1.1.3 Number.PositiveInfinity

Represents 1/0.

1.1.4 Number.Epsilon

Returns the smallest possible number.

1.1.5 Number.E

Returns `2.7182818284590451`, the value of e up to 16 decimal digits.

1.1.6 Number.PI

Returns `3.1415926535897931`, the value for Pi up to 16 decimal digits.

1.2 Information

1.2.1 Number.IsNaN

Returns true if a value is Number.NaN.

Number.IsNaN(value as number) as logical

Arguments

Argument	Description
value	Value to evaluate.

Examples

```
Number.IsNaN(1) equals false
Number.IsNaN(0/0) equals true
```

1.2.2 Number.IsEven

Returns true if a value is an even number.

Number.IsEven(value as number) as logical

Arguments

Argument	Description
value	Value to evaluate.

Examples

```
Number.IsEven(3) equals false
Number.IsEven(4) equals true
```

1.2.3 Number.IsOdd

Returns true if a value is an odd number.

Number.IsOdd(value as number) as logical

Arguments

Argument	Description
value	Value to evaluate.

Examples

```
Number.IsOdd(3) equals true Number.IsOdd(4)
equals false
```

1.3 Conversion and formatting

1.3.1 Number.FromText

Returns a number value from a text value.

Number.FromText(text as nullable text, optional culture as nullable text) as nullable number

Arguments

Argument	Description
text	Text value to convert to a number. If **text** does not represent a valid number, **Expression.Error** is thrown.
optional culture	A text value corresponding to the culture values supported on your version of Windows, such as "en-US". If the culture is not specified, the current user culture is used. For a list of culture names, see National Language Support (NLS) API Reference.

Examples

```
Number.FromText("1") equals 1
Number.FromText("a") equals error
```

1.3.2 Number.ToText

Returns a text value from a number value.

Number.ToText(number as number, optional format as nullable text, optional culture as nullable text) as nullable text

Arguments

Argument	Description
number	Number value to convert to text.
optional format	An optional text value used to format common numeric values.
optional culture	A text value corresponding to the culture values supported on your version of Windows, such as "en-US". If the culture is not specified, the current user culture is used. For a list of culture names, see National Language Support (NLS) API Reference.

Format Settings

Setting	Name	Description

D or d	Decimal	Formats the result as integer digits with an optional negative sign. The precision setting controls the number of digits in the output.
E or e	Exponential (scientific)	Exponential notation. The precision setting controls the maximum number of decimal digits (default is 6).
F or f	Fixed-point	Integral and decimal digits with optional negative sign.
G or g	General	Most compact form of either fixed-point or scientific.
N or n	Number	Integral and decimal digits, group separators, and a decimal separator with optional negative sign.
P or p	Percent	Number multiplied by 100 and displayed with a percent symbol.
R or r	Round-trip	Round-trip an identical number. The precision setting is ignored.
X or x	Hexadecimal	A hexadecimal text value.
Any other single character	Unknown setting	Throws **Expression.Error** error

1.3.3 Number.From

Returns a number value from a value.

Number.From(value as any, optional culture as nullable text) as nullable number

Arguments

Argument	Description
value	Value to convert.
optional culture	A text value corresponding to the culture values supported on your version of Windows, such as `"en-US"`. If the culture is not specified, the current user culture is used. For a list of culture names, see National Language Support (NLS) API Reference.

Values of the following types can be converted to a number value:

Type	Converts to

text	Common text formats are handled ("15", "3,423.10", "5.0E-10"). See Number.FromText for details.
logical	1 for true, 0 for false.
datetime	A double-precision floating-point number that contains an OLE Automation date equivalent.
datetimezone	A double-precision floating-point number that contains an OLE Automation date equivalent of the local date and time of value.
date	A double-precision floating-point number that contains an OLE Automation date equivalent.
time	Expressed in fractional days.
duration	Expressed in whole and fractional days.

Remarks

- If a value is null, Number.From returns null.
- If a value is a number, the same value is returned. If value is of any other type, an error is returned.

Examples

```
Number.From("4") equals 4
Number.From(#datetime(2020, 3, 20, 6, 0, 0)) equals 43910.25
```

1.3.4 Int64.From

Returns a 64-bit integer number value from a value.

In64.From(value as any, optional culture as nullable text, optional roundingMode as nullable number) as nullable number

Arguments

Argument	Description
value	Value to convert.

optional culture	A text value corresponding to the culture values supported on your version of Windows, such as "en-US". If the culture is not specified, the current user culture is used. For a list of culture names, see National Language Support (NLS) API Reference.
optional roundingMode	Specifies rounding direction when there is a tie between the possible numbers to round to.

Remarks

- If a value is null, Int64.From returns null.
- If a value is a number within range of the 64-bit integer without a fractional part, the value is returned.
- If value is of any other type or out of the range, an error is returned. See Number.FromText for converting it to a number value, then convert from a number to a 64-bit integer.
- See Number.Round for the available rounding modes, the default is RoundingMode.ToEven.

Examples

```
Int64.From("4") equals 4
Int64.From("4.5", null, RoundingMode.AwayFromZero)
```

1.3.5 Currency.From

Returns a currency value from the given value.

Currency.From(value as any, optional culture as nullable text, optional roundingMode as nullable number) as nullable number

Arguments

Argument	Description
value	Value to convert.
optional culture	A text value corresponding to the culture values supported on your version of Windows, such as "en-US". If the culture is not specified, the current user culture is used. For a list of culture names, see National Language Support (NLS) API Reference.
optional roundingMode	Specifies rounding direction when there is a tie between the possible numbers to round to.

Remarks

- If a value is null, Currency.From returns null.

- If a value is a number within range of currency, the fractional part of the value is rounded to 4 decimal digits and returned. The valid range for currency is -922,337,203,685,477.5808 to 922,337,203,685,477.5807
- If value is of any other type or out of the range, an error is returned. See Number.FromText for converting it to a number value, then convert from a number to a 64-bit integer.
- See Number.Round for the available rounding modes, the default is RoundingMode.ToEven.

Examples

```
Currency.From("1.23455") equals 1.2346
Currency.From("1.23455", "en-Us", RoundingMode.Down) equals 1.2345
```

1.4 Rounding

1.4.1 Number.RoundDown

Returns the largest integer less than or equal to a number value.

Number.RoundDown(value as nullable number) as nullable number

Arguments

Argument	Description
value	Value to round down.

Examples

```
Number.RoundDown(-1.2) equals -2
Number.RoundDown(1.2) equals 1
```

1.4.2 Number.RoundUp

Returns the larger integer greater than or equal to a number value.

Number.RoundUp(value as nullable number) as nullable number

Arguments

Argument	Description
value	Value to round up.

Examples

```
Number.RoundUp(-1.2) equals -1  Number.RoundUp(1.2)
equals 2
```

1.4.3 Number.RoundTowardZero

Returns `Number.RoundDown(x)` when `x >= 0` and `Number.RoundUp(x)` when `x < 0`.

Number.RoundTowardZero(value as nullable number) as nullable number

Arguments

Argument	Description
value	Value to round toward zero.

Examples

```
Number.RoundTowardZero(-1.2) equals -1  Number.RoundTowardZero(1.2)
equals 1
```

1.4.4 Number.RoundAwayFromZero

Returns Number.RoundUp(value) when value >= 0 and Number.RoundDown(value) when value < 0.

Number.RoundAwayFromZero(value as nullable number) as nullable number

Arguments

Argument	Description
value	Value to round away from zero.

Examples

```
Number.RoundAwayFromZero(-1.2) equals -2  Number.RoundAwayFromZero(1.2)
equals 2
```

1.4.5 Number.Round

Returns a nullable number (n) if value is an integer.

Number.Round(value as nullable number, digits as nullable number, roundingMode as nullable number) as nullable number

Arguments

Argument	Description

value	Integer value to round.
digits	Fractional part is rounded by digits.
roundingMode	Specifies rounding direction when there is a tie between the possible numbers to round to. For example, when the last digit of the number being rounded is 5 such as. 1.5 or 2.345.

Settings

Rounding mode	Description
RoundingMode.Up = 0	Adds 5e-n to the number being rounded, where n is the number of fractional digits in the number.
RoundingMode.Down = 1	Subtracts 5e-n from the number being rounded, where n is the number of fractional digits in the number.
RoundingMode.AwayFromZero = 2	The same as **RoundingMode.Up** when the number being rounded is positive; otherwise, the same as **RoundingMode.Down**.
RoundingMode.TowardZero = 3	The same as **RoundingMode.Down** when the number being rounded is positive; otherwise, the same as **RoundingMode.Up**.
RoundingMode.ToEven = 4	Applies RoundingMode.Up or RoundingMode.Down to round the last digit to even.

Remarks

- If **value** >= 0, returns n with the fractional part rounded by **digits** using **roundingMode.**
- if **value** < 0, it returns the integral part of n rounded to m-n decimal digits, using **roundingMode**, where m is the number of digits of n.
- If **roundingMode** is not specified, **`RoundingMode.ToEven`** is used.

Examples

```
Number.Round(-1.249, 2) equals -1.25
Number.Round(-1.245, 2) equals -1.24
Number.Round(1.245, 2, RoundingMode.Up) equals 1.25
Number.Round(1.245, 2, RoundingMode.Down) equals 1.24
Number.Round(1.245, 2, RoundingMode.AwayFromZero) equals 1.25
Number.Round(1.245, 2, RoundingMode.TowardZero) equals 1.24
Number.Round(1.245, 2, RoundingMode.ToEven) equals 1.24
Number.Round(-1.245, 2, RoundingMode.ToEven) equals -1.24
```

1.5 Operations

1.5.1 Number.Abs

Returns the absolute value of a number.

Number.Abs(number as nullable number) as nullable number

Arguments

Argument	Description
number	Number to evaluate.

Example

```
Number.Abs(-1) equals 1
```

1.5.2 Number.Sign

Returns 1 for positive numbers, -1 for negative numbers or 0 for zero.

Number.Sign(number as nullable number) as nullable number

Arguments

Argument	Description
number	Number to evaluate.

Examples

```
Number.Sign(-1) equals -1  Number.Sign(1)
equals 1
```

1.5.3 Number.IntegerDivide

Divides two numbers and returns the whole part of the resulting number.

Number.IntegerDivide (number1 as nullable number, number2 as nullable number, optional precision as nullable number) as nullable number

Arguments

Argument	Description
number1	Dividend or numerator.
number2	Divisor or denominator.
optional precision	Precision of the result.

Example

```
Number.IntegerDivide(9.2, 3.1) equals 2
```

1.5.4 Number.Mod

Divides two numbers and returns the remainder of the resulting number.

Number.Mod(number as nullable number, divisor as nullable number) as nullable number

Arguments

Argument	Description
number	Dividend or numerator.
divisor	Divisor or denominator.

Example

```
Number.Mod(83, 9) equals 2
```

1.5.5 Number.Power

Returns a number raised by a power.

Number.Power(number as nullable number, power as nullable number) as nullable number

Arguments

Argument	Description
number	Number to raise.

power	Power to raise by.

Example

```
Number.Power(9, 3) equals 729
```

1.5.6 Number.Sqrt

Returns the square root of a number.

Number.Sqrt(number as nullable number) as nullable number

Arguments

Argument	Description
number	The number to calculate.

Example

```
Number.Sqrt(16) equals 4
```

1.5.7 Number.Exp

Returns a number representing *e* raised to a power.

Number.Exp(number as nullable number) as nullable number

Arguments

Argument	Description
number	The number to calculate.

Examples

```
Number.Exp(0) equals 1
Number.Exp(3) equals 20.085536923187668
```

1.5.8 Number.Ln

Returns the natural logarithm of a number.

Number.Ln(number as nullable number) as nullable number

Arguments

Argument	Description
number	The number to calculate.

Examples

```
Number.Ln(1) equals 0
```

1.5.9 Number.Log

Returns the logarithm of a number to the **base**.

Number.E Number.Log(number as nullable number, base as nullable number) as nullable number

Arguments

Argument	Description
number	The number to calculate.
base	The number to which the logarithm is applied.

Remarks

If the value of the base is not specified, returns **Number.E**.

Examples

```
Number.Log(1) equals 0
Number.Log(5,5) equals 1
```

1.5.10 Number.Log10

Returns the base-10 logarithm of a number.

Number.Log10 (number as nullable number) as nullable number

Arguments

Argument	Description
number	The number to calculate.

Example

```
Number.Log10(10) equals  1
```

1.5.11 Number.Factorial

Returns the factorial of a number.

Number.Factorial(number as nullable number) as nullable number

Arguments

Argument	Description
number	The number to calculate.

Examples

```
Number.Factorial(3) equals 6
```

1.5.12 Number.Combinations

Returns the number of combinations of a given number of items for the optional combination size.

Number.Combinations
(setSize as nullable number, combinationSize as nullable number) as nullable number

Arguments

Argument	Description
setSize	Number of combination items.
combinationSize	Size of combinations.

1.5.13 Number.Permutations

Returns the number of total permutatons of a given number of items for the optional permutation size.

Number.Permutations(setSize as nullable number, permutationSize as nullable number) as nullable number

Arguments

Argument	Description
setSize	Number of permutation items.

permutationSize	Size of permutations.

1.6 Random

1.6.1 Number.Random

Returns a random fractional number between 0 and 1.

Number.Random() as number

1.6.2 Number.RandomBetween

Returns a random number between the two given number values.

Number.RandomBetween(bottom as number, top as number) as number

Arguments

Argument	Description
bottom	The lowest number in the range.
top	The highest number in the range.

1.7 Trigonometry

1.7.1 Number.Cos

Returns the cosine of a number.

Number.Cos (angle as nullable number) as nullable number

Arguments

Argument	Description
angle	The angle to calculate.

Example `Number.Cos(0)`

`equals 1`

1.7.2 Number.Sin

Returns the sine of a number.

Number.Sin (angle as nullable number) as nullable number

Arguments

Argument	Description
angle	The angle to calculate.

Example `Number.Sin(0)`

`equals 0`

1.7.3 Number.Tan

Returns the tangent of a number.

Number.Tan (angle as nullable number) as nullable number

Arguments

Argument	Description
angle	The angle to calculate.

Example

`Number.Tan(1) equals 1.557`

1.7.4 Number.Acos

Returns the arccosine of a number.

Number.Acos(angle as nullable number) as nullable number

Arguments

Argument	Description
angle	The angle to calculate.

1.7.5 Number.Asin

Returns the arcsine of a number.

Number.Asin(angle as nullable number) as nullable number

Arguments

Argument	Description
angle	The angle to calculate.

1.7.6 Number.Atan

Returns the arctangent of a number.

Number.Atan(angle as nullable number) as nullable number

Arguments

Argument	Description
angle	The angle to calculate.

1.7.7 Number.Atan2

Returns the arctangent of the division of two numbers.

Number.Atan2(angle as nullable number) as nullable number

Arguments

Argument	Description
angle	The angle to calculate.

1.7.8 Number.Cosh

Returns the hyperbolic cosine of a number.

Number.Cosh(angle as nullable number) as nullable number

Arguments

Argument	Description
angle	The angle to calculate.

1.7.9 Number.Sinh

Returns the hyperbolic sine of a number.

Number.Sinh(angle as nullable number) as nullable number

Arguments

Argument	Description
angle	The angle to calculate.

1.7.10 Number.Tanh

Returns the hyperbolic tangent of a number.

Number.Tanh(angle as nullable number) as nullable number

Arguments

Argument	Description
angle	The angle to calculate.

1.8 Bytes

1.8.1 Number.BitwiseAnd

Returns the result of a bitwise AND operation on the provided operands.

Number.BitwiseAnd (number1 as number, number2 as number) as number

Arguments

Argument	Description
number1	First operand.

number2	Second operand.

1.8.2 Number.BitwiseNot

Returns the result of a bitwise NOT operation on the provided operands.

Number.BitwiseNot(x as number, y as number) as number

Arguments

Argument	Description
x	The number to shift.
y	The number of bits to shift.

1.8.3 Number.BitwiseOr

Returns the result of a bitwise OR operation on the provided operands.

Number.BitwiseOr(x as number, y as number) as number

Arguments

Argument	Description
x	The number to shift.
y	The number of bits to shift.

1.8.4 Number.BitwiseXor

Returns the result of a bitwise XOR operation on the provided operands.

Number.BitwiseXor(x as number, y as number) as number

Arguments

Argument	Description

x	The number to shift.
y	The number of bits to shift.

1.8.5 Number.BitwiseShiftLeft

Returns the result of a bitwise shift left operation on the operands.

Number.BitwiseShiftLeft(x as number, y as number) as number

Arguments

Argument	Description
x	The number to shift.
y	The number of bits to shift.

Remarks

- If y > 0, the high-order bits shifted off are lost, and the low-order bits are filled with zeros.
- If y < 0, the low-order bits shifted off are lost, and the high-order bits are filled with zeros.

1.8.6 Number.BitwiseShiftRight

Returns the result of a bitwise shift right operation on the operands.

Number.BitwiseShiftRight(x as number, y as number) as number

Arguments

Argument	Description
x	The number to shift.
y	The number of bits to shift.

Remarks

- If y > 0, the low-order bits shifted off are lost, and the high-order bits are filled with zeros.
- If y < 0, the high-order bits shifted off are lost, and the low-order bits are filled with zeros.

2. Text

The following functions relate to Text values.

2.1 Information

2.1.1 Text.Length

Returns the number of characters in a text value.

Text.Length(text as `nullable` text) as `nullable` number

Arguments

Argument	Description
text	The input text value

Example

```
Text.Length("abc") equals 3
```

2.2 Text Comparisons

Text comparisons are performed by obtaining a comparer from **`Comparer.FromCulture`**. The comparer returns 0, a negative number, or a positive number based on the result of the comparison. The **`Comparer.Equals`** function is used to compare two text values.

Example

```
let       comparer =
Comparer.FromCulture("en-US", false)
in
   [   comparisonResult =
   comparer("a","b"),
    equalityResult   = Comparer.Equals(comparer,"a","b")
   ]
```

2.3 Conversions from and to text

2.3.1 Text.From

Returns the text representation of a number, date, time, datetime, datetimezone, logical, duration or binary value. If a value is null, Text.From returns null. The optional culture parameter is used to format the text value according to the given culture.

Text.From(value as any, optional culture as nullable text) as nullable text

Arguments

Argument	Description
value	Value to convert.
optional culture	A text value corresponding to the culture values supported on your version of Windows, such as "en-US". If the culture is not specified, the current user culture is used. For a list of culture names, see National Language Support (NLS) API Reference.

Examples

```
Text.From(1) equals "1"
Text.From(#date(2013,5,1)) equals "05/01/2013"
Text.From(#date(2013,5,1),"fr-FR") equals "01/05/2013"
```

2.3.2 Text.ToBinary

Encodes a text value into binary value using an encoding.

Text.ToBinary(text as nullable text, optional encoding as nullable number, optional includeByteOrderMark as nullable logical) as nullable binary

Arguments

Argument	Description
text	Value to encode.
optional encoding	Encoding option to apply.
optional includeByteOrderMark	Specify handling of byte order marks

2.3.3 Text.FromBinary

Decodes data from a binary value in to a text value using an encoding.

Text.FromBinary(binary as nullable binary, optional encoding as nullable number) as nullable text

Arguments

Argument	Description
binary	The value to decode.
optional encoding	Encoding option to apply.

Text encoding

```
TextEncoding.Utf8 = 65001;
TextEncoding.Utf16 = 1200;
TextEncoding.Ascii = 20127;
TextEncoding.Unicode = 1200;
TextEncoding.BigEndianUnicode = 1201, TextEncoding.Windows
= 1252;
```

2.3.4 Text.ToList

Returns a list of characters from a text value.

Text.ToList(text as text) as list

Arguments

Argument	Description
Text	The text to parse through.

Example

```
Text.ToList("abc") equals {“a”,“b”,“c”}
```

2.3.5 Character.FromNumber

Returns a number to its character value.

Character.FromNumber(number as `nullable` number) as `nullable` text

Arguments

Argument	Description
number	The number to transform.

2.3.6 Character.ToNumber

Returns a character to its number value.

Character.ToNumber(character as nullable text) as nullable number

Arguments

Argument	Description
character	The character to transform.

2.3.7 Value.FromText

Decodes a value from a textual representation, value, and interprets it as a value with an appropriate type. `Value.FromText` takes a text value and returns a number, a logical value, a null value, a DateTime value, a Duration value, or a text value. The empty text value is interpreted as a null value.

Value.FromText(value as text, optional culture as nullable text)

Arguments

Argument	Description
value	The value to transform.
optional culture	A text value corresponding to the culture values supported on your version of Windows, such as `"en-US"`. If the culture is not specified, the current user culture is used. For a list of culture names, see National Language Support (NLS) API Reference.

Examples

```
Value.FromText("1") equals 1
Value.FromText("2012/5/16") equals #date(2012,5,16)
Value.FromText("null") equals null
Value.FromText("somevalue") equals "somevalue"
```

2.3.8 Text.NewGuid

Returns a Guid value as a text value.

Text.NewGuid() as text

Example

```
Text.NewGuid() equals "b5f92cce-04d5-4cd5-be90-ee97a1070e84"
```

2.4 Extraction

2.4.1 Text.At

Returns a character starting at a zero-based offset.

Text.At(value as `nullable` text, index as number) as `nullable` text

Arguments

Argument	Description
value	The value to parse.
index	The index of the character to return.

Remarks

If the offset is greater than index, an `Expression.Error` is thrown.

Examples

```
Text.At("abcd", 0) equals "a"
Text.At("abcd", 5) equals error
```

2.4.2 Text.Range

Returns a number of characters from a text value starting at a zero-based `offset` and for `count` number of characters. .

`Text.Range(text as text, offset as number, count as nullable number) as text`

`Arguments`

Argument	Description
text	The text to parse.
offset	The index to start parsing at.
count	The number of characters to return.

Examples

```
Text.Range("abcd", 2) equals “cd”
Text.Range("abcd", 2, 1) equals “c”
```

Remarks

- If `index` is less than zero or greater than the length of the text value, an `Expression.Error` is thrown.
- If `count` is not specified or is greater than the number of characters remaining, then the remainder of the text value is returned.
- If count is a negative value, an `Expression.Error` is thrown.

2.4.3 Text.Start

Returns the count of characters from the start of a text value.

Text.Start(string as `nullable` text, count as number) as `nullable` text

Arguments

Argument	Description
string	The string value to parse.
count	The number of characters to return.

Example

```
Text.Start("abcd", 2) equals "ab"
```

2.4.4 Text.End

Returns the number of characters from the end of a text value.

Text.End(string as `nullable` text, numChars as number) as `nullable` text

Arguments

Argument	Description
string	The string value to parse.
numChars	The number of characters to return.

Example

```
Text.End("abcd", 2) equals "cd"
```

2.5 Modification

2.5.1 Text.Insert

Returns a text value with `newValue` inserted into a text value starting at a zero-based `offset`.

Text.Insert(text as `nullable` text, offset as number, newText as text) as `nullable` text

Arguments

Argument	Description
text	The text to insert into.
offset	The index to insert at.
newText	The new text to insert.

Remark

If `offset` is less than zero or more than the length of a text value, an `Expression.Error` is thrown.

Example

```
Text.Insert("abcdef",2,"X") equals "abXcdef"
```

2.5.2 Text.Remove

Removes all occurrences of a character or list of characters from a text value. The **removeChars** parameter can be a character value or a list of character values.

Text.Remove(text as `nullable` text, removeChars as any) as `nullable` text

Arguments

Argument	Description
text	The text to parse.
removeChars	A character value or a list of character values to be removed.

Examples

```
Text.Remove("a,b,;c",",") equals "ab;c"
Text.Remove("a,b,;c",{",",";"}) equals "abc"
```

2.5.3 Text.RemoveRange

Removes `count` characters at a zero-based offset from a text value.

`Text.Remove(text as nullable text, offset as number, count as number) as nullable text`

`Arguments`

Argument	Description
text	The text to parse.
offset	The index to start at.
count	The number of characters to remove.

Remarks

- If `count` is not specified, the default value of 1 is used.
- If `offset` is less than zero or more than the length of a text value, or if count if less than zero then an `Expression.Error` is thrown.

Examples

```
Text.RemoveRange("abcdef", 2) equals "abdef"
Text.RemoveRange("abcdef", 2, 2) equals "abef"
```

2.5.4 Text.Replace

Replaces all occurrences of a substring with a new text value.

Text.Replace (text as `nullable` text, old as text, new as text) as `nullable` text

Arguments

Argument	Description
text	The text to parse.
old	The text to replace.
new	The replacement text.

Example

```
Text.Replace("Thisisanorange", "orange", "apple") equals "Thisisanapple"
```

2.5.5 Text.ReplaceRange

Replaces `length` characters in a text value starting at a zero-based `offset` with the new text value.

`Text.ReplaceRange(text as nullable text, offset as number, length as number, newText as text) as nullable text`

`Arguments`

Argument	Description
text	The text to parse.
offset	The beginning of the range.

length	The length of the range.
newText	The replacement text.

Example

```
Text.Replace("abcdef", 2, 3, "xyz") equals "abxyzf"
```

2.6 Membership

2.6.1 Text.Contains

Returns true if a text value **`substring`** was found within a text value **`string`**; otherwise, false.

`Text.Contains(string as nullable text, substring as text, optional comparer as nullable function) as nullable logical`

Arguments

Argument	Description
string	The text to parse.
substring	The text to search for.
optional comparer	The optional culture aware comparer function can be provided.

Examples

```
Text.Contains("abc", "a") equals true
Text.Contains("abc", "d") equals false
```

2.6.2 Text.PositionOf

Returns the first occurrence of `substring` in a `string` and returns its position starting at `startOffset`.

Text.PositionOf(string as nullable text, substring as text, optional occurrence as nullable number, optional comparer as nullable function) as nullable number

Arguments

Argument	Description
string	The text to parse.
substring	The text to string to search for.

optional occurrence	An enum that controls the scope of operation.
optional comparer	The optional culture aware comparer function can be provided.

Occurrence Settings

Setting	Description
`Occurrence.First or Occurrence.Last`	A single position is returned.
`Occurrence.All`	A list of positions is returned for all occurrences.

Remarks

If a text value is not found, -1 is returned.

Examples

```
Text.PositionOf("ABCD", "C") equals 2
Text.PositionOf("ABCBA", "A", Occurrence.First) equals 0
Text.PositionOf("ABCBA", "A", Occurrence.Last) equals 4
Text.PositionOf("ABCBA", "A", Occurrence.All) equals {0,4}
```

2.6.3 Text.PositionOfAny

Returns the first occurrence of a text value in `list` and returns its position starting at `startOffset`.

Text.PositionOfAny(string as text, list as list, optional occurrence as nullable number) as number

Arguments

Argument	Description
string	The string to search for.
list	The list to search through.
optional occurrence	An enum that controls the scope of operation.

Settings

Setting	Description

`Occurrence.First or Occurrence.Last`	A single position is returned.
`Occurrence.All`	A list of positions is returned for all occurrences.

Remarks

If the text values are not found in the list, -1 is returned.

Examples

```
List.PositionOfAny("ABCD", {"B","C"}) equals 1
List.PositionOfAny("ABCBA", {"A","B"}, Occurrence.First) equals 0
List.PositionOfAny("ABCBA", {"A","B"}, Occurrence.Last) equals 4
List.PositionOfAny("ABCBA", {"A","B"}, Occurrence.All) equals {0,1,3,4}
```

2.6.4 Text.StartsWith

Returns a logical value indicating whether a text value `substring` was found at the beginning of a string.

Note: Only comparer functions created through the library (Comparer.FromCulture) are supported.

Text.StartsWith(string as `nullable` text, substring as text, optional comparer as nullable function) as `nullable` logical

Arguments

Argument	Description
string	The text value to parse.
substring	The string to search for.
optional comparer	An optional `comparer` can be provided to influence the result.

2.6.5 Text.EndsWith

Returns a logical value indicating whether a text value `substring` was found at the end of a `string`.

Text.EndsWith(string as `nullable` text, substring as text, optional comparer as nullable function) as `nullable` logical

Arguments

Argument	Description
string	The text value to parse.
substring	The string to search for.
optional comparer	An optional `comparer` can be provided to influence the result.

Remarks

Only comparer functions created through the library (Comparer.FromCulture) are supported.

2.7 Transformations

2.7.1 Text.Clean

Returns the original text value with non-printable characters removed.

Text.Clean(string as `nullable` text) as `nullable` text

Arguments

Argument	Description
string	The string to parse.

2.7.2 Text.Combine

Returns a text value that is the result of joining all text values with each value separated by a separator.

Text.Combine(text as list, separator as nullable text) as text

Arguments

Argument	Description
text	The list of text to combine.
separator	The separator to use when combining. This will only appear between the specified text values, not at the beginning or the end.

Example

```
Text.Combine({"a", "b", "c"}, ",") equals "a,b,c"
```

2.7.3 Text.Upper

Returns the uppercase of a text value.

Text.Upper(string as nullable text) as `nullable` text

Arguments

Argument	Description
string	The string to transform.

2.7.4 Text.Lower

Returns the lowercase of a text value.

Text.Lower(string as `nullable` text) as `nullable` text

Arguments

Argument	Description
string	The string to transform.

2.7.5 Text.Proper

Returns a text value with first letters of all words converted to uppercase.

Text.Proper(string as `nullable` text) as `nullable` text

Arguments

Argument	Description
string	The string to transform.

Example

```
Text.Proper("this is an apple") equals "This Is An Apple"
```

2.7.6 Text.PadStart

Returns a text value padded at the beginning with `pad` to make it at least `length` characters. If `pad` is not specified, whitespace is used as pad.

Text.PadStart(text as nullable text, length as number, optional pad as nullable text) as nullable text

Arguments

Argument	Description
text	The text to parse.
length	The length to pad to.
optional pad	The text to pad with. If pad is not specified, whitespace is used as pad.

Examples

```
Text.PadStart("xyz", 5, ""a"") equals "aaxyz" Text.PadStart("xyz",
9, "pad") equals error
```

2.7.7 Text.PadEnd

Returns a text value padded at the end with pad to make it at least length characters.

Text.PadEnd(text as nullable text, length as number, pad as nullable text) as nullable text

Arguments

Argument	Description
text	The text to parse.
length	The length to pad to.
pad	The text to pad with.

Remarks

If pad is not specified, whitespace is used as pad.

Example

```
Text.PadEnd("abc", 5, "a") equals “abcaa”
```

2.7.8 Text.Repeat

Returns a text value composed of the input text value repeated a number of times.

Text.Repeat(string as text, repeatCount as number) as text

Arguments

Argument	Description

string	The text to repeat.
repeatCount	The number of times to repeat the text.

Example

```
Text.Repeat("a",5) equals "aaaaa"
```

2.7.9 Text.Trim

Removes any occurrences of characters in `trimChars` from `text`.

Text.Trim(text as `nullable` text, optional trimChars as any) as `nullable` text

Arguments

Argument	Description
text	The text to trim from.
optional trimChars	A character value or a list of character values.

Remarks

- Characters are removed from the beginning and end of the text value.
- If trimChars is not specified, then whitespace characters are trimmed. Whitespace characters are defined by the Power Query formula language specification document. trimChar is either a character value or a list of character values.

Examples

```
Text.Trim("xyAyz", "x") equals "yAyz"
Text.Trim("xyAyz", {"x","y"}) equals "Ayz"
Text.Trim("xyAyz", {"x","y","z"}) equals "A"
Text.Trim("xyAyz", "xy") equals error
```

2.7.10 Text.TrimStart

Removes any occurrences of the characters in trimChars from the start of the original text value.

Text.TrimStart(text as `nullable` text, optional trimChars as `nullable` list) as `nullable` text

Arguments

Argument	Description
text	The text to trim from.
optional trimChars	A character value or a list of character values.

Remarks

- If trimChars is not specified, then whitespace characters are trimmed.

2.7.11 Text.TrimEnd

Removes any occurrences of the characters specified in `trimChars` from the end of the original text value.

Text.TrimEnd(text as nullable text, optional trimChars as nullable list) as `nullable` text

Arguments

Argument	Description
text	The text to trim from.
optional trimChars	A character value or a list of character values. If `trimChars` is not specified, then whitespace characters are trimmed.

2.7.12 Text.Split

Returns a list containing parts of a text value that are delimited by a separator text value.

Text.Split(string as text, separator as text) as list

Arguments

Argument	Description
string	The string to parse.
separator	A delimiter value.

Remarks

- If two delimiter values are adjacent, or appear at the beginning or end of the text value, the corresponding element in the returned list is an empty text value.
- If the text value does not contain `separator`, the returned array consists of a single item containing the original text value.

2.7.13 Text.SplitAny

Returns a list containing parts of a text value that are delimited by any separator text values.

Text.SplitAny(string as text, separator as text) as list

Arguments

Argument	Description
string	The string to parse.
separator	A delimiter value.

3. Logical

The following functions relate to Logical values.

3.1 Logical.FromText

Returns a logical value of true or false from a text value.

Logical.FromText(text as nullable text) as nullable logical

Arguments

Argument	Description
text	The text to evaluate.

Examples

```
Logical.FromText("true") equals true Logical.FromText("a")
equals error
```

3.2 Logical.ToText

Returns a text value from a logical value.

Logical.ToText(logical as nullable logical) as nullable text

Arguments

Argument	Description
logical	The logical value to evaluate.

Example

```
Logical.ToText(true) equals “true”
```

3.3 Logical.From

Returns a logical value from a value.

Logical.From(value as any) as nullable logical

Arguments

Argument	Description

value	Value to convert.

