Evaluation functions

http://docs.splunk.com/Documentation/Splunk/latest/SearchReference/CommonEvalFunctions

Commands

You can use these functions with the `eval`, `fieldformat`, and `where` commands, and as part of evaluation expressions.

Usage

- All functions that accept strings can accept literal strings or any field.
- All functions that accept numbers can accept literal numbers or any numeric field.

Comparison and Conditional functions

Function	Description	Example(s)
`case(X,"Y",...)`	This function takes pairs of arguments X and Y. The X arguments are Boolean expressions that will be evaluated from first to last. When the first X expression is encountered that evaluates to TRUE, the corresponding Y argument will be returned. The function defaults to NULL if none are true.	This example returns descriptions for the corresponding http status code: `... | eval description=case(error == 404, "Not found", error == 500, "Internal Server Error", error == 200, "OK")`
`cidrmatch("X",Y)`	This function returns true, when IP address Y belongs to a particular subnet X. The function uses two string arguments: the first is the CIDR	This example uses cidrmatch to set a field, `isLocal`, to "local" if the field `ip` matches the subnet, or "not local" if it does not: `... | eval isLocal=if(cidrmatch("123.132.32.0/25",ip),`

	subnet; the second is the IP address to match.	`"local", "not local")`
		
		This example uses cidrmatch as a filter:
		
		`... | where cidrmatch("123.132.32.0/25", ip)`

`coalesce(X,...)`	This function takes an arbitrary number of arguments and returns the first value that is not null.	Let's say you have a set of events where the IP address is extracted to either `clientip` or `ipaddress`. This example defines a new field called `ip`, that takes the value of either `clientip` or `ipaddress`, depending on which is not NULL (exists in that event): `... | eval ip=coalesce(clientip,ipaddress)`
`if(X,Y,Z)`	This function takes three arguments. The first argument X must be a Boolean expression. If X evaluates to TRUE, the result is the second argument Y. If, X evaluates to FALSE, the result evaluates to the third argument Z.	This example looks at the values of error and returns err=OK if error=200, otherwise returns err=Error: `... | eval err=if(error == 200, "OK", "Error")`

`like(TEXT, PATTERN)`	This function takes two arguments, a string to match TEXT and a match expression string PATTERN. It returns TRUE if and only if the first argument is like the SQLite pattern in Y. The pattern language supports exact text match, as well as % characters for wildcards and _ characters for a single character match.	This example returns islike=TRUE if the field value starts with foo:
		`... | eval is_a_foo=if(like(field, "foo%"),`
		`"yes a foo", "not a foo")`
		
		or
		
		`... | where like(field, "foo%")`

`match(SUBJECT, "REGEX")`	This function compares the regex string REGEX to the value of SUBJECT	This example returns true IF AND ONLY IF field matches the basic pattern of an IP address. Note that the example uses ^ and $ to perform a full match. `... | eval n=if(match(field,`

	and returns a Boolean value. It returns true if the REGEX can find a match against any substring of SUBJECT.	`"^\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3}$"), 1, 0)`
`null()`	This function takes no arguments and returns NULL. The evaluation engine uses NULL to represent "no value"; setting a field to NULL clears its value.	
`nullif(X,Y)`	This function is used to compare fields. The function takes two arguments, X and Y, and returns NULL if X = Y. Otherwise it returns X.	`... | eval n=nullif(fieldA,fieldB)`
`searchmatch(X)`	This function takes one argument X, which is a search string. The function returns true IF AND ONLY IF the event matches the search string.	`... | eval n=searchmatch("foo AND bar")`
`validate(X,Y,...)`	This function takes pairs of arguments, Boolean expressions X and strings Y. The function returns the string Y corresponding to the first expression X that evaluates to False and defaults to NULL if all are True.	This example runs a simple check for valid ports: `... | eval n=validate(isint(port), "ERROR: Port is not an integer", port >= 1 AND port <= 65535, "ERROR: Port is out of range")`

Conversion functions

Function	Description	Examples

`tonumber(NUMSTR,BASE)`
 	
`tonumber(NUMSTR)`
	This function converts the input string NUMSTR to a number. NUMSTR can be a field name or a value. BASE is optional and used to define the base of the number to convert to. BASE can be 2 to 36, and defaults to 10. If the `tonumber` function cannot parse a field value to a number, for example if the value contains a leading and trailing space, the function returns NULL. Use the `trim` function to remove leading or trailing spaces. If the `tonumber` function cannot parse a literal string to a number, it returns an error.	This example returns the string values in the field store_sales:
		`... | eval n=tonumber(store_sales)`
		
		This example returns "164":
		`... | eval n=tonumber("0A4",16)`

`tostring(X,Y)`	This function converts the input value to a string. If the input value is a number, it reformats it as a string. If the input value is a Boolean value, it returns the corresponding string value, "True" or "False".
	
