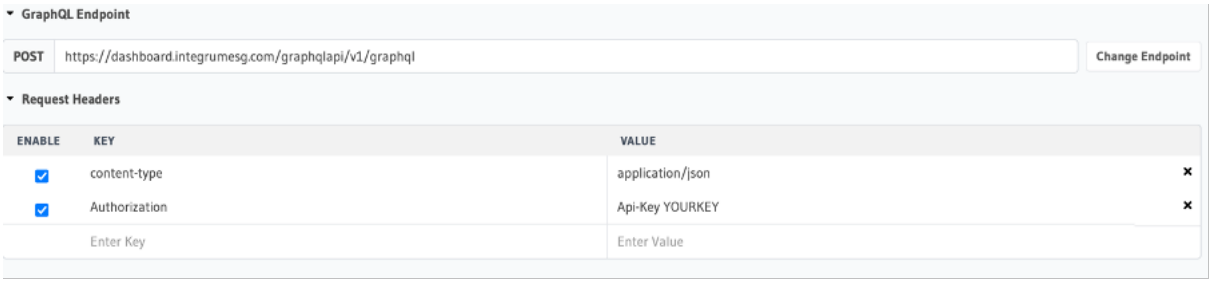
Integrum ESG GraphQL API Manual V2.7 Jul 2024

A step-by-step example of building a query in GraphiQL

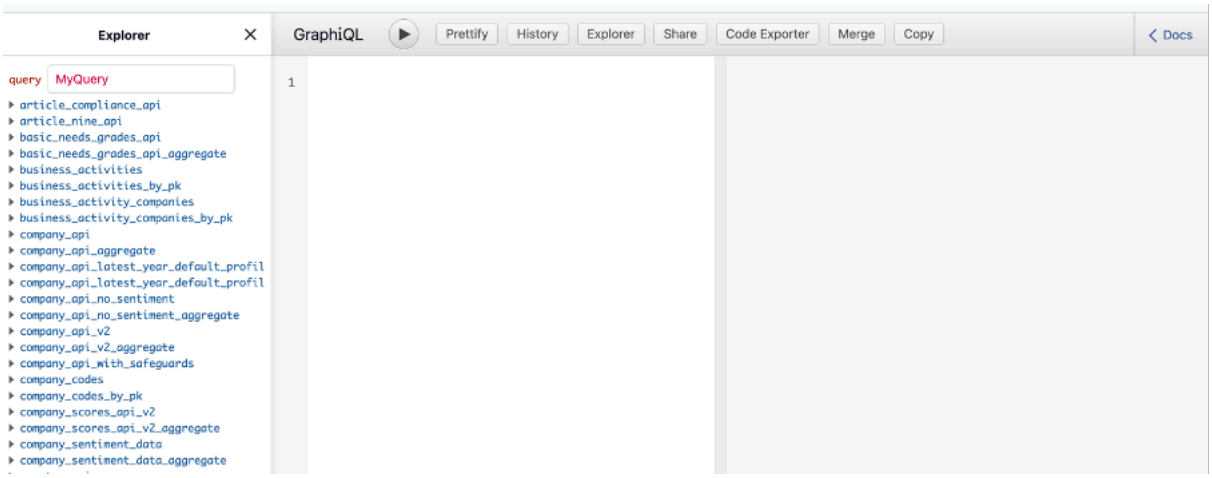
Step 1: Use GraphiQL

Go to the public GraphiQL webpage: https://cloud.hasura.io/public/graphiql
Enter our endpoint: https://dashboard.integrumesg.com/graphqlapi/v1/graphql
Add YOUR API key which you get from the steps below:

1. Log in to the Integrum ESG Dashboard: https://dashboard.integrumesg.com/dashboard/
2. Click your initials in the top right-hand corner
3. Click "Account"
4. Click "Generate API Key" on the dashboard account settings page