Type to convert

TYPE	DESCRIPTION
text	Returns a logical value of "true" or "false" from the text value. For more details, see **Logical.FromText**.
number	If value equals 0, true; otherwise, false.
any other type	An Expression.Error is thrown.

Remarks

- If value is null, Logical.From returns null.
- If a value is logical, the same value is returned.

Examples

```
Logical.From(2) equals true
```

4. Date

These functions deal with Date values.

4.1 Date.FromText

Returns a Date value from a set of date formats and culture value.

Date.FromText(date as `nullable` text, optional culture as nullable text) as `nullable` date

Arguments

Argument	Description
date	A string value to transorm.

optional culture	A text value corresponding to the culture values supported on your version of Windows, such as "en-US". If the culture is not specified, the current user culture is used. For a list of culture names, see National Language Support (NLS) API Reference.

Supported formats

- YYYY-MM-DD
- YYYYMMDD
- M/d/yyyy

Terms

- Y = years
- M = months
- D = days

Remarks

- If the culture is not specified, the current user culture is used.

Examples

```
Date.FromText("2010-02-19") equals Date,YYYY-MM-DD
```

4.2 Date.ToText

Returns a text value from a Date value.

Date.ToText(date as nullable date, optional formatas nullable text,optional culture as nullable text) as nullable text

Arguments

Argument	Description
date	The Date to parse to text.
optional format	An optional text value used to format common numeric values.
optional culture	A text value corresponding to the culture values supported on your version of Windows, such as "en-US". If the culture is not specified, the current user culture is used. For a list of culture names, see National Language Support (NLS) API Reference.

Examples

```
Date.ToText(dateTime); equals "2011-02-19"
Date.ToText(dateTime, "YYYYMMDD"); equals "2011021"
```

4.3 Date.From

Returns a date value from a value.

Date.From(value as any, optional culture as nullable text) as nullable date

Arguments

Argument	Description
value	Value to convert.
optional culture	A text value corresponding to the culture values supported on your version of Windows, such as "en-US". If the culture is not specified, the current user culture is used. For a list of culture names, see National Language Support (NLS) API Reference.

Values of the following types can be converted to a date value:

TYPE	DESCRIPTION
text	Returns a Date value from a text value. For more details, see Date.FromText.
datetime	The Date component of a value.
datetimezone	The Date component of the local date and time equivalent of a value.
number	The Date component of the datetime equivalent of the OLE Automation Date of a value.
any other type	An Expression.Error is thrown.

Remarks

- If a value is null, Date.From returns null.
- If a value is date, the same value is returned.

Examples

```
Date.From(43910) equals #date(2020,3,20)
Date.From(#datetime(1899, 12, 30, 6, 45, 12)) equals #date(1899,12,30)
```

4.4 Date.ToRecord

Returns a record containing parts of a Date value.

Date.ToRecord(date as date) as record

Arguments

Argument	Description
date	The Date to parse.

Example

```
Date.ToRecord(#date(2013, 1, 1) equals [Year=2013,Month=1,Day=1]
```

4.5 Date.Year

Returns the year from a DateTime value.

Date.Year(dateTime as datetime) as `nullable` number

Arguments

Argument	Description
date	The Date to retrieve the year for.

Example

```
Date.Year(DateTime.FromText("2011-02-19")) equals 2011
```

4.6 Date.Month

Returns the month from a DateTime value.

Date.Month(dateTime as datetime) as `nullable` number

Arguments

Argument	Description
date	The Date to retrieve the month for.

Example

```
Date.Month(DateTime.FromText("2011-02-19")) equals 2
```

4.7 Date.Day

Returns the day for a DateTime value.

Date.Day(dateTime as datetime) as `nullable` number

Arguments

Argument	Description
date	The Date to retrieve the day for.

Example

```
Date.Day(DateTime.FromText("2011-02-19")) equals 19
```

4.8 Date.AddDays

Returns a Date/DateTime/DateTimeZone value with the day portion incremented by the number of days provided. It also handles incrementing the month and year potions of the value as appropriate.

Date.AddDays(dateTime, days as number)

Arguments

Argument	Description
dateTime	The DateTime to add days to.
days	The number of days to add.

Examples

```
Date.AddDays(DateTime.FromText("2011-02-19"), 5) equals 2011-02-24
```

```
Date.AddDays(DateTime.FromText("2011-02-19"), -2) equals 2011-02-17

Date.AddDays(DateTime.FromText("2011-12-31"), 1) equals 2012-01-01
```

4.9 Date.AddWeeks

Returns a Date/DateTime/DateTimeZone value incremented by the number of weeks provided. Each week is defined as a duration of seven days. It also handles incrementing the month and year potions of the value as appropriate.

Date.AddWeeks(dateTime, weeks as number)

Arguments

Argument	Description
dateTime	The DateTime to add weeks to.
weeks	The number of weeks to add.

Examples

```
Date.AddWeeks(DateTime.FromText("2011-02-19"), 1) equals 2011-02-26
Date.AddWeeks(DateTime.FromText("2011-02-19"), -2) equals 2011-02-05
Date.AddWeeks(DateTime.FromText("2011-12-31"), 1) equals 2012-01-07
```

4.10 Date.AddMonths

Returns a DateTime value with the month portion incremented by n months.

Date.AddMonths(dateTime as datetime, numberOfMonths as number) as nullable datetime
Arguments

Argument	Description
dateTime	The DateTime to add months to.
numberOfMonths	The number of months to add.

Remarks

It also handles incrementing the year portion of the value as appropriate.

Examples

```
Date.AddMonths(DateTime.FromText("2011-02-19"), 5) equals 2011-07-19
Date.AddMonths(DateTime.FromText("2010-12-01"), 2) equals 2011-02-01
```

4.11 Date.AddQuarters

Returns a Date/DateTime/DateTimeZone value incremented by the number of quarters provided. Each quarter is defined as a duration of three months. It also handles incrementing the year potion of the value as appropriate.

Date.AddQuarters(dateTime, quarters as number)

Arguments

Argument	Description
dateTime	The DateTime to add quarters to.
quarters	The number of quarters to add.

Examples

```
Date.AddQuarters(DateTime.FromText("2011-02-19"), 1) equals 2011-05-19
Date.AddQuarters(DateTime.FromText("2011-11-30"), 1) equals 2012-02-29
```

4.12 Date.AddYears

Returns a DateTime value with the year portion incremented by n years.

Date.AddYears(dateTime as datetime, years as number) as datetime

Arguments

Argument	Description
dateTime	The DateTime to add years to.

years	The number of years to add.

Example

```
Date.AddYears(DateTime.FromText("2011-02-19"), 10) equals 2021-02-19
```

4.13 Date.IsInPreviousDay

Returns a logical value indicating whether the given Date/DateTime/DateTimeZone occurred during the previous day, as determined by the current date and time on the system.

Date.IsInPreviousDay(dateTime) as logical

Arguments

Argument	Description
dateTime	Check whether this DateTime occurred during the previous day.

Example

```
Date.IsInPreviousDay(DateTime.FromText("2013-07-08")) equals true if
today is July 9th, 2013
```

4.14 Date.IsInCurrentDay

Returns a logical value indicating whether the given Date/DateTime/DateTimeZone occurred during the current day, as determined by the current date and time on the system.

Date.IsInCurrentDay(dateTime) as logical

Arguments

Argument	Description
dateTime	Check whether this DateTime occurred during the current day.

Example

```
Date.IsInCurrentDay(DateTime.FromText("2013-07-09")) equals true if
today is July 9th, 2013
```

4.15 Date.IsInNextDay

Returns a logical value indicating whether the given Date/DateTime/DateTimeZone occurred during the next day, as determined by the next date and time on the system.

Date.IsInNextDay(dateTime) as logical

Arguments

Argument	Description
dateTime	Check whether this DateTime occurred during the next day.

Example

```
Date.IsInNextDay(DateTime.FromText("2013-07-10")) equals true if today
is July 9th, 2013
```

4.16 Date.IsInPreviousWeek

Returns a logical value indicating whether the given Date/DateTime/DateTimeZone occurred during the previous week, as determined by the current date and time on the system.

Date.IsInPreviousWeek(dateTime) as logical

Arguments

Argument	Description
dateTime	Check whether this DateTime occurred during the previous week

4.17 Date.IsInCurrentWeek

Returns a logical value indicating whether the given Date/DateTime/DateTimeZone occurred during the current week, as determined by the current date and time on the system.

Date.IsInCurrentWeek(dateTime) as logical

Arguments

Argument	Description
dateTime	Check whether this DateTime occurred during the current week.

4.18 Date.IsInNextWeek

Returns a logical value indicating whether the given Date/DateTime/DateTimeZone occurred during the next week, as determined by the next date and time on the system.

Date.IsInNextWeek(dateTime) as logical

Arguments

Argument	Description
dateTime	Check whether this DateTime occurred during the next week.

4.19 Date.IsInPreviousMonth

Returns a logical value indicating whether the given Date/DateTime/DateTimeZone occurred during the previous month, as determined by the current date and time on the system.

Date.IsInPreviousMonth(dateTime) as logical

Arguments

Argument	Description
dateTime	Check whether this DateTime occurred during the previous month.

4.20 Date.IsInCurrentMonth

Returns a logical value indicating whether the given Date/DateTime/DateTimeZone occurred during the current month, as determined by the current date and time on the system.

Date.IsInCurrentMonth(dateTime) as logical

Arguments

Argument	Description
dateTime	Check whether this DateTime occurred during the current month.

4.21 Date.IsInNextMonth

Returns a logical value indicating whether the given Date/DateTime/DateTimeZone occurred during the next month, as determined by the next date and time on the system.

Date.IsInNextMonth(dateTime) as logical

Arguments

Argument	Description
dateTime	Check whether this DateTime occurred during the next month.

4.22 Date.IsInPreviousQuarter

Returns a logical value indicating whether the given Date/DateTime/DateTimeZone occurred during the previous quarter, as determined by the current date and time on the system.

Date.IsInPreviousQuarter(dateTime) as logical

Arguments

Argument	Description
dateTime	Check whether this DateTime occurred during the previous quarter.

4.23 Date.IsInCurrentQuarter

Returns a logical value indicating whether the given Date/DateTime/DateTimeZone occurred during the current quarter, as determined by the current date and time on the system.

Date.IsInCurrentQuarter(dateTime) as logical

Arguments

Argument	Description
dateTime	Check whether this DateTime occurred during the current quarter.

4.24 Date.IsInNextQuarter

Returns a logical value indicating whether the given Date/DateTime/DateTimeZone occurred during the next quarter, as determined by the next date and time on the system.

Date.IsInNextQuarter(dateTime) as logical

Arguments

Argument	Description
dateTime	Check whether this DateTime occurred during the next quarter.

4.25 Date.IsInPreviousYear

Returns a logical value indicating whether the given Date/DateTime/DateTimeZone occurred during the previous year, as determined by the current date and time on the system.

Date.IsInPreviousYear(dateTime) as logical

Arguments

Argument	Description
dateTime	Check whether this DateTime occurred during the previous year.

4.26 Date.IsInCurrentYear

Returns a logical value indicating whether the given Date/DateTime/DateTimeZone occurred during the current year, as determined by the current date and time on the system.

Date.IsInCurrentYear(dateTime) as logical

Arguments

Argument	Description
dateTime	Check whether this DateTime occurred during the current year.

4.27 Date.IsInNextYear

Returns a logical value indicating whether the given Date/DateTime/DateTimeZone occurred during the next year, as determined by the next date and time on the system.

Date.IsInNextYear(dateTime) as logical

Arguments

Argument	Description

dateTime	Check whether this DateTime occurred during the next year.

4.28 Date.IsInYearToDate

Returns a logical value indicating whether the given Date/DateTime/DateTimeZone occurred in the period starting January 1st of the current year and ending on the current day, as determined by the current date and time on the system.

Date. IsInYearToDate(dateTime) as logical

Arguments

Argument	Description
dateTime	Check whether this DateTime occurred in the period starting January 1st of the current year and ending on the current day.

4.29 Date.IsLeapYear

Returns a logical value indicating whether the year portion of a DateTime value is a leap year.

```
Date.IsLeapYear(dateTime as nullable datetime) as nullable logical
```

Arguments

Argument	Description
dateTime	The DateTime to check.

Examples

```
Date.IsLeapYear(DateTime.FromText("2011-01-01")) equals false
Date.IsLeapYear(DateTime.FromText("2012-01-01")) equals true
```

4.30 Date.DayOfWeek

Returns a number between 1 and 7 representing the day of the week from a DateTime value.

Date.DayOfWeek(dateTime as datetime, optional firstDay as nullable number) as nullable number

Arguments

Argument	Description
dateTime	The DateTime to check against.
optional firstDay	A number value as an enum to set the first day of the week. The default is Day.Sunday.

Enum values

- `Day.Sunday = 0;`
- `Day.Monday = 1;`
- `Day.Tuesday = 2;`
- `Day.Wednesday = 3;`
- `Day.Thursday = 4;`
- `Day.Friday = 5;`
- `Day.Saturday = 6;`

Examples

```
Date.DayOfWeek(DateTime.FromText("2011-02-21")) equals 2
DateTime.DayOfWeek(DateTime.FromText("2011-02-21"), Day.Monday) equals 1
```

4.31 Date.DayOfYear

Returns a number that represents the day of the year from a DateTime value.

Date.DayOfYear(dateTime as datetime) as nullable number

Arguments

Argument	Description
dateTime	The DateTime to check against.

Examples

```
Date.DayOfYear(DateTime.FromText("2011-03-01")) equals 60
Date.DayOfYear(DateTime.FromText("2012-03-01")) equals 61
```

4.32 Date.DaysInMonth

Returns the number of days in the month from a DateTime value.

Date.DaysInMonth(dateTime as datetime) as nullable number

Arguments

Argument	Description
dateTime	The DateTime to check against.

Example

```
Date.DaysInMonthDateTime.FromText("2012-03-01")) equals 31
```

4.33 Date.QuarterOfYear

Returns a number between 1 and 4 for the quarter of the year from a DateTime value.

Date.QuarterOfYear(dateTime as datetime) as nullable number

Arguments

Argument	Description
dateTime	The DateTime to check against.

Examples

```
Date.QuarterOfYear(DateTime.FromText("2011-03-21")) equals 1
Date.QuarterOfYear(DateTime.FromText("2011-11-21")) equals 4
```

4.34 Date.WeekOfMonth

Returns a number for the count of week in the current month.

Date.WeekOfMonth(dateTime as datetime) as nullable number

Arguments

Argument	Description

dateTime	The DateTime to check against.

Example

```
Date.WeekOfMonth(DateTime.FromText("2011-08-30")) equals 5
```

4.35 Date.WeekOfYear

Returns a number for the count of week in the current year.

Date.WeekOfYear(dateTime as datetime) as nullable number

Arguments

Argument	Description
dateTime	The DateTime to check against.

Example

```
Date.WeekOfYear(DateTime.FromText("2011-02-21")) equals 8
```

4.36 Date.StartOfYear

Returns a DateTime value representing the start of the year.

Date.StartOfYear(dateTime as nullable datetime) as nullable datetime

Arguments

Argument	Description
dateTime	The DateTime to check against.

Remarks

- The date and time portions are reset to their initial values for the year.
- The timezone information is persisted.

Examples

```
dateTime=DateTimeZone.FromText("2011-02-21T12:30:00-08:00")
Date.StartOfYear(dateTime) equals 2011-01-01T00:00:00-08:00
```

4.37 Date.StartOfQuarter

Returns a Date/DateTime/DateTimeZone value representing the start of the quarter. The date and time portions are reset to their initial values for the quarter. The timezone information is persisted.

Date.StartOfQuarter(dateTime)

Arguments

Argument	Description
dateTime	The DateTime whose date and time portions are to be reset to their initial values for the quarter.

4.38 Date.EndOfQuarter

Returns a Date/DateTime/DateTimeZone value representing the end of the quarter. The date and time portions are reset to their terminating values for the quarter. The timezone information is persisted.

Date.EndOfQuarter(dateTime)

Arguments

Argument	Description
dateTime	The DateTime whose date and time portions are to be reset to their terminating values for the quarter.

4.39 Date.StartOfMonth

Returns a DateTime value representing the start of the month.

Date.StartOfMonth(dateTime as nullable datetime) as nullable datetime

Arguments

Argument	Description
dateTime	The DateTime to check against.

Remarks

- The date and time portions are reset to their initial values for the month.
- The timezone information is persisted.

Examples `dateTime = DateTimeZone.FromText("2011-02-21T12:30:00-08:00"); Date.StartOfMonth(dateTime) equals 2011-02-01T00:00:00-08:00`

4.40 Date.StartOfWeek

Returns a DateTime value representing the start of the week.

Date.StartOfWeek(dateTime as `nullable` datetime, optional firstDay as `nullable` number) as `nullable` datetime

Arguments

Argument	Description
dateTime	The DateTime to check against.
optional firstDay	An optional argument to set the first day of the week.

Enum Values

- `Day.Sunday = 0;`
- `Day.Monday = 1;`
- `Day.Tuesday = 2;`
- `Day.Wednesday = 3;`
- `Day.Thursday = 4;`
- `Day.Friday = 5;`
- `Day.Saturday = 6;`

Remarks

- The date and time portions are reset to their initial values for the week.
- The timezone information is persisted.

Examples
```
dateTime = DateTimeZone.FromText("2011-02-24T12:30:00-08:00");
Date.StartOfWeek(dateTime, Day.Monday) equals 2011-02-21T00:00:00-08:00
```

4.41 Date.StartOfDay

Returns a DateTime value for the start of the day.

Date.StartOfDay(dateTime as nullable datetime) as nullable datetime

Arguments

Argument	Description
dateTime	The DateTime to check against.

Remarks

- The date and time portions are reset to their initial values for the day.
- The timezone information is persisted.

Examples
```
dateTime = DateTimeZone.FromText("2011-02-21T12:30:00-08:00");  Date.StartOfDay(dateTime) equals 2011-02-21T00:00:00-08:00
```

4.42 Date.EndOfYear

Returns a DateTime value for the end of the year.

Date.EndOfYear(dateTime as nullable datetime) as nullable datetime

Arguments

Argument	Description
dateTime	The DateTime to check against.

Remarks

- The date and time portions are reset to their initial values for the year.

- The timezone information is persisted.

Examples `dateTime = DateTimeZone.FromText("2011-02-21T12:30:00-08:00"); Date.EndOfYear(dateTime) equals 2011-12-31T23:59:59-08:00`

4.43 Date.EndOfMonth

Returns a DateTime value for the end of the month.

Date.EndOfMonth(dateTime as nullable datetime) as nullable datetime

Arguments

Argument	Description
dateTime	The DateTime to check against.

Remarks

- The date and time portions are reset to their initial values for the month.
- The timezone information is persisted.

Examples `dateTime = DateTimeZone.FromText("2011-02-21T12:30:00-08:00"); Date.EndOfMonth(dateTime) equals 2011-02-28T23:59:59-08:00`

4.44 Date.EndOfWeek

Returns a DateTime value for the end of the week.

Date.EndOfWeek(dateTime as nullable datetime, optional lastDay as nullable number) as nullable datetime

Arguments

Argument	Description

dateTime	The DateTime to check against.
optional lastDay	A number value as an enum value to set the last day of the week. The default value for lastDay is Day.Saturday.

Day Enum Values

- `Day.Sunday = 0;`
- `Day.Monday  = 1;`
- `Day.Tuesday = 2;`
- `Day.Wednesday = 3;`
- `Day.Thursday = 4;`
- `Day.Friday = 5;`
- `Day.Saturday = 6;`

Remarks

- The date and time portions are reset to their initial values for the week.
- The timezone information is persisted.

Examples
```
dateTime = DateTimeZone.FromText("2011-02-24T12:30:00-08:00");
Date.EndOfWeek(dateTime, Day.Sunday) equals 2011-02-27T23:59:59-08:00
```

4.45 Date.EndOfDay

Returns a DateTime value for the end of the day.

Date.EndOfDay(dateTime as nullable datetime) as nullable datetime

Arguments

Argument	Description
dateTime	The DateTime to check against.

Remarks

- The date and time portions are reset to their initial values for the day.
- The timezone information is persisted.

Examples `dateTime = DateTimeZone.FromText("2011-02-21T12:30:00-08:00"); Date.EndOfDay(dateTime) equals 2011-02-21T23:59:590-08:00`

4.46 Parameter values

The following definitions are used to describe the parameter shapes that are referenced in **DateTime** functions.

4.46.1 Day

```
Day.Sunday = 0;
Day.Monday  = 1;
Day.Tuesday = 2;
Day.Wednesday = 3;
Day.Thursday = 4;
Day.Friday = 5;
Day.Saturday = 6;
```

5. Time

These functions deal with Time values.

5.1 Time.FromText

Returns a Time value from a set of date formats.

Time.FromText(time as nullable text, optional culture as nullable text) as nullable date

Arguments

Argument	Description
time	The text value representing the time.

optional culture	A text value corresponding to the culture values supported on your version of Windows, such as "en-US". If the culture is not specified, the current user culture is used. For a list of culture names, see National Language Support (NLS) API Reference.

Time formats

- hh:mm
- hh:mm:ss
- hh:mm:ss.nnnnnnn

Terms

- h = hours
- m = minutes
- s = seconds
- n = fractional seconds

Examples

```
Time.FromText("12:34:12") equals Time,hh:mm:ss
Time.FromText("12:34:12.1254425") equals hh:mm:ss.nnnnnnn
```

5.2 Time.ToText

Returns a text value from a Time value.

Time.ToText(time as nullable time, optional format as nullable text, optional culture as nullable text) as nullable text

Arguments

Argument	Description
time	The time to convert.
optional format	An optional text value used to format common numeric values.
optional culture	A text value corresponding to the culture values supported on your version of Windows, such as "en-US". If the culture is not specified, the current user culture is used. For a list of culture names, see National Language Support (NLS) API Reference.

Example

```
Time.ToText(#time(11, 56, 2)) equals "11:56AM"
```

5.3 Time.From

Returns a time value from a value.

Time.From(value as any, optional culture as nullable text) as nullable time

Arguments

Argument	Description
value	Value to convert.
optional culture	A text value corresponding to the culture values supported on your version of Windows, such as "en-US". If the culture is not specified, the current user culture is used. For a list of culture names, see National Language Support (NLS) API Reference.

Type to convert

TYPE	DESCRIPTION
text	Returns a time value from text value. For more details, see Time.FromText.
datetime	The time component of a value.
datetimezone	The time component of the local date and time equivalent of value.
number	A time equivalent to the number of fractional days expressed by value. If value is negative or greater or equal to 1, an error is returned.
any other type	An Expression.Error is thrown.

Remarks

- If value is null, Time.From returns null.
- If value is time, the same value is returned.

Examples

```
Time.From(0.7575) equals #time(18,10,48)
Time.From(#datetime(1899, 12, 30, 06, 45, 12)) equals #time(06,45,12)
```

5.4 Time.ToRecord

Returns a record containing parts of a Date value.

Time.ToRecord(time as time) as record

Arguments

Argument	Description
time	The time to parse.

Example

```
Time.ToRecord(#time(12, 1, 2)) equals [Hour=12, Minute=1, Second=2]
```

5.5 Time.StartOfHour

Returns the first value of the hour from a time value.

Time.StartOfHour(datetime as datetime) as nullable datetime

Arguments

Argument	Description
dateTime	The DateTime to check against.

Example

```
Time.StartOfHour(#datetime(2013, 4, 5, 1, 3, 45)) equals #datetime(2013,
4, 5, 1, 0, 0)
```

5.6 Time.EndOfHour

Returns a DateTime value from the end of the hour.

Time.EndOfHour(dateTime as datetime) as nullable datetime

Arguments

Argument	Description
dateTime	The DateTime to check against.

Remarks

- The time portion is reset to its terminating values for the hour.
- The timezone information is persisted.

Examples `dateTime = DateTimeZone.FromText("2011-02-21T12:30:00-08:00"); Time.EndOfHour(dateTime) equals 2011-02-21T12:59:59-08:00`

5.7 Time.Hour

Returns an hour value from a DateTime value.

Time.Hour(dateTime as datetime) as nullable number

Arguments

Argument	Description
dateTime	The DateTime to check against.

Example

```
Time.Hour(DateTime.FromText("12:56:20")) equals 12
```

5.8 Time.Minute

Returns a minute value from a DateTime value.

Time.Minute(dateTime as datetime) as nullable number

Arguments

Argument	Description
dateTime	The DateTime to check against.

Example

```
Time.Minute(DateTime.FromText("12:56:20"))  equals 56
```

5.9 Time.Second

Returns a second value from a DateTime value

Time.Second(dateTime as datetime) as nullable number

Arguments

Argument	Description
dateTime	The DateTime to check against.

Example

```
Time.Second(DateTime.FromText("12:56:20")) equals 20
```

6. DateTime

These functions deal with DateTime values.

6.1 DateTime.FromText

Returns a DateTime value from a set of date formats and culture value.

DateTime.FromText(dateTime as nullable text, optional culture as nullable text) as nullable date

Arguments

Argument	Description
dateTime	The text value to convert.
Culture	A text value corresponding to the culture values supported on your version of Windows, such as `"en-US"`. If the culture is not specified, the current user culture is used. For a list of culture names, see National Language Support (NLS) API Reference.

DateTime formats

- `YYYY-MM-DDThh:mm`
- `YYYYMMDDThh:mm`

- YYYY-MM-DDThh:mm:ss
- YYYYMMDDThh:mm:ss
- YYYY-MM-DDThh:mm:ss.nnnnnnn
- YYYYMMDDThh:mm:ss.nnnnnnn

Terms

- Y = years
- M = months
- D = days
- h = hours
- m = minutes
- s = seconds
- n = fractional seconds

Example

```
DateTime.FromText("2010-12-31T01:30:00") equals YYYY-MM-DDThh:mm:ss
```

6.2 DateTime.ToText

Returns a text value from a DateTime value.

DateTime.ToText(dateTime as nullable datetime, optional formatas nullable text, optional culture as nullable text) as nullable text

Arguments

Argument	Description
dateTime	The DateTime to convert.
optional format	An optional text value used to format common numeric values.
optional culture	A text value corresponding to the culture values supported on your version of Windows, such as "en-US". If the culture is not specified, the current user culture is used. For a list of culture names, see National Language Support (NLS) API Reference.

6.3 DateTime.From

Returns a datetime value from a value.

DateTime.From(value as any, optional culture as nullable text) as nullable datetime

Arguments

Argument	Description
value	Value to convert.
optional culture	A text value corresponding to the culture values supported on your version of Windows, such as "en-US". If the culture is not specified, the current user culture is used. For a list of culture names, see National Language Support (NLS) API Reference.

Type to convert

TYPE	DESCRIPTION
text	Returns a datetime value from a text value. For more details, see DateTime.FromText.
date	A datetime with value as the date component and 12:0:00 AM as the time component.
datetimezone	The local date and time equivalent of value.
time	A datetime with the date equivalent to the OLE Automation Date of 0 as the date component and value as the time component.
number	A datetime equivalent the OLE Automation Date expressed by value.
any other type	An Expression.Error is thrown.

Remarks

- If a value is null, DateTime.From returns null.
- If a value is datetime, the same value is returned.

Examples

```
DateTime.From(#time(06, 45, 12)) equals #datetime(1899, 12, 30, 06, 45,
12)
DateTime.From(#date(1975, 4, 4)) equals #datetime(1975, 4, 4, 0, 0, 0)
```

6.4 DateTime.FromFileTime

Returns a DateTime value from the supplied number.

DateTime.FromFileTime(fileTime as nullable number) as nullable datetime

Arguments

Argument	Description
fileTime	The fileTime is a Windows file time value that represents the number of 100-nanoseconds intervals that have elapsed since 12:00 midnight, January 1, 1601 A.D. (C.E.) Coordinated Universal Time (UTC).

Example

```
DateTime.FromFileTime(129876402529842224) equals #datetime(2012, 7, 24, 14, 50, 52.9842245)
```

6.5 DateTime.ToRecord

Returns a record containing parts of a DateTime value.

DateTime.ToRecord(dateTime as datetime) as record

Arguments

Argument	Description
dateTime	The DateTime to parse.

Example

```
DateTime.ToRecord(#datetime(2013,1,3,12,4,5)) equals
[
       Year = 2013,   Month = 1,   Day = 3,
       Hour = 12,  Minute = 4, Second = 5
]
```

6.6 DateTime.Date

Returns a date part from a DateTime value

DateTime.Date(dateTime as datetime) as nullable datetime

Arguments

Argument	Description
dateTime	The DateTime to parse.

Example

```
DateTime.Date(#datetime(2010, 5, 4, 6, 5, 4)) equals #date(2010, 5, 4)
```

6.7 DateTime.Time

Returns a time part from a DateTime value.

DateTime.Time(dateTime as datetime) as nullable time

Arguments

Argument	Description
dateTime	The DateTime to parse.

Example

```
DateTime.Time(#datetime(2010, 5, 4, 6, 5, 4)) equals #time(6, 5, 4)
```

6.8 DateTime.AddZone

Adds the timezonehours as an offset to the input datetime value and returns a new datetimezone value.

DateTime.AddZone(dateTime as nullable datetime, timezoneHours as number, optional timezoneMinutes as nullable number) as nullable datetimezone

Arguments

Argument	Description
dateTime	A DateTime to modify.
timezoneHours	The hours to add.
optional timezoneMinutes	The minutes to add.

Example

```
DateTime.AddZone(#datetime(2010, 5, 4, 6, 5, 5), 8) equals
```

```
#datetimezone(2010, 5, 4, 6, 5, 5, 8, 0)
```

6.9 DateTime.LocalNow

Returns a datetime value set to the current date and time on the system.

DateTime.LocalNow() as datetime

Remarks

The returned value does not contain timezone information.

Example

```
DateTime.LocalNow() equals 2013-03-08T14:22:42
```

6.10 DateTime.FixedLocalNow

Returns a DateTime value set to the current date and time on the system. This value is fixed and will not change with successive calls, unlike DateTime.LocalNow, which may return different values over the course of execution of an expression.

DateTime.FixedLocalNow() as datetime

6.11 DateTimeZone.FixedLocalNow

Returns a DateTimeZone value set to the current date, time, and timezone offset on the system. This value is fixed and will not change with successive calls, unlike DateTime.LocalNow, which may return different values over the course of execution of an expression.

DateTimeZone.FixedLocalNow() as datetimezone

6.12 DateTimeZone.FixedUtcNow

Returns the current date and time in UTC (the GMT timezone). This value is fixed and will not change with successive calls..

DateTimeZone.FixedUtcNow() as datetimezone

7. DateTimeZone

These functions deal DateTimeZone values.

7.1 DateTimeZone.FromText

Returns a DateTimeZone value from a set of date formats and culture value.

DateTimeZone.FromText(dateTimeZone as nullable text, optional culture as nullable text) as nullable datetimezone

Arguments

Argument	Description
dateTimeZone	
optional culture	A text value corresponding to the culture values supported on your version of Windows, such as "en-US". If the culture is not specified, the current user culture is used. For a list of culture names, see National Language Support (NLS) API Reference.

DateTimeZone formats

- YYYY-MM-DDThh:mmhh:mm
- YYYYMMDDThh:mmhh:mm
- YYYY-MM-DDThh:mm:sshh:mm
- YYYYMMDDThh:mm:sshh:mm
- YYYY-MM-DDThh:mm:ss.nnnnnnnhh:mm
- YYYYMMDDThh:mm:ss.nnnnnnnhh:mm
- YYYY-MM-DDThh:mm-hh:mm
- YYYYMMDDThh:mm-hh:mm
- YYYY-MM-DDThh:mm:ss-hh:mm
- YYYYMMDDThh:mm:ss-hh:mm
- YYYY-MM-DDThh:mm:ss.nnnnnnn-hh:mm
- YYYYMMDDThh:mm:ss.nnnnnnn-hh:mm
- YYYY-MM-DDThh:mmZ
- YYYYMMDDThh:mmZ
- YYYY-MM-DDThh:mm:ssZ
- YYYYMMDDThh:mm:ssZ
- YYYY-MM-DDThh:mm:ss.nnnnnnnZ
- YYYYMMDDThh:mm:ss.nnnnnnnZ

Terms

- Y = years

- M = months
- D = days
- h = hours
- m = minutes
- s = seconds
- n = fractional seconds
- TZD = time zone designator

Examples

```
DateTime.FromText("2010-12-31T01:30:00") equals YYYY-MM-DDThh:mm:ss
DateTime.FromText("2010-12-31T01:30:00Z") equals 2010-12-
31T01:30:00+00:00
```

7.2 DateTimeZone.ToText

Returns a text value from a DateTime value.

DateTime.ToText(dateTime as nullable datetime, optional formatas nullable text, optional culture as nullable text) as nullable text

Arguments

Argument	Description
dateTime	The DateTime to convert.
optional format	An optional text value used to format common numeric values.
optional culture	A text value corresponding to the culture values supported on your version of Windows, such as "en-US". If the culture is not specified, the current user culture is used. For a list of culture names, see National Language Support (NLS) API Reference.

7.3 DateTimeZone.From

Returns a datetimezone value from a value.

DateTimeZone.From(value as any, optional culture as nullable text) as nullable datetimezone

Arguments

Argument	Description
value	Value to convert.
optional culture	A text value corresponding to the culture values supported on your version of Windows, such as "en-US". If the culture is not specified, the current user culture is used. For a list of culture names, see National Language Support (NLS) API Reference.

