ds.search REST

Description

Searches and retrieves data from Gigya's Data Store (DS) using an SQL-like query. For security reasons this method is not available for client side SDKs, only for server side SDKs. SQL queries are converted into Gigya's proprietary query language. SQL injection attacks are not possible because queries are both created by the customer and then converted by Gigya.

A short delay is possible between the writing of account data to the data base and its availability in queries.

Note: If you plan on integrating the DS, we highly recommend reading the Data Store Guide.
The DS is a premium platform that requires separate activation. If the DS is not part of your site package please contact your Gigya Customer Engagement Executive or email support@gigya-inc.com.

Query Syntax Specification

Gigya queries use the same syntax rules as SQL, however not all standard SQL key words are available.

- When querying for string values, value must be wrapped in double quotes. e.g., SELECT * FROM data.music WHERE guitar = "Fender".
- When querying for Integer (and other non-textual fields) values, value must **not** be wrapped in quotes. e.g., SELECT * FROM data.userAccounts WHERE age = 42.
- Unsupported SQL syntax in the query string (e.g., HAVING) will produce an error.
- The query string clauses must be ordered in the following way*:
    1. Select clause
    2. From clause
    3. Where clause
    4. Filter clause
    5. Group By clause
    6. Order By clause
    7. Start clause Or/And Limit clause

- Deleted objects do not appear in queries.
- Encrypted fields are decrypted during searches but comparison operators (>, >=, <, <=) and regex expressions are not available on these fields. The *Contains* keyword can be used for case-insensitive searches on encrypted fields but does not support partial strings.
- Example queries and responses can be viewed at DS.search Examples.

select - The "select" statement accepts a comma separated list of fields or objects to retrieve. Acceptable values for this statement are:

- Field names, specifying the complete path, i.e. *data.album.photo.photoTitle_t.* Specifying partial fields names (*data.album*) will return all the fields in the path.
- Object names, specifying an object name, i.e. *user_likes* will return all the fields in the *user_likes* object.
- Partial field names (fields that contain only a part of the path to sub-objects, i.e. *data.album*) - will retrieve everything below that path.
- ***** - will retrieve every field in the schema.
- ***count(*)*** - Returns the number of objects for a particular object *type* in the data store. The result is given in the response as a single value inside the "*data"* field.
- ***as*** - create an alias (temporary title) for the returned object or field. '*Select data.track.title as song...'* will return a field called *track.song* containing the values of *data.track.title*. Example:

```
// Query
SELECT data.music.trackName AS Track

// Result
{
    "results": [
        {"track": {"song": "I will follow"} },
        {"track": {"song": "Into the Heart"} },
        {"track": {"song": "The Ocean"} },
        {"track": {"song": "A Day Without Me"} },
        {"track": {"song": "Shadows and Trees"} }
    ],
    "objectsCount": 5,
    "totalCount": 1840,
    "statusCode": 200,
    "errorCode": 0,
    "statusReason": "OK",
    "callId": "2222"
}
```

- ***sum(), min(), max(), avg(), sum_of_squares(), variance(), std() -*** Mathematical functions, must all be performed on the same numeric field. Fields with null values will be ignored.
  The name of the field on which the function is to be performed must be entered in the brackets. For example: '*Select min(track) from data.music*'.
  - *sum* - provides a total for the field in the brackets.
  - *min/max* - minimum/maximum value for the field. If no values are found, ***min*** will return "infinity" and ***max*** will return "-infinity".
  - *avg* - average value of the field.
  - *variance* - the extent that the field's values vary.
  - *std* - standard deviation, the likelihood that values vary.

from - Name of the data source. Only one data source is supported. Queries on objects in the Data Store (DS.search) must specify the object "*ty pe*" classification as defined in the DS.store method.

where - The "where" clause defines conditions for selecting items from the collection. Supported operators:

- **>** , **>=**, **<**, **<=** , **=** , ***!=*** - the left operand must be a data field name (with a proper suffix letter) and the right operand must be a constant of the same type as the field. For example: "*where track.# >= 18* ".
  - *Only **=** and **!=** can be used with encrypted fields.*
- ***and*** , ***or***
- ***contains, not contains*** - may be used only on text (string) fields and arrays.
  - Text (string) fields - support standard full text search capabilities. **Contains** is not case-sensitive. The left operand must be a text field name and the right operand must be a constant string. You can search for a specific word within the string, for example: **WHERE data.about_t CONTAINS "music"** . Underscores are treated as separators between words. Contains cannot be used to locate words within encrypted fields (you must enter the full string) and is case insensitive.
  - Arrays - the left operand must be an array field name and the right operand must be a constant of the same type as the array values. The array field name must have a suffix denoting the type of the arrays values. For example: **WHERE data.hobbies_s CONTAINS "swimming"** .