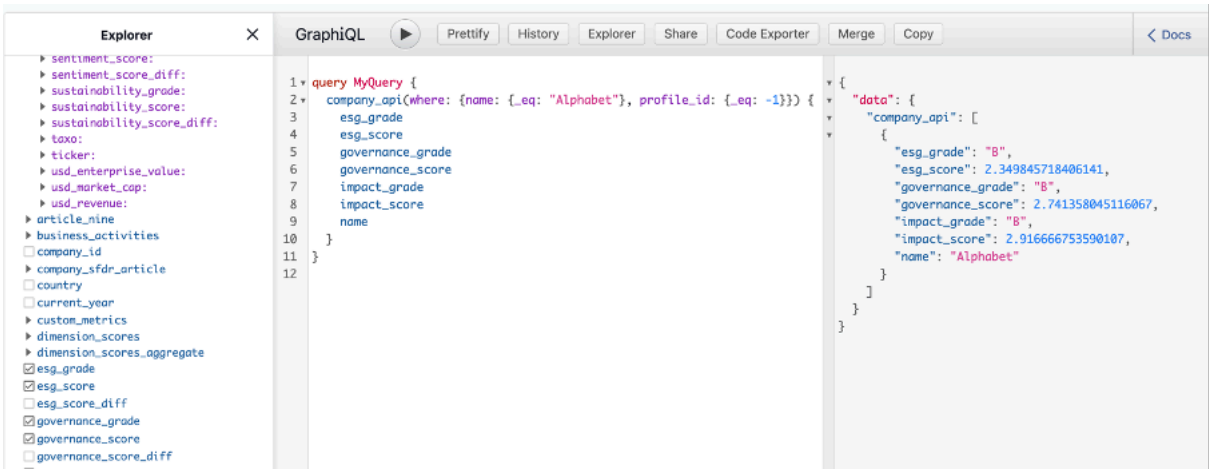
Now the explorer shows the APIs available on the left-hand side



Step 2: The top-level company query

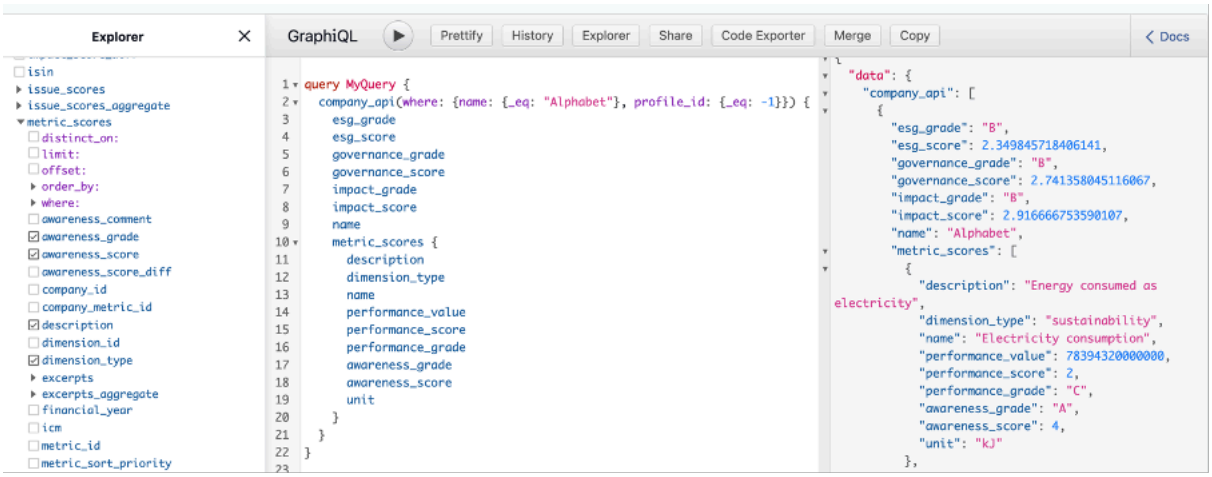
Get the top-level scores for Alphabet from the company_api. Remember that scores need to have a `profile_id` specified so we use the default equally weighted profile with the "magic" id of –1

We built this by clicking the selections in the company_api – but they do not all fit in the screenshot.



