Important Note

This tutorial describes the functionality and API of FactSet's *internal* Quant Engine Service. FactSet clients interact with the *public* Quant Engine API, which delegates work to the *internal* Quant Engine Service, but provides a somewhat different API request format and adds a few additional behaviors like request queueing and batching several "calculations" in one request.

The API request examples in this document use the *internal* API format. To interact with the public Quant Engine API, consult the specification and documentation provided on FactSet's developer portal at https://developer.factset.com/api-catalog/quant-engine-api

Overview

The Quant Engine Service allows you to retrieve data sets from FactSet's content databases. These data sets are then organized as a date, formula, identifier data cube and stored for analysis by tools such as Python Pandas, R, and others.

A data cube is created by generating a set of dates, a universe of identifiers on each date, and then evaluating FactSet formulas for each identifier on each date. There are a number of ways to define each of these inputs. The universe can be defined using a list of identifiers, a universe limiting screening code, or by a universal screen document. The formulas can be screening codes, FQL codes, or universal screen parameters. You can mix and match. For example, you can define the universe using a universal screen document, and then use that universe to fetch not only parameters from that universal screen, but also screening and FQL codes.

Note that the term data *cube* is intended as a useful mental model. It is not a true cube in the mathematical sense of the word. There are three dimensions, date, formula, and identifier, but the size of the universe dimension is not fixed. If the universe is defined by a screening formula or a universal screen, the size and composition of the universe dimension can vary from one date to another. Additionally, the type of the elements stored in the cube is different for each formula. Elements can be scalar floats, integers, and strings, or one-dimensional arrays of scalars.

Once the data cube is created, it is stored for a short amount of time, and can be queried without having to retrieve the data from scratch. The Quant Engine Service returns the data as a Data Frame serialized using the Apache Feather format. This format is natively understood by Python Pandas and R.

To generate the data cube, you send a POST request containing a JSON object that defines the input parameters. This document describes the structure of this object and the meaning of various attributes.

Overall Structure

The body of the POST request is a JSON object with a mandatory `data` attribute, and an optional `meta` attribute. The `data` attribute contains the date, universe, and formula definitions. The optional `meta` attribute contains various options that modify the overall behavior of the data generation process.

```
{
    "data" :
    {

         "dates" : { ... },
      "universe" : { ... },
      "formulas" : [ ... ]

    },

    "meta" :
    {
      ...
    }
}
```

Defining the Time Series

The top level `dates` attribute defines how the time series is generated. There are two ways to define it, as a list of dates, or as a FactSet date range. The `source` attribute specifies the method to be used.

Note that failure to generate the time series, either due to an invalid specification, or to a system problem, is a fatal error. The data set can't be created without a valid list of dates.

List of Dates

Set the `source` attribute to `dateList`, and provide a list of dates in the in the `dates` array attribute. The dates must use the ISO date format, however the time portion of the date is not used. The most typical format is *YYYYMMDD* or *YYYY-MM-DD*. Note that FactSet relative dates or date math are *NOT* supported when specifying individual dates. You must also provide a `calendar` and `frequency` attributes since these affect date feelback behavior when evaluating formulas.

```
    "dates" :
    {
           "source" : "dateList",
         "calendar" : "FIVEDAY",
        "frequency" : "D",
            "dates" : [
                "20050701",
                "2005-07-02",
                "2005-07-03",
                "20050704",
```

```
            "2005-07-05"
        ]
    }
```

FactSet Date Range

Another way to specify the time series is by using a FactSet date range. This is exactly analogous to the way you define a date range in FQL formulas and FactSet applications. Set the `source` attribute to `fdsDate`. Then specify a `startdate`, an `enddate`, a `frequency`, and a `calendar`. A list of dates will be generated from that specification. The FQL formula DATE_YYYYMMDD is used to generate the dates.