	This function requires at least one argument X; if X is a number, the second argument Y is optional and can be `"hex"` `"commas"` or `"duration"`:
	
	`tostring(X,"hex")` converts X to hexadecimal.
	
	`tostring(X,"commas")` formats X with commas and, if the number includes decimals,
		This example returns "True 0xF 12,345.68":
		`... | eval n=tostring(1==1) + " " + tostring(15, "hex") + " " + tostring(12345.6789, "commas")`
		
		This example returns `foo=615` and `foo2=00:10:15`:
		`... | eval foo=615 | eval foo2 = tostring(foo, "duration")`
		
		This example formats the column totalSales to display values with a currency symbol and commas. You must use a period between the currency value and the `tostring` function.
		`...| fieldformat totalSales="$".tostring(totalSales,"commas")`

	rounds to nearest two decimal places. `tostring(X,"duration")` converts seconds X to readable time format HH:MM:SS.	**Note:** When used with the `eval` command, the values might not sort as expected because the values are converted to ASCII. Use the `fieldformat` command with the `tostring` function to format the displayed values. The underlying values are not changed with the `fieldformat` command.

Cryptographic functions

Function	Description	Example(s)
`md5(X)`	This function computes and returns the MD5 hash of a string value X.	`... | eval n=md5(field)`
`sha1(X)`	This function computes and returns the secure hash of a string value X based on the FIPS compliant SHA-1 hash function.	`... | eval n=sha1(field)`
`sha256(X)`	This function computes and returns the secure hash of a string value X based on the FIPS compliant SHA-256 hash function.	`... | eval n=sha256(field)`
`sha512(X)`	This function computes and returns the secure hash of a string value X based on the FIPS compliant SHA-512 hash function.	`... | eval n=sha512(field)`

Date and Time functions

Function	Description	Example(s)

`now()`	This function takes no arguments and returns the time that the search was started. The time is represented in Unix time or in seconds since Epoch time.	
`relative_time(X,Y)`	This function takes an epochtime time, X, as the first argument and a relative time specifier, Y, as the second argument and returns the epochtime value of Y applied to X.	`... | eval n=relative_time(now(), "-1d@d")`
`strftime(X,Y)`	This function takes an epochtime value, X, as the first argument and renders it as a string using the format specified by Y. For a list and descriptions of format options, refer to the topic "Common time format variables".	This example returns the hour and minute from the _time field: `... | eval n=strftime(_time, "%H:%M")`
`strptime(X,Y)`	This function takes a time represented by a string, X, and parses it into a timestamp using the format specified by Y. For a list and descriptions of format options, refer to the topic "Common time format variables".	If timeStr is in the form, "11:59", this returns it as a timestamp: `... | eval n=strptime(timeStr, "%H:%M")`
`time()`	This function returns the wall-clock time with microsecond resolution. The value of time() will be different for each event based on when that event was processed by the `eval` command.	

Informational functions

Function	Description	Example(s)

`isbool(X)`	This function takes one argument X and returns TRUE if X is Boolean.	`... | eval n=if(isbool(field),"yes","no")`
		or
		`... | where isbool(field)`

`isint(X)`	This function takes one argument X and returns TRUE if X is an integer.	`... | eval n=if(isint(field), "int", "not int")`
		or
		`... | where isint(field)`

`isnotnull(X)`	This function takes one argument X and returns TRUE if X is not NULL. This is a useful check for whether or not a field (X) contains a value.	`... | eval n=if(isnotnull(field),"yes","no")`
		or
		`... | where isnotnull(field)`

