Checkbox Datatype

Overview

API Endpoints (/docs/endpoints)
Row Identifiers (/docs/row-identifiers)
RESTful Verbs (/docs/verbs)
Application Tokens (/docs/app-tokens)
Authentication (/docs/authentication)
Response Codes & Headers (/docs/response-codes)
System Fields (/docs/system-fields)
CORS & JSONP (/docs/cors-and-jsonp)

Filtering & Querying

Simple Filters (/docs/filtering)
SoQL Queries (/docs/queries/)
Paging Through Data (/docs/paging)
SoQL Function and Keyword Listing (/docs/functions/)
Data Transform Functions (/docs/transforms/)

Data Formats (/docs/formats/)

JSON (/docs/formats/json)
GeoJSON (/docs/formats/geojson)
CSV (/docs/formats/csv)
RDF-XML (/docs/formats/rdf-xml)

Datatypes (/docs/datatypes/)

Checkbox (/docs/datatypes/checkbox)
Fixed Timestamp (/docs/datatypes/fixed_timestamp)
Floating Timestamp (/docs/datatypes/floating_timestamp)
Line (/docs/datatypes/line)
Location (/docs/datatypes/location)
MultiLine (/docs/datatypes/multiline)
MultiPoint (/docs/datatypes/multipoint)
MultiPolygon (/docs/datatypes/multipolygon)
Number (/docs/datatypes/number)
Point (/docs/datatypes/point)
Polygon (/docs/datatypes/polygon)
Text (/docs/datatypes/text)
URL (/docs/datatypes/url)

Other APIs (/docs/other/)

Checkbox fields are boolean values that represent either `true` or `false`. If a value was not provided for the field, they can also be `null`. Example, in JSON (/docs/formats/json):

```
[ {
  "checkbox_column": true
} ]
```

The following operators can be used with `checkbox` fields:

Operator	Description
`!=`	`TRUE` when two checkbox booleans have the same value
`=`	`TRUE` when two checkbox booleans do not have the same value

And the following functions can be used with them:

2.1 2.0

Keyword Name	Description	Availability ❓ (/docs/endpoints)
`distinct` (/docs/functions/distinct)	Returns distinct set of records	2.1

Function Name	Description	Availability ❓ (/docs/endpoints)
`case(...)` (/docs/functions/case)	Returns different values based on the evaluation of boolean comparisons	2.1
`count(...)` (/docs/functions/count)	Returns a count of a given set of records	2.0 and 2.1

For example, in combination with an aggregation (/docs/queries/), to get the count of all of the crimes in Chicago that resulted in arrest:

▶ try it (https://data.cityofchicago.org/resource/6zsd-86xi.json?$select=count(*)&$where=arrest=true) | docs (/foundry/data.cityofchicago.org/6zsd-86xi) | copy

</> json ▾

```
⚙ https://data.cityofchicago.org/resource/6zsd-86xi.json?$select=count(*)&$where=arrest=true
(https://data.cityofchicago.org/resource/6zsd-86xi.json?$select=count(*)&$where=arrest=true)
```

Since checkbox values are already booleans, you can actually leave off the `=true` in that expression:

▶ try it (https://data.cityofchicago.org/resource/6zsd-86xi.json?$select=count(*)&$where=arrest) | docs (/foundry/data.cityofchicago.org/6zsd-86xi) | copy | </> json ▾

```
⚙ https://data.cityofchicago.org/resource/6zsd-86xi.json?$select=count(*)&$where=arrest
(https://data.cityofchicago.org/resource/6zsd-86xi.json?$select=count(*)&$where=arrest)
```

Data Transform Listing

Overview

API Endpoints (/docs/endpoints)
Row Identifiers (/docs/row-identifiers)
RESTful Verbs (/docs/verbs)
Application Tokens (/docs/app-tokens)
Authentication (/docs/authentication)
Response Codes & Headers (/docs/response-codes)
System Fields (/docs/system-fields)
CORS & JSONP (/docs/cors-and-jsonp)

Filtering & Querying

Simple Filters (/docs/filtering)
SoQL Queries (/docs/queries/)
Paging Through Data (/docs/paging)
SoQL Function and Keyword Listing (/docs/functions/)
Data Transform Functions (/docs/transforms/)

Data Formats (/docs/formats/)

JSON (/docs/formats/json)
GeoJSON (/docs/formats/geojson)
CSV (/docs/formats/csv)
RDF-XML (/docs/formats/rdf-xml)

Datatypes (/docs/datatypes/)

Checkbox (/docs/datatypes/checkbox)
Fixed Timestamp (/docs/datatypes/fixed_timestamp)
Floating Timestamp (/docs/datatypes/floating_timestamp)
Line (/docs/datatypes/line)
Location (/docs/datatypes/location)
MultiLine (/docs/datatypes/multiline)
MultiPoint (/docs/datatypes/multipoint)
MultiPolygon (/docs/datatypes/multipolygon)
Number (/docs/datatypes/number)
Point (/docs/datatypes/point)
Polygon (/docs/datatypes/polygon)
Text (/docs/datatypes/text)
URL (/docs/datatypes/url)

Other APIs (/docs/other/)

These are the transformation functions available in the Dataset Management API (/publishers/dsmapi). These functions can be used to transform and validate your data before you publish your dataset for consumption.

These functions can be used in the “Data Transforms” editor of the the Dataset Management Experience (https://support.socrata.com/hc/en-us/articles/115016067067-Using-the-Socrata-Data-Management-Experience) interface. Check out some of the examples on our Support Portal here (https://support.socrata.com/hc/en-us/articles/360034530954-Data-Transformation-Examples)!

See the Dataset Management API docs (/docs/other/publishing) for more info on how to use the transform functions as an API user.

