Querying XBRL reports submitted to the SEC

An introduction to the XBRL connector for JSONiq

Ghislain Fourny, 28msec <g@28.io>

Querying XBRL reports submitted to the SEC: An introduction to the XBRL connector for JSONiq

by Ghislain Fourny

Abstract

`us-gaap:StockholdersEquityAbstract` (and many more).

Table of Contents

List of Tables

List of Examples

Chapter 1. Introduction

Just a few years ago, most companies were generating internal or external reports using text processors and spreadsheets. This was extremely inefficient and expensive: Excel and Word files were flowing from mailbox to mailbox, information was rekeyed on the fly, copied over, reorganized manually. In addition to the amount of time spent on these processes, this was very error-prone. For a few years now, the XBRL standard has been gaining importance, allowing reports to be made in a unified format. Some regulating authorities such as the SEC even made it mandatory.

In business life, standardization matters more than perfection. While XBRL is not perfect, it is now a standard, and it does a fine job doing what it was designed for.

While this is a cost-sinking revolution, it is also a revolution about access to information: using latest database technologies, XBRL reports can be made *queryable*. In particular, it is possible to *validate* that the reports do not contain a certain number of errors, to *impute* non-submitted information, but also to *derive* new information using data at an unprecedented scale (across potentially all XBRL reports in the world, and much beyond).

NoSQL technologies

Many XBRL services providers store XBRL data in a relational database. However, very soon, the limits of homogeneous, flat tables are reached and slow down requests. Other services leverage the fact that the XBRL syntax is based on XML and store it, as is, in an XML database. Yet the raw XBRL format is not suitable for querying, or at least, for querying with reasonable performance: while XBRL uses the XML syntax, its data model is significantly different from that of XML, in at least the two following ways. If you leave tuples aside, there is nothing flatter than a physical fact. Networks that have no directed cycles (i.e., directed acyclic graphs, often even trees) can be stored in more optimal ways than raw XLink.

With NoSQL technologies, XBRL can be stored in an way that is optimized for hypercube querying. This is called NoLAP (NoSQL Online Analytical Processing). NoLAP extends the OLAP paradigm by removing hypercube rigidity and letting users generate their own hypercubes on the fly on the same pool of (fact) values. Very much like NoSQL technologies extend the SQL paradigm by replacing flat, homogeneous tables with heterogeneous, arborescent data.

The JSONiq language

JSONiq is the language used by 28msec's flagship platform, 28.io. It is a NoSQL query language that deals with heterogeneous, arborescent data. It extends JSONiq with a number of modules that make it convenient to access, simultaneously, many data sources (MongoDB databases, traditional relational databases, S3 storage on Amazon, Graph Databases, Cloudant and many more) and combine them seamlessly to extract information. This tutorial assumes that the user is, or will make himself, familiar with JSONiq. A JSONiq tutorial as well as a complete reference can be found on `http://www.jsoniq.org/`. JSONiq queries can be executed for free on the 28.io platform.

The XBRL data sources

We provide various data sources (SEC for various subsets of companies, FINREP, etc) available on the 28.io platform. These data sources expose to you the XBRL data stored in a NoSQL database (currently MongoDB), leveraging the NoLAP paradigm. Access to the data is done through the XBRL connector, which consists of JSONiq modules.

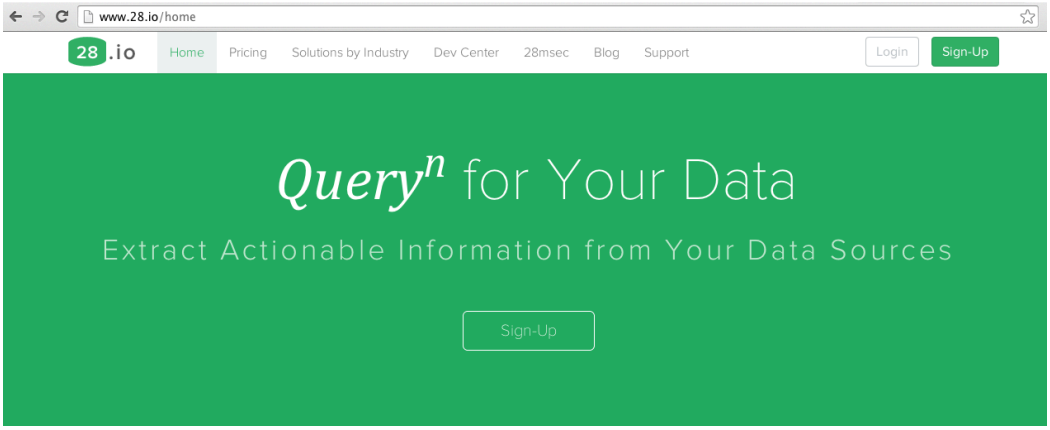
Documentation about these modules can be found at `http://www.28.io/documentation/latest/modules/bizql`.

How to run the queries

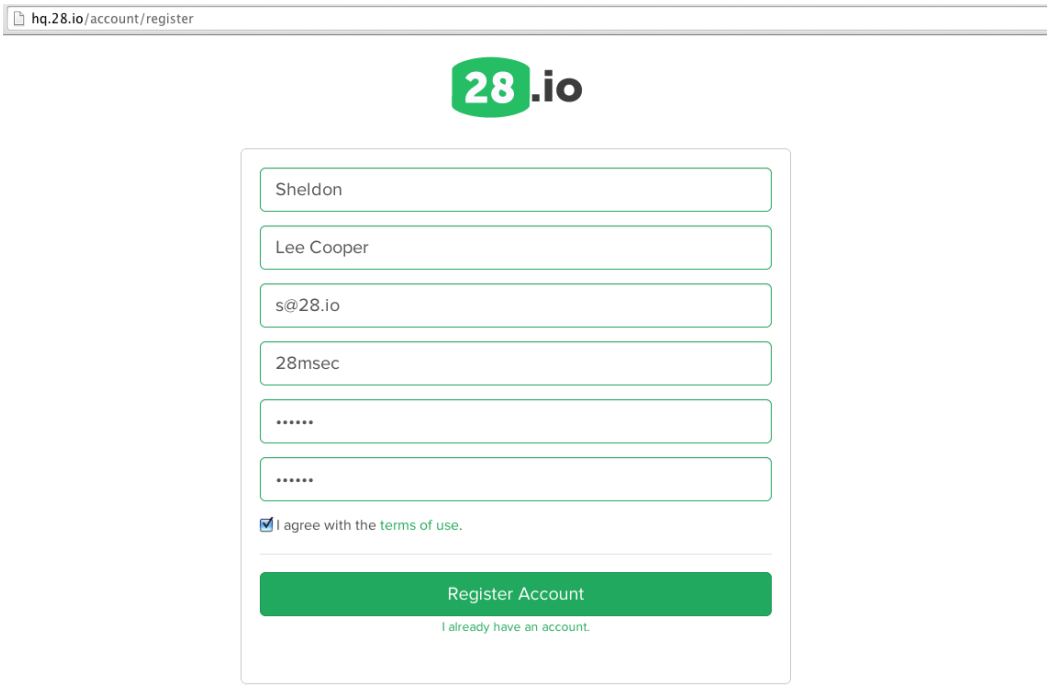
All queries shown in this tutorial can be run on our platform. You can register and use our platform for free, for a basic usage and limited to DOW30. Access to all 10-Q and 10-K filings (several thousands of companies over the last 3-4 years) is possible for a fee.

The following instructions should help you get started.

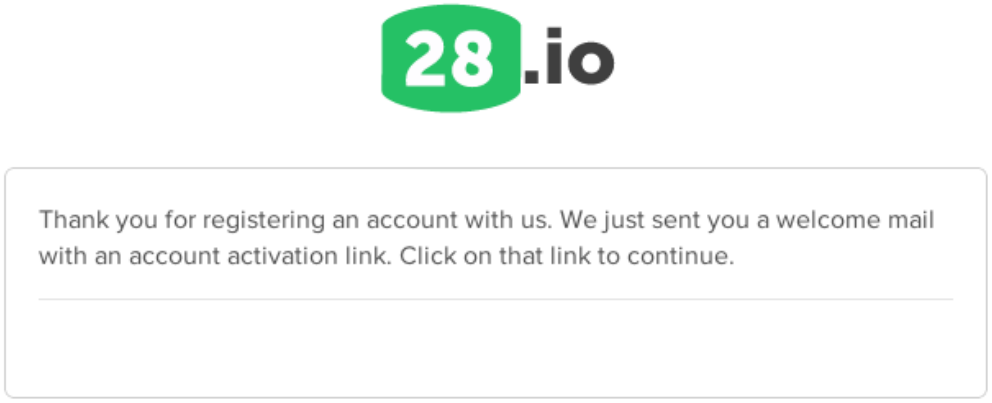
1. Go to the 28.io Platform Web site, `http://28.io/`



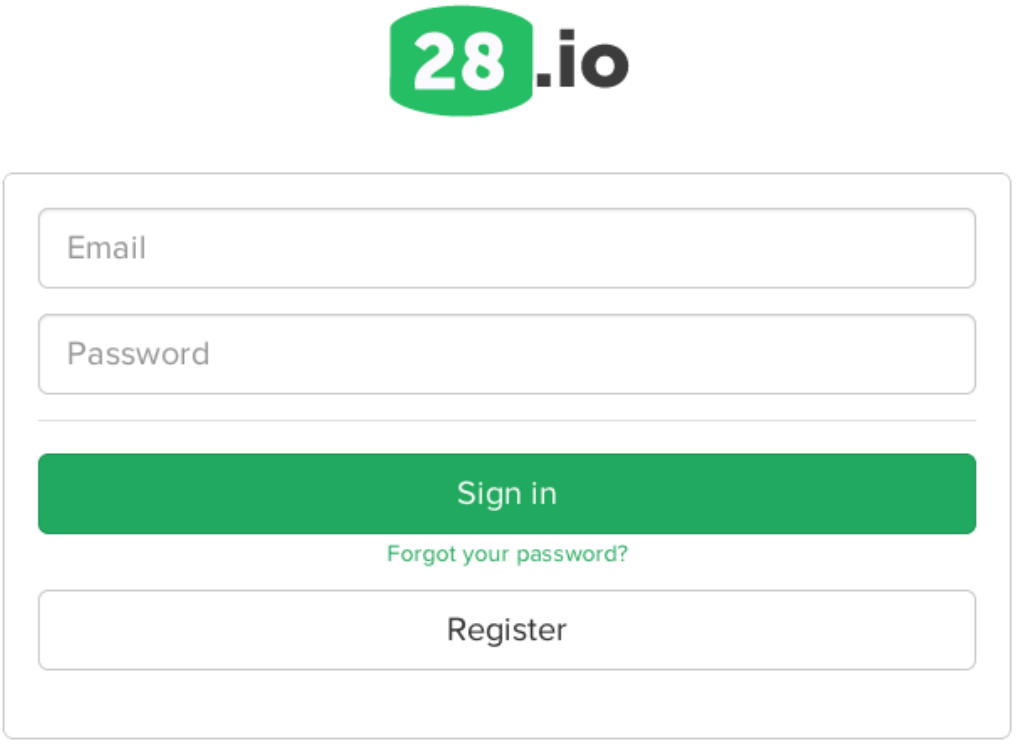
2. Click on Signup.



