APPENDIX

Formula Language Reference

This appendix documents functions and operators available in the Tableau formula language. Add these functions to a calculated field by typing the beginning of the desired function into an ad hoc calculation or calculated field. Select the full function from the drop-down list when it appears, or just continue to type the full function name. In the calculated field editor, you may display the function list by clicking the small arrow on the right side of the editor. The function list displays all available functions in alphabetical order, or a subset of related functions in alphabetical order if a category is chosen in the drop-down list above the function list. Most functions accept one or more *arguments,* or values that are supplied to the function inside parentheses and separated by commas.

Number Functions

These functions perform numeric operations and return numeric results. They also accept numeric arguments.

ABS

Returns the absolute value of a number.

ABS(number)

```
ABS([Profit])
```

returns the numeric value 74.25 if the [Profit] database field contains –74.25.

ACOS

Returns the arc cosine (inverse cosine), in radians, of the supplied number.

ACOS(number)

```
ACOS(-.5)
```

returns the numeric value 2.094395102.

ASIN

Returns the arc sine (inverse sine), in radians, of the supplied number.

ASIN(number)

```
ASIN(.5)
```

returns the numeric value .523598776.

ATAN

Returns the arc tangent (inverse tangent), in radians, of the supplied number.

ATAN(number)

```
ATAN(45)
```

returns the numeric value 1.548577785.

ATAN2

Returns the arc tangent (inverse tangent), in radians, of the two supplied numbers.

ATAN2(y number, x number)

```
ATAN2(45, 10)
```

returns the numeric value 1.352127433.

CEILING

Returns either an integer value of the argument if it contains no decimal places or the next highest integer value of the argument if it contains decimal places.

CEILING(number)

```
CEILING(25.675)
```

returns the integer value 26.

Note *CEILING is available with Tableau Data Extract data sources, as well as other limited sets of data sources. If your data source doesn't support CEILING, it won't appear in the calculated field editor function list.*

COS

Returns the cosine, in radians, of the supplied number.

COS(number)

```
COS(45)
```

returns the numeric value .525321960.

COT

Returns the cotangent, in radians, of the supplied number.

COT(number)

```
COT(45)
```

returns the numeric value .617369592.

DEGREES

Returns a value, in degrees, of the supplied radian number.

DEGREES(number)

```
DEGREES(.7865)
```

returns the numeric value 45.063130587.

DIV

Divides two arguments, returning the integer (nonfractional) result.

DIV(number1, number2)

```
DIV(100,33)
```

returns 3.000, whereas 100 / 33 returns 3.030.

EXP

Raises *e* (the base of the natural logarithm) to the specified power.

EXP(number)

```
EXP(10)
```

returns the numeric value 22,026.46484375.

HEXBINX

Returns the nearest x coordinate in a hexagonal shape (referred to as a hexagonal *bin*). Typically, latitudes and longitudes are supplied as arguments to permit hexagonal mapping rather than finer point mapping.

HEXBINX(longitude, latitude)

```
HEXBINX(-105.3364,39.6389)
```

returns -105.0.

HEXBINY

Returns the nearest y coordinate in a hexagonal shape (referred to as a hexagonal *bin*). Typically, latitudes and longitudes are supplied as arguments to permit hexagonal mapping rather than finer point mapping.

HEXBINY(longitude, latitude)

```
HEXBINY(-105.3364,39.6389)
```

returns 39.84.

LN

Returns the natural logarithm of the supplied number (or a null value if the specified number is <= 0).

LN(number)

```
LN(10)
```

returns the numeric value 2.302585125.

LOG

Returns the logarithm of the supplied number. An optional second argument, base, is used to supply a numeric base. If the second argument is not supplied, base 10 is used.

LOG(number, *[base]*)

```
LOG(100)
```

returns the numeric value 2.

```
LOG(100, 5)
```

returns the numeric value 2.861353159.

MAX

Returns either the largest aggregate value of a supplied database field if used with one argument or the larger of two supplied values if used with two arguments.

MAX(expression)
MAX(expression1, expression2)

```
MAX([Discount])
```

returns the numeric value .25 if the highest discount value in the underlying set of data records is .25 (25 percent).

```
MAX([January Sales], [February Sales])
```

returns the numeric value 24,103.00 if January Sales is 24,103 and February Sales is 18,109.25.

MIN

Returns either the smallest aggregate value of a supplied database field if used with one argument or the smaller of two supplied values if used with two arguments.

MIN(expression)
MIN(expression1, expression2)

```
MIN([Discount])
```

returns the numeric value .25 if the lowest discount value in the underlying set of data records is .25 (25 percent).

```
MIN([January Sales], [February Sales])
```

returns the numeric value 18,109.25 if January Sales is 24,103 and February Sales is 18,109.25.

PI

Returns the numeric constant of pi. This function takes no arguments, but still requires parentheses.

PI()

```
PI()
```

returns the numeric value 3.141592654.

POWER

Raises the supplied numeric value by the power specified in the second numeric argument. The ^ operator may be used for the same purpose.

POWER(number, power)

```
POWER([Rating], 3)
```

returns the numeric value 125 for a Rating field value of 5.

```
[Rating] ^ 3
```

will return the same result.

RADIANS

Returns a value, in radians, of the supplied degree number.

RADIANS(number)

```
RADIANS(45)
```

returns the numeric value .785398163.

ROUND

Rounds the supplied numeric value. If the second numeric argument is supplied, ROUND rounds to that number of significant digits. Otherwise, ROUND rounds to a whole number.

ROUND(number, *[decimals]*)

```
ROUND([Weight])
```

returns 185 for a Weight database field value of 185.724.

```
ROUND([Weight], 1)
```

returns 185.7 for a Weight database field value of 185.724.

SIGN

Returns a numeric value indicating whether the supplied numeric value is positive, negative, or zero. SIGN returns a numeric value of –1 for negative numbers, +1 for positive numbers, and 0 for zero.

SIGN(number)

```
CASE SIGN([Profit])
       WHEN 1 THEN "Profit"
       WHEN -1 THEN "Loss"
       ELSE "At Cost"
END
```

returns the string value “Profit” if the Profit database field is greater than zero, “Loss” if the field is less than zero, and “At Cost” if the field is exactly zero.

SIN

Returns the sine, in radians, of the supplied number, also in radians.

SIN(number)

```
SIN(5)
```

returns the numeric value –0.958924294.

SQRT

Returns the square root of the supplied number.

SQRT(number)

```
SQRT(64)
```

returns the numeric value 8.

SQUARE

Returns the square (raised to the power of 2) of the supplied number. The POWER function and ^ operator may also be used.

SQUARE(number)

```
SQUARE(8)
```

returns the numeric value 64.
Equivalents are

```
POWER(8, 2)
8 ^ 2
```

TAN

Returns the tangent, in radians, of the supplied number.

TAN(number)

```
TAN(1.25)
```

returns the numeric value 3.009569674.

ZN

Returns the numeric value of the supplied number if it is not null. Otherwise, returns 0.

ZN(expression)

```
ZN([Salary])
```

returns 125,000 if the Salary database field contains the value 125,000. If the Salary database field contains a null value, the result will be 0 instead of a null.

String Functions

These functions perform string operations and return various results (either strings or numbers). Depending on the function, they accept a combination of string and/or numeric arguments.

ASCII

Returns the American Standard Code for Information Interchange (ASCII) numeric value for the first character in the supplied string.

ASCII(string)

```
ASCII([Middle Initial])
```

returns the numeric value 69 if the Middle Initial database field contains a capital E.

CHAR

Converts the supplied integer numeric value into a single string character, based on the American Standard Code for Information Interchange (ASCII) value of the supplied number.

CHAR(integer)

```
[Company Name] + CHAR(10) + [City] + ", " + [State]
```

returns

Tableau Software
Seattle, WA

for a Company Name of "Tableau Software," City of "Seattle," and a State of "WA." The ASCII number 10 equates to a line feed character.

CONTAINS

Determines whether the value in the first string argument contains the value in the second string argument. If so, the function returns True. Otherwise, it returns False.

CONTAINS(string, substring)

```
IF CONTAINS([Product Name], "BD") THEN
       "BluRay Media"
ELSE
       "DVD Media"
END
```

returns the string "BluRay Media" if the characters "BD" are contained anywhere within the Product Name database field. Otherwise, the result is "DVD Media."

ENDSWITH

Determines whether the value in the first string argument ends with the value in the second string argument. If so, the function returns True. Otherwise, it returns False.

