Splunk® Cloud Services
SPL2 Search Reference current

Informational functions

Generated: 1/01/2025 11:11 pm

Informational functions

The following list contains the SPL2 functions that you can use to return information about a value.

For information about using string and numeric fields in functions, and nesting functions, see Overview of SPL2 eval functions.

cluster(<field>,<threshold>,<match>,<delims>)

This function generates a cluster label, in the form of a number, for each event based on how similar the events are to each other. The cluster label represents which cluster the event belongs to.

Usage

You can use this function with the `eval` and `where` commands, in the WHERE clause of the `from` command, and as part of evaluation expressions with other commands.

The cluster label is generated by using a clustering algorithm. The similarity of the events is determined by comparing the values in a specific field.

The following table defines the parameters you can use with the `cluster` function:

Parameter	Description
`field`	Required. The field that you want to analyze and cluster on.
`threshold`	Optional. The `threshold` parameter controls the sensitivity of the clustering. Must be a float number greater than 0.0 and less than 1.0, such as `threshold:0.5F`. The closer the threshold is to 1.0, the more similar events must be to be considered in the same cluster. The default threshold is `0.8F`.
`match`	Optional. The `match` parameter selects the method used to determine the similarity between events. There are three match methods: • `termlist` method breaks down the field into words and requires the exact same ordering of terms. • `termset` method breaks down the field into words and allows for an unordered set of terms. • `ngramset` method compares sets of trigram (3-character substrings). Using `ngramset` results in significantly slower processing on large field values and is most useful for short non-textual fields, like the `punct` field. The default method is `termlist`.
`delims`	Optional. The `delims` parameter uses a delimiter to tokenize the content of `field`, such as a comma (,) or a pipe (|). There is no default delimiter. The field value is processed as a single string.

You can use this function with the `eval` and `where` commands, in the WHERE and SELECT clauses of the `from` command, and as part of evaluation expressions with other commands.

Examples

The following example clusters the events by the values in the `_raw` field. The events are then sorted by the cluster number.

```
... | eval cluster_number = cluster(_raw) | sort - cluster_number
```

This example is similar to the previous example, but uses the `cluster` function in the SELECT clause of the `from` command:

```
from main select _raw, cluster(_raw) orderby cluster_number
```

The following example clusters the events by the values in the `_raw` field using a threshold of `0.9F` . The events are then sorted by the `cluster_label` field.

```
... | eval cluster_label = cluster(_raw, cluster_threshold:0.9F) | sort cluster_label
```

Consider this set of events:

_time	_raw
2021-07-21T00:57:58.000+00:00	{"JSESSIONID":"SD5SL6FF7ADFF53001","_raw":"12.130.60.5 - - [20/Jul/2021:17:57:58] \"POST /cart/error.do?msg=CreditDoesNotMatch&JSESSIONID=SD5SL6FF7ADFF53001 HTTP 1.1\" 200 ... }
2021-07-21T00:57:58.000+00:00	{"JSESSIONID":"SD10SL4FF2ADFF48107","_raw":"125.17.14.100 - - [20/Jul/2021:00:58:04] \"POST /cart/error.do?msg=CanNotGetCart&JSESSIONID=SD10SL4FF2ADFF48107 HTTP 1.1\" 200 ... }
2021-07-21T00:57:58.000+00:00	{"JSESSIONID":"SD6SL8FF9ADFF43007","_raw":"107.3.146.207 - - [19/Jul/2021:06:37:51] \"POST /cart/error.do?msg=CanNotGetCart&JSESSIONID=SD6SL8FF9ADFF43007 HTTP 1.1\" 200 ... }
2021-07-21T00:57:58.000+00:00	{"JSESSIONID":"SD6SL8FF9ADFF43007","_raw":"107.3.146.207 - - [18/Jul/2021:06:37:48] \"POST /cart/error.do?msg=CanNotGetCart&JSESSIONID=SD6SL8FF9ADFF43007 HTTP 1.1\" 200 ... }
2021-07-21T00:57:58.000+00:00	{"JSESSIONID":"SD1SL8FF9ADFF40501","_raw":"124.160.192.241 - - [18/Jul/2021:22:26:31] \"POST /cart/error.do?msg=CreditNotAccepted&JSESSIONID=SD1SL8FF9ADFF40501 HTTP 1.1\" 200 ... }