Type to convert

TYPE	DESCRIPTION
text	Returns a datetimezone value from a text value. For more details, see DateTimeZone.FromText.
date	A datetimezone with value as the date component, 12:0:00 AM as the time component and the offset corresponding the local time zone.
datetime	A datetimezone with value as the datetime and the offset corresponding the local time zone.
time	A datetimezone with the date equivalent of the OLE Automation Date of 0 as the date component, value as the time component and the offset corresponding the local time zone.
number	A datetimezone with the datetime equivalent the OLE Automation Date expressed by value and the offset corresponding the local time zone.
any other type	An Expression.Error is thrown.

Remarks

- If a value is null, DateTimeZone.From returns null.
- If a value is datetimezone , the same value is returned.

Example

```
DateTimeZone.From("2020-10-30T01:30:00-08:00") equals
#datetimezone(2020, 10, 30, 01, 30, 00, -8, 00)
```

7.4 DateTimeZone.FromFileTime

Returns a DateTimeZone from a number value.

DateTimeZone.FromFileTime(fileTime as nullable number) as nullable datetimezone

Arguments

Argument	Description
fileTime	The fileTime is a Windows file time value that represents the number of 100nanoseconds intervals that have elapsed since 12:00 midnight, January 1, 1601 A.D. (C.E.) Coordinated Universal Time (UTC).

Examples

```
DateTimeZone.FromFileTime(129876402529842224) equals #datetimezone(2012, 7,
24, 14, 50, 52.9842245, -7, 0)
```

7.5 DateTimeZone.ToRecord

Returns a record containing parts of a DateTime value.

DateTimeZone.ToRecord(dateTimeZone as datetimezone) as record

Arguments

Argument	Description
dateTimeZone	The DateTimeZone to convert.

Remarks

If a portion of the DateTime value is not specified with date, time or timezone, the corresponding part in the output record is not present.

Example

```
DateTime.ToRecord(DateTime.FromText("2011-02-02T11:56:02-08:00")) equals

   Year = 2011, Month = 2, Day = 2,
   Hour = 11, Minute = 56, Second = 2,
   Hours = -8, Minutes = 0
]
```

```
DateTime.ToParts(DateTime.From("11:56:02-05")) [
    Time = [Hour = 11, Minute = 56, Second = 2],
    Timezone = [Hours = -5]
]
```

7.6 DateTimeZone.ZoneHours

Returns a time zone hour value from a DateTime value.

DateTimeZone.ZoneHours(dateTime as datetimezone) as nullable number

Arguments

Argument	Description
dateTime	The DateTimeZone to check against.

Example

```
DateTimeZone.ZoneHours(DateTime.FromText("12:56:20-08:00")) equals -8
```

7.7 DateTimeZone.ZoneMinutes

Returns a time zone minute value from a DateTime value.

DateTimeZone.ZoneMinutes(dateTime as datetimezone) as nullable number

Arguments

Argument	Description
dateTime	The DateTimeZone to check against.

Example

```
DateTimeZone.ZoneMinutes(#datetimezone(2013, 2, 14, 20, 39, 47.2845649,
5, 5)) equals 5
```

7.8 DateTimeZone.LocalNow

Returns a DateTime value set to the current system date and time.

DateTimeZone.LocalNow() as datetimezone

Remarks

The return value contains timezone information representing the local timezone.

Example

```
DateTimeZone.LocalNow() equals 2011-02-20T22:19:38-08:00
```

7.9 DateTimeZone.UtcNow

Returns a DateTime value set to the current system date and time in the Utc timezone.

DateTimeZone.UtcNow() as datetimezone

Remarks

The return value contains timezone information for the Utc timezone (00:00).

Example

```
DateTimeZone.UtcNow() equals 2011-02-21T06:25:51+00:00
```

7.10 DateTimeZone.ToLocal

Returns a DateTime value from the local time zone.

DateTimeZone.ToLocal(dateTime as datetimezone) as nullable datetimezone

Arguments

Argument	Description
dateTime	The DateTimeZone to convert.

Example

```
//assuming local as PST dateTime  = DateTimeZone.FromText("2011-02-
20T22:19:27+03:00") localTime=DateTimeZone.ToLocal(dateTime) equals
2011-02-20T11:19:2708:00
```

7.11 DateTimeZone.ToUtc

Returns a DateTime value to the Utc time zone.

DateTimeZone.ToUtc(dateTime as datetimezone) as nullable datetimezone

Arguments

Argument	Description
dateTime	The DateTimeZone to convert.

Example

```
dateTime  = DateTimeZone.FromText("2011-02-20T22:19:27+03:00") utcTime =
DateTimeZone.ToUtc(dateTime)  equals 2011-02-20T19:19:27+00:00
```

7.12 DateTimeZone.SwitchZone

Changes the timezone information for the input DateTimeZone.

DateTimeZone(dateTimeZone as datetimezone, timezoneHours as number, optional timezoneMinutes as nullablenumber) as nullable datetimezone

Arguments

Argument	Description
dateTimeZone	The DateTimeZone to modify.
timezoneHours	The hours to add.
optional timezoneMinutes	The minutes to add.

Remarks

If the input value does not have a timezone component, DateTimeZone.SwitchZone throws Expression.Error.

Examples

```
DateTimeZone.SwitchZone(#datetimezone(2010, 5, 4, 6, 5, 5, 0, 0), 8)
equals #datetimezone(2010, 5, 4, 14, 5, 5, 8, 0)
```

```
DateTimeZone.SwitchZone(#datetimezone(2010, 12, 31, 11, 56, 02, 7, 30), 0,
-30) equals #datetimezone(2010, 12, 31, 3, 56, 2, 0, -30)
```

7.13 DateTimeZone.RemoveZone

Returns a datetime value with the zone information removed from the input datetimezone value.

DateTimeZone.RemoveZone(dateTimeZone as datetimezone) as nullable datetime

Arguments

Argument	Description
dateTimeZone	The DateTimeZone to modify.

Example

```
DateTimeZone.RemoveZone(#datetimezone(2010, 5, 4, 14, 5, 5, 8, 0))
equals #datetime(2010, 5, 4, 14, 5, 5)
```

8. Duration

The following functions deal with Duration values.

8.1 Duration.FromText

Returns a Duration value from a text value.

Duration.FromText(duration as nullable text) as nullable duration

Arguments

Argument	Description
Duration	The text to convert.

Duration settings

FORMAT
`[-]hh:mm[:ss]`
`[-]ddd.hh:mm[:ss]`

Note: The values within brackets [] are optional.

Format parts

PART	DESCRIPTION
[-]	The text value is prepended with an optional negative sign [-] to indicate a negative duration value.
[d]	The [d] part represents the day portion of the duration value.
[m]	The [m] part represents the minute portion of the duration value.
[s]	The [s] part represents the second portion of the duration value.

Examples

```
Duration.FromText("15:35")  equals 15 hours, 35 minutes
Duration.FromText("2.15:00") equals 2 days, 15 hours
```

8.2 Duration.ToText

Returns a text value from a Duration value.

Duration.ToText(duration as nullable duration, optional format as nullable text) as text

Arguments

Argument	Description
duration	The Duration to convert.
format	A format specified text value can be provided as an optional second argument. For supported formats, see §8.1.

Examples
```
duration = #duration(2, 5,
55, 20)
Duration.ToText(duration) equals "2.05:55:20"
```

8.3 Duration.From

Returns a duration value from a value.

Duration.From(value as any) as nullable duration

Arguments

Argument	Description
value	Value to convert.

Values of the following types can be converted to a duration value:

TYPE TO CONVERT	DESCRIPTION

text	Returns a duration value from a text value in a elapsed time format of d.h:m:s. For more details, see Duration.FromText.
number	A duration equivalent to the number of whole and fractional days expressed by value.
Any other type	An error is returned

Remarks

- If a value is null, Duration.From returns null.
- If a value is duration, the same value is returned.

Example

```
Duration.From(2.525) equals #duration(2,12,36,0)
```

8.4 Duration.ToRecord

Returns a record with parts of a Duration value.

Duration.ToRecords(duration as duration) as record

Arguments

Argument	Description
duration	The Duration to parse.

Example

```
Duration.ToRecord(#duration(2, 5, 55, 20)) equals [Days=2, Hours=5,
Minutes=55, Seconds=20]
```

8.5 Duration.Days

Returns the day component of a Duration value.

Duration.Days(duration as nullable duration) as nullable number

Arguments

Argument	Description
duration	The Duration to parse.

8.6 Duration.Hours

Returns an hour component of a Duration value.

Duration.Hours(duration as nullable duration) as nullable number

Arguments

Argument	Description
duration	The Duration to parse.

8.7 Duration.Minutes

Returns a minute component of a Duration value.

Duration.Minutes(duration as nullable duration) as nullable number

Arguments

Argument	Description
duration	The Duration to parse.

8.8 Duration.Seconds

Returns a second component of a Duration value.

Duration.Seconds(duration as nullable duration) as nullable number

Arguments

Argument	Description
duration	The Duration to parse.

Examples

```
duration1 = Duration.FromText("2.05:55:20") duration2
= Duration.FromText("15:50")
Duration.Days(duration1) equals 2
Duration.Hours(duration1) equals 5
Duration.Minutes(duration1) equals 55
Duration.Seconds(duration1) equals 20 Duration.Seconds(duration2)
equals 0
```

8.9 Duration.TotalDays

Returns the total magnitude of days from a Duration value.

Duration.TotalDays(duration as nullable duration) as nullable number Arguments

Argument	Description
duration	The Duration to parse.

8.10 Duration.TotalHours

Returns the total magnitude of hours from a Duration value.

Duration.TotalHours(duration as nullable duration) as nullable number

Arguments

Argument	Description
duration	The Duration to parse.

8.11 Duration.TotalMinutes

Returns the total magnitude of minutes from a Duration value.

Duration.TotalMinutes(duration as nullable duration) as nullable number

Arguments

Argument	Description
duration	The Duration to parse.

8.12 Duration.TotalSeconds

Returns the total magnitude of seconds from a duration value.

Duration.TotalSeconds(duration as nullable duration) as nullable number

Arguments

Argument	Description
duration	The Duration to parse.

Examples

```
let
duration =
#duration(
2,22,120,2
0)
 in
```

```
[  totaldays= Duration.TotalDays(duration) equals 3.0002
       totalhours= Duration.TotalHours(duration) equals 72.005
       totalminutes= Duration.TotalMinutes(duration) equals
       4320.33 totalseconds=Duration.TotalSeconds(duration) equals
       259220
]
```

9. Record

The following functions deal with Record values.

9.1 Information

9.1.1 Record.FieldCount

Returns the number of fields in a record.

Record.FieldCount(record as record) as number

Arguments

Argument	Description
record	The Record to check against.

Example

```
Record.FieldCount([A=1, B=2]) equals 2
```

9.1.2 Record.HasFields

Returns true if the field name or field names are present in a record.

Record.HasFields(record as record, fields as any) as logical

Arguments

Argument	Description
record	The Record to check against.
fields	A text value or a list of text values.

Examples:

```
Record.HasFields([A=1, B=2, C=3],"A") equals true
Record.HasFields([A=1, B=2, C=3],{"A", "D"}) equals false
```

9.2 Transformations

9.2.1 Record.AddField

Adds a field from a field name and value.

Record.AddField (record as record, fieldName as text, value as any,optional delayed as nullable logical) as record

Arguments

Argument	Description
record	The Record to modify.
fieldName	The value to name the field.
value	The value to add to the field.
optional delayed	Indicates whether the field value or a function that computes the field value.

Example

```
Record.AddField([A=1, B=2], "C", 3) equals [A=1, B=2, C=3]
```

9.2.2 Record.Combine

Combines the records in a list.

Record.Combine(list as list) as record

Arguments

Argument	Description
list	The list of records to combine.

Remarks

If the list contains non-record values, an error is returned.

Examples

```
Record.Combine({[A=1], [B=2]}) equals [A=1, B=2]
```

9.2.3 Record.TransformFields

Transforms fields by applying **transformOperations**. For more more information about values supported by **transformOperations**, see Parameter Values.

Record.TransformFields(record as record, transformOperations as list, optional missingField as nullable number) as record

Arguments

Argument	Description
record	The Record to modify.
transformOperations	The list of transformations to make.
optional missingField	A **MissingField** enum value to handle missing fields. The default value is MissingField.Error.

MissingField enum

```
MissingField.Error = 0;
MissingField.Ignore = 1;
MissingField.UseNull = 2;
```

Examples

```
Record.TransformFields([A="1", B=3], {"A",Number.FromText}) equals [A=1,
B=3]
Record.TransformFields([A="1", B=3],{{"A", Number.FromText},
{"B",Number.ToText}}) equals [A=1, B="3"]
```

9.2.4 Record.RemoveFields

Returns a new record that reorders the given fields with respect to each other. Any fields not specified remain in their original locations.

Record.RemoveFields(record as record, fields as any, optional missingField as nullable number) as record

Arguments

Argument	Description
record	The Record to modify.
fields	A list of two items with the names of the fields that need to exchange their order in the record.
optional missingField	A **MissingField** enum value to handle missing fields. The default value is
	MissingField.Error.

MissingField enum

```
MissingField.Error = 0;
MissingField.Ignore = 1;
MissingField.UseNull = 2;
```

Examples

```
Record.RemoveFields([A=1,B=3], "A")  equals [B=3]
Record.RemoveFields([A=1,B=3], {"A","B"}) equals  []
```

9.2.5 Record.ReorderFields

Returns a new record that reorders fields relative to each other. Any fields not specified remain in their original locations. Requires two or more fields.

Record.ReorderFields(record as record, fieldOrder as list, optional missingField as nullable number) as record

Arguments

Argument	Description
record	The Record to modify.
fieldOrder	The list of field orders to change to.
optional missingField	A **MissingField** enum value to handle missing fields. The default value is MissingField.Error.

MissingField enum

- `MissingField.Error = 0;`
- `MissingField.Ignore = 1;`
- `MissingField.UseNull = 2;`

Examples

```
Record.ReorderFields([A=1, B=2], {"B", "A"})  equals [B=2, A=1]
Record.ReorderFields([A=1, B=2, C=3, D=4],{"D" ,"B"}) equals [A=1, D=4,
C=3, B=2]
```

9.2.6 Record.RenameFields

Returns a new record that renames the fields specified. The resultant fields will retain their original order. This function supports swapping and chaining field names. However, all target names plus remaining field names must constitute a unique set or an error will occur.

Record.RenameFields(record as record, renames as list, optional missingField as nullable number) as record

Arguments

Argument	Description
record	The record to modify.
renames	The list of renames to apply.
optional missingField	A **MissingField** enum value to handle missing fields. The default value is MissingField.Error.

MissingField enum

- `MissingField.Error = 0;`
- `MissingField.Ignore = 1;`
- `MissingField.UseNull = 2;`

Remarks

- Record.RenameFields swaps and chains field names. If all target names plus remaining field names are not a unique set, an Expression.Error is thrown

Examples

```
Record.RenameFields([A=1, B=2], {"A","AA"})  equals [AA=1, B=2]
Record.RenameFields([A=1, B=2], {{"A", "AA"}, {"B", "BB"}}) equals
[AA=1, BB=2]
Record.RenameFields([A=1, B=2],{{"A", "B"}, {"B", "A"}}) equals [B=1,
A=2]
Record.RenameFields([A=1, B=2, C=3],{{"A", "B"}, {"B", "C"}, {"C",
"A"}}) equals [B=1, C=2, A=3]
Record.RenameFields([A=1, B=2], {"A", "B"}) equals error
```

9.3 Selection

9.3.1 Record.Field

Returns the value of the given field. This function can be used to dynamically create field lookup syntax for a given record. In that way it is a dynamic verison of the record[field] syntax.

Record.Field(record as record, field as text) as any

Arguments

Argument	Description
record	The record to check.
field	The field to obain the value for.

Examples

```
Record.Field([A=1, B=2, C=3], "A") equals 1
```

9.3.2 Record.FieldValues

Returns a list of field values in order of the record's fields.

Record.FieldValues(record as record) as list

Arguments

Argument	Description
record	The record to check.

Example

```
Record.FieldValues([A=1, B=2, C=3])  equals {1, 2, 3}
```

9.3.3 Record.FieldNames

Returns a list of field names in order of the record's fields.

Record.FieldNames(record as record) as list

Arguments

Argument	Description
record	The Record to check.

Example

```
Record.FieldNames([A=1, B=2, C=3]) equals {"A", "B", "C"}
```

9.3.4 Record.FieldOrDefault

Returns the value of a field from a record, or the default value if the field does not exist.

Record.FieldOrDefault(record as record, field as text, optional defaultValue as any) as any

Arguments

Argument	Description
record	The Record to check.
field	The field to return.
optional defaultValue	The default value to return if the field does not exist.

Examples

```
Record.FieldOrDefault([A=1, B=2], "C") equals null
Record.FieldOrDefault([A=1, B=2], "C", 100) equals 100
```

9.3.5 Record.SelectFields

Returns a new record that contains the fields selected from the input record. The original order of the fields is maintained.

Record.SelectFields(record as record, fields as any, optional missingField as nullable number) as record

Arguments

Argument	Description
record	The record to check.
fields	A single field name or list of field names.
optional missingField	A **MissingField** enum value to handle missing fields. The default value is MissingField.Error.

MissingField enum

- `MissingField.Error = 0;`
- `MissingField.Ignore = 1;`
- `MissingField.UseNull = 2;`

Remarks

- The original order of the fields is maintained.

Examples

```
Record.SelectFields([A=1, B=2], "B") equals [B=2]
Record.SelectFields([A=1, B=2, C=3], {"C", "B"}) equals [B=2, C=3]
```

9.4 Serialization

9.4.1 Record.ToTable

Returns a table of records containing field names and values from an input record.

Record.ToTable(record as record) as list

Arguments

Argument	Description
record	The Record to convert.

Remarks

- The type of the return value of this function is {[Name = text, Value = any]}.

Example

```
Record.ToTable([A=1, B=2, C=3]) equals {[Name="A", Value=1], [Name="B",
Value=2], [Name="C", Value=3]}
```

9.4.2 Record.FromTable

Returns a record from a table of records containing field names and values.

Record.FromTable(list as table) as record

Arguments

Argument	Description
list	The Table to check.

Remarks

- An Expression.Error is thrown if the fields are not unique.

Examples `let input = Table.FromRows({{"A",1}, {"B", 2}} {"Name", "Value"}) in Record.FromTable(input) equals [ A = 1, B = 2 ]`

9.4.3 Record.FromList

Returns a record from a list of field values and a set of field names.

Record.FromList(list as list, fields as any) as record

Arguments

Argument	Description
list	The list of values in the record to check.
fields	The set of fields corresponding to the values. The fields can be specific either by a list of text values or a record type.

Remarks

- An Expression.Error is thrown if the fields are not unique.

Examples

```
Record.FromList
   (
   {1, 2},
   {"A", "B"}
   )  equals [  A = 1, B
= 2 ]

Record.FromList
   ( {1, 2},  type [A =
   any, B = any]
   ) equals [  A = 1, B
= 2 ]
```

9.5 Parameter values

The following type definitions are used to describe the parameter values that are referenced in Record functions above.

9.5.1 MissingField option

```
MissingField.Error = 0;
```

```
MissingField.Ignore = 1; MissingField.UseNull
= 2;
```

9.5.2 Transform operations

Transform operations can be specified by either of the following values:

- A list value of two items, first item being the field name and the second item being the transformation function applied to that field to produce a new value.
- A list of transformations can be provided by providing a list value, and each item being the list value of 2 items as described above.

For examples, see description of `Record.TransformFields`

9.5.3 Rename operations

Rename operations for a record can be specified as either of:

- A single rename operation, which is represented by a list of two field names, old and new.
- Multiple rename operations, which is represented by a list value with each item being a rename operation described above.

For examples, see description of `Record.RenameFields.`

10. List

These functions deal with list of values.

10.1 Information

10.1.1 List.Count

Returns the number of items in a list.

List.Count(list as list) as number

Arguments

Argument	Description
list	The List to check.

Examples

```
List.Count({1,2,3}) equals 3 List.Count({})
equals 0
```

10.1.2 List.NonNullCount

Returns the number of items in a list excluding null values

List.NonNullCount(list as list) as number

Arguments

Argument	Description
list	The List to check.

Examples

```
List.NonNullCount({1, null}) equals 1
```

10.1.3 List.IsEmpty

Returns whether a list is empty.

List.IsEmpty(list as list) as logical

Arguments

Argument	Description
list	The List to check.

Examples

```
List.IsEmpty({}) equals true
List.IsEmpty({1, 2, 3}) equals false
```

10.2 Selection

10.2.1 List.Alternate

Returns a list with the items alternated from the original list based on a count, optional repeatInterval, and an optional offset.

List.Alternate(list as list, count as number, optional repeatInterval as nullable number, optional offset as nullable number) as list

Arguments

Argument	Description
list	The List to modify.
count	Alternate count.
optional repeatInterval	Alternate repeat interval.
optional offset	Alternation offset.

Remarks

If the repeatInterval and offset are not provided then List.Alternate is equivalent to List.Skip.

Examples

```
List.Alternate({1, 2, 3, 4, 5, 6, 7, 8, 9, 10}, 2, 2, 0) equals {3, 4,
7, 8}
```

10.2.2 List.Distinct

Filters a list down by removing duplicates. An optional equation criteria value can be specified to control equality comparison. The first value from each equality group is chosen.

For more information about equationCriteria, see Parameter Values.

List.Distinct(list as list, optional equationCriteria as any, criteria as any) as list

Arguments

Argument	Description
list	The List to check.
optional equationCriteria	An equality group equation.
criteria	Filter criteria.

Examples

```
List.Distinct({1, 2, 3, 2, 3}) equals {1, 2, 3}
List.Distinct({"a","b","A"}, each _) equals {"a", "b", "A"}
List.Distinct({"a","b","A"}, Comparer.FromCulture("en",true)) equals
```

```
{"a", "b"}
List.Distint({[a="a",b=2],[a="b",b=3],[a="A",b=4]},
{ each [a] , Comparer.FromCulture("en", true) } )
// { [ a = "a", b = 2 ],
//   [a = "b",  b = 3 ] }
```

10.2.3 List.IsDistinct

Returns whether a list is distinct.

List.IsDistinct(list as list, optional equationCriteria as any) as logical

Arguments

Argument	Description
list	The List to check.
optional equationCriteria	Equation criteria value used to control equality comparison. For more information about equationCriteria, see Parameter Values.

Examples

```
List.IsDistinct({1, 2, 3, 2, 3}) equals false
List.IsDistinct({"a","b","A"}, Comparer.FromCulture("en",false) equals
true
List.IsDistinct({"a","b","A"}, Comparer.FromCulture("en",true) equals
false
```

10.2.4 List.Select

Selects the items that match a condition.

List.Select(list as list, condition as function) as list

Arguments

Argument	Description
list	The List to check.
condition	The condition to match against.

Examples

```
List.Select({1, 3, 5}, each_ > 2) equals {3 ,5}
```

10.2.5 List.First

Returns the first value of the list or the specified default if empty. Returns the first item in the list, or the optional default value, if the list is empty. If the list is empty and a default value is not specified, the function returns.

List.First(list as list, optional defaultValue as any) as any

Arguments

Argument	Description
list	The List to check.
optional defaultValue	Default value if list is empty.

Examples

```
List.First({1, 2, 3}) equals 1 List.First({})
equals null
```

10.2.6 List.FirstN

Returns the first set of items in the list by specifying how many items to return or a qualifying condition provided by **countOrCondition**.

List.FirstN(list as list, countOrCondition as any) as any

Arguments

Argument	Description
list	The List to check.
countOrCondition	The number or condition to qualify against.

Remarks

- If a number is specified, up to that many items are returned.
- If a condition is specified as a function, all items are returned that initially meet the condition.
- Once an item fails the condition, no further items are considered.

Examples

```
List.FirstN({3, 4, 5, -1, 7, 8, 2}, 2) equals {3, 4}
List.FirstN({3, 4, 5, -1 ,7, 8, 2}, each_ > 2) equals {3, 4, 5}
```

10.2.7 List.InsertRange

Inserts items from values at the given index in the input list.

List.InsertRange(list as list, offset as number, values as list) as list

Arguments

Argument	Description
list	The List to modify.
offset	The index to insert at.
values	The values to insert.

Examples

```
List.InsertRange({"A", "B", "D"}, 2, {"C"}) equals {"A", "B", "C", "D"}
```

10.2.8 List.Last

Returns the last set of items in the list by specifying how many items to return or a qualifying condition provided by **countOrCondition**.

List.Last(list as list, optional defaultValue as any) as any

Arguments

Argument	Description
list	The List to check.
optional defaultValue	Default value if list is empty.

Remarks

- If a number is specified, up to that many items are returned.
- If a condition is specified, all items are returned that initially meet the condition. Once an item fails the condition, no further items are considered.

Examples

```
List.Last({1, 2, 3}) equals 3
```

10.2.9 List.LastN

Returns the last set of items in a list by specifying how many items to return or a qualifying condition.

List.LastN(list as list, optional countOrCondition as any) as any

Arguments

Argument	Description
list	The List to check.

optional countOrCondition	Number items to return or a qualifying condition.

Remarks

- If a number is specified, up to that many items are returned.
- If a condition is specified, all items are returned that initially meet the condition.
- Once an item fails the condition, no further items are considered

Example

```
List.LastN({3, 4, 5, -1, 7, 8, 2},1)  equals  { 2 }
```

10.2.10 List.MatchesAll

Returns true if all items in a list meet a condition.

List.MatchesAll(list as list, condition as Function) as logical

Arguments

Argument	Description
list	The List to check.
condition	The condition to qualify against.

Examples

```
List.MatchesAll({2, 4, 6}, each Number.Mod(_,2) = 0) equals true
List.MatchesAll({2, 4, 5}, each Number.Mod(_,2) = 0) equals false
```

10.2.11 List.MatchesAny

Returns true if any item in a list meets a condition.

List.MatchesAny(list as list, condition as Function) as logical

Arguments

Argument	Description
list	The List to check.
condition	The condition to qualify against.

Examples

```
List.MatchesAny({2, 4, 6}, each Number.Mod(_, 2) = 0) equals true
List.MatchesAny({1, 3, 5}, each Number.Mod(_, 2) = 0) equals false
```

10.2.12 List.Positions

Returns a list of positions for an input list.

List.Positions(list as list) as list

Arguments

Argument	Description
list	The List to check.

Remarks

When using `List.Transform` to modify a list, the list of positions can be used to give the transform access to the positions.

Examples

```
List.Positions({4, 5, 6}) equals {0, 1, 2}
```

10.2.13 List.Range

Returns a count items starting at an offset.

List.Range(list as list, offset as number, optional count as number) as list

Arguments

Argument	Description
list	The List to check.
offset	The index to start at.
optional count	Count of items to return.

Examples

```
List.Range({1..10}, 3, 5)  equals  {4, 5, 6, 7, 8}
```

10.2.14 List.Single

Returns the single item of the list or throws an Expression.Error if the list has more than one item.

List.Single(list as list) as any

Arguments

Argument	Description

list	The List to check.

Examples

```
List.Single({1}) equals 1
List.Single({1, 2, 3}) equals error
```

10.2.15 List.SingleOrDefault

Returns a single item from a list.

List.SingleOrDefault(list as list, optional default as any) as any

Arguments

Argument	Description
List	The List to check.
optional default	Specifies a default value to be returned.

Remarks

- If `list` is empty, a default value is returned instead.
- If `default` is not specified, the default value of null is assumed. If `list` has more than one item, an error is returned.

Examples

```
List.SingleOrDefault({1}) equals 1
List.SingleOrDefault({1, 2, 3}) equals error  List.SingleOrDefault({},
0) equals 0
```

10.2.16 List.Skip

Skips the first item of the list. Given an empty list, it returns an empty list. This function takes an optional parameter countOrCondition to support skipping multiple values..

List.Skip(list as list, optional countOrCondition as any) as list

Arguments

Argument	Description
list	The List to check.

optional countOrCondition	An optional parameter that skips multiple values.

Remarks

- If a number is specified, up to that many items are skipped.
- If a condition is specified, all items that meet the condition are skipped. Once an item fails the condition, no further items are considered.
- If this parameter is null, the default behavior is observed.

Examples

```
List.Skip({3, 4, 5, -1, 7, 8, 2}) equals {4, 5, -1, 7, 8, 2}
List.Skip({}) equals {}
List.Skip({3, 4, 5, -1, 7, 8, 2}, 5) equals {8, 2}
List.Skip({3, 4, 5}, 5) equals {}
List.Skip({3, 4, 5, -1, 7, 8, 2}, each _ > 0) equals {-1, 7, 8, 2}
List.Skip({3, 4, 5}, each _ > 0) equals {}
List.Skip({3, 4, 5, -1, 7, 8, 2}, null) equals {4, 5, -1, 7, 8, 2}
```

10.2.17 List.Buffer

Buffers the list in memory. The result of this call is a stable list, which means it will have a determinimic count, and order of items.

List.Buffer(list as list) as list

Arguments

Argument	Description
list	The List to buffer.

Example

```
List.Buffer(Sql:Database("localhost","northwind")[Customers]) equals
stable copy of table Customers
```

10.2.18 List.FindText

Searches a list of values, including record fields, for a text value.

List.FindText(list as list, text as text) as list

Arguments

Argument	Description
list	The List to search.

text	The value to search for.

Example

```
List.FindText(
{
    [ a = "t1", b = "t2" ],
    [ a = "test1", b = "test2" ],
    [ a = "another test", b = 5 ],
    [ a = 1, b =
2 ] }, "test")
equals
{
    [ a = "test1", b = "test2" ],
    [ a = "another test", b = 5 ]
}
```

10.3 Transformation functions

10.3.1 List.Accumulate

Accumulates a result from the list. Starting from the initial value seed this function applies the **accumulator** function and returns the final result.

List.Accumulate(list as list, seed as any, accumulator as function)as any

Arguments

Argument	Description
list	The List to check.
seed	The initial value seed.
accumulator	The value accumulator function.

Example

```
// This accumulates the sum of the numbers in the list provided.
List.Accumulate({1, 2, 3, 4, 5}, 0, (state, current) => state + current)
equals 15
```

10.3.2 List.Combine

Merges a list of lists into single list.

List.Combine(list as list) as list

Arguments

Argument	Description
list	The List of lists to merge.

Example

```
List.Combine({ {1, 2, 3, 4}, {5, 6, 7}, {8, 9} }) equals {1, 2, 3, 4, 5, 6, 7, 8, 9}
```

10.3.3 List.Transform

Performs the function on each item in the list and returns the new list.

List.Transform(list as list, transform as function) as list

Arguments

Argument	Description
list	The List to modify.
transform	The transform to apply.

Examples

```
List.Transform({1, 2}, each _ + 1) equals { 2, 3 }
```

10.3.4 List.TransformMany

Returns a list whose elements are projected from the input list.

List.TransformMany(list as list, collectionTransform as Function, resultTransform as Function) as list

Arguments

Argument	Description
list	The List to modify.
collectionTransform	The collectionTransform function is applied to each element, and the resultTransform function is invoked to construct the resulting list. The collectionSelector has the signature (x as any) => … where x is an element in list.

resultTransform	The resultTransform projects the shape of the result and has the signature (x as any, y as any) => … where x is the element in list and y is the element obtained by applying the collectionTransform to that element.

Example

```
List.TransformMany({1, 2}, (value) => {value + 1}, (oldValue, newValue)
=> oldValue * newValue) equals { 2, 6 }
```

10.3.5 List.Repeat

Returns a list that repeats the contents of an input list count times.

List.Repeat(list as list, count as number) as list

Arguments

Argument	Description
list	The List to repeat.
count	The number of times to repeat.

Examples

```
List.Repeat({1, 2, 3}, 3) equals {1, 2, 3, 1, 2, 3, 1, 2, 3}
```

10.3.6 List.ReplaceRange

Returns a list that replaces count values in a list with a replaceWith list starting at an index.

List.ReplaceRange(list as list, index as number, count as number, replaceWith as list) as list

Arguments

Argument	Description
list	The List to modify.
index	The index to start at.
count	The number of values to replace.
replaceWith	The value to replace with.

Example

```
List.ReplaceRange({1, 2, 7, 8, 9, 5}, 2, 3, {3, 4}) equals {1, 2, 3, 4,
5}
```

10.3.7 List.RemoveRange

Returns a list that removes count items starting at offset. The default count is 1.

List.RemoveRange(list as list, offset as number, optional count as nullable number) as list

Arguments

Argument	Description
list	The List to remove items from.
offset	The index to start at.
optional count	The number of items to remove.

Examples

```
List.RemoveRange({"A", "B", "C", "D"}, 2) equals {"A", "B", "D"}
List.RemoveRange({"A", "B", "C", "D"}, 1, 2) equals {"A", "D"}
```

10.3.1 List.RemoveFirstN

Returns a list with the specified number of elements removed from the list starting at the first element. The number of elements removed depends on the optional countOrCondition parameter.

List.RemoveFirstN(table as table, optional countOrCondition as any) as table

Arguments

Argument	Description
list	The List to remove items from.
optional countOrCondition	Optional number of elements or condition to remove elements, default is 1

Remarks

- If countOrCondidtion is omitted only the first element is removed
- If countOrCondidtion is a number, that many elements (starting from the top) will be removed)
- If countOrCondidtion is a condition, the elements that meet the condition will be removed until an element does not meet the condition

Examples

```
List.RemoveFirstN
    (
    {1, 2, 3, 4, 5},
    3
    ) equals
{4, 5}

List.RemoveFirstN
   (
   {5, 4, 2, 6, 1},  each
   _ > 3
   ) equals { 2,
6, 1}
```

10.3.1 List.RemoveLastN

Returns a list with the specified number of elements removed from the list starting at the last element. The number of elements removed depends on the optional countOrCondition parameter.