ENDSWITH(string, substring)

```
IF ENDSWITH([Product Name], "-BD") THEN
       "BluRay Media"
ELSE
       "DVD Media"
END
```

returns the string "BluRay Media" if the characters "-BD" appear at the end of the Product Name database field. Otherwise, the result is "DVD Media."

FIND

Searches the first value in the first string argument for the value in the second substring argument, returning a numeric value indicating the position where the second argument exists. If the optional third numeric argument is provided, FIND starts the search at that position. If the search fails, FIND returns 0.

FIND(string, substring, *[start]*)

```
FIND([Product Type-SKU], "-"]
```

returns 5 for a Product Type-SKU value of "SOFT-A103881."

```
FIND([Product Type-SKU], "-", 10]
```

returns 0 for a Product Type-SKU value of "SOFT-A103881."

FINDNTH

Searches the value in the first string argument for the *nth* occurrence in the substring value in the second string argument, where the number of occurrences to search for is supplied by the numeric third argument. The result is a numeric value indicating the position where the second argument exists within the first argument. If the search fails, FINDNTH returns 0.

FINDNTH(string, substring, occurrence)

```
FINDNTH("George Peck","e",3)
```

returns 9.

ISDATE

Tests the supplied string argument and returns True if it can be converted to a date value with the DATE function. Otherwise, returns False.

ISDATE(string)

```
IF ISDATE([DateOrderedString]) THEN
       DATE([DateOrderedString])
ELSE
       DATE("01/01/1900")
END
```

will return the string database field DateOrderedString as a date value if it contains a string value that the DATE function can properly interpret. Otherwise, it will return the date value January 1, 1900.

Simply using

```
DATE([DateOrderedString])
```

will return a null if the supplied string argument does not contain a properly formatted date. By using ISDATE, null values may be avoided.

LEFT

Returns a string value consisting of the number of characters specified in the second argument from the beginning (or left) of the string specified in the first argument.

LEFT(string, num_chars)

```
IF LEFT([Type-Part#], 3) = "BLU" THEN
       "BluRay Media"
ELSE
       "Other Media"
END
```

will test the first three characters of the Type-Part# database field and return “BluRay Media” if those characters = “BLU.” If the first three characters are anything else, the result will be “Other Media.”

LEN

Returns a numeric value indicating the number of characters contained in the supplied string argument.

LEN(string)

```
IF LEN([Phone]) = 10 THEN
       LEFT([Phone], 3)
ELSE
       "No Area Code"
END
```

returns the first three characters from the Phone database field if the Phone database field contains 10 characters. Otherwise, returns "No Area Code."

LOWER

Converts the supplied string to lowercase letters.

LOWER(string)

```
LOWER("GeOrgE")
```

returns "george."

LTRIM

Trims away leading spaces (any spaces on the left side) from the supplied string.

LTRIM(string)

```
LTRIM(" George ")
```

returns "George " (the trailing space remains).

MAX

Returns either the last alphabetic string value of a supplied database field within the underlying set of data records if used with one argument or the last alphabetic value of two supplied values if used with two arguments.

MAX(string)
MAX(string1, string2)

```
MAX([CustomerLastName])
```

returns "Williams" if the highest alphabetical value in the CustomerLastName database field within the underlying set of records is Williams.

```
MAX([City1], [City2])
```

returns "Yuba City" if the City1 database field contains Denver and the City2 database field contains Yuba City.

MID

Returns a subset of the characters from the string argument, starting at the numeric position supplied by the numeric start argument. If the optional-length numeric argument is supplied, only the number of characters specified in length is returned.

MID(string, start, *[length]*)

```
MID("Tableau Software", 4)
```

returns "leau Software."

```
MID("Tableau Software", 6, 5)
```

returns "au So."

MIN

Returns either the first alphabetic string value of a supplied database field within the underlying set of data records if used with one argument or the first alphabetic value of two supplied values if used with two arguments.

MIN(expression)
MIN(expression1, expression2)

```
MIN([CustomerLastName])
```

returns "Butler" if the lowest alphabetical value in the CustomerLastName database field within the underlying set of records is Butler.

```
MIN([City1], [City2])
```

returns "Denver" if the City1 database field contains Denver and the City2 database field contains Yuba City.

REGEXP_EXTRACT

Using the "regular expression" pattern specified in the second argument, searches the string specified in the first argument and returns the substring matching the pattern. Regular expression patterns vary by data source. For Tableau Data Extracts, the pattern conforms to ICU (International Components for Unicode) standards and must be a constant.

REGEXP_EXTRACT(string, pattern)

```
REGEXP_EXTRACT("Tableau 9: The Official Guide","[a-z]+\s+(\d+)")
```

returns 9.

Note *This function is only available with certain data sources. If your data source doesn't support the function, it won't appear in the functions list within the calculation editor. If you still want to use the function, create a Tableau Data Extract.*

REGEXP_EXTRACT_NTH

Using the "regular expression" pattern specified in the second argument, searches the string specified in the first argument and returns the substring matching the pattern. The *n*th "capturing group" is searched, with the value of *n* specified by the third argument. If the third argument is zero, the entire string is searched. Regular expression patterns vary by data source. For Tableau Data Extracts, the pattern conforms to ICU (International Components for Unicode) standards and must be a constant.

REGEXP_EXTRACT_NTH(string, pattern, index)

```
REGEXP_EXTRACT_NTH("Tableau 9: The Official Guide","[a-z]+\s+(\d+)",0)
```

returns "ableau 9".

Note *This function is only available with certain data sources. If your data source doesn't support the function, it won't appear in the functions list within the calculation editor. If you still want to use the function, create a Tableau Data Extract.*

REGEXP_MATCH

Using the "regular expression" pattern specified in the second argument, evaluates the string specified in the first argument and returns True if they match. Regular expression patterns vary by data source. For Tableau Data Extracts, the pattern conforms to ICU (International Components for Unicode) standards and must be a constant.

REGEXP_MATCH(string, pattern)

```
REGEXP_MATCH("Tableau 9: The Official Guide","[a-z]+\s+(\d+)")
```

returns a Boolean true value.

Note *This function is only available with certain data sources. If your data source doesn't support the function, it won't appear in the functions list within the calculation editor. If you still want to use the function, create a Tableau Data Extract.*

REGEXP_REPLACE

Using the "regular expression" pattern specified in the second argument, evaluates the string specified in the first argument and returns a replacement string based on the third argument.

Regular expression patterns vary by data source. For Tableau Data Extracts, the pattern conforms to ICU (International Components for Unicode) standards and must be a constant.

REGEXP_REPLACE(string, pattern, replacement)

```
REGEXP_REPLACE("Tableau 8: The Official Guide","[a-z]+\s+(\d+)","9")
```

returns "T9: The Official Guide".

Note *This function is only available with certain data sources. If your data source doesn't support the function, it won't appear in the functions list within the calculation editor. If you still want to use the function, create a Tableau Data Extract.*

REPLACE

Searches the first string for the second string. If found, it replaces it with the third string. If the second string isn't found, the original string is returned unchanged.

REPLACE(string1, string2, string3)

```
REPLACE("George is rich","rich","moderately well off")
```

returns "George is moderately well off."

RIGHT

Returns a string value consisting of the number of characters specified in the second argument from the end (or right) of the string specified in the first argument.

RIGHT(string, num_chars)

```
RIGHT([Type-Part#], 5)
```

will return "81773" for a Type-Part# database value of BLU-81773.

RTRIM

Trims away trailing spaces (any spaces on the right side) from the supplied string.

RTRIM(string)

```
LTRIM(" George ")
```

returns " George" (the leading space remains).

SPACE

Returns a string consisting of the number of space characters specified in the numeric argument.

SPACE(number)

```
"Tableau" + SPACE(5) + "Software"
```

will return "Tableau Software" (with five spaces appearing between the two words).

SPLIT

Extracts a substring from the source string (the first argument), based on a consistent separator character (delimiter) contained in the second argument, which splits the first string into "chunks." The third numeric argument indicates which "chunk" of the split string to return. If the third argument is positive, SPLIT searches from left to right to extract the chunk. If the third argument is negative, SPLIT searches from right to left. If the third argument exceeds the number of chunks, SPLIT returns an empty string.

SPLIT(String, DelimiterString, NumberOfChunk)

```
SPLIT("Evergreen,CO,United States", "," ,2)
```

returns "CO."

```
SPLIT("Evergreen,CO,United States", "," ,-3)
```

returns "Evergreen".

Note *Some data sources don't permit negative chunk numbers, whereas some have a limit (typically 10) on the number of split functions that can be used within a data source.*

STARTSWITH

Determines whether the value in the first string argument starts with the value in the second string argument. If so, the function returns True. Otherwise, it returns False.