The results look like this:

_time	_raw	cluster_label
2021-07-21T00:57:58.000+00:00	{"JSESSIONID":"SD5SL6FF7ADFF53001","_raw":"12.130.60.5 - - [20/Jul/2021:17:57:58] \"POST /cart/error.do?msg=CreditDoesNotMatch&JSESSIONID=SD5SL6FF7ADFF53001 HTTP 1.1\" 200 ... }	1
2021-07-21T00:57:58.000+00:00	{"JSESSIONID":"SD10SL4FF2ADFF48107","_raw":"125.17.14.100 - - [20/Jul/2021:00:58:04] \"POST /cart/error.do?msg=CanNotGetCart&JSESSIONID=SD10SL4FF2ADFF48107 HTTP 1.1\" 200 ... }	1
2021-07-21T00:57:58.000+00:00	{"JSESSIONID":"SD6SL8FF9ADFF43007","_raw":"107.3.146.207 - - [19/Jul/2021:06:37:51] \"POST /cart/error.do?msg=CanNotGetCart&JSESSIONID=SD6SL8FF9ADFF43007 HTTP 1.1\" 200 ... }	2

_time	_raw	cluster_label
2021-07-21T00:57:58.000+00:00	{"JSESSIONID":"SD6SL8FF9ADFF43007","_raw":"107.3.146.207 - - [18/Jul/2021:06:37:48] \"POST /cart/error.do?msg=CanNotGetCart&JSESSIONID=SD6SL8FF9ADFF43007 HTTP 1.1\" 200 ... }	3
2021-07-21T00:57:58.000+00:00	{"JSESSIONID":"SD1SL8FF9ADFF40501","_raw":"124.160.192.241 - - [18/Jul/2021:22:26:31] \"POST /cart/error.do?msg=CreditNotAccepted&JSESSIONID=SD1SL8FF9ADFF40501 HTTP 1.1\" 200 ... }	3

getfields(<filter>)

This function returns a JSON array populated with JSON objects, where each object represents a field and its value.

The array that's returned is structured like this: `[{name:<field_name>, value: <field_value>}]`.

Usage

You can use this function with the `eval` and `where` commands, in the WHERE clause of the `from` command, and as part of evaluation expressions with other commands.

The `filter` parameter is optional. When specified, the filter populates the array with only the field names that match the filter. The filter can contain up to three wildcards. When wildcards are specified, a segment array is added to the JSON object that represents the field name segments which match each wildcard.

Examples

The following example shows the results when the `getfields` function is used on a set of columns. Consider this set of events:

code	error_type
200	
401	auth

The following search uses the `getfields` function without a filter:

```
... | eval rowData = getfields()
```

The results look like this:

code	error_type	rowData
200		[{"name":"code","value":200}]
401	auth	[{"name":"code","value":401},{"name":"error_type","value":"auth"}]

The following example shows how to use a filter with the `getfields` function. Consider this set of events which contains information about the status of various servers:

_time	status_www1_east1	status_www1_south1	status_www2_west1
7 Nov 2022 9:00 PM	200	404	200
7 Nov 2022 8:00 PM	404	200	200

The field names consist of three parts:

- The word `status`
- The name of the server, such as `www1`
- The location of the server, such as `east1`

You can use a filter with wildcards to return information only from these fields and collect the statuses of all of your servers. For example:

```
... | eval serverData = getfields('status_*_*')
```

The results look like this:

