Talkwalker API

Talkwalker | 16, Avenue Monterey | L-2163 Luxembourg

Updated November 2015

Table of Contents

Talkwalker API Overview

Talkwalker Search API Overview & Example

How it works

The Talkwalker Search API allows you to retrieve up to 500 sorted results for a given timeframe within the last **30 days**. In addition, a histogram of the number of results can also be returned. You can sort the results by publication time, indexing time, engagement or other metrics. A single search query can support up to 50 operands. To create complex queries, operands may be combined using Boolean operators.

A few words about the results

Search results can be sorted by engagement, time or other metrics and be restricted to specific attribute value ranges (for example only return results published in a certain timerange). When no special filters are applied, a single search request will return results from **all media types** and **all languages** over the past **30 days** sorted by **engagement** by default. You don’t need to execute one search request for each language and media type separately. To get a smaller set of results, you can either get only the highest ranked results or get a random sample set.

A brief example (Search)

The Talkwalker API search results endpoint (`https://api.talkwalker.com/api/v1/search/results`) is used to search on the Talkwalker API. (For testing purpose the `access_token` `demo` can be used. Setting the variable `pretty=true` will return formatted results)

command:

```
curl 'https://api.talkwalker.com/api/v1/search/results?access_token=demo&q=cats&pretty=true'
```

response (all responses are UTF-8):

```
{
  "status_code" : "0",
  "status_message" : "OK",
  "request" : "GET /api/v1/search/results?access_token=demo&q=cats&pretty=true",
  "pagination" : {
    "next" : "GET /api/v1/search/results?access_token=demo&q=cats&pretty=true&offset=10",
    "total" : 298138
  },
  "result_content" : {
    "data" : [ {
      "data" : {
        "url" : "http://annukcreations.blogspot.com/2014/12/sunny-rings.html",
        "indexed" : 1417999367498,
        "search_indexed" : 1417999504832,
        "published" : 1417999319393,
        "title" : "Color and Light Inspirations in Jewelry: SUNNY RINGS :)",
        "content" : "Welcome to my colorful little island! This blog is about sharing my colorful world, my
sources of inspiration and all what fuels my imagination... Islands and kitties, beauty and art, nature and
love, and creative souls who inspire me! Thank you for following me on my journey!\n\nI am an artist and
jewelry maker from Turin, Italy and I am half Italian and half German. I have a background in Language studies
and a University degree in German and English, but I have always been fascinated by handmade objects, art,
creativity and color. This resulted in my passion for handmade jewelry! Like many jewelry makers and artists,
my first jewels were made with beads, but soon I discovered the potentials of so many materials and I
developed my very personal style. I would describe myself as a mixed-media and eclectic artist. My favorite
materials include glass, polymer clay, metal sheets and wood, but as I love experimenting the possibilities
are endless! What I love most about the creative process is the modeling and combining of materials. I
especially make rings and pendants, but you will find some pins and earrings as well. All my pieces are one-
of-a-kind, so no two pieces are the same! I love traveling and much of my work reflects the memories of places
I love. I also like to bring back from my trips beautiful and unique glass and ceramic beads and cabochons,
and found pieces such as ceramic shards and beach pottery to incorporate in my work or use as focal pieces. In
recent years,...",
        "title_snippet" : "Color and Light Inspirations in Jewelry: SUNNY RINGS :)",
        "root_url" : "http://annukcreations.blogspot.com/",
        "domain_url" : "http://blogspot.com/",
        "host_url" : "http://annukcreations.blogspot.com/",
        "parent_url" : "http://annukcreations.blogspot.com/2014/12/sunny-rings.html",
        "lang" : "en",
        "porn_level" : 0,
        "fluency_level" : 90,
        "spam_level" : 20,
        "sentiment" : 5,
        "source_type" : [ "BLOG", "BLOG_OTHER" ],
        "post_type" : [ "TEXT" ],
        "tokens_title" : [ "and Light Inspirations", "Light Inspirations", "Light Inspirations", "SUNNY
RINGS", "SUNNY RINGS", "and Light", "Inspirations", "Inspirations", "RINGS", "RINGS", "Light", "Light",
"Jewelry", "Jewelry", "Color", "Color", "SUNNY", "SUNNY" ],
        "tokens_content" : [ "Bead Hoarder Blog", "Bead Hoarder Blog"],
        "tokens_mention" : [ "@yahoo" ],
        "tags_internal" : [ "isQuestion" ],
        "article_extended_attributes" : {
          "num_comments" : 3
        },
        "source_extended_attributes" : {
          "alexa_pageviews" : 0
        },
        "extra_article_attributes" : {
          "world_data" : { }
        },
        "extra_author_attributes" : {
          "world_data" : { },
          "id" : "ex:annukcreations.blogspot.com-698904645",
          "name" : "view my complete profile",
          "gender" : "MALE"
        },
        "extra_source_attributes" : {
          "world_data" : {
            "continent" : "North America",
            "country" : "United States",
            "region" : "District of Columbia",
            "city" : "Washington, D.C.",
            "longitude" : -77.0094185808,
            "latitude" : 38.8995493765,
            "country_code" : "us"
          },
          "id" : "ex:annukcreations.blogspot.com",
          "name" : "http://annukcreations.blogspot.com/"
        },
        "engagement" : 3,
        "reach" : 0
      }
    }, {
      "data" : {
        "url" : "http://slshoeicidal.wordpress.com/2014/12/06/high-rez-snobbery-715-winter-trend-ice/",
        ... // truncated
```

more on the Talkwalker Search API

Talkwalker Streaming API Overview & Example

How it works

The Talkwalker Streaming API delivers real-time data through a persistent connection to our servers. Configure your stream with a set of filtering rules, connect to the stream and new results will be delivered in real time, as soon as they are found by our crawlers. You will not need to do any polling to receive new data.

You setup and configure the Streaming API by defining rules (Boolean query, language, media types, etc.). The Streaming API then finds and collects all relevant data and adds it to your data stream, with individually highlighted snippets per matched rule. This feature allows you to gather data from many rules through a single stream while easily matching the results back to your predefined rules.

Each rule allows filtering by title, content, author, language, URL, country, media type, and more parameters, using the same syntax as in our Talkwalker Search interface. You can also apply a list of sources to be included or excluded from the stream, to give you even further possibilities to narrow down the results you will get. A single rule can support up to 50 operands. To create complex rules, operands may be combined using Boolean Operators.

The documents are streamed in the order they are found by our crawlers and added to Talkwalker (i.e. by `search_indexed` timestamp). Custom sorting is not possible with the Streaming API (however this can be done with the Search API). The documents are grouped in timeframes which contain all documents that were indexed between the given start and end time of the timeframe.

Each result (independent on how many rules match) will be counted as 1 credit.

A brief example (Streaming)

The Talkwalker API streaming endpoint (`https://api.talkwalker.com/api/v2/stream`) is used to stream results from Talkwalker.

Creating a Stream

Command:

```
curl -XPUT https://api.talkwalker.com/api/v2/stream/s/teststream?access_token=demo -d '{ "rules" : [{
"rule_id": "rule-1", "query": "cats" }] }' -H 'Content-Type: application/json; charset=UTF-8'
```

Response:

```
{
  "status_code" : "0",
  "status_message" : "OK",
  "request" : "PUT /api/v2/stream/s/teststream?access_token=demo",
  "result_stream" : {
    "data" : [{
      "stream_id" : "teststream",
      "rules" : [{
        "rule_id" : "rule-1",
        "query" : "cats"
      }]
    }]
  }
}
```

Streaming

Example:

```
curl https://api.talkwalker.com/api/v2/stream/s/teststream/results?access_token=demo
```

The response is a stream of chunks, chunks contain meta data (`CT_CONTROL`) on the Talkwalker stream or search results (`CT_RESULT`).

response:

```
{
  "chunk_type" : "CT_CONTROL",
  "chunk_control" : {
    "timeframe_start" : 1430201017166,
    "timeframe_end" : 1430201040000,
    "stream" : [{
      "id" : "teststream",
      "status" : "active"
    }]
  }
}
{
  "chunk_type": "CT_RESULT",
  "chunk_result": {
    "data" : [ {
    "data" : {
      "url" : "http://annukcreations.blogspot.com/2014/12/sunny-rings.html",
      "indexed" : 1417999367498,
      "search_indexed" : 1417999504832,
      "published" : 1417999319393,
      "title" : "Color and Light Inspirations in Jewelry: SUNNY RINGS :)",
      "content" : "Welcome to my colorful little island! This blog is about sharing my colorful world, my sources of inspiration and all what fuels my imagination... Islands and kitties, beauty and art, nature and love, and creative souls who inspire me! Thank you for following me on my journey!\n\nI am an artist and jewelry maker from Turin, Italy and I am half Italian and half German. I have a background in Language studies and a University degree in German and English, but I have always been fascinated by handmade objects, art, creativity and color. This resulted in my passion for handmade jewelry! Like many jewelry makers and artists, my first jewels were made with beads, but soon I discovered the potentials of so many materials and I developed my very personal style. I would describe myself as a mixed-media and eclectic artist. My favorite materials include glass, polymer clay, metal sheets and wood, but as I love experimenting the possibilities are endless! What I love most about the creative process is the modeling and combining of materials. I especially make rings and pendants, but you will find some pins and earrings as well. All my pieces are one-of-a-kind, so no two pieces are the same! I love traveling and much of my work reflects the memories of places I love. I also like to bring back from my trips beautiful and unique glass and ceramic beads and cabochons, and found pieces such as ceramic shards and beach pottery to incorporate in my work or use as focal pieces. In recent years,...",
      "title_snippet" : "Color and Light Inspirations in Jewelry: SUNNY RINGS :)",
      "root_url" : "http://annukcreations.blogspot.com/",
      "domain_url" : "http://blogspot.com/",
      "host_url" : "http://annukcreations.blogspot.com/",
      "parent_url" : "http://annukcreations.blogspot.com/2014/12/sunny-rings.html",
      "lang" : "en",
      ...  // truncated
```

more on the Talkwalker Streaming API

Talkwalker Search API

Talkwalker Search Results API

```
https://api.talkwalker.com/api/v1/search/results
```

How it works

The Talkwalker Search API allows you to retrieve up to 500 sorted results for a given timeframe within the last **30 days**. In addition, a histogram of the number of results can also be returned. You can sort the results by publication time, indexing time, engagement or other metrics. A single search query can support up to 50 operands. To create complex queries, operands may be combined using Boolean operators.

A few words about the results

Search results can be sorted by engagement, time or other metrics and be restricted to specific attribute value ranges (for example only return results published in a certain timerange). When no special filters are applied, a single search request will return results from **all media types** and **all languages** over the past **30 days** sorted by **engagement** by default. You don't need to execute one search request for each language and media type separately. To get a smaller set of results, you can either get only the highest ranked results or get a random sample set.

Parameters

parameter	description	required?	default value
`access_token`	API access token	required	
`q`	The query to search for	required	
`offset`	Number of results to skip (for paging)	optional	default: 0
`hpp`	Number of hits per page (for paging)	optional	default: 10 / maximum : 500
`sort_by`	Criteria for sorting the results.	optional	default: engagement
`sort_order`	Sorting order (ascending or descending)	optional	default: desc
`hl`	Turns highlighting on or off	optional	default: 1
`pretty`	Formatted json for testing	optional	false

More on the Talkwalker Query Syntax

Credits

1 credit per returned result, at least 10 credits per call (e.g. 100 results = 100 credits, 10 results = 10 credits and 0 results = 10 credits).

Examples

Get 100 results containing the words "cats" and "dogs" but not "birds"

Set the query **cats AND dogs AND NOT birds** with `query=cats%20AND%20dogs%20AND%20NOT%20birds` (note: in URLs spaces are replaced by `%20`) and set hits per page to 100 with `hpp=100`.

```
curl
'https://api.talkwalker.com/api/v1/search/results?access_token=demo&q=cats%20AND%20dogs%20AND%20NOT%20birds&hp
p=100&pretty=true'
```

More on the Talkwalker Query Syntax

Get results containing the word "cats" sorted from new to old

To sort the results by date, set `sort_by` to `published` (to sort by the date of publication), to get the newest results first, set `sort_order=desc`.

```
curl
'https://api.talkwalker.com/api/v1/search/results?access_token=demo&q=cats&sort_by=published&sort_order=desc&p
retty=true'
```

All options for `sort_by` are : `reach`, `facebook_shares`, `facebook_likes`, `twitter_shares`, `twitter_retweets`, `twitter_followers`, `youtube_likes`, `youtube_dislikes`, `youtube_views`, `cluster_size`, `comment_count`, `published`, `search_indexed`

More on the document fields

Get results containing the word "dogs" published in american blogs

```
curl
'https://api.talkwalker.com/api/v1/search/results?access_token=demo&q=cats%20AND%20sourcetype:"BLOG"%20AND%20s
ourcecountry:us&pretty=true'
```

Talkwalker Search Histogram API

```
https://api.talkwalker.com/api/v1/search/histogram/<type>
```

How it works

With the Talkwalker Search Histogram API, you can retrieve the distribution of the number of search results for a given search query. Histograms can be made for distribution over time or over specific metrics (number of comments, number of shares, reach, retweets etc.). By setting `min` and `max` a histogram can be limited to a specific range (`min_include` and `max_include` control if those bounds are included). `interval` defines the width of the bins, the accepted values are long integers for metrics or duration values (like `7d` for 7 days) for `published` and `search_indexed` dates. When using a bin size of entire days, `timezone` allows to set a timezone to specify the begin and end of the days.

Types

type	Description
`published`	Timestamp of publication (epoch time in milliseconds)
`search_indexed`	Timestamp of indexation in Talkwalker (epoch time in milliseconds)
`reach`	The reach of an article/post represents the number of people who were reached by this article/post.
`engagement`	The engagement of an article/post is the sum of actions made by others on that article/post.
`facebook_shares`	Number of Facebook shares an article has
`facebook_likes`	Number of Facebook likes an article has
`twitter_retweets`	Number of Twitter retweets an article has
`twitter_shares`	Number of Twitter share an article has
`twitter_followers`	Number of Twitter followers a source has
`youtube_views`	Number of YouTube views a video has
`youtube_likes`	Number of YouTube likes a video has
`youtube_dislikes`	Number of YouTube dislikes a video has
`comment_count`	Number of Comments an article has

Parameters

parameter	description	required?	allowed values	default value
`access_token`	a read/write token specified in the API application	required		
`q`	The query to search for	required	Talkwalker query syntax	
`min`	Minimum value for bins	optional	Long Integer value	
`max`	Maximum value for bins	optional	Long Integer value	
`min_include`	Include min value	optional	`true` / `false`	`true`
`max_include`	Include max value	optional	`true` / `false`	`false`
`interval`	Bin Interval	optional	Long Integer (duration for `published` and `search_indexed`)	`dynamic`
`timezone`	Time zone (for interval)	optional	tz database timezone name (i.e. `Europe/Luxembourg)	`UTC`

Possible values for interval when creating a histogram over `published` or `search_indexed`: `year`, `quarter`, `month`, `week`, `day`, `hour`, `minute`, `second` as well as numeric values with the units `w` (week), `d` (day), `h` (hours), `m` (minutes), and `s` (seconds). (e.g. `5d` for 5 days or `2w` for 2 weeks).

The maximum number of histogram bins is 400, if the `min`, `max` and `interval` parameters result in a larger number of bins, an error message (HTTP 400) is returned. Try reducing the range or increasing the interval.

Credits

10 credits per call.