STARTSWITH(string, substring)

```
IF STARTSWITH([Type-Part#], "BLU") THEN
       "BluRay Media"
ELSE
       "Other Media"
END
```

will determine if the Type-Part# database field starts with the characters BLU. If so, "BluRay Media" will be returned. Otherwise, the result will be "Other Media."

TRIM

Trims trailing spaces from the beginning and ending of the supplied string.

TRIM(string)

```
TRIM(" George ")
```

returns "George."

UPPER

Converts the supplied string to uppercase letters.

UPPER(string)

```
UPPER("GeOrgE")
```

returns "GEORGE."

Date Functions

These functions perform date operations and return various results (either dates or numbers). Depending on the function, they accept a combination of date, string, or numeric arguments.

Many date functions expect a date-part argument, which is a specific string value that must be supplied:

- **"year"**
- **"quarter"**
- **"month"**
- **"day of year"**
- **"day"**
- **"weekday"**
- **"week"**
- **"hour"**
- **"minute"**
- **"second"**

DATEADD

Returns a date/time before or after the date supplied in the third argument, based on the number of "intervals" supplied in the second argument. The type of interval (days, weeks, and so forth) is determined by the string date-part supplied in the first argument.

DATEADD(date_part, interval, date)

```
DATEADD("month", 6, [Order Date-Time])
```

returns 4/1/2013 2:15:00 PM if the Order Date-Time database field contains October 1, 2012, 2:15 p.m.

DATEDIFF

Returns a numeric value indicating the number of "intervals" between the start_date argument and the end_date argument. The type of interval (days, weeks, and so forth) is determined by the string date-part supplied in the first argument.

DATEDIFF(date_part, start_date, end_date)

```
DATEDIFF("days", [Order Date], [Ship Date])
```

returns the numeric value 8 if the Order Date database field is 2/15/2013 and the Ship Date database field is 2/23/2013.

DATENAME

Returns a string value representing the spelled-out date_part supplied in the first argument of the date supplied in the second argument.

DATENAME(date_part, date)

```
DATENAME("month", [Order Date])
```

will return the string "March" if the Order Date database field is 3/10/2013.

DATEPARSE

Converts a string value (the second argument) to a true date/time value, based on a string "format mask" (the first argument), which is a series of characters that represent specific date and time parts. Characters that don't represent particular date or time representations must be placed in the format string alongside the format mask characters. If the format of the string to convert doesn't match the format mask, DATEPARSE returns Null.

DATEPARSE(formatstring, stringtoconvert)

Although occasionally dependent upon the data source and although not complete, more commonly used format string characters are

Mask Characters	Representation
MMMM	completely spelled month name (for example, "August")
MMM	three-character month name abbreviation (for example, "Aug")
MM	two-character month number (for example, "08")
M	one-character month number (for example, "8")
dd	two-character day of month
d	one-character day of month
y or yyyy	four-character year number
yy	two-character year number
h	hour number in 12-hour time
hh	hour number in 24-hour time
mm	two-character minute number
m	one-character minute number
ss	two-character second number
s	one-character second number

```
DATEPARSE("yyyy-MM-dd hh:mm:ss", "2015-08-05 16:43:20")
```

returns 8/5/2015 4:43:20 PM as a true date/time data type.

Note *DATEPARSE is only available for certain data sources. If your data source doesn't support the function, it won't appear in the functions list in the calculation editor. If you still want to use DATEPARSE, export your data source to a Tableau Data Extract, which does support DATEPARSE.*

DATEPART

Returns a numeric value indicating the date_part supplied in the first argument of the date supplied in the second argument.

DATEPART(date_part, date)

```
DATEPART("hour", [Sample Date-Time])
```

will return the numeric value 15 if the Sample Date-Time database field contains 3/10/2013 03:25:10 p.m.

DATETRUNC

Truncates, or rounds, the date supplied in the second argument to the date_part supplied in the first argument, returning a date value.

DATETRUNC(date_part, date)

```
DATETRUNC("quarter", [Order Date])
```

returns 1/1/2013 12:00:00 AM if the Order Date database field contains 3/10/2013 (the first date of the calendar quarter that 3/10/2103 falls in is 1/1/2013).

DAY

Returns a numeric value between 1 and 31 indicating the day of the month of the supplied date argument.

DAY(date)

```
DAY([Order Date])
```

returns a numeric 10 if the Order Date database field contains 3/10/2013.

ISDATE

Tests the supplied string argument and returns True if it can be converted to a date value with the DATE function. Otherwise, returns False.

ISDATE(string)

```
IF ISDATE([DateOrderedString]) THEN
       DATE([DateOrderedString])
ELSE
       DATE("01/01/1900")
END
```

will return the string database field DateOrderedString as a date value if it contains a string value that the DATE function can properly interpret. Otherwise, it will return the date value January 1, 1900.

Simply using

```
DATE([DateOrderedString])
```

will return a null if the supplied string argument does not contain a properly formatted date. By using ISDATE, null values may be avoided.

MAX

Returns either the latest date value of a supplied database field within the underlying set of data records if used with one argument or the latest date value of two supplied values if used with two arguments.

MAX(date)
MAX(date1, date2)

```
MAX([Order Date])
```

returns 12/15/2012, if the latest Order Date in the underlying set of records is 12/15/2012.

```
MAX([Paid Date], [Invoice Due Date])
```

returns 5/10/2013 if the Paid Date database field contains 5/5/2013 and the Invoice Due Date database field contains 5/10/2013.

MIN

Returns either the earliest date value of a supplied database field within the underlying set of data records if used with one argument or the earliest date value of two supplied values if used with two arguments.

MIN(date)
MIN(date1, date2)

```
MIN([Order Date])
```

returns 1/22/2012, if the earliest Order Date in the underlying set of records is 1/22/2012.

```
MIN([Paid Date], [Invoice Due Date])
```

returns 5/5/2013 if the Paid Date database field contains 5/5/2013 and the Invoice Due Date database field contains 5/10/2013.

MONTH

Returns a numeric value between 1 and 12 indicating the month of the supplied date argument.

MONTH(date)

```
MONTH([Order Date])
```

returns a numeric 3 if the Order Date database field contains 3/10/2013.

NOW

Returns a date/time value consisting of the current date and time as returned by the database. Although NOW doesn't require any arguments, parentheses are still required.

NOW()

```
NOW()
```

returns 5/30/2013 3:44:41 PM if executed on May 30, 2013, at 3:44:41 p.m.

TODAY

Returns a date value consisting of the current date as returned by the database. Although TODAY doesn't require any arguments, parentheses are still required.

TODAY()

```
DATEDIFF("day", [Due Date], TODAY())
```

returns a numeric value 33 if the Due Date database field is 4/27/2013 and the database server's current date is 5/30/2013.

Caution *NOW and TODAY retrieve values from the underlying database server, not your local Tableau client computer. If the database server you are using is set to a different time zone, you may not see the results you expect.*

YEAR

Returns a numeric value indicating the year of the supplied date argument.

YEAR(date)

```
YEAR([Order Date])
```

returns a numeric 2,013 if the Order Date database field contains 3/10/2013.

Type Conversion Functions

These functions convert from one data type to another.

DATE

Converts the supplied expression to a date value. The supplied expression may be a number (representing the number of days since 1/1/1900), string, or date/time value.

DATE(expression)

```
IF ISDATE([DateOrderedString]) THEN
       DATE([DateOrderedString])
ELSE
       DATE("01/01/1900")
END
```

will return the string database field DateOrderedString as a date value if it contains a string value that the DATE function can properly interpret. Otherwise, it will return the date value January 1, 1900.

Simply using

```
DATE([DateOrderedString])
```

will return a null if the supplied string argument does not contain a properly formatted date. By using ISDATE, null values may be avoided.

DATETIME

Converts the supplied expression to a date/time value. The supplied expression may be an integer or real number (representing the number of fractional days since 1/1/1900), string, or date/time value.

DATETIME(expression)

```
DATETIME([OrderDateTimeString])
```

returns the date/time value 2/10/2013 3:42:00 PM if the string value of the OrderDateTime String database field contains the string "2013-02-10 15:42:00."

FLOAT

Converts the supplied string or number to a floating-point (noninteger) number.

FLOAT(expression)

```
FLOAT([Discount String])
```

will return the floating-point numeric value .08500 if the Discount String database field contains the string ".085."

INT

Converts the supplied string or number to an integer (whole) number. Any decimal values appearing in the original argument are ignored (truncated)—the result will not be rounded up.

INT(expression)

```
INT([Unit Price])
```

returns the whole number 7 if the floating-point Unit Price database field contains 7.75.