Function Name	Description
`+` (/docs/transforms/add)	Keep a number’s sign
`and` (/docs/transforms/and)	Logical and of two boolean values
`||` (/docs/transforms/concatenate)	concatenate two strings
`/` (/docs/transforms/divide)	Divide a number by another
`=` (/docs/transforms/equal)	Return true if the left side equals the right
`==` (/docs/transforms/equal)	Return true if the left side equals the right
`^` (/docs/transforms/exponent)	No documentation is available.
`>` (/docs/transforms/greater_than)	Return true if the value on the left is greater than the value on the right
`>=` (/docs/transforms/greater_than_equal)	Return true if the value on the left is greater than or equal to the value on the right
`<` (/docs/transforms/less_than)	Return true if the value on the left is less than the value on the right
`<=` (/docs/transforms/less_than_equal)	Return true if the value on the left is less than or equal to the value on the right
`%` (/docs/transforms/modulo)	Find the remainder(modulus) of one number divided by another
`*` (/docs/transforms/multiply)	Multiply two numbers together
`not` (/docs/transforms/not)	Invert a boolean
`<>` (/docs/transforms/not_equal)	Return true if the left side does not equal the right
`!=` (/docs/transforms/not_equal)	Return true if the left side does not equal the right
`or` (/docs/transforms/or)	Logical or of two boolean values
`-` (/docs/transforms/subtract)	Subtract a number from another
`abs` (/docs/transforms/abs)	Produce the absolute value of a number
`between` (/docs/transforms/between)	Return true if the left is within the range of the right values
`case` (/docs/transforms/case)	Evaluate a series of true/false expressions (predicates) and return the next consequent.
`centroid` (/docs/transforms/centroid)	returns the geometric centroid of a polygon or multipolygon. Please refer to
`coalesce` (/docs/transforms/coalesce)	Take the leftmost non-null value.
`contains` (/docs/transforms/contains)	tell whether or not a string contains another string
`county_boundary` (/docs/transforms/county_boundary)	Returns the boundary of the US county as a multipolygon. The state name is not case sensitive.
`date_extract_d` (/docs/transforms/date_extract_d)	Extract the day from the date as an integer
`date_extract_dow` (/docs/transforms/date_extract_dow)	Extracts the day of the week as an integer between 0 and 6 where
`date_extract_hh` (/docs/transforms/date_extract_hh)	Extract the hour the date as an integer
`date_extract_m` (/docs/transforms/date_extract_m)	Extract the month as an integer
`date_extract_mm` (/docs/transforms/date_extract_mm)	Extract the minute from the date as an integer
`date_extract_ss` (/docs/transforms/date_extract_ss)	Extract the second from the date as an integer
`date_extract_woy` (/docs/transforms/date_extract_woy)	Extracts the week of the year as an integer between 0 and 51
`date_extract_y` (/docs/transforms/date_extract_y)	Extract the year as an integer

Function Name	Description
`date_trunc_y` (/docs/transforms/date_trunc_y)	Truncates a calendar date at the year threshold
`date_trunc_ym` (/docs/transforms/date_trunc_ym)	Truncates a calendar date at the year/month threshold
`date_trunc_ymd` (/docs/transforms/date_trunc_ymd)	Truncates a calendar date at the year/month/day threshold
`datetime_add_d` (/docs/transforms/datetime_add_d)	Adds or subtracts the specified number of days to the timestamp
`datetime_add_hh` (/docs/transforms/datetime_add_hh)	Adds or subtracts the specified number of hours to the timestamp
`datetime_add_mm` (/docs/transforms/datetime_add_mm)	Adds or subtracts the specified number of minutes to the timestamp
`datetime_add_ss` (/docs/transforms/datetime_add_ss)	Adds or subtracts the specified number of seconds to the timestamp
`datetime_diff` (/docs/transforms/datetime_diff)	Calculates the difference between two dates in seconds, minutes, hours, days, business days, weeks, calendar weeks, months, or years.
`domain_categories` (/docs/transforms/domain_categories)	Returns the categories currently configured on the domain. Useful primarily
`domain_licenses` (/docs/transforms/domain_licenses)	Returns the licenses currently configured on the domain. Useful primarily
`email_parse` (/docs/transforms/email_parse)	Parse an email. This is best effort as most things are actually
`ensure_within` (/docs/transforms/ensure_within)	ensure_within is a function which takes a point and a multipolygon
`error` (/docs/transforms/error)	Make an error. This is useful in conjunction with a case function,
`floating_timestamp_day` (/docs/transforms/floating_timestamp_day)	Extract the day from a calendar date
`floating_timestamp_day_of_week` (/docs/transforms/floating_timestamp_day_of_week)	Extract the day of the week as an integer between 0 and 6 where Sunday is 0.
`floating_timestamp_hour` (/docs/transforms/floating_timestamp_hour)	Extract the hour from a calendar date
`floating_timestamp_minute` (/docs/transforms/floating_timestamp_minute)	Extract the minute from a calendar date
`floating_timestamp_month` (/docs/transforms/floating_timestamp_month)	Extract the month from a calendar date
`floating_timestamp_second` (/docs/transforms/floating_timestamp_second)	Extract the second from a calendar date
`floating_timestamp_week_of_year` (/docs/transforms/floating_timestamp_week_of_year)	Extract the week from a calendar date as an integer between 0 and 51.
`floating_timestamp_year` (/docs/transforms/floating_timestamp_year)	Extract the year from a calendar date
`forgive` (/docs/transforms/forgive)	forgive can take an optional default argument
`from_polyline` (/docs/transforms/from_polyline)	convert a linestring encode in Google's polyline format with the given precision to a Line
`geocode` (/docs/transforms/geocode)	geocode is a function which takes human readable addresses
`geocode_esri` (/docs/transforms/geocode_esri)	geocode_esri is a function which takes human readable addresses
`grapheme_length` (/docs/transforms/grapheme_length)	the length of a piece of text in unicode grapheme clusters.
`greatest` (/docs/transforms/greatest)	return the largest value among its arguments (ignoring null)
`hash` (/docs/transforms/hash)	Construct a hash value from a string value using either the md5 or sha256 algorithm.
`haversine_distance` (/docs/transforms/haversine_distance)	Return the distance of the line using haversine formula
`http_get` (/docs/transforms/http_get)	Make an HTTP Get request to a URL. The response is returned. If the server
`in` (/docs/transforms/in)	Whether or not a value is in a set of other values
`is_empty` (/docs/transforms/is_empty)	Returns whether or not the input is empty. Empty means null values,
`is_not_null` (/docs/transforms/is_not_null)	Whether or not a value is not null