List.RemoveRange(list as list, offset as number, optional count as nullable number) as list

Arguments

Argument	Description
list	The List to remove items from.
optional countOrCondition	Optional number of elements or condition to remove elements, default is 1

Remarks

- If countOrCondidtion is omitted only the first element is removed
- If countOrCondidtion is a number, that many elements (starting from the top) will be removed)
- If countOrCondidtion is a condition, the elements that meet the condition will be removed until an element does not meet the condition

Examples

```
List.RemoveLastN
    (
```

```
    {1, 2, 3, 4, 5},
    3
    ) equals
{1, 2}

List.RemoveLastN
   (
   {5, 4, 2, 6, 4},  each
   _ > 3
   ) equals {5,
4, 2}
```

10.3.2 List.Reverse

Returns a list that reverses the items in a list.

List.Reverse(list as list) as list

Arguments

Argument	Description
list	The List to modify.

Examples

```
List.Reverse({1, 2, 3, 4, 5}) equals {5, 4, 3, 2, 1}
```

10.3.3 List.RemoveMatchingItems

Removes all occurrences of the given values in the list.

List.RemoveMatchingItems(list as list, values as list, optional equationCriteria as any) as list

Arguments

Argument	Description
list	The List to modify.
values	The list of values to remove.
optional equationCriteria	An optional equation criteria value to control equality comparison. For more information about the equationCriteria, see Parameter Values.

Example

```
List.RemoveMatchingItems ({"A", "B", "C", "B", "A"}, {"A", "C"}) equals
{"B", "B"}
```

10.3.4 List.RemoveItems

Removes items from list1 that are present in list2, and returns a new list.

List.RemoveItems(list1 as list, list2 as list) as list

Arguments

Argument	Description
list1	The List to modify.
list2	The list of items to remove.

Example

```
List.RemoveItems({1, 2, 3, 3}, {3}) equals { 1, 2}
```

10.3.5 List.RemoveNulls

Removes null values from a list.

List.RemoveNulls(list as list) as list

Arguments

Argument	Description
list	The List to modify.

Example

```
List.RemoveNulls({1, null, 2}) equals {1, 2}
```

10.3.6 List.ReplaceMatchingItems

Replaces occurrences of existing values in the list with new values using the provided equationCriteria. Old and new values are provided by the replacements parameters. An optional equation criteria value can be specified to control equality comparisons. For details of replacement operations and equation criteria, see Parameter Values..

List.ReplaceMatchingItems(list as list, replacements as any ,optional equationCriteria as any) as list

Arguments

Argument	Description

list	The List to modify.
replacements	The replacements to make.
optional equationCriteria	An optional equation criteria value to control equality testing.

Examples

```
List.ReplaceMatchingItems ({1, 2, 3, 4, 5}, {{2, -2}}) equals { 1, -2,
3, 4, 5}
List.ReplaceMatchingItems ({1, 2, 3, 4, 5}, {{2, -2}, {3, -3}}) equals {
1, -2, -3, 4, 5}
```

10.3.7 List.ReplaceValue

Searches a list of values for the value and replaces each occurrence with the replacement value.

List.ReplaceValue(list as list, oldValue as any, newValue as any, replacer as function) as list

Arguments

Argument	Description
list	The List to modify.
oldValue	The value to replace.
newValue	The new value to replace with.
replacer	A function provided as replacer determines the kind of values that are being replaced. Built-in functions be be used such as Replacer.ReplaceText and Replacer.ReplaceValue.

Examples

```
List.ReplaceValue({"a", "B", "a", "a"}, "a", "A", Replacer.ReplaceText)
equals {"A", "B", "A", "A"}
```

10.4 Membership functions

Since all values can be tested for equality, these functions can operate over heterogeneous lists.

10.4.1 List.Contains

Returns true if a value is found in a list.

List.Contains(list as list, value as any, optional equationCriteria as any) as logical

Arguments

Argument	Description
list	The List to check.
value	The value to check for.
optional equationCriteria	An optional equation criteria value to control equality testing.

Examples

```
List.Contains({1, 2, 3}, 2) equals true
List.Contains({1, 2, 3}, 4) equals false
```

10.4.2 List.ContainsAll

Returns true if all items in values are found in a list.

List.ContainsAll(list as list, values as list,optional equationCriteria as any) as logical

Arguments

Argument	Description
list	The List to check.
values	The list of values to check for.
optional equationCriteria	An optional equation criteria value to control equality comparisons. For more information about equality comparisons, see Parameter Values.

Examples

```
List.ContainsAll({1, 2, 3}, {2, 3}) equals true
List.ContainsAll({1, 2, 3}, {2, 4}) equals false
```

10.4.3 List.ContainsAny

Returns true if any item in values is found in a list.

List.ContainsAny(list as list, values as list,optional equationCriteria as any) as logical

Arguments

Argument	Description
list	The List to check.

values	The list of values to check for.
optional equationCriteria	An optional equation criteria value to control equality comparisons. For more information about equality comparisons, see Parameter Values.

Examples

```
List.ContainsAny({1, 2, 3}, {2, 4}) equals true
List.ContainsAny({1, 2, 3}, {4, 5}) equals false
```

10.4.4 List.PositionOf

Finds the first occurrence of a value in a list and returns its position.

List.PositionOf(list as list, value as any, optional occurrence as nullable number,optional equationCriteria as any) as any

Arguments

Argument	Description
list	The List to check.
value	The value to check for.
optional occurrence	An enum that controls the scope of operation.
optional equationCriteria	An optional equation criteria value to control equality comparisons. For more information about equality comparisons, see Parameter Values.

Occurrence settings

SETTING	DESCRIPTION
Occurrence.First and Occurrence.Last	Returns a single position.
Occurrence.All	Returns a list of positions with all occurrences.

Remarks

If the value is not found in the list, -1 is returned

Examples

```
List.PositionOf({"A", "B", "C", "D"}, "C") equals 2
List.PositionOf({"A", "B", "C", "B", "A"}, "A", Occurrence.First)
equals 0
```

```
List.PositionOf({"A", "B", "C", "B", "A"}, "A", Occurrence.Last) equals
4
List.PositionOf({"A", "B", "C", "B", "A"}, "A", Occurrence.All) equals
{0, 4}
```

10.4.5 List.PositionOfAny

Finds the first occurrence of any value in values and returns its position.

List.PositionOfAny(list as list, values as list, optional occurrence as nullable number, optional equationCriteria as any) as any

Arguments

Argument	Description
list	The List to check.
values	The list of values to check for.
optional occurrence	An enum that controls the scope of operation.
optional equationCriteria	An optional equation criteria value to control equality comparisons. For more information about equality comparisons, see Parameter Values .

Occurrence settings

SETTING	DESCRIPTION
Occurrence.First and Occurrence.Last	Returns a single position.
Occurrence.All	Returns a list of positions with all occurrences.

Remarks

If the value is not found in the list, -1 is returned

Examples

```
List.PositionOfAny({"A", "B", "C", "D"}, {"B", "C"}) equals 1
```

```
List.PositionOfAny({"A", "B", "C", "B", "A"}, {"A", "B"},
Occurrence.First) equals 0
List.PositionOfAny({"A", "B", "C", "B", "A"}, {"A", "B"},
Occurrence.Last) equals 4
List.PositionOfAny({"A", "B", "C", "B", "A"}, {"A", "B"},
Occurrence.All) equals {0, 1, 3, 4}
```

10.4.6 List.AnyTrue

Returns true if any expression in a list in true

List.AnyTrue(list as list) as logical

Arguments

Argument	Description
list	The List to check.

Example

```
List.AnyTrue({2=0, false, 1 < 0 }) equals false
```

10.4.7 List.AllTrue

Returns true if all expressions in a list are true

List.AllTrue(list as list) as logical

Arguments

Argument	Description
list	The List to check.

Example

```
List.AllTrue({true, 2=2}) equals true
```

10.5 Set operations

10.5.1 List.Difference

Returns the items in list 1 that do not appear in list 2. Duplicate values are supported.

List.Difference(list1 as list, list2 as list,optional equationCriteria as any) as list

Arguments

Argument	Description
list1	The List to check with.

list2	The List to check against.
optional equationCriteria	An optional equation criteria value to control equality comparisons. For more information about equality comparisons, see Parameter Values.

Examples

```
List.Difference({1..10}, {2..3,5..7}) equals {1,4,8,9,10}
List.Difference({1}, {1,2,3}) equals {}
List.Difference({1, 1, 1}, {1}) equals {1, 1}
```

10.5.2 List.Intersect

Returns a list from a list of lists and intersects common items in individual lists. Duplicate values are supported.

List.Intersect(list as list /* { List } */,optional equationCriteria as any) as list

Arguments

Argument	Description
list	The List of lists to check.
optional	An optional equation criteria value to control equality comparisons. For
equationCriteria	more information about equality comparisons, see Parameter Values.

Remarks

If nothing is common in all lists, an empty list is returned.

Examples

```
List.Intersect({ {1..5}, {2..6}, {3..7} }) equals {3..5}
List.Intersect({ {1..5}, {4..8}, {7..11} }) equals {}
List.Intersect({ {1, 1, 1, 2}, {1, 1, 2, 2} }) equals {1, 1, 2}
```

10.5.3 List.Union

Returns a list from a list of lists and unions the items in the individual lists. The returned list contains all items in any input lists. Duplicate values are matched as part of the Union.

List.Union(list as list,optional equationCriteria as any) as list

Arguments

Argument	Description
list	The List of lists to check.
optional equationCriteria	An optional equation criteria value to control equality comparisons. For more information about equality comparisons, see Parameter Values.

Examples

```
List.Union({ {1..5}, {2..6}, {3..7} }) equals {1..7}
List.Union({ {1..5}, {4..8}, {7..11} }) equals {1..11}
List.Union({ {1, 1, 1, 2}, {1, 1, 2, 2} }) equals {1, 1, 1, 2, 2}
```

10.6 Ordering

Ordering functions perform comparisons. All values that are compared must be comparable with each other. This means they must all come from the same datatype (or include null, which always compares smallest). Otherwise, an Expression.Error is thrown.

Comparable data types

- Number
- Duration
- DateTime
- Text
- Logical
- Null

10.6.1 List.Sort

Returns a sorted list using comparison criterion.

List.Sort(list as list, optional comparisonCriteria as any) as list

Arguments

Argument	Description
list	The List to modify.
optional comparisonCriteria	Controls the sort order. For more information about equality comparisons, see Parameter Values.

Remarks

- To control the order, comparison criterion can be an Order enum value.
- To compute a key to be used for sorting, a function with one argument can be used.
- To both select a key and control order, comparison criterion can be a list containing the key and order.

- To completely control the comparison, a function with two arguments can be used that returns -1, 0, or 1 given the relationship between the left and right inputs. `Value.Compare` is a method that can be used to delegate this logic.

Examples

```
List.Sort({2, 1}) equals {1, 2}
List.Sort({1, 2}, Order.Descending) equals {2, 1}
List.Sort({1, 2}, Order.Ascending) equals {1, 2}
List.Sort({1, 2}, each 1/_) equals{2, 1}
List.Sort({2, 1}, {each 1/_, Order.Descending}) equals {1, 2}
List.Sort({1, 2}, (x, y) => Value.Compare(1/x, 1/y)) equals {2, 1}
```

10.6.2 List.Max

Returns the maximum item in a list, or the optional default value if the list is empty.

List.Max(list as list, optional default as any, optional comparisonCriteria as any, optional includeNulls as nullable logical) as any

Arguments

Argument	Description
list	The List to check.
optional default	The default value to return.
optional comparisonCriteria	An optional comparison criteria value to control equality testing. If this argument is null, the default comparer is used.
optional includeNulls	The Logical value whether or not to include null values in the return list.

Example

```
List.Max({1, 4, 7, 3, -2, 5}, 1) equals 7
```

10.6.3 List.MaxN

Returns the maximum values in the list. After the rows are sorted, optional parameters may be specified to further filter the result

List.MaxN(list as list, countOrCondition as any, optional comparisonCriteria as any, optional includeNulls as nullable logical) as list

Arguments

Argument	Description
list	The List to check.
countOrCondition	Specifies the number of values to return or a filtering condition.
optional comparisonCriteria	Specifies how to compare values in the list.
optional includeNulls	The Logical value whether or not to include null values in the return list.

Example

```
List.MaxN({3, 4, 5, -1, 7, 8, 2}, 5) equals {8, 7, 5, 4, 3}
```

10.6.4 List.Min

Returns the minimum item in a list, or the optional default value if the list is empty.

List.Min(list as list, optional default as any, optional comparisonCriteria as any, optional includeNulls as nullable logical) as any

Arguments

Argument	Description
list	The List to check.
optional default	The default value to return.
optional comparisonCriteria	Specifies how to compare values in the list. If this argument is null, the default comparer is used.
optional includeNulls	The Logical value whether or not to include null values in the return list.

Example

```
List.Min({1, 4, 7, 3, -2, 5}) equals -2
```

10.6.5 List.MinN

Returns the minimum values in a list.

List.MinN(list as list, countOrCondition as any, optional comparisonCriteria as any, optional includeNulls as nullable logical) as list

Arguments

Argument	Description
list	The List to check.
countOrCondition	Specifies the number of values to return or a filtering condition.
optional comparisonCriteria	Specifies how to compare values in the list.
optional includeNulls	The Logical value whether or not to include null values in the return list.

Example

```
List.MinN({3, 4, 5, -1, 7, 8, 2}, 5) equals {-1, 2, 3, 4, 5}
```

10.6.6 List.Median

Returns the median item from a list.

List.Median(values as list, comparisonCriteria as any) as any

Arguments

Argument	Description
values	The List to check.
comparisonCriteria	The comparison criteria to use.

Remarks

- If the list is empty, an Expression.Error is thrown.
- If there is an even number of items, the function chooses the smaller of the two median items.

Example

```
List.Median({5, 3, 1, 7, 9}) equals 5
```

10.7 Averages

These functions operate over homogeneous lists of Numbers, DateTimes, and Durations.

10.7.1 List.Average

Returns an average value from a list in the datatype of the values in the list.

List.Average(list as list) as any

Arguments

Argument	Description
list	The List to check.

Remarks

If the list is empty, an Expression.Error is thrown.

Examples

```
List.Average({1, 2, 3}) equals 2
List.Average({#duration(0, 0, 30, 0), #duration(0, 0, 40, 0)}) equals
#duration(0, 0, 35, 0)
List.Average({#date(2011,1,1), #date(2011,1,2), #date(2011,1,3)})
equals #datetime(2011,1,2) List.Average({}) equals null
```

10.7.2 List.StandardDeviation

Returns the standard deviation from a list of values. List.StandardDeviation performs a sample based estimate. The result is a number for numbers, and a duration for DateTimes and Durations.

List.StandardDeviation(list as list) as any

Arguments

Argument	Description
list	The List to check.

Remarks

- If the list is empty, an Expression.Error is thrown.

Example

```
List.StandardDeviation({1..5}) equals 1.5811388300841898
```

10.7.3 List.Mode

Returns an item that appears most commonly in a list.

List.Mode(list as list, optional equationCriteria as any)as any

Arguments

Argument	Description
list	The List to check.
optional equationCriteria	Controls the sort order. For more information about equality comparisons, see Parameter Values.

Remarks

- If more than 1 item appears with the same maximum frequency, the last item in the first appearance order is chosen.
- If the list is empty, an Expression.Error is thrown.

Examples

```
List.Mode({"A", 1, 4, 5, 2, "B", 3, 5, 5, 4, 4}) equals 5
```

10.7.4 List.Modes

Returns all items that appear with the same maximum frequency.

List.Modes(list as list, optional equationCriteria as any)as list

Arguments

Argument	Description
list	The List to check.
optional equationCriteria	Controls the sort order. For more information about equality comparisons, see Parameter Values.

Remarks

- If the list is empty, an Expression.Error is thrown.

Examples

```
List.Modes({"A", 1, 4, 5, 2, "B", 3, 5, 5, "A", 4, 4, "A"}) equals {"A",
4, 5}
```

10.8 Addition

These functions work over homogeneous lists of Numbers or Durations.

10.8.1 List.Sum

Returns the sum from a list.

List.Sum(list as list) as any

Arguments

Argument	Description
list	The List to check.

Remarks

If the list is empty, an Expression.Error is thrown.

Examples

```
List.Sum({1, 2, 3}) equals 6
List.Sum({#duration(0, 0, 0, 15), #duration(0, 0, 0, 30)}) equals
#duration(0, 0, 0, 45)
List.Sum({}) equals error
```

10.9 Numerics

These functions only work over numbers.

10.9.1 List.Covariance

Returns the covariance from two lists as a number.

List.Covariance(list1 as list, list2 as list) as number

Arguments

Argument	Description
list1	The List to compare with.
list2	The List to compare against.

10.9.2 List.Product

Returns the product from a list of numbers.

List.Product(list as list) as number

Arguments

Argument	Description

list	The List to check.

Remarks

If the list is empty, an Expression.Error is thrown.

Examples

```
List.Product({2, 3, 4}) equals 24 List.Product({})
equals error
```

10.10 Generators

These functions generate list of values.

10.10.1 List.DateTimes

Returns a list of datetime values from size count, starting at start and adds an increment to every value.

List.DateTimes(start as datetime, count as number, increment as duration) as { datetime }

Arguments

Argument	Description
start	The DateTime to start at.
count	The number of values to return.
increment	The increment to add.

Example

```
List.DateTimes(#time(10, 0, 0), #duration(0, 1, 0, 0), 5) equals
{#time(10, 0, 0), #time(11, 0, 0), #time(12, 0, 0),  #time(13, 0, 0),
#time(14, 0, 0) }
```

10.10.2 List.Dates

Returns a list of date values from size count, starting at start and adds an increment to every value.

List.Dates(start as date, count as number, increment as duration) as { datetime }

Arguments

Argument	Description
start	The Date to start at.
count	The number of values to return.
increment	The increment to add.

10.10.3 List.DateTimeZones

Returns a list of of datetimezone values from size count, starting at start and adds an increment to every value.

List.DateTimeZones(start as datetimezone, count as number, increment as duration) as { datetime }

Arguments

Argument	Description
start	The DateTimeZone to start at.
count	The number of values to return.
increment	The increment to add.

10.10.4 List.Durations

Returns a list of durations values from size count, starting at start and adds an increment to every value.

List.Durations(start as duration, count as number, increment as duration) as { duration }

Arguments

Argument	Description
start	The Duration to start at.
count	The number of values to return.
increment	The increment to add.

Examples

```
List.Durations(#duration(0, 1, 0, 0), #duration(0, 1, 0, 0), 5) equals
{#duration(1, 0, 0), #duration(2, 0, 0), #duration(3, 0, 0) ,
  #duration(4,0,0),#duration(5,0,0)}
```

10.10.5 List.Generate

Generates a list from a value function, a condition function, a next function, and an optional transformation function on the values.

List.Generate(start as () as any, condition as (item as any) as logical, next as (item as any) as any, optional transformer as (item as any) as any) as list

Arguments

Argument	Description
start	A function that provides the initial value
condition	A condition which controls the sequence of generation
next	A function that generates the next value for an iteration
transfomer	A transformation function to apply to each item in the list

Examples

```
List.Generate(()=>0, each _ < 10, each _ + 1) equals { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 }
List.Generate(()=>[A=1,B=1], each [A] < 5, each [A=[A]+1, B=[B]+1], each [A]+[B]) equals { 2, 4, 5, 8 }
```

10.10.6 List.Numbers

Returns a list of numbers from size count starting at initial, and adds an increment. The increment defaults to 1.

List.Numbers(start as number, count as number, optional increment as nullable number) as { Number }

Arguments

Argument	Description
start	The first number in the sequence.
count	How many numbers to return.
optional increment	The number to increment each number.

Examples

```
List.Numbers(1, 5) equals {1, 2, 3, 4, 5}
List.Numbers(1, 8, 3) equals {1, 4, 7, 10, 13, 16, 19, 22}
```

10.10.7 List.Random

Returns a list of count random numbers, with an optional seed parameter.

List.Random(count as number, optional seed as `nullable` number) as { Number }

Arguments

Argument	Description
count	How many random numbers to return.
optional seed	Randomization seed value.

Example

```
List.Random(10) equals { 0.44298228502412434, 0.11142372065755712,
0.81061893087374925, 0.69705957299892773, 0.84984056970562816,
0.45717397865707704, 0.27344677656583805, 0.51387371612427468,
0.14493200795023331, 0.89694489161341684 }
```

10.11 Parameter values

10.11.1 Occurrence

specification

```
Occurrence.First = 0;
```

- `Occurrence.Last = 1;`
- `Occurrence.All = 2;`

10.11.2 Sort order

- `Order.Ascending = 0;  Order.Descending = 1;`

10.11.3 Equation criteria

Equation criteria for list values can be specified as either a

- A function value that is either ○ A key selector that determines the value in the list to apply the equality criteria, or ○ A comparer function that is used to specify the kind of comparison to apply. Built in comparer functions can be specified, see section for Comparer functions.
- A list value which has ○ Exactly two items ○ The first element is the key selector as specified above ○ The second element is a comparer as specified above.

For more information and examples, see List.Distinct.

10.11.4 Comparison criteria

Comparison criterion can be provided as either of the following values:

- A number value to specify a sort order. For more inforarmtion, see sort order in Parameter values.
- To compute a key to be used for sorting, a function of 1 argument can be used.
- To both select a key and control order, comparison criterion can be a list containing the key and order.
- To completely control the comparison, a function of 2 arguments can be used that returns 1, 0, or 1 given the relationship between the left and right inputs. `Value.Compare` is a method that can be used to delegate this logic.

For more information and examples, see List.Sort.

10.11.5 Replacement operations

Replacement operations are specified by a list value, each item of this list must be

- A list value of exactly two items
- Fist item is the old value in the list, to be replaced
- Second item is the new which should replace all occurrences of the old value in the list

11. Table

These functions deal with Table values.

11.1 Table construction

11.1.1 Table.FromRows

Creates a table from the list where each element of the list is a list that contains the column values for a single row.

Table.FromRows(rows as list, optional columns as any) as table

Arguments

Argument	Description
rows	The List to convert.
optional columns	An optional list of column names, or a table type.

Examples

```
Table.FromRows({ {1, 3}, {2, 4}}, {"a", "b"}) equals Table.FromRecords(
{ [ a = 1, b = 3], [ a = 2, b = 4 ] })
```

11.1.2 Table.FromList

Converts a list into a table by applying the specified splitting function to each item in the list.

Table.FromList(list as list, optional splitter as nullable function, optional columns as any, optional default as any, optional extraValues as any) as table

Arguments

Argument	Description
list	The List to convert.
optional splitter	Splitter function.
optional columns	A list of text values specifying the column names of the resulting table.
optional default	A default can be provided to be used for missing values in the table.
optional extraValues	Extra values for each item in the list.

Example

```
Table.FromList({[A=1, B=null], [A="2", B=2]}, Record.FieldValues, {"A",
"B"}) equals Table.FromRecords({ [ A = 1, B = null], [ A = "2", B = 2
]})
```

11.1.3 Table.FromColumns

Returns a table from a list containing nested lists with the column names and values.

Table.FromColumns(lists as list, optional columns as any) as table

Arguments

Argument	Description
lists	The List of lists to convert.

optional columns	Optional parameter to provide names and types for the columns.

Remarks

If some columns have more values then others, the missing values will be filled with the default value, 'null', if the columns are nullable.

Examples

```
 Table.FromColumns({ {1, 2, 3}, {4, 5}, {6, 7, 8, 9} }) equals
Table.FromRecords({[Column1 = 1, Column2 = 4, Column3 = 6], [Column1 =
2, Column2 = 5, Column3 = 7 ], [Column1 = 3, Column2 = null, Column3 =
8], [Column1 = null, Column2 = null, Column3 = 9]})
```

```
Table.FromColumns({ {1, 2, 3}, {4, 5}, {6, 7, 8, 9} }, {“a”, “b”,
“c”}) equals   Table.FromRecords({ [
    //    a = 1,
    //    b = 4,
    //    c = 6
    //    ], [
    //    a = 2,
    //    b = 5,
    //    c = 7
    //    ], [
    //    a = 3,
    //    b = null,
    //    c = 8
    //    ], [
    //    a = null,
    //    b = null,
    //    c = 9
    //    ]})
```

11.1.4 Table.FromRecords

Returns a table from a list of records.

Table.FromRecords(records as list, optional columns as any) as table

Arguments

Argument	Description
records	The List to convert.
optional columns	An optional list of column names, or a table type could be provided as the second argument, which much match names found in the records. Record fields which don't appear in the list are ignored.

Example

```
Table.FromRecords({ [a=1, b=2], [a=null, b = 3] }) equals { [a = 1, b = 2], [ a = null, b = 3 ] }
Table.FromRecords({ [a=1,b=2],[a=null, b = 3] }, type table [a=number, b=text]) equals { [a = 1, b = 2], [ a = null, b = 3 ] }
```

11.1.5 Table.FromValue

Returns a table with a column containing the provided value or list of values.

Table.FromValue (value as any) as table

Arguments

Argument	Description
Value	The value to convert.

Examples

`Table.FromValue(1)` equals Table.FromRecords({ [Value = 1] })
`Table.FromValue({1,2})` equals Table.FromRecords({ [Value = 1], [Value = 2]})

11.2 Conversions

11.2.1 Table.ToColumns

Returns a list of nested lists each representing a column of values in the input table.

Table.ToColumns(table as table) as list

Arguments

Argument	Description
table	The Table to convert.

Example

`Table.ToColumns(Table.FromRecords({[A=1, B = 2],[A = 2, B =3]}))` equals {{ 1, 2 }, { 2, 3 }}

11.2.2 Table.ToList

Returns a table into a list by applying the specified combining function to each row of values in a table.

Table.ToList(table as table, optional combiner as nullable function) as list

Arguments

Argument	Description
table	The Table to convert.
optional combiner	The combiner function is applied to each row in the table to produce a single value for the row.

Examples

```
let  input =
    Table.FromRows({{"a","b","c"}},{"A","B","C"})
    in
       Table.ToList(input, Combiner.CombineTextByDelimiter("-"))
 equals {"a-b-c"}
```

11.2.3 Table.ToRecords

Returns a list of records from an input table.

Table.ToRecords(table as table) as list

Arguments

Argument	Description
table	The Table to convert.

Example

```
Table.ToRecords(Table.FromRows({{"1", "2"}},{"a", "b"})) equals {[a =
"1", b = "2"]}
```

11.2.4 Table.ToRows

Returns a nested list of row values from an input table.

Table.ToRows(table as table) as list

Arguments

Argument	Description
table	The Table to check.

Example

```
 Table.ToRows(Table.FromRecords({[A=1, B = 2],[A = 2, B =3]}))
equals {{1, 2}, {2, 3}}
```

11.3 Information

11.3.1 Table.IsEmpty

Returns true if the table does not contain any rows.

Table.IsEmpty(table as table) as logical

Arguments

Argument	Description
table	The Table to check.

Example

```
let  emptyTable = Table.FromRows({}),  tableValue
= Table.FromRows({{1, 2},{3, 4}},{"a", "b"}) in
[  test1 = Table.IsEmpty(emptyTable),  // true  test2 =
Table.IsEmpty(tableValue)  // false ]
```

11.3.2 Table.RowCount

Returns the number of rows in a table.

Table.RowCount(table as table) as number

Arguments

Argument	Description
table	The Table to check.

Examples

```
let   tableValue = Table.FromRows({{1,2}, {3,4}},
{"a","b"}) in
  Table.RowCount tableValue)  equals  2
```

11.3.3 Table.ColumnCount

Returns the number of columns in a table.

Table.ColumnCount(table as table) as number

Arguments

Argument	Description
table	The Table to check.

Example

```
let
tableValu
e =
Table.Fro
mRows({{1
,2},
{3,4}},
{"a","b"}
) in
  Table.ColumnCount(tableValue) equals 2
```

11.3.4 Table.PartitionValues

Returns information about how a table is partitioned.

Table.PartitionValues = (table as table) as table;

Arguments

Argument	Description
table	The input table.

Remarks

A table is returned where each column is a partition column in the original table, and each row corresponds to a partition in the original table.

11.4 Row operations

11.4.1 Table.AlternateRows

Returns a table containing an alternating pattern of the rows from a table.

Table.AlternateRows(table as table, offset as number, skip as number, take as number) as table

Remarks

Table.AlternateRows is similar to `List.Alternate` but requires a table as input.

Example

```
Table.AlternateRows(Table.FromRecords({[A=1], [A=3], [A=5]}), 1, 1, 1)
equals Table.FromRecords({[A=1], [A=3]})
```

11.4.2 Table.Combine

Returns a table that is the result of merging a list of tables. The tables must all have the same row type structure.

Table.Combine(tables as list) as table

Arguments

Argument	Description
tables	The List of tables to combine.

Example

```
Table.Combine({Table.FromRecords({[A = 1], [A = 2], [A = 3]}),
Table.FromRecords({[A = 4], [A = 5], [A = 6]}), Table.FromRecords({[A =
7], [A = 8], [A = 9]})}) equals Table.FromRecords ({[A = 1], [A = 2], [A
= 3], [A = 4], [A = 5], [A = 6], [A = 7], [A = 8], [A = 9]})
```

11.4.3 Table.FirstN

Returns the first row(s) of a table, depending on the countOrCondition parameter.

Table.FirstN(table as table, optional countOrCondition as any) as table

Arguments

Argument	Description

Table	The Table to check.
optional countOrCondition	Depending on the type, more than one row will be returned.

Remarks

- If `countOrCondition` is a number, many rows (starting at the top) will be returned.
- If `countOrCondition` is a condition, the rows that meet the condition will be returned until a row does not meet the condition.

Example

```
Table.FirstN(Table.FromRecords({[a = 1, b = 2], [a = 3, b = 4], [a = 5,
b = 6]}), 2) equals Table.FromRecords({ [a = 1, b = 2], [a = 3, b =
4] })
```

11.4.4 Table.First

Returns the first row from a table.

Table.First(table as table, optional default as any) as any

Arguments

Argument	Description
table	The Table to check.
optional default	Optional default value.

Remarks

- If the table is empty, **Table.First** returns null.

Example

`Table.First(Table.FromRecords({[a = 1, b = 2], [a = 3, b = 4]}))` equals [a = 1, b = 2]

11.4.5 Table.InsertRows

Returns a table with the list of rows inserted into the table at an index. Each row to insert must match the row type of the table..

Table.InsertRows(table as table, offset as number, rows as list) as table

Arguments

Argument	Description
table	The Table to insert rows into.
offset	The row number to insert at.
rows	The List of rows to insert.

Remark

- Table.InsertRows is similar to `List.InsertRange` but requires a table as input.

Example

```
Table.InsertRows(Table.FromRecords({[a = "A", b = "a"], [a = "B", b =
"b"]}), 2, {[a = "C", b = "c"]}) Equals Table.FromRecords({[a = "A", b =
"a"], [a = "B", b = "b"], [a = "C", b = "c"]})
```

11.4.6 Table.LastN

Returns the last row(s) from a table, depending on the **countOrCondition** parameter.

Table.LastN(table as table, countOrCondition as any) as table

Arguments

Argument	Description
table	The Table to check.
countOrCondition	Depending on the type, more than one row will be returned.

Remarks

- If countOrCondition is a number, that many rows will be returned starting from the end of the table.
- If countOrCondition is a condition, the rows that meet the condition will be returned in ascending position until a row does not meet the condition.

Example

```
Table.LastN(Table.FromRecords(
       {[a = 1, b = 2],
        [a = 3, b = 4],
        [a = 5, b = 6]}), 2) equals Table.FromRecords({[a
= 3, b = 4], [a = 5, b = 6]})
```

11.4.7 Table.Last

Returns the last row of a table.

Table.Last(table as table, optional default as) as any

Arguments

Argument	Description
table	The Table to check.
optional default	Optional default value.

Remark

- If the table is empty, Table.Last returns null.

Example

```
Table.Last(Table.FromRecords({[a = 1, b = 2], [a = 3, b = 4]})) equals
[a = 3, b = 4]
```

11.4.8 Table.MatchesAllRows

Returns true if all of the rows in a table meet a condition.

Table.MatchesAllRows(table as table, condition as function) as logical

Arguments

Argument	Description
table	The Table to check.
condition	The condition to match.

Remark

- Table.MatchesAllRows is similar to `List.MatchesAll` but requires a table argument.

Example

```
Table.MatchesAllRows(Table.FromRecords({[A=2], [A=4], [A=6]}), each
Number.Mod([A], 2) = 0) equals true
```

11.4.9 Table.MatchesAnyRows

Returns true if any of the rows in a table meet a condition.

Table.MatchesAnyRows(table as table, condition as function) as logical

Arguments

Argument	Description
table	The Table to check.
condition	The condition to match.

Remark

- Table.MatchesAnyRows is similar to `List.MatchesAny` but requires a table argument.

Example

```
Table.MatchesAnyRows(Table.FromRecords ({[A=2],[A=4],[A=5]}), each
Number.Mod([A], 2) = 0) equals true
```

11.4.10 Table.Partition

Partitions the table into a list of `groups` number of tables, based on the value of the `column` of each row and a `hash` function. The `hash` function is applied to the value of the `column` of a row to obtain a hash value for the row. The hash value modulo `groups` determines in which of the returned tables the row will be placed.

