transformations. You can use FQL statements to:

- Apply filters that evaluate to `true` or `false` based on the contents of each Segment event. If the statement evaluates to `true`, the transformation is applied, and if it is `false` the transformation is not applied.
- Define new properties based on the result of an FQL statement.

In addition to boolean and equality operators like `and` and `>=`, FQL has built-in functions that make it more powerful such as `contains( str, substr )` and `match( str, pattern )`.

Examples

Given the following JSON object:

```
{
  "event": "Button Clicked",
  "type": "track",
  "context": {
    "library": {
      "name": "analytics.js",
      "version": "1.0"
    }
  },
  "properties": {
    "features": ["discounts", "dark-mode"]
  }
}
```

The following FQL statements will evaluate as follows:

FQL	RESULT
`event = 'Button Clicked'`	`true`
`event = 'Screen Tapped'`	`false`
`context.path.path = '/login'`	`false`
`type = 'identify' or type = 'track'`	`true`
`event = 'Button Clicked' and type = 'track'`	`true`
`match( context.library.version, '1.*' )`	`true`
`match( context.library.version, '2.*' )`	`false`
`type = 'track' and ( event = 'Click' or match( event, 'Button *' ) )`	`true`
`!contains( context.library.name, 'js' )`	`false`
`'dark-mode' in properties.features`	`true`
`'blink' in properties.features`	`false`

Field Paths

FQL statements may refer to any field in the JSON object including top-level properties like `userId` or `event` as well as nested properties like `context.library.version` or `properties.title` using dot-separated paths. For example, the following fields can be pointed to by the associated field paths:

```
{
  "type": "...",       // type
  "event": "...",      // event
  "context": {         // context
    "library": {       // context.library
      "name": "..."    // context.library.name
    },
    "page": {          // context. page
      "path": "...",   // context.page.path
    }
  }
}
```

Escaping Field Paths

If your field name has a character not in the set of `{a-z A-Z 0-9 _ -}`, you must escape it using a `\` character. For example, the nested field below can be referred to by `properties.product\ 1.price`:

```
{
  "properties": {
    "product 1": {
      "price": "19.99"
    }
  }
}
```

Operators

Boolean

OPERATOR	LEFT SIDE	RIGHT SIDE	RESULT
`and`	`bool` or `null`	`bool` or `null`	`true` if the left and right side are both `true`, `false` otherwise.
`or`	`bool` or `null`	`bool` or `null`	`true` if at least one side is `true`, `false` if either side is `false` or `null`.

Unary

OPERATOR	RIGHT SIDE	RESULT
`!`	`bool`	Negates the right-hand side.

Comparison

OPERATOR	LEFT SIDE	RIGHT SIDE	RESULT
`=`	`string`, `number`, `list`, `bool`, or `null`	`string`, `number`, `list`, `bool`, or `null`	`true` if the left and right side are the same type and are strictly equal, `false` otherwise.
`!=`	`string`, `number`, `list`, `bool`, or `null`	`string`, `number`, `list`, `bool`, or `null`	`true` if the left and right side are different types or if they are not strictly equal, `false` otherwise.
`>`	`number`	`number`	`true` if the left side is greater than the right side.
`>=`	`number`	`number`	`true` if the left side is greater than or equal to the right side.
`<`	`number`	`number`	`true` if the left side is less than the right side.
`<=`	`number`	`number`	`true` if the left side is less than or equal to the right side.
`in`	`string`, `number`, `bool`, or `null`	`list`	`true` if the left side is contained in the list of values.