Function Name	Description
`is_null` (/docs/transforms/is_null)	Whether or not a value is null
`is_within` (/docs/transforms/is_within)	is_within is a function which takes a point and a multipolygon
`json_array_contains` (/docs/transforms/json_array_contains)	Test if a json array contains an item. If the JSON passed to this function is not an array,
`json_pluck` (/docs/transforms/json_pluck)	Pluck a value out of a JSON string. The returned value will be a SoQL Json value.
`json_pluck_boolean` (/docs/transforms/json_pluck_boolean)	Pluck a boolean value out of a JSON string. The returned value must be a boolean, otherwise
`json_pluck_number` (/docs/transforms/json_pluck_number)	Pluck a number value out of a JSON string. The returned value must be a number, otherwise
`json_pluck_text` (/docs/transforms/json_pluck_text)	Pluck a text value out of a JSON string. The returned value may be a primitive like a
`least` (/docs/transforms/least)	return the smallest value among its arguments (ignoring null)
`left_pad` (/docs/transforms/left_pad)	Pad `text` with the minimum number of copies of `pad` to reach `desired_length`.
`length` (/docs/transforms/length)	the length of a piece of text in unicode code points. This is usually, but not
`like` (/docs/transforms/like)	If a string is like another string.
`location_address` (/docs/transforms/location_address)	Extract the address from a location
`location_city` (/docs/transforms/location_city)	Extract the city from a location
`location_point` (/docs/transforms/location_point)	Extract the point from a location
`location_state` (/docs/transforms/location_state)	Extract the state from a location
`location_to_point` (/docs/transforms/location_to_point)	Turn a location value into a point
`location_zip` (/docs/transforms/location_zip)	Extract the zip from a location
`lower` (/docs/transforms/lower)	lowercase a string
`make_location` (/docs/transforms/make_location)	This function has been deprecated. Please use the make_point function instead.
`make_point` (/docs/transforms/make_point)	function to make a point out of a Y (latitude) and X (longitude) coordinate.
`make_url` (/docs/transforms/make_url)	No documentation is available.
`not_between` (/docs/transforms/not_between)	Return true if the left is not within the range of the right values
`not_in` (/docs/transforms/not_in)	Whether or not a value is absent from a set of other values
`not_like` (/docs/transforms/not_like)	If a string is not like another string.
`parse_address` (/docs/transforms/parse_address)	Extract a street address from a full US address.
`parse_city` (/docs/transforms/parse_city)	Extract a city from a full US address.
`parse_point` (/docs/transforms/parse_point)	Extract the point from a full US address with point.
`parse_state` (/docs/transforms/parse_state)	Extract a state from a full US address.
`parse_zip` (/docs/transforms/parse_zip)	Extract a ZIP code from a full US address.
`point_latitude` (/docs/transforms/point_latitude)	Extract the latitude from a point
`point_longitude` (/docs/transforms/point_longitude)	Extract the longitude from a point
`polylabel` (/docs/transforms/polylabel)	Returns a point that must exist within the polygon borders. It uses the recursive grid-based algorithm described here: https://github.com/mapbox/polylabel#how-the-algorithm-works. When given a multipolygon, the point it returns is within the largest (by area) sub-polygon.
`random_number_between` (/docs/transforms/random_number_between)	Returns a random float using a uniform distribution between the lower and upper values supplied: random_number_between(lower, upper)
`random_number_normal` (/docs/transforms/random_number_normal)	Returns a random float using a normal distribution with the mean and variance supplied: random_number_normal(mean, variance)

