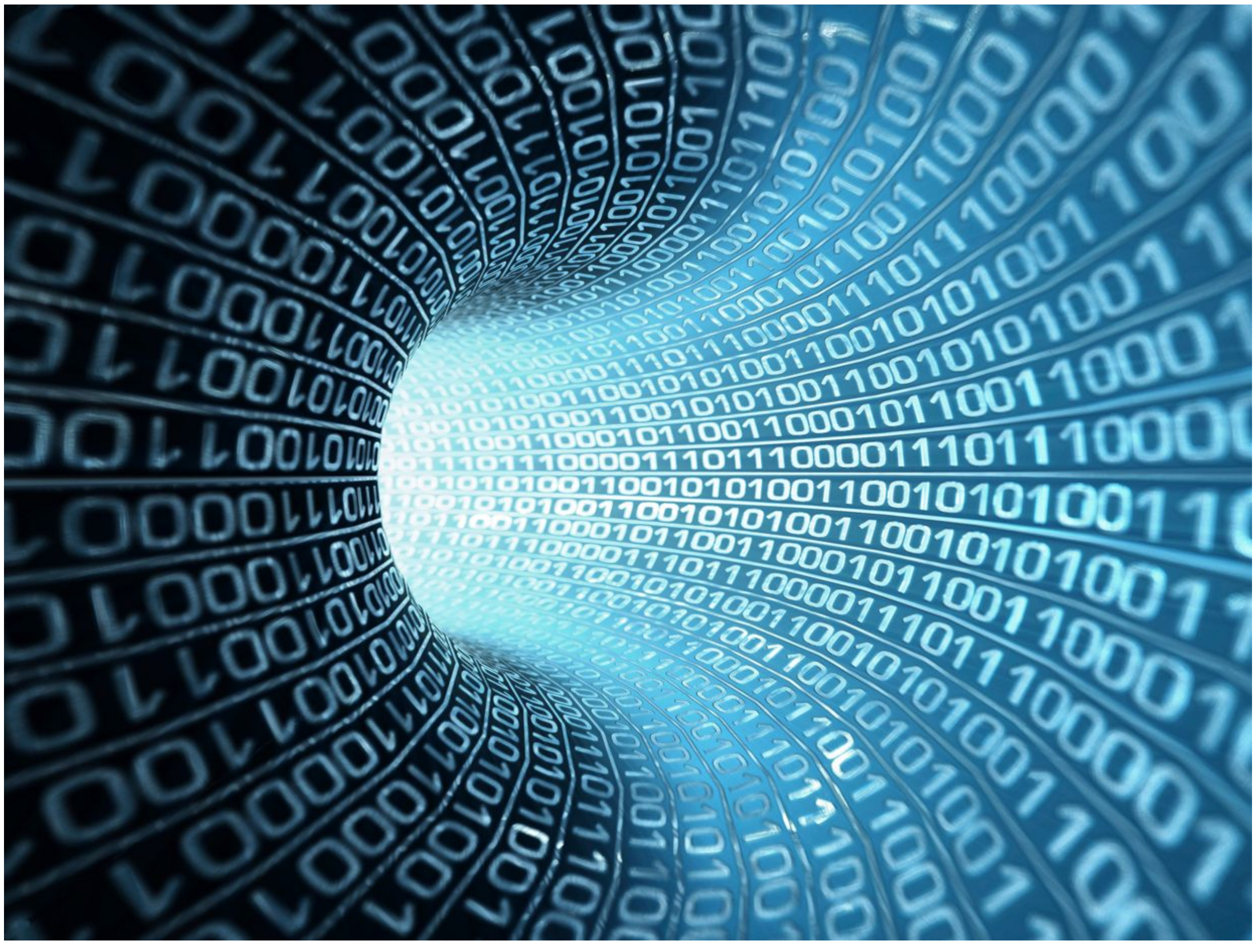
XBRL Application Programming Interface (API)

Version 1



XBRL US

XBRL API - Version 1

1 Overview

The XBRL API is designed to standardize the method used to request XBRL data from any database containing XBRL-formatted data. Prior to the introduction of a standard API, developers building an XBRL application had to take the following steps:

1. Acquire the data from an authoritative source
2. Process that data
3. Load that data into a database using a transformation and load process.
4. Generate a reporting mechanism to extract the stored information in meaningful ways.
5. Establish a database maintenance plan to accommodate software updates, document restatements and general operational maintenance.

Any developer interested in building an XBRL application historically needed to build this data collection infrastructure.

Alternatively, a software developer interested in developing an XBRL application could negotiate with a data utility that offers XBRL-formatted data, and build an application based on that utility's proprietary infrastructure. This represents a significant barrier to entry because the developer must negotiate a data agreement even before starting to develop software, and more importantly, adopting proprietary infrastructure effectively locks the developer into relying on a single platform, limiting the ability to easily add additional data sources or switch providers.

The purpose of the XBRL API is to provide a unified interface to stored XBRL data which any data utility can adopt. This unified interface allows developers with limited XBRL knowledge to learn a single interface to access many data repositories. This expands access to the market for available data, while also encouraging the use of fundamental structured data.

The XBRL API can be used in several ways. The API can be used with a stand alone REST client; it can be integrated into new software; or it can be accessed by web software. To access the API, each use must be authenticated. After authentication, queries can be run by submitting combinations of urls and parameters.

2 Structure of the API

The API is structured to allow you to return the details of XBRL objects. These objects are used to classify XBRL data. These objects generally fall into two categories: facts and data taxonomy. Each object has multiple properties that can be used to define the query. Some are shared and others are unique to the object. Objects can also be nested allowing specific information about the main object to be retrieved.

2.1 Types

Factual Data

Reported values that would appear in an XBRL instance are considered “factual data”. For example, 0.85 is factual data in an XBRL instance representing Basic earnings per share for Microsoft. Additional information is included in the API to assist in comparability. The object name to access this information is: fact. Factual data also includes objects related to how the data is organized. For example collections of data can be grouped in a report. Information about a report is classified using the report object. Data applicable to a specific entity who is reporting is also captured and the details are in the entity object.

Taxonomy Data (Metadata)

All factual data has metadata associated with it, which can include labels, references, calculations between concepts, how the data is presented, which base taxonomies are used, etc. The objects to access this information are: dts, concept, label, reference, parts, network and relationship. These objects can be used to access all the data that would normally be defined in an XBRL taxonomy.

2.2 Object Property Format

Each object has properties that can be accessed by requesting them from the API. The property of the object is defined using a dot notation. To get the value of a property for an object, the following format is used: *Object.property.* For example, to return the value of the fact, use “fact.value”. These properties can also be used to search for data. Not all properties of the object are searchable.

Some of the properties may be defined as components of other objects and are inherited from these objects.

2.3 Nested Objects

Nested objects allow you to pull sub-object information from the main object. Sub-objects that can be used are dependent on the main object. For example, facts can be returned for specific reports, and relationships can be returned for specific networks. The nesting structure of the API is discussed in the rest of the document.

3 Global Query Parameters

In addition to being able to access data through a specific object, data that is returned via the API can be controlled by defining the data to be returned, indicating how many records should be returned, and in which sequence they should be returned.

3.1 Fields

By default, not all fields are returned when you make a query. You can choose the fields that you want returned with the fields query parameter. This is useful for making your API calls more efficient.

```
/api/v1/fact/{fact-id}?fields=fact.value
```

This will return the value of a fact for a given fact id. The fact id is defined as part of the end point i.e. {fact.id}

The fields parameter allows as many fields to be returned by the user as they need. Each field requested is separated using a comma without white space. The order of the fields returned is based on the position in the fields string. For example, submitting the request below will return the value of the given fact first, and the concept name second.

```
/api/v1/fact/141024005?fields=fact.value,concept.local-name
```

The response in json would be as follows:

```
[
        {
        "fact.value": "469033000",
        "concept.local-name": "Assets",
        "fact.id": 141024005
        }
]
```

Note that the fact.id is also returned, as this was passed as a parameter.