_time	serverData	status_www1_east1	status_www1_south
7 Nov 2022 9:00 PM	[]	NULL	NULL
	[{"name":"status_www1_east1","value":200,"segments":["www1","east1"]}]	200	NULL
	[{"name":"status_www1_east1","value":404,"segments":["www1","south1"]}]	NULL	404
	[{"name":"status_www1_east1","value":200,"segments":["www2","west1"]}]	NULL	NULL
7 Nov 2022 8:00 PM	[]	NULL	NULL
	[{"name":"status_www1_east1","value":404,"segments":["www1","east1"]}]	404	NULL
	[{"name":"status_www1_east1","value":200,"segments":["www1","south1"]}]	NULL	200
	[{"name":"status_www1_east1","value":200,"segments":["www2","west1"]}]	NULL	NULL

isarray(<value>)

The function returns TRUE if the value is an array.

Usage

You can use this function with the `eval` and `where` commands, in the WHERE clause of the `from` command, and as part of evaluation expressions with other commands. Because the function returns a Boolean value, it is supported differently in different product contexts:

- In searches, you can use the `isarray` function directly with the `where` command, but the `eval` command can't directly accept a Boolean value. You must specify the function inside another function, such as the `if` function, which can accept a Boolean value as input.
- In Edge Processor and Ingest Processor pipelines, the `eval` command is able to directly accept Boolean values, so you can use the `isarray` function directly with both the `where` and `eval` commands.

The `<value>` argument can be a field name or a value.

Examples

The following example returns `True` because `[1, 2, 3]` is an array.

```
... | eval result = if(isarray("[1, 2, 3]"), "True", "False")
```

The following example returns `False` because `1` is not an array.

```
... | eval result = if(isarray(1), "True", "False")
```

isbool(<value>)

This function returns TRUE if the value is Boolean.

Usage

Use this function with other functions that return Boolean data types, such as `cidrmatch` and `mvfind`.

This function cannot be used to determine if field values are "true" or "false" because field values are either string or number data types. Instead, use syntax such as `<fieldname>=true OR <fieldname>=false` to determine field values.

You can use this function with the `eval` and `where` commands, in the WHERE clause of the `from` command, and as part of evaluation expressions with other commands. Because the function returns a Boolean value, it is supported differently in different product contexts:

- In searches, you can use the `isbool` function directly with the `where` command, but the `eval` command can't directly accept a Boolean value. You must specify the function inside another function, such as the `if` function, which can accept a Boolean value as input.
- In Edge Processor and Ingest Processor pipelines, the `eval` command is able to directly accept Boolean values, so you can use the `isbool` function directly with both the `where` and `eval` commands.

Example

The following example shows how to use the `where` command to determine if the values in the `encrypted` field are Boolean values.

```
... | where isbool(encrypted)
```

isdouble(<value>)

The function returns TRUE if the value is a double value.

Usage

You can use this function with the `eval` and `where` commands, in the WHERE clause of the `from` command, and as part of evaluation expressions with other commands. Because the function returns a Boolean value, it is supported differently in different product contexts:

- In searches, you can use the `isdouble` function directly with the `where` command, but the `eval` command can't directly accept a Boolean value. You must specify the function inside another function, such as the `if` function,

which can accept a Boolean value as input.
- In Edge Processor and Ingest Processor pipelines, the `eval` command is able to directly accept Boolean values, so you can use the `isdouble` function directly with both the `where` and `eval` commands.

The `<value>` argument can be a field name or a value.

Examples

The following example returns `True` because `3.546` is a double.

```
... | eval result = if(isdouble(3.546), "True", "False")
```

The following example returns `False` because `1000000` is not a double.

```
... | eval result = if(isdouble(1000000), "True", "False")
```

isint(<value>)

This function returns TRUE if the value is an integer.

Usage

You can use this function with the `eval` and `where` commands, in the WHERE clause of the `from` command, and as part of evaluation expressions with other commands. Because the function returns a Boolean value, it is supported differently in different product contexts:

- In searches, you can use the `isint` function directly with the `where` command, but the `eval` command can't directly accept a Boolean value. You must specify the function inside another function, such as the `if` function, which can accept a Boolean value as input.
- In Edge Processor and Ingest Processor pipelines, the `eval` command is able to directly accept Boolean values, so you can use the `isint` function directly with both the `where` and `eval` commands.