MAKEDATE

Converts three numbers (year as first argument, month as second argument, and day as third argument) into a date value.

MAKEDATE(yearnumber, monthnumber, daynumber)

```
MAKEDATE(2015, 8, 5)
```

returns 8/5/2015 as a true date value.

Note *MAKEDATE is only available with Tableau Data Extracts and a limited set of other data sources.*

MAKEDATETIME

Converts separate date (the first argument) and time (the second argument) values into a combined date/time value. The first argument expression can be either a date value or a date/time value (in which case, only the date is extracted). The second argument must be a date/time value, from which only the time is extracted.

MAKEDATETIME(expression, time)

```
MAKEDATETIME(DATE("8/5/2015"), MAKETIME(17,12,15))
```

returns 8/5/2015 5:12:15 PM as a date/time value.

Note *MAKEDATETIME is only available with Tableau Data Extracts and a limited set of other data sources.*

MAKETIME

Converts three numbers (hour as first argument, minute as second argument, and second as third argument) into a time value. Generally, this function is used when another function requires a time value to be supplied. Because Tableau doesn't support a time-only value, MAKETIME will return the converted time on 12/30/1899 if used in Tableau directly.

MAKETIME(hour, minute, second)

```
MAKETIME(17, 12, 15)
```

returns 12/30/1899 5:12:15 PM as a date/time value.

```
MAKEDATETIME(DATE("8/5/2015"), MAKETIME(17,12,15))
```

returns 8/5/2015 5:12:15 PM as a date/time value.

Note *MAKETIME is only available with Tableau Data Extracts and a limited set of other data sources.*

STR

Converts any value to a string.

STR(expression)

```
STR([Order DateTime])
```

returns the string value "2013-04-10 14:15:00" if the Order DateTime database field contains a date/time value of 4/10/2013 2:15 PM.

Logical Functions

These functions perform various logical tests, returning certain values resulting from the tests. In some cases, comparison operators are used with these functions. Comparison operators recognized by Tableau include

- = *or* ==
- <
- >
- <=
- >=
- <> *or* !=
- **AND**
- **OR**

CASE / WHEN / ELSE / END

CASE defines an initial expression that is evaluated by one or more following WHEN tests. When a WHEN test is True, the corresponding THEN value is returned. An optional ELSE clause will return a value if all WHEN tests fail. If there is no ELSE clause and all WHEN tests fail, a null is returned.

CASE <expression> WHEN <value1> THEN <return1> . . . [ELSE <else>] END

```
CASE LEFT([ProductType-SKU], 3)
       WHEN "HDW" THEN "Hardware"
       WHEN "SFT" THEN "Software"
       WHEN "ACC" THEN "Accessory"
END
```

will return "Software" for a ProductType-SKU value of "SFT-18385."

```
CASE LEFT([ProductType-SKU], 3)
       WHEN "HDW" THEN "Hardware"
       WHEN "SFT" THEN "Software"
       WHEN "ACC" THEN "Accessory"
END
```

will return null for a ProductType-SKU value of "APP-15432."

```
CASE LEFT([ProductType-SKU], 3)
       WHEN "HDW" THEN "Hardware"
       WHEN "SFT" THEN "Software"
       WHEN "ACC" THEN "Accessory"
       ELSE "Uncategorized"
END
```

will return "Uncategorized" for a ProductType-SKU value of "APP-15432."

IF / ELSE / ELSEIF / THEN / END

Performs the test specified in the first expression. If the test evaluates to True, returns the value specified in the THEN clause. Optionally, additional ELSEIF/THEN tests may be executed to perform more tests if the first test failed. An optional ELSE clause may be added to return a result if all IF and ELSEIF tests fail.

IF <expression> THEN <then> [ELSEIF <expression2> THEN <then2>...] [ELSE <else>] END

```
IF LEFT([ProductType-SKU], 3) = "HDW" THEN
       "Hardware"
ELSEIF LEFT([ProductType-SKU], 3) = "SFT" THEN
       "Software"
ELSEIF LEFT([ProductType-SKU], 3) = "ACC" THEN
       "Accessory"
ELSEIF [ProductPrice] > 1000 THEN
       "Premium Category"
ELSE
       "Uncategorized"
END
```

will return "Premium Category" for the combination of a ProductType-SKU value of "APP-15432" and ProductPrice of 2,500.

Note *Although the CASE construct (discussed prior to IF) provides similar logic that may be easier to read and maintain, it does not provide the ability to include more than one test or more than one data type in a test, as does IF.*

IFNULL

Tests whether the first expression evaluates to a null value. If so, the second expression is returned. If not, the first expression is returned.

IFNULL(expression1, expression2)

```
IFNULL([Unit Price] * [Discount], [Unit Price])
```

returns the calculated results of Unit Price multiplied by Discount if neither of the fields contains null. Otherwise, Unit Price is returned.

IIF

Performs the logic test specified in the first argument (the first argument may also be a Boolean field). If the test evaluates to True, returns the value supplied by the second Then argument. If the test evaluates to False, returns the values specified in the third Else argument. If the optional fourth Unknown argument is provided, it will be returned if the test evaluates to Unknown (perhaps because the test returns a null value).

IIF(test, then, else, *[unknown]*)

```
IIF([Discount] < 0, [Unit Price], [Unit Price] * [Discount], [Unit Price])
```

returns the value of Unit Price if the Discount is less than 0. Otherwise, returns the calculated results of Unit Price multiplied by Discount. If Discount is null, returns Unit Price.

ISDATE

Tests the supplied string argument and returns True if it can be converted to a date value with the DATE function. Otherwise, returns False.

ISDATE(string)

```
IF ISDATE([DateOrderedString]) THEN
       DATE([DateOrderedString])
ELSE
       DATE("01/01/1900")
END
```

will return the string database field DateOrderedString as a date value if it contains a string value that the DATE function can properly interpret. Otherwise, it will return the date value January 1, 1900.

Simply using

```
DATE([DateOrderedString])
```

will return a null if the supplied string argument does not contain a properly formatted date. By using ISDATE, null values may be avoided.

ISNULL

Returns True if the supplied expression evaluates to a null value. Otherwise, returns False.

ISNULL(expression)

```
IF ISNULL([Discount]) THEN
        [Unit Price]
ELSE
        [Unit Price] * [Discount]
END
```

returns the Unit Price if the Discount field is null. Otherwise, the calculated results of Unit Price multiplied by Discount are returned.

ZN

Returns the numeric value of the supplied number if it is not null. Otherwise, returns 0.

ZN(expression)

```
ZN([Salary])
```

returns 125,000 if the Salary database field contains the value 125,000. If the Salary database field contains a null value, the result will be 0 instead of a null.

Aggregate Functions

These functions perform in-calculation aggregation. Although Tableau typically aggregates by default, these functions provide for more precise aggregation, if desired, within a calculated field.

Examples within this section (with the exception of EXCLUDE, FIXED, and INCLUDE) are based on the sample data set illustrated in Figure A-1.

RowID	Product	Amount
1	SFT	24.50
2	SFT	85.45
3	HRD	1150.00
4	ACC	245.10
5	HRD	750.00
6	ACC	Null
7	SFT	249.00
8	ACC	9.25
9	SFT	750.00
10	ACC	12.50

Figure A-1 Sample data for aggregate functions

ATTR

Returns the value of the supplied expression if all rows in the group of data contain the same value. Otherwise, returns an asterisk (*).

ATTR(expression)

```
ATTR([Product])
```

returns * if there is no Product filter applied or Product is not added to a shelf. Otherwise, the value of Product is returned.

AVG

Returns the numeric average of the supplied numeric value.

AVG(number)

```
AVG([Amount])
```

returns 363.977777778. Note that a null value does not figure into the average calculation.

COUNT

Returns the numeric count of records based on the supplied expression (any data type is allowed).

COUNT(expression)

```
COUNT([RowID])
```

returns 10.

```
COUNT([Amount])
```

returns 9, as null values do not increment the COUNT function.

COUNTD

Returns the numeric count of distinct (or unique) records based on the supplied expression (any date type is allowed).

COUNTD(expression)

```
COUNTD([RowID])
```

returns 10, as RowID is unique.

```
COUNTD([Amount])
```

returns 8, as there are two occurrences of the same value, as well as a null value (which does not increment COUNTD).

```
COUNTD([Product])
```

returns 3, as there are three unique occurrences of a product type.

Note *COUNTD is not available with a limited set of legacy data sources or Microsoft Access.*

EXCLUDE

Calculates an aggregate expression in a separate data source query, *excluding* any existing "level of detail" dimensions used on the chart. More than one dimension may be excluded by separating them with commas.