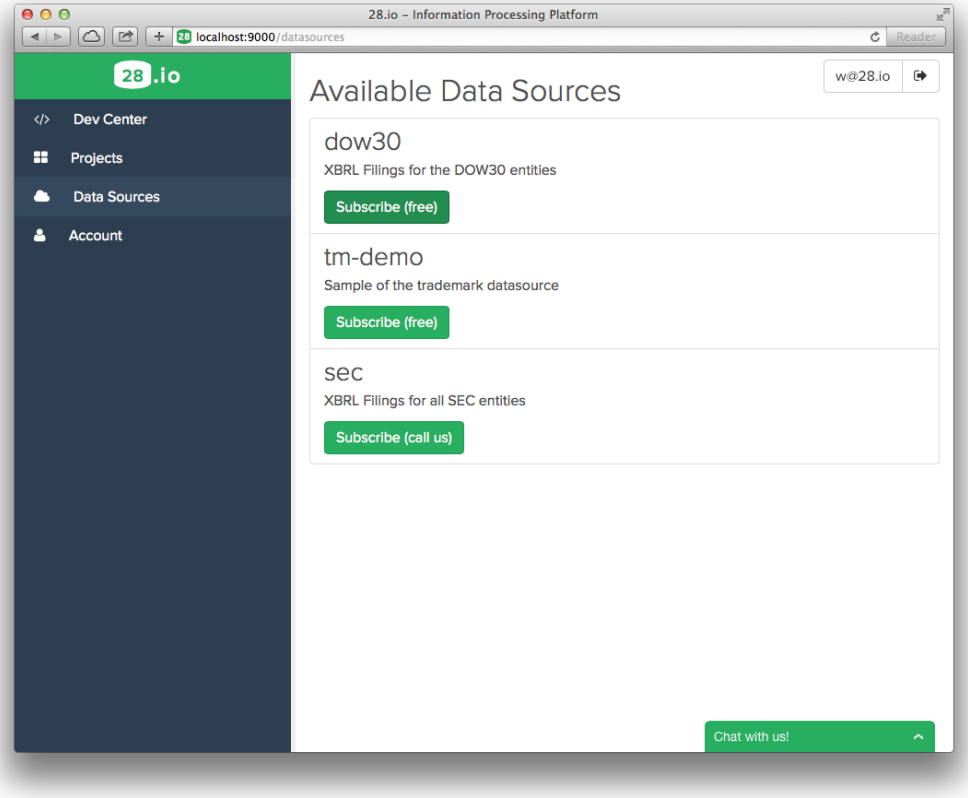
3. Enter your information as well as desired credentials, and confirm.



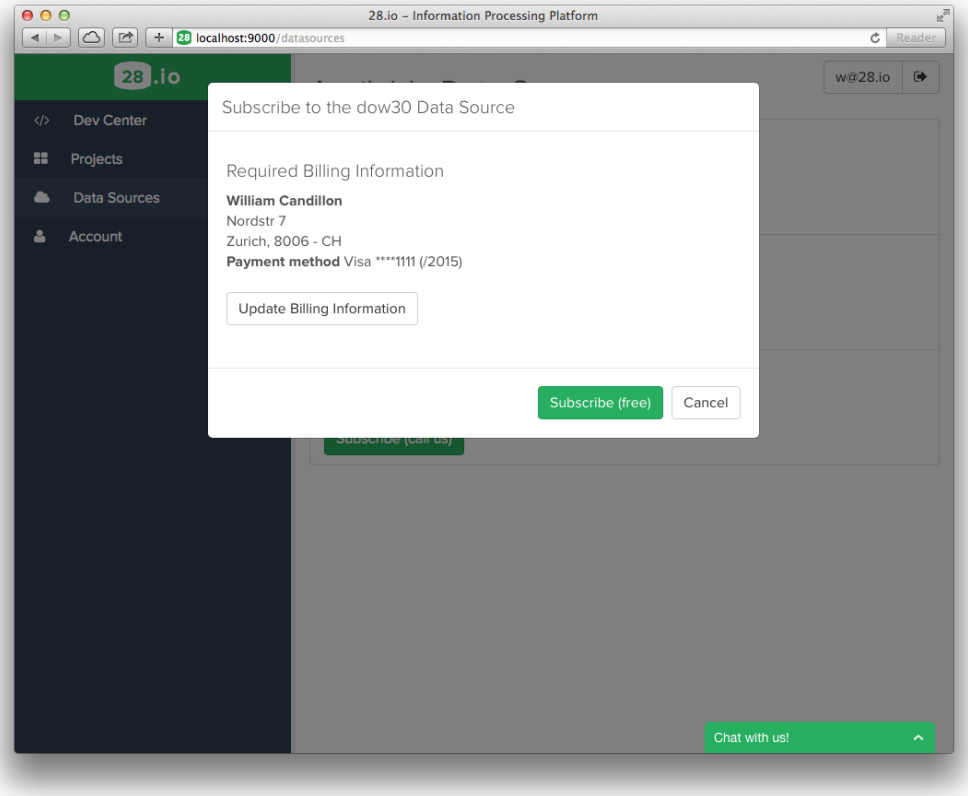
4. Check your e-mail (including spam folder) and validate your e-mail address.



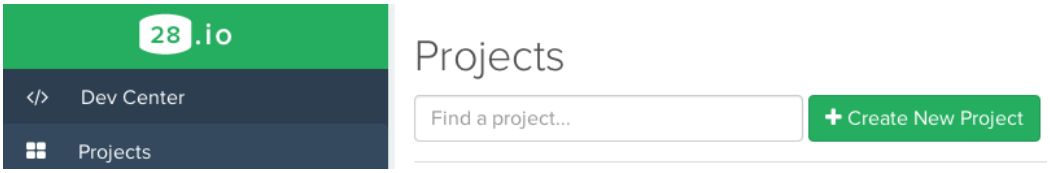
5. Login using your chosen credentials.



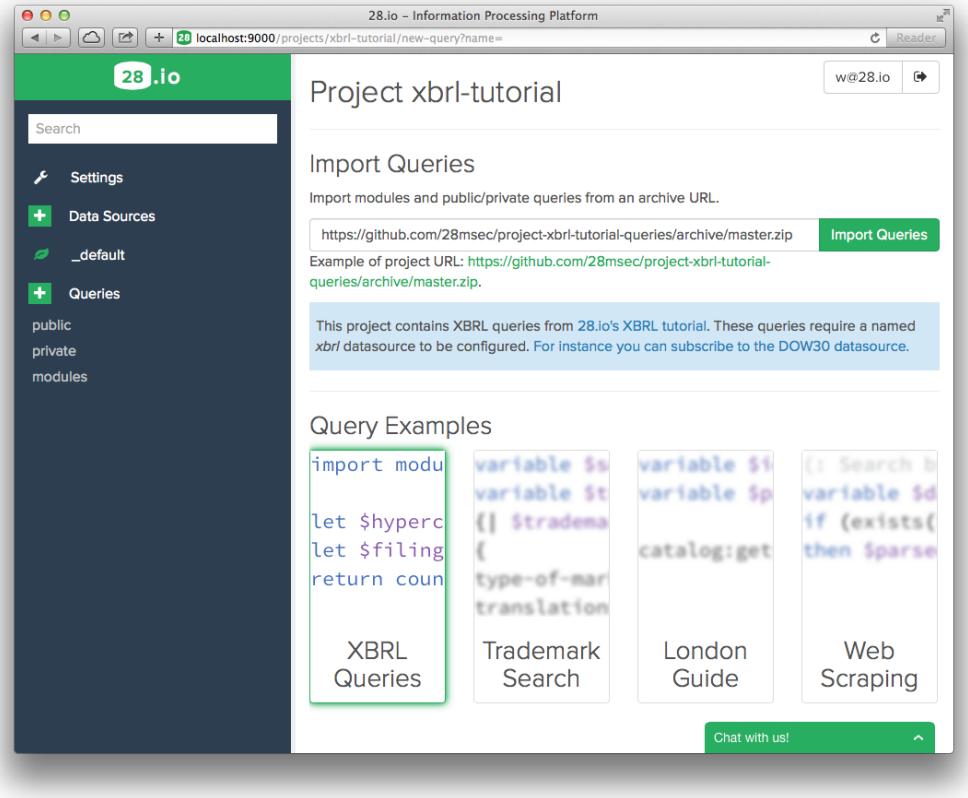
6. Click on `Data Sources` to view the list of data sources to which you can subscribe.