`isnull(X)`	This function takes one argument X and returns TRUE if X is NULL.	`... | eval n=if(isnull(field),"yes","no")`
		or
		`... | where isnull(field)`

`isnum(X)`	This function takes one argument X and returns TRUE if X is a number.	`... | eval n=if(isnum(field),"yes","no")`
		or
		`... | where isnum(field)`

`isstr(X)`	This function takes one argument X and returns TRUE if X is a string.	`... | eval n=if(isstr(field),"yes","no")`
		or
		`... | where isstr(field)`

`typeof(X)`	This function takes one argument and returns a string representation of its type.	This example returns "NumberStringBoolInvalid": `... | eval n=typeof(12) + typeof("string") + typeof(1==2) + typeof(badfield)`

Mathematical functions

Function	Description	Examples

`abs(X)`	This function takes a number X and returns its absolute value.	This example returns the absnum, whose values are the absolute values of the numeric field `number`: `... | eval absnum=abs(number)`
`ceil(X)`, `ceiling(X)`	This function rounds a number X up to the next highest integer.	This example returns n=2: `... | eval n=ceil(1.9)`
`exact(X)`	This function renders the result of a numeric eval calculation with a larger amount of precision in the formatted output.	`... | eval n=exact(3.14 * num)`
`exp(X)`	This function takes a number X and returns the exponential function e^x.	The following example returns $y=e^3$: `... | eval y=exp(3)`
`floor(X)`	This function rounds a number X down to the nearest whole integer.	This example returns 1: `... | eval n=floor(1.9)`
`ln(X)`	This function takes a number X and returns its natural log.	This example returns the natural log of the values of bytes: `... | eval lnBytes=ln(bytes)`
`log(X,Y)` `log(X)`	This function takes either one or two numeric arguments and returns the log of the first argument X using the second argument Y as the base. If the second argument Y is omitted, this function evaluates the log of number X with base 10.	`... | eval num=log(number,2)`
`pi()`	This function takes no arguments and returns the constant pi to 11 digits of precision.	`... | eval area_circle=pi()*pow(radius,2)`
`pow(X,Y)`	This function takes two numeric arguments X and Y and returns X^Y.	`... | eval area_circle=pi()*pow(radius,2)`
`round(X,Y)`	This function takes one or two numeric arguments X and Y, returning X rounded to the amount of decimal places specified by Y. The default is to round to an integer.	This example returns n=4: `... | eval n=round(3.5)`

		This example returns n=2.56: `... | eval n=round(2.555, 2)`
`sigfig(X)`	This function takes one argument X, a number, and rounds that number to the appropriate number of significant figures.	`1.00*1111 = 1111`, but `... | eval n=sigfig(1.00*1111)` returns n=1110.
`sqrt(X)`	This function takes one numeric argument X and returns its square root.	This example returns 3: `... | eval n=sqrt(9)`

Multivalue functions

Function	Description	Example(s)
`commands(X)`	This function takes a search string, or field that contains a search string, X and returns a multivalued field containing a list of the commands used in X. (This is generally not recommended for use except for analysis of audit.log events.)	`... | eval x=commands("search foo | stats count | sort count")` returns a multivalued field X, that contains 'search', 'stats', and 'sort'.
`mvappend(X,...)`	This function takes an arbitrary number of arguments and returns a multivalue result of all the values. The arguments can be strings, multivalue fields or single value fields.	`... | eval fullName=mvappend(initial_values, "middle value", last_values)`
`mvcount(MVFIELD)`	This function takes a field MVFIELD. The function returns the number of values if it is a multivalue, 1 if it is a single value field, and NULL otherwise.	`... | eval n=mvcount(multifield)`
`mvdedup(X)`	This function takes a multivalue field X and returns a multivalue field with its duplicate values removed.	`... | eval s=mvdedup(mvfield)`