The `startdate` and `enddate` attributes may use either the ISO date format, or FactSet relative date and date math format.

```
    "dates" :
    {
          "source" : "fdsDate",
       "startdate" : "0",
         "enddate" : "-30D",
       "frequency" : "D",
        "calendar" : "FIVEDAY"
    }
```

Overriding the Universal Screen Calendar

If your universe is defined using a screen document (see below), the calendar you specified in the dates object could be different from the calendar set in your universal screen document. This may lead to inconsistencies in the data. For example, your FQL formulas might be fetched using the calendar FIVEDAY specified in the dates object, but your universal screen parameters will be fetched using the SEVENDAY calendar set in the screening document.

By default the Quant Engine Service will *override* the screen document calendar with the one you specified in the dates object. To turn off the override, you can set the optional `overrideUniversalScreenCalendar` attribute to `false`. For example

```
    "dates" :
    {
          "source" : "fdsDate",
       "startdate" : "0",
         "enddate" : "-30D",
       "frequency" : "D",
        "calendar" : "FIVEDAY"
        "overrideUniversalScreenCalendar": false
    }
```

Defining the Universe

The top level `universe` attribute defines how the universe is generated. There are three ways to specify the universe. You can use a

- List of identifiers
- Universe limiting screening code
- Universal screen document

The `source` attribute specifies the method to be used.

Recall that the universe is generated in the context of a specific date, so that an `identifierUniverse` will be the same for every date, but the set of identifiers produced by a `screeningExpressionUniverse` can vary from one date to another. Note that failure to generate a universe for *any* date is considered to be a fatal error. The data set can't be created without a valid universe on all dates.

Identifier Universe

Set the `source` attribute to `identifierUniverse` and provide a valid list of `identifiers`. You must also specify the `universeType`. Currently only two universe types are supported, `EQUITY` and `DEBT`. Identifiers can be tickers, cusips, sedols, or FactSet identifiers. The type of identifiers accepted may vary depending on your CACCESS entitlements.

```
"universe" :
{
        "source" : "identifierUniverse",
  "universeType" : "EQUITY",
    "identifiers" : [
      "AAPL",
      "MSFT",
      "IBM"
    ]
}
```

Screening Expression Universe

Set the `source` attribute to `screeningExpressionUniverse` and provide a valid `universeExpr`. This can be any logical screening code that limits the universe. You must also specify the `universeType`. Currently only two universe types are supported, `EQUITY` and `DEBT`. This type of universe definition accepts an optional `securityExpr` attribute that allows you to specify an alternative way to name securities. If your universe expression returns cusips as identifiers, you may want to choose TICKER instead as the identifier that will be used in the data set. The value of `securityExpr` is currently limited to

```
* TICKER
* CUSIP
* SEDOL
* AVAIL(FI_IDENTIFIER(CUSIP),FI_IDENTIFIER(DEFAULT_ID))
* AVAIL(FSYM_SECURITY_PERM_ID("DEFAULT"), CUSIP)'
```

The last two identifier expressions are useful for DEBT universes.

```
"universe" :
{
        "source" : "screeningExpressionUniverse",
  "universeExpr" : "ISON_DOW",
  "universeType" : "EQUITY",
  "securityExpr" : "TICKER"
}
```

Universal Screen Universe

You may use a Universal Screen document as the source of the universe. Set the `source` attribute to `universalScreenUniverse` and provide a valid `screen`. This must be a fully qualified FactSet screen document name. The screen will be calculated on each date, and the universe of identifiers will be the securities that passed the screen. When using this type of universe, all the universe properties such as the universe type, the kind of identifier, and so on, are completely determined by the Universal Screen document. You can only modify them by modifying the Universal Screen document itself.

```
"universe" :
{
  "source" : "universalScreenUniverse",
  "screen" : "Personal:/Screens/TestScreen"
}
```

Note that if the screen is not using an `EQUITY` universe, it will be interpreted as `DEBT` for the purposes of evaluating content formulas.

Defining the Formulas

The top level `formulas` attribute of the request specifies an array of formulas to evaluate. There are three ways to define a formula. You can use a screening expression, an FQL expression, or a Universal Screen parameter reference. Note that unlike date and universe generation errors, failure to evaluate an individual formula is not a fatal error. Such formulas will be represented by an array of float NaN values in the data set. String missing values will be represented by a None (Null) value. The error status of each formula is recored in the data set.