{EXCLUDE dimension *[, additionaldimensions…]* : AggregateExpression}

```
{EXCLUDE [Category] : AVG([Sales])}
```

will calculate average sales by any dimension or dimensions currently in use on the chart, excluding the category before the average is calculated.

FIXED

Calculates an aggregate expression in a separate data source query, *ignoring* any existing "level of detail" dimensions used on the chart. More than one dimension may be used by separating them with commas. An additional option is referred to as a *table scoped* expression, whereby no dimensions are specified after the FIXED keyword. An alternative table scoped expression eliminates the FIXED keyword and merely includes an aggregate expression surrounded by curly braces.

{FIXED dimension *[, additionaldimensions…]* : AggregateExpression}
{FIXED : AggregateExpression} or **{Aggregate Expression}**

```
{FIXED [Customer Name] : YEAR(MIN([Order Date]))}
```

LOD : Level of detail
Avg(Sales) --> Sum (Sales) / Count of rows
Region vs avg(Sales) --> sum of sales in that region/ count of rows per region/ avg(Include(customer name):sum sales) --> sum of sales in that region / count of distinct customers --> customer level detail added

INCLUDE

Calculates an aggregate expression in a separate data source query *in addition to* any existing "level of detail" dimensions used on the chart. More than one dimension may be included by separating them with commas.

{INCLUDE dimension *[, additionaldimensions…]* : AggregateExpression}

```
{INCLUDE [Product Name] : MAX([Quantity])}
```

will calculate the highest quantity per product name within other dimensions used on the chart.

MAX

Returns the largest numeric value, latest date, or last alphabetical value based on the supplied expression (any data type is allowed).

MAX(expression)

```
MAX([Product])
```

returns the string "SFT," as that is the last alphabetical value in the Product field.

```
MAX([Amount])
```

returns the number 1,150, the largest Amount value.

MEDIAN

Returns the numeric median of the supplied numeric value.

MEDIAN(number)

```
MEDIAN([Amount])
```

returns 245.1. Note that the null amount does not figure into the median calculation.

Note *MEDIAN is not available with a limited set of legacy data sources or Microsoft Access.*

MIN

Returns the smallest numeric value, earliest date, or first alphabetical value based on the supplied expression (any data type is allowed).

MIN(expression)

```
MIN([Product])
```

returns the string "ACC," as that is the first alphabetical value in the Product field.

```
MIN([Amount])
```

returns the number 9.25, the smallest Amount value.

PERCENTILE

Returns the value representing the percentile specified in the second fractional numeric argument (which must be between 0 and 1) of the expression specified in the first argument.

PERCENTILE(expression, fractionalnumber)

```
PERCENTILE([Amount], .40)
```

returns 3,275.80.

Note *PERCENTILE is not available with all data sources. If the function name does not appear in the function list within the calculation editor, extract your data to a Tableau Data Extract, which does support PERCENTILE.*

STDEV

Returns the standard deviation of the supplied numeric value.

STDEV(number)

```
STDEV([Amount])
```

returns 416.143059920.

STDEVP

Returns the population standard deviation of the supplied numeric value.

STDEVP(number)

```
STDEVP([Amount])
```

returns 392.343439484.

SUM

Returns the sum of the supplied numeric value.

SUM(expression)

```
SUM([Amount])
```

returns 3,275.8.

VAR

Returns the variance of the supplied numeric value.

VAR(number)

```
VAR([Amount])
```

returns 173,175.046319444.

VARP

Returns the population variance of the supplied numeric value.

VARP(number)

```
VARP([Amount])
```

returns 153,933.374506173.

Pass-through Functions

Although many functions in the Tableau formula language may look familiar to those who work with SQL databases, not every function or feature of an individual database server is provided in Tableau. Should you desire to call a native SQL function from your particular database server, Tableau provides a series of *pass-through functions.* These functions are not interpreted or modified by Tableau before being sent to the database server (hence, the name RAWSQL). As such, any native function available in your underlying SQL database may be used in a calculated field.

Pass-through functions are available for each data type that Tableau recognizes (real, integer, string, and others). There are also nonaggregate and aggregate versions of each pass-through function. Nonaggregate versions will be automatically aggregated by Tableau, as with standard database fields. Aggregate versions, which generally will be used when passing aggregate functions to the underlying database, will be not be reaggregated by Tableau.

You may pass as many arguments as necessary to the underlying SQL function. However, because the database may not be able to interpret Tableau's field-naming convention, you must add argument "placeholders" in the RAWSQL specified as the first argument. These placeholders are represented by a percent sign (%) followed by sequential numbers (the first argument will be %1, the fifth %5, and so on). You must then supply the actual Tableau field names to pass to the placeholders as additional arguments in the RAWSQL syntax.

For example, if you want to return a random number from an underlying Microsoft SQL Server database, the following calculated field, using the SQL Server RAND function, may be used. Note that the RAND function makes use of an optional "seed" value. This is being supplied by a unique order ID number from Tableau.

```
RAWSQL_REAL("RAND(%1)",[Order ID])
```

If you uncheck Aggregate Measures from the Analysis drop-down menu or add the unique Order ID to your visualization, Tableau will display the underlying random numbers from SQL Server. However, as with any other standard numeric database value, the random number calculated field will be aggregated as a SUM otherwise.

If you want the pass-through function to make use of a value aggregated on the database server, use the AGG versions of the pass-through function. For example:

```
RAWSQLAGG_REAL("FLOOR(SUM(%1))",[Order Amount])
```

will subtotal the passed Order Amount field on the database server and return the results of the SQL Server native FLOOR function. Much as with standard calculated fields that use

Tableau aggregate functions, calculated fields using AGG versions of pass-through functions will not be reaggregated by Tableau and will appear with an AGG indicator when used on the workspace.

RAWSQLAGG_BOOL

Returns a Boolean (True or False) result based on the supplied SQL expression. The result will not be reaggregated by Tableau.

RAWSQLAGG_BOOL("sql_expression", *[argument1], . . . [argumentN]*)

RAWSQLAGG_DATE

Returns a date result based on the supplied SQL expression. The result will not be reaggregated by Tableau.

RAWSQLAGG_DATE("sql_expression", *[argument1], . . . [argumentN]*)

RAWSQLAGG_DATETIME

Returns a date/time result based on the supplied SQL expression. The result will not be reaggregated by Tableau.

RAWSQLAGG_DATETIME("sql_expression", *[argument1], . . . [argumentN]*)

RAWSQLAGG_INT

Returns an integer (whole number) result based on the supplied SQL expression. The result will not be reaggregated by Tableau.

RAWSQLAGG_INT("sql_expression", *[argument1], . . . [argumentN]*)

RAWSQLAGG_REAL

Returns a real (fractional) result based on the supplied SQL expression. The result will not be reaggregated by Tableau.

RAWSQLAGG_REAL("sql_expression", *[argument1], . . . [argumentN]*)

RAWSQLAGG_STR

Returns a string result based on the supplied SQL expression. The result will not be reaggregated by Tableau.

RAWSQLAGG_STR("sql_expression", *[argument1], . . . [argumentN]*)

RAWSQL_BOOL

Returns a Boolean (True or False) result based on the supplied SQL expression.

RAWSQL_BOOL("sql_expression", *[argument1], . . . [argumentN]*)

RAWSQL_DATE

Returns a date result based on the supplied SQL expression.

RAWSQL_DATE("sql_expression", *[argument1], . . . [argumentN]*)

RAWSQL_DATETIME

Returns a date/time result based on the supplied SQL expression.

RAWSQL_DATETIME("sql_expression", *[argument1], . . . [argumentN]*)

RAWSQL_INT

Returns an integer (whole number) result based on the supplied SQL expression.

RAWSQL_INT("sql_expression", *[argument1], . . . [argumentN]*)

RAWSQL_REAL

Returns a real (fractional) result based on the supplied SQL expression.

RAWSQL_REAL("sql_expression", *[argument1], . . . [argumentN]*)

RAWSQL_STR

Returns a string result based on the supplied SQL expression.

RAWSQL_STR("sql_expression", *[argument1], . . . [argumentN]*)

User Functions

These functions retrieve user and group information from a connected Tableau Server. By using these functions, you may tailor your worksheet or dashboard behavior according to who is logged in to Tableau Server (see Chapter 9).

FULLNAME

Returns the full name of the current Tableau user. If logged in to Tableau Server, the Tableau Server full name is returned. Otherwise, the domain/Windows user is returned. Even though this function accepts no arguments, the parentheses are required.