Table.Partition (table as table, column as text, groups as number, hash as function) as list

Arguments

Argument	Description
table	The Table to modify.
column	The column to apply hash to.
groups	The number of groups to make.
hash	The hash function to apply.

Example

```
Table.Partition(Table.FromRecords({[A=1], [A=2], [A=3], [A=4], [A=5],
[A=6]}),"A", 2, each _) equals Table.FromRecords({[A=2], [A=4], [A=6]})
```

11.4.11 Table.FromPartitions

Returns a table that is the result of combining a set of partitioned tables into new columns. The type of the column can optionally be specified, the default is any.

Table.FromPartitions (partitionColumn as text, partitions as list, optional partitionColumnType as nullable type) as table

Arguments

Argument	Description
partitionColumn	The name of the column where the values from the paritions will added.
partitions	The list of partitions to combine, specificed in {value, table} pairs.
Optional partitionColumnType	The type of the resulting column (default is any).

Example

```
Table.FromPartitions("Year", {{1994, Table.FromPartitions("Month", {{"Jan",
Table.FromPartitions("Day", {{1, #table({"Foo"},{{"Bar"}})}, {2,
#table({"Foo"},{{"Bar"}})})})}, {"Feb", Table.FromPartitions("Day", {{3,
#table({"Foo"},{{"Bar"}})}, {4, #table({"Foo"},{{"Bar"}})})})})})})
equals Table.FromRecords({ [Foo = "Bar", Day = 1, Month = "Jan", Year = 1994],
[Foo = "Bar", Day = 2, Month = "Jan", Year = 1994], [Foo = "Bar", Day = 3,
Month = "Feb", Year = 1994], [Foo = "Bar", Day = 4, Month = "Feb", Year =
1994]}, {"Foo", "Day", "Month", "Year"})
```

11.4.12 Table.Range

Returns the specified number of rows from a table starting at an offset.

Table.Range(table as table, offset as number, optional count as nullable number) as table

Arguments

Argument	Description
table	Source table.
Offset	Starting row offset.
optional count	Optional number of rows to return. If count is not provided, all rows are returned starting from offset.

Remark

- `Table.Range is similar to List.Range but requires a table as input.`

Example

```
Table.Range(Table.FromRecords({[A=1], [A=2], [A=3]}), 1) equals
Table.FromRecords({[A=2]})
```

11.4.13 Table.RemoveRows

Returns a table with the specified number of rows removed from the table starting at an offset.

Table.RemoveRows(table as table, offset as number, optional count as nullable number) as table

Arguments

Argument	Description
table	The Table to remove rows from.
offset	The row to start removal at.
optional count	Optional number of rows to remove. Default **count** is 1.

Remarks

- Table.RemoveRows is similar to `List.RemoveRange` but requires a table as input.

Examples

```
Table.RemoveRows(Table.FromRecords({[A=1], [A=2], [A=3]}), 1) equals
Table.FromRecords({[A=1], [A=3]})
```

11.4.14 Table.RemoveF irstN

Returns a table with the specified number of rows removed from the table starting at the first row. The number of rows removed depends on the optional countOrCondition parameter.

Table.RemoveFirstN(table as table, optional countOrCondition as any) as table

Arguments

Argument	Description
table	The Table to remove rows from.
optional countOrCondition	Optional number of rows or condition to remove rows, default is 1

Remarks

- If countOrCondidtion is omitted only the first row is removed

- If countOrCondidtion is a number, that many rows (starting from the top) will be removed)
- If countOrCondidtion is a condition, the rows that meet the condition will be removed until a rows does not meet the condition

Examples

```
Table.RemoveFirstN(
   Table.FromRecords({
      [a = 1, b = 2],
      [a = 3, b = 4],
      [a = 5, b = 6]}
      ),
   2)  equals  Table.FromRecords({[a = 5, b
= 6]}, {
    "a",
    "b"
})

Table.RemoveFirstN(
   Table.FromRecords({
      [a = 1, b = 2],
      [a = 3, b = 4],
      [a = -5, b = -6],
      [a = 7, b = 3]}
      ),  each _
   [a] > 0
   ) equals Table.FromRecords({[a = -5, b
= -6],
    [a = 7, b = 3]}, {
    "a",
    "b"
})
```

11.4.15 Table.RemoveLastN

Returns a table with the specified number of rows removed from the table starting at the last row.

The number of rows removed depends on the optional countOrCondition parameter.

Table.RemoveLastN(table as table, optional countOrCondition as any) as table

Arguments

Argument	Description
table	The Table to remove rows from.
optional countOrCondition	Optional number of rows or condition to remove rows, default is 1

Remarks

- If countOrCondidtion is omitted only the first row is removed
- If countOrCondidtion is a number, that many rows (starting from the top) will be removed)
- If countOrCondidtion is a condition, the rows that meet the condition will be removed until a rows does not meet the condition

Examples

```
Table.RemoveLastN(
   Table.FromRecords({
      [a = 1, b = 2],
      [a = 3, b = 4],
      [a = 11, b = 12],
      [a = 13, b = 14]
      }),
   1)
equals  Table.FromRecords({[a = 1, b = 2],
    [a = 3, b = 4],
    [a = 11, b = 12]}, {
    "a",
    "b"
})

Table.RemoveLastN(
  Table.FromRecords({
     [a = 1, b = 2],
     [a = 3, b = 4],
     [a = -5, b = -6],
     [a = 7, b = 3]
     }),  each _
   [a] > 0)
equals Table.FromRecords({[a = 1, b = 2],
    [a = 3, b = 4],
```

```
    [a = -5, b = -6]}, {
    "a",
    "b"
})
```

11.4.16 Table.RemoveRowsWithErrors

Returns a table with all rows removed from the table that contain an error in at least one of the cells in a row.

Table.RemoveRowsWithErrors(table as table, optional columns as nullable list) as table

Arguments

Argument	Description
table	The Table to modify.
optional columns	Only cells in the column list are inspected for errors.

Remarks

Only errors detected by directly accessing the cell are considered. Errors nested more deeply, such as a structured value in a cell, are ignored.

Examples

```
Table.RemoveRowsWithErrors(Table.FromRecords({[A=],[A=2], [A=3]}))
equals Table.FromRecords({[A=2], [A=3]})
```

11.4.17 Table.Repeat

Returns a table containing the rows of the table repeated the **count** number of times.

Table.Repeat(table as table, count as number) as table

Arguments

Argument	Description
table	The Table to check.
count	The number of times to repeat the table.

Example

```
Table.Repeat(Table.FromRecords({[A=1], [A=2]})2) equals
Table.FromRecords({[A=1], [A=2], [A=1], [A=2]})
```

11.4.18 Table.ReplaceRows

Returns a table where the rows beginning at an offset and continuing for count are replaced with the provided rows.

Table.ReplaceRows(table as table, offset as number, count as number, rows as list) as table

Arguments

Argument	Description
table	The Table to modify.
offset	The beginning row index.
count	The number of rows to replace.
rows	The List of replacement rows.

Remarks

- Table.ReplaceRows is similar to List.ReplaceRange but requires a table as input.

Example

```
Table.ReplaceRows(Table.FromRecords({[A=1], [A=2], [A=3], [A=4],
[A=5]}), 1, 3, {[A=6], [A=7]}) equals Table.FromRecords({[A=1], [A=6],
[A=7], [A=5]})
```

11.4.19 Table.ReverseRows

Returns a table with the rows in reverse order.

Table.ReverseRows(table as table) as table

Arguments

Argument	Description
table	The Table to modify.

Remarks

- Table.ReverseRows is similar to `List.Reverse` but requires a table as input.

Examples

```
Table.ReverseRows(Table.FromRecords({[A=1], [A=2], [A=3]})) equals
Table.FromRecords({[A=3], [A=2], [A=1]})
```

11.4.20 Table.SelectRows

Returns a table containing only the rows that match a condition.

Table.SelectRows(table as table, condition as function) as table

Arguments

Argument	Description
table	The Table to check.
condition	The condition to match.

Remarks

- Table. SelectRows is similar to `List. Select` but requires a table as input.

Examples

```
Table.SelectRows(Table.FromRecords({[A=1], [A=2], [A=3]}), each [A] > 2)
equals Table.FromRecords({[A=3]})
```

11.4.21 Table.SelectRowsWithErrors

Returns a table with only the rows from **table** that contain an error in at least one of the cells in a row.

Table.SelectRowsWithErrors(table as table, optional columns as nullable list) as table

Arguments

Argument	Description
table	The Table to check.
optional columns	Only cells in the column list are inspected for errors.

Remarks

- Only errors detected by directly accessing the cell are considered. Errors nested more deeply, such as a structured value in a cell, are ignored.

Examples

```
Table.SelectRowsWithErrors(Table.FromRecords({[A=...], [A=2], [A=3]}))
equals Table.FromRecords({[A=...]})
```

11.4.22 Table.SingleRow

Returns a single row from a table.

Table.SingleRow(table as table) as record

Arguments

Argument	Description
table	The Table to check.

Remarks

- Table.SingleRow is similar to `List.Single` but requires a table as input.

Examples

```
Table.SingleRow(Table.FromRecords({[A=1]})) equals [A=1]
```

11.4.23 Table.Skip

Returns a table that does not contain the first row or rows of the table.

Table.Skip(table as table, optional countOrCondition as any) as table

Arguments

Argument	Description
table	The Table to modify.
optional countOrCondition	The number of rows to skip.

Remarks

- **Table.Skip** is similar to **List.Skip** but requires a table as input.
- If countOrCondition is a number, that many rows (starting at the top) will be skipped.
- If countOrCondition is a condition, the rows that meet the condition will be skipped until a row does not meet the condition.

Examples

```
Table.Skip(Table.FromRecords({[A=1], [A=2], [A=3]})) equals
Table.FromRecords({[A=2], [A=3]})
```

```
Table.Skip(Table.FromRecords({[A=1], [A=2], [A=3), 2) equals
Table.FromRecords({[A=3]})
```

11.4.24 Table.FindText

Returns a table containing only the rows that have the specified text within one of their cells or any part thereof.

Table.FindText (table as table, text as text) as table

Arguments

Argument	Description
table	The Table to modify.
text	The text to compare against.

Examples

```
Table.FindText( Table.FromRecords({[A=1,B=”hello”],[A=2,B=”world”]})
“hello”) equals Table.FromRecords({[A=1,B=”hello”]})
```

11.5 Column operations

11.5.1 Table.Column

Returns the values from a column in a table.

Table.Column(table as table, column as text) as list

Arguments

Argument	Description
table	The Table to check.
column	The column to check.

Remarks

- **Table.Column** is similar to **Record.Field** but requires a table as input.

Examples

```
Table.Column(Table.FromRecords({[A=1,B=2,C=3],[A=4,B=5,C=6]}), “A”)
equals {1,4}
```

11.5.2 Table.ColumnNames

Returns the names of columns from a table.

Table.ColumnNames(table as table) as {Text}

Arguments

Argument	Description
table	The Table to check.

Remarks

- **Table.ColumnNames** is similar to **Record.FieldNames** but requires a table as input.

Examples

```
Table.ColumnNames(Table.FromRecords({[A=1,B=2,C=3]})) equals
{"A","B","C"}
```

11.5.3 Table.HasColumns

Returns true if a table has the specified column or columns.

Table.HasColumns(table as table, columns as any) as logical

Arguments

Argument	Description
table	The Table to check.
columns	The columns to check for as a text value or a list of text values.

Remarks

- **Table.HasColumns** is similar to **`Record.HasFields`** but requires a table as input.

Examples

```
Table.HasColumns(Table.FromRecords({[A=1,B=2,C=3]}),"A") equals true
Table.HasColumns(Table.FromRecords({[A=1,B=2,C=3]}),{"A","B"}) equals
true
Table.HasColumns(Table.FromRecords({[A=1]}),{"A","B"}) equals false
```

11.5.4 Table.PrefixColumns

Returns a table where the columns have all been prefixed with a text value.

Table.PrefixColumns(table as table, prefix as text) as table

Arguments

Argument	Description
table	The Table to modify.

prefix	The prefix to add to every text value.

Examples

```
Table.PrefixColumns(Table.FromRecords({[A=1, B=2]}), "MyTable") equals
Table.FromRecords({[MyTable.A=1, MyTable.B=2]})
```

11.5.5 Table.PromoteHeaders

Promotes the first row of the table into its header or column names.

Table.PromoteHeaders(table as table) as table

Arguments

Argument	Description
table	The Table to modify.

Remarks

Table.PromoteHeaders guarentees uniqueness of column names in the promoted header row by renaming column names that have a naming conflict.

Examples

```
Table.PromoteHeaders(Table.FromRecords({[Column1="A", Column2="B",
Column3=6], [Column1="Hello", Column2="World", Column3=3], [Column1=57,
Column2="Goodbye", Column3=7]})) equals Table.FromRecords({[A="Hello",
B="World", #"6"=3], [A=57, B="Goodbye", #"6"=7]})
```

11.5.6 Table.DemoteHeaders

Demotes the header row down into the first row of a table.

Table.DemoteHeaders(table as table) as table

Arguments

Argument	Description
table	The Table to modify.

Example

```
Table.DemoteHeaders(Table.FromRecords({[A="a", B="b"]})) equals
Table.FromRecords({[Column1 = "A", Column2 = "B"], [Column1 = "a",
Column2 = "b"]
```

11.5.7 Table.RemoveColumns

Returns a table without a specific column or columns.

Table.RemoveColumns(table as table, columns as any, optional missingField as nullable number) as table

Arguments

Argument	Description
table	The Table to modify.
columns	A text value or a list of text values with the names of the columns to remove. missingField is a number value provided to specify handling for missing fields.
optional missingField	The default value of missingField is **MissingField.Error**. For more information, see Parameter Values.

Remarks

`Table.RemoveColumns` is similar to `Record.RemoveFields` applied to every row in a table.

Examples

```
Table.RemoveColumns(Table.FromRecords({[A=1, B=2, C=3]}), "A") equals
Table.FromRecords({[B=2, C=3]})

Table.RemoveColumns(Table.FromRecords({[A=1, B=2, C=3]}), "E") equals
error
```

11.5.8 Table.ReorderColumns

Returns a table with specific columns in an order relative to one another, without changing the order of the columns that aren't specified.

Table.ReorderColumns(table as table, columnOrder as list, optional missingField as nullable number) as table

Arguments

Argument	Description
table	The Table to modify.

columnOrder	The list of columns to arrange in the specific order.
optional missingField	The default value of missingField is **MissingField.Error**. For more information, see Parameter Values.

Remarks

- Table.ReorderColumns is similar to `Record.ReorderFields` applied to every row in a table.
- Columns that are not specified will remain in the same location and the specified columns will be ordered around them.

Examples

```
Table.ReorderColumns(Table.FromRecords({[A=1, B=2]}), {"B","A"}) equals
Table.FromRecords({[B=2, A=1]})

Table.ReorderColumns(Table.FromRecords({[A=1, B=2, C=3, D=4]}),
{"D","B"}) equals Table.FromRecords({[A=1, D=4, C=3, B=2]})
Table.ReorderColumns(Table.FromRecords({[A=1, B=2]}), {"B", "X"}) equals
Expression.Error error

Table.ReorderColumns(Table.FromRecords({[A=1, B=2]}), {"B", "X"},
MissingField.Ignore) equals  Table.FromRecords({[A=1, B=2]})

Table.ReorderColumns(Table.FromRecords({[A=1, B=2]}), {"B", "X"},
MissingField.UseNull) equals Table.FromRecords({[A=1, B=2, X=null]})

Table.ReorderColumns(Table.FromRecords({[A=1, B=2, C=2]}), {"C", "A"})
equals Table.FromRecords({[C=2, B=2, A=1]})
```

11.5.9 Table.RenameColumns

Returns a table with the columns renamed as specified.

Table.RenameColumns(table as table, renames as list, optional missingField as nullable number) as table

Arguments

Argument	Description

table	The Table to modify.
renames	The list of values to rename to.
optional missingField	The default value of missingField is **MissingField.Error**. For more information, see Parameter Values.

Remarks

Table.RenameColumns is similar to `Record.RenameFields` applied to every row in a table.

Examples

```
Table.RenameColumns(Table.FromRecords({[A=1, B=2]}), {"A", "AA"}) equals
Table.FromRecords({[AA=1, B=2]})

Table.RenameColumns(Table.FromRecords({[A=1, B=2]}), {{"A", "B"},{"B",
"A"}}) equals Table.FromRecords({[B=2, A=1]})

Table.RenameColumns(Table.FromRecords({[A=1, B=2]}), {"X" ,"Y"}) equals
Expression.Error

Table.RenameColumns(Table.FromRecords({[A=1, B=2]}), {"X", "Y"},
MissingField.Ignore) equals Table.FromRecords({[A=1, B=2]})

Table.RenameColumns(Table.FromRecords({[A=1, B=2]}), {"X", "Y"},
MissingField.UseNull) equals Table.FromRecords({[A=1, B=2, Y=null]})
```

11.5.10 Table.SelectColumns

Returns a table that contains only specific columns.

Table.SelectColumns(table as table, columns as any, optional missingField as any) as table

Arguments

Argument	Description
table	The Table to modify.
columns	The list of columns to copy.
optional missingField	The default value of missingField is **MissingField.Error**. For more information, see Parameter Values.

Remarks

Table.SelectColumns is similar to `Record.SelectFields` applied to every row in a table.

Examples

```
Table.SelectColumns(Table.FromRecords({[A=1, B=2]}), "B") equals
Table.FromRecords({[B=2]})

Table.SelectColumns(Table.FromRecords({[A=1, B=2, C=3]}), {"C", "B"})
equals Table.FromRecords({[B=2, C=3]})

Table.SelectColumns(Table.FromRecords({[A=1, B=2]}), "X") equals
Expression.Error

Table.SelectColumns(Table.FromRecords({[A=1, B=2]}), "X",
MissingField.Ignore) equals Table.FromRecords({[]})

Table.SelectColumns(Table.FromRecords({[A=1 ,B=2]}), "X",
MissingField.UseNull) equals Table.FromRecords({[X=null]})
```

11.5.11 Table.Pivot

Given the table value, and an attribute column containing the specified pivotValues, creates new columns for each of the pivot values and assigns them values from the valueColumn. An optional aggregationFunction can be provided to handle multiple occurrence of the same key value in the attribute column.

Table.Pivot(table as table, pivotValues as list, attributeColumn as text, valueColumn as text, optional aggregationFunction as nullable function) as table

Arguments

Argument	Description
table	The Table to modify.
pivotValues	The values to transform.
attributeColumn	The column to make the attribute.
valueColumn	The column to make the value.

optional aggregationFunction	Function to aggregate values.

Examples

// Simple input with no key + attribute conflicts. In other words, (key,attribute) is unique.

```
Table.Pivot(
      Table.FromRecords(
        { [ key = “x”, attribute = “a”, value = 1 ],
          [ key = “x”, attribute = “c”, value = 3 ],
          [ key = “y”, attribute = “a”, value = 2 ],
          [ key = “y”, attribute = “b”, value = 4 ] }),
      { “a”, “b”, “c” }, “attribute”, “value”).

// Returns:
// Table.FromRecords({
//    [ key = “x”, a = 1, b = null, c = 3 ],
//    [ key = “y”, a = 2, b = 4, c = null ]}).
```

// Same input as above, but with a conflict introduced for key=”y”, attribute=”a”. Note that this example would throw an exception in SQL Server as well.

```
Table.Pivot(
      Table.FromRecords(
        { [ key = “x”, attribute = “a”, value = 1 ],
          [ key = “x”, attribute = “c”, value = 3 ],
          [ key = “y”, attribute = “a”, value = 2 ],
          [ key = “y”, attribute = “a”, value = 8 ],
          [ key = “y”, attribute = “b”, value = 4 ] }),
      { “a”, “b”, “c” }, “attribute”, “value”).
// Returns:
// Table.FromRecords({
//    [ key = “x”, a = 1, b = null, c = 3 ],
//    [ key = “y”, a = Exception, b = 4, c = null ]}).
```

// Same input as Example 2, but with an additional function specified to resolve the conflict – in this case, to take the minimum value. Note that this resolution method is the same as the PIVOT clause in SQL Server and most other DBMS's.

```
Table.Pivot(
      Table.FromRecords(
        { [ key = "x", attribute = "a", value = 1 ],
          [ key = "x", attribute = "c", value = 3 ],
          [ key = "y", attribute = "a", value = 2 ],
          [ key = "y", attribute = "a", value = 8 ],
          [ key = "y", attribute = "b", value = 4 ] }),
      { "a", "b", "c" }, "attribute", "value", List.Min).
// Returns:
// Table.FromRecords({
//    [ key = "x", a = 1, b = null, c = 3 ],
//   [ key = "y", a = 2, b = 4, c = null ]}).
```

11.5.12 Table.Unpivot

Given a list of table columns, transforms those columns into attribute-value pairs.

Table.Unpivot(table as table, pivotColumns as list, attributeColumn as text, valueColumn as text) as table

Arguments

Argument	Description
table	The Table to modify.
pivotColumns	The columns to transform.
attributeColumn	The column to make the attribute.
valueColumn	The column to make the value.

Remarks

The transformation is patterned after the SQL UNPIVOT operator.

Examples

```
Table.Unpivot(Table.FromRecords({[ key = "x", a = 1, b = null, c = 3
```

```
]}), { "a", "b", "c" }, "attribute", "value") equals
Table.FromRecords({ [ key = "x", attribute = "a", value = 1 ], [ key =
"x", attribute = "c", value = 3 ] })

Table.Unpivot(Table.FromRecords({ [ key = "x", a = 1, b = null, c = 3 ]
}), { "a", "b", "c" }, "key", "value") equals Expression.Error
(duplicate column name)
```

11.5.13 Table.UnpivotOtherColumns

Translates all columns other than a specified set into attribute-value pairs, combined with the rest of the values in each row.

Table.Unpivot(table as table, pivotColumns as list, attributeColumn as text, valueColumn as text) as table

Arguments

Argument	Description
table	The Table to modify.
pivotColumns	The columns to skip transformation.
attributeColumn	The column to make the attribute.
valueColumn	The column to make the value.

Remarks

The transformation is patterned after the SQL UNPIVOT operator.

Examples

```
Table.UnpivotOtherColumns(Table.FromRecords({   [ key = "x", a = 1, b =
2, c = 3 ],   [ key = "y", a = 4, b = 5, c = 6 ]  }), { "key" }, "m",
"n") equals  Table.FromRecords({ [ key = "x", attribute = "a", value = 1
], [ key = "x", attribute = "c", value = 3 ] })  equals

Table.FromRecords({
   [ key = "x", m = "a", n = 1],
   [ key = "x", m = "b", n = 2],
   [ key = "x", m = "c", n = 3],
   [ key = "y", m = "a", n = 4],
   [ key = "y", m = "b", n = 5],
   [ key = "y", m = "c", n = 6]})
```

11.6 Transformation

Parameters for Group options

- `GroupKind.Global = 0;`
- `GroupKind.Local = 1;`

Parameters for Join kinds

- `JoinKind.Inner = 0;`
- `JoinKind.LeftOuter = 1;`
- `JoinKind.RightOuter = 2;`
- `JoinKind.FullOuter = 3;`
- `JoinKind.LeftAnti = 4;`
- `JoinKind.RightAnti = 5`

Join Algorithm

The following JoinAlgorithm values can be specified to Table.Join

- `JoinAlgorithm.Dynamic 0,`
- `JoinAlgorithm.PairwiseHash 1,`
  `JoinAlgorithm.SortMerge 2,`
  `JoinAlgorithm.LeftHash 3,`
  `JoinAlgorithm.RightHash 4,`
  `JoinAlgorithm.LeftIndex 5,`
  `JoinAlgorithm.RightIndex 6,`

11.6.1 Example data

The following tables are used by the examples in this section.

Customers table

```
Customers = Table.FromRecords({
  [CustomerID = 1, Name = "Bob", Phone = "123-4567"],
  [CustomerID = 2, Name = "Jim", Phone = "987-6543"],
  [CustomerID = 3, Name = "Paul", Phone = "543-7890"],
  [CustomerID = 4, Name = "Ringo", Phone = "232-1550"]
}
```

Orders table

```
Orders = Table.FromRecords({
  [OrderID = 1, CustomerID = 1, Item = "Fishing rod", Price = 100.0],
  [OrderID = 2, CustomerID = 1, Item = "1 lb. worms", Price = 5.0],
```

```
  [OrderID = 3, CustomerID = 2, Item = "Fishing net", Price = 25.0],
  [OrderID = 4, CustomerID = 3, Item = "Fish tazer", Price = 200.0],
  [OrderID = 5, CustomerID = 3, Item = "Bandaids", Price = 2.0],
  [OrderID = 6, CustomerID = 1, Item = "Tackle box", Price = 20.0],
  [OrderID = 7, CustomerID = 5, Item = "Bait", Price = 3.25],
  [OrderID = 8, CustomerID = 5, Item = "Fishing Rod", Price = 100.0],
  [OrderID = 9, CustomerID = 6, Item = "Bait", Price = 3.25]
})
```

11.6.2 Table.AddColumn

Adds a column named newColumnName to a table.

Table.AddColumn(table as table, newColumnName as text, columnGenerator as function, optional columnType as nullable type) as table

Arguments

Argument	Description
table	The Table to modify.
newColumnName	The name of the new column to add.
columnGenerator	New column generator function.
optional columnType	Optional column type for new column

Remarks

The values for the column are computed using the specified function from each row.

Examples

```
  Table.AddColumn(Table.FromRecords({[A=1, B=2],[A=3, B=4]}), "C", each
[A] + [B]) equals Table.FromRecords({[A=1, B=2, C=3], [A=3, B=4, C=7]})
```

11.6.3 Table.AddIndexColumn

Returns a table with a new column with a specific name that, for each row, contains an index of the row in the table.

Table.AddIndexColumn(table as table, newColumnName as text, optional initialValue as nullable number, optional increment as nullable number) as table

Arguments

Argument	Description

table	The Table to modify.
newColumnName	The name of the new column.
optional initialValue	The initial column index. The default initial index is 0.
optional increment	The column index increment. The default increment is 1.

Examples

```
Table.AddIndexColumn(Table.FromRecords({[A=45, B=13],[A=33, B=76]}),
"Index") equals Table.FromRecords({[A=45, B=13, Index=0],[A=33, B=76,
Index=1]})
Table.AddIndexColumn(Table.FromRecords({[A=45, B=13],[A=33, B=76]}),
"Index", 1, 2) equals Table.FromRecords({[A=45, B=13,Index=1],[A=33,
B=76, Index=3]})
```

11.6.4 Table.AddJoinColumn

Performs a nested join between `table1` and `table2` from specific columns and produces the join result as a `newColumnName` column for each row of `table1`.

Table.AddJoinColumn(table1 as table, key1 as any, table2 as function, key2 as any, newColumnName as text) as table

Arguments

Argument	Description
table1	The Table to modify.
key1	The table1 column to join.
table2	The Table to check against.
key2	The table2 column to join.
newColumnName	The name of the new column.

Remarks

- Table.AddJoinColumn is similar to Table.Join except that the join results are presented in a nested rather than flattened table.
- Table.AddJoinColumn performs a left outer join by default, other join types are supported in Table.Join or Table.NestedJoin

- The type of the resulting table is computed by taking the type of `table1` and appending a column `newColumnName` with a type that is the type of `table2`.
- For more information about joining tables, see Table.Join.

Example

```
Table.AddJoinColumn(
    Customers, {"CustomerID"},
    Orders, {"CustomerID"},
    "Orders" )  equals
Table.FromRecords(
{[CustomerID = 1, Name = "Bob", Phone = "123-4567",
Orders = Table.FromRecords({
 [OrderID = 1, CustomerID = 1,
Item = "Fishing rod", Price = 100.0],
 [OrderID = 2, CustomerID = 1,
Item = "1 lb. worms", Price = 5.0],
 [OrderID = 6, CustomerID = 1,
Item = "Tackle box", Price = 20.0]})
],
 [CustomerID = 2, Name = "Jim", Phone = "987-6543",
Orders = Table.FromRecords({
 [OrderID = 3, CustomerID = 2,
Item = "Fishing net", Price = 25.0]})
],
 [CustomerID = 3, Name = "Paul", Phone = "543-7890",
Orders = Table.FromRecords({
 [OrderID = 4, CustomerID = 3,
Item = "Fish tazer", Price = 200.0],
 [OrderID = 5, CustomerID = 3,
Item = "Bandaids", Price = 2.0]})
],
 [CustomerID = 4, Name = "Ringo", Phone = "232-1550",
Orders = Table.FromRecords({})
]} as
table [
CustomerID = number,
Name = text,
Phone = text,
Orders = table [
OrderID = number,
CustomerID = number,
Item = text,
Price = number
```

```
]]
```

11.6.5 Table.AddKey

Add a key to `table`.

Table.AddKey(table as table, columns as list, isPrimary as logical) as table

Arguments

Argument	Description
table	The Table to modify.
columns	The subset of column names that defines the key.
isPrimary	Specifies whether the key is primary.

Examples `let`

```
        tableType = type table [Id = number, Name = text],
    table = Table.FromRecords(
              {[Id = 1, Name = "Hello There"], [Id = 2, Name = "Good
           Bye"]}),     resultTable =
    Table.AddKey(table, {"Id"}, true) in
    resultTable
equals
 Table.FromRecords(
       {[Id = 1, Name = "Hello There"],
        [Id = 2, Name = "Good Bye"]},
     {
        "Id",
        "Name"
    })
```

11.6.6 Table.AggregateTableColumn

Aggregates tables nested in a specific column into multiple columns containing aggregate values for those tables.

Table.AggregateTableColumn(table as table, column as text, aggregations as list) as table

Arguments

Argument	Description
table	Table to aggregate.
column	Column containing nexted table to aggragate.
aggregations	Specify the columns containing the tables to aggregate, the aggregation functions to apply to the tables to generate their values, and the names of the aggregate columns to create.

Example

```
Table.AggregateTableColumn(
   Table.FromRecords(
      {[t = Table.FromRecords({[a=1, b=2, c=3], [a=2,b=4,c=6]}), b = 2]},
   type table [t = table [a=number, b=number, c=number], b = number] ),
   , "t",
   {{"a", List.Sum, "sum of t.a"},
    {"b", List.Min, "min of t.b"},
    {"b", List.Max, "max of t.b"},
    {"a", List.Count, "count of t.a"}}) equal
Table.FromRecords(
  {[#"sum of t.a" = 3,
     #"min of t.b" = 2,
     #"max of t.b" = 4,
#"count of t.a" = 2,
b = 2]})
```

11.6.7 Table.ExpandListColumn

Given a column of lists in a table, create a copy of a row for each value in its list.

Table.ExpandListColumn(table as table, column as text) as table

Arguments

Argument	Description
table	The Table to modify.
column	The column to expand.

Examples

`Table.ExpandListColumn(Table.FromRecords({[a = {1, 2, 3}, b = 2]}), “a”)` equals {[a = 1, b = 2], `[a = 2, b = 2], [a = 3, b = 2]`}

11.6.8 Table.ExpandRecordColumn

Expands a column of records into columns with each of the values.

Table.ExpandRecordColumn(table as table, column as text, fieldNames as list, optional newColumnNames as nullable list) as table

Arguments

Argument	Description
table	The Table to modify..
column	The column to expand.
fieldNames	List of field names.
optional newColumnNames	Optional list of new column names.

11.6.9 Table.ExpandTableColumn

Expands a column of records or a column of tables into multiple columns in the containing table.

Table.ExpandTableColumn(table as table, column as text, columnNames as list, optional newColumnNames as nullable list) as table

Arguments

Argument	Description
table	The Table to modify.
column	The column to expand.
columnNames	List of column names.
optional newColumnNames	Optional list of new column names.

11.6.10 Table.FillDown

Replaces `null` values in the specified column or columns of the table with the most recent non-`null` value in the column.

Table.FillDown(table as table, columns as any) as table

Arguments

Argument	Description
table	The Table to modify.
columns	The column or columns to modify.

Examples

```
Table.FillDown(
    Table.FromRecords({[Place=1,    Name="Bob"],
     [Place=null, Name="John"],
     [Place=2,    Name="Brad"],
     [Place=3,    Name="Mark"],
     [Place=null, Name="Tom"],
     [Place=null, Name="Adam"]}),
    "Place")  equals
Table.FromRecords( {[Place=1, Name="Bob"],
 [Place=1, Name="John"],
 [Place=2, Name="Brad"],
 [Place=3, Name="Mark"],
 [Place=3, Name="Tom"],
[Place=3, Name="Adam"]})
```

11.6.1 Table.FillUp

Returns a table from the table specified where the value of the next cell is propagated to the null values cells above in the column specified.

Table.FillUp(table as table, columns as any) as table

Arguments

Argument	Description
table	The Table to modify.
columns	The column or columns to modify.

Examples

```
Table.FillUp(
   Table.FromRecords({
      [a = 1, b = 2],
      [a = 3, b = null],
      [a = 5, b = 3]}
      ),
   {"b"})  equals  Table.FromRecords({[a
= 1, b = 2],
```

```
    [a = 3, b = 3],
    [a = 5, b = 3]}, {
    "a",
    "b"
})
```

11.6.2 Table.Group

Groups table rows by the values of `key` columns for each row.