Function Name	Description
`regex_capture` (/docs/transforms/regex_capture)	function to capture a piece of text based on a regular expression
`regex_named_capture` (/docs/transforms/regex_named_capture)	capture a piece of text based on a regular expression
`regex_replace` (/docs/transforms/regex_replace)	function to replace a piece of text based on a regular expression
`region_code` (/docs/transforms/region_code)	Turn a point into the ID of a region, based on which region the point falls within. For example, if this dataset can produce
`region_code_label` (/docs/transforms/region_code_label)	Identical to region_code, but returns a text value.
`repair_geometry` (/docs/transforms/repair_geometry)	Attempt to repair the geometry.
`replace` (/docs/transforms/replace)	replace text with another piece of text
`replace_first` (/docs/transforms/replace_first)	replace the first occurrence of a piece of text with another piece of text
`reproject` (/docs/transforms/reproject)	reproject a geometry from one projection to another.
`reproject_to_wgs84` (/docs/transforms/reproject_to_wgs84)	function to reproject a geometry to WGS84. This will allow the geometry
`right_pad` (/docs/transforms/right_pad)	Pad `text` with the minimum number of copies of `pad` to reach `desired_length`.
`round` (/docs/transforms/round)	Round a number to a given precision. Trailing zeros are removed by default. Negative precisions round numbers to the left of the decimal.
`set_projection` (/docs/transforms/set_projection)	function to explicitly set the projection value on geometries which do not have projection
`simplify` (/docs/transforms/simplify)	Returns a simplified version of the Line, Polygon, MultiLine, or MultiPolygon using
`simplify_preserve_topology` (/docs/transforms/simplify_preserve_topology)	Returns a simplified version of the Line, Polygon, MultiLine, or MultiPolygon using
`slice` (/docs/transforms/slice)	Get a substring of a specified length of a text from a start index
`source_created_at` (/docs/transforms/source_created_at)	Get the fixed timestamp that this data source was created (ie: started uploading or importing).
`split_select` (/docs/transforms/split_select)	function to split a piece of text on a token, and then select
`starts_with` (/docs/transforms/starts_with)	tell whether or a not a string is prefixed with another string
`state_boundary` (/docs/transforms/state_boundary)	returns the boundary of the US state
`title_case` (/docs/transforms/title_case)	Make string title case with the exception of small words as defined by NYT Style Guide:
`to_boolean` (/docs/transforms/to_boolean)	cast a value to a true or false
`to_checkbox` (/docs/transforms/to_checkbox)	No documentation is available.
`to_fixed_timestamp` (/docs/transforms/to_fixed_timestamp)	Turn a text value into a datetime with a fixed timezone.
`to_floating_timestamp` (/docs/transforms/to_floating_timestamp)	Turn a text value into a floating datetime. “Floating” means the timezone
`to_json` (/docs/transforms/to_json)	cast a text value to json
`to_line` (/docs/transforms/to_line)	parse a WKT (text) representation of a line into a line value
`to_location` (/docs/transforms/to_location)	This function has been deprecated. Please use the to_point function instead.
`to_multiline` (/docs/transforms/to_multiline)	convert a line into a multiline
`to_multipoint` (/docs/transforms/to_multipoint)	convert a point into a multipoint
`to_multipolygon` (/docs/transforms/to_multipolygon)	convert a polygon into a multipolygon
`to_number` (/docs/transforms/to_number)	cast a value to a number
`to_point` (/docs/transforms/to_point)	parse a WKT (text) representation of a point into a point value

Function Name	Description
`to_polygon` (/docs/transforms/to_polygon)	parse a WKT (text) representation of a polygon into a polygon value
`to_text` (/docs/transforms/to_text)	No documentation is available.
`to_url` (/docs/transforms/to_url)	No documentation is available.
`trim` (/docs/transforms/trim)	trim characters off the start and end of a string
`trim_leading` (/docs/transforms/trim_leading)	trim characters off the start of a string
`trim_trailing` (/docs/transforms/trim_trailing)	trim characters off the end of a string
`upper` (/docs/transforms/upper)	uppercase a string
`uri_parse` (/docs/transforms/uri_parse)	Parse a URI.
`url_decode` (/docs/transforms/url_decode)	URL Decode a value
`url_description` (/docs/transforms/url_description)	Extract the description part of a link.
`url_encode` (/docs/transforms/url_encode)	URL Encode a value.
`url_url` (/docs/transforms/url_url)	Extract the url part of a link.
`validate_geometry` (/docs/transforms/validate_geometry)	Test that the geometry is valid.
`xml_pluck` (/docs/transforms/xml_pluck)	Pluck a value out of an XML string using XPath. The returned value will be a string.

Datatypes

Overview

Filtering & Querying

There are many core datatypes in SODA. What datatypes you may find depends on the version of your API endpoint:

2.1 2.0

Datatype	Availability » (/docs/endpoints)
Checkbox (/docs/datatypes/checkbox)	2.0 and 2.1
Fixed Timestamp (/docs/datatypes/fixed_timestamp)	2.0 and 2.1
Floating Timestamp (/docs/datatypes/floating_timestamp)	2.0 and 2.1
Line (/docs/datatypes/line)	2.1
Location (/docs/datatypes/location)	2.0 and 2.1
MultiLine (/docs/datatypes/multiline)	2.1
MultiPoint (/docs/datatypes/multipoint)	2.1
MultiPolygon (/docs/datatypes/multipolygon)	2.1
Number (/docs/datatypes/number)	2.0 and 2.1
Point (/docs/datatypes/point)	2.1
Polygon (/docs/datatypes/polygon)	2.1
Text (/docs/datatypes/text)	2.0 and 2.1
URL (/docs/datatypes/url)	2.0 and 2.1

Fixed Timestamp Datatype

Overview

API Endpoints (/docs/endpoints)
Row Identifiers (/docs/row-identifiers)
RESTful Verbs (/docs/verbs)
Application Tokens (/docs/app-tokens)
Authentication (/docs/authentication)
Response Codes & Headers (/docs/response-codes)
System Fields (/docs/system-fields)
CORS & JSONP (/docs/cors-and-jsonp)

Filtering & Querying

Simple Filters (/docs/filtering)
SoQL Queries (/docs/queries/)
Paging Through Data (/docs/paging)
SoQL Function and Keyword Listing (/docs/functions/)
Data Transform Functions (/docs/transforms/)

Data Formats (/docs/formats/)

JSON (/docs/formats/json)
GeoJSON (/docs/formats/geojson)
CSV (/docs/formats/csv)
RDF-XML (/docs/formats/rdf-xml)

Datatypes (/docs/datatypes/)

Checkbox (/docs/datatypes/checkbox)
Fixed Timestamp (/docs/datatypes/fixed_timestamp)
Floating Timestamp (/docs/datatypes/floating_timestamp)
Line (/docs/datatypes/line)
Location (/docs/datatypes/location)
MultiLine (/docs/datatypes/multiline)
MultiPoint (/docs/datatypes/multipoint)
MultiPolygon (/docs/datatypes/multipolygon)
Number (/docs/datatypes/number)
Point (/docs/datatypes/point)
Polygon (/docs/datatypes/polygon)
Text (/docs/datatypes/text)
URL (/docs/datatypes/url)