The `<value>` argument can be a field name or a value.

Examples

The following example shows how to use the `isint` function with the `if` function. This example evaluates whether the value of the `product_id` field is an integer. If the value of the `product_id` field is an integer, then the `isint` function returns TRUE and adds the value `int` in the `result` field.

```
... | eval result=if(isint(product_id),"int", "not int")
```

The following example shows how to use the `isint` function with the `where` command. This example determines if the value in the `my_data` field is an integer.

```
... | where isint(my_data)
```

ismv(<value>)

The function returns TRUE if the value is a multivalue.

Usage

You can use this function with the `eval` and `where` commands, in the WHERE clause of the `from` command, and as part of evaluation expressions with other commands. Because the function returns a Boolean value, it is supported differently in different product contexts:

- In searches, you can use the `ismv` function directly with the `where` command, but the `eval` command can't directly accept a Boolean value. You must specify the function inside another function, such as the `if` function, which can accept a Boolean value as input.
- In Edge Processor and Ingest Processor pipelines, the `eval` command is able to directly accept Boolean values, so you can use the `ismv` function directly with both the `where` and `eval` commands.

The `<value>` argument can be a field name or a value.

Examples

The following example returns `True` because the value in the `number_list` field is a multivalue.

```
... | eval number_list=split("1, 2, 3", ",") | eval result=if(ismv(number_list), "True", "False")
```

The result looks like this:

_time	number_list	result
2024-12-11 00:49:31	1 2 3	True

isnotnull(<value>)

This function returns TRUE if the value is not NULL.

Usage

This function is useful for checking for whether or not a field contains a value.

You can use this function with the `eval` and `where` commands, in the WHERE clause of the `from` command, and as part of evaluation expressions with other commands. Because the function returns a Boolean value, it is supported differently in different product contexts:

- In searches, you can use the `isnotnull` function directly with the `where` command, but the `eval` command can't directly accept a Boolean value. You must specify the function inside another function, such as the `if` function, which can accept a Boolean value as input.
- In Edge Processor and Ingest Processor pipelines, the `eval` command is able to directly accept Boolean values, so you can use the `isnotnull` function directly with both the `where` and `eval` commands.

The `<value>` argument can be a field name or a value.

Examples

The following example shows how to use the `isnotnull` function with the `if` function. This example evaluates whether the `name` field contains a value. If the `name` field is not empty, then the `isnotnull` function returns TRUE and adds the value

yes in the result field.

```
... | eval result=if(isnotnull(name),"yes","no")
```

The following example shows how to use the isnotnull function with the where command. This example determines if the my_data field contains a value.

```
... | where isnotnull(my_data)
```

isnull(<value>)

This function returns TRUE if the value is NULL.

Usage

This function is useful for checking whether or not a field contains a value.

You can use this function with the eval and where commands, in the WHERE clause of the from command, and as part of evaluation expressions with other commands. Because the function returns a Boolean value, it is supported differently in different product contexts:

- In searches, you can use the isnull function directly with the where command, but the eval command can't directly accept a Boolean value. You must specify the function inside another function, such as the if function, which can accept a Boolean value as input.
- In Edge Processor and Ingest Processor pipelines, the eval command is able to directly accept Boolean values, so you can use the isnull function directly with both the where and eval commands.

The <value> argument can be a field name or a value.

Examples

The following example shows how to use the isnull function with the if function. This example evaluates whether the name field contains a value. If the name field is not empty, then the isnull function returns FALSE and adds the value no in the result field.

```
... | eval result=if(isnull(name),"yes","no")
```

The following example shows how to use the isnull function with the where command. This example determines if the my_data field contains a value.

```
... | where isnull(my_data)
```

isnum(<value>)

This function returns TRUE if the value is a number.