Screening Expressions

Set the `source` attribute to `screeningExpression` and provide a valid screening `expr`. This can be any valid screening code, including codes that return arrays (see the Array Valued Formulas section for details). You must also specify the `name` attribute. Screening codes can be long and complex, and the `name` is used as an alias when displaying the data set. Note that currently all `name` values must be unique due to the limitations of the data serialization format we use. This restriction will be removed in the future. If your names are not unique, they will be deduplicated by adding a . suffix.

```
"formulas" :
[
  {
    "source" : "screeningExpression",
      "expr" : "P_PRICE",
      "name" : "Price"
  },
  {
    "source" : "screeningExpression",
      "expr" : "FF_DIV_YLD",
      "name" : "Dividend Yield"
  },
  {
    "source" : "screeningExpression",
      "expr" : "FG_GICS_SECTOR",
      "name" : "Sector"
  }
]
```

FQL Formulas

Set the `source` attribute to `fqlExpression` and provide a valid FQL `expr`. You must also specify the `name` attribute. Actual FQL codes can be long and complex, and the `name` is used as an alias when displaying the data set. Note that currently all `name` values must be unique due to the limitations of the data serialization format we use. This restriction will be removed in the future. If your names are not unique, they will be deduplicated by adding a . suffix.

FactSet FQL formulas vary widely in the types of arguments they take and the kinds of values they produce. The Quant Engine Service only works with FQL formulas that are *compatible* with the data "cube" model. In short, they must take a date argument, a universe of identifiers, and produce a value for each (date, identifier) combination (see the Notes on FQL Formulas section for more details). This value *must* be a scalar or an 1-dimensional array.

The identifiers are provided to the FQL formula automatically based on the universe you defined. You have to make sure the FQL expression works for those identifiers.

However, the presence and position of the date arguments can't be deduced automatically since it depends on the particular FQL formula. You have to provide a hint by inserting placeholders into the formula text indicating the position of the date arguments. The placeholders are `#DATE`, which indicates the postion of the start and end arguments, and `#FREQ` which indicates the position of the frequency argument. In the example below the `OS_TOP_HLDR_POS` code demonstrates the use of these placeholders. When fetching data, these placeholders are replaced with the appropriate values based on the time series definition.

Note that for some formulas the date argument is not required or is implicit, such as the FG_GICS_SECTOR code below. Such formulas are allowed, but make sure to check they produce "data cube" compatible values.

Be careful with the `#FREQ` argument. Some formulas don't work with all possible frequencies, check each formula's documentation.

```
    "formulas": [
      {
        "source": "fqlExpression",
        "expr": "OS_TOP_HLDR_POS(ALL,#DATE,#DATE,#FREQ,,S,SEC)",
        "name": "Top Holder Positions"
      },
      {
        "source": "fqlExpression",
        "expr": "FG_GICS_SECTOR",
        "name": "Sector"
      },
    ]
```

FQL Formulas and the NOW argument

Some FQL formulas like `P_PRICE` accept the `NOW` argument. The Quant Engine Service is designed to work with historical data only, and ***any formulas that return intraday data are NOT supported***. You will not be prevented from using `NOW` in your formulas, but beware that the results are undefined. The Quant Engine Service works by chunking requests by date, and occasionally by a universe subset. Therse requests are then processed in parallel at different times, potentially separated by minutes. A formula like `P_PRICE(NOW)` is a function of fetch time, NOT the backtest date, and therefore different chunks may contain different values for `P_PRICE(NOW)`.

FQL Formulas and FQL Formula Attributes

The Quant Engine Service ***does NOT support FQL Formula attributes***. You will not be prevented from using FQL attributes, but beware that the results are undefined. Some attributes might work, but others will result in errors.