Note: You can only search words that are part of a sentence, you cannot search for parts of a word.

- ***in()*** - only retrieve items if the field contains one of the list values. For example: *'select * from data.music where track.album_artist in ("Elvis", "Beatles", "Santana")'* will return tracks recorded by the specified artists.
- ***is null*** , ***is not null***
- ***not***
- ***regex*** ('') - defines a search term using regex formatting. The regex syntax can be found in: Regular Expression Language Reference. Regex patterns cannot be used on encrypted fields.

order by - The "*order by*" clause specifies a list of fields by which to sort the result objects.

limit - Using the "limit" clause, you may specify the maximum number of returned result objects. If not specified, the default is **300**. The maximum limit value accepted is **10000**. If the search is sent with openCursor = true, *limit* is not guaranteed.

start - The "start" clause (not an SQL standard clause) may be used for paging. The clause specifies the start index from which to return result objects.

The '*select - from - where - order by*' query creates an (internal) indexed list of objects. By using the *start* & *limit* clauses, you will receive a subset of this list, starting with the *start* index and ending with *start+limit* index.

Note: When implementing paging, there is no guarantee that there will be no duplications or that recently added data will show up in the query results.

Notes:

- Always prefer accessing data objects directly using their IDs (with ds.get) rather than using search, whenever possible. Accessing data objects directly using their OIDs is faster and optimal for run-time.
- After data is published to the DS, building the indexes is done asynchronously for performance reasons, and may take up to one second to complete.
- Please note, when using a cursor, the number of results in a batch (limit) is not guaranteed.
- When querying for large numbers of results (>500) it is Best Practice to use the **openCursor** and then the **cursorId** with the **nextCursorId** value rather than attempting to page through results using timestamps of the user records.

Query Optimization

Below are a few points to note regarding query optimization:

1. Query execution is based on clause position and is executed from left to right.
2. Place clauses that have the greatest impact on records returned at the beginning of your SQL statement. For example, to retrieve a list of male users over the age of 25:

   This is because filtering first by gender automatically reduces the result set by half, so the server only needs to run the next filter on half of the overall population.
3. A NOT clause (NOT or !) is executed on a single statement immediately to it's right, after analyzing the statement. A single statement can hold several conditions inside parentheses.
4. Date ranges are calculated much more efficiently using a *timestamp* field rather than a *date* field.
5. Use of regex is computationally intensive and can significantly increase response time.
6. AND clauses take precedence over OR clauses (i.e., AND clauses are executed before OR clauses).
7. Use parentheses to modify default precedence (e.g., to execute an OR operation before an AND operation).

Pagination

When running long queries (>5,000 records returned), it's best practice to paginate your results using cursors. If you do not use cursors, **results are limited to a total of 5,000 records per query** (not just per page).

To use cursors, during the first request, pass *query=<query to execute>* and *openCursor=true*. The response will include the *nextCursorId* field, containing a cursor ID to be used in the next request. On subsequent requests, pass *cursorId=<last response's nextCursorId>* and **do not** submit the query again. The absence of the *nextCursorId* field in a response indicates the end of the result set.

When using openCursor, you cannot use 'START'.

Request URL

Where **<Data_Center>** is:

- **`us1.gigya.com`** - For the US data center.
- **`eu1.gigya.com`** - For the European data center.
- **`au1.gigya.com`** - For the Australian data center.
- **`ru1.gigya.com`** - For the Russian data center.
- **`cn1.gigya-api.cn`** - For the Chinese data center.

If you are not sure of your site's data center, see Finding Your Data Center.