Table.Group(table as table, key as any, aggregatedColumns as list, optional groupKind as nullable number, optional comparer as nullable function) as table

Arguments

Argument	Description
table	The Table to modify.
key	The key columns.
aggregatedColumns	Specifies the names and function return types.
optional groupKind	The type of group. Possible values are GroupKind.Global (default) and GroupKind.Local.
optional comparer	An optional argument that determines equality between group keys.

Remarks

- The type of the resulting table is computed by preserving the columns that make up the group key, including their types, and appending new columns with names and types according to the names and function return types specified in the `aggregatedColumns` argument.
- For each group, a record is constructed containing the key columns, including their values, along with any aggregated columns from the `aggregatedColumns` argument. A table of these group results is returned.
- A group can be local (GroupKind.Local) or global (GroupKind.Global). A local group is formed from a consecutive sequence of rows from an input table with the same key value. A global group is formed from all rows in an input table with the same key value. Multiple local

groups may be produced with the same key value but only a single global group is produced for a given key value.

- The default groupKind value is GroupKind.Global.
- The Table.Group function may also be used to nest the rows in a group.

Examples

```
Table.Group(Orders, "CustomerID", {"Total", each List.Sum([Price])})
equals    Table.FromRecords(  {[CustomerID  =  1,  Total  =  125.0],
[CustomerID = 2, Total = 25.0],
 [CustomerID = 3, Total = 202.0],
 [CustomerID = 5, Total = 103.25],
 [CustomerID = 6, Total = 3.25]})
Table.Group(
    Orders,
    "CustomerID",
    {"Total", each List.Sum([Price])},
    GroupKind.Local)
Table.FromRecords({[CustomerID = 1, Total = 105.0],
 [CustomerID = 2, Total = 25.0],
 [CustomerID = 3, Total = 202.0],
 [CustomerID = 1, Total = 20.0],
 [CustomerID = 5, Total = 103.25],
 [CustomerID = 6, Total = 3.25]})
Table.Group(
    Orders,
    "CustomerID",
    {{"Total", each List.Sum([Price])},
     {"Orders", (rows) => rows}})
Table.FromRecords(({[CustomerID = 1, Total = 125.0, Orders = {
 [OrderID = 1, CustomerID = 1,
Item = "Fishing rod", Price = 100.0],
 [OrderID = 2, CustomerID = 1,
Item = "1 lb. worms", Price = 5.0],
[OrderID = 6, CustomerID = 1,
Item = "Tackle box", Price = 20.0]}
],
 [CustomerID = 2, Total = 25.0, Orders = {
 [OrderID = 3, CustomerID = 2,
Item = "Fishing net", Price = 25.0]}
],
 [CustomerID = 3, Total = 202.0, Orders = {
 [OrderID = 4, CustomerID = 3,
Item = "Fish tazer", Price = 200.0],
```

```
 [OrderID = 5, CustomerID = 3,
Item = "Bandaids", Price = 2.0]}
],
 [CustomerID = 5, Total = 103.25, Orders = {
 [OrderID = 7, CustomerID = 5, Item
= "Bait", Price = 3.25],
 [OrderID = 8, CustomerID = 5,
Item = "Fishing Rod", Price = 100.0]}
],
 [CustomerID = 6, Total = 3.25, Orders = {
 [OrderID = 9, CustomerID = 6,
Item = "Bait", Price = 3.25]}
]})
```

11.6.3 Table.Join

Joins the rows of `table1` with the rows of `table2` based on the equality of the values of the key columns selected by table1, `key1` and table2, `key2`.

Table.Join(table1 as table, key1 as any, table2 as table, key2 as any, optional joinKind as nullable number, optional joinAlgorithm as nullable number) as table

joinKind Settings

- Inner (default)
- Left Outer
- Right Outer
- Full Outer

Remarks

- The table resulting from an **inner join** contains a row for each pair of rows from `table1` and `table2` that were determined to match based on the specified key columns. The joined row contains the column values from the `table1` row followed by the column values from the `table2` row.
- An Expression.Error is thrown if a column with the same name to appear in both tables of the join unless the column is selected by both `key1` and `key2` and the join is an inner join. In such a case, only the column from `table1` will appear in the join result.
- The `Table.PrefixColumns` function can be used to rename all columns of one, or both, of the tables prior to a join operation to avoid issues arising from duplicate column names between the tables.

- An **outer join** ensures that all rows in an input table appear in the join result. Rows that did not have a match in the other table are joined with a "default" row containing `null` values for all of its columns. A left outer join ensures that all rows of `table1` appear in the result, a right outer join ensures that all rows of `table2` appear in the result, and a full outer join ensures that all rows of both `table1` and `table2` appear in the result.
- The type of the resulting table is computed by appending the columns (and their types) from `table2` to the columns (and types) from `table1`. For **inner joins**, duplicate columns are resolved by selecting the typing of the column from `table1` (as described above). For **outer joins**, the columns of the table that may have "default" rows are made `nullable` in the joined table.

Examples

```
Table.Join(Customers, "CustomerID", Orders, "CustomerID")
equals  Table.FromRecords(
{[CustomerID = 1, Name = "Bob", Phone = "123-4567",
OrderID = 1, Item = "Fishing rod", Price = 100.0],
 [CustomerID = 1, Name = "Bob", Phone = "123-4567", OrderID
= 2, Item = "1 lb. worms", Price = 5.0],
 [CustomerID = 1, Name = "Bob", Phone = "123-4567", OrderID
= 6, Item = "Tackle box", Price = 20.0],
 [CustomerID = 2, Name = "Jim", Phone = "987-6543",
OrderID = 3, Item = "Fishing net", Price = 25.0],
 [CustomerID = 3, Name = "Paul", Phone = "543-7890", OrderID
= 4, Item = "Fish tazer", Price = 200.0],
 [CustomerID = 3, Name = "Paul", Phone = "543-7890",
OrderID = 5, Item = "Bandaids", Price = 2.0]})


Table.Join(
    Customers, "CustomerID",
    Table.PrefixColumns(Orders, "Order"), "Order.CustomerID",
JoinKind.LeftOuter)  equals  Table.FromRecords(
{[CustomerID = 1, Name = "Bob", Phone = "123-4567",
Order.OrderID = 1, Order.CustomerID = 1,
Order.Item = "Fishing rod", Order.Price = 100],
 [CustomerID = 1, Name = "Bob", Phone = "123-4567",
Order.OrderID = 2, Order.CustomerID = 1,
Order.Item = "1 lb. worms", Order.Price = 5],
 [CustomerID = 1, Name = "Bob", Phone = "123-4567",
Order.OrderID = 6, Order.CustomerID = 1,
Order.Item = "Tackle box", Order.Price = 20],
 [CustomerID = 2, Name = "Jim", Phone = "987-6543",
Order.OrderID = 3, Order.CustomerID = 2,
```

```
Order.Item = "Fishing net", Order.Price = 25],
 [CustomerID = 3, Name = "Paul", Phone = "543-7890",
Order.OrderID = 4, Order.CustomerID = 3,
Order.Item = "Fish tazer", Order.Price = 200],
 [CustomerID = 3, Name = "Paul", Phone = "543-7890",
Order.OrderID = 5, Order.CustomerID = 3,
Order.Item = "Bandaids", Order.Price = 2],
 [CustomerID = 4, Name = "Ringo", Phone = "323-1550",
Order.OrderID = null, Order.CustomerID = null, Order.Item
= null, Order.Price = null]})
```

11.6.4 Table.NestedJoin

Joins the rows of the tables based on the equality of the keys. The results are entered into a new column.

Table.NestedJoin(table1 as table, key1 as any, table2 as any, key2 as any, newColumnName as text, optional joinKind as nullable number) as table

Arguments

Argument	Description
table1	The left table
key1	The key column in the left table
table2	The right table
key2	The key column in the right table
newColumnName	The name of the new column the results are entered into
Optional joinKind	The type of join to perform

joinKind Settings

- Left Outer (default)
- Inner
- Right Outer
- Full Outer

11.6.5 Table.Keys

Returns a list of key column names from a table.

Table.Keys(table as table) as list

Arguments

Argument	Description
table	Table to return a list of key column names from.

11.6.6 Table.ReplaceKeys

Returns a new table with new key information set in the **keys** argument.

Table.ReplaceKeys(table as table, keys as list) as table

Arguments

Argument	Description
table	Table to modify.
keys	A list with two fields: Columns and Primary. Columns is a list of columns that are keys. Primary is a primary key.

Example

```
Table.ReplaceKeys(Table.FromRecords({[A={[B=1], [B=2]}, C=1]}), {[Columns =
{"C"}, Primary = true]})
```

11.6.7 Table.CombineColumns

The inverse of Table.SplitColumns, Table.Combine columns merge columns using a combiner function to produce a new column.

Table.CombineColumns(table as table, sourceColumns as list, combiner as function, column as text) as table

Arguments

Argument	Description
table	The Table to modify.
sourceColumns	The list of columns to combine.
combiner	The table combiner function.
column	The column to modify.

Example

```
Table.CombineColumns(Table.FromRecords({ [A.1 = "a", A.2 = "b", B = "c" ],
[A.1 = "b", A.2 = "c", B = "d"]}, {"A.1","A.2","B"}),{"A.1",
```

```
"A.2"},Combiner.CombineTextByDelimiter(","),"Merged")
// Table.FromRecords({ [
//        Merged = "a,b",
//        B = "c"
//    ], [
//        Merged = "b,c",
//        B = "d"
//    ]
//}, {
//    "Merged",
//    "B"
//})
```

11.6.8 Table.SplitColumn

Returns a new set of columns from a single column applying a splitter function to each value.

Table.SplitColumn(table as table, sourceColumn as text, splitter as function, optional columnNamesOrNumber as any, optional default as any, optional extraValues as any) as record

Arguments

Argument	Description
table	The Table to modify.
sourceColumn	The column to modify.
splitter	
columnNamesOrNumber	List of column names that do not conflict with columns from the target table.
optional default	Default value.
extraValues	Handles of extra values or overflow values.

Example

```
Table.SplitColumn(Table.FromRecords({[A="a,b", B="c"],[A="b,c",
B="d"]}),"A",Splitter.SplitTextByDelimiter(","),2)
// Table.FromRecords({ [
```

```
//        A.1 = "a",
//        A.2 = "b",
//        B = "c"
//    ], [
//        A.1 = "b",
//        A.2 = "c",
//        B = "d"
//    ]
//}, {
//    "A.1",
//    "A.2",
//    "B"
//})
```

11.6.9 Table.TransformColumns

Transforms columns from a table using a function.

Table.TransformColumns(table as table, transformOperations as list, optional defaultTransformation as nullable function, optional missingField as nullable number) as table

Arguments

Argument	Description
table	The Table to modify.
transformOperations	The list of transformOperations to run.
optional defaultTransformation	The default table transformation.
optional missingField	Missing field value.

Remarks

Table.TransformColumns is similar to `Record.TransformFields` applied to every row in a table.

Examples

```
Table.TransformColumns(
    Table.FromRecords({[A="1", B=2], [A="5", B=10]}),
    {"A", Number.FromText})  equals
Table.FromRecords({[A=1,B=2], [A=5,B=10]})
```

```
Table.TransformColumns(
    Table.FromRecords({[A="1",B=2], [A="5", B=10]}),
    {{"A", Number.FromText},     {"B", each _ + 1}})
equals  Table.FromRecords({[A=1,B=3], [A=5,B=11]})

Table.TransformColumns(
    Table.FromRecords({[A="1",B=2], [A="5", B=10]}),
    {"X", Number.FromText})
equals  Expression.Error

Table.TransformColumns(
    Table.FromRecords({[A="1",B=2], [A="5", B=10]}),
    {"X", Number.FromText},
MissingField.Ignore)
 equals  Table.FromRecords({[A="1",B=2], [A="5",B=10]})

Table.TransformColumns(
    Table.FromRecords({[A="1",B=2], [A="5", B=10]}),
    {"X", Number.FromText},
MissingField.UseNull)
 equals  Table.FromRecords({[A="1",B=2,X=/* Expression.Error*/],
 [A="5",B=10,X=/* Expression.Error error*/]})
```

11.6.10 Table.TransformColumnTypes

Transforms the column types from a table using the specified type.

Table.TransformColumnTypes(table as table, typeTransformations as list, optional culture as nullable text) as table

Arguments

Argument	Description
table	The Table to modify.
typeTransformations	The List of typeTransofrmations to make.

optional culture	A text value corresponding to the culture values supported on your version of Windows, such as "en-US". If the culture is not specified, the current user culture is used. For a list of culture names, see National Language Support (NLS) API Reference.

Examples

```
Table.TransformColumnTypes(
    Table.FromRecords({[A="1",B=2], [A="5", B=10]}),
    {"A", type number})  equals
Table.FromRecords({[A=1,B=2], [A=5,B=10]})

Table.TransformColumnTypes(
    Table.FromRecords({[A="1",B=2], [A="5", B=10]}),
    {"X", type number})
equals  Expression.Error

Table.TransformColumnTypes(
    Table.FromRecords({[A="1/10/1990",B="29,000"], [A="2/10/1990",
B="29.000"]}),
    {{"A", type date}, {"B", type number}}, "en-US")
equals  Table.FromRecords({[A=#date(1990, 10, 1),B=29000],
[A=#date(1990, 10, 2),B=29]})
```

11.6.11 Table.TransformRows

Transforms the rows from a table using a `transform` function.

Table.TransformRows(table as table, transform as function) as list

Arguments

Argument	Description
table	The Table to modify.
transform	The transform function to use.

Remarks

Table.TransformRows is similar to `List.Transform` but requires a table as input.

Examples

```
Table.TransformRows(
    Table.FromRecords({[A=1], [A=2], [A=3], [A=4], [A=5]}),
each [A])  equals {1, 2, 3, 4, 5}
```

11.6.12 Table.Transpose

Returns a table with columns converted to rows and rows converted to columns from the input table.

Table.Transpose(table as table, optional columns as any) as table

Arguments

Argument	Description
table	The Table to modify.
optional columns	Columns to transform.

Examples

```
Table.PromoteHeaders(
    Table.Transpose(
        Table.DemoteHeaders(
            Table.FromRecords({
                [Name = "Full Name", Value = "Fred"],
                [Name = "Age", Value = 42],
                [Name = "Country", Value = "UK"]
            })
        )
    ) )
equals
Table.
FromRe
cords(
{ [
        Name = "Value",
        #"Full Name" = "Fred",
        Age = 42,
        Country = "UK"
    ]
})
```

11.6.13 Table.ReplaceValue

Replaces `oldValue` with `newValue` in specific columns of a table, using the provided replacer function, such as text.Replace or Value.Replace.

Table.ReplaceValue(table as table, oldValue as any, newValue as any,replacer as function, columnsToSearch as {Text}) as table

Arguments

Argument	Description
table	The Table to modify.
oldValue	The old value to modify.
newValue	The new value to replace with.
replacer	The replacer function to use, such as Text.Replace or Value.Replace.
columnsToSearch	The list of columns to search through.

Examples

```
Table.ReplaceValue(
    Table.FromRecords({[A=1, B="hello"],[A=2, B="world"]}),
    "hello",
    "goodbye" Text.Replace,
      {"B"})  equals Table.FromRecords({[A=1,
B="goodbye"],[A=2, B="world"})

Table.ReplaceValue(
    Table.FromRecords({[A=1, B="hello"],[A=2, B="world"]}),
    "or",
    "ur", Text.Replace,
      {"B"})  equals Table.FromRecords({[A=1,
B="hello"],[A=2, B="wurld"})
```

11.6.1 Table.ReplaceErrorValues

Replaces the error values in the specified columns with the corresponding specified value.

Table.ReplaceErrorValues(table as table, errorReplacement as list) as table

Arguments

Argument	Description
table	The Table to modify.
errorReplacement	The list of columns and the value to replace the errors with. The form of the list is {{column1, value1},…}

Remarks

There may be only one replacement value per column, specifying the column more than one will result in an error

Examples

```
Table.ReplaceErrorValues(
   Table.FromRows({{1,"hello"},{3,...}}, {"A","B"}),
   {"B", "world"})  equals Table.FromRecords({[A = 1, B = "hello"], A
= 3, B = "world"]},
{"A", "B"})
```

11.7 Membership

11.7.1 Parameters for membership checks

Occurrence specification

- `Occurrence.First  = 0`
- `Occurrence.Last   = 1`
- `Occurrence.All    = 2`

11.7.2 Table.Contains

Determines whether the a record appears as a row in the table.

Table.Contains(table as table, row as record, optional equationCriteria as any) as logical

Arguments

Argument	Description
table	The Table to check.
row	The row to check for.
optional equationCriteria	An optional value that specifies how to control comparison between the rows of the table.

Remarks

- Table.Contains is similar to `List.Contains` but requires a table as input.

Examples

```
Table.Contains(
    Table.FromRecords( {[A=1, B=2]}),
    [A=1, B=3]) equals false

Table.Contains(
    Table.FromRecords({[A=1, B=2]}),
    [A=1, B=3],
"A") equals true
```

11.7.3 Table.ContainsAll

Determines whether all of the specified records appear as rows in the table.

Table.ContainsAll(table as table, rows as list, optional equationCriteria as any) as logical

Arguments

Argument	Description
table	The Table to check.
rows	The List of rows to check for.
optional equationCriteria	An optional value that specifies how to control comparison between the rows of the table.

Remarks

- `Table.ContainsAll is similar to List.ContainsAll but requires a table as input.`

Examples

```
Table.ContainsAll(
    Table.FromRecords( {[A=1, B=2],[A=2, B=3],[A=3, B=4]}),
    {[A=1, B=3],[A=2, B=4]},
    "A") equals true
```

11.7.4 Table.ContainsAny

Determines whether any of the specified records appear as rows in the table.

Table.ContainsAny(table as table, rows as list, optional equationCriteria as any) as logical

Arguments

Argument	Description
table	The Table to check.
rows	The List of rows to check for.
optional equationCriteria	An optional value that specifies how to control comparison between the rows of the table.

Remarks

- Table.ContainsAny is similar to `List.ContainsAny` but requires a table as input.

Examples

```
Table.ContainsAny(
    Table.FromRecords( {[A=1, B=2],[A=2, B=3],[A=3, B=4]}),
    {[A=1, B=2],[A=2, B=4]},
    {"A", "B"}) equals true
```

11.7.5 Table.Distinct

Removes duplicate rows from a table, ensuring that all remaining rows are distinct.

Table.Distinct(table as table, optional equationCriteria as any) as table

Arguments

Argument	Description
table	The Table to modify.
optional equationCriteria	An optional value that specifies how to control comparison between the rows of the table

Examples

```
Table.Distinct(
    Table.FromRecords( {[A=1, B=2],[A=3, B=4],[A=1, B=6]}),
    "A") equals Table.FromRecords( {[A=1,
B=2],[A=3, B=4]})

Table.Distinct(
    Table.FromRecords( {[A=1,B=2],[A=3,B=4],[A=1,B=6]}),
    {"A",                   "B"})                   equals
Table.FromRecords( {[A=1,B=2],[A=3,B=4],[A=1,B=6]})
```

11.7.6 Table.IsDistinct

Determines whether a table contains only distinct rows.

Table.IsDistinct(table as table, optional equationCriteria as any) as logical

Arguments

Argument	Description
table	The Table to check.
optional equationCriteria	An optional value that specifies how to control comparison between the rows of the table.

11.7.7 Table.PositionOf

Determines the position or positions of a row within a table.

Table.PositionOf(table as table, row as record, optional occurrence as nullable number, optional equationCriteria as any) as any

Arguments

Argument	Description
table	The Table to check.
row	The row to check for.
optional occurrence	The number for the appropriate occurrence specification.
optional equationCriteria	An optional value that specifies how to control comparison between the rows of the table.

Occurrence specification

- `Occurrence.First  = 0`
- `Occurrence.Last   = 1`
- `Occurrence.All    = 2`

Remarks

- Table.PositionOf is similar to `List.PositionOf` but requires a table as input.

Examples

```
Table.PositionOf(
    Table.FromRecords({[A=1, B=2],[A=3, B=4],[A=1 B=6]}),
    [A=3,B=4]) equals 1

Table.PositionOf(
    Table.FromRecords({[A=1, B=2],[A=3, B=4],[A=1, B=6]}),
    [A=1],
    Occurrence.All,
"A") equals {0, 2}
```

11.7.8 Table.PositionOfAny

Determines the position or positions of any of the specified rows within the table.

Table.PositionOfAny(table as table, rows as list, optional occurrence as nullable number, optional equationCriteria as any) as any

Arguments

Argument	Description
table	The Table to check.
rows	The List of rows to check for.
optional occurrence	The number for the appropriate occurrence specification.
optional equationCriteria	An optional value that specifies how to control comparison between the rows of the table.

Occurrence specification

- `Occurrence.First  = 0`
- `Occurrence.Last   = 1`
- `Occurrence.All    = 2`

Remarks

- Table.PositionOfAny is similar to `List.PositionOfAny` but requires a table as input.

Examples

```
Table.PositionOfAny(
    Table.FromRecords({[A=1, B=2],[A=3, B=4],[A=1, B=6]}),
```

```
    {[A=2, B=6],[A=3, B=4]})
equals 1

Table.PositionOfAny(
    Table.FromRecords({[A=1, B=2],[A=3, B=4],[A=1, B=6]}),
    {[A=3, B=7],[A=1, B=6]},

Occurrence.All,
"A")  equals {0,
1, 2}
```

11.7.9 Table.RemoveMatchingRows

Removes all occurrences of rows from a table.

Table.RemoveMatchingRows(table as table, rows as list, optional equationCriteria as any) as table

Arguments

Argument	Description
table	The Table to modify.
rows	The List of rows to remove.
optional equationCriteria	An optional value that specifies how to control comparison between the rows of the table.

Remarks

Table.RemoveMatchingRows is similar to `List.RemoveMatchingItems` but requires a table as input.

Example

```
Table.RemoveMatchingRows(Table.FromRecords({[A=1, B=2],[A=3, B=4],[A=1,
B=6]}), {[A=1]}, "A")
 equals Table.FromRecords({[A=3,B=4]})
```

11.7.10 Table.ReplaceMatchingRows

Replaces specific rows from a table with the new rows.

Table.ReplaceMatchingRows(table as table, replacements as list, optional equationCriteria as any) as table

Arguments

Argument	Description
table	The Table to modify.
replacements	The List of replacement rows.
optional equationCriteria	An optional value that specifies how to control comparison between the rows of the table.

Remarks

- `Table.ReplaceMatchingRows is similar to List.ReplaceMatchingRows but requires a table as input.`
- The new rows must be compatible with the type of the table .

Example

```
Table.ReplaceMatchingRows (
    Table.FromRecords({[A=1, B=2],[A=3, B=4],[A=1, B=6]}),
    {[A=1],[A=90, B=100]},
    "A")
 equals Table.FromRecords({[A=90, B=100],[A=3, B=4],[A=90, B=100]})
```

11.8 Ordering

11.8.1 Example data

The following tables are used by the examples in this section.

Employees table

```
Employees = Table.FromRecords(
    {[Name="Bill",   Level=7,  Salary=100000],
     [Name="Barb",   Level=8,  Salary=150000],
     [Name="Andrew", Level=6,  Salary=85000],
     [Name="Nikki",  Level=5,  Salary=75000],
     [Name="Margo",  Level=3,  Salary=45000],
[Name="Jeff",   Level=10, Salary=200000]},
type table [     Name = text,
    Level = number,
    Salary = number
])
```

11.8.2 Table.MaxN

Returns the largest N rows from a table. After the rows are sorted, the countOrCondition parameter must be specified to further filter the result.

Table.MaxN(table as table, comparisonCriteria as any, countOrCondition as any) as table

Arguments

Argument	Description
table	The Table to check.
comparisonCriteria	Largest N rows comparison criteria.
countOrCondition	After the rows are sorted, **countOrCondition** further filters the result.

The **`countOrCondition`** argument has two possible settings:

Argument	Description
as a number	A list of items up to `countOrCondition` items in ascending order is returned.
as a condition	A list of items that initially meet the condition is returned. Once an item fails the condition, no further items are considered.

Examples

```
Table.MaxN(Employees, "Salary", 3)  equals
Table.FromRecords({[Name="Jeff", Level=10, Salary=200000]
 [Name="Barb", Level=8,  Salary=150000]
 [Name="Bill", Level=7,  Salary=100000]})

Table.MaxN(Employees, "Salary", each [Level] > 7)
  equals  Table.FromRecords( {[Name="Jeff", Level=10, Salary=200000]
[Name="Barb", Level=8,  Salary=150000]})
```

11.8.3 Table.Max

Returns the largest row or rows from a table using a `comparisonCriteria`.

Table.Max(table as table, comparisonCriteria as any, optional default as any) as any

Arguments

Argument	Description
table	The Table to check.
comparisonCriteria	largest row or rows comparison criteria.
optional default	Default value.

Remarks

Table.Max is similar to `List.Max` but requires a table as input.

Example

```
Table.Max(Employees, "Salary") equals  [Name="Jeff", Level=10,
Salary=200000]
```

11.8.4 Table.MinN

Returns the smallest N rows in the given table. After the rows are sorted, the countOrCondition parameter must be specified to further filter the result.

Table.MinN(table as table, comparisonCriteria as any, countOrCondition as any) as table

Arguments

Argument	Description
table	The Table to check.
comparisonCriteria	Smallest N rows comparison criteria.
countOrCondition	After the rows are sorted, countOrCondition further filters the result.

The **`countOrCondition`** argument has two possible settings:

Setting	Description
as a number	A list of items up to `countOrCondition` items in ascending order is returned.
as a condition	A list of items that initially meet the condition is returned. Once an item fails the condition, no further items are considered.

Examples

```
Table.MinN(Employees, "Salary", 3)  equals
Table.FromRecords({[Name="Margo",  Level=3, Salary=45000]
 [Name="Nikki",  Level=5, Salary=75000]
 [Name="Andrew", Level=6, Salary=85000]})

Table.MinN(Employees, "Salary", each [Level] < 6)  equals
Table.FromRecords({[Name="Margo", Level=3, Salary=45000]
 [Name="Nikki", Level=5, Salary=75000]})
```

11.8.5 Table.Min

Returns the smallest row or rows from a table using a `comparisonCriteria`.

Table.Min(table as table, comparisonCriteria as any, optional default as any) as table

Arguments

Argument	Description
table	The Table to check.
comparisonCriteria	Smallest row or rows comparison criteria.
optional default	Default value.

Remarks

- `Table.Min is similar to List.Min but requires a table as input.`

Examples

```
Table.Min(Employees, "Salary") equals [Name="Margo", Level=3,
Salary=45000]
```

11.8.6 Table.Sort

Sorts the rows in a table using a `comparisonCriteria` or a default ordering if one is not specified.

Table.Sort(table as table, optional comparisonCriteria as any) as table

Arguments

Argument	Description
table	The Table to modify.
comparisonCriteria	Sort comparison criteria.

Remarks

- Table.Sort is similar to `List.Sort` but requires a table as input.

Examples

```
Table.Sort(
    Table.FromRecords({[A=1 ,B=2], [A=2, B=3], [A=1, B=1]}),
    "B")
 equals Table.FromRecords({[A=1, B=1], [A=1, B=2], [A=2, B=3]})

Table.Sort(
   Table.FromRecords( {[A=1, B=2], [A=2, B=3], [A=1, B=1]}),
    {{"B", Order.Descending}, "A"})
 equals Table.FromRecords({[A=3, B=2], [A=1, B=2], [A=1, B=1]})
```

11.9 Other

11.9.1 Table.Buffer

Buffers a table into memory, isolating it from external changes during evaluation.

Table.Buffer(table as table) as table

Arguments

Argument	Description
table	The Table to buffer.

Remarks

Table.Buffer is similar to `List.Buffer` but requires a table as input.

Example

```
Table.Buffer(Sql.Database("localhost", "Northwind")[Customers]) equals
Buffered copy of the Customers table
```

11.10 Parameter values

11.10.1 Naming output columns

This parameter is a list of text values specifying the column names of the resulting table. This parameter is generally used in the Table construction functions, such as Table.FromRows and Table.FromList.

11.10.2 Comparison criteria

Comparison criterion can be provided as either of the following values:

- A number value to specify a sort order. See sort order in the parameter values section above.
- To compute a key to be used for sorting, a function of 1 argument can be used.
- To both select a key and control order, comparison criterion can be a list containing the key and order.

- To completely control the comparison, a function of 2 arguments can be used that returns 1, 0, or 1 given the relationship between the left and right inputs. `Value.Compare` is a method that can be used to delegate this logic.

For examples, see description of Table.Sort.

11.10.3 Count or Condition critieria

This criteria is generally used in ordering or row operations. It determines the number of rows returned in the table and can take two forms, a number or a condition:

- A number indicates how many values to return inline with the appropriate function
- If a condition is specified, the rows containing values that initially meet the condition is returned. Once a value fails the condition, no further values are considered.

See Table.FirstN or Table.MaxN.

11.10.4 Handling of extra values

This is used to indicate how the function should handle extra values in a row. This parameter is specified as a number, which maps to the options below.

```
ExtraValues.List = 0
ExtraValues.Error = 1
ExtraValues.Ignore = 2
```

For more information, see **Table.FromList**.

11.10.5 Missing column handling

This is used to indicate how the function should handle missing columns. This parameter is specified as a number, which maps to the options below.

```
MissingField.Error = 0;
MissingField.Ignore = 1;
MissingField.UseNull = 2;
```

This is used in column or transformation operations. For Examples, see Trable.TransformColumns.

11.10.6 Sort Order

This is used to indicate how the results should be sorted. This parameter is specified as a number, which maps to the options below.

```
    Order.Ascending = 0
Order.Descending = 1
```

11.10.7 Equation criteria

Equation criteria for tables can be specified as either a

- A function value that is either ○ A key selector that determines the column in the table to apply the equality criteria, or
    - A comparer function that is used to specify the kind of comparison to apply. Built in comparer functions can be specified, see section for Comparer functions.
- A list of the columns in the table to apply the equality criteria For examples, look at description for Table.Distinct.

12. Cube

12.1 Cube.Dimensions

Returns a table containing the set of available dimensions within the cube.

```
Cube.Dimensions(cube as table) as table
```

Arguments

Argument	Description
cube	The cube from which to obtain the table of dimensions.

Remarks

Each dimension is a table containing a set of dimension attributes. Each dimension attribute is represented as a column in the dimension table. Dimensions can be expanded in the cube using Cube.AddAndExpandDimensionColumn.

12.2 Cube.Measures

Returns a table containing the set of available measures within the cube.

```
Cube.Measures(cube as table) as table
```

Arguments

Argument	Description
cube	The cube from which to obtain the table of measures.

Remarks

Each measure is represented as a function. Measures can be applied to the cube using Cube.AddMeasureColumn.

12.3 Cube.AddAndExpandDimensionColumn

Merges the specified dimension table into the cube's filter context and changes the dimensional granularity of the filter context by expanding the specified set of dimension attributes. Adds the

specified set of dimension attributes to the tabular view of the cube with columns of **newColumnNames** (or **attributeNames** if not specified).

```
Cube.AddAndExpandDimensionColumn(cube as table, dimensionSelector as any,
attributeNames as {text}, optional newColumnNames as nullable {text}) as
table
```

Arguments

Argument	Description
cube	The cube with which to merge the dimension table.
dimensionSelector	Either the ID (text) of the dimension table to apply or a dimension table.
attributeNames	The dimension attributes (columns of the dimension table) to expand in the cube after applying the dimension table.
newColumnNames	The (optional) column names for the expanded attributes of the dimension table in the tabular view over the cube.

Remarks

The dimension table must be a tabular view of a dimension from the same cube as **cube**.

The dimension table may contain its own filter context, adjusted by means of standard cube and table functions.

The resultant filter context is the intersection of the filter context of **cube** and the filter context of the dimension table.

Slicing of a cube may be performed by adding a filtered dimension table without expanding any of its attributes.

Measure columns are recalculated (i.e. “floated”) after this operation is applied.

The same dimension attribute may be expanded multiple times. Subsequent expansions of the dimension attribute have no effect on the dimensional granularity of the cube’s filter context.

12.4 Cube.CollapseAndRemoveColumns

Changes the dimensional granularity of the filter context for the cube by collapsing the attributes mapped to the specified columns. The columns are also removed from the tabular view of the cube.

```
Cube.CollapseAndRemoveColumns(cube as table, columnNames as {text}) as table
```

Arguments

Argument	Description
cube	The cube from which columns should be collapsed and removed.
columnNames	The names of the columns (representing dimension attributes) to collapse and remove.

Remarks

The columns being removed must represent dimension attributes.

Slicing of a cube may be performed by filtering and then collapsing and removing a set of columns.

Measure columns are recalculated (i.e. “floated”) after this operation is applied.

Columns representing the same dimension attribute may be collapsed and removed multiple times. The final collapse and remove of a column representing an attribute will have an effect on the dimensional granularity of the cube’s filter context; all earlier collapse and remove operations will have no effect.

12.5 Cube.Transform

Applies the list of transform functions on the cube specified.

```
Cube.Transform(cube as table, transforms as list) as table
```

Arguments

Argument	Description
cube	The cube to apply the transforms to
transforms	A list of transformation functions

Remarks

12.6 Cube.DisplayFolders

Returns a nested tree of tables representing the display folder hierarchy of the objects (e.g. dimensions and measures) available for use in the cube. Note this should not be used to navigate objects within the cube, see Remarks for more details.

Cube.DisplayFolders(cube as table) as table

Arguments

Argument	Description
cube	The cube from which to obtain the tree of display folders.