FQL Formula Return Type Hint

The Quant Engine Service attempts to automatically determine if an FQL formula returns a scalar or a 1-dimensional array for a given identifier and date. However, due to the diversity of FQL formulas in FactSet, sometimes the heuristic fails, and the Quant Engine Service guesses incorrectly, causing an error. This occurs most commonly with FQL formulas that return arrays. Therefore it is good practice to specify a return type hint for such formulas. Set the `isArrayReturnType` attribute of such FQL formulas to `true`, and Quant Engine will treat the return as an array. Note that it is NOT correct to set this hint to `true` for formulas that do not actually produce arrays. That will result in an error.

In the example below, the price formula returns an array of prices. Since we know for a fact that this formula produces arrays, we can explicitly say so and avoid potential edge cases.

```
{
        "data" :
        {
          "universe" :
          {
                  "source" : "identifierUniverse",
            "universeType" : "EQUITY",
             "identifiers" : [
                "03748R74",
                "S8112735"
              ]
          },

          "dates" :
          {
                  "source" : "dateList",
                "calendar" : "FIVEDAY",
               "frequency" : "D",
                   "dates" : [
                      "20050701"
                ]
          },

          "formulas" :
          [
            {
                          "source" : "fqlExpression",
                            "expr" : "P_PRICE(#DATE,#DATE-5D,#FREQ)",
                            "name" : "Price",
                "isArrayReturnType" : true
            }
          ]
        }
}
```

And here's the data

```
    DATE UNIVERSE                                                          Price
20050701 03748R74 [52.40118, 52.247963, 51.903217, 51.328648, 51.137115, 50.894524]
20050701 S8112735                                                          [nan]
DATE        object
UNIVERSE    object
Price       object
dtype: object
```

Screening and FQL Formulas with Date Offsets

Recall that each formula is fetched in the context of a particular backtest date. The optional `dateOffset` attribute can be used to modify the backtest date for that particular formula. The value of this argument must be a FactSet relative date modifier such as `+5M` or `-1AY`. When a formula containing a dateOffset is evaluated:

- The universe comes from the current backtest date
- The data is stored under the current backtest date
- But the fetch date is the current backtest date modified by the date offset

Using the screening formula P_PRICE, and assuming a current backtest date of 20050701

```
      "formulas" :
      [
        {
              "source" : "screeningExpression",
                "expr" : "P_PRICE",
                "name" : "Price"
        },
        {
              "source" : "screeningExpression",
                "expr" : "P_PRICE",
                "name" : "Price +1AY",
          "dateOffset" : "+1AY"
        }
      ]
    }
```

The data will look like

```
        DATE UNIVERSE  Price 0  Price +1AY
20050701      IBM    74.67       76.82
20050701       GE   277.92      263.68
20050701        C   461.60      482.50
DATE           object
UNIVERSE       object
Price 0       float64
Price +1AY    float64
dtype: object
```

And the info section of the cube will include information about the offset date.

Note that date offsets only work for Screening and FQL expressions, universal screen documents do not support date offsets.

```
                    UUID    expr  code desc     wall  type       name offset offset_date              source     DATE
o49ncU6oTU645OYkOnj7vA== P_PRICE     0      1079.130 FLOAT    Price 0      0             screeningExpression 20050701
uuKAgRMqSSmg2zdPNHEXwQ== P_PRICE     0        29.791 FLOAT Price +1AY   +1AY    20060630 screeningExpression 20050701
```

Universal Screen Parameters

Set the `source` attribute to `universalScreenParameter` and provide a valid parameter `referenceName`. Each parameter in a Universal Screen document has an automatically assigned reference name such as P4, or a user assigned reference name. You must also specify the `name` attribute. Note that currently all `name` values must be unique due to the limitations of the data serialization format we use. This restriction will be removed in the future. If your names are not unique, they will be deduplicated by adding a . suffix.

Note that to use Universal Screen parameters as data set formulas, the universe must be defined by a Universal Screen. Parameter references only make sense in the context of a particular Universal Screen document.