Other APIs (/docs/other/)

Fixed timestamps represent an exact point on the time-line, limited to millisecond precision, encoded as ISO8601 Times (https://en.wikipedia.org/wiki/ISO_8601#Times), with a timezone offset. The timezone offset returned will always be expressed in UTC (Coordinated Universal Time), with a special UTC designator (“Z”). For example:

```
[ {
  "date_time_column": "2014-10-13T00:00:00.000Z"
} ]
```

When writing data, accuracy to only the second is required, but the service will always return precision to the millisecond. Additionally, you can use offsets like `-0800` (or `-08:00`). Using this offset system, `America/Los_Angeles` is expressed as `UTC-08:00`, or `UTC-07:00` in the summer. The offset `-08:00` indicates that `America/Los_Angeles` time is obtained from UTC by adding `-08:00`, that is, by subtracting 8 hours. Accordingly, `2020-10-16T18:00:00.000Z` and `2020-10-16T10:00:00.000-0800` are two different but equally valid representations of the same `fixed_timestamp` value.

If you are a data publisher, please note that the SODA Producer API (/publishers/soda-producer/soda-producer-basics) converts values with non-UTC offsets into the equivalent UTC representation of the value, so if you write a value like `2020-10-16T10:00:00.000-0800` to a `fixed_timestamp` field, the SODA Consumer API (/consumers/getting-started) will return that value as `2020-10-16T18:00:00.000Z`.

The following operators can be used to compare and manipulate `fixed_timestamp` fields:

Operator	Description
`<`	`TRUE` when the first date is earlier than the second date
`<=`	`TRUE` when the first date is earlier than or at the same time as the second date
`>`	`TRUE` when the first date is after the second date
`>=`	`TRUE` when the first date is after or at the same time as the second date
`!=`	`TRUE` when two dates are not at the same time
`=`	`TRUE` when two dates are at the same time
`IS NULL`	`TRUE` for dates that are `NULL`.
`IS NOT NULL`	`TRUE` for dates that are not `NULL`.

And the following functions can be used to filter and manipulate them:

2.1 2.0

Keyword Name	Description	Availability ❓ (/docs/endpoints)
`distinct` (/docs/functions/distinct)	Returns distinct set of records	2.1

Function Name	Description	Availability ❓ (/docs/endpoints)
`between ... and ...` (/docs/functions/between)	Returns TRUE for values in a given range	2.1
`case(...)` (/docs/functions/case)	Returns different values based on the evaluation of boolean comparisons	2.1
`count(...)` (/docs/functions/count)	Returns a count of a given set of records	2.0 and 2.1
`in(...)` (/docs/functions/in)	Matches values in a given set of options	2.1
`max(...)` (/docs/functions/max)	Returns the maximum of a given set of numbers	2.1
`min(...)` (/docs/functions/min)	Returns the minimum of a given set of numbers	2.1
`not between ... and ...` (/docs/functions/not_between)	Returns TRUE for values not in a given range	2.1
`not in(...)` (/docs/functions/not_in)	Matches values not in a given set of options	2.1

For example, to get all of the rows that were created before October 16, 2020 at 06:00:00 AM, we can query on the `:created_at` system field (/docs/system-fields), which is a `fixed_timestamp` type:

▶ try it (https://soda.demo.socrata.com/resource/6yvf-kk3n.json?$where=:created_at%20%3C%20'2020-10-15T03:00:00.000Z')

📘 docs (/foundry/soda.demo.socrata.com/6yvf-kk3n) | 📋 copy | </> json ▾

```
https://soda.demo.socrata.com/resource/6yvf-kk3n.json?$where=:created_at < '2020-10-15T03:00:00.000Z'
(https://soda.demo.socrata.com/resource/6yvf-kk3n.json?$where=:created_at%20%3C%20'2020-10-15T03:00:00.000Z')
```

Text strings will be automatically be cast when used in comparisons, as shown above.

Floating Timestamp Datatype

Floating timestamps represent an instant in time with millisecond precision, with no timezone value, encoded as ISO8601 Times (https://en.wikipedia.org/wiki/ISO_8601#Times) with no timezone offset. When writing data, accuracy to only the second is required, but the service will always return precision to the millisecond. For example:

```
[ {
  "date_time_column": "2014-10-13T00:00:00.000"
} ]
```

Datasets will either specify what timezone they should be interpreted in, or you can usually assume they're in the timezone of the publisher. For example, a dataset published by the City of Chicago (https://data.cityofchicago.org) will be published in Central Standard Time. While functionally a `floating_timestamp` is distinct from a `text` datatype, it may be helpful to think of the *value* of a `floating_timestamp` as simply a `text` string, with no inherent timezone information.

The following operators can be used to compare and manipulate `floating_timestamp` fields:

Operator	Description
`<`	`TRUE` when the first date is earlier than the second date
`<=`	`TRUE` when the first date is earlier than or at the same time as the second date
`>`	`TRUE` when the first date is after the second date
`>=`	`TRUE` when the first date is after or at the same time as the second date
`!=`	`TRUE` when two dates are not at the same time
`=`	`TRUE` when two dates are at the same time
`IS NULL`	`TRUE` for dates that are `NULL`.
`IS NOT NULL`	`TRUE` for dates that are not `NULL`.