Subexpressions

You can use parentheses to group subexpressions for more complex “and / or” logic as long as the subexpression evaluates to true or false:

FQL
`type = 'track' and ( event = 'Click' or match( 'Button *', event ) )`
`( type = 'track' or type = 'identify' ) and ( properties.enabled or match( traits.email, '*@company.com' ) )`

FQL
`!( type in ['track', 'identify'] )`

Functions

FUNCTION	RETURN TYPE	RESULT
`contains( s string, sub string )`	`bool`	Returns `true` if string `s` contains string `sub`.
`length( list or string )`	`number`	Returns the number of elements in a list or number of bytes (not necessarily characters) in a string. For example, `a` is 1 byte and `□` is 3 bytes long. Please note that you can't use this function with JSON as the argument. Using JSON may result in the function not working.
`lowercase( s string )`	`string`	Returns `s` with all uppercase characters replaced with their lowercase equivalent.
`uppercase( s string )`	`string`	Returns `s` with all lowercase characters replaced with their uppercase equivalent.
`snakecase( s string )`	`string`	Returns `s` with all space characters replaced by underscores. For example, `kebabcase("test string")` returns `test_string`.
`kebabcase( s string )`	`string`	Returns `s` with all space characters replaced by dashes. For example, `kebabcase("test string")` returns `test-string`.
`titlecase( s string )`	`string`	Returns `s` with all space characters replaced by dashes. For example, `titlecase("test string")` returns `Test String`.
`typeof( value )`	`string`	Returns the type of the given value: `"string"`, `"number"`, `"list"`, `"bool"`, or `"null"`.
`match( s string, pattern string )`	`bool`	Returns `true` if the glob pattern `pattern` matches `s`. See below for more details about glob matching.
`bool( list or string or number or nil )`	`bool`	Converts the value to a boolean value.
`string( list or string or number or nil )`	`string`	Converts the value to a string value.
`number( number or string )`	`number`	Converts the value to a number value.

Functions handle `null` with sensible defaults to make writing FQL more concise. For example, you can write `length( userId ) > 0` instead of `typeof( userId ) = 'string' and length( userId ) > 0`.

FUNCTION	RESULT
`contains( null, string )`	`false`
`length( null )`	`0`
`lowercase( null )`	`null`
`typeof( null )`	`"null"`
`match( null, string )`	`false`

`match( string, pattern )`

The `match( string, pattern )` function uses "glob" matching to return `true` if the given string fully matches a given pattern. Glob patterns are case sensitive. If you only need to determine if a string contains another string, you should use `contains()`.

PATTERN	SUMMARY
`*`	Matches zero or more characters.
`?`	Matches one character.
`[abc]`	Matches one character in the given list. In this case, `a`, `b`, or `c` will be matched.
`[a-z]`	Matches a range of characters. In this case, any lowercase letter will be matched.
`\x`	Matches the character `x` literally. This is useful if you need to match `*`, `?` or `]` literally. For example, `\*`.

PATTERN	RESULT	REASON
`match( 'abcd', 'a*d' )`	`true`	`*` matches zero or more characters.
`match( '', '*' )`	`true`	`*` matches zero or more characters.
`match( 'abc', 'ab' )`	`false`	The pattern must match the full string.
`match( 'abcd', 'a??d' )`	`true`	`?` matches one character only.
`match( 'abcd', '*d' )`	`true`	`*` matches one or more characters even at the beginning or end of the string.
`match( 'ab*d', 'ab\*d' )`	`true`	`\*` matches the literal character `*`.
`match( 'abCd', 'ab[cC]d' )`	`true`	`[cC]` matches either `c` or `C`.
`match( 'abcd', 'ab[a-z]d' )`	`true`	`[a-z]` matches any character between `a` and `z`.
`match( 'abcd', 'ab[A-Z]d' )`	`false`	`[A-Z]` matches any character between `A` and `Z` but `c` is not in that range because it is lowercase.

Error Handling

If your FQL statement is invalid (for example `userId = oops"`), your Segment event will not be sent on to downstream Destinations. Segment defaults to not sending the event to ensure that invalid FQL doesn't cause sensitive information like PII to be incorrectly sent to Destinations.

For this reason, Segment recommends that you use the Destination Filters "Preview" API to test your filters without impacting your production data.

This page was last modified: 04 Dec 2024

Help improve these docs!

Edit this page

Request docs change

Was this page helpful?

Yes

No

Get started with Segment

Segment is the easiest way to integrate your websites & mobile apps data to over 300 analytics and growth tools.

Your work e-mail

Request Demo

or

Create free account

Privacy

Terms

Website Data Collection Preferences