Remarks

The nested tree of display folder tables provides a user-friend way to organize and navigate to objects within the cube. It is not guaranteed to remain stable. It is recommended to use stable means of navigating to objects within the cube (e.g. Cube.Dimensions or Cube.Measures) if the target objects are to be used from within a query.

12.7 Cube.AttributeMemberId

Returns the unique member identifier from a member property value.

Cube.AttributeMemberId(value as any) as any

Arguments

Argument	Description
value	The member property value from which to return the unique member ID.

Remarks

The nested tree of display folder tables provides a user-friend way to organize and navigate to objects within the cube. It is not guaranteed to remain stable. It is recommended to use stable means of navigating to objects within the cube (e.g. Cube.Dimensions or Cube.Measures) if the target objects are to be used from within a query.

13. Values

The following functions provide information about values.

13.1 Value.Equals

Returns whether two values are equal.

Value.Equals(value1 as any, value2 as any, optional precision as nullable number) as logical

Arguments

Argument	Description
value1	The left value to compare.
value2	The right value to compare.
optional precision	Precision of comparison. The default value is Precision.Double.

Examples

```
Value.Equals(2,4) equals false
Value.Equals(2,2) equals true
```

13.2 Value.NullableEquals

Returns a logical value or null based on two values .

Value.NullableEquals(value1 as any, value2 as any, optional precision as nullable number) as nullable logical

Arguments

Argument	Description
value1	The right value to compare.
value2	The left value to compare.
optional precision	Precision of comparison. The default value is Precision.Double.

Remarks

If either of the argument is null, it applies a nullable equality rules; otherwise, the same result as Value.Equals.

Example

```
Value.NullableEquals(1, null) equals null
```

13.3 Value.Compare

Returns 1, 0, or -1 based on **value1** being greater than, equal to, or less than the **value2**. An optional comparer function can be provided.

Value.Compare(value1 as any, value2 as any, optional precision as nullable number) as

Arguments

Argument	Description
value1	The left value to compare.
value2	The right value to compare.
optional precision	Precision of comparison. The default value is Precision.Double.

13.4 Arithmetic operations

The built-in arithmetic operators (+, -, *, /) use Double Precision. The following library functions can be used to request these operations using a specific precision model.

13.4.1 Value.Add

Value.Add(value1 as any, value2 as any, optional precision as nullable number) as any

Arguments

Argument	Description
value1	The left value to add.
value2	The right value to add.
optional precision	Precision of comparison. The default value is Precision.Double.

13.4.2 Value.Subtract

Value.Subtract(value1 as any, value2 as any, optional precision as nullable number) as any

Arguments

Argument	Description

value1	The left value to subtract.
value2	The right value to subtract.
optional precision	Precision of comparison. The default value is Precision.Double.

13.4.3 Value.Divide

Value.Divide(value1 as any, value2 as any, optional precision as nullable number) as any

Arguments

Argument	Description
value1	Dividend, or numerator.
value2	Divisor, or denominator.
optional precision	Precision of comparison. The default value is Precision.Double.

13.4.4 Value.Multiply

Value.Multiply(value1 as any, value2 as any, optional precision as nullable number) as any

Arguments

Argument	Description
value1	The left number to multiply.
value2	The right number to multiply.
optional precision	Precision of comparison. The default value is Precision.Double.

13.4.5 Paramter types

Precision specification

```
Precision.Double   = 0,
Precision.Decimal  = 1,
```

13.5 Value.As

Value.As is the function corresponding to the as operator in the formula language. The expression value as type asserts that the value of a value argument is compatible with type as per the is operator. If it is not compatible, an error is raised.

Value.As(value as any, type as type) as any

Arguments

Argument	Description
value	The value as.
type	Asserts that the value of a value argument is compatible with type.

13.6 Value.Is

Value.Is is the function corresponding to the is operator in the formula language. The expression value is type returns true if the ascribed type of vlaue is compatible with type, and returns false if the ascribed type of value is incompatible with type.

Value.Is(value as any, type as type) as logical

Arguments

Argument	Description
value	The value is.
type	Type of vlaue is compatible with type

13.7 Value.ReplaceType

A value may be ascribed a type using Value.ReplaceType. Value.ReplaceType either returns a new value with the type ascribed or raises an error if the new type is incompatible with the value's native primitive type. In particular, the function raises an error when an attempt is made to ascribe an abstract type, such as any. When replacing a the type of a record, the new type must have the same number of fields, and the new fields replace the old fields by ordinal position, not by name. Similarly, when replacing the type of a table, the new type must have the same number of columns, and the new columns replace the old columns by ordinal position.

Value.Replacetype(value as any, replacedType as type) as any

Arguments

Argument	Description
value	Value to replace.
replacedType	As type.

13.8 Value.Type

The ascribed type of a value may be obtained using Value.Type.

Example

```
Value.Type(Value.ReplaceType( {1}, type {number}) equals type {number}
```

14. Metadata

The following function relate to metadata information on values.

14.1 Value.Metadata

Returns a record containing the input's metadata.

Value.Metadata(value as any) as record

Arguments

Argument	Description
value	The value to retrieve metadata for.

Example

```
Value.Metadata(1 meta [meta = 1]) equals [ meta = 1]
```

14.2 Value.RemoveMetadata

Removes the metadata on the value and returns the original value.

Value.RemoveMetadata(value as any) as any

Arguments

Argument	Description
value	The value to remove metadata from.

Example

```
Value.RemoveMetadata(1 meta [meta = 1])  equals 1
```

14.3 Value.ReplaceMetadata

Replaces the metadata on a value with the new metadata record provided and returns the original value with the new metadata attached.

Value.ReplaceMetadata(value as any, newMeta as record) as any

Arguments

Argument	Description

value	The value to modify.
newMeta	The new metadata to replace the old metadata with..

Example

```
Value.ReplaceMetadata(1 meta [meta = 1], [meta=2]) equals  1 meta [meta
= 2]
```

15. Accessing data

The following deals with external data.

15.1 Return values

Functions in this section access data and return table values. Most of these functions return a table value that is called a **navigation table**. A **navigation table** is a two column table. The first column contains the name of an item and the corresponding second column contains the value of that item. This shape is primarily used by the Power Query user interface to provide navigation experience over the potentially large hierarchical data returned.

15.2 Access.Database

Returns a structural representation of an Microsoft Access database. The database argument is The return value is a record, where each field represents a table in the Access database.

Access.Database(database as binary) as table

Arguments

Argument	Description
database	A binary value that is the return value functions including `File.Contents` and `Web.Contents`.

Example

```
Access.Database(File.Contents("c:\users\myuser\Desktop\mydb.accdb"))
```

15.3 ActiveDirectory.Domains

Returns a table with Domain information available in the current domain or optional Active Directory forest.

ActiveDirectory.Domains(optional forestRootDomainName as nullable text) as table

Arguments

Argument	Description
optional	`Optional` Active Directory forest.
forestRootDomainName	

15.4 Csv.Document

Returns the contents of a CSV document as a table using the specified encoding.

Csv.Document(source as any, optional columns as any, optional delimiter as nullable text, optional extraValues as nullable number, optional encoding as nullable number) as table

Arguments

Argument	Description
source	The CSV file to parse.
optional columns	Optional column names.
optional delimiter	Delimiters between values.
optional extraValues	Specification for extra value handling.
optional encoding	Encoding value.

Remarks

QuoteStyle.Csv is used during the parsing. With this QuoteStyle, a double quote character is used to demarcate such regions, and a pair of double quote characters is used to indicate a single double quote character within such a region.

15.5 Marketplace.Subscriptions

Returns feeds offered by the Microsoft Azure Marketplace to which the current user has subscribed.

Marketplace.Subscriptions() as table

15.6 Excel.Workbook

Returns a table representing sheets in the given excel workbook.

Excel.Workbook(workbook as binary, optional useHeaders as nullable logical) as table

Arguments

Argument	Description
workbook	The workbook to retrieve the sheets for.
optional useHeaders	Use the first row of the excel sheets as table headers.

Examples

```
Excel.Workbook(File.Contents("localExcelFile.xlsx"))
```

15.7 Excel.CurrentWorkbook

Returns the tables in the current Excel workbook

Excel.CurrentWorkbook() as table

15.8 Exchange.Contents

Returns a table of contents from a Microsoft Exchange Account.

Exchange.Contents() as table

15.9 Facebook.Graph

Returns a table containing content from the Facebook graph .

Facebook.Graph(url as text) as any

Arguments

Argument	Description
url	The text that represents URL.

15.10 File.Contents

Returns the binary contents of the file located at a path.

File.Contents(path as text) as binary

Arguments

Argument	Description
path	The path to the file to retrieve contents for.

Example

```
File.Contents("c:\users\myuser\Desktop\file.txt")
```

15.11 Folder.Contents

Returns a table containing the properties and contents of the files and folders found at **path**.

Folder.Contents(path as text) as table

Arguments

Argument	Description
path	The path to the folder to retrieve contents for.

15.12 Folder.Files

Returns a table containing a row for each file found at a folder path, and subfolders. Each row contains properties of the folder or file and a link to its content.

Folder.Files(path as text) as table

Arguments

Argument	Description
path	The path to the folder to retrieve files for.

15.13 Hdfs.Contents

Returns a table containing a row for each folder and file found at the folder **url**, {0}, from a Hadoop file system. Each row contains properties of the folder or file and a link to its content.

Hdfs.Contents(url as text) as table

Arguments

Argument	Description
url	The URL to check the contents of.

15.14 Hdfs.Files

Returns a table containing a row for each file found at the folder **url**, {0}, and subfolders from a Hadoop file system. Each row contains properties of the file and a link to its content.

Hdfs.Files(url as text) as table

Arguments

Argument	Description
url	The URL to check the files of.

15.15 Json.Document

Returns the contents of a JSON document. The contents may be directly passed to the function as text, or it may be the binary value returned by a function like File.Contents.

Json.Document(jsonText as any, `optional encoding as nullable number`) as any

Arguments

Argument	Description
jsonText	Json formatted text.
optional encoding	The encoding value.

Examples

```
Json.Document(""glossary"": { ""title"": ""example glossary"" } }")
equals  [glossary = [title = "example glossary"]]
```

15.16 OData.Feed

Returns a table of OData feeds offered by an OData **serviceUri**.

OData.Feed(serviceUri as text, optional headers as nullable record, optional options as any) as table

Arguments

Argument	Description
serviceUri	A Uri where the OData service document is located.
optional headers	Specifying this value as a record will supply additional headers to an
	HTTP request
optional options	An optional

options Settings

Setting	Description
Query	Programmatically add query parameters to the URL without having to worry about escaping.
Headers	Specifying this value as a record will supply additional headers to a HTTP request.
ApiKeyName	If the target site has some notion of an API key, this parameter can be used to specify the name (not the value) of the key parameters that must be used in the URL. The actual key value is provided in the credentials dialog.

Timeout	Specifying this value as a duration will change the timeout for an HTTP request. The default value is 100 seconds.
EnableBatch	A logical (true/false) value that sets whether to allow generation of an OData $batch request if the MaxUriLength is exceeded (default if false).
MaxUriLength	A number that indicates the max length of an allowed URI to send to an OData service. If exceeded and EnableBatch is true, then the request will be made to an OData $batch endpoint, otherwise it will fail (default is 2048).

Remarks

- metadataUriValue is the Uri at which the OData metadata document is located (can be omitted if there is no metadata document, or if the document is at the canonical $metadata location). Optional HTTP headers may be specified with the OData feed request.
- The third parameter was previously useConcurrentRequests, a logical value. This was updated to be an options record, however specifying a logical value in its place will still work as expected.

Examples

```
OData.Feed("http://services.odata.org/Northwind/Northwind.svc")
```

15.17 Soda.Feed

Returns the resulting table of a CSV file that can be accessed using the SODA 2.0 API. The URL must point to a valid SODA-compliant source that ends in a .csv extension.

Soda.Feed(url as text) as table

Arguments

Argument	Description
url	The URL pointing to the SODA compliant .csv source

15.18 Sql.Databases

Returns a table with references to databases located on a SQL Server instance. Returns a navigation table.

Sql.Databases(server as text) as table

Arguments

Argument	Description
server	The name of the SQL Server instance.

Examples

```
Sql.Databases("localhost") equals Table.FromRecords({ [ Name = "Northwind",
Value = Table.FromRecords(..) ], [ Name = "AdventureWorks", Value =
Table.FromRecords(..) ] })
```

15.19 Sql.Database

Returns a table containing SQL tables located on a SQL Server instance database.

Sql.Database(server as text, database as text, optional options as nullable record) as record

Arguments

Argument	Description
server	The name of the SQL Server instance.
datab	The database in the SQL Server instance.
optional options	An options record to control the behavior of this function.

options Settings

Setting	Description
MaxDegreeOfParallelism as number or integer	Sets the value of the "maxdop" query clause in the generated SQL query.
CreateNavigationProperties as logical	Sets whether to generate navigation properties on the returned values. Default is true.
NavigationPropertyNameGenerator as function	Provides a function that is used for the creation of names for navigation properties.

Query as text	Provide a query text that is run on the server to return values
CommandTimeout as duration	Specifies how long the server-side query is allowed to run before it is cancelled. The default value is 10 minutes.

Examples

```
Sql.Database("localhost", "Northwind", [Query= select * from Customers])
```

15.20 Oracle.Database

Returns a table with data relating to the tables in the specified Oracle Database.

Oracle.Database(server as text, optional options as nullable record) as table

Arguments

Argument	Description
server	The name of the Oracle Database server.
optional options	An options record to control the behavior of this function.

options Settings

Setting	Description
Query as text	Provide a query text that is run on the server to return values
CommandTimeout as duration	Specifies how long the server-side query is allowed to run before it is cancelled. The default value is 10 minutes.

15.21 DB2.Database

Returns a table with data relating to the tables in the specified DB2 Database.

DB2.Database(server as text, database as text, optional options as nullable record) as table

Arguments

Argument	Description
server	The name of the DB2 Database server.
datab	The name of the database.

optional options	An options record to control the behavior of this function.

options Settings

Setting	Description
Query as text	Provide a query text that is run on the server to return values
CommandTimeout as duration	Specifies how long the server-side query is allowed to run before it is cancelled. The default value is 10 minutes.

15.22 MySQL.Database

Returns a table with data relating to the tables in the specified MySQL Database.

MySQL.Database(server as text, database as text, optional options as nullable record) as table

Arguments

Argument	Description
Server	The name of the MySQL Database server.
Database	The name of the database.
optional options	An options record to control the behavior of this function.

options Settings

Setting	Description
Query as text	Provide a query text that is run on the server to return values
CommandTimeout as duration	Specifies how long the server-side query is allowed to run before it is cancelled. The default value is 10 minutes.
TreatTinyAsBoolean as logical	Determines wheather to force tinyint columns on the server as logical values. The default value is true.
OldGuids as logical	Sets whether char(36) columns (if false) or binary(16) columns (if true) will be treated as GUIDs. The default value is true.

15.23 Sybase.Database

Returns a table with data relating to the tables in the specified Sybase Database.

Sybase.Database(server as text, database as text, optional options as nullable record) as table
Arguments

Argument	Description
Server	The name of the Sybase Database server.
Database	The name of the database.
optional options	An options record to control the behavior of this function.

options Settings

Setting	Description
Query as text	Provide a query text that is run on the server to return values
CommandTimeout as duration	Specifies how long the server-side query is allowed to run before it is cancelled. The default value is 10 minutes.

15.24 SharePoint.Contents

Returns a table containing a row for each folder and document found at the SharePoint site **url**. Each row contains properties of the folder or file and a link to its content.

SharePoint.Contents(url as text) as table

Arguments

Argument	Description
url	The URL for the SharePoint site.

15.25 SharePoint.Files

Returns a table containing a row for each document found at the SharePoint site url, and subfolders. Each row contains properties of the folder or file and a link to its content.

SharePoint.Files(url as text) as table

Arguments

Argument	Description
url	The URL for the SharePoint site.

15.26 SharePoint.Tables

Returns a table containing the result of a SharePoint List as an OData feed.

SharePoint.Tables(url as text) as table

Arguments

Argument	Description
url	The URL for the SharePoint site.

15.27 Teradata.Database

Returns a table with data relating to the tables in the specified Teradata Database.

Teradata.Database(server as text, optional options as nullable record) as table Arguments

Argument	Description
Server	The name of the Teradata server.
optional options	An options record to control the behavior of this function.

options Settings

Setting	Description
Query as text	Provide a query text that is run on the server to return values
CommandTimeout as duration	Specifies how long the server-side query is allowed to run before it is cancelled. The default value is 10 minutes.

15.28 PostgreSQL.Database

Returns a table with data relating to the tables in the specified PostgreSQL Database.

PostgreSQL.Database(server as text, optional options as nullable record) as table Arguments

Argument	Description
Server	The name of the Teradata server.
optional options	An options record to control the behavior of this function.

options Settings

Setting	Description

Query as text	Provide a query text that is run on the server to return values
CommandTimeout as duration	Specifies how long the server-side query is allowed to run before it is cancelled. The default value is 10 minutes.

15.29 Web.Contents

Returns the contents downloaded from a web `url` as a binary value.

Web.Contents(url as text, optional options as nullable record) as binary Arguments

Argument	Description
url	The URL for the Web site.
Options	An options record to control the behavior of this function.

options Fields

Field	Description
Query	Programmatically add query parameters to the URL.
ApiKeyName	Specify the name of the API key parameter for the target site. The actual key is provided in the credentials dialog.
Content	The content of the POST web request (specifying this values changes the web request from a GET to a POST).
Headers	Specifying this value as a record will supply additional headers to an HTTP request
Timeout	Specifying this value as a duration will change the timeout for an HTTP request. The default value is 100 seconds.
ExcludedFromCacheKey	Specifying this value as a record will exclude the listed HTTP header keys from being part of the calculation for caching data.
IsRetry	Specifying this logical value as true will ignore any existing response in the cache when fetching data.
ManualStatusHandling	Specifying this value as a list will prevent any builtin handling for HTTP requests whose response has one of these status codes.

Example

```
Web.Contents("www.microsoft.com") equals  The binary contents from the
URL www.microsoft.com when accessed via HTTP
```

15.30 Web.Page

Returns the contents of an HTML webpage as a table.

Web.Page(html as text) as table

Arguments

Argument	Description
html	HTML elements.

15.31 Xml.Document

Returns the contents of an XML document as a hierarchical table (list of records).

Xml.Document(contents as any, optional options as nullable record, optional encoding as nullable number) as table

Arguments

Argument	Description
contents	The contents may be directly passed to the function as text, or it may be the binary value returned by a function like File.Contents or Web.Contents.
options	Xml document options.
encoding	Encoding value.

Remarks

The output of the function has a tabular shape. Each row in the table corresponds to a node at the current level of depth. Descending into the XML tree is done through accessing the "Value" property of a given row.

The precise shape of the output table is as follows:

```
Value.Type(Xml.Document("<a></a>")) =
type {[
    Name = text,
    Namespace = text,
    Value = any,
```

```
    Attributes = {[
        Name = text,
        Namespace = text,
        Value = text
    ]}
]}
```

Examples

```
Xml.Document("<a></a>")
equals  { [
   Name = "a",
   Namespace = "",
Value = {},
   Attributes = {}
   ] }
```

15.32 Xml.Tables

Returns the contents of an XML document as a nested collection of flattened tables.

Xml.Tables(contents as any, optional options as nullable record, optional encoding as nullable number) as table

Arguments

Argument	Description
Contents	The contents may be directly passed to the function as text, or it may be the binary value returned by a function like File.Contents or Web.Contents.
optional options	Control the behavior of this function.
encoding	Encoding value.

options Settings

Setting	Description
NavigationTable	If true, output tables are navigation tables. Default value is true.

Example

```
Xml.Tables("<books>
                <book>
```

```
                    <name>Book1</name>
      </book><book>
                    <name>Book2</name>
                 </book>
              </books>")
equals
(Table.FromRecords({ [
        Name = "book",
        Table =
(Table.FromRecords({ [                  nam
e = "Book1"
            ],
[                  name =
"Book2"
            ]
        )
```

15.33 HdInsight.Containers

Returns a navigational table containing all containers found in the HDInsight account. Each row has the container name and table containing its files.

HdInsight.Containers(accountName as text) as table

Arguments

Argument	Description
accountName	The name of the HDInsight account to check.

15.34 HdInsight.Contents

Returns a navigational table containing all containers found in the HDInsight account. Each row has the container name and table containing its files.

HdInsight.Contents(accountName as text) as table

Arguments

Argument	Description
accountName	The name of the HDInsight account to check.

15.35 HdInsight.Files

Returns a table containing a row for each folder and file found at the container URL, and subfolders from an HDInsight account. Each row contains properties of the file/folder and a link to its content.

HdInsight.Files(accountName as text, containerName as text) as table

Arguments

Argument	Description
accountName	The name of the HDInsight account to check.
containerName	The name of the table.

15.36 AzureStorage.Blobs

Returns a navigational table containing all containers found in the Azure Storage account. Each row has the container name and a link to the container blobs.

AzureStorage.Blobs(accountName as text) as table

Arguments

Argument	Description
accountName	The name of the Azure Storage account to access.

15.37 AzureStorage.Tables

Returns a navigational table containing a row for each table found at the account URL from an Azure storage vault. Each row contains a link to the azure table.

AzureStorage.Tables(accountName as text) as table

Arguments

Argument	Description
accountName	The name of the Azure Storage account to access.

15.38 SapBusinessObjects.Universes

Connects to the SAP BusinessObjects BI Universe at the specified URL and returns the set of available universes.

SapBusinessObjects.Universes(url as text) as table

Arguments

Argument	Description
url	The URL of the SAP BusinessObjects BI Universe to connect to.

Remarks

The function returns a top-level table of two rows. The row with Id = “Universes” contains a nested table of all of the universes available at the URL. The row with Id = “DisplayFolders” contains a nested tree of tables representing the display folder hierarchy of the available universes.

The table of universes provides a stable path to access a particular universe by its ID.

The nested tree of display folder tables provides a user-friendly way to organize and navigate to universes and is not guaranteed to remain stable.

Examples

```
SapBusinessObjects.Universes("http://sap.example.com:6405/biprws")
```

15.39 Salesforce.Data

Connects to the Salesforce Objects API and returns the set of available objects (i.e. Accounts).

Salesforce.Data(optional loginUrl as any, optional options as nullable record) as table

Arguments

Argument	Description
optional loginUrl	The environment login URL which indicates how to connect to the account. If no environment is provided, the default (http://login.salesforce.com) is used.
optional options	An options record to control the behavior of this function.

options Settings

Setting	Description

CreateNavigationProperties as logical	Sets whether to generate navigation properties on the returned values. Default is false.

Remarks

The function returns a list of all the objects in the Salesforce model the user has access to. Objects are displayed by their display name. Custom objects and custom fields are also shown.

Examples

```
Salesforce.Data()
Salesforce.Data(null, [CreateNavigationProperties=true])
Salesforce.Data("https://test.salesforce.com")
```

15.40 Salesforce.Reports

Connects to the Salesforce Reports API and returns the set of available reports.

Salesforce.Reports(optional loginUrl as any) as table

Arguments

Argument	Description
optional loginUrl	The environment login URL which indicates how to connect to the account. If no environment is provided, the default (http://login.salesforce.com) is used.

Remarks

The function returns a list of all the reports in Salesforce the user has access to. Reports accessed through the API have a limit of 2000 rows. Reports are evaluated with their default parameters.

Examples

```
Salesforce.Reports()
Salesforce.Reports("https://test.salesforce.com")
```

15.1 AnalysisServices.Databases

Returns the Analysis Services databases on a particular host.

AnalysisServices.Databases(server as text, optional options as nullable record) as table

Arguments

Argument	Description
server	The server to connect to.
optional options	An options record to control the behavior of this function.

options Settings

Setting	Description
Culture	A culture name specifying the culture for the data. This corresponds to the “Locale Identifier” connection string property.

Remarks

The function returns the available cubes and the items under the cube using display folders. The resulting output will be a single table, based on the choice of items selected. If items are not related, an undersired result may be returned.

Examples

```
AnalysisServices.Databases(“servername”)
```

15.2 ODBC.Query

Connects to a generic provider with the given connection string and returns the result of evaluating the query.

Odbc.Query(connectionString as any, query as text) as table

Arguments

Argument	Description
connectionString	The connection string used to connect to the ODBC provider. This can be text or a record of property value pairs.
query	The query to evaluate in the provider to return results.

Remarks

Connection string properties that are secure or secret, such as credentials, should not be provided in the function. Instead they should be provided in the credentials which also has an option to provide connection string parameters.

Examples

```
Odbc.Query("Driver={SQL Anywhere 16};Host=montegoref7",  "select * from
Products")
```

```
Odbc.Query([Driver="PostgreSQL Unicode", Server="montegoref7",
Database="Northwind"], "select * from ""public""."Products"""")
```

16. Uri

16.1 Uri.Combine

Returns a Uri based on the combination of the base and relative parts.

Uri.Combine(baseUri as text, relativeUri as text) as text

Arguments

Argument	Description
baseUri	The left part of the URI to combine.
relativeUri	The right part of the URI to combine.

16.2 Uri.Parts

Returns a record value with the fields set to the parts of a Uri text value.

Uri.Parts(absoluteUri as text) as [Scheme = text, Host = text, Port = number, Path = text, Query = record, Fragment = text, UserName = text, Password = text]

Arguments

Argument	Description
absoluteUri	The absolute Uri.

Example

```
Uri.Parts("http://www.microsoft.com")
equals [
   Scheme = "http",
   Host = "www.microsoft.com",
   Port = 80,
   Path = "/",
   Query = [],
   Fragment = "",
   UserName = "",
   Password = ""
]
```

16.3 Uri.BuildQueryString

Assembles the record into a URI query string, escaping characters as necessary.

Uri.BuildQueryString(query as records) as text

Arguments

Argument	Description
Query	The record of query parameters to assemble into a query string.

Example

```
Uri.BuildQueryString([a="1", b="+$"])
equals "a=1&b=%2B%24"
```

16.4 Uri.EscapeDataString

Encodes special characters in the input data according to the reules of RFC 3986.

Uri.EscapeString(data as text) as text

Arguments

Argument	Description
data	The data as text to encode.

Example

```
Uri.EscapeDataString(“+money$”)
equals "%2Bmoney%24"
```

17. Binary Formats

The following functions are used to define and read binary file formats.

17.1 Example

Consider a hypothetical file format which contains a 32-bit unsigned integer, followed by that number of points. Each point is a record consisting of two 16-bit signed integer values (x and y). All integers in the file use Big Endian byte order.

When viewed in hex, a sample file might look like this:

```
00 00 00 02    // number of points (2)
00 03 00 04    // point (x=3, y=4)
00 05 00 06    // point (x=5, y=6)
```

This file format can be read using the binary format functions, as follows:

```
let   fileContents = #binary(       {0x00, 0x00, 0x00,
0x02,
       0x00, 0x03, 0x00, 0x04,
0x00, 0x05, 0x00, 0x06}),
pointFormat =
BinaryFormat.Record([       x =
BinaryFormat.SignedInteger16,       y
= BinaryFormat.SignedInteger16
  ]),
  fileFormat = BinaryFormat.Choice(
      BinaryFormat.UnsignedInteger32,
      (count) => BinaryFormat.List(pointFormat, count))
in
    fileFormat(fileContents)
// {
//   [x = 3, y = 4],
//   [x = 5, y = 6]
// }
```

17.2 Reading numbers

17.2.1 BinaryFormat.Byte

A binary format that reads an 8-bit unsigned integer.

BinaryFormat.Byte(binary as binary) as any

Arguments

Argument	Description
binary	An 8-bit unsigned integer

17.2.2 BinaryFormat.SignedInteger16

A binary format that reads a 16-bit signed integer.

BinaryFormat.SignedInteger16(binary as binary) as any

Arguments

Argument	Description
binary	A 16-bit signed integer

17.2.3 BinaryFormat.UnsignedInteger16

A binary format that reads a 16-bit unsigned integer.

BinaryFormat.UnsignedInteger16(binary as binary) as any

Arguments

Argument	Description
binary	A 16-bit unsigned integer

17.2.4 BinaryFormat.SignedInteger32

A binary format that reads a 32-bit signed integer.

BinaryFormat.SignedInteger32(binary as binary) as any

Arguments

Argument	Description
binary	A 32-bit signed integer

17.2.5 BinaryFormat.UnsignedInteger32

A binary format that reads a 32-bit unsigned integer.

BinaryFormat.UnsignedInteger32(binary as binary) as any

Arguments

Argument	Description
binary	A 32-bit unsigned integer

17.2.6 BinaryFormat.SignedInteger64

A binary format that reads a 64-bit signed integer.

BinaryFormat.SignedInteger64(binary as binary) as any

Arguments

Argument	Description
binary	A 64-bit signed integer

17.2.7 BinaryFormat.UnsignedInteger64

A binary format that reads a 64-bit unsigned integer.

BinaryFormat.UnsignedInteger64(binary as binary) as any

Arguments

Argument	Description
binary	A 64-bit unsigned integer

17.2.8 BinaryFormat.Single

A binary format that reads a 4-byte IEEE single-precision floating point value.

BinaryFormat.Single(binary as binary) as any

Arguments

Argument	Description
binary	A 4-byte IEEE single-precision floating point value

17.2.9 BinaryFormat.Double

A binary format that reads an 8-byte IEEE double-precision floating point value.

BinaryFormat.Double(binary as binary) as any

Arguments

Argument	Description
binary	An 8-byte IEEE double-precision floating point value

17.2.10 BinaryFormat.Decimal

A binary format that reads a .NET 16-byte decimal value.

BinaryFormat.Decimal(binary as binary) as any

Arguments

Argument	Description
binary	A .NET 16-byte decimal value

17.2.11 BinaryFormat.7BitEncodedSignedInteger

A binary format that reads a 64-bit signed integer that was encoded using a 7-bit variable-length encoding.

BinaryFormat.7BitEncodedSignedInteger(binary as binary) as any

Arguments

Argument	Description
binary	A 64-bit signed integer that was encoded using a 7-bit variable-length encoding.

17.2.12 BinaryFormat.7BitEncodedUnsignedInteger

A binary format that reads a 64-bit unsigned integer that was encoded using a 7-bit variable-length encoding.

BinaryFormat.7BitEncodedUnsignedInteger(binary as binary) as any

Arguments

Argument	Description
binary	A 64-bit unsigned integer that was encoded using a 7-bit variable-length encoding.

17.3 Controlling byte order

The default byte order for binary formats is `ByteOrder.BigEndian`. To change this use the `BinaryFormat.ByteOrder` function.

17.3.1 BinaryFormat.ByteOrder

Returns a binary format with the specified byte order.

BinaryFormat.ByteOrder(binaryFormat as function, byteOrder as number) as function

Arguments

Argument	Description
binaryFormat	The binary format that will be used to read the value.

byteOrder	The most signficant byte appears first in Big Endian byte order. The least significant byte appears first in Little Endian byte order.
	`ByteOrder.LittleEndian = 0`
	`ByteOrder.BigEndian = 1`

Example
```
let binaryData = #binary({0x01, 0x00}),
littleEndianFormat = BinaryFormat.ByteOrder(
   BinaryFormat.UnsignedInteger16, ByteOrder.LittleEndian)
in littleEndianFormat(binaryData)
equals 1
```

17.4 Reading binary data

17.4.1 BinaryFormat.Binary

Returns a binary format that reads a binary value.

BinaryFormat.Binary(optional length as nullable number) as function

Arguments

Argument	Description
optional length	Length of bytes.

Remarks

- If a `length` is specified, the binary value will contain that many bytes.
- If `length` is not specified, the binary value will contain the remaining bytes.

17.5 Reading text

17.5.1 BinaryFormat.Text

Returns a binary format that reads a text value. The optional `encoding` value specifies the encoding of the text.

BinaryFormat.Text(length as number, optional encoding as nullable number) as function

Arguments

Argument	Description
length	The number of bytes to decode.
optional encoding	The encoding of the text.

Remarks

- If the encoding is not specified, then the encoding is determined from the Unicode byte order marks.
- If no byte order marks are present, then `TextEncoding.Utf8` is used.

Example

```
// Decode two bytes as ASCII text.
let binaryData = #binary({65, 66,
   67}),
   textFormat = BinaryFormat.Text(2, TextEncoding.Ascii)
in textFormat(binaryData) equals "AB"
```

17.6 Reading lists

17.6.1 BinaryFormat.List

Returns a binary format that reads a sequence of items and returns a list.

BinaryFormat.List(binaryFormat as function, optional countOrCondition as any) as function

Arguments

Argument	Description
binaryFormat	The binary format of each item.
optional countOrCondition	See Remarks

Remarks

There are three ways to determine the number of items read:

- If the `countOrCondition` is not specified, then the binary format will read until there are no more items.
- If the `countOrCondition` is a number, then the binary format will read that many items.
- If the `countOrCondition` is a function, then that function will be invoked for each item read. The function returns `true` to continue, and false to `stop` reading items. The final item is included in the list.

Examples

```
// Read bytes until the end of the data.
let binaryData = #binary({1, 2,
   3}),
   listFormat = BinaryFormat.List(BinaryFormat.Byte)
in listFormat(binaryData)
   equals {1, 2, 3}
// Read two bytes. let binaryData =
#binary({1, 2, 3}),
   listFormat = BinaryFormat.List(BinaryFormat.Byte, 2)
in listFormat(binaryData)
   equals {1, 2}
// Read bytes until the byte value is greater than or equal to two.
let binaryData = #binary({1, 2, 3}),
   listFormat = BinaryFormat.List(BinaryFormat.Byte, (x) => x < 2)
in listFormat(binaryData)
   equals {1, 2}
```

17.7 Reading records

17.7.1 BinaryFormat.Record

Returns a binary format that reads a record. Each field in the record can have a different binary format.