Examples

Get a histogram over the last 8 days of online news results containing the word "birds"

Set the query to `birds%20sourcetype:"ONLINENEWS"`. By default the Talkwalker Search Histogram API return results over the last seven days.

```
curl
'https://api.talkwalker.com/api/v1/search/histogram/published?access_token=demo&q=birds%20sourcetype:\"ONLINENEWS\"&interval=day&pretty=true'
```

response:

```
{
  "status_code" : "0",
  "status_message" : "OK",
  "request" : "GET
/api/v1/search/histogram?access_token=demo&q=birds%20sourcetype:\"ONLINENEWS\"&interval=7d&pretty=true",
  "result_histogram" : {
    "header" : {
      "v" : [ "Number Results" ]
    },
    "data" : [ {
      "t" : 1417478400000,
      "v" : [ 4366.0 ]
    }, {
      "t" : 1417564800000,
      "v" : [ 3385.0 ]
    }, {
      "t" : 1417651200000,
      "v" : [ 4233.0 ]
    }, {
      "t" : 1417737600000,
      "v" : [ 4071.0 ]
    }, {
      "t" : 1417824000000,
      "v" : [ 2571.0 ]
    }, {
      "t" : 1417910400000,
      "v" : [ 2191.0 ]
    }, {
      "t" : 1417996800000,
      "v" : [ 3275.0 ]
    }, {
      "t" : 1418083200000,
      "v" : [ 1140.0 ]
    } ]
  }
}
```

Get a histogram with a resolution of 6 hours over the last 7 days of results containing the word "birds"

Set `interval` to `6h` for 4 values per day.

```
curl
'https://api.talkwalker.com/api/v1/search/histogram/published?access_token=demo&q=birds&interval=6h&pretty=tru
e'
```

The `interval` parameter accepts the values `year`, `quarter`, `month`, `week`, `day`, `hour`, `minute`, `second` as well as numeric values with the units `w` (week), `d` (day), `h` (hours), `m` (minutes), and `s` (seconds).

Get a histogram over a specific range

Set `min` to `1390176000000` and `max` to `1390608000000` to get a histogram of results published between 20.01.2014 and 25.01.2014 with start timestamp included and end timestamp excluded (default values).

```
curl
'https://api.talkwalker.com/api/v1/search/histogram/published?access_token=demo&q=birds&min=1390176000000&max=
1390608000000&pretty=true'
```

The `min` and `max` parameters accept timestamps in epoch format (milliseconds after 1.1.1970 UTC).

Get a histogram and statistics over engagement

For types different from `published` and `search_indexed`, the histogram API also returns statistics (average, minimum, maximum and sum) over every bin.

```
curl 'https://api.talkwalker.com/api/v1/search/histogram/engagement?access_token=demo&q=birds&pretty=true'
```

response

```
{
  "status_code" : "0",
  "status_message" : "OK",
  "request" : "GET /api/v1/search/histogram/engagement?access_token=demo&q=birds&pretty=true",
  "result_histogram" : {
    "header" : {
      "v" : [ "Number Results" ]
    },
    "data" : [ {
      "v" : [ 333989.0 ],
      "k" : 0.0,
      "val" : [ {
        "count" : 333989,
        "min" : 0.0,
        "max" : 80759.0,
        "avg" : 22.01215608897299,
        "sum" : 7351818.0
      } ]
    }, {
      "v" : [ 5.0 ],
      "k" : 82254.0,
      "val" : [ {
        "count" : 0,
        "sum" : 0.0
      } ]
    }...
    // truncated
    {
      "v" : [ 1.0 ],
      "k" : 740286.0,
      "val" : [ {
        "count" : 1,
        "min" : 822531.0,
        "max" : 822531.0,
        "avg" : 822531.0,
        "sum" : 822531.0
      } ]
    } ]
  }
}
```

Talkwalker Search API and Talkwalker Projects

```
https://api.talkwalker.com/api/v1/search/p/<project_id>
```

How it works

Talkwalker users can use the topics defined in their project with the Talkwalker API. Topics can be used with the Search Results API and the Search Histogram API. This allows Talkwalker users to use the queries from their projects and to retrieve the documents they get in their Talkwalker project including changes and tags that were done in Talkwalker. In addition to the 30 days of search, the full history of Talkwalker projects is available in the search API, when used in combination with a Talkwalker project.

Parameters

parameter	description	required?	default value
`access_token`	API access token	required	
`q`	The query to search for.	required	
`offset`	Number of results to skip (for paging)	optional	default: 0
`hpp`	Number of hits per page (for paging)	optional	default: 10 / maximum : 500
`sort_by`	Criteria for sorting the results	optional	default: engagement
`sort_order`	Sorting order (ascending or descending)	optional	default: desc
`hl`	Turns highlighting on or off	optional	default: true
`topic`	One or more topics or panels that are defined in the Talkwalker project	optional, multiple	

Credits

1 credit per returned result, minimum 10 credits per Search Result API call.
10 credits per Search Histogram API call.
No credits for project list, topic list, document update and document delete calls.

Get a list of all projects linked to an API application

Use the private `access_token` from your API application on the `https://api.talkwalker.com/api/v1/search/info` endpoint to get the list of all linked projects.

```
curl 'https://api.talkwalker.com/api/v1/search/info?access_token=<access_token>'
```

Parameters

parameter	description	required?	default value
`access_token`	a read/write token specified in the API application	required	

Rate Limit

This endpoint is limited to 10 calls per minute, the result should be stored.

Get a list of all resources

Resources are data retrieval settings from a Talkwalker project. This can be search-topics, filters, monitored-pages, source-panels, events, or saved-objects for for embedding in external tools.

To get a list of the resources defined in a Talkwalker project use the `project_id` and the `access_token` on the `https://api.talkwalker.com/api/v2/talkwalker/p/<project_id>/resources` endpoint.

```
curl 'https://api.talkwalker.com/api/v2/talkwalker/p/<project_id>/resources?access_token=<access_token>'
```

Parameters

parameter	description	required?	values
`access_token`	a read/write token specified in the API application	required	
`type`	filter on the type of resources	optional	search, filter, page, event, panel, savedobject
`object_type`	filter on types of saved objects	optional	name of the saved-object type (name of the embedding destination)

Example: Get all saved objects from a project that were saved for embedding in an external tool called `myapp`.

```
curl
'https://api.talkwalker.com/api/v2/talkwalker/p/<project_id>/resources?access_token=<access_token>&type=shared
object&object_type=myapp'
```

Instead of using an `access_token` the Talkwalker API can also be used with OAuth 2.0 authentication (see the chapter on `access_tokens` and `OAuth 2.0`).

Rate Limit

This endpoint is limited to 20 calls per minute, the result should be stored.

Get search results and histograms for topics

The Project Search Result API `https://api.talkwalker.com/api/v1/search/p/<project_id>/results` and the the Project Search Histogram API `https://api.talkwalker.com/api/v1/search/p/<project_id>/histogram` can be used with the same parameters as the normal Search Result API and the Search Histogram API. Additionally to search a specific topic of a Talkwalker Project, set the parameter `topic` to one or more topic-IDs.

Modifiying documents with the Talkwalker API

Single Documents

To change result documents, use the `https://api.talkwalker.com/api/v2/search/p/<project_id>/<operation>` endpoint. Creating new documents can be done on the `create` operation, updating documents is done with the `update` operation. Deletion and un-deletion of documents can be done on the `delete` and `undelete` operations respectively.

The fields `url`, `published`, and `content` are required. When left empty, some fields (for example `sourcetype`, `posttype` and `language`) will be filled automatically with default values or automatically extracted values.

Examples:

Create

```
curl -XPOST 'https://api.talkwalker.com/api/v2/docs/p/<project_id>/create?access_token=<access_token>' -d '
{
  "url" : "http://www.example.com/docs/doc1.html",
  "title" : "This is a title",
  "content" : "Example content. Really not that much.",
  "tags_marking" : "read",
  "published" : "1430136532000"
}' -H 'Content-Type: application/json; charset=UTF-8'
```

Update

Setting a new title field, adding an `important` tag, and removing the `read` tag:

```
curl -XPOST 'https://api.talkwalker.com/api/v2/docs/p/<project_id>/update?access_token=<access_token>' -d '
{
  "url" : "http://www.example.com/docs/doc1.html",
  "title" : "This is a new title",
  "content" : "Example content. Really not that much.",
  "+tags_marking" : ["important"],
  "-tags_marking" : ["read"],
  "extra_author_attributes" : {
    "name" : null
  },
  "published" : "1430136532000"
}' -H 'Content-Type: application/json; charset=UTF-8'
```

Fields that are of type array, can be updated in three ways: using `"<fieldname>"` to replace the whole array, `"+<fieldname>"` to add an item to the array, and `"-<fieldname>"` to remove an item. Fields can be cleared by explicitly setting them `null`.

Delete

Deleting a document:

```
curl -XPOST 'https://api.talkwalker.com/api/v2/docs/p/<project_id>/delete?access_token=<access_token>' -d '
{
  "url" : "http://www.example.com/docs/doc1.html"
}' -H 'Content-Type: application/json; charset=UTF-8'
```

Undelete

Deleting a document:

```
curl -XPOST 'https://api.talkwalker.com/api/v2/docs/p/<project_id>/undelete?access_token=<access_token>' -d '
{
  "url" : "http://www.example.com/docs/doc1.html"
}' -H 'Content-Type: application/json; charset=UTF-8'
```

Multiple Documents

Multiple documents can be manipulated using the `https://api.talkwalker.com/api/v2/search/p/<project_id>` endpoint. The execution order of the given document operations is not guaranteed (multiple operations on a single document in a single request should be avoided).

```
curl -XPOST 'https://api.talkwalker.com/api/v2/docs/p/<project_id>/delete?access_token=<access_token>' -d '
[{
  "create": {
    "url": "http://www.example.com/docs/doc1.html",
    "title" : "This is the title of doc 1",
    "content" : "and this is the content of doc 1",
  }
}, {
  "update": {
    "url": "http://www.example.com/docs/doc2.html",
    "title" : "This is the title of doc 2",
    "content" : "and this is the content of doc 2",
  }
}, {
  "delete": {
    "url": "http://www.example.com/docs/doc3.html"
  }
}]' -H 'Content-Type: application/json; charset=UTF-8'
```

Parameters

parameter	description	required?	values
`access_token`	a read/write token specified in the API application	required	

parameter	description	required?	values
`return_entry`	Specifies if the modified document should be returned	optional	hide (default), show

See Talkwalker Documents

Talkwalker Streaming API

Source

```
https://api.talkwalker.com/api/v2/stream
```

How it works

The Talkwalker Streaming API delivers real-time data through a persistent connection to our servers. Configure your stream with a set of filtering rules, connect to the stream and new results will be delivered in real time, as soon as they are found by our crawlers. You will not need to do any polling to receive new data.

You setup and configure the Streaming API by defining rules (Boolean query, language, media types, etc.). The Streaming API then finds and collects all relevant data and adds it to your data stream, with individually highlighted snippets per matched rule. This feature allows you to gather data from many rules through a single stream while easily matching the results back to your predefined rules.

Each rule allows filtering by title, content, author, language, URL, country, media type, and more parameters, using the same syntax as in our Talkwalker Search interface. You can also apply a list of sources to be included or excluded from the stream, to give you even further possibilities to narrow down the results you will get. A single rule can support up to 50 operands. To create complex rules, operands may be combined using Boolean Operators.

The documents are streamed in the order they are found by our crawlers and added to Talkwalker (i.e. by `search_indexed` timestamp). Custom sorting is not possible with the Streaming API (however this can be done with the Search API). The documents are grouped in timeframes which contain all documents that were indexed between the given start and end time of the timeframe.

Each result (independent on how many rules match) will be counted as 1 credit.

Stream Format

Stream

A Stream, its rules, queries and panels are represented by the following json object. `stream_id`, `rule_id` and `panel_id` are used to reference streams, rules and panels and have to be unique within a project. `stream_id` and `rule_id` are also used in the results to specify which rule or stream matched a result.

Example:

```
{
  "stream_id" : "teststream",
  "rules" : [{
    "rule_id": "rule-1",
      "query": "cats"
  },{
    "rule_id": "teststream-dogs-toppanel",
    "query": "dogs",
    "panel": {
      "referenced_panel":["toppanel"]
    }
  }]
}
```

Stream ids, rule ids, panel ids, etc can only contain lowercase letters, numbers and the characters `-` and `_`. They have to start with a lower case letter.

json fields

parameter	description	required?	default value
`stream_id`	id we want to reference this stream with	required	
`rules`	a set of rules for this stream	optional	

A set of rules can be either an array of strings to be matched or for a more advance usage a rule is defined as the following object:

parameter	description	required?	default value
`rule_id`	id we want to reference this rule with (will also be returned when the rule matched)	optional	
`query`	a query defining this rule	optional*	
`panel.referenced_panel`	a set of panels that are being applied to this rule	optional*	
`panel.matching`	matching can be 'all' or 'any' (if doc needs to be in all panels or in a single panel)	optional	any

*Note: either a query or a panel must be set

The Talkwalker API returns a sequence of chunks, **in version 2 (`/v2/stream`) the format of the sequence has been changed, chunks are delivered in a flat list, separated by newline characters (`\r\n`)**. Each chunk contains a document or stream information. Result documents have `"chunk_type" : "CT_RESULT"`, `CT_CONTROL` identifies control chunks (containing information about the next result chunks) and `CT_ERROR` identifies error message chunks.

Result Chunk

```
{
  "chunk_type" : "CT_RESULT",
  "chunk_result" : {
    "data" : {
      "data" : { <default result data (see simple search)> },
      "highlighted_data" : [ {
        "title_snippet" : "<title snippet for rule>",
        "content_snippet" : "<content snippet for rule>",
        "matched":{
          "rule_id" : "rule1",
          "stream_id" : "stream2",
          "panel_id : ["panel1","panel2"],
          "rule_query : "cats AND dods"  // if rule_id is not set on rule
        }
      }]
    }
  }
}
```

Control Chunk

```
{
  "chunk_type" : "CT_CONTROL",
  "chunk_control" : {
    "timeframe_start" : <start time>,
    "timeframe_end" : <stop time>
  }
}
```

Error Chunk

```
{
  "chunk_type" : "CT_ERROR",
  "chunk_error" : {
    "status_code" : "<code>",
    "status_message" : "<error message>",
    "data" : [{
        "key" : "errdetail",
        "value" : ["some details"]
      }
    ]
  }
}
```

Credits

Each result (independent on how many rules match) will be counted as 1 credit. If no credits are left, the stream is stopped and a control chunk containing the timestamp of the end of the stream (needed for resuming) is sent. API calls which don’t return any results are not counted. The documents are billed after every completed timeframe, if a stream gets disconnected a non completed timeframe will not be billed. (When resuming a disconnected stream, a partially streamed timeframe has to be restarted and streamed again.) When the parameter `max_hits` is set, only the specified maximum number of results will be billed, even if the entire timeframe gets streamed after reaching the limit.

Order and Timing of Chunks

It is not possible to do any custom sorting with the Talkwalker Streaming API. The data is grouped in unsorted timeframes, which will be returned in the order the data was added to Talkwalker. (This can be a different order than the order the data was published in.)

The number of results chunks in a timeframe is not limited! When implementing a client application, store or process the results in a reasonable batch size (to limit memory usage and prevent out of memory) and do not wait for a completed timeframe.

Stream Results

To start streaming the results from a stream at least one rule needs to be defined. The results are available at `https://api.talkwalker.com/api/v/stream/s/<stream_id>/results`.

Example: Start a stream:

```
curl https://api.talkwalker.com/api/v2/stream/s/teststream/results?access_token=demo
```

Example: Resume a disconnected stream: Set the parameter `stream_resume` to the start timestamp ('timeframe_start') of the last `CT_CONTROL` chunk. Since the results in a timeframe are not sorted, the streaming of the entire timeframe has to be restarted.

```
curl
https://api.talkwalker.com/api/v2/stream/s/teststream/results?access_token=demo&stream_resume=1388534400000
```

Parameters

parameter	description	required?	default value
`access_token`	a read/write token specified in the API application	required	
`q`	The query to search for.	optional	
`stream_resume`	Resumes the stream from this starting point	optional	now
`stream_stop`	Stops the stream at this point	optional	
`max_hits`	Stops the stream after the given number of hits	optional	

`stream_stop` can be used to specify an end timestamp for the stream. When the number of documents in `max_hits` is reached, the remaining documents of the timeframe are still streamed but not billed. After this, a control chunk containing the timestamp needed to resume the stream is send.

Multiple stream ids

To stream results of multiple streams through one single connection, all of the streaming endpoints accept multiple streams in the `/s/<stream_id>` parameter. The following syntax can be used:

	example	description
single	`test-stream`	a single stream
multiple	`test1,test2,test3`	a list of streams
prefix	`test*`	every stream that starts with test
all	`*`	all defined streams
exclude	`test*,-test1`	every stream that stats with test except test1

While streaming the matched streams are expanded on the start of every chunk, so that new streams get picked up automatically on a running connection. Streaming will fail in case no stream matches the multiple streams description (anymore).