And the following functions can be used to filter and manipulate them:

2.1 2.0

Keyword Name	Description	Availability ❓ (/docs/endpoints)
`distinct` (/docs/functions/distinct)	Returns distinct set of records	2.1

Function Name	Description	Availability ❓ (/docs/endpoints)
`between ... and ...` (/docs/functions/between)	Returns TRUE for values in a given range	2.1
`case(...)` (/docs/functions/case)	Returns different values based on the evaluation of boolean comparisons	2.1
`count(...)` (/docs/functions/count)	Returns a count of a given set of records	2.0 and 2.1
`date_extract_d(...)` (/docs/functions/date_extract_d)	Extracts the day from the date as an integer.	2.1
`date_extract_dow(...)` (/docs/functions/date_extract_dow)	Extracts the day of the week as an integer between 0 and 6 (inclusive).	2.1
`date_extract_hh(...)` (/docs/functions/date_extract_hh)	Extracts the hour of the day as an integer between 0 and 23 (inclusive).	2.1
`date_extract_m(...)` (/docs/functions/date_extract_m)	Extracts the month as an integer.	2.1
`date_extract_mm(...)` (/docs/functions/date_extract_mm)	Extracts the minute from the time as an integer.	2.1
`date_extract_ss(...)` (/docs/functions/date_extract_ss)	Extracts the second from the time as an integer.	2.1
`date_extract_woy(...)` (/docs/functions/date_extract_woy)	Extracts the week of the year as an integer between 0 and 51 (inclusive).	2.1
`date_extract_y(...)` (/docs/functions/date_extract_y)	Extracts the year as an integer.	2.1
`date_trunc_y(...)` (/docs/functions/date_trunc_y)	Truncates a calendar date at the year threshold	2.0 and 2.1
`date_trunc_ym(...)` (/docs/functions/date_trunc_ym)	Truncates a calendar date at the year/month threshold	2.0 and 2.1

Function Name	Description	Availability ❓ (/docs/endpoints)
`date_trunc_ymd(...)` (/docs/functions/date_trunc_ymd)	Truncates a calendar date at the year/month/date threshold	2.0 and 2.1
`greatest(...)` (/docs/functions/greatest)	Returns the largest value among its arguments, ignoring NULLs.	2.1
`in(...)` (/docs/functions/in)	Matches values in a given set of options	2.1
`least(...)` (/docs/functions/least)	Returns the smallest value among its arguments, ignoring NULLs.	2.1
`max(...)` (/docs/functions/max)	Returns the maximum of a given set of numbers	2.1
`min(...)` (/docs/functions/min)	Returns the minimum of a given set of numbers	2.1
`not between ... and ...` (/docs/functions/not_between)	Returns TRUE for values not in a given range	2.1
`not in(...)` (/docs/functions/not_in)	Matches values not in a given set of options	2.1

For example, to get all of the crimes that occurred between noon and 2PM on January 10th, 2015 in Chicago:

▶ try it (https://data.cityofchicago.org/resource/6zsd-86xi.json?$where=date%20between%20'2015-01-10T12:00:00'%20and%20'2015-01-10T14:00:00')

📓 docs (/foundry/data.cityofchicago.org/6zsd-86xi) | copy | </> json ▾

```
⚙ https://data.cityofchicago.org/resource/6zsd-86xi.json?$where=date between '2015-01-10T12:00:00' and '2015-01-10T14:00:00' (https://data.cityofchicago.org/resource/6zsd-86xi.json?$where=date%20between%20'2015-01-10T12:00:00'%20and%20'2015-01-10T14:00:00')
```

Text strings will be automatically be cast when used in comparisons, as shown above.

Line Datatype

Overview

API Endpoints (/docs/endpoints)
Row Identifiers (/docs/row-identifiers)
RESTful Verbs (/docs/verbs)
Application Tokens (/docs/app-tokens)
Authentication (/docs/authentication)
Response Codes & Headers (/docs/response-codes)
System Fields (/docs/system-fields)
CORS & JSONP (/docs/cors-and-jsonp)

Filtering & Querying

Simple Filters (/docs/filtering)
SoQL Queries (/docs/queries/)
Paging Through Data (/docs/paging)
SoQL Function and Keyword Listing (/docs/functions/)
Data Transform Functions (/docs/transforms/)

Data Formats (/docs/formats/)

JSON (/docs/formats/json)
GeoJSON (/docs/formats/geojson)
CSV (/docs/formats/csv)
RDF-XML (/docs/formats/rdf-xml)

Datatypes (/docs/datatypes/)

Checkbox (/docs/datatypes/checkbox)
Fixed Timestamp (/docs/datatypes/fixed_timestamp)
Floating Timestamp (/docs/datatypes/floating_timestamp)
Line (/docs/datatypes/line)
Location (/docs/datatypes/location)
MultiLine (/docs/datatypes/multiline)
MultiPoint (/docs/datatypes/multipoint)
MultiPolygon (/docs/datatypes/multipolygon)
Number (/docs/datatypes/number)
Point (/docs/datatypes/point)
Polygon (/docs/datatypes/polygon)
Text (/docs/datatypes/text)
URL (/docs/datatypes/url)

Other APIs (/docs/other/)

The `Line` datatype represents a path on the Earth as a sequence of WGS84 Latitude and Longitude pairs. The location is encoded as a GeoJSON “linestring” (https://geojson.org/geojson-spec.html#linestring). Example:

```
{
  "type": "LineString",
  "coordinates": [
    [102.0, 0.0], [103.0, 1.0], [104.0, 0.0], [105.0, 1.0]
  ]
}
```

Heads up! Contrary to the normal convention of "latitude, longitude" ordering in the `coordinates` property, GeoJSON orders the coordinates as "longitude, latitude" (X coordinate, Y coordinate), as other GIS coordinate systems are encoded. Note that the SoQL `within_box` and `within_circle` functions use the more conventional ordering, however.

The following operators can be used with `line` fields:

Operator	Description
`IS NULL`	`TRUE` for values that are `NULL`.
`IS NOT NULL`	`TRUE` for values that are not `NULL`.