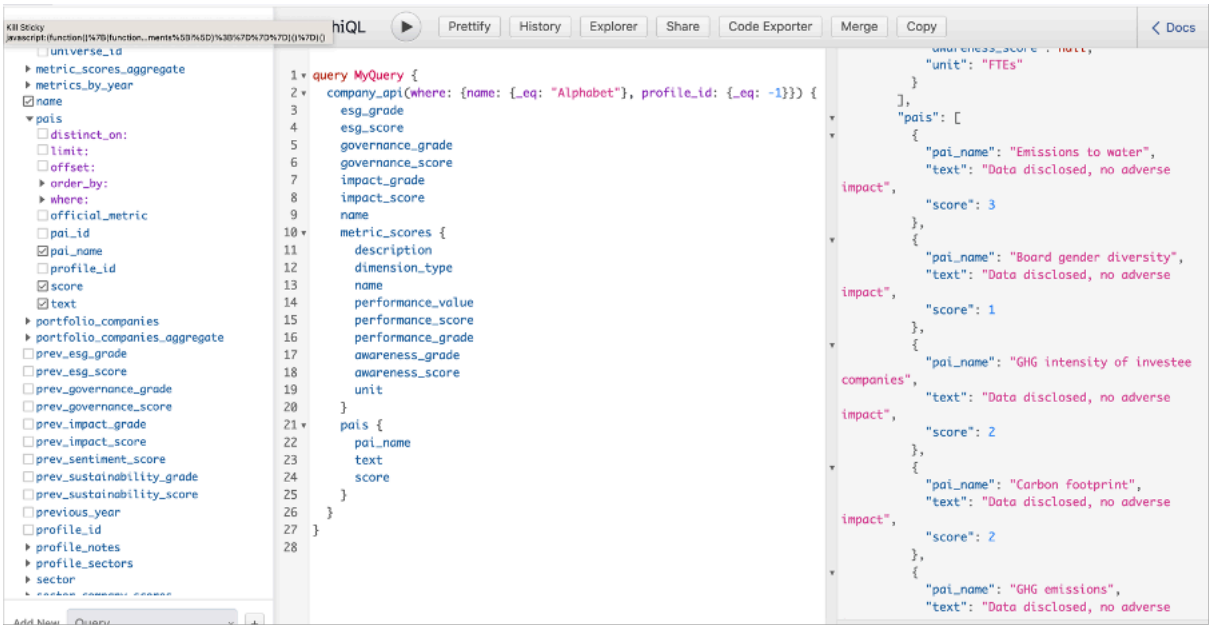
Step 3: Metrics

Now select the metrics from the metric_scores nested element



Step 4: PAIs

Now select the PAIs from the nested pais element



Step 5: Overall SFDR Article

Now select the overall SFDR Article from the metrics page, which is another nested element – `company_sfdr_article`.

Here is the resulting query:

```
query MyQuery {
  company_api(where: {name: {_eq: "Alphabet"}, profile_id: {_eq: -1}}) {
    esg_grade
    esg_score
    sustainability_grade
    sustainability_score
    governance_grade
    governance_score
    impact_grade
    impact_score
    name
    metric_scores {
      description
      dimension_type
      name
      performance_value
      performance_score
      performance_grade
      awareness_grade
      awareness_score
      unit
    }
    pais {
      pai_name
      text
      score
    }
    company_sfdr_article {
      article
    }
  }
}
```

Detailed Sentiment API

The detailed sentiment (down to 15-minute granularity) is available with the:

`company_sentiment_data`

root element.

There is pagination on this table – with a limit of 10000 rows.

The API results can be paginated using the limit and offset parameters.

See the example Python program in the appendix.

Sentiment Granularity

The 15-minute sentiment data has several months' history. There is another year or so of weekly granularity data.

Sentiment Aggregation

The `sentiment_api_v2` offers a rolling seven-day aggregation of sentiment.

```
query MyQuery {
  sentiment_api_v2(wher
  ...BasicNeedse: { company: { name: { _eq: "Diageo" } } }) {
    positive_comments
    neutral_comments
    negative_comments
    end_time
  }
}
```

PAI, Article 9, Taxonomy

Our mappings and assessments are also available via the API.