To return all the properties of an object, a wild card of * can be used with the property, for example:

```
/api/v1/fact/{fact-id}?fields=fact.*
```

This will return all the properties of the fact object in alphabetical order. Note that this will include properties of other objects associated with the fact such as concept.local-name or period.instant or report.id.

3.2 Sort

Any value returned can be sorted in ascending or descending order, by adding an additional property to a field value. In the following example, we want to return all filings made by a specific company, sorted in order from the latest filing to the earliest. To do this we sort based on the timestamp that the filing was received. For example:

```
api/v1/report/search?entity.cik=0001493040&fields=report.*,report.accepted-
timestamp.sort(DESC)
```

This call searches the report object for a specific entity identifier called a CIK[1] and returns the report details. Note that the fields include *report.*.* This will return all the properties of the report object. In addition, the *report.accepted-timestamp* field is defined with a sort property with a parameter of DESC. This means the call will return all reports for this CIK sorted by accepted timestamp in a descending order. Multiple sort criteria can be defined and the sort sequence is determined by the order defined in the fields parameter. In the above example, we could sort by document type, then by accepted timestamp as follows:

```
fields=report.*,report.document-type.sort(DESC),report.accepted-timestamp.s
ort(DESC)
```

Fields can be sorted either in descending (DESC) or ascending (ASC) order.

3.3 Limit

In addition to sorting on a field, it is possible to limit the number of records returned for a given object. A limit can only be added to an object type and not a property. For example, to limit the number of reports in the above query the limit property would appear on the report as follows:

```
report.limit(10)
```

A limit **cannot** be used on a property like report.document-type:

[1] Central Index Key which is a 10-digit number used on the Securities and Exchange Commission's computer systems to identify corporations and individuals who have filed disclosure with the SEC.

```
report.document-type.limit(10)
```

To limit the number of records returned to 2, the following syntax is used:

```
fields=report.*,report.document-type.sort(DESC),report.limit(2)
```

3.4 Search

The search keyword is used as an endpoint in many URIs. This indicates to the API that you will be searching on a property of the object. A search can be performed on one object at a time, and is based on the first object that appears in the URI.

For example, in the following uri, entity.cik is the property searched on with the report object:

```
/api/v1/report/search?entity.cik=0001493040&fields=report.*
```

Multiple parameters can be included in a search. In the case below, the API will return facts for a given CIK for Assets for 2017 in the first quarter:

```
/api/v1/fact/search?concept.local-name=Assets&period.fiscal-year=2017&perio
d.fiscal.period=1Q&Entity.cik=0000001&fields=fact.value
```

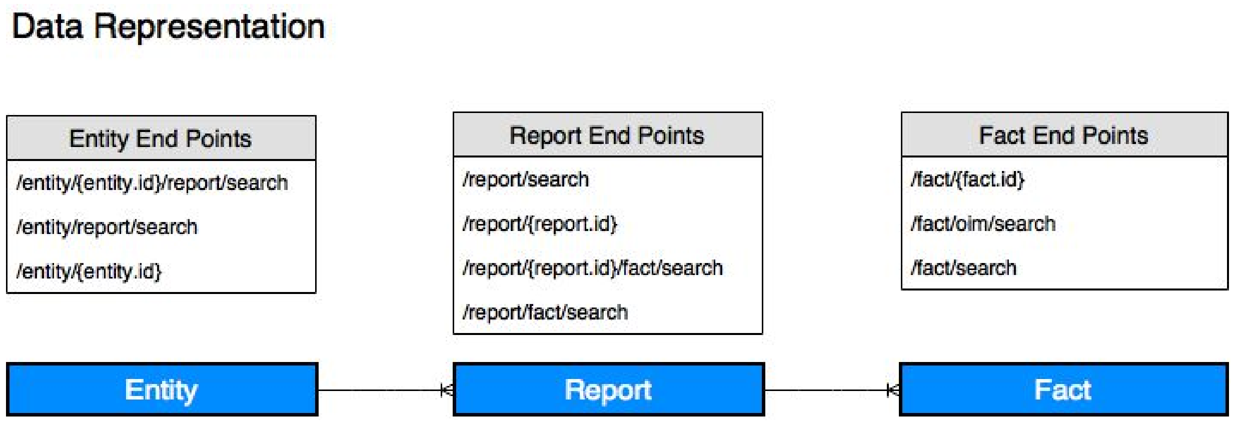
A user cannot search on a property that is not a component of the object. In the example above the endpoint is referencing the fact object (fact appears after v1) so a user must search on a valid fact property. If a user searched on the property network.arcrole-uri the result would fail with the following message:

```
{"status": "Invalid Parameter",
"body": "The url contains the search parameter network.arcrole-uri. Only the
following search parameters can be used with this url: period.calendar-period,
period.fiscal-period, period.year, period.fiscal-year, period.id, period.fiscal-id,
concept.namespace, concept.local-name, concept.id, concept.is-base,
concept.is-monetary, report.id, report.accession, report.entry-url,
report.restated-index, report.restated, report.sec-url, report.sic-code,
entity.cik, entity.id, fact.id, fact.value, fact.ultimus-index, fact.ultimus,
fact.has-dimensions, fact.is-extended, fact.hash, unit, dts.id, dts.entry-point,
dts.target-namespace, dimensions.id, dimension.namespace, dimension.local-name,
member.local-name."
}
```

Note that the error message returns all the valid search criteria of the fact object.

4 Factual Objects

Factual objects are used to represent data that is not defined in an XBRL taxonomy or metadata. The API includes a fact object, a report object and entity object. The fact object is used for data from any source. The report object is only applicable when data is defined in a report. The following diagram shows the end points to access data in each of these objects



4.1 Fact Object

Facts can be accessed via three fact object end points.

A fact can be returned based on its identifier:

```
/api/v1/fact/{fact-id}
```

Or a fact can be returned by searching for a set of facts based on search parameters:

```
/api/v1/fact/search
```

The API allows the user to search on a number of properties of a fact but not on all of them. The properties that can be searched on are listed below:

A fact can be returned based on search parameters:

```
/api/v1/fact/search?concept.local-name=Assets&period.fiscal-year=2017&perio
d.fiscal-period=1Q&entity.cik=000000001&fields=value, report.id
```

This will return facts for assets in the first quarter of fiscal year 2017 for the entity with a CIK of 000000001.

Facts can also be accessed from the report object:

```
/api/v1/report/190220/fact/search?period.fiscal-year=2016&fields=report.id,
report.document-type,report.address,report.filing-date.sort(DESC),report.en
tity-name,entity.cik,concept.local-name,fact.value,unit,period.fiscal-perio
d,period.fiscal-year,dimensions,fact.limit(10)
```

This will return all fact values for the given report 190220 where the fiscal year is equal to 2016 and will limit the fact records to 10 items.