Parameters

Required	Name	Type	Description
	query	string	A SQL-like query specifying the data to retrieve. Please refer to the Query language specification section above.
	querySig	string	An HMAC_SHA1 signature proving that the search call is in fact coming from your client application, in order to prevent fraud. Follow the instructions in Constructing a Signature using the following base-string: ***query*** + "_" + ***expTime*** . When using cursors, this parameter should only be sent with the initial request and omitted from subsequent requests. This parameter is required *only* when calling the search method from **client** side (i.e., Mobile SDKs)
	expTime	string	The GMT time when the ***querySig*** parameter should expire. The expected format is the Unix time format (i.e., the number of seconds since Jan. 1st 1970). Gigya checks the time when the search request is received. If the time succeeds expTime, the request is considered forged. This parameter is required *only* when calling the search method from **client** side (i.e., Mobile SDKs)
	openCursor	Boolean	When set to true, the search response will include, in addition to the first page, another field named **nextCursorId**, which is used to fetch the next batch of results. This parameter should only be used on the first request and later should be removed from the request. When openCursor is true, the *Limit* clause sets the number of results returned in the batch. **Note:** You cannot use a cursor if you have a 'group by' or when using 'start'.
	cursorId	string	The cursor ID that contains the **nextCursorId** value received in the first search call. Note: You cannot pass both **cursorId** and **query** on the same request - **cursorId** brings the next page for the search for which it was opened. Also, the time between search requests using a **cursorId** must not exceed 5 minutes. **Note:** Each request should contain a different **cursorId** obtained from the response of the **previous** request (not the first) using the **nextCursorId** field. The exception to this rule is when a request fails or when a particular result set needs to be resent; in this case, resend the same **cursorID** (as long as it has not expired) to receive its associated result set.
	timeout	integer	The timeout for the request (in milliseconds). Default value is 20000 (20 seconds). Maximum allowed value is 60000 (60 seconds).
	format	string	Determines the format of the response. The options are: • *json* (default) • *jsonp* - if the format is jsonp then you are required to define a *callback* method (see parameter below).
	callback	string	This parameter is *required* only when the *format* parameter is set to *jsonp* (see above). In such cases this parameter should define the name of the callback method to be called in the response, along with the jsonp response data.
	context	string/JSON	This parameter may be used to pass data through the current method and return it, unchanged, within the response.
	dontHandleScreenSet	Boolean	This parameter may be used in order to suppress the showing of screen-sets as a result of API calls. Default is **false**.
	httpStatusCodes	Boolean	The default value of this parameter is *false*, which means that the HTTP status code in Gigya's response is always 200 (OK), even if an error occurs. The error code and message is given within the response data (see below). If this parameter is set to *true*, the HTTP status code in Gigya's response would reflect an error, if one occurred.

Authorization Parameters

Each REST API request must contain identification and authorization parameters.

Some REST APIs may function without these authorization parameters, however, when that occurs, these calls are treated as **client-side** calls and all client-side rate limits will apply. In order to not reach client-side IP rate limits that may impact your implementation when using server-to-server REST calls, it is **Recommended Best Practice** to always sign the request or use a secret. A non-exhaustive list of REST APIs that this may apply to are as follows:

- accounts.login
- socialize.login
- accounts.notifyLogin
- socialize.notifyLogin
- accounts.finalizeRegistration
- accounts.linkAccounts

Please refer to the Authorization Parameters section for details.

Response Data

Field	Type	Description

errorCode	integer	The result code of the operation. Code '0' indicates success, any other number indicates failure. For a complete list of error codes, see the Error Codes table.
errorMessage	string	A short textual description of an error, associated with the errorCode, for logging purposes. This field will appear in the response only in case of an error.
errorDetails	string	This field will appear in the response only in case of an error and will contain the exception info, if available.
fullEventName	string	The full name of the event that triggered the response. This is an internally used parameter that is not always returned and **should not** be relied upon by your implementation.
callId	string	Unique identifier of the transaction, for debugging purposes.
time	string	The time of the response represented in ISO 8601 format, i.e., yyyy-mm-dd-Thh:MM:ss.SSSZ or
statusCode	integer	The HTTP response code of the operation. Code '200' indicates success. This property is deprecated and only returned for backward compatibility.
statusReason	string	A brief explanation of the status code. This property is deprecated and only returned for backward compatibility.

data	Array	An array of data objects retrieved from the DS.
objectsCount	integer	The number of objects returned in the "*data*" array.
totalCount	integer	The total number of object that satisfy the query in the DB. This is useful for knowing how many objects are in the DB, when fetching limited amount using the "*limit*" parameter.
nextCursorId	string	Used to fetch the next batch of results. This parameter is not returned on the last batch of results, its absence means that the result set is finished.

