Quant Engine API Endpoint

URL https://api.factset.com

Quant Engine API is an API for retrieving quant style datasets from FactSet content databases. A quant style dataset is created by generating a set of dates, a universe on each date, and then fetching a “cube” of content via screening expressions, FQL expressions, or universal screen parameters for each universe on each date.

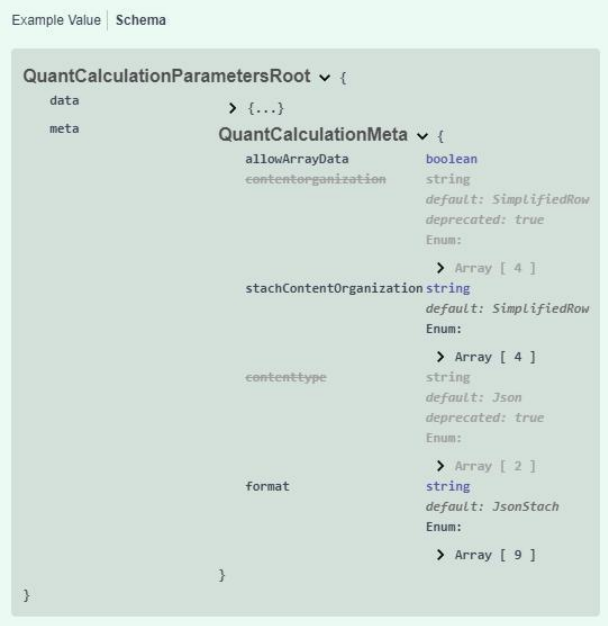
Depending on the size of the universe, number of dates and formulas, the data set may take many seconds, minutes, or even hours to generate. Therefore, the Quant Engine API uses an asynchronous request/response pattern.

There is a data "cube" creation endpoint, a status endpoint, and two endpoints for retrieving the data. One of the retrieval endpoints fetches the data part, which contains the universe and formula data, and the other fetches the info part, which contains error, performance, and other information about the results.

Data "Cube" Creation Endpoint

POST to /analytics/engines/quant/v3/calculations

- Body is a json object described in the Swagger schema
- Note that the request body’s “meta” parent object has a “format” field used to specify the result format. If no value is passed in, the result will default to JsonStach. Refer to the documentation here for more information on FactSet’s STACH format.



- If feather is specified, the response will be an application/octet-stream binary response with the data encoded in the Apache PyArrow Feather format. It is meant for consumption by clients than can convert this to a Python Pandas or R Data Frame

- If JsonStach is specified, the response will use FactSet's STACH format to represent the data and info portion of the Data "Cube".

- Optional cache-control header that accepts the max-stale directive (see the section below for more information regarding cache-control)

- Maximum 50 POST requests allowed in a 5 second window for each API. The same can be verified using the various Rate-Limit headers available in the API response.
  X-FactSet-Api-RateLimit-Limit - Number of allowed requests for the time window.
  X-FactSet-Api-RateLimit-Remaining - Number of requests left for the time window.
  X-FactSet-Api-RateLimit-Reset - Number of seconds remaining till rate limit resets.

- Responds with:

- A 202 if the request was accepted for processing

- A 201 if the request includes 1 calculation unit and finishes within a short period of time

- A 400 if the request object contains invalid calculation parameters

- Refer to the Swagger spec for information on other possible response codes and their schema

- The Location header contains the URL of the status resource

- Response body contains a resource status json object

- The resource status json object contains the calculation “id” attribute used to poll for status and retrieve the results.

- Refer to the Swagger spec for more information on the response schema

PUT to /analytics/engines/quant/v3/calculations/{id}

- Same functionality as the POST endpoint, however this endpoint also allows users to create custom calculation ids via the required “id” path parameter. It creates/updates and runs the calculation specified in the request body. A user can then poll a pre-constructed status URL and fetch results from a pre-constructed results URL.

- Sample workflow:
- PUT to /analytics/engines/quant/v3/calculations/**jan-dataset**

- GET to /analytics/engines/quant/v3/calculations/**jan-dataset**/status

- GET to /analytics/engines/quant/v3/calculations/**jan-dataset**/units/{unitId}/result

- Same cache-control functionality, request/response body schema, and headers as the POST request.