Fact Fields

The Fact object has the following attributes that can be returned as fields:

Fields	Search	Description
period		
period.start		Start date.
period.end		End date.
period.instant		Point in time date.
period.calendar-period	**T**	Calendar period i.e 1Q.
period.fiscal-period	**T**	Fiscal period i.e 1Q.
period.year	**T**	Year of the fact.
period.fiscal-year	**T**	Fiscal year of the fact.
period.id		Id of the period.
concept		
concept.namespace	**T**	The full namespace of the concept.
concept.local-name	**T**	The local name of the concept.
concept.period-type		The period type of the concept.
concept.datatype		The datatype of the concept.

concept.balance-type		The balance type of the concept.
concept.id	**T**	Object id of the concept.
report		
report.id	**T**	Object id of the report.
report.accession	**T**	Accession identifier associated with the report if applicable.
report.filing-date		Filing date of the report.
report.type		Report type name, i.e. 10-K.
report.url		URL location of the report.
report.restated-index	**T**	The restated index associated with the fact.
report.restated	**T**	Boolean indicator if the report is restated.
entity		
entity.cik	**T**	The CIK (Central Index Key) of the entity.
entity.id	**T**	Object id of the entity.
entity.name		Legal name of the entity.
entity.scheme		Scheme of the identifier.
fact		
fact.id		Object id of the fact.
fact.value		Returns the value of the fact.
fact.decimals		Returns the decimals of the fact if applicable.
fact.xml-id	**T**	Returns the xml-id of the fact if applicable.
fact.ultimus-index	**T**	The ultimus index associated with the fact.
fact.ultimus	**T**	Boolean indicator if it is the last

		reported fact.
fact.has-dimensions	**T**	Boolean indicator if the fact has an associated dimensional breakdown.
unit		Returns the qname measure of the unit. Returns the numerator if it is a fraction.
unit.symbol		Returns the symbol of the fact.
unit.numerator		Returns the numerator of the unit.
unit.denominator		Returns the denominator of the unit.
dts		
dts.id	**T**	The dts id associated with the fact.
dts.entry-point	**T**	The url entry point associated with the dts of the fact.
dts.target-namespace	**T**	Namespace of the DTS.
dimensions		Returns a dictionary of key values, pairs of dimension concepts and member concepts.
dimension.local-name	**T**	The name of a dimension associated with a fact. (Use of this in a field can duplicate records where a fact has multiple dimensions associated with it, the dimensions field can be used instead to get a group).
member.local-name	**T**	The name of a member associated with a fact. (Use of this in a field can duplicate records where a fact has multiple dimensions associated with it, the dimensions field can be used instead to get a group).
dimensions.id	**T**	Object id of the dimension.
dimensions.count		Number of dimensions associated with the fact.

4.2 Report Object

Reports can be accessed via four report object end points.

A report can be returned based on its identifier:

```
/api/v1/report/{report-id}
```

Or a report can be returned by searching for a report based on parameters such as sic code and document type. The following request will return the details of the reports for sic code 4911 which was a 10-K[2]:

```
/api/v1/report/search?&report.sic-code=4911&report.document-type=10-K&repor
t.period-index=1,2,3,4&fields=report.*,report.accepted-timestamp.sort(DESC)
```

The period index is used to pull the last 4 reports as the 10-K will be one of the last 4 reports.

A report can also be returned using information from the fact object. This is useful as it allows a user to search for reports that contain specific information that is not associated with the report object. It also allows the user to return all the facts associated with a given report.

```
/api/v1/report/fact/search?report.is-most-current=true&report.sic-code=3841
&concept.local-name=EntityFilerCategory&fact.value=Smaller%20Reporting%20Co
mpany&fields=report.entity-name,entity.cik,fact.value
```

Lastly the following entry point can be used to search on a fact if the report identifier is known:

```
/api/v1/report/{report.id}/fact/search
```

Report Fields

The report object has the following attributes that can be returned as fields or used to query the object:

Fields	**Search**	**Description**
dts		

[2] A Form 10-K is an annual report required by the U.S. Securities and Exchange Commission (SEC), that gives a comprehensive summary of a company's financial performance.

dts.id	**T**	The taxonomy identifier used to produce the report.
entity		
entity.cik	**T**	The CIK (Central Index Key) of the entity.
entity.id	**T**	Object id of the entity.
entity.ticker	**T**	Ticker of the entity.
report		
report.accepted-timestamp		Acceptance date of the report.
report.id	**T**	Object id of the report.
report.accession	**T**	Accession identifier associated with the report if applicable.
report.address		Business Address of the reporter.
report.base-taxonomy		Taxonomy the report uses.
report.filing-date		Filing date of the report
report.document-type	**T**	Report type name, i.e. 10-K
report.entity-name	**T**	Name of the reporting entity
report.entry-type	**T**	
report.entry-url	**T**	The url entry point of a discoverable taxonomy set. This is also referred to as the entry point for a taxonomy. This represents the DTS entry point for a specific report.
report.filing-date		The date that the filing was published.
report.is-most-current	**T**	Boolean that indicates if the report is the latest report.
report.period-end		The balance date of the report.
report.period-index	**T**	A sequence that indicates the relative age of the report with 1 being the most recent report.

report.restated-index	**T**	A numerical indicator that can be used to identify if a report has been restated. If the value is 1 it indicates that this is the latest report. If the value is 2 it means that an updated copy of the report has been filed.
report.restated	**T**	Boolean indicator if the report is restated.
report.sec-url	**T**	The unique url at which the report can be accessed from the internet.
report.sic-code	**T**	The industry SIC code associated with the entity when the report was filed.

Nesting of Facts

Property values of the fact object are nested when the report object calls the fact object using the following endpoint:

`/api/v1/report/fact/search?`

Any properties that are common to both objects such as dts.id are consolidated into the upper most level to avoid repetition. This happens where properties are shared between two objects, when they are nested. For example, the following call requests a report and the value of assets in the report 190220.

```
/api/v1/report/190220/fact/search?concept.local-name=Assets&fact.has-dimens
ions=false&fields=report.document-type,report.filing-date.sort(DESC),entity
.cik,period.year,dts.id,fact.value
```

This will return a nested json object with the report information encapsulating the fact information. The dts.id, for example, is part of the fact and report object, but will only appear as part of the report object. The figure below shows what the return object looks like in a json format. Note that the fact.id is always returned when a nest fact object is returned.

If the entity object calls the report object the same nesting logic applies.

```
{
      "report.document-type": "10-Q",
      "report.filing-date": "2017-06-06",
      "entity.cik": "0001090872",
      "dts.id": 256944,
      "fact": {
            "data": [
                        {
                        "period.year": 2017,
                        "fact.value": "8016000000",
                        "fact.id": 153589871
                        },
                        {
                        "period.year": 2016,
                        "fact.value": "7794000000",
                        "fact.id": 153590308
                        }
                  ]
            }
}
```

4.3 Entity Object

Entities can be accessed via three entity object end points. First, an entity can be returned based on an entity identifier:

```
/api/v1/entity/{entity.id}
```

The entity.id is an internal id used by any system and should only be used internally by an application after the entity id has been determined from another API call.