Response Example

```
{
        "results": [{
            "oid": "050",
            "lastUpdated": 1344151787611,
            "lastUpdatedTime": "2012-08-05T07:29:47Z",
            "created": 1344151532974,
            "createdTime": "2012-08-05T07:25:32Z",
            "data": {
                "FN": 7,
                "Fn": 7,
                "field1": 7,
                "field2": 7,
                "field3": 7,
                "field4": "7",
                "field5": "7",
                "field6": "7",
                "field7": "7",
                "field8": "7",
                "field10": "d1_0",
                "field11": "d1_0",
                "field12": "d1_0",
                "field13": "d1_0",
                "fn": 7
            }},
        {
            "oid": "1c675e20f7724c93b9ca82aa7b206b41",
            "lastUpdated": 1344172297960,
        "lastUpdatedTime": "2012-08-05T13:11:37Z",
        "created": 1344172297960,
        "createdTime": "2012-08-05T13:11:37Z",
        "data": {
            "BloodType": "A",
            "Dislikes": "Evil",
            "EyeColor": "Brown",
            "HairColor": "Brown",
            "Height": "179 cms",
            "Likes": "Justice",
            "Occupation": "QA",
            "Weight": "94.1 kgs"
        }},
    {
        "UID": "_gid_t6BWv8yeJG6QZyq7Wtav1Q==",
        "oid": "51b5d8e72c664520a27c7020ff2f1494",
        "lastUpdated": 1344414722808,
        "lastUpdatedTime": "2012-08-08T08:32:02Z",
        "created": 1344414722808,
        "createdTime": "2012-08-08T08:32:02Z",
        "data": {
            "BloodType": "A",
            "Dislikes": "Evil",
            "EyeColor": "Brown",
            "HairColor": "Brown",
            "Height": "179 cms",
            "Likes": "Justice",
            "Occupation": "QA",
            "Weight": "94.1 kgs"
        }},
    {
        "oid": "677cc8b3437b40959b9b1288add0a2ec",
        "lastUpdated": 1344172153286,
        "lastUpdatedTime": "2012-08-05T13:09:13Z",
        "created": 1344172153286,
        "createdTime": "2012-08-05T13:09:13Z",
        "data": {
            "BloodType": "A",
            "Dislikes": "Evil",
            "EyeColor": "Brown",
            "HairColor": "Brown",
            "Height": "179 cms",
            "Likes": "Justice",
            "Occupation": "QA",
            "Weight": "94.1 kgs"
        }},
    {
        "oid": "8b8972c3b3114db2a0acdfa2617693bf",
        "lastUpdated": 1344428184385,
        "lastUpdatedTime": "2012-08-08T12:16:24Z",
        "created": 1344428184385,
        "createdTime": "2012-08-08T12:16:24Z",
        "data": {
            "BloodType": "A",
            "Dislikes": "Evil",
            "EyeColor": "Brown",
            "HairColor": "Brown",
            "Height": "179 cms",
            "Likes": "Justice",
            "Occupation": "QA",
            "Weight": "94.1 kgs"
        }},
    {
        "oid": "super1",
        "lastUpdated": 1344172332701,
        "lastUpdatedTime": "2012-08-05T13:12:12Z",
        "created": 1344172332701,
        "createdTime": "2012-08-05T13:12:12Z",
        "data": {
            "BloodType": "A",
            "Dislikes": "Evil",
            "EyeColor": "Brown",
            "HairColor": "Brown",
            "Height": "179 cms",
            "Likes": "Justice",
            "Occupation": "QA",
            "Weight": "94.1 kgs"
        }}],
    "objectsCount": 9,
    "totalCount": 9,
    "statusCode": 200,
    "errorCode": 0,
    "statusReason": "OK",
    "callId": "fb5e8b258afc4016ad454cc9fd1d64ec",
        "time": "2015-03-22T11:42:25.943Z"
    }
```

More Information

ds.search Examples