BinaryFormat.Record(record as record) as function

Arguments

Argument	Description
record	The format of the record

Remarks

If a field contains a value that is not a binary format value, then no data is read for that field, and the field value is echoed to the result.

Example

```
// Read a record containing one 16-bit integer and one 32-bit integer.
let binaryData = #binary({
       0x00, 0x01,
       0x00, 0x00, 0x00, 0x02}), recordFormat
   = BinaryFormat.Record([
       A = BinaryFormat.UnsignedInteger16,
       B = BinaryFormat.UnsignedInteger32
        ]) in
recordFormat(binaryData)
equals [A = 1, B = 2]
```

17.8 Limiting input

17.8.1 BinaryFormat.Length

Returns a binary format that limits the amount of data that can be read. Both `BinaryFormat.List` and `BinaryFormat.Binary` can be used to read until end of the data. `BinaryFormat.Length` can be used to limit the number of bytes that are read.

BinaryFormat.Length(binaryFormat as function, length as number) as function

Arguments

Argument	Description
binaryFormat	The binary format to limit.
length	The number of bytes to read

Example

Limit the number of bytes read to 2 when reading a list of bytes.

```
let binaryData = #binary({1, 2, 3}), listFormat
= BinaryFormat.Length(
      BinaryFormat.List(BinaryFormat.Byte),
2) in listFormat(binaryData)  equals {1, 2}
```

17.9 Controlling what comes next

17.9.1 BinaryFormat.Choice

Returns a binary format that chooses the next binary format based on a value that has already been read.

BinaryFormat.Choice(binaryFormat as function, choice as function, optional combine as function, optional type as nullable type) as function

Arguments

Argument	Description
binaryFormat	The binary format that will be used to read the value.
choice	Choice for the next binary format.
optional combine	The function to combine the first value with the second value that was read.
optional type	The type of binary format that will be returned by the choice function. Either `type any`, `type list`, or `type binary` may be specified.

Remarks

- If `type list` or `type binary` is used, then the system may be able to return a streaming binary or list value instead of a buffered one, which may reduce the amount of memory necessary to read the format.
- The binary format value produced by this function is processed in five stages:
    1. The specified `binaryFormat` is used to read a value.
    2. The value is passed to the `choice` function.
    3. The choice function inspects the value and returns a second binary format.
    4. The second binary format is used to read a second value.
    5. The second value is returned.

- To preserve the first value read, a record binary format can be used to echo the value as a field.

Examples

// Read a list of bytes where number of elements is determined by the first byte. `let`

```
    binaryData = #binary({2, 3, 4, 5}),
listFormat = BinaryFormat.Choice(
        BinaryFormat.Byte,
        (length) => BinaryFormat.List(BinaryFormat.Byte, length)) in
    listFormat(binaryData)  equals
{3, 4}
```

// Read a list of bytes where the number of elements is determined by the first byte, and preserve the first byte read. `let`

```
    binaryData = #binary({2, 3, 4, 5}),
listFormat = BinaryFormat.Choice(
        BinaryFormat.Byte,
        (length) =>
BinaryFormat.Record([            length =
length,
            list = BinaryFormat.List(BinaryFormat.Byte, length)
        ]))
in
    listFormat(binaryData)  equals
[ length = 2, list = {3, 4} ]
```

// Read a list of bytes where number of elements is determined by the first byte using a streaming list.

```
let
    binaryData = #binary({2, 3, 4, 5}),
listFormat = BinaryFormat.Choice(
        BinaryFormat.Byte,
        (length) => BinaryFormat.List(BinaryFormat.Byte,
length),         type list) in
```

```
    listFormat(binaryData)  equals
{3, 4}
```

17.9.2 BinaryFormat.Null

A binary format that reads zero bytes and returns `null`.

BinaryFormat.Null = (binary as binary) as any => …

17.10 Reading a group of items

17.10.1 BinaryFormat.Group

Returns a binary format that reads a group of items. Each item value is preceded by a unique value called the "key value". The result is a list of item values.

```
BinaryFormat.Group = (binaryFormat as function, group as list, optional
extra as nullable function, optional lastKey as any) as function => ...
```

Arguments

Argument	Description
binaryFormat	The binary format of the key value.
group	Provides information about the group of known items.
Optional extra	Specifies a function that will return a binary format value for the value following any key that was unexpected. If the extra parameter is not specified, then an error will be raised if there are unexpected key values.
Optional lastKey	Specifies the key that signals the end of the group. If not specified, the group ends when the input ends.

The **group** parameter specifies a list of item definitions. Each item definition is a list, containing 3-5 values, as follows:

- **Key value**. The value of the key that corresponds to the item. This must be unique within the set of items.
- **Item format**. The binary format corresponding to the value of the item. This allows each item to have a different format.
- **Item occurrence**. The occurrence value for how many times the item is expected to appear in the group. Required items that are not present cause an error. Required or optional duplicate items are handled like unexpected key values.

- **Default item value (optional)**. If the default item value appears in the item definition list and is not `null`, then it will be used instead of the default. The default for repeating or optional items is `null`, and the default for repeating values is an empty list `{ }`.
- **Item value transform (optional)**. If the item value transform function is present in the item definition list and is not `null`, then it will be called to transform the item value before it is returned. The transform function is only called if the item appears in the input (it will never be called with the default value).

Examples

The following assumes a key value that is a single byte, with 4 expected items in the group, all of which have a byte of data following the key. The items appear in the input as follows:

- Key 1 is required, and does appear with value `11`.
- Key 2 repeats, and appears twice with value `22`, and results in a value of `{ 22, 22 }`.
- Key 3 is optional, and does not appear, and results in a value of `null`.
- Key 4 repeats, but does not appear, and results in a value of { }.
- Key 5 is not part of the group, but appears once with value `55`. The extra function is called with the key value 5, and returns the format corresponding to that value (`BinaryFormat.Byte`). The value `55` is read and discarded.

```
let     b =
#binary(     {
        1, 11,
        2, 22,
        2, 22,
        5, 55,
        1,
11      }),
    f = BinaryFormat.Group(
        BinaryFormat.Byte,
        {
            { 1, BinaryFormat.Byte, Occurrence.Required },
            { 2, BinaryFormat.Byte, Occurrence.Repeating },
            { 3, BinaryFormat.Byte, Occurrence.Optional },
            { 4, BinaryFormat.Byte, Occurrence.Repeating }
        },
        (extra) =>
BinaryFormat.Byte) in     f(b)

// { 11, { 22, 22 }, null, { } }
```

The following example illustrates the item value transform and default item value. The repeating item with key 1 sums the list of values read using `List.Sum`. The optional item with key 2 has a default value of 123 instead of `null`.

```
let     b =
#binary(     {
        1, 101,
        1,
102      }),
    f = BinaryFormat.Group(
        BinaryFormat.Byte,
        {
            { 1, BinaryFormat.Byte, Occurrence.Repeating,
              0, (list) => List.Sum(list) },
            { 2, BinaryFormat.Byte, Occurrence.Optional, 123 }
        }) in
f(b) // { 203,
123 }
```

17.10.2 Occurrence values

Occurrence values are used with **BinaryFormat.Group** to specify how many times an item in a group is expected to appear.

```
Occurrence.Optional = 0
Occurrence.Required = 1
```

17.11 Transforming what was read

17.11.1 BinaryFormat.Transform

Returns a binary format that will transform the values read by another binary format.

BinaryFormat.Transform(binaryFormat as function, transform as function) as function

Arguments

Argument	Description
binaryFormat	The binary format that will be used to read the value.
transform	Invoked with the value read, and returns the transformed value.

Example

// Read a byte and add one to it. `let`

```
    binaryData = #binary({1}),
transformFormat = BinaryFormat.Transform(

BinaryFormat.Byte,
(x) => x + 1) in
transformFormat(binaryData)  equals
2
```

18. Binary

The following functions interact with binary data.

18.1 Binary.FromText

Decodes data from a text form into binary.

Binary.FromText (text as text, encoding as number) as Binary

Arguments

Argument	Description
text	The text to decode.
encoding	The encoding option to apply.

Binary encoding

- `BinaryEncoding.Base64 = 0;`
- `BinaryEncoding.Hex = 1;`

18.2 Binary.ToText

Encodes binary data into a text form.

Binary.ToText(binary as binary, encoding as number) as text

Arguments

Argument	Description
binary	The binary data to encode.

encoding	The encoding option to apply.

Binary encoding

- `BinaryEncoding.Base64 = 0;`
- `BinaryEncoding.Hex = 1;`

18.3 Binary.From

Returns a binary value from the given value.

Binary.From(Value as any, optional encoding as nullable number) as nullable binary

Arguments

Argument	Description
value	A binary value from the text representation. For more information, see Binary.FromText.
optional encoding	The encoding option to apply.

Binary encoding

- `BinaryEncoding.Base64 = 0;`
- `BinaryEncoding.Hex = 1;`

Remarks

- If the given value is null, Binary.From returns null. If the given value is binary, the same value is returned.
- Values of the following types can be converted to a binary value:
- If value is of any other type, an error is returned.

Example

```
Binary.From("1011") equals Binary.FromText("1011",
BinaryEncoding.Base64)
```

18.4 Binary.Combine

Combines a list of binaries into a single binary.

Binary.Combine(binaries as list) as binary

Arguments

Argument	Description
binaries	The list of binary values to combine.

18.5 Binary.Length

Returns the length of binary values.

Binary.Length(binary as binary) as number

Arguments

Argument	Description
binary	The binary value to check.

18.6 Binary.ToList

Converts a binary value into a list of numbers

Binary.ToList(binary as binary) as list

Arguments

Argument	Description
binary	The binary value to convert.

18.7 Binary.FromList

Converts a list of numbers into a binary value

Binary.FromList(list as list) as binary

Arguments

Argument	Description
list	The list to convert.

18.8 Binary.Buffer

Buffers the binary value in memory. The result of this call is a stable binary value, which means it will have a deterministic length and order of bytes.

Binary.FromBuffer (binary as nullable binary) as nullable binary

Arguments

Argument	Description
Binary	The binary value to buffer in memory.

19. Lines

The following functions relate to lines.

19.1 Lines.FromText

Converts a text value to a list of text values split at lines breaks.

Lines.FromText(text as text, optional quoteStyle as nullable number, optional includeLineSeparators as nullable logical) as list

Arguments

Argument	Description
text	The text to parse.
optional quoteStyle	Style of quote.
optional includeLineSeparators	If includeLineSeparators is true, then the line break characters are included in the text. If a delimiter is specified, then line breaks may appear within quotes

19.2 Lines.FromBinary

Converts a binary value to a list of text values split at lines breaks.

Lines.FromBinary(binary as Binary, optional quoteStyle as nullable number, optional includeLineSeparators as nullable logical, optional encoding as nullable number) as list

Arguments

Argument	Description
binary	The binary value to parse.
optional quoteStyle	Determines whether there is quoting within the value that should be used to preserve line breaks and for which delimiters are not significant.

optional includeLineSeparators	Determines whether the line break characters are included in the line. This is useful when the actual line break is significant and needs to be preserved. If not specified, then it defaults to false.If includeLineSeparators is true, then the line break characters are included
	in the text.
encoding	The encoding value.

quoteStyle Settings

Setting	Description
`QuoteStyle.None (default)`	No quoting behavior is needed.
`QuoteStyle.Csv`	Quoting is as per Csv. A double quote character is used to demarcate such regions, and a pair of double quote characters is used to indicate a single double quote character within such a region.

Remarks

If a delimiter is specified, then line breaks may appear within quotes.

19.3 Lines.ToBinary

Converts a list of text into a binary value using the specified encoding and lineSeparator.The specified lineSeparator is appended to each line. If not specified then the carriage return and line feed characters are used.

Lines.ToBinary(lines as list, optional lineSeparator as nullable text, optional encoding as nullable number, optional includeByteOrderMark as nullable logical)as binary

Arguments

Argument	Description
lines	The list of lines to convert.
optional lineSeparator	Determines whether the line break characters are included in the line. This is useful when the actual line break is significant and needs to be preserved. If not specified, then it defaults to false.If includeLineSeparators is true, then the line break characters are included in the text.

optional encoding	The encoding option to apply.
optional includeByteOrderMark	The byte order mark to include.

Binary encoding

- `BinaryEncoding.Base64 = 0;`
- `BinaryEncoding.Hex = 1;`

19.4 Lines.ToText

Converts a list of text into a single text. The specified lineSeparator is appended to each line. If not specified then the carriage return and line feed characters are used.

Lines.ToText (lines as list, optional lineSeparator as nullable text) as text

Arguments

Argument	Description
lines	The list of lines to convert.
optional lineSeparator	Determines whether the line break characters are included in the line. This is useful when the actual line break is significant and needs to be preserved. If not specified, then it defaults to false.If includeLineSeparators is true, then the line break characters are included in the text.

Examples

```
Lines.FromText("A,""B#(cr)C""#(cr)#(lf)1,2", true, null)
{
"A,""B#(cr)",
"C""#(cr)#(lf)",
"1,2"
}

Lines.FromText("A,""B#(cr)C""#(cr)#(lf)1,2")
{
"A,""B",
"C""",
"1,2"
}

Lines.FromText("A,""B#(cr)C""#(cr)#(lf)1,2", null, ",")
```

```
{
"A,""B#(cr)C""",
"1,2"
}
```

20. Expression

These function relate to expression evaluation.

20.1 Expression.Evaluate

Evaluates the a Text expression and returns the evaluated value.

Expression.Evaluate(expression as text, optional environment as [...]) as any

Arguments

Argument	Description
expression	The expression to evaluate.
optional environment	The expression environment.

Examples

```
   Expression.Evaluate("1 + 1") // 2
   Expression.Evaluate("1 +")   // Error
   Expression.Evaluate(
      "section Section1; shared X = 1;"
)  equals  Error, only expressions are supported
   Expression.Evaluate(
      "Record.Field([A=1], ""A"")"
)  equals  error. Unknown identifier "Record.Field".
   Expression.Evaluate(
      "Record.Field([A=1], ""A"")",
      [Record.Field              =
Record.Field] )  equals  1  let x =
1
in
      Expression.Evaluate("x") // Error. Unknown identifier "x".
    let x =
   1
in
      Expression.Evaluate(
```

```
        "x",
        #shared
    ) // Error. Unknown identifier "x".
 let x =
1
in
    Expression.Evaluate("x", [x = x]) // 1
 section; shared
MyText = "ABC";
MyResult = Expression.Evaluate(
    "Text.StartsWith(MyText, ""A"")",
    #shared
); // true
section;
MyText = "ABC";
MyResult = Expression.Evaluate(
    "Text.StartsWith(MyText, ""A"")",
    #shared
); // Error. Unknown identifier "MyText" (since MyText isn't shared).
```

20.2 Expression.Constant

Returns a constant text literal from a value.

Expression.Constant(value as any) as text

Examples

```
Expression.Constant(1)  equals  "1"
Expression.Constant(1 + 1)  equals  "2"
Expression.Constant(true)  equals  "true"
Expression.Constant("abc")  equals  """abc"""
Expression.Constant("#(tab)")  equals  """#(#)(tab)"""
Expression.Constant(#date(2011, 1, 1))  equals  "#date(2011, 1, 1)"
Expression.Constant((x) => x)  equals  Error: Functions not supported
Expression.Constant({1, 2, 3})  equals  Error: Lists not supported
Expression.Constant([a = 1 + 1])  equals  Error: Records not supported
```

20.3 Expression.Identifier

Returns a text value that can be used as an identifier from a text value.

Expression.Identifier(name as text) as text

Arguments

Argument	Description
name	The text to identify.

Examples

```
Expression.Identifier("foo") equals "foo"
Expression.Identifier("10 lbs") equals "#""10 lbs"""
Expression.Identifier("try") equals "#""try"""
Expression.Identifier("") equals "#""""
Expression.Identifier(null) equals Error
```

Example of combined use

```
Expression.Evaluate(
    // "let x = 1 in x"
      "let " &
         Expression.Identifier("x") & " = " & Expression.Constant(1) &
      " in " &
         Expression.Identifier("x")
) equals 1
```

21. Function

These functions relate to Function values.

21.1 Function

Returns the type that represents all function values

21.2 Function.Invoke

Invokes the given function using the specified arguments and returns the result.

Function.Invoke(function as function, args as list) as any

Arguments

Argument	Description
function	The function to invoke.
args	The list of required arguments.

Examples

```
Function.Invoke(Record.FieldNames, {[A=1,B=2]}) equals {"A", "B"}
```

21.3 Function.InvokeAfter

Invokes the given function after the specified duration has passed.

Function.Invoke(function as function, delay as duration) as any

Arguments

Argument	Description
function	The function to invoke.
Delay	The duration of the delay.

Examples

```
Function.Invoke(() => 1, #duration(1, 0, 0, 0))
```

22. Error

The following functions deal with error handling.

22.1 Error.Record

Returns a record containing fields “Reason”, “Message”, and “Detail” set to the provided values. The record can be used to raise or throw an error.

Error.Record(reason as text, message as text, detail as any) as record

Arguments

Argument	Description
reason	The value to set the Reason as.
message	The value to set the Message as.
detail	The value to set the Detail as.

Examples

```
error Error.Record("InvalidCondition","An error has occured", null)
equals  error with Reason: “InvalidCondition” and Message “An error has
occurred”
```

23. Comparer

The following related to value comparison, relative to an environment and locale.

23.1 Comparer.FromCulture

Returns a comparer function given the culture and a logical value for case sensitivity for the comparison. The default value for `ignoreCase` is false. The value for culture are well known text representations of locales used in the .NET framework.

Comparer.FromCulture(culture as text, optional ignoreCase as nullable logical) as function

Arguments

Argument	Description
culture	A text value corresponding to the culture values supported on your version of Windows, such as `"en-US"`. If the culture is not specified, the current user culture is used. For a list of culture names, see National Language Support (NLS) API Reference.
optional ignoreCase	Logical value whether or not to ignore the case.

Examples

```
 let
  comparer1 = Comparer.FromCulture("en-us", false),
comparer2 = Comparer.FromCulture("en-us", true)     in
[
       Test1 =  comparer1("a","A"),  equals  -1
   Test2 =  comparer2("a","A")  equals  0
     ]
```

23.2 Comparer.Equals

Returns a logical value based on the equality check over the two given values.

Comparer.Equals(comparer as function, x as any, y as any) as logical

Arguments

Argument	Description
comparer	The comparer function is used to perform the operation.
x	The left value to compare.
y	The right value to compare.

Example `let`

```
  comparer1 = Comparer.FromCulture("en-us", false),
comparer2 = Comparer.FromCulture("en-us", true)     in
[
         Test1 =  Comparer.Equals(comparer1,"a","A"),  equals  false
       Test2 = Comparer.Equals(comparer2,"a","A")  equals  true
     ]
```

23.3 Comparer.Ordinal

Returns a comparer function which uses Ordinal rules to compare values.

Comparer.Ordinal(x as any, y as any) as number

Arguments

Argument	Description
x	The left value to compare.
y	The right value to compare.

Examples

```
Comparer.Equals(Comparer.Ordinal, "a","A")  equals false
```

23.4 Culture.Current

Returns the current culture of the system.

Example:

```
Culture.Current default equals “en-US”
```

24. Splitter

The following functions deal with splitter functions. Splitter functions are used by other functions in the library such as `Table.FromList` and `Table.SplitColumn` to create tabular structure from a given list or column values.

24.1 Parameter values

Quote styles

```
QuoteStyle.None = 0;
QuoteStyle.Csv  = 1;
```

24.2 Splitter.SplitTextByDelimiter

Returns a function that will split text according to a delimiter.

Splitter.SplitTextByDelimiter(delimiter as text, optional quoteStyle as nullable number) as function

Arguments

Argument	Description
delimiter	The delimiter character is used to identify at what points to split the string. The delimiter character is not included in the split values. A trailing delimiter character will yield an additional empty text value. The split values contain all characters between the delimiters. This function will always produce at least one value.
optional quoteStyle	The quoteStyle acts as described in Lines.FromText. By default, it is QuoteStyle.Csv.

Quote styles

- `QuoteStyle.None = 0;`
- `QuoteStyle.Csv  = 1;`

24.3 Splitter.SplitTextByEachDelimiter

Returns a function that splits text by each delimiter in turn.

Splitter.SplitTextByEachDelimiter(delimiters as list, optional quoteStyle as nullable number) as function

Arguments

Argument	Description
Delimiters	The delimiter characters are used to identify at what points to split the string. The delimiter character is not included in the split values. A trailing delimiter character will yield an additional empty text value. The split values contain all characters between the delimiters. This function will always produce at least one value.
optional quoteStyle	The quoteStyle acts as described in Lines.FromText. By default, it is QuoteStyle.Csv.

Remarks

Splitter.SplitTextByEachDelimiter is similar to Splitter.SplitTextByDelimiter except that each delimiter is used once in order to determine the points at which to break the text.

24.4 Splitter.SplitTextByAnyDelimiter

Returns a function that splits text by at any of the delimiters.

Splitter.SplitTextByEachDelimiter(delimiters as list, optional quoteStyle as nullable number) as function

Arguments

Argument	Description
Delimiters	The delimiter characters are used to identify at what points to split the string. The delimiter character is not included in the split values. A trailing delimiter character will yield an additional empty text value. The split values contain all characters between the delimiters. This function will always produce at least one value.
optional quoteStyle	The quoteStyle acts as described in Lines.FromText. By default, it is QuoteStyle.Csv.

Remarks

Splitter.SplitTextByAnyDelimiter is similar to Splitter.SplitTextByDelimiter except that multiple delimiters may be used to specify the points at which to break the text.

24.5 Splitter.SplitTextByWhitespace

Returns a function that splits text according to whitespace.

Splitter.SplitTextByWhitespace(optional quoteStyle as nullable number) as function

Arguments

Argument	Description
optional quoteStyle	The quoteStyle acts as described in Lines.FromText. By default, it is QuoteStyle.Csv.

Remarks

- Splitter.SplitTextByWhitespace is similar to SplitTextByAnyDelimiter where the delimiters provided are all characters for which char.IsWhitespace returns true.
- SplitTextByWhitespace will consider any non-zero sequence of whitespace characters a delimiter.

24.6 Splitter.SplitTextByRanges

Returns a function that splits text according to the specified ranges.

Splitter.SplitTextByRanges(ranges as list) as function

Arguments

Argument	Description
ranges	The ranges to split on.

Remarks

Each item in ranges should specify a tuple of offset and length (where offset zero refers to the first character). The subset of characters of the line denoted by each tuple is returned as a separate item. If the offset or length is less than zero, an error is thrown. Otherwise, if the tuple is out of range of the line, spaces are used to fill out the value. Therefore, the list returned will have the same cardinality as ranges, and each item will be of the length specified in the corresponding tuple. There is no checking for overlap of tuple ranges.

24.7 Splitter.SplitTextByPositions

Returns a function that splits text according to the specified positions.

Splitter.SplitTextByPositions(positions as list) as function

Arguments

Argument	Description
positions	The positions to split on.

Remarks

Each item in positions should be a non-negative number indicating the position at which to break the text, and each item must be greater than or equal to the previous.

SplitTextByPositions works by computing a set of ranges by using the difference between subsequent positions as lengths (with the last position of effectively infinite length) and delegating to SplitTextByRanges. The list returned will have the same cardinality as that of the positions.

24.8 Splitter.SplitTextByLengths

Returns a function that splits text according to the specified lengths.

Splitter.SplitTextByLengths(lengths as list) as function

Arguments

Argument	Description
lengths	The lengths to split on.

Remarks

Each item in lengths should be a non-negative number indicating the number of characters to use for each item. SplitTextByLengths works by computing a set of ranges by adding each subsequent length to compute the next position, and delegating to SplitTextByRanges. The list returned will have the same cardinality as that of the positions.

25. Combiner

The following functions deal with combiner functions. Combiner functions are used by other library functions that merge values, such as `Table.ToList` and `Table.CombineColumns`. The function is applied to each row in the table to produce a single value for each row.

25.1.1 Combiner.CombineTextByDelimiter

Returns a function that takes a list of text and merges them into a single text.

Combiner.CombineTextByDelimiter(delimiters as list, optional quoteStyle as nullable number) as function

Arguments

Argument	Description
delimiters	Separates the values.
optional quoteStyle	Determines whether there is quoting within the value that should be used to preserve line breaks and for which delimiters are not significant.

quoteStyle Settings

Setting	Description
`QuoteStyle.None`	The text in the list is concatenated.
`QuoteStyle.Csv (default)`	Values containing quotes, line feeds, or the specified delimiter are escaped to conform to the escaped production of CSV. For more information, see **Error! Reference source not found**

25.1.2 Combiner.CombineTextByEachDelimiter

Returns a function that takes a list of text and merges them into a single text.

Combiner.CombineTextByDelimiter(delimiters as list, optional quoteStyle as number) as function

Arguments

Argument	Description
delimiters	The delimiter characters are used to identify at what points to split the string. The delimiter character is not included in the split values. A trailing delimiter character will yield an additional empty text value. The split values contain all characters between the delimiters. This function will always produce at least one value.
optional quoteStyle	Determines whether there is quoting within the value that should be used to preserve line breaks and for which delimiters are not significant.

Remarks

- Combiner.CombineTextByEachDelimiter is similar to CombineTextByDelimiter except that each delimiter is used in turn.
- An error is thrown by the resulting function if the cardinality of the line passed to it exceeds the cardinality of the delimiters.

25.1.3 Combiner.CombineTextByRanges

Returns a function that merges a list of text into a single text.

Combiner.CombineTextByRanges(ranges as list, optional template as nullable text) as function

Arguments

Argument	Description
ranges	The ranges to combine at.
optional template	The Combiner template.

Remarks

- Each position identifies a tuple of position and length for a line where the text value should be placed. If the length of a text value for a given tuple exceeds the length specified by that tuple, then the value is truncated to fit. There is no checking for overlap of tuple ranges. If there are fewer text items than ranges, empty text will be used. If there are fewer ranges than text items, then they will not be emitted.

- The template specifies the default characters over which the individual items are placed. If not specified empty text is used.

25.1.4 Combiner.CombineTextByPositions

Returns a function that merges a list of text into a single text.

Combiner.CombineTextByPositions(positions as list, optional template as nullable text) as function

Arguments

Argument	Description
positions	The positions to combine at.
optional template	The Combiner template.

Remarks

This function behaves similar to CombineTextByRanges, except that the positions are used to determine the locations of the text. As in Splitter.SplitTextByRanges, each position must be non-negative and larger than the previous position. As in SplitTextByPositions, CombineTextByPositions works by delegating to CombineTextByRanges.

25.1.5 Combiner.CombineTextByLengths

Returns a function that merges a list of text into a single text.

Combiner.CombineTextByLengths(lengths as list, optional template as nullable text) as function

Arguments

Argument	Description
lengths	The lengths to combine on.
optional template	The Combiner template.

Remarks

- Combiner.CombineTextByLengths is similar to CombineTextByRanges, except that the lengths are used to determine the locations of the text.
- As in Splitter.SplitTextByLengths, each length must be non-negative.
- As in SplitTextByLengths, CombineTextByLengths works by delegating to CombineTextByRanges.

26. Replacer

The following functions deal with replacer functions. Replacer functions are used by other functions in the library to replace a given value in a structure.

26.1 Replacer.ReplaceText

This function be provided to List.ReplaceValue or Table.ReplaceValue to do replace of text values in list and table values respectively.

Replacer.ReplaceText (text as nullable text, old as text, new as text) as nullable text

Arguments

Argument	Description
text	The text value to make the replacement in.
old	The old value to replace.
new	The new value to replace with.

26.2 Replacer.ReplaceValue

This function be provided to List.ReplaceValue or Table.ReplaceValue to do replace values in list and table values respectively.

Replacer.ReplaceValue(value as any, old as any, new as any) as any

Arguments

Argument	Description
value	The value to make replacements in.
old	The old value to replace.
new	The new value to replace with.

27. Type

The following functions relate to Type values.

27.1 Type.ClosedRecord

The given type must be a record type returns a closed version of the given record type (or the same type, if it is already closed)

Type.ClosedRecord(#"type" as type) as type

Example

```
Type.ClosedRecord( type [ A = number,…] ) equals type [A=number]
```

27.2 Type.ForFunction

Creates a function type from the given arguments.

Type.ForFunction(signature as record, min as number) as type

Arguments

Argument	Description
signature	A Record value that contains fields for ReturnType and Parameters. Parameters is itself a record with all the parameter values assigned to their expected types.
min	The minium number of arguments required to invoke the function.

Example

```
Type.ForFunction([ReturnType = number, Parameters = [X = number]], 1)
```

27.3 Type.ForList

Returns a list type for an item type.

Type.ForList(itemType as list) as type

Arguments

Argument	Description
itemType	The itemType to convert.

Example

```
Type.ForList({type number}) equals type {number}
```

27.4 Type.ForNullable

Returns a nullable type for a type.

Type.ForNullable(itemType as list) as type

Arguments

Argument	Description
itemType	The type to convert.

Example

```
Type.ForNullable({type number}) equals type nullable number
```

27.5 Type.ForRecord

Returns a Record type from a fields record.

Type.ForRecord(fields as record, open as logical) as type

Arguments

Argument	Description
fields	The record to convert.
open	A logical value indicating if the returned type should be an open record.

Example

```
Type.ForRecord([X = [Type = type number, Optional = false], Y = [Type
= type number, Optional = true]], true)  equals type [ X = number,
optional Y = number,... ]
```

27.6 Type.ForTable

Constructs a table type from a row type as a record type.

Type.ForTable((rowType as type) as type

Arguments

Argument	Description
rowType	The row type to convert.

Example

```
Type.ForTable(type []) equals type table []
Type.ForTable(type [A = number])  equals  type table [A = number]
```

27.7 Type.FunctionParameters

Returns a record with field values set to the name of the parameters of a function type, and their values set to their corresponding types.

Type.FunctionParameters(functionType as type) as record

Arguments

Argument	Description
functionType	The function type to check.

Examples

```
Type.FunctionParameters(type function () as any) equals  []
Type.FunctionParameters(type function (x as number, y as text) as any)
equals [ x = number, y = text ]
```

27.8 Type.FunctionReturn

Returns a type returned by a function type.

Type.FunctionReturn(type as type) as type

Examples

```
Type.FunctionReturn(type function () as any) equals type any
Type.FunctionReturn(type function () as [A = number]) equals type [A =
number]
```

27.9 Type.FunctionRequiredParameters

Returns a number indicating the minimum number of parameters required to invoke the a type of function.

Type.FunctionRequiredParameters(#"type" as type) as number

Examples

```
Type.FunctionRequiredParameters( type function () as any) equals 0
Type.FunctionRequiredParameters( type function (x as number) as any)
equals 1
```

27.10 Type.IsNullable

Returns true if a type is a nullable type; otherwise, false.

Type.IsNullable(#"type" as type) as logical

Examples

```
Type.IsNullable(type nullable number) equals true
Type.IsNullable(number) equals false
```

27.11 Type.IsOpenRecord

Returns whether a record type is open.

Type.IsOpenRecord(#"type" as type) as logical

Examples

```
Type.IsOpenRecord(type [ A = number,…]) equals true
Type.IsOpenRecord(type [ A = number]) equals false
```

27.12 Type.ListItem

Returns an item type from a list type.

Type.ListItem(#"type" as type) as type

Example

```
Type.ListItem (type { number }) equals type number
```

27.13 Type.NonNullable

Returns the non nullable type from a type.

Type.NonNullable(#"type" as type) as type

Example

```
Type.NonNullable(type nullable number) equals type number
```

27.14 Type.OpenRecord

Returns an opened version of a record type, or the same type, if it is already open.

Type.OpenRecord(#"type" as type) as type

Example

```
Type.OpenRecord( type [ A = number] ) equals type [ A = number, …]
```

27.15 Type.RecordFields

Returns a record describing the fields of a record type with each field of the returned record type having a corresponding name and a value that is a record of the form `[ Type = type, Opional = logical ]`.

Type.RecordFields(#"type" as type) as record

Example

```
Type.RecordFields(type [ A = number, optional B = any])
equals [
A = [Type = number, Optional = false],
B = [Type = any, Optional = true]
]
```

27.16 Type.TableRow

Returns a row type from a table type.

Type.TableRow(table as type) as type

Example

```
Type.TableRow(
       Value.Type(
          Value.ReplaceType(
          Table.FromRecords({[A=1]})
           , type table [ A = number] )
       )
    )
         equals type [    A =
number  ]
```

27.17 Type.TableKeys

Returns keys from a table type.

Type.TableKeys(tableType as type) as list

27.18 Type.AddTableKey

Add a key to a table type.

Type.AddTableKey (table as type, columns as list, isPrimary as logical) as type

Arguments

Argument	Description
table	The table type to modify.
columns	Columns that define the key.
isPrimary	Logical stating whether or not this is the primary key.

Example

```
Type.AddTableKey(tableType, {"A", "B"}, false) equals  add a non-primary
key that combines values from columns A and B
```

27.19 Type.ReplaceTableKeys

Replaces the keys in a table type.

Type.ReplaceTableKeys(tableType as type, keys as list) as type

Arguments

Argument	Description
tableType	The table type to modify.
keys	The list of keys to replace.

Example

```
Type.ReplaceTableKeys(tableType, {}) equals  returns type value with all
keys removed
```