Stream ids, rule ids and panel ids all must be unique within the project.

Rate Limit

This endpoint is limited to 5 calls per minute. Only one connection can be opened, if multiple streams were defined, they must be streamed through one single connection (see above how to select multiple streams).

Managing Streams

Stream Create and Stream Definition

Creating a new Stream and getting the definition of a stream are done on the `https://api.talkwalker.com/api/v2/stream/s/<streamid>` endpoint, using the methods PUT and GET.

Parameters

Endpoint parameters:

parameter	description	required?	default value
`access_token`	a read/write token specified in the API application	required	

Example: create a new stream

```
{
  "stream_id" : "teststream",
  "rules" : [{
    "rule_id": "rule-1",
    "query": "cats"
  }]
}
```

Command:

```
curl -XPUT https://api.talkwalker.com/api/v2/stream/s/teststream?access_token=demo -d '{ "rules" : [{
"rule_id": "rule-1", "query": "cats" }] }' -H 'Content-Type: application/json; charset=UTF-8'
```

Response:

```
{
  "status_code" : "0",
  "status_message" : "OK",
  "request" : "PUT /api/v2/stream/s/teststream?access_token=demo",
  "result_stream" : {
    "data" : [{
      "stream_id" : "teststream",
      "rules" : [{
        "rule_id" : "rule-1",
        "query" : "cats"
      }]
    }]
  }
}
```

Example: get the stream `teststream`

```
curl -XGET https://api.talkwalker.com/api/v2/stream/s/teststream?access_token=demo
```

The response will be the same as before.

Rate Limit

This endpoint is limited to 20 calls per minute.

Stream Delete

The `https://api.talkwalker.com/api/v2/stream/s/<stream_id>` endpoint is used to delete a stream.

Example:

```
curl -XDELETE 'https://api.talkwalker.com/api/v2/stream/s/teststream?access_token=demo&pretty=true'
```

Parameters

parameter	description	required?	default value
`access_token`	a read/write token specified in the API application	required	

Rate Limit

This endpoint is limited to 20 calls per minute.

Stream Info

The `https://api.talkwalker.com/api/v2/stream/info` endpoint returns a list of all Talkwalker API Streams linked to a Talkwalker API access token.

Example:

```
curl 'https://api.talkwalker.com/api/v2/stream/info?access_token=demo&pretty=true'
```

Response:

```
{
  "status_code" : "0",
  "status_message" : "OK",
  "request" : "GET /api/v2/stream/info?access_token=demo",
  "result_streaminfo" : {
    "data" : [{
      "name" : "teststream"
    }]
  }
}
```

Parameters

Endpoint parameters:

parameter	description	required?	default value
`access_token`	a read/write token specified in the API application	required	

Rate Limit

This endpoint is limited to 20 calls per minute, the result should be stored.

Rules

The `https://api.talkwalker.com/api/v2/stream/s/<stream_id>/r/<rule_id>` resource is used to set new rules for an existing stream. Rules are used to filter out unwanted results on a stream. Talkwalker Streaming API rules are specified in the Talkwalker query syntax.

The response only includes the requested, created or deleted rule.

Example:

Add a rule to limit a stream to only German results

```
curl -XPUT https://api.talkwalker.com/api/v2/stream/s/teststream/r/rule-1?access_token=demo -d '
{
  "query":"lang:de"
}'
-H "Content-Type: application/json; charset=UTF-8"
```

Response:

```
{
  "status_code" : "0",
  "status_message" : "OK",
  "request" : "PUT /api/v2/stream/s/teststream/r/rule-1?access_token=demo",
  "result_stream" : {
    "data" : [{
      "stream_id" : "teststream",
      "rules" : [{
        "rule_id" : "rule-1",
        "query" : "lang:de"
      }]
    }]
  }
}
```

Get an existing rule:

```
curl -XGET https://api.talkwalker.com/api/v2/stream/s/teststream/r/rule-1?access_token=demo
```

Delete an existing rule:

```
curl -XDELETE https://api.talkwalker.com/api/v2/stream/s/teststream/r/rule-1?access_token=demo
```

Rules that are not in valid Talkwalker query syntax will be rejected (error `400 - 4 Error in query`), in this case the old rules will not be replaced.

Parameters

Endpoint parameters:

parameter	description	required?	default value
`access_token`	a read/write token specified in the API application	required	

Rate Limit

This endpoint is limited to 20 calls per minute.

Panels

The Panel defines a source set that is considered for streaming. It can contain a whitelist with an include query `include_query` or a blacklist with exclude query `exclude_query`. To create, get or delete a panel use the `https://api.talkwalker.com/api/v2/panel/a/<panel_id>` endpoint. Panels are defined using the Talkwalker query syntax.

Example: Create a the panel with `"include_query" : ["lang:de", "lang:fr"]` and `"exclude_query": ["sourcecountry:lu"]` for the stream `teststream` to restrict the stream to German and French results which are not from Luxembourg.

```
curl -XPUT https://api.talkwalker.com/api/v2/panel/a/testpanel?access_token=demo -d '
{
  "include_query" : [
    "lang:de",
    "lang:fr"
  ],
  "exclude_query" :[
    "sourcecountry:lu"
  ]
}' -H "Content-Type: application/json; charset=UTF-8"
```

Response:

```
{
  "status_code" : "0",
  "status_message" : "OK",
  "request" : "PUT /api/v2/panel/a/testpanel?access_token=demo",
  "result_panel" : {
    "data" : [{
      "panel_id" : "testpanel",
      "include_query" : [
        "lang:de",
        "lang:fr"
      ],
      "exclude_query" : [
        "sourcecountry:lu"
      ]
    }]
  }
}
```

Getting a panel

```
curl -XGET https://api.talkwalker.com/api/v2/panel/a/testpanel?access_token=demo
```

Deleting a panel

Panels that are still referenced may not be deleted.

```
curl -XDELETE https://api.talkwalker.com/api/v2/panel/a/testpanel?access_token=demo
```

Panels that are not in valid Talkwalker query syntax will be rejected (error `400 - 4 Error in query`), in this case the old panels will not be replaced.

Getting a list of all panels

```
curl -XGET https://api.talkwalker.com/api/v2/panel/info?access_token=demo
```

Response:

```
{
  "status_code" : "0",
  "status_message" : "OK",
  "request" : "GET /api/v2/panel/info?access_token=demo",
  "result_panel" : {
    "data" : [ {
      "panel_id" : "panel1"
    }, {
      "panel_id" : "panel2"
    }, {
      "panel_id" : "panel3"
    } ]
  }
}
```

Parameters

Endpoint parameters:

parameter	description	required?	default value
`access_token`	a read/write token specified in the API application	required	

Rate Limit

This endpoint is limited to 20 calls per minute.

Matching of Streams, Rules and Panels

When a document matches a rule, `highlighted_data` is included in the result entry. When multiple rules match a query, `highlight_data` is repeated for every rule that matches.

Example:

```
highlighted_data {
  matched {
    rule_id: "rule-1",
    stream_id: "stream-1",
    panel_id: ["panel-1","panel-2"],
    rule_query: "cats OR dogs" // if rule_id is not set on rule
  }
  title_snippet: "Cats are...",
  content_snippet: "... cats are ..",
}
```

Quota on Streams

A quota can be specified for each stream. This quota allows to limit the number of results delivered through a stream per hour, day or month. After the limit has been reached this stream will be deactivated until the next period begins. The connection will stay open even if the stream, some of the streams or all streams are deactivated. Information about disabled streams is delivered through periodic control chunks.

Example:

```
curl -XPUT https://api.talkwalker.com/api/v2/stream/s/teststream/quota?access_token=demo -d '
{
  "allowance":1000,
  "reset":"daily",
  "timezone":"UTC",
  "reference_time":"2015-01-01T00:00:00.000Z" // or long
}'
```

The reference time defines a reference time in relation to the period and timezone. Its usage depends on period:

period	reference time	"explanation"
`hourly`	beginning of hour + (reference % hour)	minute in hour
`daily`	beginning of day + (reference % day)	hour in day
`weekly`	beginning of week + (reference % week)	day of week
`monthly`	beginning of month + (reference % month)	day of month

Request information about a quota on a stream:

Example:

```
curl -XGET https://api.talkwalker.com/api/v2/stream/s/teststream/quota?access_token=demo
```

Response:

```
{
  "status_code" : "0",
  "status_message" : "OK",
  "request" : "PUT /api/v2/stream/s/teststream/quota?access_token=demo",
  "result_stream" : {
    "data" : [{
      "stream_id" : "teststream",
      "quota" : {
        "allowance" : 10000,
        "reset" : "hourly",
        "timezone" : "UTC",
        "period_start" : "2015-04-27T08:00:00.000Z",
        "period_reset":"2015-04-27T09:00:00.000Z",
        "usage":0,
        "status":"active",
        "reference_time":"2015-01-01T00:00:00.000Z"
      }
    }]
  }
}
```

To remove the quote from a stream:

Example:

```
curl -XDELETE https://api.talkwalker.com/api/v2/stream/s/teststream/quota?access_token=demo
```

A Reset can also be triggered manually if a rule should be reactivated, the usage will then be reset to 0 for the current period:

```
curl -XPOST https://api.talkwalker.com/api/v2/stream/s/<streamid>/quota/reset?access_token=demo
```

If the quota on a stream gets full before the end of a chunk, the data for the current chunk is still fully delivered. Reactivation of a stream occurs at chunk boundaries. Chunk boundaries are aligned with the different reset times.

Additional Information on Quota in Control Chunks

The information delivered through the control-chunk contains the list of streams requested by the connection. It contains the number of results delivered per stream, the remaining quota if applicable, the status of the stream (if it has been deactivated because of the quota). The number of remaining credits on the account can be requested through the credits API.

Control chunks will have the following additional information:

```
{
  "timeframe_start": 1427216400000,
  "timeframe_end": 1427216460000,
  "stream":[ {
    "id":"stream-1",
    "allowance": 10000,
    "usage": 5000,
    "reset": 1427241600000,
    "status":"active"
  } ]
}
```

Temporarily Disable Streams

```
POST https://api.talkwalker.com/api/v2/stream/s/<stream_id>/enable
POST https://api.talkwalker.com/api/v2/stream/s/<stream_id>/disable
```

These endpoints allow to temporarily disable a stream or to eanble it. Disabling a stream has the same effect, as a stream which has reached its quota. Disabled streams are shown in control chunks with `"status" : "disabled"`. New created streams are enabled, while creating you can explicitly specify `"enabled" : true` or `"enabled" : false`.

Talkwalker Streaming API and Talkwalker Projects

```
https://api.talkwalker.com/api/v2/stream/s/<stream_id>/p/<project_id>/results
```

How it works

Talkwalker users can use the topics defined in their project with the Talkwalker API. Topics can be used with the Streaming Results API. To limit the results of a predefined stream to those matching a topic `topic` to that topic's ID (multiple topics can be set). see Talkwalker Search API and Talkwalker Projects

Example: Setup a stream that streams all new data for a Talkwalker Project. You will need your custom API application access token.

To find the Id of your project use:

```
curl 'https://api.talkwalker.com/api/v1/search/info?access_token=<access_token>'
```

To get a list of all topics:

```
curl 'https://api.talkwalker.com/api/v1/search/p/<project_id>/topics/list?access_token=<access_token>'
```

To create the stream:

```
curl -XPUT  'https://api.talkwalker.com/api/v2/stream/s/teststream?access_token=<access_token>' -d
'{"streamid":"teststream"}' -H 'Content-Type: application/json; charset=UTF-8'
```

To start the stream:

```
curl
https://api.talkwalker.com/api/v2/stream/s/teststream/p/<project_id>/results?access_token=<access_token>&topic
=<topic_id_1>&topic=<topic_id_2>
```

See FAQ for more examples

Talkwalker Single Sign-on API

Source

```
https://api.talkwalker.com/api/v2/auth/
```

Note: The Single Sign-on API needs a special access token (of type authentication) and the endpoints must be called via a secure connection (`HTTPS`).

Talkwalker Login Url

```
curl 'https://api.talkwalker.com/api/v2/auth/u/<user_id>/loginurl?access_token=<access_token>'
```

The Talkwalker Single Sign-on API is used to retrieve a single sign on URL for a Talkwalker account or application. To get such an URL, the `/loginurl` endpoint is used,the returned login URL is only valid for 10 seconds. The alternative endpoint `api.talkwalker.com/api/v2/auth/loginurl?access_token=<access_token>` can be used to login without specifying a user, the returned login url will authenticate as the account administrator.

Parameters

parameter	description	required?	default value
`access_token`	Authentication access token	required	
`project_id`	ID of a Talkwalker project	required	
`page`	Menu page that will be opened on login	optional	home_screen
`view`	View that will be shown on login	optional	home_screen
`logout_url`	Url the user will be redirected to on logout	optional	default login page

parameter	description	required?	default value
`token_timeout`	Timeout for the generated login token	optional	10s
`pretty`	Formatted json for testing	optional	false

`token_timeout` accepts values in minutes or seconds (for example `5s` or `1m`) with a maximum time of `30m`.

Either `page` can be set (`monitor`, `dashboard` or `home_screen`) to lead the user to a generic menu or `view` can be set to lead to a specific stored view. To get a list of all views see below.

Example:

```
https://api.talkwalker.com/api/v2/auth/u/<user_id>/loginurl?access_token=<access_token>&pretty=true
```

Result:

```
{
  "status_code" : "0",
  "status_message" : "OK",
  "request" : "GET /api/v2/auth/u/<user_id>/loginurl?access_token=<access_token>&pretty=true",
  "result_loginurl" : {
    "single_sign_on_url" : "/app/login?login_token=<token>&user_id=<user_id>",
    "user_id" : "<user_id>",
    "expiration_date" : 1423064059056
  }
}
```

Logout

```
curl 'https://api.talkwalker.com/api/v2/auth/u/<user_id>/logout?access_token=<access_token>'
```

The `/logout`-endpoint is used to log a user out from talkwalker and to invalidate all tokens that were created for this user. All sessions for this user (either authenticated with a single sign on URL, or with a password) will be closed.

User List

```
curl 'https://api.talkwalker.com/api/v2/auth/users?access_token=<access_token>'
```

This endpoint returns a list of all the users in an account and the projects they have access to.

Example:

```
https://api.talkwalker.com/api/v2/auth/users?access_token=<access_token>&pretty=true
```

Result:

```
{
  "status_code" : "0",
  "status_message" : "OK",
  "request" : "GET /api/v2/auth/users?access_token=<access_token>&pretty=true",
  "result_users" : {
    "user" : [ {
      "user_name" : "Admin 1",
      "user_email" : user_1@site.com",
      "user_id" : "user_id_1",
      "project" : [ {
        "project_id" : "project_id_1",
        "project_name" : "Project 1",
        "account_id" : "account_id_1",
        "account_name : "account_name_1",
        "access_level" : "ACCOUNT_ADMIN"
      }, {
        "project_id" : "project_id_2",
        "project_name" : "Project 2",
        "account_id" : "account_id_1",
        "account_name : "account_name_1",
        "access_level" : "ACCOUNT_ADMIN"
      }, {
        "project_id" : "project_id_3",
        "project_name" : "Project 3",
        "account_id" : "account_id_1",
        "account_name : "account_name_1",
        "access_level" : "ACCOUNT_ADMIN"
      } ]
    }, {
      "user_name" : "User 2",
      "user_email" : user_2@site.com",
      "user_id" : "user_id_2",
      "project" : [ {
        "project_id" : "project_id_2",
        "project_name" : "Project 2",
        "account_id" : "account_id_1",
        "account_name : "account_name_1",
        "access_level" : "FULL_TOOL"
      } ]
    } ]
  }
}
```

Project List

```
curl 'https://api.talkwalker.com/api/v2/auth/projects?access_token=<access_token>'
```

This endpoint returns a list of all the projects in an account and the users that have access.