And the following functions can be used with them:

2.1 2.0

Keyword Name	Description	Availability (/docs/endpoints)
`distinct` (/docs/functions/distinct)	Returns distinct set of records	2.1

Function Name	Description	Availability (/docs/endpoints)
`case(...)` (/docs/functions/case)	Returns different values based on the evaluation of boolean comparisons	2.1
`convex_hull(...)` (/docs/functions/convex_hull)	Returns the minimum convex geometry that encloses all of another geometry	2.1
`count(...)` (/docs/functions/count)	Returns a count of a given set of records	2.0 and 2.1
`extent(...)` (/docs/functions/extent)	Returns a bounding box that encloses a set of geometries	2.1
`intersects(...)` (/docs/functions/intersects)	Allows you to compare two geospatial types to see if they intersect or overlap each other	2.1
`num_points(...)` (/docs/functions/num_points)	Returns the number of vertices in a geospatial data record	2.1
`simplify(...)` (/docs/functions/simplify)	Reduces the number of vertices in a line or polygon	2.1
`simplify_preserve_topology(...)` (/docs/functions/simplify_preserve_topology)	Reduces the number of vertices in a line or polygon, preserving topology	2.1
`within_box(...)` (/docs/functions/within_box)	Returns the rows that have geodata within the specified box, defined by latitude, longitude corners	2.0 and 2.1
`within_circle(...)` (/docs/functions/within_circle)	Returns the rows that have locations within a specified circle, measured in meters	2.0 and 2.1
`within_polygon(...)` (/docs/functions/within_polygon)	Returns the rows that have locations within the specified box, defined by latitude, longitude corners	2.1

Location Datatype

Location is a type that contains latitude, longitude and address. It may be accessed as an object with 3 keys in it. These keys are:

- The `latitude` of the location. This may be null, if the object has been recently uploaded. This *must* be decimal degrees, for example: `41.8657007325722`
- The `longitude` of the location. This may be null, if the object has been recently uploaded. This *must* be in decimal degrees, for example: `-87.76110202195098`
- The `human_address`, which is a JSON object containing the U.S. address of the location. This may be null. The object has the following keys:
  - `address` - The street address of the location.
  - `city` - The city this address is in
  - `state` - The state this address is in
  - `zip` - The zip code for this address

The following functions can be used with `location` (/docs/datatypes/location) fields:

2.1 | 2.0

Keyword Name	Description	Availability (/docs/endpoints)

Function Name	Description	Availability (/docs/endpoints)
`case(...)` (/docs/functions/case)	Returns different values based on the evaluation of boolean comparisons	2.1
`within_box(...)` (/docs/functions/within_box)	Returns the rows that have geodata within the specified box, defined by latitude, longitude corners	2.0 and 2.1
`within_circle(...)` (/docs/functions/within_circle)	Returns the rows that have locations within a specified circle, measured in meters	2.0 and 2.1

For example, to query for all of the Seattle Fire 911 Calls (https://data.seattle.gov/Public-Safety/Seattle-Real-Time-Fire-911-Calls/kzjm-xkqj) calls within 500 meters of the Socrata offices in Seattle:

▶ try it (https://data.seattle.gov/resource/kzjm-xkqj.json?$where=within_circle(report_location,%2047.59815,%20-122.334540,%20500))

docs (/foundry/data.seattle.gov/kzjm-xkqj) copy </> json ▾

```
https://data.seattle.gov/resource/kzjm-xkqj.json?$where=within_circle(report_location, 47.59815, -122.334540, 500) (https://data.seattle.gov/resource/kzjm-xkqj.json?$where=within_circle(report_location,%2047.59815,%20-122.334540,%20500))
```

MultiLine Datatype

The `MultiLine` datatype represents a set of paths on the Earth as sequences of WGS84 Latitude and Longitude pairs. The location is encoded as a GeoJSON “multilinestring” (https://geojson.org/geojson-spec.html#multilinestring). Example:

```
{
  "type": "MultiLineString",
  "coordinates": [
    [ [100.0, 0.0], [101.0, 1.0] ],
    [ [102.0, 2.0], [103.0, 3.0] ]
  ]
}
```

Heads up! Contrary to the normal convention of "latitude, longitude" ordering in the `coordinates` property, GeoJSON orders the coordinates as "longitude, latitude" (X coordinate, Y coordinate), as other GIS coordinate systems are encoded. Note that the SoQL `within_box` and `within_circle` functions use the more conventional ordering, however.

The following operators can be used with `multiline` fields:

Operator	Description
`IS NULL`	`TRUE` for values that are `NULL`.
`IS NOT NULL`	`TRUE` for values that are not `NULL`.

And the following functions can be used with them:

2.1 2.0

Keyword Name	Description	Availability ❓ (/docs/endpoints)
`distinct` (/docs/functions/distinct)	Returns distinct set of records	2.1

Function Name	Description	Availability ❓ (/docs/endpoints)
`case(...)` (/docs/functions/case)	Returns different values based on the evaluation of boolean comparisons	2.1
`convex_hull(...)` (/docs/functions/convex_hull)	Returns the minimum convex geometry that encloses all of another geometry	2.1
`count(...)` (/docs/functions/count)	Returns a count of a given set of records	2.0 and 2.1
`extent(...)` (/docs/functions/extent)	Returns a bounding box that encloses a set of geometries	2.1
`intersects(...)` (/docs/functions/intersects)	Allows you to compare two geospatial types to see if they intersect or overlap each other	2.1
`num_points(...)` (/docs/functions/num_points)	Returns the number of vertices in a geospatial data record	2.1
`simplify(...)` (/docs/functions/simplify)	Reduces the number of vertices in a line or polygon	2.1
`simplify_preserve_topology(...)` (/docs/functions/simplify_preserve_topology)	Reduces the number of vertices in a line or polygon, preserving topology	2.1
`within_box(...)` (/docs/functions/within_box)	Returns the rows that have geodata within the specified box, defined by latitude, longitude corners	2.0 and 2.1
`within_circle(...)` (/docs/functions/within_circle)	Returns the rows that have locations within a specified circle, measured in meters	2.0 and 2.1
`within_polygon(...)` (/docs/functions/within_polygon)	Returns the rows that have locations within the specified box, defined by latitude, longitude corners	2.1