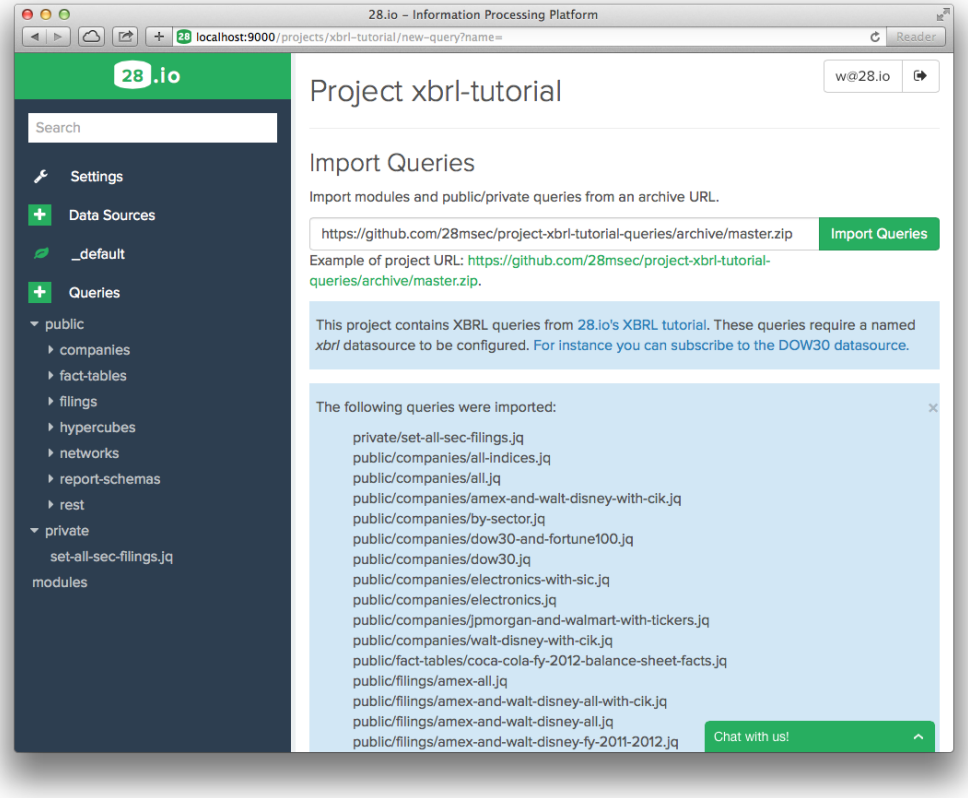
7. Confirm your billing information (DOW30 is for free). Then go to the projects page.



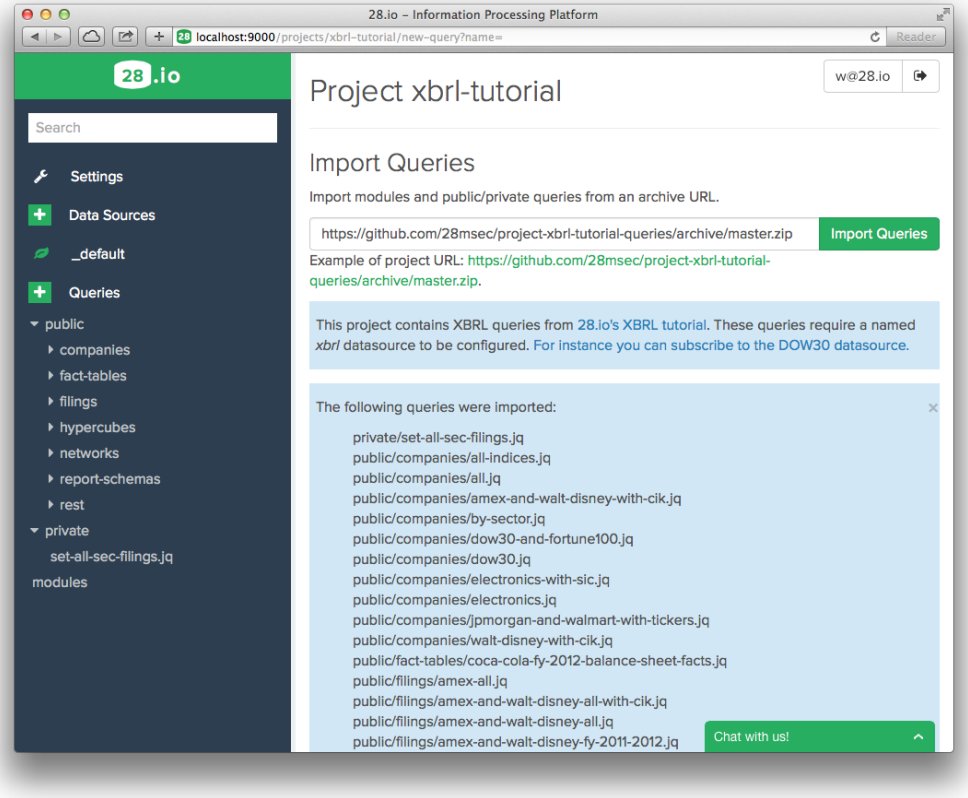
8. Create a new project by picking a name.



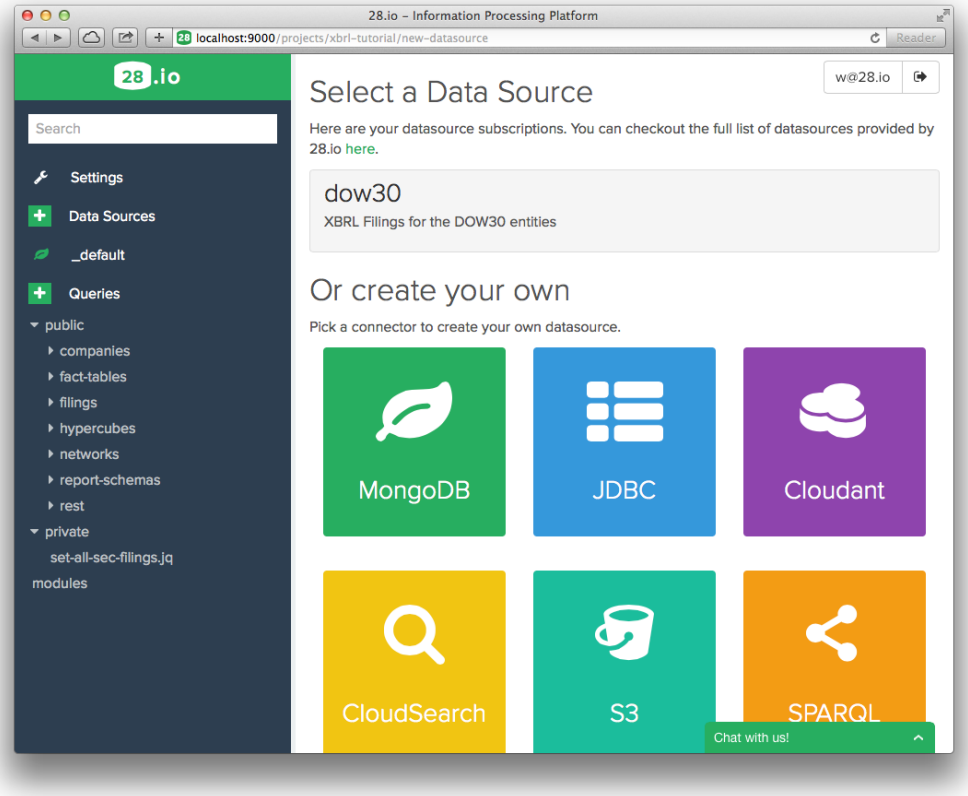
9. Pick a query name, ending with .jq (for JSONiq).



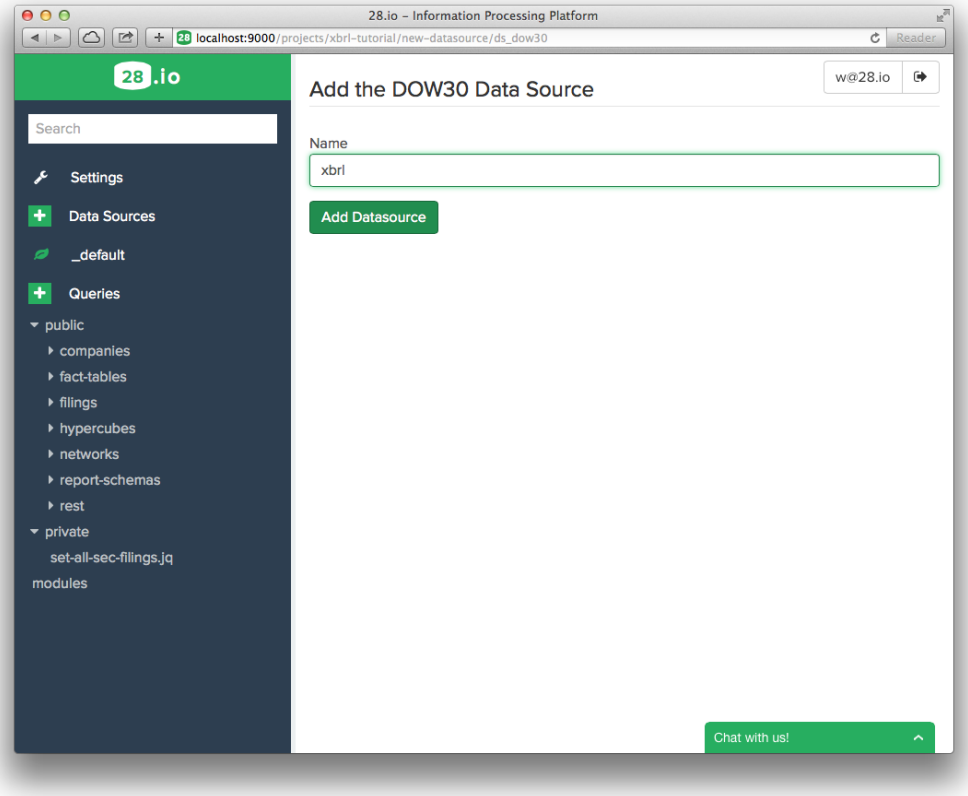
10.Enter the URL of a github repository from which to import queries. The URL of this tutorial's repository can be entered by simply clicking the example link. Then click "Import Queries."