`mvfilter(X)`	This function filters a multivalue field based on an arbitrary Boolean expression X. The Boolean expression X can reference ONLY ONE field at a time. **Note:**This function will return NULL values of the field `x` as well. If you don't want the NULL values, use the expression: `mvfilter(x!=NULL)`.	This example returns all of the values in field email that end in .net or .org: `... | eval n=mvfilter(match(email, "\.net$") OR match(email, "\.org$"))`
`mvfind(MVFIELD,"REGEX")`	This function tries to find a value in multivalue field X that matches the regular expression REGEX. If a match exists, the index of the first matching value is returned (beginning with zero). If no values match, NULL is returned.	`... | eval n=mvfind(mymvfield, "err\d+")`
`mvindex(MVFIELD,STARTINDEX, ENDINDEX)` `mvindex(MVFIELD,STARTINDEX)`	This function takes two or three arguments, field MVFIELD and numbers STARTINDEX and ENDINDEX, and returns a subset of the multivalue field using the indexes provided. For `mvindex(mvfield, startindex, [endindex])`, endindex is inclusive and optional. Both startindex and endindex can be negative, where -1 is the last element. If endindex is not specified, it returns only the value at startindex. If the indexes are out of range or invalid, the result is NULL.	Since indexes start at zero, this example returns the third value in "multifield", if it exists: `... | eval n=mvindex(multifield, 2)`
`mvjoin(MVFIELD,STR)`	This function takes two	This example joins together the

	arguments, multivalue field MVFIELD and string delimiter STR. The function concatenates the individual values of MVFIELD with copies of STR in between as separators.	individual values of "foo" using a semicolon as the delimiter: `... | eval n=mvjoin(foo, ";")`
`mvrange(X,Y,Z)`	This function creates a multivalue field for a range of numbers. This function can contain up to three arguments: a starting number X, an ending number Y (exclusive), and an optional step increment Z. If the increment is a timespan such as '7'd, the starting and ending numbers are treated as epoch times.	This example returns a multivalue field with the values 1, 3, 5, 7, 9. `... | eval mv=mvrange(1,11,2)`
`mvsort(X)`	This function uses a multivalue field X and returns a multivalue field with the values sorted lexicographically.	`... | eval s=mvsort(mvfield)`
`mvzip(X,Y,"Z")`	This function takes two multivalue fields, X and Y, and combines them by stitching together the first value of X with the first value of field Y, then the second with the second, and so on. The third argument, Z, is optional and is used to specify a delimiting character to join the two values. The default delimiter is a comma. This is similar to Python's zip command.	`... | eval nserver=mvzip(hosts,ports)`

Statistical functions

In addition to these functions, a comprehensive set of statistical functions is available to use with the stats, chart, and related commands.

Function	Description	Example(s)
`max(X,...)`	This function takes an arbitrary number of numeric or string arguments, and returns the max; strings are greater	This example returns either "foo" or field, depending on the value

	than numbers.	of field: `... | eval n=max(1, 3, 6, 7, "foo", field)`
`min(X,...)`	This function takes an arbitrary number of numeric or string arguments, and returns the min; strings are greater than numbers.	This example returns either 1 or field, depending on the value of field: `... | eval n=min(1, 3, 6, 7, "foo", field)`
`random()`	This function takes no arguments and returns a pseudo-random integer ranging from zero to 2^{31}-1, for example: 0…2147483647	

Text functions

Function	Description	Examples
`len(X)`	This function returns the character length of a string X.	`... | eval n=len(field)`
`lower(X)`	This function takes one string argument and returns the lowercase version. The upper() function also exists for returning the uppercase version.	This example returns the value provided by the field username in lowercase. `... | eval username=lower(username)`
`ltrim(X,Y)` `ltrim(X)`	This function takes one or two arguments X and Y and returns X with the characters in Y trimmed from the left side. If Y is not specified, spaces and tabs are removed.	This example returns x="abcZZ": `... | eval x=ltrim(" ZZZZabcZZ ", " Z")`
`replace(X,Y,Z)`	This function returns a string formed by substituting string Z for every occurrence of regex string Y in string X. The third argument Z can also reference groups that are matched in the	This example returns date with the month and day numbers switched, so if the input was 1/14/2015 the return value would be 14/1/2015: `... | eval n=replace(date,`

	regex.	`"^(\d{1,2})/(\d{1,2})/", "\2/\1/")`
`rtrim(X,Y)` `rtrim(X)`	This function takes one or two arguments X and Y and returns X with the characters in Y trimmed from the right side. If Y is not specified, spaces and tabs are removed.	This example returns n="ZZZZabc": `... | eval n=rtrim(" ZZZZabcZZ ", " Z")`