FULLNAME()

```
FULLNAME()
```

returns “George Peck” if the full name (not the user name) of the user currently logged in to Tableau Server is George Peck.

ISFULLNAME

Returns True if the supplied string argument is the same as the current FULLNAME (see the "FULLNAME" entry previously in this section). Otherwise, returns False.

ISFULLNAME(string)

```
ISFULLNAME("George Peck")
```

returns True if the user currently logged in to Tableau Server has a full name (not a user name) of "George Peck."

ISMEMBEROF

Returns True if the supplied string argument is the same as a Tableau Server group that the current user is a member of. Otherwise, returns False.

ISMEMBEROF(string)

```
IF ISMEMBEROF("HR") THEN
    STR([Salary])
ELSE
    "Not Available"
END
```

returns the string conversion of the Salary database field if the current user is a member of the HR group. Otherwise, returns "Not Available."

ISUSERNAME

Returns True if the supplied string argument is the same as the current USERNAME (see the "USERNAME" entry later in this section). Otherwise, returns False.

ISUSERNAME(string)

```
ISUSERNAME("GPeck")
```

returns True if the user currently logged in to Tableau Server has a user name (not a full name) of "GPeck."

USERDOMAIN

Returns the domain of the Tableau Server the current user is logged in to. If the user is not logged in to Tableau Server, the current Windows domain is returned. Even though this function accepts no arguments, the parentheses are required.

USERDOMAIN()

```
USERDOMAIN()
```

returns "local" if the Tableau Server the user is currently logged in to is not in an Active Directory domain.

USERNAME

Returns the user name (not the full name) of the user currently logged in to Tableau Server. If the user is not logged in to Tableau Server, the Windows user name is returned. Even though this function accepts no arguments, the parentheses are required.

USERNAME()

```
USERNAME()
```

returns "GPeck" if the user name (not the full name) of the user currently logged in to Tableau Server is GPeck.

```
USERNAME() = "Administrator" OR [Employee Login] = USERNAME()
```

when added to the Filters shelf, returns all records if the administrator is logged in. Otherwise, it only returns records where the Employee Login database field matches the user name (not the full name) of the user currently logged in to Tableau Server.

Table Calculation Functions

These functions perform table calculation manipulation (see Chapter 6 for detailed information on table calculations). These functions force the calculated field they're in to evaluate after Tableau has returned an aggregated set of data from the underlying data source. As such, any field arguments provided must appear in aggregate (by way of SUM(), AVG(), or other aggregate functions described earlier in this appendix).

Any calculated field using these functions will automatically become a table calculation. As such, they will appear with a delta icon when used on the workspace. And, as with standard table calculations, *direction* and *scope* may be specified for the calculated field by way of the Compute Using option from the field's context menu (direction and scope are covered in detail in Chapter 6). The direction/scope choices will determine the "partition" the function applies to.

The examples in this section are based on the text table illustrated in Figure A-2. For these examples, direction/scope has been set to Pane Down, indicating that table calculations will calculate from top to bottom, resetting at a new "pane," or year. As such, this example contains two partitions. Due to the Pane Down partition setting, only the first Accessories row is illustrated.

		Accessories	Hardware	Software
2014	October	350	550	250
	November	60	1,225	300
	December	1,025	660	300
2015	January	650	1,325	1,175
	February	300	260	200
	March	500	260	250

Figure A-2 Table calculation function example

FIRST

Returns the offset from the current position to the starting position in the partition. Even though this function accepts no arguments, the parentheses are required.

FIRST()

```
FIRST()
```

returns

Amount	Calc
350	0
60	-1
1,025	-2
650	0
300	-1
500	-2

INDEX

Returns the number of the current position in the partition. Even though this function accepts no arguments, the parentheses are required.

INDEX()

```
INDEX()
```

returns

Amount	Calc
350	1
60	2
1,025	3
650	1
300	2
500	3

LAST

Returns the offset from the current position to the ending position in the partition. Even though this function accepts no arguments, the parentheses are required.

LAST()

```
LAST()
```

returns

Amount	Calc
350	2
60	1
1,025	0
650	2
300	1
500	0

LOOKUP

Looks up another value from earlier in the partition (if the numeric offset argument is negative) or later in the partition (if the numeric offset is positive). The offset determines how many positions before or after the current position to retrieve. If offset is not supplied, the offset may be supplied with the "Relative To" option on the table calculation context menu.

LOOKUP(expression, *[offset]*)

```
LOOKUP(SUM([Amount]), -1)
```

returns

Amount	Calc
350	
60	350
1,025	60
650	
300	650
500	300

PREVIOUS_VALUE

Returns the value of the previous occurrence of the calculated field. If this is the first occurrence of the calculated field, the expression argument is returned. Otherwise, this function builds a cumulative value as it progresses through the data partition.

PREVIOUS_VALUE(expression)

```
PREVIOUS_VALUE(0) + SUM([Amount])
```

returns

Amount	Calc
350	350
60	410
1,025	1,435
650	650
300	950
500	1,450

RANK

Returns the "standard competition" rank for each value. If the values are tied, the same rank is returned and the next rank normally in sequence is skipped. By default, items are ranked in descending order. An optional second argument will rank in ascending order.

RANK(expression *['asc'/'desc']*)

```
RANK(SUM([Amount]))
```

returns

Amount	Calc
250	3
300	1
300	1
1,175	1
200	3
250	2

Note *To better illustrate RANK, the preceding illustration shows the third column (Software) from Figure A-2.*

RANK_DENSE

Returns the "dense" rank for each value. If the values are tied, the same rank is returned and the next rank is returned sequentially without any ranks being skipped. By default, items are ranked in descending order. An optional second argument will rank in ascending order.

RANK_DENSE(expression *['asc'/'desc']*)

```
RANK_DENSE(SUM([Amount]))
```

returns

Amount	Calc
250	2
300	1
300	1
1,175	1
200	3
250	2

Note *To better illustrate RANK_DENSE, the preceding illustration shows the third column (Software) from Figure A-2.*

RANK_MODIFIED

Returns the "modified competition" rank for each value. If the values are tied, the same rank is returned; however, the initial rank that the first value would have achieved is skipped. By default, items are ranked in descending order. An optional second argument will rank in ascending order.

RANK_MODIFIED(expression *[,'asc'/'desc']*)

```
RANK_MODIFIED(SUM([Amount]))
```

returns

Amount	Calc
250	3
300	2
300	2
1,175	1
200	3
250	2

Note *To better illustrate RANK_MODIFIED, the preceding illustration shows the third column (Software) from Figure A-2.*

RANK_PERCENTILE

Returns the percentile rank for each value. By default, items are ranked in descending order. An optional second argument will rank in ascending order.

RANK_PERCENTILE(expression *[,'asc'/'desc']*)

```
RANK_PERCENTILE(SUM([Amount]))
```

returns

Amount	Calc
250	0
300	1
300	1
1,175	1
200	0
250	1

Note *To better illustrate RANK_PERCENTILE, the preceding illustration shows the third column (Software) from Figure A-2.*

RANK_UNIQUE

Returns the unique rank for each value. If the values are tied, a sequential rank is still assigned. By default, items are ranked in descending order. An optional second argument will rank in ascending order.

RANK_UNIQUE(expression *[,'asc'/'desc']*)

```
RANK_UNIQUE(SUM([Amount]))
```

returns

Amount	Calc
250	3
300	1
300	2
1,175	1
200	3
250	2

Note *To better illustrate RANK_UNIQUE, the preceding illustration shows the third column (Software) from Figure A-2.*

RUNNING_AVG

Returns the running average of the supplied expression as of the current position in the partition.

RUNNING_AVG(expression)

```
RUNNING_AVG(SUM([Amount]))
```

returns

Amount	Calc
350	350
60	205
1,025	478.333333333
650	650
300	475
500	483.333333333

RUNNING_COUNT

Returns the running count of the supplied expression as of the current position in the partition.

RUNNING_COUNT(expression)

```
RUNNING_COUNT(SUM([Amount]))
```

returns

Amount	Calc
350	1
60	2
1,025	3
650	1
300	2
500	3

RUNNING_MAX

Returns the running maximum of the supplied expression as of the current position in the partition.

RUNNING_MAX(expression)

```
RUNNING_MAX(SUM([Amount]))
```

returns

Amount	Calc
350	350
60	350
1,025	1,025
650	650
300	650
500	650

RUNNING_MIN

Returns the running minimum of the supplied expression as of the current position in the partition.