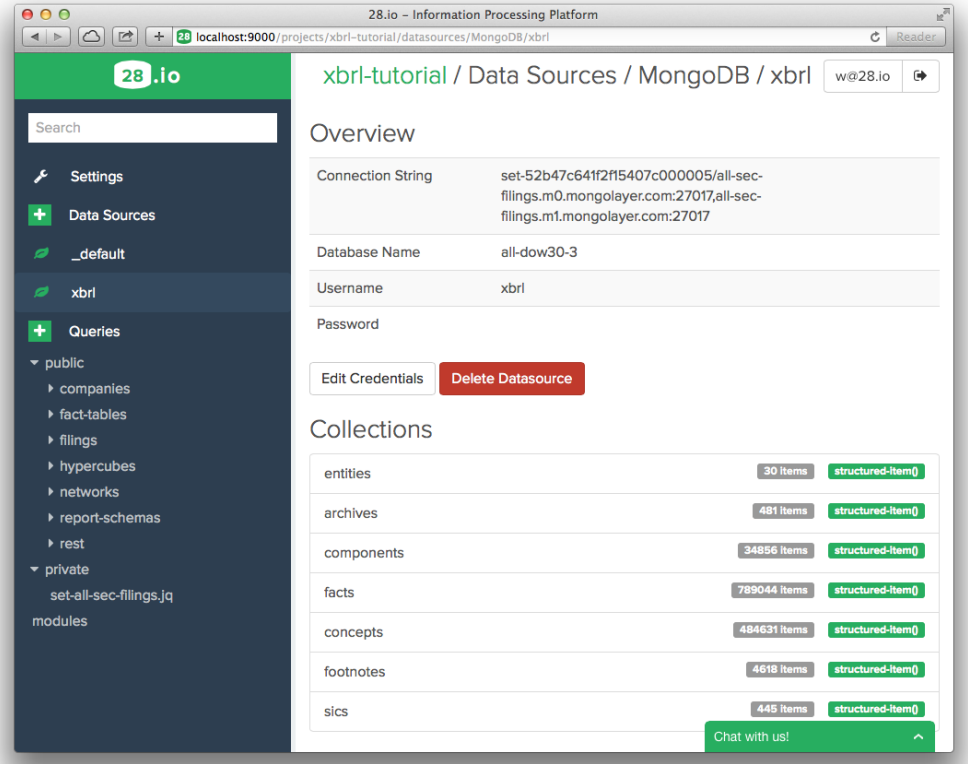
11.Click on "+" next to Data Sources to add a data source to this project.



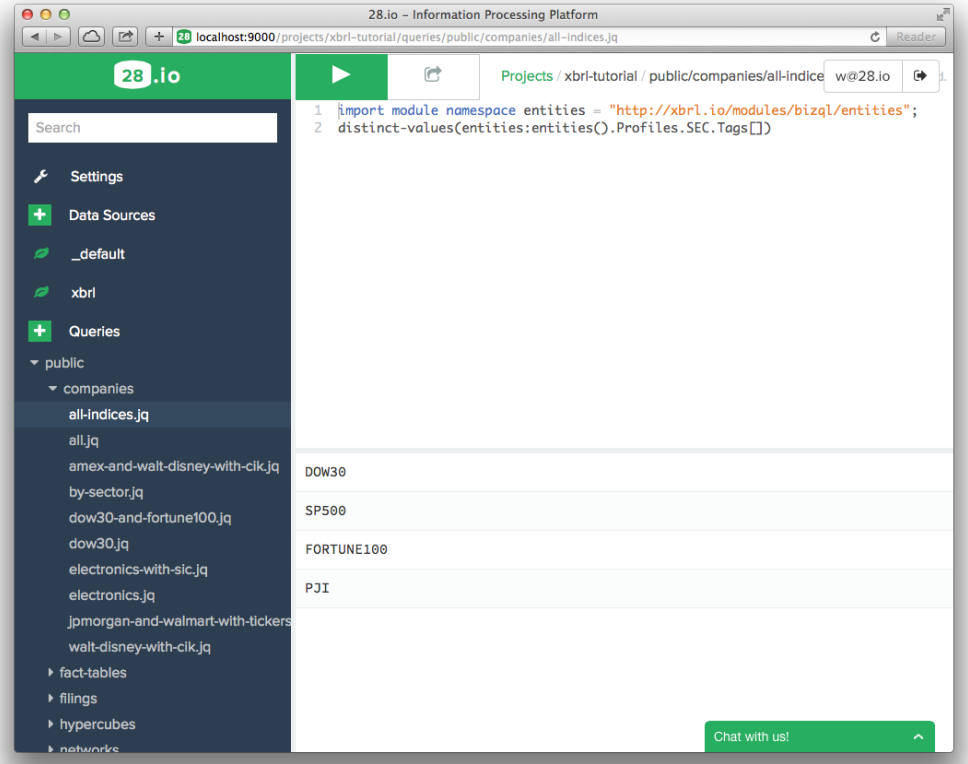
12.Click on DOW30 (This is the data source that you have subscribed to earlier).



13.Type "xbrl" in the text box (this is very important, as the XBRL connector specifically connects to the data source identified by this name in your project) and click "Add Data Source".



14. Select a query. The code appears on the right. Click on the white arrow on a green background to execute it. The results appear below the source code.



15.Type your query, and click on the white triangle on a green background to execute it.

The queries in this tutorial

All queries shown in this tutorial are available as an xbrl.io project on `https://github.com/28msec/project-xbrl-tutorial-queries`. If you make changes to the project and would like to save them in your own github repository, or if you would like to import another repository, you can use the 28 command line tool (available at `https://github.com/28msec/28`).

Organization of the tutorial

XBRL is about submitting *facts*. Facts are reported by companies, which are introduced in Chapter 2, *Companies*.

In real life though, facts are not reported one by one, but are submitted to an authority like the SEC in **archives**. A bit like you don't buy pop corn one chunk at a time, but in a pop corn package.

A filing is made of facts, but also contains metainformation on these facts. This metainformation is called taxonomy in the XBRL world, while the factual part is called an XBRL instance. It contains, from an abstract perspective (we don't go into the low-level details of networks, etc):

- Concept declarations: this is a description of the terminology used by the facts.
- User-friendly labels for the concepts, which can be used for fancy front-ends and editors, as well as human-readable documentation on the concepts.
- Hypercube declarations, and their dimensions (dimensions also have fancy labels, etc).

- Structures, like hierarchies of concepts that can be used on a fancy front-end as well to nicely display the facts.
- Formulas, that can be used to either validate the reported facts (example: the assets match the equities and liabilities) or to compute new, non reported facts.

Filings are explained in Chapter 3, *Filings*.

Typically, a filing is reported by a single company, called the reporting entity, and all facts are attached to it.

A filing is typically subdivided in networks. A network is just a subset of an archive that "makes sense", like a balanced sheet, or metadata on the report, or an income statement. Networks are explained in Chapter 4, *Networks*.

Just like elephants live in herds, whey they are wild, facts usually live in tables. For each network, a fact table can be obtained as shown in Chapter 5, *Fact tables*.

A hypercube provides a very convenient way to query for facts: give me a hypercube, and I can give you the facts that belong to it.

The hypercube can be directly taken from an archive (like a balanced sheet statement table). But it can also be generated ad hoc, which covers even more ground: ad hoc hypercubes allow the organization of facts across multiple entities, and across multiple archives, that is, across multiple reporting periods. This is shown in Chapter 6, *Hypercube Querying*.

Ad hoc hypercubes are typically integrated in report schemas. A report schema can provide, in addition to a hypercube, concept maps and business rules. We will come back on this later.

Chapter 2. Companies

Companies submit filings to the SEC via the Edgar system.

In the XBRL world, companies correspond to reporting **entities.**

The XBRL connector provides two modules for working with companies. One of them is generic, the other one offers functionality that is specific to the SEC. Modules are identified with URIs (they look very much like Web site addresses), in this case:

```
http://xbrl.io/modules/bizql/entities
http://xbrl.io/modules/bizql/profiles/sec/companies
```

Looking Up Companies

Let us begin with a very simple query that just lists the SEC companies. It is generic, so it uses the entities module:

Example 2.1. All companies

```
import module namespace entities =
    "http://xbrl.io/modules/bizql/entities";

entities:entities()
```

This query is just a function call. A function is identified with its prefix (here `entities:`) and a name (here `entities`). This function takes no parameters.

In the XBRL connector, a company is represented by a JSON object, like so:

Example 2.2. A company in JSON format

```
{
  "_id" : "http://www.sec.gov/CIK 0000320193",
  "Profiles" : {
    "SEC" : {
      "Name" : "SEC",
      "CompanyName" : "APPLE INC",
      "CompanyType" : "Corporation",
      "CIK" : "0000320193",
      "SIC" : "3571",
      "SICDescription" : "ELECTRONIC COMPUTERS",
      "SICGroup" : 3,
      "Taxonomy" : "ci",
      "Sector" : "Industrial/Commercial Machinery/Computer Equip
ment",
      "Tickers" : [ "aapl" ],
      "Tags" : [ "SP500", "FORTUNE100" ],
      "IsTrust" : false
    }
  }
}
```

In the case of companies, most of the information is SEC-specific, and is embedded in a Profiles.SEC subobject. You will see in other parts of the tutorial, that the Profiles.SEC subobject is not restricted to companies, but also applies to all other objects.

Companies can be retrieved by tags (for example, indices) using the `companies-for-tags` function, in the companies module, like so:

Example 2.3. All Dow 30 companies

```
import module namespace companies =
    "http://xbrl.io/modules/bizql/profiles/sec/companies";

companies:companies-for-tags("DOW30")
```

Almost all functions in the XBRL connector accept sequences as parameters. For example, you can get DOW30 and FORTUNE100 companies in a single call. Mind the double parenthesis: you are passing one single sequence as a parameter, not two strings as two parameters.

Example 2.4. All Dow 30 and Fortune 100 companies (companies in both indices will be returned twice)

```
import module namespace companies =
    "http://xbrl.io/modules/bizql/profiles/sec/companies";

companies:companies-for-tags( ("DOW30", "FORTUNE100") )
```

Companies are identified with CIKs. A CIK can be used to retrieve a company

Example 2.5. Walt Disney by CIK

```
import module namespace companies =
    "http://xbrl.io/modules/bizql/profiles/sec/companies";

companies:companies(1001039)
```

Here too, you can pass sequences.