- Refer to the Swagger spec for more information on the response schema

Data "Cube" Status Endpoint

GET to /analytics/engines/quant/v3/calculations/{id}/status

- Gets status of calculation request
- Required “id” path parameter
- Responds with:
- A 200 once the calculation has finished
- A 202 if the calculation is not yet completed
- A 200 if the calculation has one unit and is completed with an error.
- A 400 if an invalid id is provided
- Refer to the Swagger spec for information on other possible response codes and their schema
- The Location header contains the URL of the results resource

Data "Cube" Output Endpoints

GET to /analytics/engines/quant/v3/calculations/{id}/units/{unitId}/result for Data

- This endpoint fetches the data results, which contains the universe and formula data
- The response contains a DATES table listing the dates, a FORMULAS table listing the formula UUIDs, and several <DATE>_<DATA|INFO>_<UUID> tables containing the data or the info section, depending on which endpoint was queried. Additionally, the data tables contain a STACH MetaDataItem describing the shape of the data in each table (recall that some formulas may return arrays for each individual (date, identifier) combination, but the tables store it as a flat array).
- Required “id” and “unitId” path parameters provided from the location header in the Get Quant Engine calculation status by id endpoint
- Responds with:
- A 200 once the calculation has completed.
- Returns response in the format specified in the “format” field of the “meta” parent object.
- A 400 if an invalid id is provided
- Refer to the Swagger spec for information on other possible response codes and their schemas

GET to /analytics/engines/quant/v3/calculations/{id}/units/{unitId}/info for Info

- This endpoint fetches the calculation result info, which contains error, performance, and other information about the results.
- Required “id” and “unitId” path parameters provided from the location header in the Get Quant Engine calculation status by id endpoint
- Responds with:
- A 200 once the calculation has completed.
- Returns response in the format specified in the “format” field of the “meta” parent object.
- A 400 if an invalid id is provided
- Refer to the Swagger spec for information on other possible response codes and their schemas

GET to /analytics/engines/quant/v3/calculations for all calculations

- This endpoint returns all calculation requests. The data is sorted in descending order by request time and by default one page will fetch 20 calculations.
- Requires page number, default = 1
- Responds with:
- A 200 with a list of all the calculation requests in a page.
- A 400 if an invalid page number is provided, example = 0.
- Refer to the Swagger spec for information on other possible response codes and their schemas

Common Error Scenarios

The error handling strategy depends on the severity of the error.

A few examples of such errors are:

- Syntax and semantic errors in the request body, such as invalid json and misspelled attribute names
- Errors when fetching critical data from content, such as the universe or the set of dates
- Authentication errors

Some errors are specific to a particular formula, and do not affect the rest of the data. In such cases the caller is responsible for querying the info resource for any errors associated with a particular formula. Some common examples of such errors are:

- Typos in formula text, e.g. P_PRICR instead of P_PRICE
- Referencing a non-existent universal screen parameter
- Using an unrecognized identifier when defining a static universe

Cache-Control

- The cache-control header gives clients the ability to fetch pre-calculated results for a specified period (up to 12 hours), or explicitly request an ad hoc calculation.
- *By default, all results are stored for 12 hours.*
- Setting "Cache-Control": "max-stale=<staleness limit in seconds>" allows clients to fetch precalculated results with any subsequent POST requests if they were last calculated within the staleness limit. The max cache-control value is "max-stale=43200" (12 hours). Note that by default, max-stale will be set to 0.
- Once set, the API will check to see if the stored results are within the staleness limit.
    - If they are within the limit AND the request contains only 1 calculation unit:
        - ✦ Then, the request will return a "201" response code with the results in the body of the POST/PUT response AND the status polling URL in the Location header. This allows for a quick response and eliminating unnecessary points accrual
    - If they are within the limit AND the request contains multiple calculation units:
        - ✦ Then, the request will return a "200" response code with the result URLs in the body of the POST/PUT response. This allows for a quick response and eliminating unnecessary points accrual
    - If the results are not within the staleness limit, a brand-new calculation request will be triggered to get the latest results. Note that the caching period starts after the individual calculation unit has completed
- To immediately request the latest results, override the cache by setting "max-stale=0" in the Cache-control header parameter.
- Note that the max-stale value only controls when a new calculation will be triggered with any subsequent POST or PUT requests (provided that the request parameters stay the same). All results generated will always be stored for 12 hours, the max-stale value will not affect this.
- Max-stale is the only cache-control directive supported by the API.