`spath(X,Y)`	This function takes two arguments: an input source field X and an spath expression Y, that is the XML or JSON formatted location path to the value that you want to extract from X. If Y is a literal string, it needs quotes, `spath(X,"Y")`. If Y is a field name (with values that are the location paths), it doesn't need quotes. This may result in a multivalued field. Read more about the `spath` search command.	This example returns the values of locDesc elements:
		`... | eval locDesc=spath(_raw, "vendorProductSet.product.desc.locDesc")`
		This example returns the hashtags from a twitter event:
		`index=twitter | eval output=spath(_raw, "entities.hashtags")`

`split(X,"Y")`	This function takes two arguments, field X and delimiting character Y. It splits the value(s) of X on the delimiter Y and returns X as a multivalue field.	`... | eval n=split(foo, ";")`
`substr(X,Y,Z)`	This function takes either two or three arguments, where X is a string and Y and Z are numeric. It returns a substring of X, starting at the index specified by Y with the number of characters specified by Z. If Z is not given, it returns the rest of the string. The indexes follow SQLite semantics; they start at 1. Negative indexes can be used to indicate a start from the end of the string.	This example concatenates "str" and "ing" together, returning "string": `... | eval n=substr("string", 1, 3) + substr("string", -3)`

`trim(X,Y)` `trim(X)`	This function takes one or two arguments X and Y and returns X with the characters in Y trimmed from both sides. If Y is not specified, spaces and tabs are removed.	This example returns "abc": `... | eval n=trim(" ZZZZabcZZ ", " Z")`
`upper(X)`	This function takes one string argument and returns the uppercase version. The lower() function also exists for returning the lowercase version.	This example returns the value provided by the field username in uppercase. `... | eval n=upper(username)`
`urldecode(X)`	This function takes one URL string argument X and returns the unescaped or decoded URL string.	This example returns "http://www.splunk.com/download?r=header": `... | eval n=urldecode("http%3A%2F%2Fwww.splunk.com%2Fdownload%3Fr%3Dheader")`

Trigonometry and Hyperbolic functions

Function	Description	Examples
`acos(X)`	This function computes the arc cosine of X, in the interval [0,pi] radians.	`... | eval n=acos(0)` `... | eval degrees=acos(0)*180/pi()`
`acosh(X)`	This function computes the arc hyperbolic cosine of X, in radians.	`... | eval n=acosh(2)`
`asin(X)`	This function computes the arc sine of X, in the interval [-pi/2,+pi/2] radians.	`... | eval n=asin(1)` `... | eval degrees=asin(1)*180/pi()`
`asinh(X)`	This function computes the arc hyperbolic sine of X, in radians.	`... | eval n=asinh(1)`
`atan(X)`	This function computes the arc tangent of X, in the interval [-pi/2,+pi/2] radians.	`... | eval n=atan(0.50)`

`atan2(Y, X)`	This function computes the arc tangent of Y, X in the interval [-pi,+pi] radians. Y is a value that represents the proportion of the y-coordinate. X is the value that represents the proportion of the x-coordinate. To compute the value, the function takes into account the sign of both arguments to determine the quadrant.	`.. | eval n=atan2(0.50, 0.75)`
`atanh(X)`	This function computes the arc hyperbolic tangent of X, in radians.	`... | eval n=atanh(0.500)`
`cos(X)`	This function computes the cosine of an angle of X radians.	`... | eval n=cos(-1)` `... | eval n=cos(pi())`
`cosh(X)`	This function computes the hyperbolic cosine of X radians.	`... | eval n=cosh(1)`
`hypot(X,Y)`	This function computes the hypotenuse of a right-angled triangle whose legs are X and Y. The function returns the square root of the sum of the squares of X and Y, as described in the Pythagorean theorem.	`... | eval n=hypot(3,4)`
`sin(X)`	This function computes the sine.	`... | eval n=sin(1)` `... | eval n=sin(90 * pi()/180)`
`sinh(X)`	This function computes the hyperbolic sine.	`... | eval n=sinh(1)`
`tan(X)`	This function computes the tangent.	`... | eval n=tan(1)`
`tanh(X)`	This function computes the hyperbolic tangent.	`... | eval n=tanh(1)`

See also

Statistical and charting functions, eval, fieldformat, where