Example 2.6. American Express and Walt Disney by CIK

```
import module namespace companies =
    "http://xbrl.io/modules/bizql/profiles/sec/companies";

companies:companies((4962, 1001039))
```

Companies by SIC (electronic and other electrical equipment, semiconductors and related devices).

Example 2.7. American Express and Walt Disney by CIK

```
import module namespace companies =
    "http://xbrl.io/modules/bizql/profiles/sec/companies";

companies:companies-for-SIC(("3600", "3674"))
```

Example 2.8. Companies by sector (electronic)

```
import module namespace companies =
    "http://xbrl.io/modules/bizql/profiles/sec/companies";

companies:companies-for-sector(
   "Electronic/Other Electrical Equipment/Components"
)
```

Example 2.9. JPMorgan and Walmart by ticker

```
import module namespace companies =
    "http://xbrl.io/modules/bizql/profiles/sec/companies";

companies:companies-for-tickers(("jpm", "wmt"))
```

Using FLWOR expressions, you can build many interesting queries just with companies. For example, this groups company names by sector, using the Sector and CompanyName fields in the SEC Profile.

Example 2.10. All company names grouped by sector

```
import module namespace entities =
    "http://xbrl.io/modules/bizql/entities";

for $e in entities:entities()
let $s := $e.Profiles.SEC.Sector
group by $s
return {
  sector : $s,
  entities : [ $e.Profiles.SEC.CompanyName ]
}
```

Querying Companies

In the former section, we showed how to query indices. But which indices are available? Even if there were no documentation, the answer lies at your fingertips.

One of the provided pieces of information for a company is the Tags field, which we use to stamp companies with indices. Using navigation syntax (the dot navigates deep into an object, double square brackets with an integer into an array, empty square brackets extract the entire array) as well as the distinct-values builtin function, it is very easy to discover, in real time, all indices currently used over the entire database:

Example 2.11. Discovering all indices in half a second's time

```
import module namespace entities =
    "http://xbrl.io/modules/bizql/entities";

distinct-values(entities:entities().Profiles.SEC.Tags[])
```

Having done so, you will discover that the following indices are available.

Table 2.1. Available indices

Index	Tag
S&P 500	SP500
Dow 30	DOW30
Fortune 100	FORTUNE100
Providence Journal's Impact 50	PJI

The SECXBRL.info REST API for companies

For people without developer background, we provide a REST API that allows you to look up companies and, say, import them into an Excel spreadsheet. The API is documented here [http://www.secxbrl.info/api]

Retrieve a company

You can retrieve:

- All companies with `http://secxbrl.xbrl.io/api/entities.jq`
- One or several companies by CIK with the `cik` parameter, like so:

  `http://secxbrl.xbrl.io/api/entities.jq?cik=1412090&cik=4962`

- Companies by (one or several) tag with the `tag` parameter, like so:

  `http://secxbrl.xbrl.io/api/entities.jq?tag=DOW30`

- One or several companies by ticker with the `ticker` parameter, like so:

  `http://secxbrl.xbrl.io/api/entities.jq?ticker=AAPL&ticker=GOOG`

Select a format

You can choose the format in which you would like to retrieve company information. By default, JSON will be output.

- In JSON with the `json` parameter, like so:

  `http://secxbrl.xbrl.io/api/entities.jq?tag=DOW30&format=json`

- In XML with the `xml` parameter, like so:

  `http://secxbrl.xbrl.io/api/entities.jq?tag=DOW30&format=xml`

- In CSV (comma-separated values, which you can open in good old Excel) with the `csv` parameter, like so:

  `http://secxbrl.xbrl.io/api/entities.jq?tag=DOW30&format=csv`

Chapter 3. Filings

In the former chapter, we explained how to deal with companies. We now go into the details of the filings that these companies submitted to the SEC.

Physically, companies submit XBRL instances, possibly together with an extension taxonomy.

The XBRL connector abstracts from this by grouping the instance and the DTS (us-gaap taxonomy + extension) into what is called an archive.

The XBRL connector provides two modules for working with filings. One of them is generic, the other one offers functionality that is specific to the SEC. In this case:

```
http://xbrl.io/modules/bizql/archives
http://xbrl.io/modules/bizql/profiles/sec/filings
```

Looking Up Filings

Let us begin with a very simple query that just lists the SEC filings. It is generic, so it uses the archives module:

Example 3.1. All filings

```
import module namespace archives =
    "http://xbrl.io/modules/bizql/archives";
archives:archives()
```

In the XBRL connector, a filing is represented by a JSON object, like so (in this case, Coca Cola's report for Q1 2013):

Example 3.2. A filing in JSON format

```
{
  "_id" : "0000021344-13-000017",
  "Entity" : "http://www.sec.gov/CIK 0000021344",
  "InstanceURL" : "http://www.sec.gov/Archives/edgar/data/21344/
000002134413000017/ko-20130329.xml",
  "Profiles" : {
    "SEC" : {
      "Name" : "SEC",
      "FormType" : "10-Q",
      "FilingDate" : "04/25/2013",
      "FileNumber" : "001-02217",
      "AcceptanceDatetime" : "20130425115725",
      "Period" : "20130329",
      "AssistantDirector" : "9",
      "SECFilingPage" : "http://www.sec.gov/Archives/edgar/data/
21344/000002134413000017/0000021344-13-000017-index.htm",
      "DocumentPeriodEndDate" : "2013-03-29",
      "Fiscal" : {
        "DocumentFiscalPeriodFocus" : "Q1",
        "DocumentFiscalYearFocus" : 2013
      },
      "Generator" : "WebFilings"
    }
  },
  "Namespaces" : {
```

```
    "link" : "http://www.xbrl.org/2003/linkbase",
    "ko" : "http://www.thecoca-colacompany.com/20130329",
    "xbrli" : "http://www.xbrl.org/2003/instance",
    "xs" : "http://www.w3.org/2001/XMLSchema",
    "us-gaap" : "http://fasb.org/us-gaap/2012-01-31",
    "nonnum" : "http://www.xbrl.org/dtr/type/non-numeric",
    "xbrldt" : "http://xbrl.org/2005/xbrldt",
    "num" : "http://www.xbrl.org/dtr/type/numeric",
    "dei" : "http://xbrl.sec.gov/dei/2012-01-31",
    "iso4217" : "http://www.xbrl.org/2003/iso4217"
  },
  "Statistics" : {
    "NumHypercubes" : 25,
    "NumNetworks" : 86,
    "NumDistinctExplicitDimensions" : 21,
    "NumDistinctDomains" : 22,
    "NumDistinctMembers" : 93,
    "NumDistinctConcretePrimaryItemsInHypercubes" : 244,
    "NumDistinctAbstractPrimaryItemsInHypercubes" : 58,
    "NumDistinctConcretePrimaryItemsNotInHypercubes" : 60,
    "NumDistinctAbstractPrimaryItemsNotInHypercubes" : 160,
    "Profiles" : {
      "SEC" : {
        "Name" : "SEC",
        "NumExtensionConcepts" : 73,
        "NumExtensionAbstracts" : 80,
        "NumDistinctReportElementNamesEndingWithTable" : 17,
        "NumDistinctReportElementNamesEndingWithAxis" : 21,
        "NumDistinctReportElementNamesEndingWithDomain" : 22,
        "NumDistinctReportElementNamesEndingWithMember" : 71,
        "NumDistinctReportElementNamesEndingWithLineItems" : 19,
        "NumDistinctReportElementNamesEndingWithAbstract" : 53,
        "NumDistinctReportElementNamesEndingWithAnythingElse" :
319,
        "NumExtensionFacts" : 217
      }
    },
    "NumFacts" : 990,
    "NumFootnotes" : 1,
    "NumComponents" : 53
  }
}
```

Some of the information in here is not specific to the SEC: the reporting entity, the URL to the physical XBRL instance, the namespaces to which prefixes (used in facts) correspond, and some statistics (number of facts, of hypercubes, etc).

Just like companies, filing objects also have SEC-specific information that is embedded in a Profiles.SEC subobject. It contains, for example, the SEC form type, the filing date, the reported fiscal period, etc. There are also SEC-specific statistics.

You can ask the filings submitted by a given company with the `filings:filings-for-companies` function:

Example 3.3. American Express's filings

```
import module namespace companies =
    "http://xbrl.io/modules/bizql/profiles/sec/companies";
import module namespace filings =
```

```
    "http://xbrl.io/modules/bizql/profiles/sec/filings";

let $amex := companies:companies(4962)
return filings:filings-for-companies($amex)
```

As always, any sequence is accepted as input:

Example 3.4. All filings by American Express and Walt Disney

```
import module namespace companies =
    "http://xbrl.io/modules/bizql/profiles/sec/companies";
import module namespace filings =
    "http://xbrl.io/modules/bizql/profiles/sec/filings";

let $amex := companies:companies(4962)
let $disney := companies:companies(1001039)
return filings:filings-for-companies( ($amex, $disney) )
```

For convenience, you can always pass a CIK instead of the company object, which greatly simplifies the query. This applies to all functions taking a sequence of companies as a parameter.

Example 3.5. Filings by Amex and Disney in a single call

```
import module namespace filings =
    "http://xbrl.io/modules/bizql/profiles/sec/filings";

filings:filings-for-companies( (4962, 1001039) )
```

If you are looking for a specific fiscal period or year, the fiscal module can help.

Example 3.6. Filings by Amex and Disney, FY 2011 and FY 2012

```
import module namespace fiscal =
    "http://xbrl.io/modules/bizql/profiles/sec/fiscal/core";

fiscal:filings-for-entities-and-fiscal-periods-and-years(
    (4962, 1001039),
    "FY",
    (2011, 2012)
)
```

Diving into a filing

Once you have one or more filing objects, you can query them. There are two main ways to do it:

- Using JSONiq navigation (with dots and square brackets).

  Using functions.