How to use FQL formulas

FactSet's FQL formulas vary widely in the number and type of arguments they expect. They also produce distinct kinds of results depending on the formula.

Because the data is organized as a cube indexed by an identifier, date, and formula, the FQL expressions must be compatible with this structure. To be used with the Quant Engine API "cube" endpoint, each FQL formula must satisfy several conditions:

- Must take a date range argument, or the date must be implicit in the formula's behavior (e.g., it always uses the NOW date).
- Must operate on a universe of identifiers (e.g., it cannot be a formula that produces economic data for a country).
- Must produce a scalar or 1d array of data for each security on each date. If there are formulas that produce higher dimensional data, they are not (currently) supported.

The position of the start, end, and frequency date arguments is different for different FQL formulas. The Quant Engine API "cube" endpoint must be told the position of these arguments. The #DATE and #FREQ placeholders are used to accomplish this. When the Quant Engine API sees these placeholders in a formula, it will substitute them for the appropriate dates and frequency.

Some examples of valid "fqlExpression" definitions.

Any "typical" content code works. Here, the formula works without an explicit date argument and needs no placeholders to be evaluated correctly.

```
{
  "source": "fqlExpression",
    "expr": "FG_GICS_SECTOR",
    "name": "Sector (FQL)"
}
```

Most formulas require a date argument, but the position varies. Here the date arguments start at position 0.

```
{
  "source": "fqlExpression",
    "expr": "P_PRICE(#DATE,#DATE,#FREQ)",
    "name": "Price (FQL)"
}
```

Versus this formula where the date arguments start at position 1.

```
{
    "source": "fqlExpression",
      "expr": "FF_SALES(,#DATE,#DATE,#FREQ)",
      "name": "Sales (FQL)"
}
```

Formulas above produce a scalar value for each security on each date. You can use array valued FQL formulas as well.

```
{
  "source": "fqlExpression",
   "expr": "OS_TOP_HLDR_POS(ALL,#DATE,#DATE,M,,S,SEC)",
   "name": "Top Holders (FQL)"
}
```

Notice that in the above definition the frequency was hard coded as 'M' instead of letting the Quant Engine API substitute the frequency used in the "dates" definition. This is allowed, but proceed with caution, make sure that you have a good reason for that and understand the data that will be generated.

A note on NaN values

FactSet content databases use FactSet specific NaN sentinels which are not recognized by other programming languages and environments such as Python Pandas. The Quant Engine API does not currently convert them to a universal format, largely because aside from IEEE 754 floating point NaNs there is no agreed upon standard for integer and string NaNs. Additionally, there are some limitations in the current data serialization format we use for Python. In future releases we will provide an option to standardize NaNs to a particular computing environment such as Python Pandas.

As a result, be aware that the following values are FactSet's NaN sentinels.

```
Float: -1.0e+22
Float: -1.5e+21
Integer: -2147483648
String: @NA
String: $$FDS_US_@NA$$
```

A note on data set sizes and performance

The current implementation of Quant Engine API constructs data sets in-memory. In practice this means that we do not support arbitrarily large data sets. To achieve good performance, keep your universe size to several thousand identifiers, several hundred dates, and a few dozen formulas. As we scale up the system, these limitations will be gradually removed.

Optimization notes

Prefer screening expressions to FQL formulas. Only use FQL expressions if the data is not available via screening codes.

Be aware that data sets that use universal screen documents to generate the universe and retrieve screen parameters are orders of magnitude slower than universes generated by simple screening expressions.

Troubleshooting

The following steps are recommended to troubleshoot errors from any of the different APIs:

- Calculation specific endpoints (POST, PUT, GET Status, GET Results, GET All Calculations)
    - Record the below response headers so that FactSet's API support team can analyse your specific request/response:
        - x-factset-api-calculation-instance-id
        - x-factset-api-calculation-id
        - x-datadirect-request-key
    - Record the response body when the response is an error response. All HTTP status codes equal to and greater than 400 are considered error responses.
    - Reach out to your account team with the above information for assistance.