The second endpoint allows the searching of entities based on the entity objects properties:

```
/api/v1/entity/report/search
```

This end point also allows the details of a report associated with the entity to be returned. The following will give a listing of all the filing dates for all reports made by the entity with the ticker aray. It will also include all the properties of the entity object. This is because the wildcard “*” is used on the entity object.

```
/api/v1/entity/report/search?entity.ticker=aray&fields=entity.*,report.filing-date
```

The third entry point is similar to the second, except the {entity.id} can be passed as a parameter to the endpoint:

```
/entity/{entity.id}/report/search
```

Entity Fields

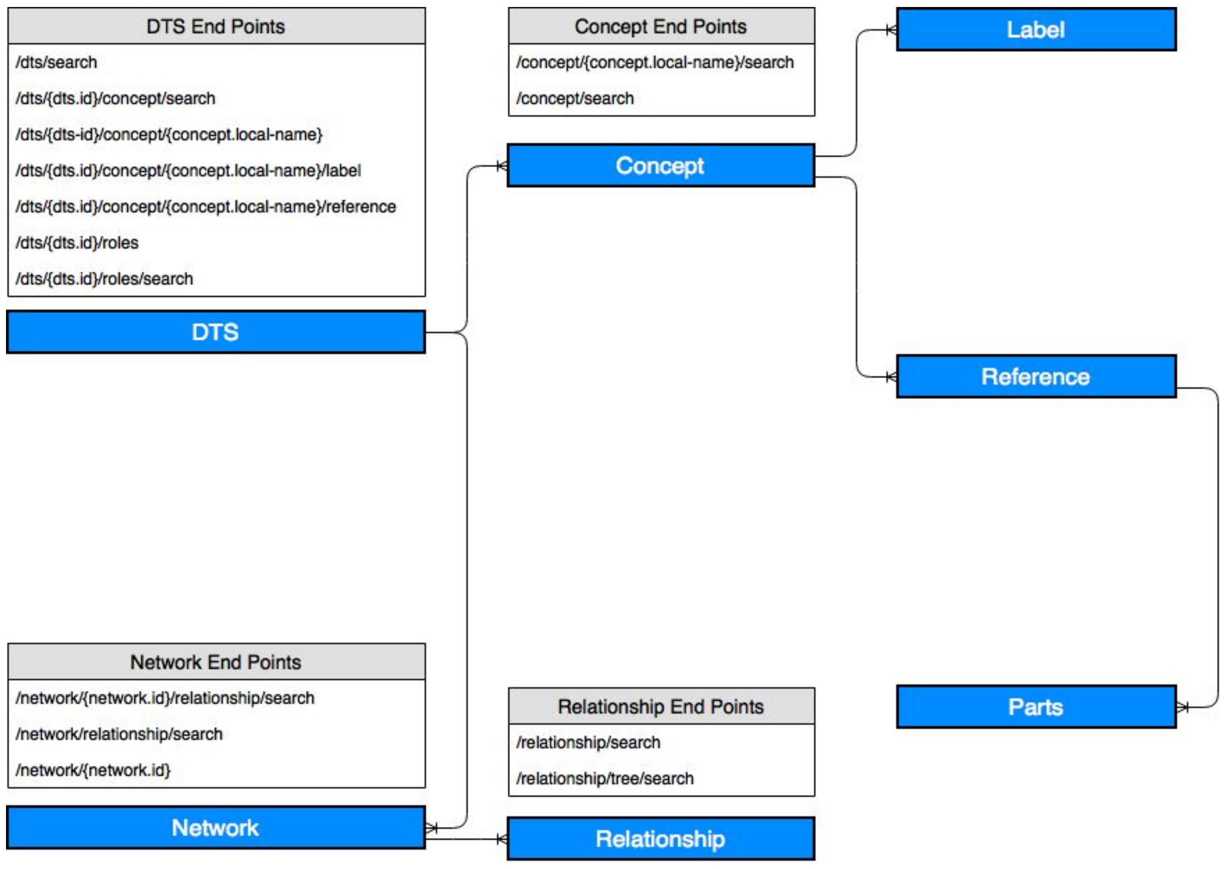
The report object has the following attributes that can be returned as fields or used to query the object:

Fields	Search	Description
entity		
entity.cik	**T**	The CIK (Central Index Key) of the entity.
entity.id	**T**	The internal identifier used to identify an entity. This will be replaced with the LEI[3] when the SEC supports the LEI standard.
entity.name	**T**	The name of the entity reporting.
entity.scheme		The scheme of the identifier associated with a fact, report or DTS. A fact could have multiple entity identifiers and this indicates the identifier that was used.
entity.ticker	**T**	Ticker of the entity.

[3] The Legal Entity Identifier (LEI) is the International ISO standard 17442. LEIs are identification codes that enable consistent and accurate identification of all legal entities that are parties to financial transactions, including non-financial institutions.

5 Taxonomy (Metadata) Objects

Taxonomy objects are used to represent data that is defined in an XBRL taxonomy or metadata. The API includes a Concept object, a DTS object, a Label object, a Reference object, a Reference Parts Object, a Network Object, a Relationship Object, and a Document Object. Each of these objects is explained below. The following diagram shows the end points to access data in each of these objects.



DTS End Points
/dts/search
/dts/{dts.id}/concept/search
/dts/{dts-id}/concept/{concept.local-name}
/dts/{dts.id}/concept/{concept.local-name}/label
/dts/{dts.id}/concept/{concept.local-name}/reference
/dts/{dts.id}/roles
/dts/{dts.id}/roles/search
DTS
Concept End Points
/concept/{concept.local-name}/search
/concept/search
Concept
Label
Reference
Parts
Network End Points
/network/{network.id}/relationship/search
/network/relationship/search
/network/{network.id}
Network
Relationship End Points
/relationship/search
/relationship/tree/search
Relationship

5.1 Concept Object

Concepts can be accessed via two concept object end points.
A concept is an object that can be referenced by a concept object identifier.

```
/api/v1/concept/{concept-id}/search?fields=concept.local-name,
concept-namespace
```

This will return the concept properties requested using the fields parameter for every dts. For this reason a dts should always be used in the search parameter.

To search on the properties associated with the concept object, the second search endpoint is used. The format of the second end point is as follows:

```
/api/v1/concept/search?
```

```
/api/v1/concept/search?concept.local-name=Assets&dts=1234&fields=concept.lo
cal-name, dts.id
```

This will return the details of the concept Assets in the dts 1234.

All concepts appear in a dts either of a filing or the base taxonomy. To get the details of a concept, the dts should be provided. If no dts is provided, the API will return every dts a concept participates in, as this could result in thousands of dts results. For example, each extension taxonomy is a dts. If no dts is provided, the API will return all the dts' associated with the concept.

Concept Fields

The concept object has the following attributes that can be returned as fields:

Fields	Search	Description
concept		
concept.balance-type		The balance type of a concept. This can be either debit, credit or not defined.
concept.datatype		The datatype of the concept such as monetary or string.
concept.id	**T**	A unique identification id of the concept that can be searched on. This is a faster way to retrieve the details of a fact, however it is namespace specific and will only search for the use of a concept for a specific schema.
concept.is-abstract		True if the concept is an abstract.