For example, if you would like to use the statistics to count the facts in FY 2011 and 2012 filings by Apple and Google:

```
import module namespace fiscal =
    "http://xbrl.io/modules/bizql/profiles/sec/fiscal/core";

let $filings :=
  fiscal:filings-for-entities-and-fiscal-periods-and-years(
      (4962, 1001039),
```

```
      "FY",
      (2011, 2012) )
return sum($filings.Statistics.NumFacts)
```

Some of the fields are available, for convenience, with functions (in this case, the archive ID as well as the fiscal period and year):

Example 3.7. Discover Amex's filings by fiscal year and period

```
import module namespace archives =
    "http://xbrl.io/modules/bizql/archives";

import module namespace companies =
    "http://xbrl.io/modules/bizql/profiles/sec/companies";
import module namespace filings =
    "http://xbrl.io/modules/bizql/profiles/sec/filings";

import module namespace fiscal =
    "http://xbrl.io/modules/bizql/profiles/sec/fiscal/core";

let $amex := companies:companies(4962)
for $filing in filings:filings-for-companies($amex)
return {
  AID: archives:aid($filing),
  Period: fiscal:fiscal-period($filing),
  Year: fiscal:fiscal-year($filing)
}
```

The SECXBRL.info REST API for filings

We also provide a REST API that allows you to look up filings and, say, import them into an Excel spreadsheet. The API is documented here [http://www.secxbrl.info/api]

Retrieve a filing

You can retrieve a filing given the CIKs of (one or several) companies with the `cik` parameter like so:

```
http://secxbrl.xbrl.io/api/filings.jq?cik=320193&cik=1288776
```

If you do not know the CIK of the company you are looking for, you can also use the `tag` or `ticker` parameter like in the entities API. Or you can use the entities REST API, explained in the former chapter.

You can retrieve specify a fiscal period or year with the `fiscalPeriod` and `fiscalYear` parameters like so:

```
http://secxbrl.xbrl.io/api/filings.jq?
cik=320193&fiscalYear=2012&fiscalPeriod=Q1&fiscalPeriod=Q2
```

You can use fiscalYear=LATEST to retrieve the latest year.

Select a format

You can also choose the format in which you would like to retrieve filing information, like in the entities API.

For example, for Excel:

```
http://secxbrl.xbrl.io/api/filings.jq?
cik=320193&cik=1288776&format=csv
```

Chapter 4. Networks

A filing contains a lot of information. It makes sense to split this information in smaller components. For example a quarterly report has a balanced sheet, an income statement, some generic information about the filing, etc. In the SEC world, these are called *networks*.

Physically, companies submit XBRL instances, possibly together with an extension taxonomy.

The XBRL connector provides two modules for working with SEC networks. One of them is generic, the other one offers functionality that is specific to the SEC. In this case:

```
http://xbrl.io/modules/bizql/components
http://xbrl.io/modules/bizql/profiles/sec/networks
```

However, the networks module is the most useful for working with SEC data, because of all the specificities of Edgar filings.

Looking Up Networks

Let us begin with a very simple query that just lists the SEC networks contained in a filing, say, Coca Cola's Q1 for 2012.

Example 4.1. The various components in Coca Cola's Q1 2012 filing.

```
import module namespace fiscal =
    "http://xbrl.io/modules/bizql/profiles/sec/fiscal/core";
import module namespace sec-networks =
    "http://xbrl.io/modules/bizql/profiles/sec/networks";

let $filing :=
    fiscal:filings-for-entities-and-fiscal-periods-and-years(
      21344,
      "Q1",
      2012
    )
return sec-networks:networks-for-filings($filing)
```

In the XBRL connector, a network is represented by a JSON object. This object contains complex XBRL machinery to deal with factual structures (XBRL hypercubes, presentation networks, etc). Normally, you do not need to look into it, apart maybe from the statistics section that is similar to those of filings.

A much more useful, SEC-specific representation of the network is called the model structure and we explain later how to access it.

A certain number of networks are common to all filings, and stamped with a disclosure. Here is, for example, how to retrieve the balance sheet in the above filing.

```
import module namespace fiscal =
    "http://xbrl.io/modules/bizql/profiles/sec/fiscal/core";
import module namespace sec-networks =
    "http://xbrl.io/modules/bizql/profiles/sec/networks";

let $filing :=
    fiscal:filings-for-entities-and-fiscal-periods-and-years(
      21344,
```

```
      "FY",
      2012
    )
return sec-networks:networks-for-filings-and-disclosures(
    $filing,
    $sec-networks:BALANCE_SHEET
)
```

Other examples of available disclosures include the balance sheet, the income statement, the cash flow statement, document and entity information.

The Model Structure

```
import module namespace fiscal =
    "http://xbrl.io/modules/bizql/profiles/sec/fiscal/core";
import module namespace sec-networks =
    "http://xbrl.io/modules/bizql/profiles/sec/networks";

let $filing :=
    fiscal:filings-for-entities-and-fiscal-periods-and-years(
      21344,
      "FY",
      2012
    )
let $balance-sheet := sec-networks:networks-for-filings-and-disclosures(
    $filing,
    $sec-networks:BALANCE_SHEET
)
return sec-networks:model-structures($balance-sheet)
```

A model structure is just a hierarchy of so-called *SEC report elements* of different kinds. In that case:

```
{
  "Kind" : "Abstract",
  "Name" : "us-gaap:StatementOfFinancialPositionAbstract",
  "Label" : "Statement of Financial Position [Abstract]",
  "Children" : [ {
    "Kind" : "Table",
    "Name" : "us-gaap:StatementTable",
    "Label" : "Statement [Table]",
    "Children" : [ {
      "Kind" : "LineItems",
      "Name" : "us-gaap:StatementLineItems",
      "Label" : "Statement [Line Items]",
      "Children" : [ {
        "Kind" : "Abstract",
        "Name" : "us-gaap:AssetsAbstract",
        "Label" : "Assets [Abstract]",
        "Children" : [ {
          "Kind" : "Concept",
          "Name" : "us-gaap:Assets",
          "Label" : "Assets",
          "IsNillable" : true,
          "IsAbstract" : false,
          "PeriodType" : "instant",
          "Balance" : "debit",
          "SubstitutionGroup" : "xbrli:item",
          "DataType" : "xbrli:monetaryItemType",
```

```
          "BaseType" : "xbrli:monetary",
          "ClosestSchemaBuiltinType" : "xs:decimal",
          "IsTextBlock" : false
        }, ... ]
      }, ... ]
    }, {
      "Kind" : "Axis",
      "Name" : "us-gaap:StatementScenarioAxis",
      "Label" : "Scenario [Axis]",
      "Children" : [ {
        "Kind" : "Member",
        "Name" : "us-gaap:ScenarioUnspecifiedDomain",
        "Label" : "Scenario, Unspecified [Domain]"
      } ]
    } ]
  } ]
}
```

Here is a list of all reports elements (which form the model structure) used by SEC:

Table 4.1. Report elements

Report Element	What it is
Table	This is a "container" for all facts in this component. While it is called table, *hypercube* would be a more precise terminology because it can have much more than two dimensions.
LineItems	This is the top-level element for the "rows" of the table.
Abstract	These are virtual rows, i.e., no facts are reported against them. They help organize concepts.
Concept	These are rows against which facts get reported.
Axis	Several facts can be reported for the same concept, but in different contexts. Axes help organize this. For example, an axis can be used to delimit a region of the world to which the fact applies.
Member	A member is an axis value. Members can be organized in hierarchies as well (which is easy to understand if you think of regions of the world for example).

The SECXBRL.info REST API for networks

We also provide a REST API that allows you to look up networks and, say, import them into an Excel spreadsheet. The API is documented here [http://www.secxbrl.info/api]

Retrieve a network

You can retrieve the networks (components) in a filing using the `aid` parameter, like so:

```
http://secxbrl.xbrl.io/api/components.jq?aid=0000021344-13-000017
```

If you do not know the AIDs of the filings, you can use the same parameters as in the filings API (`cik`, `tag`, `ticker`, `fiscalYear`, `fiscalPeriod`), or use the filings REST API to retrieve it.

Select a format

You can also choose the format in which you would like to retrieve network information, like in the entities and filings APIs.

For example, for Excel:

```
http://secxbrl.xbrl.io/api/components.jq?
aid=0000021344-13-000017&format=csv
```

Chapter 5. Fact tables

When you have a network, you can ask for all facts that it contains. This is called a fact table.

Once you have your hands on a network, it's very easy to ask for the fact table, with the sec-networks:facts() function.

Facts

XBRL is about reporting facts. A fact is a reporting atom, meaning that it's the smallest chunk that can be reported. For example, a fact can be that Apricot Inc. owns 30 billion dollars in assets on december 31st, 2012, that a HAL stock was worth 31.41$ on this day, or that Mr. Sugarmountain lived in Palo Alto between 2004 and 2012.

A fact has a certain number of properties:

- A value: the meat of the fact. It can a string, a number, etc.
- A concept: it's **what** this fact is about. For example: assets, or revenues, etc.
- A unit: it's **of what** if it is a number (example: USD/share for a dividend, etc)
- An entity: it's **who** reported this fact. For example, Apricot Inc.
- A period: it's **when** this fact applies. A period can be an instance (like May 4th, 2013) or a duration (January 1st thru December 31st, 2013).
- A number of further characterizing dimensions, such as a region of the world, a company department, etc.

Looking up a fact table for an SEC network

As we said earlier, facts are not alone: in the wild, they live in tables. Let us get back to the balance sheet of Coca Cola for FY 2012, and ask for the fact table.

Example 5.1. The fact table for a balance sheet by Coca Cola

```
import module namespace fiscal =
    "http://xbrl.io/modules/bizql/profiles/sec/fiscal/core";
import module namespace sec-networks =
    "http://xbrl.io/modules/bizql/profiles/sec/networks";

let $filing :=
    fiscal:filings-for-entities-and-fiscal-periods-and-years(
      21344,
      "FY",
      2012
    )
let $network :=
    sec-networks:networks-for-filings-and-disclosures(
      $filing,
      $sec-networks:BALANCE_SHEET
    )
return sec-networks:facts($network)
```

Below is an example of fact.