The taxo, article_nine and pais are array sub-elements nested under the company_api

The company_sfdr_article is an object relationship in the company_api

This example shows fetching each of these elements by adding taxo to the step-by-step example.

```
query MyQuery {
  company_api(where: { company_id: { _eq: 1 }, profile_id: { _eq: -1 } }) {
    esg_grade
    esg_score
    company_id
    country
    isin
    name
    taxo {
      contribution
      eu_taxo_aligned
      full_text
      harm
      taxo_objective
      text
    }
    article_nine {
      article_9_requirement
      contribution
      full_text
      harm
      name
      score
      text
    }
    pais {
      pai_name
      score
      text
    }
    company_sfdr_article {
      article
    }
  }
}
```

V2 API

The `company_api` root element was designed for surfacing scores and to handle a wider range of use cases, we also have a `company_api_v2` root element with a more general nesting structure and including sentiment time-series (aggregated by week)

Sentiment data is available for a different (wider) population of companies as there are data distribution restrictions on the scores API where source data from other providers has been involved.

This can be useful as it unifies the varying granularity. Other aggregations should be done client side.

Metrics by year and raw metrics

The `company_api` and the metrics scores nested within it behave like the dashboard and has the current and previous year scores for each company, where the actual year that it relates to may vary across companies depending on their reporting dates.

In some integration cases, it can be better to use the actual year, so the `metrics_by_year` nested element in the company API has individual entries for each year

```
query MyQuery {
  company_api(where: { company_id: { _eq: 1 }, profile_id: { _eq: -1 } }) {
    esg_grade
    esg_score
    company_id
    country
    isin
    name
    metrics_by_year(where: { metric_id: { _eq: 44 } }) {
      awareness_comment
      awareness_grade
      awareness_score
      description
      dimension_type
      financial_year
      name
      performance_comment
      performance_grade
      performance_score
      performance_value
      unit
    }
  }
}
```

The `metrics_by_year` still only covers the current and previous year, as the scores are computed dynamically (according to the profile) and we have the same coverage as the dashboard.

A deeper history is available for the underlying metric values. There are no dimension or company-level scores calculated – so no profile is required – and so it is exposed as a v2 API

`raw_metrics_api_v2` and nested under `company_api_v2` as raw_metrics

Here is an example of fetching all of the metrics for a company for a particular (non-current) year.

```
query MyQuery {
  company_api_v2(where: { id: { _eq: 1 } }) {
    raw_metrics(where: { financial_year: { _eq: 2018 } }) {
      awareness_comment
      awareness_grade
      awareness_score
      dimension_name
      dimension_type
      financial_year
      metric_description
      metric_display_dp
      performance_value
      scaled_performance_value
      metric_unit
    }
  }
}
```

Business Activities

The (excludable) business activities are nested under the company_api.

Example:

```
query MyQuery {
  company_api(where: { company_id: { _eq: 1 }, profile_id: { _eq: -1 } }) {
```

```
    country
    current_year
    esg_grade
    esg_score
    business_activities {
      business_activity {
        name
      }
    }
  }
}
```

Detailed Sentiment Python Example (with pagination)

```
import requests
import json
import pandas as pd

# can get the count with this

q1 = """
query MyQuery {
  company_sentiment_data_aggregate(where: { company: { name: { _eq: "Diageo" } } }) {
    aggregate {
      count
    }
  }
}
"""


# NOTE: {} in GraphQL are doubled here as we are putting this through the "format" function to inject
the

# offset and limit

q2 = """
query MyQuery {{
  company_sentiment_data(where: {{company: {{name: {{_eq: "Diageo"}}}}}}, limit: {0}, offset: {1})
{{
    company {{
      name
    }}
    time
    sentiment
    source
    text
  }}
}}
"""


url = 'https://dashboard.integrumesg.com/graphqlapi/v1/graphql'


def get_graphql(q):
    r = requests.post(
        url, headers={'Authorization': 'Api-Key XXXX'}, json={'query': q})
    print(r.status_code)
    json_data = json.loads(r.text)
    # print(json_data)
```

```
    return (json_data)

count_data = get_graphql(q1)
print(count_data)
limit = 10000
offset = 0
df = None

while True:
    q = q2.format(limit, offset)
    print(q)
    json_data = get_graphql(q)
    df2 = pd.json_normalize(json_data, record_path=[
                            'data', 'company_sentiment_data'])
    offset += limit
    s = json_data['data']['company_sentiment']
    print(len(s))
    df = pd.concat([df, df2])
    if len(s) == 0:
        break

print(df)
```

Portfolio and Codes Examples

We can create a portfolio in the Integrum ESG system by uploading it on the dashboard – or via the portfolio API – described in the Integrum ESG API Portfolio Management document.