RUNNING_MIN(expression)

```
RUNNING_MIN(SUM([Amount]))
```

returns

Amount	Calc
350	350
60	60
1,025	60
650	650
300	300
500	300

RUNNING_SUM

Returns the running sum of the supplied expression as of the current position in the partition.

RUNNING_SUM(expression)

```
RUNNING_SUM(SUM([Amount]))
```

returns

Amount	Calc
350	350
60	410
1,025	1,435
650	650
300	950
500	1,450

SIZE

Returns the number of positions in the partition. Even though this function accepts no arguments, the parentheses are required.

SIZE()

```
SIZE()
```

returns

Amount	Calc
350	3
60	3
1,025	3
650	3
300	3
500	3

TOTAL

Returns the total sum of the supplied expression over the partition.

TOTAL(expression)

```
TOTAL(SUM([Amount]))
```

returns

Amount	Calc
350	1,435
60	1,435
1,025	1,435
650	1,450
300	1,450
500	1,450

WINDOW_AVG

Returns the overall average of the supplied expression over the partition (referred to as the "window"). If optional numeric start and end arguments are supplied, the average is reduced to that range.

WINDOW_AVG(expression *[,start, end]*)

```
WINDOW_AVG(SUM([Amount]))
```

returns

Amount	Calc
350	478.333333333
60	478.333333333
1,025	478.333333333
650	483.333333333
300	483.333333333
500	483.333333333

```
WINDOW_AVG(SUM([Amount]),0,LAST()-1)
```

returns

Amount	Calc
350	205
60	60
1,025	
650	475
300	300
500	

WINDOW_COUNT

Returns the overall count of the supplied expression over the partition (referred to as the "window"). If optional numeric start and end arguments are supplied, the count is reduced to that range.

WINDOW_COUNT(expression *[,start, end]*)

```
WINDOW_COUNT(SUM([Amount]))
```

returns

Amount	Calc
350	3
60	3
1,025	3
650	3
300	3
500	3

```
WINDOW_COUNT(SUM([Amount]),0,LAST()-1)
```

returns

Amount	Calc
350	2
60	1
1,025	0
650	2
300	1
500	0

WINDOW_MAX

Returns the overall maximum of the supplied expression over the partition (referred to as the "window"). If optional numeric start and end arguments are supplied, the maximum is reduced to that range.

WINDOW_MAX(expression *[,start, end]*)

```
WINDOW_MAX(SUM([Amount]))
```

returns

Amount	Calc
350	1,025
60	1,025
1,025	1,025
650	650
300	650
500	650

```
WINDOW_MAX(SUM([Amount]),0,LAST()-1)
```

returns

Amount	Calc
350	350
60	60
1,025	
650	650
300	300
500	

WINDOW_MEDIAN

Returns the overall median of the supplied expression over the partition (referred to as the "window"). If optional numeric start and end arguments are supplied, the median is reduced to that range.

WINDOW_MEDIAN(expression *[,start, end]*)

```
WINDOW_MEDIAN(SUM([Amount]))
```

returns

Amount	Calc
350	350
60	350
1,025	350
650	500
300	500
500	500

```
WINDOW_MEDIAN(SUM([Amount]),0,LAST()-1)
```

returns

Amount	Calc
350	205
60	60
1,025	
650	475
300	300
500	

WINDOW_MIN

Returns the overall minimum of the supplied expression over the partition (referred to as the "window"). If optional numeric start and end arguments are supplied, the minimum is reduced to that range.

WINDOW_MIN(expression *[,start, end]*)

```
WINDOW_MIN(SUM([Amount]))
```

returns

Amount	Calc
350	60
60	60
1,025	60
650	300
300	300
500	300

```
WINDOW_MIN(SUM([Amount]),0,LAST()-1)
```

returns

Amount	Calc
350	60
60	60
1,025	
650	300
300	300
500	

WINDOW_PERCENTILE

Returns the overall percentile, specified in the fractional second argument, of the supplied expression over the partition (referred to as the "window"). If optional numeric start and end arguments are supplied, the percentile is reduced to that range.

WINDOW_PERCENTILE(expression, percentile *[,start, end]*)

```
WINDOW_PERCENTILE(SUM([Amount]), .4)
```

returns

Amount	Calc
350	292
60	292
1,025	292
650	460
300	460
500	460

WINDOW_STDEV

Returns the overall standard deviation of the supplied expression over the partition (referred to as the "window"). If optional numeric start and end arguments are supplied, the standard deviation is reduced to that range.

WINDOW_STDEV(expression *[,start, end]*)

```
WINDOW_STDEV(SUM([Amount]))
```

returns

Amount	Calc
350	495.134661818
60	495.134661818
1,025	495.134661818
650	175.594229214
300	175.594229214
500	175.594229214

```
WINDOW_STDEV(SUM([Amount]),0,LAST()-1)
```

returns

Amount	Calc
350	205.060966544
60	
1,025	
650	247.487373415
300	
500	

WINDOW_STDEVP

Returns the overall population standard deviation of the supplied expression over the partition (referred to as the "window"). If optional numeric start and end arguments are supplied, the population standard deviation is reduced to that range.

WINDOW_STDEVP(expression *[,start, end]*)

```
WINDOW_STDEVP(SUM([Amount]))
```

returns

Amount	Calc
350	404.275758473
60	404.275758473
1,025	404.275758473
650	143.372087784
300	143.372087784
500	143.372087784

```
WINDOW_STDEVP(SUM([Amount]),0,LAST()-1)
```

returns

Amount	Calc
350	145
60	0
1,025	
650	175
300	0
500	

WINDOW_SUM

Returns the overall sum of the supplied expression over the partition (referred to as the "window"). If optional numeric start and end arguments are supplied, the sum is reduced to that range.

WINDOW_SUM(expression *[,start, end]*)

```
WINDOW_SUM(SUM([Amount]))
```

returns

Amount	Calc
350	1,435
60	1,435
1,025	1,435
650	1,450
300	1,450
500	1,450

```
WINDOW_SUM(SUM([Amount]),0,LAST()-1)
```

returns

Amount	Calc
350	410
60	60
1,025	
650	950
300	300
500	

WINDOW_VAR

Returns the overall variance of the supplied expression over the partition (referred to as the "window"). If optional numeric start and end arguments are supplied, the variance is reduced to that range.

WINDOW_VAR(expression *[,start, end]*)

```
WINDOW_VAR(SUM([Amount]))
```

returns

Amount	Calc
350	245,158.33333
60	245,158.33333
1,025	245,158.33333
650	30,833.33333
300	30,833.33333
500	30,833.33333

```
WINDOW_VAR(SUM([Amount]),0,LAST()-1)
```

returns

Amount	Calc
350	42,050
60	
1,025	
650	61,250
300	
500	

WINDOW_VARP

Returns the overall population variance of the supplied expression over the partition (referred to as the "window"). If optional numeric start and end arguments are supplied, the population variance is reduced to that range.

WINDOW_VARP(expression *[,start, end]*)

```
WINDOW_VARP(SUM([Amount]))
```

returns

Amount	Calc
350	163,439
60	163,439
1,025	163,439
650	20,556
300	20,556
500	20,556

```
WINDOW_VARP(SUM([Amount]),0,LAST()-1)
```

returns

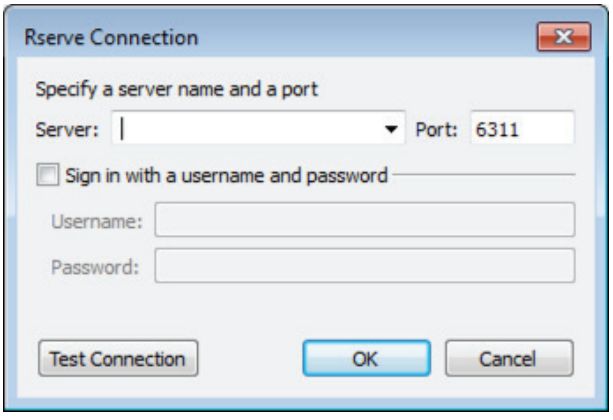
Amount	Calc
350	21,025
60	0
1,025	
650	30,625
300	0
500	

Functions to Call External R Logic

Tableau supports an external connection to an RServe instance. RServe is an open standard for exposing R functions to other systems. R is an open-source programming language often used for more advanced statistical analysis and calculations.

To connect Tableau to an RServe instance, choose Help | Settings And Performance | Manage R Connection from the drop-down menus. Specify the IP address or server name and, optionally, nonstandard port number of the desired RServe instance. If security has

been configured on the server, check the appropriate box and specify the user ID and password.