Example 5.2. The fact object format

```
{
  "Archive" : "0000021344-13-000007",
  "_id" : "68a552aa-ff68-4fb0-be34-c4507b1fbc00",
  "IsInDefaultHypercube" : true,
  "Aspects" : {
    "xbrl:Concept" : "ko:AvailableForSaleSecuritiesAndCostMethod
Investments",
    "xbrl:Entity" : "http://www.sec.gov/CIK 0000021344",
    "xbrl:Period" : "2012-12-31",
    "xbrl:Unit" : "iso4217:USD",
    "us-gaap:StatementScenarioAxis" : "us-gaap:ScenarioUnspecifi
edDomain"
  },
  "Profiles" : {
    "SEC" : {
      "Name" : "SEC",
      "Fiscal" : {
        "Period" : "FY",
        "Year" : 2012
      },
      "DocEndDate" : "2012-12-31",
      "Accepted" : "20130227122203",
      "IsExtension" : true
    }
  },
  "Type" : "NumericValue",
  "Value" : 1232000000,
  "Decimals" : -6,
  "AuditTrails" : [ {
    "Data" : {
      "Dimension" : "us-gaap:StatementScenarioAxis",
      "Member" : "us-gaap:ScenarioUnspecifiedDomain"
    },
    "Type" : "hypercubes:dimension-default",
    "Label" : "Default dimension value"
    }
  ]
}
```

Note the Profiles.SEC section which also exists for facts, like companies, filings and networks. This section mostly contains the fiscal period and year against which the yearly or quaterly report was reported, as well as the end date of the report, the acceptance date by SEC, and a boolean that indicates whether the concept exists for all companies (often prefixed with us-gaap:), or was made up by the company reporting this fact.

The fact also contains an audit trail, that we will detail more in the hypercube section.

The SECXBRL.info REST API for fact tables

We also provide a REST API that allows you to look up fact tables and, say, import them into an Excel spreadsheet. The API is documented here [http://www.secxbrl.info/api]

Retrieve the fact table associated with a network

You can retrieve the fact table of a components using the `cid` parameter.

```
http://secxbrl.xbrl.io/api/facttable-for-component.jq?cid=66887390-
ee56-44a7-a897-62eefe944476
```

You can also use the parameters from the components API: For example, the fact table for Coca Cola's balance sheet for FY 2012 can be retrieved with

```
http://secxbrl.xbrl.io/api/facttable-for-component.jq?
_method=POST&format=xml&cik=21344&fiscalYear=2012
&fiscalPeriod=FY&disclosure=BalanceSheet
```

The format parameter is also available as usual.

Chapter 6. Hypercube Querying

In the former section, we showed how to get the fact table associated with an SEC network. You may have seen at several place occurrences of the word "Hypercube". The facts in an SEC network are actually organized in a hypercubic structure. We now go more into detais on this, and eventually show how you can build your own hypercubes and query for fact tables, outside of any SEC network.

Hypercubes

Concepts, entities, periods, units, as well as further dimensions are called **aspects**. It is possible to organize facts along **hypercubes**, the dimensions of which are these aspects. Usually, there will be either zero or one fact for each possible tuple of aspects. If there is more, it often indicates an inconsistency in the submission.

For example, if we stick to the USD unit, you can imagine a cube whose width represents concepts (Assets, Revenues), the height of which represents entities (Amex, Disney) and the depth of which represents periods (year 2012, year 2013, year 2014). This hypercube potentially contains 2 x 2 x 3 = 12 facts.

Note: From a practical perspective, it is disputed whether units are considered aspects, or whether they "stick" to the value.

Building your own hypercube

The simplest hypercube you can imagine is made of four "standard" aspects: concept, entity, period and unit. It does not restrict the value of any of these aspects. It is called dimensionless, because in the XBRL universe, the four basic aspects are not considered XBRL dimensions.

Example 6.1. The dimensionless hypercube

```
import module namespace hypercubes =
    "http://xbrl.io/modules/bizql/hypercubes";

hypercubes:dimensionless-hypercube()
```

The result of this query shows you the object representation of a hypercube.

Example 6.2. The object format of hypercubes

```
{
  "Name" : "xbrl:DimensionlessHypercube",
  "Label" : "Dimensionless Hypercube",
  "Aspects" : {
    "xbrl:Concept" : {
      "Name" : "xbrl:Concept",
      "Label" : "Implicit XBRL Concept Dimension"
    },
    "xbrl:Period" : {
      "Name" : "xbrl:Period",
      "Label" : "Implicit XBRL Period Dimension"
    },
    "xbrl:Entity" : {
      "Name" : "xbrl:Entity",
      "Label" : "Implicit XBRL Entity Dimension"
    },
    "xbrl:Unit" : {
      "Name" : "xbrl:Unit",
```

```
      "Label" : "Implicit XBRL Unit Dimension",
      "Default" : "xbrl:NonNumeric"
    }
  }
}
```

The facts in a hypercube can be queried with the function `sec:facts-for-hypercube`. There is a very high number of facts in this hypercube: all those without extra dimensions. Hundreds of thousands of them. Very often, when you query for facts against this dimensionless hypercube, you are asking for specifing archives. The function sec:facts-for-hypercube allows you to do so with its optional second parameter (the first being the hypercube).

Let's count all those facts that are in a given archive (here 1423).

Example 6.3. Retrieving the facts in the dimensionless hypercube for some filings

```
import module namespace hypercubes =
    "http://xbrl.io/modules/bizql/hypercubes";

import module namespace archives =
    "http://xbrl.io/modules/bizql/archives";

import module namespace sec =
    "http://xbrl.io/modules/bizql/profiles/sec/core";
import module namespace fiscal =
    "http://xbrl.io/modules/bizql/profiles/sec/fiscal/core";

let $hypercube := hypercubes:dimensionless-hypercube()
let $filing :=
  fiscal:filings-for-entities-and-fiscal-periods-and-years(
    (4962, 1001039),
    "FY",
    (2011, 2012)
  )
return count(sec:facts-for-hypercube(
  $hypercube,
  {
    Filter: {
      Archive: archives:aid($filing)
    }
  }
))
```

Instead of looking at a single archive, you can look across archives, for concepts like `us-gaap:Assets` and `us-gaap:Equity`. In order to do so, you need to modify the dimensionless hypercube add filter on these two concepts. There are over 70k such facts.

```
import module namespace hypercubes =
    "http://xbrl.io/modules/bizql/hypercubes";
import module namespace sec =
    "http://xbrl.io/modules/bizql/profiles/sec/core";

let $hypercube := hypercubes:dimensionless-hypercube({
  Concepts: [ "us-gaap:Assets", "us-gaap:Equity" ]
})
return count(sec:facts-for-hypercube($hypercube))
```

You can also build your own hypercube. To restrict a dimension, just add a field (One of `Concepts`, `Entities`, `Periods`, `Units`) with an array of values. Below we show you how to make a

restriction on DOW30 companies with just a small modification of the dimensionless hypercube function call.

```
import module namespace hypercubes =
    "http://xbrl.io/modules/bizql/hypercubes";

import module namespace sec =
    "http://xbrl.io/modules/bizql/profiles/sec/core";
import module namespace companies =
    "http://xbrl.io/modules/bizql/profiles/sec/companies";

let $hypercube := hypercubes:dimensionless-hypercube({
  Concepts: [ "us-gaap:Assets", "us-gaap:Equity" ],
  Entities: [ companies:companies-for-tags("DOW30")._id ]
})
return count(sec:facts-for-hypercube($hypercube))
```

If you begin to query across archives, and attempt to filter on periods, you will very soon notice that it is hard, because fiscal years differ from company to company. Technically, fiscal years and periods (FY, Q1, Q2, Q3) are not hypercube dimensions, but you can still filter for them using this Profiles.SEC.Fiscal part that annotates objects using the second parameter of `sec:facts-for-hypercube`. For example, here is how to get all assets and equities (no extra dimensions) for FY 2012 (6847 of them)

```
import module namespace hypercubes =
    "http://xbrl.io/modules/bizql/hypercubes";

import module namespace sec =
    "http://xbrl.io/modules/bizql/profiles/sec/core";

let $hypercube := hypercubes:dimensionless-hypercube({
  Concepts: [ "us-gaap:Assets", "us-gaap:Equity" ]
})
return count(sec:facts-for-hypercube(
    $hypercube,
    {
        Filter: {
            Profiles: {
                SEC: {
                    Fiscal: {
                        Period: "FY",
                        Year: 2012
                    }
                }
            }
        }
    }
))
```

What was done for the basic four dimensions (like xbrl:Equity) applies to extra dimensions as well. There is a more elaborate version of `hypercubes:dimensionless-hypercube` called `hypercubes:user-defined-hypercube` that allows you to add any number of dimensions, as well as restrict on them or add default values. Here you can ask for facts reported against the `us-gaap:DividendsCommonStock` concept, with a dimension `us-gaap:StatementEquityComponentsAxis` restricted on a value of `us-gaap:CommonStockMember`. There are 513 of them.

```
import module namespace hypercubes =
    "http://xbrl.io/modules/bizql/hypercubes";
```

```
import module namespace sec =
    "http://xbrl.io/modules/bizql/profiles/sec/core";

let $hypercube := hypercubes:user-defined-hypercube({
  "xbrl:Concept" : {
    Domain: [ "us-gaap:DividendsCommonStock" ]
  },
  "us-gaap:StatementEquityComponentsAxis" : {
    Domain: [ "us-gaap:CommonStockMember" ]
  }
})
return count(sec:facts-for-hypercube($hypercube))
```

Finally, you can combine extra dimensions, restricting several dimensions, filtering on fiscal years, etc. Let's ask for the companies that submitted, for FY 2011, a fact against the `us-gaap:DividendsCommonStock` concept, with a dimension `us-gaap:StatementEquityComponentsAxis` that has a value of `us-gaap:CommonStockMember`. There's only one and it's Walt Disney. And it takes less than one second to ask for this.

```
import module namespace facts =
    "http://xbrl.io/modules/bizql/facts";
import module namespace hypercubes =
    "http://xbrl.io/modules/bizql/hypercubes";

import module namespace sec =
    "http://xbrl.io/modules/bizql/profiles/sec/core";
import module namespace companies =
    "http://xbrl.io/modules/bizql/profiles/sec/companies";

let $hypercube := hypercubes:user-defined-hypercube({
  "xbrl:Entity" : {
    Domain: [  companies:companies-for-tags("DOW30")._id ]
  },
  "us-gaap:StatementEquityComponentsAxis" : {
    Domain: [ "us-gaap:CommonStockMember" ]
  }
})
let $fact := sec:facts-for-hypercube(
  $hypercube,
  {
    Filter: {
      Profiles: {
        SEC: {
          Fiscal: {
            Period: "FY",
            Year: 2012
          }
        }
      }
    }
  }
)
return companies:companies($fact ! facts:entity-for-fact($$))
    .Profiles.SEC.CompanyName
```

The SEC modules provide many functions that just wrap these queries in nicer code, and there are more to come. The documentation is there for you to find them.