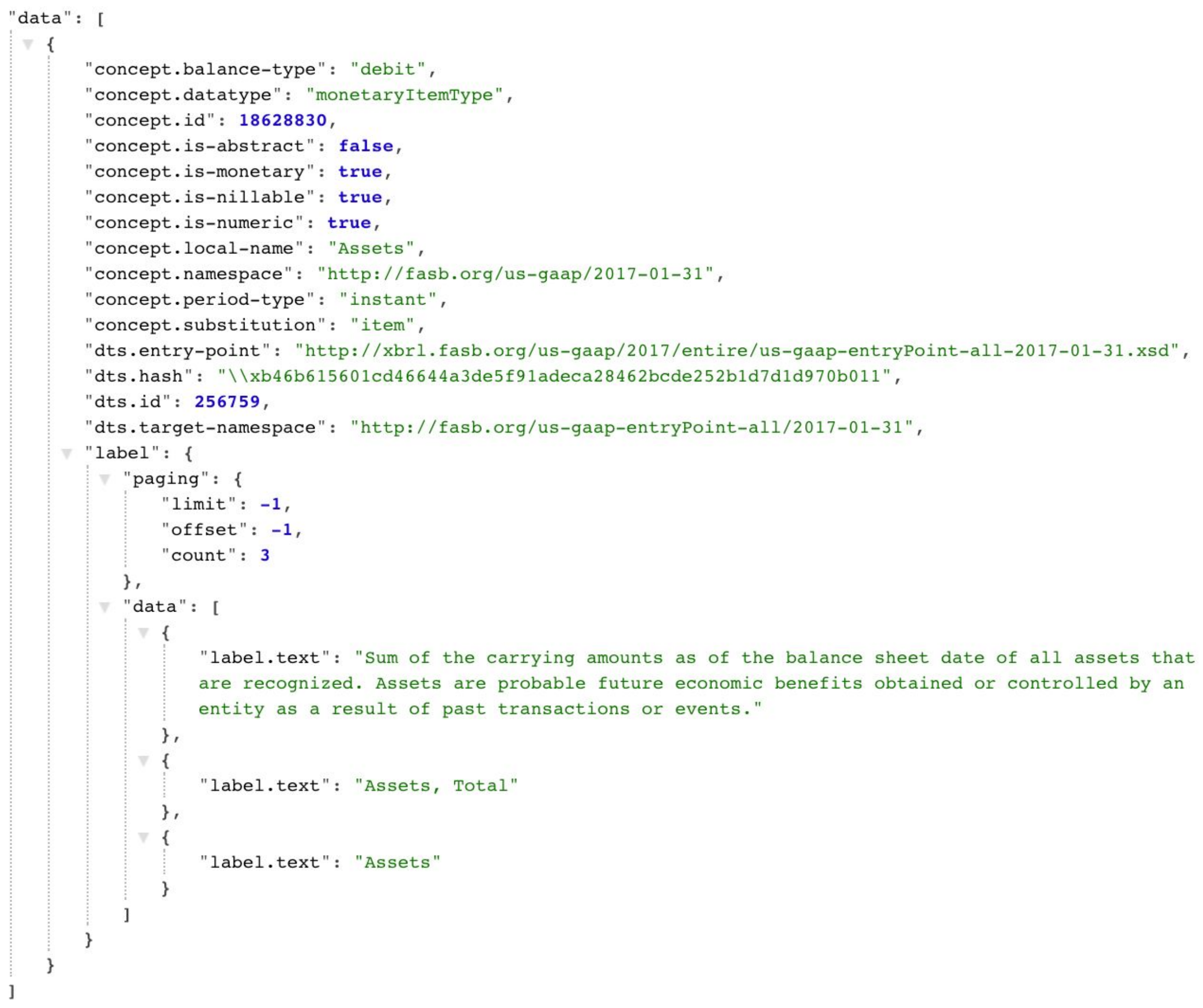
concept.is-monetary		True if is monetary.
concept.is-numeric		True if is numeric.
concept.local-name	**T**	The concept name in the base schema of a taxonomy excluding the namespace, such as Assets or Liabilities. Use this to search across multiple taxonomies where the local name is known to be consistent over time.
concept.namespace	**T**	The full namespace of the concept.
concept.period-type		The period type of the concept. This can be either duration or instant.
concept.substitution		Substitution group of the concept.
dts		
dts.id	**T**	The dts id associated with the concept.
dts.entry-point	**T**	The url entry point associated with the dts of the concept.
dts.hash	**T**	Hashed canonical key of the taxonomy dts.
dts.target-namespace	**T**	The target namespace of a discoverable taxonomy set. (DTS)

Nesting of Labels & References

Return fields from the concept object can also specify properties of the label, references and parts objects. If the dts is provided then these will be returned as nested children of the concept. For example the following will return the details of the Assets concepts and its label text.

```
/api/v1/concept/Assets/search?dts.id=256759&fields=concept.*,label.text
```

The data that is returned is shown in the figure below:



```
"data": [
  {
    "concept.balance-type": "debit",
    "concept.datatype": "monetaryItemType",
    "concept.id": 18628830,
    "concept.is-abstract": false,
    "concept.is-monetary": true,
    "concept.is-nillable": true,
    "concept.is-numeric": true,
    "concept.local-name": "Assets",
    "concept.namespace": "http://fasb.org/us-gaap/2017-01-31",
    "concept.period-type": "instant",
    "concept.substitution": "item",
    "dts.entry-point": "http://xbrl.fasb.org/us-gaap/2017/entire/us-gaap-entryPoint-all-2017-01-31.xsd",
    "dts.hash": "\\xb46b615601cd46644a3de5f91adeca28462bcde252b1d7d1d970b011",
    "dts.id": 256759,
    "dts.target-namespace": "http://fasb.org/us-gaap-entryPoint-all/2017-01-31",
    "label": {
      "paging": {
        "limit": -1,
        "offset": -1,
        "count": 3
      },
      "data": [
        {
          "label.text": "Sum of the carrying amounts as of the balance sheet date of all assets that
          are recognized. Assets are probable future economic benefits obtained or controlled by an
          entity as a result of past transactions or events."
        },
        {
          "label.text": "Assets, Total"
        },
        {
          "label.text": "Assets"
        }
      ]
    }
  }
]
```

To get the reference parts associated with the concept as well as the parts, the object can be added as a return field such as the following:

```
/api/v1/concept/Assets/search?dts.id=256759&fields=concept.*,label.text,
parts.*
```

The table below shows the properties that can be returned as fields associated with the concept object. In fact, these are the properties of the label, reference, and parts objects.

Fields	Search	Description
labels		Returns the label object associated with the concept.
label.id		The id of the label.

label.text		Text of a label.
label.lang		The language of a label.
label.role	**T**	The role of the label.
label.role-short	**T**	The end name of the label-role.
reference		
reference.id	**T**	The identifier of the reference.
reference.role	**T**	Role of the reference such as presentation.
reference.role-definition		Definition of the reference role.
reference.role-short		Short name of the reference role.
parts		
parts.local-name		The local name of the part such as “Publisher”.
parts.namespace		The namespace of the part such as “http://www.xbrl.org/2006/ref”.
parts.order		The sequence order of the part.
parts.part-value		The value of the part such as “FASB.
reference.id	**T**	The reference id associated with the part.

5.2 DTS Object

The details of the dts or taxonomy object can be accessed via seven end points.

The first endpoint enables a search for a dts based on the properties associated with the dts object. The endpoint is as follows:

```
/api/v1/dts/search
```

This endpoint can be used to look up the dts details of a taxonomy like the US GAAP taxonomy as follows:

```
/api/v1/dts/search?dts.taxonomy-name=US%20GAAP%202017&fields=dts.*
```

This will return the dts details of the US GAAP 2017 taxonomy.

The second endpoint enables returning the details of concepts associated with the US GAAP taxonomy based on the properties of the concept object.

```
/api/v1/dts/{dts.id}/concept/search
```

The third endpoint can be used to lookup all the concepts or a single concept in a specific taxonomy by providing the dts id. The following returns the local names of all the elements in the US GAAP 2017 taxonomy. The id of 256759 is the US GAAP taxonomy.

```
/api/v1/dts/256759/concept/search?fields=concept.local-name
```

The following returns the details of Assets from the US GAAP taxonomy and the text label.

```
/api/v1/dts/256759/concept/search?concept.local-name=Assets&fields=concept.
*,label.text
```

The following entry point is a variation on the above, but includes the element name in the endpoint.

```
/api/v1/d/dts/{dts-id}/concept/{concept.local-name}
```

A query using this endpoint for Assets would look like the following:

```
/api/v1/dts/256759/concept/Assets?fields=concept.limit(10).concept.local-name.sort(
ASC),concept.period-type,concept.substitution,concept.datatype,concept.is-abstract,
concept.id,reference.limit(2),reference.role,parts,parts.*
```