An advantage of the manual creation method is that it looks up the given ISINs in a list of all known ISIN mappings and finds the appropriate company or country if it is covered in the system.

The result mapping can be queried in the GraphQL API

```
query MyQuery {
  portfolio_companies_api(where: { portfolio_name: {_ilike: "%MyPortfolio" } }) {
    name
    isin
    weighting
  }
}
```

In this example, this is the ISIN that was uploaded in the portfolio definition - and is often different from the primary ISIN in the company.

And we can use the nested items (like company_scores) from the portfolio companies:

```
query MyQuery {
  portfolio_companies_api(where: { portfolio_name: { _ilike: "%MyPortfolio" } }) {
    isin
    weighting
    company_scores(where: { profile_id: { _eq: -1 } }) {
      name
      esg_score
      esg_grade
    }
  }
}
```

Note that we do not need a `profile_id` in the where clause for the `portfolio_companies_api` as there are no scores involved - just weightings. But we do need the `profile_id` for the `company_scores`.

In V2.3 we have also added the full list of ISIN mappings to the V2 API:

```
query MyQuery {
  company_api_v2(where: { isins: { code: { _in: ["DE000DTR0CK8"] } } }) {
    isin
    name
  }
}
```

A detailed metric example

```
fragment Metrics on company_api {
  sustainability_score_diff
  governance_score_diff
  impact_score_diff
  company_id
  dimension_scores(order_by: {dimension_sort_priority: asc, dimension_type: desc}) {
    name
    dimension_type
    tag
    awareness_grade
    awareness_score
    awareness_score_diff
    performance_grade
    performance_score
    performance_score_diff
    dimension_grade
    dimension_score
    dimension_score_diff
    dimension_weighting
    weighted_dimension_score
    sector_scores {
      awareness_grade
      performance_grade
      dimension_grade
    }
    metric_scores(
      where: {profile_id: {_eq: $profileId}, name: {_in: $metricscore_name}}
      order_by: {metric_sort_priority: asc}
    ) {
      name
      financial_year
      sector_scores {
        awareness_grade
        performance_grade
      }
      excerpts(where: {company_id: {_eq: $companyId}}) @include(if: $option3) {
        page
        report_type
        image_url
        score
      }

    }
  }
}

query Option3($companyId: Int, $profileId: Int, $option3: Boolean = true, $metricscore_name:
[String]!) {
  company_api(
    where: {company_id: {_eq: $companyId}, profile_id: {_eq: $profileId}}
  ) {
    ...Metrics
```

```
  }
}
```

This example fetches detailed information, including excerpts and image screenshot url for a particual metric for a particular company.

The url needs to be prefixed with our S3 bucket address of 'https://s3.eu-west-2.amazonaws.com/integrum-production/' for fetching.

The example also makes use of graphql fragments which is a good way of building up reuseable elements of graphql queries.

Bulk Extract Examples

If you want to use the default equally weighted profile then the fastest way is to use the daily extract of scores under the default equally-weighted profile in `company_cube_history` The `record_date` element is the date of the extract.

Compression

A whole extract of companies and metrics runs to 100s MB so is large for individual requests.

Compression can be applied at the content level by requesting gzip as a content type.

This example shows getting an extract from a command line using `curl`. It could equally be done in Python using a streaming gzip module.