MultiPoint Datatype

The `MultiPoint` datatype is very similar to the `Location` (/docs/datatypes/location) datatype. It represents one or more locations on the Earth as a WGS84 Latitude and Longitude. The location is encoded as a GeoJSON "multipoint" (https://geojson.org/geojson-spec.html#multipoint). Example:

```
{
  "type": "MultiPoint",
  "coordinates": [ [100.0, 0.0], [101.0, 1.0] ]
}
```

Heads up! Contrary to the normal convention of "latitude, longitude" ordering in the `coordinates` property, the GeoJSON orders the coordinates as "longitude, latitude" (X coordinate, Y coordinate), as other GIS coordinate systems are encoded. Note that the SoQL `within_box` and `within_circle` functions use the more conventional ordering, however.

The following operators can be used on `multipoint` fields:

Operator	Description
`IS NULL`	`TRUE` for values that are `NULL`.
`IS NOT NULL`	`TRUE` for values that are not `NULL`.

And the following functions can be used with them:

2.1 2.0

Keyword Name	Description	Availability (/docs/endpoints)
`distinct` (/docs/functions/distinct)	Returns distinct set of records	2.1

Function Name	Description	Availability (/docs/endpoints)
`case(...)` (/docs/functions/case)	Returns different values based on the evaluation of boolean comparisons	2.1
`convex_hull(...)` (/docs/functions/convex_hull)	Returns the minimum convex geometry that encloses all of another geometry	2.1
`count(...)` (/docs/functions/count)	Returns a count of a given set of records	2.0 and 2.1
`extent(...)` (/docs/functions/extent)	Returns a bounding box that encloses a set of geometries	2.1
`intersects(...)` (/docs/functions/intersects)	Allows you to compare two geospatial types to see if they intersect or overlap each other	2.1
`num_points(...)` (/docs/functions/num_points)	Returns the number of vertices in a geospatial data record	2.1
`within_box(...)` (/docs/functions/within_box)	Returns the rows that have geodata within the specified box, defined by latitude, longitude corners	2.0 and 2.1
`within_circle(...)` (/docs/functions/within_circle)	Returns the rows that have locations within a specified circle, measured in meters	2.0 and 2.1
`within_polygon(...)` (/docs/functions/within_polygon)	Returns the rows that have locations within the specified box, defined by latitude, longitude corners	2.1

MultiPolygon Datatype

Overview

Filtering & Querying

Some geometries may be polygons with “holes” in the center, or may be made up of multiple disconnected polygons. In that case, the GeoJSON “multipolygon” (https://geojson.org/geojson-spec.html#multipolygon) is used. Example:

```
{
  "type": "MultiPolygon",
  "coordinates": [
    [[[102.0, 2.0], [103.0, 2.0], [103.0, 3.0], [102.0, 3.0], [102.0, 2.0]]],
    [[[100.0, 0.0], [101.0, 0.0], [101.0, 1.0], [100.0, 1.0], [100.0, 0.0]],
     [[100.2, 0.2], [100.8, 0.2], [100.8, 0.8], [100.2, 0.8], [100.2, 0.2]]]
  ]
}
```

> *Heads up!* Contrary to the normal convention of "latitude, longitude" ordering in the `coordinates` property, GeoJSON orders the coordinates as "longitude, latitude" (X coordinate, Y coordinate), as other GIS coordinate systems are encoded. Note that the SoQL `within_box` and `within_circle` functions use the more conventional ordering, however.

The following operators can be used on `multipolygon` fields:

Operator	Description
`IS NULL`	`TRUE` for values that are `NULL`.
`IS NOT NULL`	`TRUE` for values that are not `NULL`.

And the following functions can be used with them:

2.1 2.0

Keyword Name	Description	Availability (/docs/endpoints)
`distinct` (/docs/functions/distinct)	Returns distinct set of records	2.1

Function Name	Description	Availability (/docs/endpoints)
`case(...)` (/docs/functions/case)	Returns different values based on the evaluation of boolean comparisons	2.1
`convex_hull(...)` (/docs/functions/convex_hull)	Returns the minimum convex geometry that encloses all of another geometry	2.1
`count(...)` (/docs/functions/count)	Returns a count of a given set of records	2.0 and 2.1
`extent(...)` (/docs/functions/extent)	Returns a bounding box that encloses a set of geometries	2.1
`intersects(...)` (/docs/functions/intersects)	Allows you to compare two geospatial types to see if they intersect or overlap each other	2.1
`num_points(...)` (/docs/functions/num_points)	Returns the number of vertices in a geospatial data record	2.1
`simplify(...)` (/docs/functions/simplify)	Reduces the number of vertices in a line or polygon	2.1
`simplify_preserve_topology(...)` (/docs/functions/simplify_preserve_topology)	Reduces the number of vertices in a line or polygon, preserving topology	2.1
`within_box(...)` (/docs/functions/within_box)	Returns the rows that have geodata within the specified box, defined by latitude, longitude corners	2.0 and 2.1
`within_circle(...)` (/docs/functions/within_circle)	Returns the rows that have locations within a specified circle, measured in meters	2.0 and 2.1
`within_polygon(...)` (/docs/functions/within_polygon)	Returns the rows that have locations within the specified box, defined by latitude, longitude corners	2.1

Closely related to the MultiPolygon datatype is the `Polygon (/docs/datatypes/polygon)`.