Also note that references to non-existent parameters will result in an array of NaNs in the generated data set.

```
    "formulas" :
    [
      {
              "source" : "universalScreenParameter",
        "referenceName" : "P4",
                "name" : "Price"
      },
      {
              "source" : "universalScreenParameter",
        "referenceName" : "MY_DIV_YIELD",
                "name" : "Dividend Yield"
      },
      {
              "source" : "universalScreenParameter",
        "referenceName" : "I_AM_AN_INVALID_REFERENCE",
```

```
                "name" : "No Such Parameter"
      }
```

A common pattern is to request *all* parameters from a Universal Screen. Instead of listing them one by one, there is a shortcut. Set the `source` attribute to `allUniversalScreenParameters`, and *all* the parameters will be included as formulas. These formulas will take their `name` attribute from the 'Header' property of the parameter. Since all `name` values must be unique, make sure that your screen parameter 'Header' properties are all unique as well. This limitation will be removed in the future. If the names are not unique, they will be deduplicated by adding a . suffix.

```
    "formulas" :
    [
      {
        "source" : "allUniversalScreenParameters"
      }
    ]
```

Let's Generate a Data Set

Let's use the data, universe, and formula definitions to generate some data sets. Below are four examples of requests, the data sets they generate, and some comments about typical usage patterns.

Basic Data Set

- The Time Series is a simple list of dates (remember to use the YYYYMMDD or YYYY-MM-DD format)
- The Universe is a fixed set of equity identifiers
- We retrieve the sector and earnings per share using simple screening codes

```
{
  "data" :
  {
        "dates":
        {
               "source" : "dateList",
             "calendar" : "FIVEDAY",
             "frequency" : "M",
                "dates" : [
                    "20050701",
                    "20070701",
                    "20120701"
              ]
        },

        "universe":
        {
                 "source" : "identifierUniverse",
          "universeType" : "EQUITY",
            "identifiers" : [
              "AAPL",
              "MS",
              "GE"
            ]
        },

        "formulas":
        [
          {
            "source" : "screeningExpression",
              "expr" : "FG_GICS_SECTOR",
              "name" : "Sector (SCR)"
          },
          {
            "source" : "screeningExpression",
              "expr" : "FF_EPS",
              "name" : "Eps (SCR)"
          }
        ]
    }
}
```

We request the Quant Engine Service to return the data set as a Pandas Data Frame, and print it.

```
       DATE UNIVERSE            Sector (SCR)  Eps (SCR)
0  20050701     AAPL  Information Technology     0.0127
1  20050701       MS              Financials     4.0590
2  20050701       GE             Industrials     1.6100
3  20070701     AAPL  Information Technology     0.0811
4  20070701       MS              Financials     7.0658
5  20070701       GE             Industrials     2.0215
6  20120701     AAPL  Information Technology     0.9886
7  20120701       MS              Financials     1.2338
8  20120701       GE             Industrials     1.2313
DATE             object
UNIVERSE         object
Sector (SCR)     object
Eps (SCR)       float64
dtype: object
```

Switch to a FactSet Date Range and add an FQL formula

- The time series is now defined by a date range
- We're pulling in Sales information using FF_SALES as an FQL expression
- Note the use of #DATE and #FREQ placeholders to identify the date argument positions
- We could have evaluated FF_SALES as a screening expression instead
- You should use screening codes to get data whenever possible because they are faster
- Only use FQL expressions to get data not available via screening expressions

```
{
  "data" :
  {
        "dates":
        {
               "source" : "fdsDate",
            "calendar" : "FIVEDAY",
           "startdate" : "0",
             "enddate" : "-5M",
           "frequency" : "M"
        },

        "universe":
        {
                 "source" : "identifierUniverse",
          "universeType" : "EQUITY",
            "identifiers" : [
              "AAPL",
              "MS",
              "GE"
            ]
        },

        "formulas":
        [
          {
            "source" : "screeningExpression",
              "expr" : "FG_GICS_SECTOR",
              "name" : "Sector (SCR)"
          },
          {
            "source" : "screeningExpression",
              "expr" : "FF_EPS",
              "name" : "Eps (SCR)"
          },
          {
            "source" : "fqlExpression",
              "expr" : "FF_SALES(MON,#DATE,#DATE,#FREQ)",
              "name" : "Sales (FQL)"
          }
        ]
    }
}
```