Usage

You can use this function with the eval and where commands, in the WHERE clause of the from command, and as part of evaluation expressions with other commands. Because the function returns a Boolean value, it is supported differently in different product contexts:

- In searches, you can use the `isnum` function directly with the `where` command, but the `eval` command can't directly accept a Boolean value. You must specify the function inside another function, such as the `if` function, which can accept a Boolean value as input.
- In Edge Processor and Ingest Processor pipelines, the `eval` command is able to directly accept Boolean values, so you can use the `isnum` function directly with both the `where` and `eval` commands.

The `<value>` argument can be a field name or a value.

Examples

The following example shows how to use the `isnum` function with the `if` function. This example evaluates whether the value of the `population` field is a number. If the `population` field is a number, then the `isnum` function returns TRUE and adds the value `yes` in the `result` field.

```
... | eval result=if(isnum(population),"yes","no")
```

The following example shows how to use the `isnum` function with the `where` command. This example determines if the value in the `my_data` field is a number.

```
... | where isnum(my_data)
```

isobject(<value>)

The function returns TRUE if a string is a valid JSON object.

Usage

You can use this function with the `eval` and `where` commands, in the WHERE clause of the `from` command, and as part of evaluation expressions with other commands. Because the function returns a Boolean value, it is supported differently in different product contexts:

- In searches, you can use the `isobject` function directly with the `where` command, but the `eval` command can't directly accept a Boolean value. You must specify the function inside another function, such as the `if` function, which can accept a Boolean value as input.
- In Edge Processor and Ingest Processor pipelines, the `eval` command is able to directly accept Boolean values, so you can use the `isobject` function directly with both the `where` and `eval` commands.

The `<value>` argument can be a field name or a value.

Examples

The following example returns `False` because the value in the `games` field isn't a valid JSON object.

```
... | eval games = "Ticket to Ride, Settlers of Catan" | eval result = if(isobject("games"), "True",
"False")
```

The following example returns `True` because the value in the `games` field is a valid JSON object.

```
... | eval games = "{\"type\": \"competitive\", \"name\": \"Ticket to Ride\"}" | eval result =
if(isobject(games), "True", "False")
```

isstr(<value>)

This function returns TRUE if the value is a string.

Usage

You can use this function with the `eval` and `where` commands, in the WHERE clause of the `from` command, and as part of evaluation expressions with other commands. Because the function returns a Boolean value, it is supported differently in different product contexts:

- In searches, you can use the `isstr` function directly with the `where` command, but the `eval` command can't directly accept a Boolean value. You must specify the function inside another function, such as the `if` function, which can accept a Boolean value as input.
- In Edge Processor and Ingest Processor pipelines, the `eval` command is able to directly accept Boolean values, so you can use the `isstr` function directly with both the `where` and `eval` commands.

The `<value>` argument can be a field name or a value.

Examples

The following example shows how to use the `isstr` function with the `if` function. This example evaluates whether the value of the `user_account` field is a string. If the `user_account` field is a string, then the `isstr` function returns TRUE and adds the value `yes` in the `result` field.

```
... | eval result=if(isstr(user_account),"yes","no")
```

The following example shows how to use the `isstr` function with the `where` command. This example determines if the value in the `my_data` field is a string.

```
... | where isstr(my_data)
```

typeof(<value>)

This function returns the data type of the value.

Usage

You can use this function with the `eval` and `where` commands, in the WHERE clause of the `from` command, and as part of evaluation expressions with other commands.

Examples

The following example 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)
```

The following example creates a single result using an empty dataset literal.

```
from [{ }]
```

For example:

_time
2019-08-23T10:03:01.000-0700

To determine the data type of the `_time` field, use the `eval` command with the `typeof` function. For example:

```
| from [{ }] | eval t=typeof(_time)
```

The results are:

_time	t
2019-08-23T10:03:01.000-0700	Number

See also

Function information

- Quick Reference for SPL2 eval functions
- Overview of SPL2 eval functions
- Naming function arguments in the *SPL2 Search Manual*

Related information

- Dataset literals in the *SPL2 Search Manual*