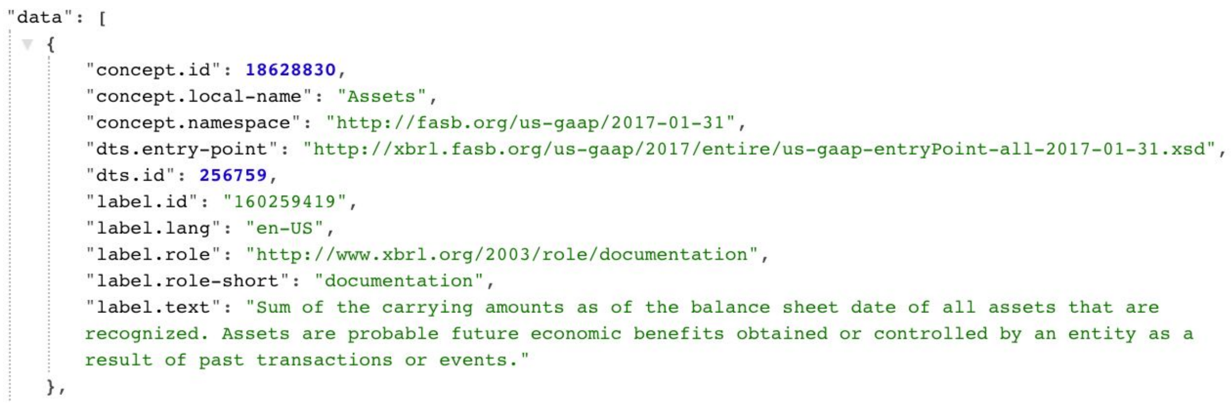
This fourth endpoint will return all data in the label object for a specific concept in a specific dts.

```
/api/v1/dts/{dts.id}/concept/{concept.local-name}/label
```

For example, the following returns the label objects for Assets.

```
/api/v1/dts/256759/concept/Assets/label?fields=label.*
```

The resulting data is as follows:



```
"data": [
  ▼ {
      "concept.id": 18628830,
      "concept.local-name": "Assets",
      "concept.namespace": "http://fasb.org/us-gaap/2017-01-31",
      "dts.entry-point": "http://xbrl.fasb.org/us-gaap/2017/entire/us-gaap-entryPoint-all-2017-01-31.xsd",
      "dts.id": 256759,
      "label.id": "160259419",
      "label.lang": "en-US",
      "label.role": "http://www.xbrl.org/2003/role/documentation",
      "label.role-short": "documentation",
      "label.text": "Sum of the carrying amounts as of the balance sheet date of all assets that are
      recognized. Assets are probable future economic benefits obtained or controlled by an entity as a
      result of past transactions or events."
    },
```

This fifth endpoint will return all data in the reference object for a specific concept in a specific dts.

```
/api/v1/dts/{dts.id}/concept/{concept.local-name}/reference
```

The sixth endpoint is used to return network or graph information. This will return network trees within a specific dts.

```
api/v1/dts/{dts.id}/network
```

The seventh endpoint allows you to define the specific networks that you want returned from a dts using a search.

```
api/v1/dts/{dts.id}/network/search
```

For example to return all the calculation networks and associated relationships, the following api call can be made.

```
/api/v1/dts/177604/network/search?network.link-name=calculationLink&fields=
network.arcrole-uri,network.link-name,network.role-description.sort(ASC),ne
twork.role-uri,network.id,relationship.*
```

Because the relationship object fields are requested with “relationship.*” field parameter, the relationships between concepts will also be returned for each calculation network.

DTS Fields

The dts object has the following attributes that can be returned as fields:

Fields	Search	Description
dts		
dts.entity-name		The entity name associated with the dts.
dts.hash	**T**	Hashed canonical key of the taxonomy dts.
dts.id	**T**	The dts id associated with the concept.
dts.entry-point	**T**	The url entry point associated with the dts of the concept.
dts.taxonomy	**T**	The broad name of the taxonomy such as “US GAAP” or “Solar”.
dts.taxonomy-name	**T**	Specific name of the taxonomy and version year such as US GAAP 2012.
dts.version		The version number of the taxonomy.
report.accession	**T**	The source report identifier associated with the report.
report.id	**T**	The report identifier that uses a given taxonomy.

5.3 Network Object

The details of the network object and associated relationships can be accessed via three endpoints.

The first endpoint enables a search for a network based on the properties associated with the network object. The endpoint is as follows:

```
/api/v1/network/relationship/search
```

This endpoint will always return nested relationships associated with the network. Even if no relationship attribute is provided, the relationship.id will be returned.

This endpoint is used to return the relationships associated with a given network such as the following:

```
/api/v1/network/relationship/search?network.id=27624452&fields=network.link
-name,network.id,relationship,relationship.id.sort(ASC),relationship.*
```

Because the network.id is specific to a dts, the dts.id does not have to be provided. However the dts.id can be provided as a search parameter, which would make this similar to the end points associated with the dts object.

The second endpoint allows you to return the details of a network if you know the network id.

```
/api/v1//network/{network.id}
```

This endpoint for example is used as follows to return the details of a specific network.

```
/api/v1/network/27624452?fields=network.*
```

If you ask for relationship information when using the above endpoint it will not be returned. To get relationship information, the third endpoint can be used, which allows you to search on specific relationships as well.

```
/api/v1/network/{network.id}/relationship/search
```

For example this endpoint can be used to search for relationships in a network that have a certain parent.

```
/api/v1/network/27624452/relationship/search?relationship.source-name=Balan
ceSheetComponentsDisclosureAbstract&fields=network.*,relationship.*,relatio
nship.limit(4),relationship.tree-sequence.sort(ASC)
```

Network Fields

The network object has the following attributes that can be returned as fields:

Fields	Search	Description
dts		
dts.entry-point	**T**	The entry-point associated with the dts.

dts.id	**T**	The dts id associated with the concept.
network		
network.arcrole-uri	**T**	URI that identifies the link types, such as parent-child. However, this is the full uri of http://www.xbrl.org/2003/arcrole/parent-child.
network.id	**T**	Unique integer identifier used to identify a specific network. A different identifier is used for networks with the same role but different linkbase types.
network.link-name	**T**	Name that identifies the link type. This corresponds to a linkbase i.e. presentationLink, calculationLink, definitionLink.
network.role-description		The human readable description of the network role. In some filing regimes this is used to order the networks.
network.role-uri	**T**	The URI of the network role. This would appear as a URI describing the reporting group i.e. http://www.bc.com/role/DisclosureBalanceSheetComponentsDetails.

5.4 Relationship Object

The details of the relationship object can be accessed via two endpoints.

The first endpoint enables a search for a relationship based on the properties associated with the relationship object. The endpoint is as follows:

```
/api/v1/relationship/search
```

This endpoint allows you to return data about the network and the relationships in it. The following example shows how you can return all the root nodes in the calculation relationships of a given taxonomy.

```
/api/v1/relationship/search?dts.id=177604&network.link-name=calculationLink
&relationship.tree-depth=1&relationship.order=1&fields=relationship.*
```

By using the tree depth attribute and order attribute, we return all the root nodes in a given network.

The second endpoint returns the same data as above but returns the resulting data in a nested tree rather than a flat list of relationships.

```
/api/v1/relationship/tree/search
```

In the example below, the API returns a tree representing a balance sheet presentation.

```
/api/v1/relationship/tree/search?network.id=27624452&fields=relationship.id
,relationship.source-is-abstract,network.id.sort(DESC)
```

This endpoint will always return some required fields, specifically the relationship.source-name, relationship.target-name, relationship.tree-depth, and relationship.tree-sequence. These fields are required to build the hierarchy and are always returned when this endpoint is used.