R function calls all use the same basic syntax:

```
SCRIPT_<DATA_TYPE>("<RFunction(.arg1 [,.argn…])", TableauAggregate1 [, TableauAggregateN…])
```

First, you need to determine the type of data the R function you are calling returns. Tableau expects four types of return values: Boolean (true/false), integer (whole number), real (floating point/decimals), and string (text). Your calculated field should use the corresponding SCRIPT function.

Next, determine how many arguments the R function requires. For example, if the R function requires three arguments, the first two being decimal numbers and the third being a string, you'll need to specify (within quotation marks) the actual name of the R function, followed by an open parenthesis, three argument "placeholders" .arg1, .arg2, and .arg3, and a closing parenthesis. These placeholders indicate where the actual values you pass from Tableau will be placed. Follow the R function with a comma, followed by the Tableau aggregated measures and dimensions you want to pass to R, separated by commas. In the example just described, you might pass two numbers and one string, such as SUM([Profit]), AVG([Shipping Cost]), and MAX([Country/Region]).

For example, if an R function called "correlation" has been created to assign a correlation between two values, with the result being passed back to Tableau as a floating-point value, and you want to analyze the correlation between total sales and total profit, you would use the following:

```
SCRIPT_REAL("correlation(.arg1, .arg2)", SUM([Sales]), SUM([Profit]))
```

SCRIPT_BOOL

SCRIPT_BOOL("<RFunction(.arg1 *[,.argn…]*)", TableauAggregate1 *[, TableauAggregateN…]*)

SCRIPT_INT

SCRIPT_INT("<RFunction(.arg1 *[,.argn…]*)", TableauAggregate1 *[, TableauAggregateN…]*)

SCRIPT_REAL

SCRIPT_REAL("<RFunction(.arg1 *[,.argn…]*)", TableauAggregate1 *[, TableauAggregateN…]*)

SCRIPT_STR

SCRIPT_STR("<RFunction(.arg1 *[,.argn…]*)", TableauAggregate1 *[, TableauAggregateN…]*)

Note *R SCRIPT functions are processed at the same time as Tableau Table Calculations after the workbook data source has been queried and data source results already returned to Tableau. As such, R SCRIPT functions may only be passed aggregated dimensions or measures, such as those using SUM() and ATTR() functions.*

Data Source–Specific Functions

These functions are only available with either a limited set of data sources or a single specific data source. If you are using a supported data source, the function will appear in the available function list in the calculated field editor.

CEILING

Returns either an integer value of the argument if it contains no decimal places or the next highest integer value of the argument if it contains decimal places. Available with Tableau Data Extract data sources, Google BigQuery, and Hadoop Hive data sources.

CEILING(number)

```
CEILING(25.675)
```

returns the integer value 26.

FLOOR

Returns either an integer value of the argument if it contains no decimal places or the next lowest integer value of the argument if it contains decimal places. Available with Tableau Data Extract data sources, Google BigQuery, and Hadoop Hive data sources.

FLOOR(number)

```
FLOOR(25.675)
```

returns the integer value 25.

GET_JSON_OBJECT

Returns a string representing the embedded JSON object within the JSON string specified in the first argument based on the JSON path specified in the second argument. This function is only available with the Hadoop Hive data source.

GET_JSON_OBJECT(JSON string, JSON path)

Given a data field named HiveJSONString that consists of the following JSON construct:

```
{
    "Foo": "XYZ",
    "Bar": "20150801100000",
    "Quux": {
        "QuuxId": 1234,
        "QuuxName": "George"
    }
}
```

The following:

```
GET_JSON_OBJECT([HiveJSONString], "$.Quux.QuuxName")
```

returns "George".

PARSE_URL

Returns a substring from the URL specified in the first argument based on the url_part. Valid url_part string specified by the second argument, which can consist of 'HOST', 'PATH', 'QUERY', 'REF', 'PROTOCOL', 'AUTHORITY', 'FILE', and 'USERINFO'. This function is only available with the Hadoop Hive data source.

PARSE_URL(string, url_part)

```
PARSE_URL('http://www.AblazeGroup.com', 'HOST')
```

returns "www.AblazeGroup.com".

PARSE_URL_QUERY

Evaluates the query string portion of the URL specified with the first argument, returning the position of the specified query parameter key specified in the second argument. This function is only available with the Hadoop Hive data source.

PARSE_URL_QUERY(string, key)

```
PARSE_URL_QUERY("http://www.AblazeGroup.com?source=google&campaign=15", "campaign")
```

returns "15".

XPATH_BOOLEAN

Returns true if the XPath expression supplied by the second argument matches a node or evaluates to true when compared with the first argument. This function is only available with the Hadoop Hive data source.

XPATH_BOOLEAN(XML string, XPath expression string)

```
XPATH_BOOLEAN('<values> <value id="0">6</value><value id="1">8</value>', 'values/value[@id="1"] = 8')
```

returns True.

XPATH_DOUBLE/XPATH_FLOAT

Searches the XML supplied by the first argument, returning the floating-point (decimal) value of the XPath expression supplied by the second argument. These functions are only available with the Hadoop Hive data source.

XPATH_DOUBLE(XML string, XPath expression string)
XPATH_FLOAT(XML string, XPath expression string)

```
XPATH_DOUBLE('<values><value>4.0</value><value>2.7</value> </values>', 'sum(value/*)')
```

returns = 6.7.

XPATH_INT

Searches the XML supplied by the first argument, returning the integer (nondecimal) value of the XPath expression supplied by the second argument. This function is only available with the Hadoop Hive data source.

XPATH_INT(XML string, XPath expression string)

```
XPATH_INT('<values><value>4</value><value>7</value> </values>', 'sum(value/*)')
```

returns = 11.

XPATH_LONG/XPATH_SHORT

Searches the XML supplied by the first argument, returning the numeric value of the XPath expression supplied by the second argument. If the value cannot be evaluated to a number, zero is returned. These functions are only available with the Hadoop Hive data source.

XPATH_LONG(XML string, XPath expression string)
XPATH_SHORT(XML string, XPath expression string)

```
XPATH_LONG('<values><value>1</value><value>George</value> </values>','sum(value/*)')
```

returns 0.

XPATH_STRING

Searches the XML text specified in the first argument for the "node" specified by the second argument, returning the text of the node. This function is only available with the Hadoop Hive data source.

XPATH_STRING(XML string, XPath expression string)

```
XPATH_STRING('<sites ><url domain="org">http://www.kuvo.org</url> <url
domain="com">http://www.AblazeGroup.com</url></sites>', 'sites/url[@
domain="org"]')
```

returns
"http://www.kuvo.org".

DOMAIN

Searches the URL supplied, returning the domain. This function is only available with the Google BigQuery data source.

DOMAIN(string_url)

```
DOMAIN('http://www.AblazeGroup.com:8080/default.asp')
```

returns "AblazeGroup.com".

GROUP_CONCAT

Combines, or "concatenates," values from multiple underlying records into a single comma-delimited string. This function is only available with the Google BigQuery data source.

GROUP_CONCAT(expression)

```
GROUP_CONCAT(ProductType)
```

returns "Software,Hardware,Accessory".

HOST

Searches the URL supplied, returning the hostname. This function is only available with the Google BigQuery data source.

HOST(string_url)

```
DOMAIN('http://localhost:8080/default.html')
```

returns "localhost:8080".

LOG2

Returns the logarithm base 2 of a number. This function is only available with the Google BigQuery data source.

LOG2(number)

```
LOG2(64)
```

returns 6.00.

LTRIM_THIS

Left trims the first argument with any leading occurrence of the second argument removed. This function is only available with the Google BigQuery data source.

LTRIM_THIS(string, substring)

```
LTRIM_THIS('*** George !!!','*** ')
```

returns "George !!!".

RTRIM_THIS

Right trims the first argument with any trailing occurrence of the second argument removed. This function is only available with the Google BigQuery data source.

RTRIM_THIS(string, substring)

```
RTRIM_THIS('*** George !!!',' !!!')
```

returns "*** George".

TIMESTAMP_TO_USEC

Converts a TIMESTAMP data type to a UNIX timestamp in microseconds. This function is only available with the Google BigQuery data source.

TIMESTAMP_TO_USEC(expression)

USEC_TO_TIMESTAMP

Converts a UNIX timestamp in microseconds to a TIMESTAMP data type. This function is only available with the Google BigQuery data source.

USEC_TO_TIMESTAMP(expression)

TLD

Searches the supplied URL, returns the top-level domain (plus any country domain) in the URL. This function is only available with the Google BigQuery data source.

TLD(string_url)

```
TLD('http://www.yahoo.co.tw:80/index.html')
```

returns "co.tw".