Example:

```
https://api.talkwalker.com/api/v2/auth/projects?access_token=<access_token>&pretty=true
```

Result:

```
{
  "status_code" : "0",
  "status_message" : "OK",
  "request" : "GET /api/v2/auth/projects?access_token=<access_token>&pretty=true",
  "result_projects" : {
    "project" : [ {
      "project_id" : "project_id_1",
      "project_name" : "Project 1",
      "account_id" : "account_id_1",
      "account_name : "account_name_1",
      "user" : [ {
        "user_id" : "user_id_1",
        "user_name" : "Admin 1",
        "user_email" : user_1@site.com",
        "access_level" : "ACCOUNT_ADMIN"
      } ]
    }, {
      "project_id" : "project_id_2",
      "project_name" : "Project 2",
      "account_id" : "account_id_1",
      "account_name : "account_name_1",
      "user" : [ {
        "user_id" : "user_id_1",
        "user_name" : "Admin 1",
        "user_email" : user_1@site.com",
        "access_level" : "ACCOUNT_ADMIN"
      }, {
        "user_id" : "user_id_2",
        "user_name" : "User 2",
        "user_email" : user_2@site.com",
        "access_level" : "FULL_TOOL"
      } ]
    }, {
      "project_id" : "project_id_3",
      "project_name" : "Project 3",
      "account_id" : "account_id_1",
      "account_name : "account_name_1",
      "user" : [ {
        "user_id" : "user_id_1",
        "user_name" : "Admin 1",
        "user_email" : user_1@site.com",
        "access_level" : "ACCOUNT_ADMIN"
      } ]
    } ]
  }
}
```

View List

```
curl 'https://api.talkwalker.com/api/v2/talkwalker/p/<project_id>/views?access_token=<access_token>'
```

This endpoint returns a list of all the views in a project. Note: This endpoint is part of the Talkwalker Project API and needs a `read_write` access token.

Result:

```
{
  "status_code" : "0",
  "status_message" : "OK",
  "request" : "GET /api/v2/talkwalker/p/<project_id>/views?access_token=<access_token>&pretty=true",
  "result_views" : {
    "projects" : [ {
      "id" : "<project_id>",
      "title" : "Project 1",
      "dashboards" : [ {
        "id" : "id1",
        "title" : "Dashboard 1"
      }, {
        "id" : "id2",
        "title" : "Dashboard 2"
      }, {
        "id" : "id3",
        "title" : "Dashboard 3"
      }, {
        "id" : "id4",
        "title" : "Dashboard 4"
      }, {
        "id" : "id5",
        "title" : "Dashboard 5"
      }, {
        "id" : "id6",
        "title" : "Dashboard 6"
      } ]
    } ]
  }
}
== Talkwalker Channelmonitoring API
```

Channelmonitoring suggest

This provides the same functionality as the pagemonitoring suggest in the talkwalker. Given a string (url, name, ...) and a type (default = auto), it will provide several candidates.

Command:

```
curl -XGET
https://api.talkwalker.com/api/v2/talkwalker/p/<projectid>/monitoring/suggest?input=<url/string>&type=auto&acc
ess_token=<access_token>
```

Response

```
{
  "status_code" : "0",
  "status_message" : "OK",
  "request" : "...",
  "result_monitoring_pages" : {
    "data" : [ {
      "title" : "ABC",
      "type" : "facebook-page",
      "access_url" : "http://facebook.com/296043200790",
      "query" :
"channel:\"vtwqablxreaaaacgbieemqkdivbe6t2lcicgmzlfmqnci2duorydulzpo53xonf4zdsnrqgqztembqg44ta\""
    }, ...  ]
  }
}
```

Fetch query

Input: the access_url and the site monitoring type

Output: query to be used in stream

Command:

```
curl -XGET https://api.talkwalker.com/api/v2/talkwalker/p/<project_id>/monitoring/fetch?type=twitter-
user&access_url=http%%3A%%2F%%2Ftwitter.com%%2Flufthansa&access_token=<access_token>
```

Response

```
{
  "status_code" : "0",
  "status_message" :  "OK",
  "request" : "GET /api/v2/talkwalker/p/<project_id>/monitoring/fetch?type=twitter-
user&access_url=http%%3A%%2F%%2Ftwitter.com%%2Flufthansa&access_token=<access_token>",
  "result_monitoring_pages" : {
    "data" : [{
      "title" : "Lufthansa",
      "type" : "twitter-user",
      "access_url" : "http://twitter.com/lufthansa",
      "query" :
"channel:\"vtwqablxreaaaacgbieemqkdivbe6t2lcicgmzlfmqnci2duorydulzpo53xonf4zdsnrqgqztembqg44ta\""
    }]
  }
}
```

Talkwalker Query Syntax

A single search query can support up to 50 operands and be up to 1024 characters long in length. To create complex queries, operands may be combined using Boolean Operators.

All queries are executed in their unaccented and case insensitive form, thus a search for "éléVE" will also match all documents with the word "eleve". No language stemming is being done, thus a search for the "children" won’t return results with the word "child".

Special Transformations

These transformations apply when a query contains no operators from the query syntax (quotes, `AND`, `OR`, wildcards etc, see below).

Words with only capital letters (and special chars `+-&`) are executed as exact (case sensitive) raw data search (`ABC` = `++"ABC"`, `A&B` = `++"A&B"`).

Screen names (`@name`), hashtags (`#hashtag`), cashtags (`$cashtag`) as well as words containing a dash (`-`), a plus (`+`) or an ampersand (`&`) are executed as (case insensitive) raw data search (`@username` = `+"@username"` , `p&t` = `+"p&t"`).

If a query contains multiple simple words (no special characters like (`#@+-&`), no operators and is not only capital letters, it is executed as a proximity search. The maximum number of jumps is set to (#words - 1) * 10 (`cat dog mouse bird` = `"cat dog mouse bird"~30`).

To prevent this behaviour use the explicit query syntax below. (instead of `cat dog mouse` use `cat OR dog OR mouse`, `cat AND dog AND mouse` or `"cat dog mouse"` to search for one of the words, all the words or the exact phrase.

Boolean Operators

AND	`AND` combines two keywords: `BMW AND bike` will find all entries which mention the keyword BMW and the keyword bike.	`BMW AND bike`
AND NOT	`AND NOT` excludes a word of an entry: `BMW AND NOT` bike will find all entries with the keyword BMW, but only if the notion bike is not contained in the same article.	`BMW AND NOT bike`
OR	`OR` means that a least one of the terms which are linked by an OR have to be mentioned in the same article: `BMW OR bike` will find all entries that include either the keyword BMW or the keyword bike.	`BMW AND NOT bike BMW OR bike`
Exclusion of Keywords	Negative filters can be created by using the operator `NOT`.	`NOT coupons`
Phrase Search	Quotes `""` are used for finding keyword sequences: `"BMW series"` will find all entries which contain the phrase "BMW series". In contrast the search query `BMW AND series` does not respect the order.	`"bmw series"`
Combinations	Brackets `()` are used to group several keywords in a way that operators can be applied on multiple terms within the brackets (distributive law). `BMW AND (motorcycle OR car)` is a shortform for `(BMW AND motorcycle) OR (BMW AND car)`	`BMW AND (motorcycle OR car)`
Wildcard Search	The Wildcard operator `*` is a character that stands for 0 or any possible number. Wildcards are only accepted at the end of a keyword: `Luxemb*` will find all entries including keyworks like Luxembourg , Luxemburg, Luxemburgisch or any other keyword with the prefix Luxemb.	`Luxemb*`
Wildcard Search – one character	The question mark `?` has a similar function as the wildcard operator, but only replaces exactly one character, i.e. it is useful in consideration of British and American English, e.g.: `reali?ation` finds realisation but also realization.	`reali?ation`
Proximity Search	The tilde symbol `~` analyses the surroundings of a character string which is enclosed in quotes (consisting at least two words). You cannot combine the tilde with the wildcard operator. e.g. `"obama merkel"~5` finds "A statement released from the White House said Obama, Monti and Merkel agreed on certain steps" (3 jumps between both words), `"obama merkel"~5` finds every entry, containing the keywords obama and merkel within an interval of maximum of 5 jumps.	`"obama merkel"~5`
Fuzzy X Search	The tilde symbol `~X` after a word searches for words similar to the given word. The value after the tilde (0, 1 or 2) defines the number of changed characters. `roam~1` will also find foam.	`roam~1`
Fuzzy Search	The tilde symbol `~` after a word will find this word as a two part word with a hyphen, space or other special character in it. `carsharing~` will find `carsharing`, `car-sharing`, `car sharing` etc	`carsharing~`
Raw Data Search	A simple `+` in front of a keyword samples an exact character string including special characters and punctuation, it does not consider lower and upper cases. It also works with brackets and tilde: `+"l'or  al"` or `+"d&g"` etc	`+"l'oréal"`
Exact Raw Data Search	Two `++` in front of a keyword samples an exact character string including special characters and punctuation, it does consider lower and upper cases. It also works with brackets and tilde: `++"L'Oréal"`	`++"L'Oréal"`
NEAR/x	The `NEAR/x` operator works similar to the proximity search operator, but also works with parentheses and thus can be used with multiple terms. (default value for x: 15)	`(BMW OR Audi) NEAR/3 (motorcycle OR car)`
ONEAR/x	Same as `NEAR/x` but respects the order of terms.	`(BMW OR Audi) ONEAR/3 (motorcycle OR car)`

Sentence Search	The SENTENCE operator works similar to the NEAR/x operator. It searches for keywords that appear in the same sentence. SENTENCE can also be used with multiple terms.	`(BMW OR Audi) SENTENCE (motorcycle OR car)`
Ordered Sentence Search	Same as `SENTENCE` but respects the order of terms in the sentence.	`(BMW OR Audi) OSENTENCE (motorcycle OR car)`

Note:
In phrase search and raw data phrase search (`""` or `+""`) the number and type white space characters are ignored. For example `"BMW series"` (one space) will also match documents which contain `"BMW  series"` (two spaces) and vice versa.

White space characters include spaces, tabs and new line characters, also transitions between letters and special characters are considered as whitespace. For example `+"P&T"` will match `P&T` but also `P& T` and `P & T`.

Advanced Search Options:

Single Keyword Search	Search for simple brands, products, keywords, etc.	`Apple`
Title Search	It searches within the title of an article. `title:sixt` will find all results which contain the keyword sixt within the title. `title:"obama merkel"~5` matches with: Obama Seeking Ally Finds Merkel a Tough Sell	`title:sixt` `title:"obama merkel"~5`
Content Search	It searches within the article `content:sixt` will find all results which mention the keyword within the main text of the article.	`content:sixt`
Author Search	It searches for authors of articles. `author:Franz` will find all results containing articles which defined Franz as author.	`author:Franz`
Language Search	It searches for languages of articles. `lang:de` only indicates German results.	`lang:de`
Source Country Restriction	It searches for the country of origin of sources. `sourcecountry:de` filters all articles from German sources and which were published in Germany.	`sourcecountry:de`
Author Country Restriction	If the author is in a specifiy country when writing posts, `authorcountry:fr` limits results to ones from French authors.	`authorcountry:de`
Source Type Restriction	`sourcetype:"BLOG"` restricts results to a specific media/source type. Returns only BLOG entries.	`sourcetype:BLOG`
Comments Search	Find only comments by setting `is:comment` or without comments (`-is:comment`)	`is:comment`
Retweets Search	Find only retweets with `is:retweet` or exclude retweets with `-is:retweet` and get only original posts	`is:retweet`
Twitter Reply Search	Find only tweets that are replies to other tweets	`is:twitter_reply`
Questions Search	Search for questions. `is:question` will find only documents that are questions.	`is:question`
Image Search	`contains:image` returns those documents that include images	`contains:image`
Audio Search	`contains:audio` returns those documents that include audio	`contains:audio`
Video Search	`contains:video` returns those documents that include videos	`contains:video`

Talkwalker Tags Search	`is:important` finds all documents that were manually tagged as important in Talkwalker. `is:read` finds documents that were read (original document link opened). `is:checked` finds documents were the sentiment has been checked manually in the project	`is:important, is:read, is:checked`
Score Search	`score:n` finds all documents that were manually tagged with the respective score. (In a Talkwalker project scores can be added to a selected document by pressing the number keys)	`score:4`
Post Type Search	`posttype:IMAGE` allows to search only for documents of type image. Possible values are TEXT LINK IMAGE VIDEO AUDIO.	`posttype:LINK`

Url based Search

Url Search	`url:` returns the document with this exact url. Prefix Wildcard (e.g. *apple) matching is not supported.	`url:http://twitter.com/bmw/status/561925861155561473`
Parent Url Search	`parenturl:` returns all child documents (comments or retweets) from a document specifed by the given url. E.g. Give me all the comments for this document url.	`parenturl:http://twitter.com/bmw/status/561925861155561473`
Host Url Restriction	`hosturl:"www.spiegel.de"` returns all the documents from the host www.spiegel.de	`hosturl:"http://www.spiegel.de/"`
Domain Url Restriction	`domainurl:spiegel.de"` returns all the documents from the domain Spiegel.de. Pay attention not to insert www. into the query	`domainurl:"http://spiegel.de/"`
Site Search	`site:twitter.com/bmw/` returns all documents from the site `twitter.com/bmw/`. `site:googleblog.blogspot.com` returns documents from `googleblog.blogspot.com`. Pay attention to end with a `/` if the site includes a specific path (`/bmw/`) but not if it ends with the top level domain (`.com`)	`site:googleblog.blogspot.com` `site:blogspot.com` `site:twitter.com/bmw/`
In Urls Search	`inurls:facebook` returns all documents which have the keyword `facebook` **anywhere** in their url, or which have it in any **referenced url** in the content.	`inurls:facebook`

Metric (Minimum / Maximum) Restrictions

`metric_name:>n` , `metric_name:<n` and `metric_name:n` return only documents which match a specific value or range of a metric. Following tables explains the possible metrics

metric_name	Description	Example
`reach`	The reach of an article/post represents the number of people who were reached by this article/post.	`reach:>100`
`engagement`	The engagement of an article/post is the sum of actions made by others on that article/post.	`engagement:<1000`
`facebook_shares`	Number of Facebook share an article has	`facebook_shares:0`
`facebook_likes`	Number of Facebook likes an article has	`facebook_likes:>0`
`twitter_retweets`	Number of Twitter retweets an article has	`twitter_retweets:>1000`
`twitter_shares`	Number of Twitter share an article has	`twitter_shares:0`

metric_name	Description	Example
`twitter_followers`	Number of Twitter followers a source has	`twitter_followers:>1000`
`youtube_views`	Number of YouTube views a video has	`youtube_views:>100000`
`youtube_likes`	Number of YouTube likes a video has	`youtube_likes:>100`
`youtube_dislikes`	Number of YouTube dislikes a video has	`youtube_dislikes:>0`
`instagram_likes`	Number of Instagram likes a post has	`instagram_likes:>0`
`instagram_followers`	Number of Instagram followers a post has	`instagram_followers:>100`
`comment_count`	Number of Comments an article has	`comment_count:>0`
`published`	Timestamp of publication (epoch time in milliseconds)	`published:>1420731027000`
`searchindexed`	Timestamp of indexation in Talkwalker (epoch time in milliseconds)	`searchindexed:>1420731027000`
`sample`	Get a random sample of the results (percent of the total number of results i.e. setting 25 will return one of four the documents) values:1-100	`sample:25`
`sample_million`	Similar to sample_percent, with higher precision (i.e. setting 2000 will return one of 500 documents) values:1-1000000	`sample_million:2000`
`sentiment`	The detected sentiment of the article (values -5 (negative) to 5 (prositive)). `sentiment:positive`, `sentiment:negative` and `sentiment:neutral` map to the respective sentiment ranges of Talkwalker	`sentiment:>0` `sentiment:negative`

Geographic Restrictions

Note: Some documents have precise geographic data in form of GPS measured coordinates provided by the source. For other documents this data is based on source metadata, with a certain precision level. These levels (ordered from lowest precision to highest) are: `country`, `region` and `city` (extracted data) and `coordinates` (exact data).

The coordinates for lower precision geographic data are equal to their capital.

Restriction	Description	Example
`sourcegeo`	Restricts the results to a rectangular geographic area defined by the coordinates (`latitude,longitude`) of the upper left and lower right corner.	`sourcegeo:50.3,5.7;49.4,6.5`
`sourcegeo_resolution`	Restricts to documents that have a minimum precision level of location data. Possible levels are `coordinates`, `city`, `region` and `country`. default: all documents	`sourcegeo_resolution:coordinates`

Example: Search for documents that are in a box that roughly corresponds to Luxembourg and have exact coordinates.