Relationship Fields

The relationship object has the following attributes that can be returned as fields:

Fields	Search	Description
dts		
dts.id	**T**	The dts id associated with the concept.
network		
network.arcrole-uri	**T**	URI that identifies the link types, such as parent-child. However, this is the full uri of http://www.xbrl.org/2003/arcrole/p

		arent-child.
network.id	**T**	Unique integer identifier used to identify a specific network. A different identifier is used for networks with the same role but different linkbase types.
network.link-name	**T**	Name that identifies the link type. This corresponds to a linkbase i.e. presentationLink, calculationLink, definitionLink.
network.role-description		The human readable description of the network role. In some filing regimes this is used to order the networks.
network.role-uri	**T**	The URI of the network role. This would appear as a URI describing the reporting group i.e. http://www.bc.com/role/DisclosureBalanceSheetComponentsDetails.
relationship		
relationship.id	**T**	A unique identifier associated with the relationship.
relationship.order	**T**	The order of the relationships relative to each other when viewed as a tree.
relationship.preferred-label	**T**	The preferred label attribute value associated with a relationship.
relationship.source-concept-id	**T**	The id of the concept that the relationship comes from.
relationship.source-is-abstract		The abstract indicator (boolean) of the concept that the relationship comes from.
relationship.source-name	**T**	The name of the concept that the relationship comes from.
relationship.source-namespace	**T**	The namespace of the concept that the relationship comes from.

relationship.target-concept-id	**T**	The id of the concept that the relationship goes to.
relationship.target-datatype		The datatype of the concept that the relationship goes to.
relationship.target-is-abstract		The abstract indicator (boolean) of the concept that the relationship goes to.
relationship.target-name	**T**	The name of the concept that the relationship goes to.
relationship.target-namespace	**T**	The namespace of the concept that the relationship goes to.
relationship.tree-depth	**T**	When viewed as a tree how many jumps is the relationship from the root node.
relationship.tree-sequence	**T**	The order in which the relationship appears in the entire network.
relationship.weight		The preferred label attribute value associated with a relationship.

6 Handling Paging

The API returns information about the number of records returned. In the following example the user limited the results to 10 records using the limit function defined earlier:

```
/api/v1/report/search?report.is-most-current=true&fields=report.*,report.li
mit(10),report.accepted-timestamp.sort(DESC)
```

As part of this query, the API returns paging information that shows the limit of 10, the offset and the count. The offset indicates the starting point. Note that the offset is set to -1. This means that the records returned were from the start and an offset was not provided to the API. See example below.

```
"paging": {
      "limit": 10,
      "offset": -1,
      "count": 10
},
```

To get the next 10 records, the API has to define the starting point. This is done using the offset function. In the previous request we got the first 10 reports sorted by the accepted timestamp. So the offset for the next request should be set to 10.

```
/api/v1/report/search?report.is-most-current=true&fields=report.*,report.li
mit(10),report.accepted-timestamp.sort(DESC),report.offset(10)
```

In the api request the report now has an offset of 10. This will return the following paging information as well as the data:

```
"paging": {
      "limit": 10,
      "offset": 10,
      "count": 10
},
```

It is important when offsetting data, that the data being returned is sorted. This ensures that records are not duplicated or missed when extracted from the database. This is currently the only pagination method supported by the API.

7 Error Messages

The API supports a number of error messages for the following situations:

7.1 Invalid Search parameter

If an invalid search parameter is provided the following message is returned.

```
{"status": "Invalid Parameter",
"body": "The url contains the search parameter network.arcrole-uri. Only the
following search parameters can be used with this url: period.calendar-period,
period.fiscal-period, period.year, period.fiscal-year, period.id,
period.fiscal-id, concept.namespace, concept.local-name, concept.id,
concept.is-base, concept.is-monetary, report.id, report.accession,
report.entry-url, report.restated-index, report.restated, report.sec-url,
report.sic-code, entity.cik, entity.id, fact.id, fact.value,
fact.ultimus-index, fact.ultimus, fact.has-dimensions, fact.is-extended,
fact.hash, unit, dts.id, dts.entry-point, dts.target-namespace, dimensions.id,
dimension.namespace, dimension.local-name, member.local-name."
}
```

7.2 Invalid Endpoint

If an invalid endpoint is provided, a 404 error is returned.

```
{"status": "404",
"body": "The requested URL
/api/v1/report2/search?report.is-most-current=true&fields=report.*,repo
rt.limit(10),report.accepted-timestamp.sort(DESC),report.offset(10) was
not found."}
```

7.3 Fields Attribute Missing

The API requires that the fields attribute is provided when returning data from the API.

```
{"status": "FieldsNotFound",
"body": "Fields parameter required"}
```

7.4 Invalid Search Value

The values provided as a search parameter must have the correct type. If the type differs from that expected an exception is returned.

```
{"status": "Invalid Parameter",
"body": "The value entered for dts.id of 256759a is not a valid
integer. "}
```

7.5 Invalid Object Name

The object name defined in the return fields must be a valid object name.

```
{"status": "Fields With Invalid Value",
"body": "Fields parameter using an asterisk must provide aa valid
object name, the object name provided of \"concept2\"  is not defined.
A valid object could be fact.*. The object name should match the name
after api version number. i.e. use concept based on :
/api/v1/concept/"}
```

7.6 Wildcard with no Object Name

A return field using the asterisk wildcard must have an object name. The entry fields=* is invalid and returns the following message:

```
{"status": "Fields With Invalid Value",
"body": "Fields parameter using an asterisk must provide an objectname
i.e. fact.*"}
```

7.7 Invalid Integer on a Limit or Offset Function

The value passed to the limit or offset functions must be an integer. If not the following is returned.

```
{"status": "Invalid Limit Parameter",
"body": "The limit parameter must be an integer"}
```

7.8 Invalid Sort Parameter

The value passed to the sort parameter must be either ASC or DESC in uppercase.

```
{"status": "Invalid Sort Parameter",
"body": "The sort parameter must be ASC or DESC"}
```

7.9 Exceeded Limit Amount

The API limits the number of records that can be returned. Each user has a default that applies to them. If the limit is exceed in the limit function then an exception is reported with the users actual limit.

```
{"status": "Invalid Limit Amount",
"body": "The limit property of 2000 must be less than the user limit
amount. Your user limit amount is 1000 records."}
```

Using the XBRL API

8 Authentication

The XBRL US API and database uses two-legged OAuth2, an authorization framework that enables applications to obtain limited access to user accounts on an HTTP service, for authentication. You must have a client id and secret, as well as a username and password, created at the XBRL US website, to access the API.

To use the XBRL US API and database for your application, you must follow these steps:

1) **Authorize the Client:** Register yourself and obtain credentials
2) **Authenticate**: Validate your request and receive a token
3) **Make a Request**: Use the token when making requests to the API

8.1 Client Authorization

To obtain authorization, go to the XBRL US website (http://xbrl.us/apirequest) and log in. Click "Create Client", type in a short name to distinguish between ids and click "Submit" to generate a client id and client secret. Be sure to save the secret as there's no way to recover it if lost. You can have multiple applications access the API by creating a different code for each one. If the secret is lost, a new id and secret can be created. Be sure to delete the old id.

This page will have a list of the descriptions and client ids.

Authentication

Before using the API, you must generate an authentication token to be used with each request, by submitting an https POST using the "application/x-www-form-urlencoded" format to the url: *https://betaapi.xbrl.us/oauth2/token*.

Requesting a token

To initially receive a token submit the following information to the url above.

grant_type	REQUIRED. Value must be "password"
client_id	REQUIRED. Value received during client authorization
client_secret	REQUIRED. Value received during client authorization

username	REQUIRED. User who wants to use the resources
password	REQUIRED. Users password
platform	OPTIONAL. Keyword to distinguish if the user is authenticating from different applications.

The response will provide the following in JSON format

access_token	String used for future requests (Guid, 36B)
token_type	Always “bearer”
expires_in	Number of seconds before the access_token expires
refresh_token	String used to refresh access_token (Guid, 36B)
refresh_token_expires_in	Number of seconds before the refresh_token expires
platform	Keyword used when token was requested.