We request the Quant Engine Service to return the data set as a Pandas Data Frame, and print it.

```
    DATE UNIVERSE           Sector (SCR)  Eps (SCR)  Sales (FQL)
20210930     AAPL Information Technology     5.6140     365817.0
20210930       MS             Financials     6.4655      60391.0
20210930       GE            Industrials     4.6676      75369.0
20211029     AAPL Information Technology     5.6140     365817.0
20211029       MS             Financials     6.4655      60391.0
20211029       GE            Industrials     4.6676      75369.0
20211130     AAPL Information Technology     5.6140     365817.0
20211130       MS             Financials     6.4655      60391.0
20211130       GE            Industrials     4.6676      75369.0
20211231     AAPL Information Technology     5.6140     378697.0
20211231       MS             Financials     8.0298      61352.0
20211231       GE            Industrials    -2.6512      74174.0
20220131     AAPL Information Technology     5.6140     378697.0
20220131       MS             Financials     8.0298      61352.0
20220131       GE            Industrials    -2.6512      74174.0
20220228     AAPL Information Technology     5.6140     378697.0
20220228       MS             Financials     8.0298      61352.0
20220228       GE            Industrials    -2.6512      74174.0
DATE             object
UNIVERSE         object
Sector (SCR)     object
Eps (SCR)       float64
Sales (FQL)     float64
dtype: object
```

Change the Universe to a Screening Expression and Add an Array Valued FQL formula

- Change the universe to DOW companies with Price > 200, we need to use a screening expression universe
- Add a P_PRICE screening formula as a sanity check
- Add the OS_TOP_HLDR_POS formula, note that this formula returns an *array of values* for each (date, identifier) combination
- Note that **MMM** only appears in the **20210531** universe

```
{
  "data" :
  {
        "dates":
        {
               "source" : "fdsDate",
             "calendar" : "FIVEDAY",
            "startdate" : "0",
              "enddate" : "-5M",
            "frequency" : "M"
        },

        "universe" :
        {
                "source" : "screeningExpressionUniverse",
          "universeExpr" : "(ISON_DOW AND P_PRICE > 200)=1",
          "universeType" : "EQUITY",
          "securityExpr" : "TICKER"
        },

        "formulas":
        [
          {
            "source" : "screeningExpression",
              "expr" : "FG_GICS_SECTOR",
              "name" : "Sector (SCR)"
          },
          {
            "source" : "screeningExpression",
              "expr" : "P_PRICE",
              "name" : "Price (SCR)"
          },
          {
            "source" : "fqlExpression",
              "expr" : "FF_SALES(MON,#DATE,#DATE,#FREQ)",
              "name" : "Sales (FQL)"
          },
          {
            "source": "fqlExpression",
            "expr": "OS_TOP_HLDR_POS(3,#DATE,#DATE,#FREQ,,S,SEC)",
            "name": "Top 3 Holder Pos",
            "isArrayReturnType" : true
          }
        ]
    }
}
```

We request the Quant Engine Service to return the data set as a Pandas Data Frame, and print it.