Luxembourg’s north end is at around `50.3°`, south is at `49.4°`, west at `5.7°` and east at `6.5°`, the upper left corner is `50.3,5.7` the lower right corner is `49.4,6.5`. The final query is : `sourcegeo:50.3,5.7;49.4,6.5 AND sourcegeo_resolution:coordinates`.

Special Query Modifiers

All queries are executed in their unaccented and case insensitive form on the content and the title of documents. To change this behaviour, use `flag:<modifier_name>` to enable special query modes.

Modifier Name	Description	Example
`matchinurls`	Query will also match URLs and links.	`flag:matchinurls`
`matchauthor`	Query will also match author field	`flag:matchauthor`
`matchexact`	Use Raw data search as default. All keywords are considered as case-insensitive exact character string including special characters and punctuation.	`flag:matchexact`
`matchexactcase`	Use Exact raw data search as default. All keywords are considered as case-insensitive exact character string including special characters and punctuation.	`flag:matchexactcase`
`matchfuzzywords`	Use Fuzzy Search as default. All keywords will also match combined words `carsharing` will match words like `carsharing`, `car-sharing` or `car sharing`.	`flag:matchfuzzywords`

The special modifiers can be combined: `carsharing flag:matchauthor flag:matchfuzzywords` searches for words like carsharing, car sharing or car-sharing in the fields `title`, `content` and `author_name`.

Note: When `matchinurls` or `matchauthor` is set, API results will not have highlighting in snippets when one of these fields is matched. == Talkwalker Documents

Fields

field name	name	Write access through API	Comment	Possible field values
`url`	URL	y¹	Normalized URL of the article	Unique Url, for example: http://blog.talkwalker.com/en/how-to-export-data-from-talkwalker/
`matched_query`	Matched Query	n	Query which matched. On streaming, this information is present in extra entrydata.	
`matched_profile`	Matched Profile	n	Profile/Rule which matched. On streaming, this information is present in extra entrydata.	
`indexed`	Indexed	n	When article was added to Talkwalker System	Java Timestamp, for example: 1392821902000
`search_indexed`	Search Indexed	n	When article was indexed by Talkwalker search system after postprocessing	Java Timestamp, for example: 1392821902000
`published`	Published	y	When article was published	Java Timestamp, for example: 1392821902000
`title`	Title	y	Text version of the source title	
`content`	Content	y²	Text version of the content	

field name	name	Write access through API	Comment	Possible field values
`title_snippet`	Title Snippet	n	If a match occurred in the title, this field will contain the snippet related to the query set in the datafeed. On streaming, this information is present in extra entrydata.	
`content_snippet`	Content Snippet	n	If a match occurred in the article, this field will contain the snippet related to the query set in the datafeed. On streaming, this information is present in extra entrydata.	
`root_url`	Root URL	n	Url of the subsection of the site where article was posted on.	Example: www.zeit.de/blogs/
`domain_url`	Domain URL	n	Url of the domain where article was posted on	Example: zeit.de
`host_url`	Host URL	n	Url of the host where article was posted on	Example: www.zeit.de
`parent_url`	Parent URL	n	Url of the parent of the article. This is the post this url is refering to, e.g. in case of a comment the main article, in case of a message board post, the main post in the thread	
`lang`	Language of the Article	y	The language of the article	
`porn_level`	Pornography Level	y[3]	Statistical Calculation of the pornographic Level	0-100. 100 = Pornographic Content
`fluency_level`	Fluency Level	y	Statistical Calculation of the fluency level (Data Range: 0-100). The Fluency Level of an article if low if the article is composed of stacked words without punctuation marks.	0-100. 100="Normal" Text
`spam_level`	Spam Level	y	Spam level of the source.	0-100. 100="Spam", > 50 can be considered as spam
`sentiment`	Sentiment	y	Sentiment of text. Negative, neutral or positive	-5, -4, -3, -2, -1, 0, 1, 2, 3, 4, 5. (-5 being negative and 5 being positive)
`source_type`	Source Type	y[2]	Source type of the post. Source type can be any string and be user defined	ONLINENEWS, BLOG ... default: OTHER

field name	name	Write access through API	Comment	Possible field values
`post_type`	Post Type	y	Type of the post. If it's a text post, an image post, video post or anything else.	default: TEXT
`cluster_id`	Cluster Id	n	Url of main cluster entry. Will group identical/similar stories from multiple sources together	
`meta_cluster_id`	Meta Cluster Id	n	Url of main cluster entry. Will group identical/similar clusters together	
`tags_internal`	Internal Tags 1	n	Only in Talkwalker project. Tags used internally. E.g. automatically set tags	
`tags_marking`	Internal Tags 2	y	Only in Talkwalker project. Tags used internally. E.g. automatically set tags	important, read, checked, replied
`tags_customer`	Customer Tags	y	Only in Talkwalker project. Tags added by users of Talkwalker	
`tags_plugin`	Plugin tags	y	Only in Talkwalker project. Tags added by plugins in Talkwalker	

See the chapter on Protocols, Encodings and Value Field Options for possible values for the fields `sourcetype`, `lang`, or `country_code`.

[1] Can not be changed after creating a new document.
[2] Must not be null or empty.
[3] Extracted automatically when left empty.

Content

Talkwalker provides result snippets for all content. In all cases, the `content` field only contains the first words of the document, in addition, we provide the part of the document which matches the query in the `content_snippet` field. In the Streaming API a snippet is provided for every matching rule.

URLs

To filter on specific websites in a query, the fields `domain_url` and `host_url` can be used. `host_url` is used for specific hosts like `www.talkwalker.com` or `blog.talkwalker.com`, while `domain_url` would filter on all host in a specific domain (i.e. `domain_url:blog.talkwalker.com` would return all results of the domain `talkwalker.com` also those from `www.talkwalker.com` while `host_url:blog.talkwalker.com` would return only results from `blog.talkwalker.com` not from `www.talkwalker.com`).

Sentiment

Talkwalker uses natural language processing (NLP) to compute a general sentiment for the documents in our index. The accuracy of automatic detection is limited by irony, sarcasm and misspellings in the documents. Sentiment analysis is available for:

Language	Language Code	Language	Language Code
`Albanian`	sq	`Hungarian`	hu
`Arabic`	ar	`Italian`	it
`Chinese`	zh_cn, zh_tw	`Korean`	ko
`Croatian`	hr	`Malay`	ms
`Czech`	cs	`Norwegian`	no
`Danish`	da	`Polish`	pl
`Dutch`	nl	`Portuguese`	pt
`English`	en	`Russian`	ru
`Finnish`	fi	`Slovak`	sk
`Flemish`	nl	`Spanish`	es
`French`	fr	`Swedish`	sv
`German`	de	`Turkish`	tr

Reach

The reach of an article/post represents the number of people who were reached by this article/post. Note that the views only get set to a proper value if the host of the URL is either a domain (like theguardian.com) or if it is a domain with a well-known 3rd-level-subdomain in front (mainly applies to www, e.g. www.theguardian.com). Reach is set to 0 for other hosts, i.e. hosts with other 3rd-level-subdomains, like on foobar.blogspot.com, as using the Alexa views of the domain would assign much too high reach to mere sub-hosts otherwise.

Reach is calculated in the following ways:
Blogs; News Sites; Forums: Number of Page Views
Facebook: The Number of Fans of the Page (Note: Only available for public pages, which are monitored by Talkwalker, we don't collect any fan counts for user profiles)
Twitter: The number of Followers of the author

Images

Optional

images	MEDIA_ENTRY	Write access through API	Comment
`url`	Image Url	y	Link to Image
`width`	Image Width	y	Width of image, if available
`height`	Image Height	y	Height of image, if available
`legend`	Image legend	y	Legend text

Videos

Optional

videos	MEDIA_ENTRY	Write access through API	Comment
`url`	Video Url	y	Link to Image
`width`	Video Width	y	Width of image, if available
`height`	Video Height	y	Height of image, if available
`legend`	Video legend	y	Legend text of video

Attributes

These fields are only set for certain post types.

Article extended attributes fields will be updated for up to 1 month.

The source extended attributes represent the exact value at publication.

Not all urls will have all meta data, e.g.:

- Blog, news and messageboard posts (not their comments), will only have facebook_shares, twitter_shares set.
- All the other types will only be set if the sourcetype is of the same type and if the data is available.

article_extended_attributes	ARTICLE_EXTENDED_ATTRIBUTES	Write access through API	Comment
`facebook_shares`	Article Facebook Shares	y	Number of Facebook share an article has
`facebook_likes`	Article Facebook Likes	y	Number of Facebook likes an article has
`twitter_retweets`	Article Twitter Retweets	y	Number of Twitter retweets an article has
`twitter_likes`	Article Twitter Likes	y	Number of Twitter likes an article has
`url_views`	Article URL Views	y	
`pinterest_likes`	Article Pinterest Likes	y	Number of Pinterest likes an image has
`pinterest_pins`	Article Pinterest Pins	y	Number of Pinterest pins an image has
`pinterest_repins`	Article Pinterest Re-Pins	y	Number of Pinterest re-pins an article has
`youtube_views`	YouTube Video Views	y	Number of YouTube views a video has
`youtube_comments`	YouTube Video Comments	y	Number of YouTube comments a video has
`youtube_likes`	YouTube Video Likes	y	Number of YouTube likes a video has

article_extended_attributes	ARTICLE_EXTENDED_ATTRIBUTES	Write access through API	Comment
`youtube_dislikes`	YouTube Video Dislikes	y	Number of YouTube dislikes a video has
`instagram_likes`	Instagram Image Likes	y	Number of Instagram likes an image has
`twitter_shares`	Article Twitter Shares	y	Number of Twitter share an article has

source_extended_attributes	SOURCE_EXTENDED_ATTRIBUTES	Write access through API	Comment
`alexa_pageviews`	Alexa Page Views	y	
`facebook_followers`	Facebook Followers	y	Number of Facebook followers a source has
`twitter_followers`	Twitter Followers	y	Number of Twitter followers a source has
`instagram_followers`	Instagram Followers	y	Number of Instagram follows a source has
`pinterest_followers`	Pinterest Followers	y	Number of Pinterest follows a source has

article_attributes	ATTRIBUTES	Write access through API	Comment
`worlddata/continent`	Article Continent	n	Continental location of the article
`worlddata/country`	Article Country	n	Country location of the article
`worlddata/region`	Article Region	n	Regional location of the article
`worlddata/city`	Article City	n	City location of the article
`worlddata/longitude`	Article Longitude	n	Longitudinal location of the article
`worlddata/latitude`	Article Latitude	n	Latitudinal location of the article
`country_code`		y	
`resolution`		n	Resolution of the geo data extraction
`id`	Article ID	n	
`type`	Article Type	n	
`name`	Article Name	n	
`birthdate`	Article birth date	n	
`gender`	Article Gender	n	
`image_url`	Article Image URL	n	
`short_name`	Article Short Name	n	
`url`	Article URL	n	URL of the article

(For documents which don’t include location data, these fields are approximated)

author_attributes	ATTRIBUTES	Write access through API	Comment
`worlddata/continent`	Author Continent	n	Continental location of the author
`worlddata/country`	Author Country	n	Country location of the author
`worlddata/region`	Author Region	n	Regional location of the author
`worlddata/city`	Author City	n	City location of the author
`worlddata/longitude`	Author Longitude	n	Longitudinal location of the author
`worlddata/latitude`	Author Latitude	n	Latitudinal location of the author
`country_code`		y	
`resolution`		n	Resolution of the geo data extraction
`id`	Author ID	y	
`type`	Author Type	n	
`name`	Author Name	y	Name of the author
`birthdate`	Author Birthdate	n	Birthdate of the author
`gender`	Author Gender	y	Gender of the author
`image_url`	Author Image URL	y	
`short_name`	Author Short Name	y	
`url`	Author URL	y	Url to the profile of the author

(For documents which don’t include location data, these fields are approximated)

source_attributes	ATTRIBUTES	Write access through API	Comment
`worlddata/continent`	Source Continent	n	Continental location of the source
`worlddata/country`	Source Country	n	Country location of the source
`worlddata/region`	Source Region	n	Regional location of the source
`worlddata/city`	Source City	n	City location of the source
`worlddata/longitude`	Source Longitude	n	Longitudinal location of the source
`worlddata/latitude`	Source Latitude	n	Latitudinal location of the source
`country_code`		y	
`resolution`		n	Resolution of the geo data extraction
`id`	Source ID	y	
`type`	Source Type	n	
`name`	Source Name	y	
`birthdate`	Source Birthdate	n	

source_attributes	ATTRIBUTES	Write access through API	Comment
`gender`	Source Gender	n	
`image_url`	Source Image URL	y	
`short_name`	Source Short Name	n	
`url`	Source URL	y	URL of the source

(For documents which don't include location data, these fields are approximated)

Evolution and stability of document fields

The structure of the documents will not be changed. Existing fields will not be removed and their formatting will not be changed. Occasionally, new fields will be added to the documents and the order of fields can change, please take this into account when implementing a custom client.

Streaming

(repeated extra entries for each matching rule, available in streaming only)

Extra Fields

On streaming, this information is present in extra entrydata

Field Name	Name	Write access through API	Comment
`highlighted_data`	Highlighted Data	n	Content and title snipped of matched rules queries and panels
`matched`	Matched	n	Stream, Rule and Panel which were matched.
`rule_id`	matched rule	n	ID of matched rule
`rule_query`	matched rule	n	Query of matched rule (when id is not set)
`stream_id`	matched stream	n	ID of matched stream
`panel_id`	matched panel	n	ID of matched Panel
`matched_profile`	Matched Profile	n	Profile which matched (if Talkwalker)
`title_snippet`	Title Snippet	n	If a match occurred in the title, this field will contain the snippet related to the query set in the datafeed.
`content_snippet`	Content Snippet	n	If a match occurred in the article, this field will contain the snippet related to the query set in the datafeed.

Protocols, Encodings and Value Field Options

Protocols and Encodings

The Talkwalker API uses HTTP protocol 1.1. The Streaming API streams documents using the HTTP 1.1 Chunked transfer encoding mechanism.

The data is compressed using gzip: "Accept-Encoding:gzip" must be set in the header. The Encoding used is `UTF-8`.

Evolution of JSON fields

The structure of the json responses will not be changed. Existing fields will not be removed and their formatting will not be changed. However, new fields will be added to the responses and the order of fields can change, please take this into account when implementing a custom client.

Value options

The Following tables contain possible options and formats for certain fields.