Note that we can nest the `company_cube_history` element under `company_api_v2` as we want to get the primary ISIN as it is useful to have an identifier for integration.

```
ENDPOINT=https://dashboard.integrumesg.com/graphqlapi/v1/graphql
q2="
query MyQuery {
  company_api_v2 {
    country
    current_year
    isin
    name
    company_cube_history(where: {record_date: {_eq: \\\"2024-01-18\\\"}, yeardiff: {_eq: 0}}) {
      awareness_grade
      awareness_score
      level
      level1
      level2
      level3
      metric_unit
      performance_grade
      performance_score
      performance_value
      record_date
      scaled_performance_value
      sd
      total_grade
      total_score
      weighted_dimension_score
    }
  }
}
"
eq=`echo "$q2" | tr '\n' ' '`
curl -H "Authorization:Api-Key XXXX" -H "Content-Type: application/json" -H "Accept-Encoding: gzip" -
X POST -d "{\"query\": \"$eq\"}" $ENDPOINT > bulk.json.gz
```

Multi-year History Extract

The company_api behaves the same in the dashboard in that it only calculates scores and grades for the current and previous years. The full history is available in the company_cube_history_all_years element. As noted in the API Guide the scores returned are slightly different as the peer comparisons are all of data in the same financial year.

The structure of the data is very similar to the cube structure above. The data can be extract raw:

```
query screenerQuery {
  company_cube_history_all_years
   {
    awareness_grade
    awareness_score
    company_id
    country
    financial_year
    level
    level1
    level2
    level3
    metric_unit
    name
    parent_sector
    performance_grade
    performance_score
    performance_value
    profile_id
    scaled_performance_value
    sd
    sector
    sector_id
    total_grade
    total_score
    universe_id
  }
}
"
eq=`echo "$q" | tr '\n' ' '`
curl -H "Authorization:Api-Key XXXX" -H "Content-Type: application/json" -H "Accept-Encoding: gzip" -
X POST -d "{\"query\": \"$eq\"}" https://dashboard.integrumesg.com/graphqlapi/v1/graphql >
bulk_history_raw.json.gz
```

Or nested under the company_api_v2 for more metadata:

```
query screenerQuery {
 company_api_v2 {
    country
    current_year
    isin
    name
  company_cube_history_all_years
   {
    awareness_grade
    awareness_score
    company_id
    country
    financial_year
    level
    level1
    level2
    level3
    metric_unit
    name
```

```
    parent_sector
    performance_grade
    performance_score
    performance_value
    profile_id
    scaled_performance_value
    sd
    sector
    sector_id
    total_grade
    total_score
    universe_id
  }
}
}
"
eq=`echo "$q" | tr '\n' ' '`
curl -H "Authorization:Api-Key XXXX" -H "Content-Type: application/json" -H "Accept-Encoding: gzip" -
X POST -d "{\"query\": \"$eq\"}" https://dashboard.integrumesg.com/graphqlapi/v1/graphql >
bulk_history.json.gz
```

Complete Proxy Data

As mentioned in the API guide it is not very efficient to try to get all of the score data for all of the proxy companies as each region/sector combination will (by definition) have the same data. In this case it's better to get the proxy scores separately from the proxy companies.

Firstly the proxy data using the cube api - and using compression as above

```
q="
query screenerQuery {
  proxy_cube_api(where: {profile_id: {_eq: -1}, yeardiff: {_eq: 0}}) {
    awareness_grade
    awareness_score
    country
    level
    level1
    level2
    level3
    metric_unit
    performance_grade
    performance_score
    performance_value
    scaled_performance_value
    sd
    sector
    total_grade
    total_score
    weighted_dimension_score
    yeardiff
  }
}
"
eq=`echo "$q" | tr '\n' ' '`
curl -H "Authorization:Api-Key XXX" -H "Content-Type: application/json" -H "Accept-Encoding: gzip" -X
POST -d "{\"query\": \"$eq\"}" https://dashboard.integrumesg.com/graphqlapi/v1/graphql
```

And then the proxyied companies with just the important metadata (name, ISIN, country and sector)

```
q="
query MyQuery {
  company_api_with_proxies(where: {profile_id: {_eq: -1}}) {
    country
```

```
    company_id
    sector { name }
    isin
    name
  }
}"
eq=`echo "$q" | tr '\n' ' '`
curl -H "Authorization:Api-Key XXX" -H "Content-Type: application/json" -H "Accept-Encoding: gzip" -X
POST -d "{\"query\": \"$eq\"}" https://dashboard.integrumesg.com/graphqlapi/v1/graphql
```