Refreshing a token

If the access_token expires, a new one can be requested by submitting the refresh token received during the initial authentication. The following information should be submitted to the same url as before:

grant_type	REQUIRED. Must be “refresh_token”
client_id	REQUIRED. Value received during client authorization
client_secret	REQUIRED. Value received during client authorization
refresh_token	REQUIRED. Refresh token received during validation
platform	OPTIONAL. Keyword to distinguish if the user is authenticating multiple times

The response matches the same response received during the original authentication.

8.2 Making a Request

When submitting a request to the API, a bearer token must be submitted in the header under the Authorization key including the access token acquired when it was either requested or refreshed.

Example:

```
GET /api/v1/report/190220/fact/search
Host: betaapi.xbrl.us
Authorization: Bearer <token>
```

The base uri for making requests is: *https://betaapi.xbrl.us*.

9 Constructing a Request for Information

A request for information is built of two parts. The uri defines what kind of request is being made and what object(s) is being queried. The query string defines other details of the request. The basic format is:

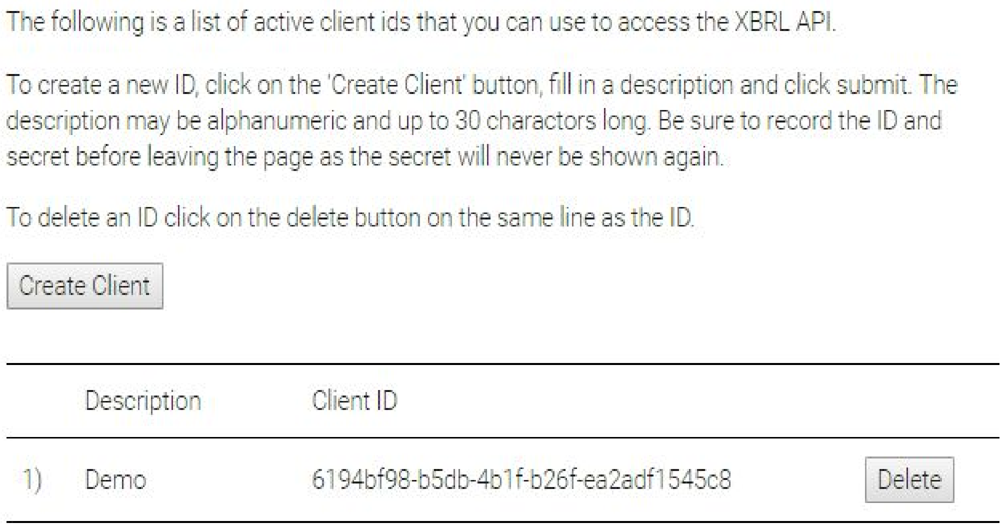
Part	**Format**	**Example**
URI	/api/v1/<object>/search	/api/v1/fact/search
Query	<object property>=value&fields=value	entity.cik=0001138723&fields=fact.id

10 Getting Started

Developers interested in accessing a beta version of the XBRL APIs can get started using this process:

These steps only need to be done once:

1. Request access to the XBRL APIs Beta by filling out this form: https://xbrl.us/home/use/data-analysis-toolkit/xbrl-api/
2. After receiving a response, if you don't have a login, then create a login on the xbrl.us web site: https://xbrl.us/wp-login.php?action=register
3. Get your XBRL US database API by visiting this page: https://xbrl.us/data-analysis-toolkit/ and logging in. Scroll towards the bottom of the page and click on the "Create Client" button, as shown in the diagram below, to get your API key. Save the information you receive.



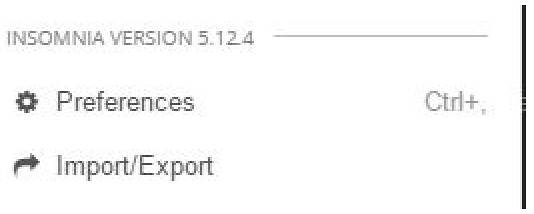
The following is a list of active client ids that you can use to access the XBRL API.

To create a new ID, click on the 'Create Client' button, fill in a description and click submit. The description may be alphanumeric and up to 30 charactors long. Be sure to record the ID and secret before leaving the page as the secret will never be shown again.

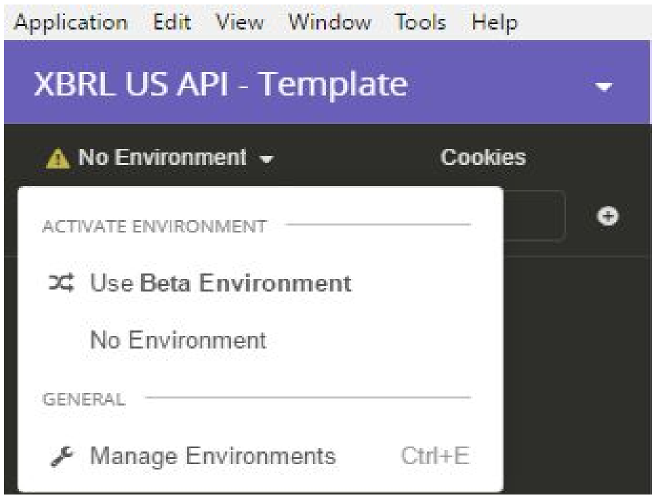
To delete an ID click on the delete button on the same line as the ID.

	Description	Client ID	
1)	Demo	6194bf98-b5db-4b1f-b26f-ea2adf1545c8	Delete

4. Download:
   a. The appropriate version of the Insomnia REST Client: https://insomnia.rest/download/
   b. Template file
5. After starting Insomnia click on the down arrow button on the right side of the purple area in the upper left. Chose Import/Export from the drop down menu.



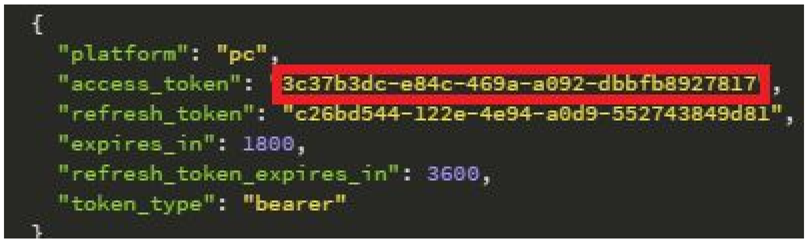
6. Click on Import->from file and chose the file you downloaded in 4b
7. Open the same dropdown menu and chose the workspace you imported. The name will be "XBRL US API - Template"
8. In the upper left under the purple area click on the arrow next to "No Environment" and chose "Manage Environments".



9. Click on Base Environment, then in the area on the right fill out using your website username and password as well as the information you received from the website (step 3): client_id, client_secret, password, username. Click "DONE" in the lower right of the menu.
10. Click the right arrow next to "No Environment" and chose "Use Beta Environment"

The following needs to be done when starting a session or if your token expires:

1. Request a token:
    a. In the left menu click on "Authentication API" and chose "Request Token" underneath it. Click on the Send button.
    b. On the right side the response will appear. Highlight and copy the access_token.



```
{
  "platform": "pc",
  "access_token": "3c37b3dc-e84c-469a-a092-dbbfb8927817",
  "refresh_token": "c26bd544-122e-4e94-a0d9-552743849d81",
  "expires_in": 1800,
  "refresh_token_expires_in": 3600,
  "token_type": "bearer"
}
```

    c. Go into the "Manage Environments" menu and select "Beta Environment". Replace the token value with value received in the previous step. Click Done.
2. You can now click on the "Meta Information" or "API Test Requests" folders to display example requests. Click send to receive the response in the right application window.