Source Type Options

Media Source Types	
`ONLINENEWS`	All news sites
`ONLINENEWS_MAGAZINE`	Printed magazines sites
`ONLINENEWS_NEWSPAPER`	Printed newspaper sites
`ONLINENEWS_PRESSRELEASES`	Results from sites that publish press releases
`ONLINENEWS_TVRADIO`	TV or radio stations
`ONLINENEWS_AGENCY`	News agencies
`ONLINENEWS_OTHER`	News results that do not fall under of the other news categories
`BLOG`	All blog sites
`MESSAGEBOARD`	All forums and message boards
`SOCIALMEDIA`	All social media sites
`SOCIALMEDIA_TWITTER`	Results from Twitter
`SOCIALMEDIA_FACEBOOK`	Results from Facebook
`SOCIALMEDIA_YOUTUBE`	Results from YouTube
`SOCIALMEDIA_LINKEDIN`	Results from LinkedIn
`SOCIALMEDIA_GOOGLEPLUS`	Results from Google+
`SOCIALMEDIA_FLICKR`	Results from Flickr
`SOCIALMEDIA_FOURSQUARE`	Results from Foursquare
`SOCIALMEDIA_INSTAGRAM`	Results from Instagram

Media Source Types	
SOCIALMEDIA_MIXCLOUD	Results from Mixcloud
SOCIALMEDIA_SOUNDCLOUD	Results from SoundCloud
SOCIALMEDIA_VIMEO	Results from Vimeo
SOCIALMEDIA_DAILYMOTION	Results from Dailymotion
OTHER	Everything else which does not fit into the above listed categories

Language Options

ABKHAZIAN	ab	HERERO	hz	PALI	pi
AFAR	aa	HINDI	hi	PANJABI	pa
AFRIKAANS	af	HIRI MOTU	ho	PERSIAN	fa
AKAN	ak	HUNGARIAN	hu	POLISH	pl
ALBANIAN	sq	ICELANDIC	is	PORTUGUESE	pt
AMHARIC	am	IDO	io	PUSHTO	ps
ARABIC	ar	IGBO	ig	QUECHUA	qu
ARAGONESE	an	INDONESIAN	id	RAETO ROMANCE	rm
ARMENIAN	hy	INTERLINGUA	ia	ROMANIAN	ro
ASSAMESE	as	INTERLINGUE	ie	RUNDI	rn
AVARIC	av	INUKTITUT	iu	RUSSIAN	ru
AVESTAN	ae	INUPIAQ	ik	SAMOAN	sm
AYMARA	ay	IRISH	ga	SANGO	sg
AZERBAIJANI	az	ITALIAN	it	SANSKRIT	sa
BAMBARA	bm	JAPANESE	ja	SARDINIAN	sc
BASHKIR	ba	JAVANESE	jv	SCOTTISH GAELIC	gd
BASQUE	eu	KANNADA	kn	SERBIAN	sr
BELARUSIAN	be	KANURI	kr	SHONA	sn
BENGALI	bn	KASHMIRI	ks	SICHUAN YI	ii
BIHARI	bh	KAZAKH	kk	SINDHI	sd
BISLAMA	bi	KHMER	km	SINHALESE	si
BOSNIAN	bs	KIKUYU	ki	SLOVAK	sk
BRETON	br	KINYARWANDA	rw	SLOVENIAN	sl
BULGARIAN	bg	KIRGHIZ	ky	SOMALI	so
BURMESE	my	KOMI	kv	SOUTHERN SOTHO	st

CATALAN	ca	KONGO	kg	SOUTH NDEBELE	nr
CHAMORRO	ch	KOREAN	ko	SPANISH	es
CHECHEN	ce	KURDISH	ku	SUNDANESE	su
CHINESE	zh	KWANYAMA	kj	SWAHILI	sw
CHINESE SIMPLIFIED	zh cn	LAO	lo	SWATI	ss
CHINESE TRADITIONAL	zh tw	LATIN	la	SWEDISH	sv
CHURCH SLAVIC	cu	LATVIAN	lv	TAGALOG	tl
CHUVASH	cv	LIMBURGISH	li	TAHITIAN	ty
CORNISH	kw	LINGALA	ln	TAJIK	tg
CORSICAN	co	LITHUANIAN	lt	TAMIL	ta
CREE	cr	LUBA KATANGA	lu	TATAR	tt
CROATIAN	hr	LUXEMBOURGISH	lb	TELUGU	te
CZECH	cs	MACEDONIAN	mk	THAI	th
DANISH	da	MALAGASY	mg	TIBETAN	bo
DIVEHI	dv	MALAY	ms	TIGRINYA	ti
DUTCH	nl	MALAYALAM	ml	TONGA	to
DZONGKHA	dz	MALTESE	mt	TSONGA	ts
ENGLISH	en	MANX	gv	TSWANA	tn
ESPERANTO	eo	MAORI	mi	TURKISH	tr
ESTONIAN	et	MARATHI	mr	TURKMEN	tk
EWE	ee	MARSHALLESE	mh	TWI	tw
FAROESE	fo	MOLDAVIAN	mo	UIGHUR	ug
FIJIAN	fj	MONGOLIAN	mn	UKRAINIAN	uk
FINNISH	fi	NAURU	na	URDU	ur
FRENCH	fr	NAVAJO	nv	UZBEK	uz
FRISIAN	fy	NDONGA	ng	VENDA	ve
FULAH	ff	NEPALI	ne	VIETNAMESE	vi
GALLEGAN	gl	NORTHERN SAMI	se	VOLAPUK	vo
GANDA	lg	NORTH NDEBELE	nd	WALLOON	wa
GEORGIAN	ka	NORWEGIAN	no	WELSH	cy
GERMAN	de	NORWEGIAN BOKMAL	nb	WOLOF	wo
GREEK	el	NORWEGIAN NYNORSK	nn	XHOSA	xh
GREENLANDIC	kl	NYANJA	ny	YIDDISH	yi

GUARANI	gn	OCCITAN	oc	YORUBA	yo
GUJARATI	gu	OJIBWA	oj	ZHUANG	za
HAITIAN	ht	ORIYA	or	ZULU	zu
HAUSA	ha	OROMO	om		
HEBREW	he	OSSETIAN	os		

Country Options

AFGHANISTAN	af	GIBRALTAR	gi	PALESTINE	ps
ALAND ISLANDS	ax	GREECE	gr	PANAMA	pa
ALBANIA	al	GREENLAND	gl	PAPUA NEW GUINEA	pg
ALGERIA	dz	GRENADA	gd	PARAGUAY	py
AMERICAN SAMOA	as	GUADELOUPE	gp	PERU	pe
ANDORRA	ad	GUAM	gu	PHILIPPINES	ph
ANGOLA	ao	GUATEMALA	gt	PITCAIRN	pn
ANGUILLA	ai	GUERNSEY	gg	POLAND	pl
ANTARCTICA	aq	GUINEA	gn	PORTUGAL	pt
ANTIGUA AND BARBUDA	ag	GUINEA BISSAU	gw	PUERTO RICO	pr
ARGENTINA	ar	GUYANA	gy	QATAR	qa
ARMENIA	am	HAITI	ht	REUNION	re
ARUBA	aw	HEARD ISLAND AND MCDONALD ISLANDS	hm	ROMANIA	ro
AUSTRALIA	au	HONDURAS	hn	RUSSIA	ru
AUSTRIA	at	HONG KONG	hk	RWANDA	rw
AZERBAIJAN	az	HUNGARY	hu	SAINT BARTHELEMY	bl
BAHAMAS	bs	ICELAND	is	SAINT HELENA	sh
BAHRAIN	bh	INDIA	in	SAINT KITTS AND NEVIS	kn
BANGLADESH	bd	INDONESIA	id	SAINT LUCIA	lc
BARBADOS	bb	IRAN	ir	SAINT MARTIN	mf
BELARUS	by	IRAQ	iq	SAINT PIERRE AND MIQUELON	pm
BELGIUM	be	IRELAND	ie	SAINT VINCENT AND THE GRENADINES	vc
BELIZE	bz	ISLE OF MAN	im	SAMOA	ws
BENIN	bj	ISRAEL	il	SAN MARINO	sm

BERMUDA	bm	ITALY	it	SAO TOME AND PRINCIPE	st
BHUTAN	bt	JAMAICA	jm	SAUDI ARABIA	sa
BOLIVIA	bo	JAPAN	jp	SENEGAL	sn
BONAIRE SINT EUSTASIUS AND SABA	bq	JERSEY	je	SERBIA	rs
BOSNIA AND HERZEGOVINA	ba	JORDAN	jo	SERBIA AND MONTENEGRO	cs
BOTSWANA	bw	KAZAKHSTAN	kz	SEYCHELLES	sc
BOUVET ISLAND	bv	KENYA	ke	SIERRA LEONE	sl
BRAZIL	br	KIRIBATI	ki	SINGAPORE	sg
BRITISH INDIAN OCEAN TERRITORY	io	KUWAIT	kw	SINT MAARTEN	sx
BRITISH VIRGIN ISLANDS	vg	KYRGYZSTAN	kg	SLOVAKIA	sk
BRUNEI	bn	LAOS	la	SLOVENIA	si
BULGARIA	bg	LATVIA	lv	SOLOMON ISLANDS	sb
BURKINA FASO	bf	LEBANON	lb	SOMALIA	so
BURUNDI	bi	LESOTHO	ls	SOUTH AFRICA	za
CAMBODIA	kh	LIBERIA	lr	SOUTH GEORGIA AND THE SOUTH SANDWICH ISLANDS	gs
CAMEROON	cm	LIBYA	ly	SOUTH KOREA	kr
CANADA	ca	LIECHTENSTEIN	li	SOUTH SUDAN	ss
CAPE VERDE	cv	LITHUANIA	lt	SPAIN	es
CAYMAN ISLANDS	ky	LUXEMBOURG	lu	SRI LANKA	lk
CENTRAL AFRICAN REPUBLIC	cf	MACAO	mo	SUDAN	sd
CHAD	td	MACEDONIA	mk	SURINAME	sr
CHILE	cl	MADAGASCAR	mg	SVALBARD AND JAN MAYEN	sj
CHINA	cn	MALAWI	mw	SWAZILAND	sz
CHRISTMAS ISLAND	cx	MALAYSIA	my	SWEDEN	se
COCOS ISLANDS	cc	MALDIVES	mv	SWITZERLAND	ch
COLOMBIA	co	MALI	ml	SYRIA	sy
COMOROS	km	MALTA	mt	TAIWAN	tw
CONGO	cg	MARSHALL ISLANDS	mh	TAJIKISTAN	tj
COOK ISLANDS	ck	MARTINIQUE	mq	TANZANIA	tz
COSTA RICA	cr	MAURITANIA	mr	THAILAND	th

COTE DIVOIRE	ci	MAURITIUS	mu	THE DEMOCRATIC REPUBLIC OF CONGO	cd
CROATIA	hr	MAYOTTE	yt	TIMOR LESTE	tl
CUBA	cu	MEXICO	mx	TOGO	tg
CURACAO	cw	MICRONESIA	fm	TOKELAU	tk
CYPRUS	cy	MOLDOVA	md	TONGA	to
CZECH REPUBLIC	cz	MONACO	mc	TRINIDAD AND TOBAGO	tt
DENMARK	dk	MONGOLIA	mn	TUNISIA	tn
DJIBOUTI	dj	MONTENEGRO	me	TURKEY	tr
DOMINICA	dm	MONTSERRAT	ms	TURKMENISTAN	tm
DOMINICAN REPUBLIC	do	MOROCCO	ma	TURKS AND CAICOS ISLANDS	tc
ECUADOR	ec	MOZAMBIQUE	mz	TUVALU	tv
EGYPT	eg	MYANMAR	mm	UGANDA	ug
EL SALVADOR	sv	NAMIBIA	na	UKRAINE	ua
EQUATORIAL GUINEA	gq	NAURU	nr	UNITED ARAB EMIRATES	ae
ERITREA	er	NEPAL	np	UNITED KINGDOM	uk
ESTONIA	ee	NETHERLANDS	nl	UNITED STATES	us
ETHIOPIA	et	NETHERLANDS ANTILLES	an	UNITED STATES MINOR OUTLYING ISLANDS	um
FALKLAND ISLANDS	fk	NEW CALEDONIA	nc	URUGUAY	uy
FAROE ISLANDS	fo	NEW ZEALAND	nz	US VIRGIN ISLANDS	vi
FIJI	fj	NICARAGUA	ni	UZBEKISTAN	uz
FINLAND	fi	NIGER	ne	VANUATU	vu
FRANCE	fr	NIGERIA	ng	VATICAN	va
FRENCH GUIANA	gf	NIUE	nu	VENEZUELA	ve
FRENCH POLYNESIA	pf	NORFOLK ISLAND	nf	VIETNAM	vn
FRENCH SOUTHERN TERRITORIES	tf	NORTHERN MARIANA ISLANDS	mp	WALLIS AND FUTUNA	wf
GABON	ga	NORTH KOREA	kp	WESTERN SAHARA	eh
GAMBIA	gm	NORWAY	no	YEMEN	ye
GEORGIA	ge	OMAN	om	ZAMBIA	zm
GERMANY	de	PAKISTAN	pk	ZIMBABWE	zw
GHANA	gh	PALAU	pw		

API Account

Access Token

Demo

To try the Talkwalker API, you can use the access token *demo* (`access_token=demo`). With this token you can try the Search API (results and histogram) and the streaming API. Accessing the Talkwalker API with this token, will not return any social media results, only results from blogs, forums and news are returned. (this token can be used for testing only)

Your own Access Token

To use the Talkwalker API with the topics from your Talkwalker or to get results from social media (Twitter, Facebook…) you need to apply and get your own access tokens.

- `read_write` access tokens are necessary for search, channel monitoring, updating and deleting documents in a project and for creating streams, deleting streams, setting panels and setting rules.
- `authentication` access tokens are necessary when using the Authentication API.

To get an access token please contact us.

OAuth 2.0

For an integration of private Talkwalker widgets and data in external applications, Talkwalker and the Talkwalker API can authenticate users via OAuth 2.0. Every external application that wants to use such data needs OAuth 2.0 credentials for Talkwalker (a `client_id` and `client_secret`) and needs to provide a redirect URL.

To ask for permission to access private data, the external application redirects the user to:

```
http://www.talkwalker.com/app/oauth/authorize?client_id=<client_id>&response_type=code&redirect_uri=<redirect_uri_encoded>&scope=projects
```

After the user has granted permission, he will be redirected to the redirect URL provided by the external application. This redirect will include a query string with a access code parameter (`?code=<access_code>`).

To get the actual OAuth access token for a user, the external application makes a POST request to: `http://www.talkwalker.com/app/oauth/access_token?client_id=<client_id>&client_secret=<client_secret>&grant_type=authorization_code&redirect_uri=<redirect_uri_encoded>&code=<authorization_code>` with the header : `Content-Type: application/x-www-form-urlencoded`

The Talkwalker server will respond with a body of the following form: `access_token=<oauth_access_token>`

The external application can now use the OAuth access token instead of a Talkwalker API access token. Instead of setting the query string field `access_token`, the requests must contain the header field

`Authorization` to `Bearer <oauth_access_token>`.

for more information about OAuth 2.0 see `http://oauth.net/2/`

OAuth 2.0 Setup

To get a `client_id` and a `client_secret` please contact us. You will have to provide one or more `redirect_uri` (for development purposes localhost is allowed).

Credits / Pricing

Monthly Reset of Credits

The credits will be reset every month, on the day of the subscription at 03:00 UTC. (Note that the monthly new results in Talkwalker projects are reset on the first of a new month at 0:00 UTC)

Remaining Credits Endpoint

The endpoint `https://api.talkwalker.com/api/v1/status/credits` is used to get an overview of consumed credits and API calls.

Response:

```
{
  "status_code" : "0",
  "status_message" : "OK",
  "request" : "GET /api/v1/status/credits?access_token=demo",
  "result_creditinfo" {
  "used_credits_monthly" : 0,
  "used_credits_onetime" : 0,
  "remaining_credits_monthly" : 0,
  "remaining_credits_onetime" : 0,
  "next_billing_period" : 1419634800000,
  "estimate_credits_used_until_end_of_billing_period" : 0,
  "monthly_total" : 0
  }
}
```

Rate Limit

This endpoint is limited to 10 calls per minute, the result should be stored.

FAQ

How to stream all documents from a Talkwalker project?

The following command creates a stream "test" used to stream the documents to your application.

```
curl -XPUT 'https://api.talkwalker.com/api/v2/stream/s/test?access_token=<access_token>' -d '{}' -H "Content-
Type: application/json; charset=UTF-8"
```

You can then use the "test" stream to stream all documents in real time from your Talkwalker project to your application. This will return in real time all new results which have been found since the time you executed below command:

```
curl 'https://api.talkwalker.com/api/v2/stream/s/test/p/<project_id>/results?access_token=<access_token>'
```

This will stream the data to your application. For each entry (or for every second if there are no entries) our server will send you a newline.