The SECXBRL.info REST API for user-defined hypercubes

We also provide a REST API that allows you to build your hypercubes and ask for fact tables and, say, import them into an Excel spreadsheet. The API is documented here [http://www.secxbrl.info/api]

Build your hypercube with the REST API

You can build your own hypercube by using dimension names as fields in the query string, along with one or several values. For example, "xbrl:Concept=us-gaap:Goodwill" will filter for concepts named us-gaap:Goodwill.

For the entity dimension, for convenience, you can use the same parameters as in the companies API: cik, ticker, tag, etc -- instead of "xbrl:Entity".

For the concept dimension, use the more convenient "concept" parameter name. For example: `http://secxbrl.xbrl.io/api/facts.jq?concept=us-gaap:Goodwill&cik=4962`

For requiring a dimension, without restriction, use a value of "ALL". For example: `http://secxbrl.xbrl.io/api/facts.jq?concept=us-gaap:DividendsCommonStock&tag=DOW30&us-gaap:StatementEquityComponentsAxis=ALL`

You can specify a default dimension value using "dimensionname:default", like so: `http://secxbrl.xbrl.io/api/facts.jq?concept=us-gaap:DividendsCommonStock&tag=DOW30&us-gaap:StatementEquityComponentsAxis=ALL&us-gaap:StatementEquityComponentsAxis:default=sec:myDefault`

The last query in the former section corresponds to `http://secxbrl.xbrl.io/api/facts.jq?concept=us-gaap:DividendsCommonStock&tag=DOW30&us-gaap:StatementEquityComponentsAxis=us-gaap:CommonStockMember&fiscalYear=2012&fiscalPeriod=FY`

Chapter 7. Report schemas

For convenience, there are report schemas which provide convenient querying hypercubes, and more.

By default, your project contains no report schema. In this tutorial, we provide one that you can download under `http://facs.28.io/process-report-schema.jq`. JSONiq is powerful enough to do this for you with just one click (to 28.io, the Web is nothing but yet another data source...).

Example 7.1. Download the report schema with the Fundamental Accounting Concepts.

```
import module namespace http-client =
    "http://zorba.io/modules/http-client";

let $schema := parse-json(http-client:get-text(
    "http://facs.28.io/process-report-schema.jq"
).body.content)
return
if(is-available-collection("reportschemas"))
then {
    truncate("reportschemas");
    insert("reportschemas", $schema);
}
else
    create("reportschemas", $schema);

"Schema successfully added."
```

(Yes, JSONiq can do updates and side effects, but, ssssshhhh! don't tell anybody we showed you that).

Fundamental Accounting Concepts

Querying across multiple filings, even across companies, is not easy. This is because companies might not report their data in the same way. The Fundamental Accounting Concepts report schema provides an easy way to do so.

The report schema module is named `http://xbrl.io/modules/bizql/report-schemas`

Example 7.2. Asking for the FAC report schema

```
import module namespace report-schemas =
    "http://xbrl.io/modules/bizql/report-schemas";

report-schemas:report-schemas("FundamentalAccountingConcepts")
```

A report schema looks very much like a network, except that it is not bound to any filing. In particular, it also has a model structure.

Example 7.3. Asking for the FAC report schema

```
import module namespace report-schemas = "http://xbrl.io/modules/bizql/report-s
import module namespace sec-networks =
    "http://xbrl.io/modules/bizql/profiles/sec/networks";

let $schema := report-schemas:report-schemas("FundamentalAccountingConcepts")
return sec-networks:model-structures($schema)
```

But they have more than that. They may define their own concepts (beginning with the `fac:` prefix) and map them to real concepts with a so-called `concept map`.

Example 7.4. Asking for American Express's assets with the mapped fac:Assets concept

```
import module namespace hypercubes =
    "http://xbrl.io/modules/bizql/hypercubes";
import module namespace sec =
    "http://xbrl.io/modules/bizql/profiles/sec/core";
import module namespace fiscal =
    "http://xbrl.io/modules/bizql/profiles/sec/fiscal/core";

let $hypercube := hypercubes:dimensionless-hypercube()
let $filing := fiscal:filings-for-entities-and-fiscal-periods-and-years(
    4962,
    "FY",
    2012
)
return sec:facts-for-archives-and-concepts(
    $filing,
    "fac:Assets",
    {
        Hypercube: $hypercube,
        concept-maps: "FundamentalAccountingConcepts"
    }
)
```

For convenience, if you directly would like to retrieve all facts for a schema, you can use `sec:facts-for-schema` with the name of the desired schema, and the archive for which you would like the facts.

Example 7.5. Asking for Amex' FACs

```
import module namespace sec =
    "http://xbrl.io/modules/bizql/profiles/sec/core";
import module namespace fiscal =
    "http://xbrl.io/modules/bizql/profiles/sec/fiscal/core";

let $filing :=
    fiscal:filings-for-entities-and-fiscal-periods-and-years(
      4962,
      "FY",
      2012 )
return sec:facts-for-schema(
    "FundamentalAccountingConcepts",
    $filing
)
```

You can also ask for them across several archives.

Example 7.6. Asking for the FACs of Amex and Disney

```
import module namespace sec =
    "http://xbrl.io/modules/bizql/profiles/sec/core";
import module namespace fiscal =
    "http://xbrl.io/modules/bizql/profiles/sec/fiscal/core";
```

```
let $filing :=
    fiscal:filings-for-entities-and-fiscal-periods-and-years(
      (4962, 1001039),
      "FY",
      2012 )
return sec:facts-for-schema(
    "FundamentalAccountingConcepts",
    $filing
)
```

The SECXBRL.info REST API for concept maps

We also provide a REST API that allows you to look up mapped concepts and, say, import them into an Excel spreadsheet. The API is documented here [http://www.secxbrl.info/api]

Retrieve a fact with a concept map

You can use a concept map with the facts API using the `map` parameter, together with the name of the containing report schema, like so:

```
http://secxbrl.xbrl.io/api/facts.jq?concept=fac:Assets
&tag=DOW30&map=FundamentalAccountingConcepts
```

Chapter 8. Design Your Own REST API

If the above REST API does not cover your needs, you can design your own REST API.

Getting the parameters from the query string

We now explain how to do a simple API for just retrieve a company with its ticker.

The first thing you need to know is that, whenever you write a public query, its results becomes available through a REST call.

`http://project-name.xbrl.io/my-query.jq`

For example, you can call the query returning all entities with the following URI:

`http://project-name.xbrl.io/companies/all.jq`

For more elaborate queries though, the following endpoint is better suited:

`http://project-name.xbrl.io/v1/_queries/public/companies/all.jq`

However, it is more strict, meaning that it will complain if the query has side effects and if the method is GET -- you have to use POST for side-effecting queries.

Finally, this more elaborate endpoint is stricter about types, meaning, you need to use a function to declare the type of what the query returns.

For the sake of simplicity, we use the simpler endpoint in ths rest of this chapter.

Useful functions

You will need to read parameters from the query string. There is a function named parameter-values, which lives in the `http://www.28msec.com/modules/http-request` namespace. It takes a parameter name, and returns the sequence of all its values (it may have several if it appears several times in the request URL).

Example 8.1. Read a parameter's value(s) from the query string

```
import module namespace companies =
    "http://xbrl.io/modules/bizql/profiles/sec/companies";
import module namespace request =
    "http://www.28msec.com/modules/http/request";

let $tickers := request:parameter-values("t")
return companies:companies-for-tickers($tickers)
```

Here is a possible way to invoke this query via the REST API:

GET `http://project-name.xbrl.io/rest-api/query-string-parameter.jq?t=wmt&t=GOOG`

There are a few more useful functions. For example, `request:method-get()`, `method-post()`, etc, return a boolean and allow you to know which method the consumer used.

Until now, we used the request module. Likewise, there is a response module `http://www.28msec.com/modules/http-response`, with which you can tune the output (HTTP code, MIME type, etc).

Example 8.2. Read a parameter's value(s) from the query string

```
import module namespace companies =
    "http://xbrl.io/modules/bizql/profiles/sec/companies";
import module namespace request =
    "http://www.28msec.com/modules/http-request";
import module namespace response =
    "http://www.28msec.com/modules/http-response";

let $tickers := request:param-values("t")
return switch(true)
        case request:method-get() and exists($tickers)
        return {
          response:content-type("application/json");
          companies:companies-for-tickers($tickers)
        }
        case request:method-get() and empty($tickers)
        return {
          response:status-code(404);
          ()
        }
        default return {
          response:content-type("text/plain");
          "Method not supported."
        }
```

The above query reacts on the method and on whether companies are found to demonstrate a few of the available features:

- If the method is GET and facts are found, it returns them as JSON.

  GET `http://project-name.xbrl.io/rest/more-methods.jq?t=wmt&t=GOOG`

- If the method is GET and no facts are found, it returns a 404 NOT FOUND.

  GET `http://project-name.xbrl.io/rest/more-methods.jq?t=dummy`

- If the method is different, it returns a message in plain text.

  DELETE `http://project-name.xbrl.io/rest/more-methods.jq?t=wmt&t=GOOG`

That's it for now!

With these building blocks, and together with the 28.io platform and modules documentation, you are all set! Do not hesitate to ask questions on our Zendesk support platform. We'll be very happy to assist you.

Index