Below is an example of the data you will receive:

```
{
  "chunk_type" : "CT_CONTROL",
  "chunk_control" : {
    "timeframe_start" : 1409906205401,
    "timeframe_end" : 1409906265618
  }
}
{
  "chunk_type" : "CT_RESULT",
  "chunk_result" : {
    "data" : {
      "data" : {
        "url" : "http://www.facebook.com/permalink.php?id=45012929134&story_fbid=10152329058194135",
        "matched_profile" : [
          "hznwvi3k_5imn0wzqr36f"
        ],
        "indexed" : 1409906120127,
        "search_indexed" : 1409906245484,
        "published" : 1409902879000,
        "title" : "",
        "content" : "Cn u hlp me abt my dstv account",
        "title_snippet" : "",
        "content_snippet" : "Cn u hlp me abt my <b>dstv</b> account",
        "root_url" : "http://www.facebook.com/45012929134",
        "domain_url" : "http://facebook.com/",
        "host_url" : "http://www.facebook.com/",
        "parent_url" : "http://www.facebook.com/permalink.php?id=45012929134&story_fbid=10152329058194135",
        "lang" : "en",
        "porn_level" : 0,
        "fluency_level" : 100,
        "spam_level" : 0,
        "sentiment" : 0,
        "source_type" : [
          "SOCIALMEDIA",
          "SOCIALMEDIA_FACEBOOK"
        ],
        "post_type" : [
          "TEXT"
        ],
        "article_extended_attributes" : {
          "num_comments" : 1
        },
        "source_extended_attributes" : {
          "alexa_pageviews" : 60438000000
        },
        "extra_article_attributes" : {
          "world_data" : {
          }
        },
        "extra_author_attributes" : {
          "world_data" : {
          },
          "id" : "fb:100007373088511",
          "name" : "S'bu Dlokweni",
          "gender" : "UNKNOWN",
          "image_url" : "https://graph.facebook.com/100007373088511/picture",
          "url" : "http://www.facebook.com/profile.php?id=100007373088511"
        },
        "extra_source_attributes" : {
          "world_data" : {
            "continent" : "Africa",
            "country" : "South Africa",
            "region" : "Orange Free State",
            "city" : "Bloemfontein",
            "longitude" : 26.2299128812,
            "latitude" : -29.1199938774,
            "country_code" : "za"
          }
        },
        "engagement" : 1,
        "reach" : 0
      }
    }
  }
}
```

It consists of CT_DATA (the data entries) and CT_CONTROL (the control entries). One example CT_CONTROL stream is shown below:

```
[{"chunk_type":"CT_CONTROL","chunk_control":{"timeframe_start":1409906135111,"timeframe_end":1409906205401}}
```

In this case, all results from 1409906135111 to 1409906205401 will be streamed to the application.

In case of disconnection (e.g. connection issue, application got restarted), you can provide the latest timeframe_start as a starting point as a value for the parameter stream_resume:

```
curl
'https://api.talkwalker.com/api/v2/stream/s/test/p/<project_id>/results?access_token=<access_token>&stream_res
ume=1409906135111'
```

Below command returns the list of topics, which can then be used to only stream a certain topic and not all topics:

```
curl 'https://api.talkwalker.com/api/v2/talkwalker/p/<project_id>/resources?access_token=<access token>'
curl 'https://api.talkwalker.com/api/v2/stream/s/test/p/<project id>/results?access_token=<access
token>&topic=<topic id 1>&topic=<topic id 2>'
```

How to stream all documents from a Talkwalker project for a specific month?

The following command creates a stream "test" used to stream the documents to your application.

```
curl 'https://api.talkwalker.com/api/v2/stream/test?access_token=<access_token>' -d '{}' -H "Content-Type:
application/json; charset=UTF-8"
```

You can then use the "test" stream to stream all documents from August 2014 from your Talkwalker project to your application. To get only the documents from August set a query `published:>1406851200000 AND published:<1409529600000` to restrict the stream to documents from August and set `stream_resume=1406851200000` to start the stream on August 1. Set a `stream_stop` time later than the end of August so you get all documents from August, also those that were found and streamed later (for example use the current time : `stream_stop=1422543275000`).

Note: To get all documents from August, do not set stream_stop to the end of August. Documents that were published in August could have been added to the stream at a later point as we only found them later.

```
curl
'https://api.talkwalker.com/api/v2/stream/s/test/p/<project_id>/results?access_token=<access_token>&q=publishe
d:>1406851200000%20AND%20published:<1409529600000&stream_resume=1406851200000'
```

How to get the documents of the last hour of a Talkwalker project?

To get the results from the last hour, set `stream_resume` to the epoch time one hour (i.e. 3600000 milliseconds) ago and `stream_stop` to the most recent time. You will get all the documents that have been found during the last hour.

Note: these are the documents that were found during this period (timestamp in `search_indexed`) the documents were not necessarily published during the last hour, thus the set of documents is not equal to the set shown for the last hour in Talkwalker. When documents that were published earlier are found (and streamed), they are added to Talkwalker for the period they were published in.

```
curl
'https://api.talkwalker.com/api/v2/stream/s/test/p/<project_id>/results?access_token=<access_token>&stream_res
ume=1420531486000&stream_stop=1420535086000'
```

How to stream all documents from Talkwalker Page Monitoring

The following command creates a stream "test" used to stream the documents to your application.

```
curl -XPUT 'https://api.talkwalker.com/api/v2/stream/create?access_token=<access_token>' -d
'{"streamid":"test"}' -H "Content-Type: application/json; charset=UTF-8"
```

You can then use the "test" stream to stream all documents from page monitoring by settings topic to `page`:

```
curl 'https://api.talkwalker.com/api/v2/stream/s/test/p/<project id>/results?access_token=<access
token>&topic=page'
```

How to eliminate retweets or comments from a stream?

To remove retweets and retrieve only the original Tweets add `-is:retweet` (or `-is:comment`) to the rules of a stream.

If you want to remove all retweets from an entire stream you can also add a query (`-is:retweet`) when getting the results of a stream.

```
curl 'https://api.talkwalker.com/api/v2/stream/s/test/p/<project_id>/results?access_token=<access_token>&q=-
is:retweet'
```

How to get only documents of a Talkwalker project that include special keywords

To get a stream of only a subset of the documents of a Talkwaker project, you can set up rules for your stream. Rules are expressed in the Talkwaker query syntax. `https://api.talkwalker.com/api/v2/stream/s/<stream_id>/r/<rule_id>` is used to set new rules for an existing stream. If you define more than one rule, the stream will return any documents that match at least one rule.

```
curl -XPUT https://api.talkwalker.com/api/v2/stream/s/teststream/r/rule-1?access_token=demo -d '{
"query":"keyword1 AND keyword2" }' -H "Content-Type: application/json; charset=UTF-8"
```

The stream will now only return documents that match "keyword1 AND keyword2", the field `data.highlighted_data.matched.rule_id` indicates which rules were matched.

How to use a single stream for multiple applications / clients?

To use one stream to retrieve data for more than one application / client, rules are used. Set a separate rule (using the Talkwaker query syntax) for each application.

```
curl -XPUT https://api.talkwalker.com/api/v2/stream/s/teststream?access_token=<access_token> -d
'{"rules":[{"rule_id" : "rule-1", "query" : "foo"},{"rule_id" : "rule-2", "query" : "bar"}]}'
```

The returned results will be in the format below. The documents can be separated using `matched_query`, which indicates which rule the result belongs to.

```
{
  "chunk_type" : "CT_RESULT",
  "chunk_result" : {
    "data" : {
      "data" : { <default result data (see simple search)> },
      "highlighted_data" : [ {
        "matched" : {
          "rule_id" : "rule-1"
        }
        "title_snippet" : "<title snippet for rule>",
        "content_snippet" : "<content snippet for rule>"
      } ]
    }
  }
}
```

How to get the number of results grouped by media types?

The Talkwalker API provides only documents and histograms, to group results into custom sets, you have to get all the results and then compute those sets locally. Alternatively you can perform separate searches (or histograms) for each of the groups you want to create (use the Talkwalker query syntax to restrict the results to those matching a single group).

How to get the ids of Talkwalker Topics?

To get a list of the search-topics defined in a Talkwalker project use the `project_id` and the `access_token` on the `https://api.talkwalker.com/api/v2/talkwalker/p/<project_id>/resources` endpoint with the filter `type=search`.

```
curl
'https://api.talkwalker.com/api/v2/talkwalker/p/<project_id>/resources?access_token=<access_token>&type=search
'
```

The result could look like this:

```
{
  "status_code" : "0",
  "status_message" : "OK",
  "request" : "GET /api/v2/talkwalker/p/<project_id>/resources?access_token=<access_token>&type=search",
  "result_resources" : {
    "projects" : [ {
      "id" : "<project_id>",
      "title" : "Air France",
      "topics" : [ {
        "id" : "search|1",
        "title" : "Category 1",
        "nodes" : [ {
          "id" : "search|1|1",
          "title" : "topic 1"
        }, {
          "id" : "search|1|2",
          "title" : "topic 2"
        } ]
      }, {
        "id" : "search|2",
        "title" : "Catergory 2",
        "nodes" : [ {
          "id" : "search|2|1",
          "title" : "topic 1"
        }, {
          "id" : "search|2|2",
          "title" : "topic 2"
        }, {
          "id" : "search|2|2",
          "title" : "topic 3"
        } ]
      } ]
    } ]
  }
}
```

To get results for all projects in 'search' use `search` as topic ID. To use a single topic use the id of the topic (for example `search|2|1`) for topic 1 of category 2 in search).

Code Examples

Streaming Client Examples

PHP

Note: This example needs the php cURL library and PHP 5.5.

client.php

```
<?php

class TalkwalkerApiStreamingClientExample
{
  private $url;
  private $token;

  # internal
  private $finished = FALSE;
  private $resume_ts;
  private $unprocessed_data = '';
  private $header_size = -1;
  private $header = '';
  private $header_complete = FALSE;
  private $wait_for_retry = 0;
  private $error_data = '';

  public function __construct($url, $token) {
    $this->url = $url;
    $this->token = $token;

  }

  function setCurlOptions($ch) {
    curl_setopt($ch, CURLOPT_CONNECTTIMEOUT, 30);
    curl_setopt($ch, CURLOPT_TIMEOUT, 90);
    curl_setopt($ch, CURLOPT_FAILONERROR, FALSE);
    curl_setopt($ch, CURLOPT_HEADER, TRUE);
    curl_setopt($ch, CURLOPT_USERAGENT, 'PhpExampleClient/1.0.0');
    curl_setopt($ch, CURLOPT_ENCODING, 'gzip');
  }

  public function run($streamid, $project, $start_ts, $stop_ts) {
    $this->resume_ts = $start_ts;
     while (!$this->finished) {
```

```
    $this->unprocessed_data = '';
    $this->error_data = '';
    $this->header_size = -1;
    $this->header_complete = FALSE;
    $this->header = '';

    $ch = curl_init();
    $_url = $this->url . '/v2/stream/s/' . $streamid;
    if(!empty($project)) {
      $_url .= '/p/' . $project;
    }
    $_url .= '/results?';
    $_url .= 'access_token=' . $this->token;
    $_url .= '&stream_resume=' . $this->resume_ts . '&stream_stop=' . $stop_ts;

    curl_setopt($ch, CURLOPT_URL, $_url);
    curl_setopt($ch, CURLOPT_HTTPGET, 1);
    $this->setCurlOptions($ch);

    $headers = array(
        'Cache-Control: no-cache',
        'Pragma: no-cache',
        'Content-Language: en-US');
    curl_setopt($ch, CURLOPT_HTTPHEADER, $headers);
    curl_setopt($ch, CURLOPT_WRITEFUNCTION, array($this, "read_stream"));

    curl_exec($ch);

    // check if something is in $error_data

    // check error code
    $http_status = curl_getinfo($ch, CURLINFO_HTTP_CODE);

    if (curl_errno($ch) == 0 && $http_status == 200) {
      $this->finished = TRUE;
    }

    // else: error occurred
    if ($http_status > 0 && $http_status != 200) {
      $this->onStatusError($this->error_data);
    }

    curl_close($ch);

    if (!$this->finished) {
      if ($this->wait_for_retry > 0) {
        echo "SERVICE UNAVAILABLE \n";
        echo "WAITING " . $this->wait_for_retry . "s UNTIL RETRYING\n";
        sleep($this->wait_for_retry);
        $this->wait_for_retry = 0;
      } else {
        sleep(5); // 60
      }
    }
  }
}

function read_stream($ch, $data) {
  $http_status = curl_getinfo($ch, CURLINFO_HTTP_CODE);
  $header_size = curl_getinfo($ch, CURLINFO_HEADER_SIZE);

  $this->unprocessed_data = $this->unprocessed_data . $data;

  // read the header when it is complete
  if ($this->header_size < $header_size) {
    $this->header_size = $header_size;
    $header_complete = FALSE;
  } else {
    $header_complete = TRUE;
  }
  $partial_header = substr($this->unprocessed_data, 0, $header_size);
  if ($header_complete && $this->header == '') {
    $this->header = substr($this->unprocessed_data, 0, $header_size);
    $this->unprocessed_data = substr($this->unprocessed_data, $header_size);
  }


  if ($header_complete && $http_status == 200) {
    // split on '\r\n'
    $arr_data = explode("\r\n", $this->unprocessed_data);
    $count = count($arr_data);
    for ($i = 0; $i < $count; $i++) {
      $line = $arr_data[$i];
      // try parse json
      if (strlen($line) > 0) {
        $json = json_decode($line);
        if ($json == NULL) {
          // put it back only if last element
          if ($i == $count-1) {
            $this->unprocessed_data = $line;
          } else {
            $this->finished = TRUE;
            $this->handleParseError($line);
          }
        } else {
          if (isset($json->chunk_type)) {
            switch ($json->chunk_type) {
              case "CT_ERROR":
                $this->handleStreamError($json->chunk_error);
                break;
              case "CT_CONTROL":
                if (isset($json->chunk_control->timeframe_start)) {
                  $this->resume_ts = $json->chunk_control->timeframe_start;
                }
                $this->handleStreamControl($json->chunk_control);
                break;
              case "CT_RESULT":
                $this->handleStreamResult($json->chunk_result);
                break;
              default:
                $this->unhandledStreamChunk($json);
                break;
            }
          } else {
            $this->unhandledStreamChunk($json);
            break;
          }
        }
      } else {
        $this->unprocessed_data = '';
      }
    }
  } elseif ($http_status == 503) {
    $header_array = $this->parseHeader($partial_header);
    if (array_key_exists('Retry-After', $header_array)) {
      $this->wait_for_retry = $header_array['Retry-After'];
    }
  } else {
    $this->error_data = $this->error_data . $data;
  }

  return strlen($data);
}

function onStatusError($str) {
  echo "START ERROR \n{$str}\n";
}

function handleParseError($str) {
  echo "Could not parse '{$str}'\n";
}

function handleStreamError($err) {
  echo "ERROR\n";
  var_dump($err);
}
```

```
function handleStreamControl($ctrl) {
  echo "CONTROL [{$ctrl->timeframe_start} TO {$ctrl->timeframe_end}]\n";
}

function handleStreamResult($res) {
  if (isset($res->data->data->url)) {
    echo "RESULT: {$res->data->data->url}\n";
  }
}

function unhandledStreamChunk($json) {
  echo "UNHANDLED\n";
  var_dump($json);
}

function parseHeader($header) {
  $headers = array();
  foreach (explode("\r\n", $header) as $i => $line)
    if ($i === 0) {
      $headers['http_code'] = $line;
     } else {
      if($line != '') {
        list ($key, $value) = explode(': ', $line);
        $headers[$key] = $value;
      }
    }
  return $headers;
}

function createStream($name) {
  $ch = curl_init();
  $stream = new stdClass;
  $stream->streamid = $name;

  $_url = $this->url . '/v1/stream/create?';
  $_url .= 'access_token=' . $this->token;

  $this->setCurlOptions($ch);
  curl_setopt($ch, CURLOPT_URL, $_url);
  curl_setopt($ch, CURLOPT_POST, 1);
  curl_setopt($ch, CURLOPT_RETURNTRANSFER, TRUE);

  $headers = array(
     'Cache-Control: no-cache',
     'Pragma: no-cache',
     'Content-Language: en-US');
  curl_setopt($ch, CURLOPT_HTTPHEADER, $headers);
  curl_setopt($ch, CURLOPT_POSTFIELDS, json_encode($stream));
```

```
    $result = curl_exec($ch);
    curl_close($ch);
    $answer = json_decode($result);
    if($answer != null && $answer->status_code != '0')  {
      echo $result;
      return;
    }
    echo 'CREATED STREAM : '. $name . "\n";
    return $name;
  }

  function deleteStream($name) {
    $ch = curl_init();

    $_url = $this->url . '/v1/stream/s/' . $name;
    $_url .= '/delete?';
    $_url .= 'access_token=' . $this->token;

    $this->setCurlOptions($ch);
    curl_setopt($ch, CURLOPT_URL, $_url);
    curl_setopt($ch, CURLOPT_CUSTOMREQUEST, "DELETE");
    curl_setopt($ch, CURLOPT_RETURNTRANSFER, TRUE);

    $headers = array(
       'Cache-Control: no-cache',
       'Pragma: no-cache',
       'Content-Language: en-US');
    curl_setopt($ch, CURLOPT_HTTPHEADER, $headers);

    $result = curl_exec($ch);
    curl_close($ch);
    $answer = json_decode($result);
    if($answer != null && $answer->status_code != '0')  {
      echo $result;
      return;
    }
    echo 'DELETED STREAM : '. $name . "\n";
    return $name;
  }
}

/** Test call method */
function main() {
  $url = 'https://api.talkwalker.com/api/v2/stream/s/<stream_id>/p/<project_id>/results?access_token=<token>';
  $start_ts = time() * 1000;
  $stop_ts = time() * 1000 + 60*60*1000;

  $example = new TalkwalkerApiStreamingClientExample($url, $start_ts, $stop_ts);
  $example->run();
```

```
}

main();

?>
```

Java

client.java

```
package com.trendiction.api.client.streamapi.streaming2;

import java.io.BufferedReader;
import java.io.ByteArrayInputStream;
import java.io.ByteArrayOutputStream;
import java.io.DataOutputStream;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.net.HttpURLConnection;
import java.net.URL;
import java.net.URLConnection;
import java.util.HashMap;
import java.util.Map;
import java.util.concurrent.atomic.AtomicLong;
import java.util.zip.GZIPInputStream;

import org.apache.commons.io.IOUtils;
import org.codehaus.jackson.node.JsonNodeFactory;
import org.codehaus.jackson.node.ObjectNode;

import com.fasterxml.jackson.core.JsonFactory;
import com.fasterxml.jackson.core.type.TypeReference;
import com.fasterxml.jackson.databind.ObjectMapper;

import com.trendiction.config.Time;

/**
 * Example class can be used as an example.
 * It is invoked via the ExampleTest class in this test case
 */
public class TalkwalkerApiStreamingClientExample {
  private final String url;
  private final String token;
  private final String stream_id;
  private final long start_ts;
  private final long stop_ts;

  public TalkwalkerApiStreamingClientExample(String url, String token, String stream_id, long start_ts, long
stop_ts) {
    this.url = url;
    this.token = token;
    this.stream_id = stream_id;
    this.start_ts = start_ts;
    this.stop_ts = stop_ts;
  }

  public void run() throws InterruptedException, IOException {
    deleteStream();
    System.out.println("CREATING STREAM");
    createStream();
    AtomicLong resume_ts = new AtomicLong(start_ts);
    boolean finished = false;

    while (!finished) {
      try {
        String _url = url + "/v2/stream/s/" + stream_id + "/results?access_token=" + token + "&stream_resume="
+ resume_ts.get() + "&stream_stop="
            + stop_ts;

        // connect
        URL request = new URL(_url);
        URLConnection connection = request.openConnection();

        connection.setConnectTimeout(30000);
        connection.setReadTimeout(90000);

        HttpURLConnection httpConnection = (HttpURLConnection) connection;
        httpConnection.setRequestMethod("GET");
        httpConnection.setRequestProperty("User-Agent", "JavaExampleClient/1.0.0");
        httpConnection.setRequestProperty("Accept-Encoding", "gzip");

        connection.setUseCaches(false);
        connection.setRequestProperty("Content-Language", "en-US");

        httpConnection.connect();

        int httpCode = httpConnection.getResponseCode();

        // getting the correct input stream
        if (httpCode == 200) {
          try (InputStream is = httpConnection.getInputStream()) {
            try {
              readStream(httpConnection, is, resume_ts);
            } catch (IOException ioe) {
              //stream or connection was interrupted, retry with next iteration
            }
          }
        } else if (httpCode == 503) {
          // the service is currently unavailable
          int secondsToWait = httpConnection.getHeaderFieldInt("Retry-After", 60);
          System.out.println("TEMPORARILY UNAVAILABLE");
          System.out.println("WAITING " + secondsToWait + "s UNTIL RETRYING");
          Thread.sleep(secondsToWait * 1000);
        } else {
          // when encountering an error, we exit loop
          try (InputStream is = httpConnection.getErrorStream()) {
            readError(httpConnection, is, httpCode);
          } catch (IOException e) {
            e.printStackTrace();
          } finally {
            finished = true;
          }
        }
      } catch (IOException ex) {
        // try again
        ex.printStackTrace();
        // sleep a minute
        Thread.sleep(60 * 1000);
      }
    }
    deleteStream();
  }

  private void readError(HttpURLConnection httpConnection, InputStream errorInputStream, int httpCode)
      throws IOException {
    ByteArrayOutputStream bos = new ByteArrayOutputStream();
    byte[] dataBuf = new byte[1024 * 1024];

    // read answer
    while (true) {
      int read = errorInputStream.read(dataBuf, 0, dataBuf.length);
      if (read == -1) {
        break;
      }

      bos.write(dataBuf, 0, read);
    }

    InputStream is = new ByteArrayInputStream(bos.toByteArray());
    if ((httpConnection.getContentEncoding() != null) && (httpConnection.getContentEncoding().equals("gzip")))
{
      is = new GZIPInputStream(is);
    }

    // read json using jackson json (another library may be used here)
    JsonFactory factory = new JsonFactory();
    ObjectMapper mapper = new ObjectMapper(factory);
    TypeReference<HashMap<String, Object>> typeRef = new TypeReference<HashMap<String, Object>>() {
    };
    HashMap<String, Object> o = mapper.readValue(is, typeRef);
  }

  private void readStream(HttpURLConnection httpConnection, InputStream inputStream, AtomicLong resumeTs)
      throws IOException {
    // reading the stream and invoking the listener
    InputStream is = inputStream;
    if ((httpConnection.getContentEncoding() != null) && (httpConnection.getContentEncoding().equals("gzip")))
{
      is = new GZIPInputStream(is);
    }

    BufferedReader reader = new BufferedReader(new InputStreamReader(is, "UTF-8"), 100);

    String line;
    while ((line = reader.readLine()) != null) {
      // parse json (use an available json parser)

      // skip empty lines
      if (line.isEmpty()) {
        continue;
      }

      JsonFactory factory = new JsonFactory();
      ObjectMapper mapper = new ObjectMapper(factory);
      TypeReference<HashMap<String, Object>> typeRef = new TypeReference<HashMap<String, Object>>() {
      };
      HashMap<String, Object> o = mapper.readValue(line, typeRef);

      Object oType = o.get("chunk_type");
      if (oType != null && oType instanceof String) {
        String type = (String) oType;
        switch (type) {
          case "CT_ERROR":
            Map<String, Object> errorChunk = getAsMap(o, "chunk_error");
            handleStreamError(errorChunk);
            break;
          case "CT_CONTROL":
            Map<String, Object> controlChunk = getAsMap(o, "chunk_control");
            if (controlChunk != null) {
              Long timeframeStart = getAsT(controlChunk, "timeframe_start", Long.class);
              if (timeframeStart != null) {
                resumeTs.set(timeframeStart);
              }
            }
            handleStreamControl(controlChunk);
            break;
          case "CT_RESULT":
            Map<String, Object> resultChunk = getAsMap(o, "chunk_result");
            handleStreamResult(resultChunk);
            break;
          default:
            unhandledStreamChunk(o);
            break;
        }
      } else {
        unhandledStreamChunk(o);
      }
    }
  }

  protected static Map<String, Object> getAsMap(Map<String, Object> o, String key) {
    if (o != null) {
      Map<String, Object> ret = null;
      Object oRet = o.get(key);
      if (oRet != null && oRet instanceof Map) {
        return (Map<String, Object>) oRet;
      }
    }
    return null;
  }

  protected static <T> T getAsT(Map<String, Object> o, String key, Class<T> clz) {
    if (o != null) {
      Map<String, Object> ret = null;
      Object oRet = o.get(key);
      if (oRet != null && clz.isInstance(oRet)) {
        return (T) oRet;
      }
    }
    return null;
  }

  protected void onInitializationError(Map<String, Object> errorData) {
    System.out.println("ERROR: " + errorData);
  }

  protected void handleStreamError(Map<String, Object> errorChunk) {
    System.out.println("ERROR: " + errorChunk);
  }

  protected void handleStreamControl(Map<String, Object> controlChunk) {
    System.out.println("CONTROL: " + controlChunk);
  }
```

```java
protected void handleStreamResult(Map<String, Object> resultChunk) {
  Map<String, Object> resultData = getAsMap(resultChunk, "data");
  Map<String, Object> entryData = getAsMap(resultData, "data");
  String url = getAsT(entryData, "url", String.class);
  System.out.println("RESULT: " + url);
}

protected void unhandledStreamChunk(Map<String, Object> unhandledChunk) {
  System.out.println("UNHANDLED: " + unhandledChunk);
}

protected void createStream() throws IOException {
  String _url = url + "/v1/stream/create?access_token=" + token;

  // connect
  URL request = new URL(_url);
  URLConnection connection = request.openConnection();

  connection.setConnectTimeout(30000);
  connection.setReadTimeout(90000);

  HttpURLConnection httpConnection = (HttpURLConnection) connection;
  httpConnection.setRequestMethod("POST");
  httpConnection.setRequestProperty("User-Agent", "JavaExampleClient/1.0.0");
  httpConnection.setRequestProperty("charset", "utf-8");
  httpConnection.setDoOutput(true);
  httpConnection.setDoInput(true);
  connection.setUseCaches(false);
  connection.setRequestProperty("Content-Language", "en-US");

  DataOutputStream wr = new DataOutputStream(connection.getOutputStream());

  JsonNodeFactory factory = JsonNodeFactory.instance;
  ObjectNode on = factory.objectNode();
  on.put("streamid", stream_id);

  System.out.println(on.toString());

  wr.writeBytes(on.toString());

  wr.flush();
  wr.close();

  httpConnection.connect();

  int httpCode = httpConnection.getResponseCode();

  if (httpCode != 200) {
    System.out.println("ERROR");
```

```
      System.out.println(IOUtils.toString(httpConnection.getInputStream(), "UTF-8"));
    } else {
      System.out.println("CREATED");
    }
  }

  protected void deleteStream() throws IOException {
    String _url = url + "/v1/stream/s/" + stream_id + "/delete?access_token=" + token;

    // connect
    URL request = new URL(_url);
    URLConnection connection = request.openConnection();

    connection.setConnectTimeout(30000);
    connection.setReadTimeout(90000);

    HttpURLConnection httpConnection = (HttpURLConnection) connection;
    httpConnection.setRequestMethod("DELETE");
    httpConnection.setRequestProperty("User-Agent", "JavaExampleClient/1.0.0");
    httpConnection.setRequestProperty("charset", "utf-8");
    httpConnection.setDoOutput(true);
    httpConnection.setDoInput(true);
    connection.setUseCaches(false);
    connection.setRequestProperty("Content-Language", "en-US");

    httpConnection.connect();

    int httpCode = httpConnection.getResponseCode();

    if (httpCode != 200) {
      System.out.println("ERROR");
      try {
        System.out.println(IOUtils.toString(httpConnection.getInputStream(), "UTF-8"));
      } catch (Exception e) {
        e.printStackTrace();
      }
    } else {
      System.out.println("DELETED");
    }
  }
```

Throubleshooting

Error Codes

http code	status code	message	description
200	0	OK	Default answer
500	1	Internal Server Error	An unexpected exception was encountered.
500	2	Search Execution Exception	An unexpected exception was encountered. Related to the search
400	3	Parameter Missing	Required parameters are missing. The missing parameters are provided in key 'params'.
400	4	Error in query	Could not parse query. The details can be found under 'details'.
400	5	Invalid parameter value	A parameter has an unacceptable value. The parameter is listed under 'param' and the details under 'details'.
401	7	Invalid, missing or inactive access token	The access token is either missing or the provided value is invalid.
401	8	Call limit exceeded for this endpoint	The called endpoint has a limited call frequency, the values should be cached by the client.
401	9	No credits left.	The account ran out of credits.
403	10	API application is inactive	The API account is inactive. 'appId' gives the id of that account.
403	11	No such application linked	The provided id is not linked in the API to any project or application.
403	12	Linked application inactive or deleted	The linked application is inactive or deleted.
403	13	Access denied: Insufficient access rights.	The used access token does not have enough access rights. 'rights_req' will list the required access rights, 'rights_got' lists the access rights provided by that access token.
404	15	Wrong stream id. No such stream defined.	A non existing stream was accessed.
400	16	Invalid operation on document	The search document modification operation is not supported. 'reason' and 'details' will provide more information.
400	17	Could not parse json	The JSON that was passed via POST could not be properly interpreted (it was not in the expected format).
400	18	Invalid operation on stream	Modifying a stream failed. See 'reason' for details.
403	19	Number of rules to set exceeds maximum number of rules	Exceeded the maximum allowed rules for this API account. 'number_max' is the limit, 'number_available' how many we can save and 'number_saving' the number we tried to save
403	20	Cannot create any more streams	Exceeded maximum amount of streams ('number_max')
403	21	A stream with this name already exists	The stream 'streamid' is already defined.

http code	status code	message	description
`403`	`22`	Number of sources to set exceeds maximum number of sources	Exceeded the maximum allowed sources (whitelist or blacklist) for this API account. 'number_max' is the limit, 'number_available' how many we can save and 'number_saving' the number we tried to save.
`403`	`23`	Stream has no rules defined	Exception when trying to stream with a stream that has no rule defined.
`403`	`24`	Stream got disconnected because newer stream running	A new stream (same streamid) is connected, so the old stream will be disconnected.
`403`	`25`	Stream got disconnected	The stream was disconnected due to the given reason.
`404`	`26`	Endpoint or action not found	The called endpoint was not found.
`403`	`27`	Connection is not secure, must use HTTPS	Authentication API endpoints need to be called using HTTPS.
`404`	`28`	User was not found in this application	This user id does not exist or is not linked to this project.
`403`	`29`	Access to this project is forbidden	This project can not be accessed with the given access_token.
`429`	`30`	Limit of maximum concurrent streams reached	Too many streams running in parallel for this account.
`404`	`31`	Could not find rule with id	A rule with the given id could not be found.
`404`	`32`	Could not find panel with id	A panel with the given id could not be found.
`403`	`33`	Panel is still referenced	This panel could not be deleted, it is still used in a stream.
`505`	`-`	HTTP Version Not Supported	The Talkwalker Streaming API supports HTTP 1.1 or newer.
`400`	`34`	Url is malformed	The given URL for channel monitoring is malformed
`400`	`35`	Could not execute action in Talkwalker	Error in connecting to a Talkwalker project
`403`	`36`	Access prohibited	Access prohibited due to access restriction settings

Error Handling

Streaming API

Resuming a disconnected stream

A stream can be disconnected for several reason: given maximum of hits (`max_hits`) reached, `stream_stop` reached, no credits left, server issues or connection problems. To resume a disconnected stream, set the parameter `stream_resume` to the start timestamp ('timeframe_start') of the last `CT_CONTROL` chunk. Since the results in a timeframe are not sorted, the streaming of the entire timeframe has to be restarted to make sure that no documents are lost.

```
curl
https://api.talkwalker.com/api/v2/stream/s/teststream/results?access_token=demo&stream_resume=1388534400000
```

The Streaming API returns different results for the same topic than the Talkwalker application.

Possible reasons:

Different queries or source filters:

Use `https://api.talkwalker.com/api/v2/stream/s/<stream_id>?access_token=demo&pretty=true` to make sure that no additional rules and source blacklists are set.

Documents are streamed at indexation time

Talkwalker finds most documents briefly after they were created, at this moment they are added to Talkwalker, and streamed via the API. Documents that are found later (i.e. some time after they were published on the original webpage), will be added to Talkwalker with their original publication time (timestamp the `published` field) along with the documents that were found earlier. In the Streaming API they only appear at the moment they were found (timestamp in 'search_indexed' field).

- Solutions:
  - with a query on `published` (`published:>1388534400000 AND published:<1388544400000`) a stream with a start point (`stream_resume`) of the beginning of the time range and a stop point (`stream_stop`) equal to the current time returns the same results as Talkwalker.
  - when adding the streamed results to a local database, you can group them later by the value in the `published` field.

Time zones

Timeranges in the Talkwalker application relative to the timezone set up under *General Settings - Project display options - Time zone* , while the Talkwalker API uses Unix Time (Epoch Time) in milliseconds (no time zones). This can make results, that are equal, appear to be different in the API.

No maximum of documents in the current month

While the Talkwalker application applies a maximum of found documents per month, the Talkwalker API returns all documents that can be found for th current given month. When the API is used with a Talkwalker project, the full project history is available.