DATE	UNIVERSE	Sector (SCR)	Price (SCR)	Sales (FQL)	Top 3 Holder Pos
20210930	MSFT	Information Technology	281.92	176251.0	[102992934.0, 2908770.0, 1669375.0]
20210930	UNH	Health Care	390.74	279321.0	[1834462.0, 1415996.0, 968115.0]
20210930	GS	Financials	378.03	75544.0	[8610503.0, 974404.0, 132137.0]
20210930	BA	Industrials	219.94	62798.0	[141719.0, 85309.0, 72458.0]
20210930	MCD	Consumer Discretionary	241.11	22527.6	[88500.0, 43154.0, 21581.0]
20210930	HD	Consumer Discretionary	328.26	144415.0	[186544.0, 79415.0, 69996.0]
20210930	V	Information Technology	222.75	24105.0	[1610000.0, 250856.0, 139715.0]
20211029	MSFT	Information Technology	331.62	176251.0	[102992934.0, 2908770.0, 1669375.0]
20211029	UNH	Health Care	460.47	279321.0	[1834462.0, 1415996.0, 968115.0]
20211029	GS	Financials	413.35	75544.0	[8610503.0, 974404.0, 132137.0]
20211029	BA	Industrials	207.03	62798.0	[141719.0, 85309.0, 72458.0]
20211029	CAT	Industrials	204.01	48437.0	[288000.0, 276550.0, 275008.0]
20211029	MCD	Consumer Discretionary	245.55	22527.6	[88500.0, 43154.0, 21581.0]
20211029	HD	Consumer Discretionary	371.74	147699.0	[186544.0, 79415.0, 69996.0]
20211029	V	Information Technology	211.77	24105.0	[1610000.0, 250856.0, 139715.0]
20211130	MSFT	Information Technology	330.59	176251.0	[102992934.0, 2908770.0, 894691.0]
20211130	UNH	Health Care	444.22	279321.0	[1709813.0, 1411127.0, 968115.0]
20211130	GS	Financials	380.99	75544.0	[8610503.0, 957404.0, 132137.0]
20211130	MCD	Consumer Discretionary	244.60	22527.6	[88500.0, 43154.0, 21581.0]
20211130	HD	Consumer Discretionary	400.61	147699.0	[186544.0, 79415.0, 69996.0]
20211231	MSFT	Information Technology	336.32	184903.0	[102992934.0, 2800000.0, 894691.0]
20211231	UNH	Health Care	502.14	287597.0	[1526450.0, 1406707.0, 968115.0]
20211231	GS	Financials	382.55	64321.0	[8610503.0, 957404.0, 132137.0]
20211231	BA	Industrials	201.32	62286.0	[141719.0, 85309.0, 70957.0]
20211231	CAT	Industrials	206.74	50984.0	[288000.0, 276550.0, 275008.0]
20211231	MCD	Consumer Discretionary	268.07	23222.9	[88600.0, 43154.0, 21581.0]
20211231	HD	Consumer Discretionary	415.01	147699.0	[186544.0, 79415.0, 69996.0]
20211231	V	Information Technology	216.71	25477.0	[2400000.0, 1758783.0, 272074.0]
20220131	MSFT	Information Technology	310.98	184903.0	[102992934.0, 2800000.0, 894691.0]
20220131	UNH	Health Care	472.57	287597.0	[1457424.0, 1397810.0, 968115.0]
20220131	GS	Financials	354.68	64321.0	[8610503.0, 957404.0, 132137.0]
20220131	BA	Industrials	200.24	62286.0	[141719.0, 85309.0, 70957.0]
20220131	CAT	Industrials	201.56	50984.0	[288000.0, 276550.0, 275008.0]
20220131	MCD	Consumer Discretionary	259.45	23222.9	[88600.0, 43154.0, 21581.0]
20220131	HD	Consumer Discretionary	366.98	151157.0	[186544.0, 79415.0, 69996.0]
20220131	V	Information Technology	226.17	25477.0	[2400000.0, 1758783.0, 264965.0]
20220228	MSFT	Information Technology	298.79	184903.0	[102992934.0, 2800000.0, 894691.0]
20220228	UNH	Health Care	475.87	287597.0	[1455304.0, 1382810.0, 968115.0]
20220228	GS	Financials	341.29	64321.0	[952404.0, 123205.0, 117701.0]
20220228	BA	Industrials	205.34	62286.0	[132118.0, 93980.0, 85309.0]
20220228	MCD	Consumer Discretionary	244.77	23222.9	[88600.0, 43154.0, 25829.0]
20220228	HD	Consumer Discretionary	315.83	151157.0	[204092.0, 86056.0, 76634.0]

```
20220228        V Information Technology       216.12      25477.0    [2400000.0, 1758783.0, 264965.0]
DATE                 object
UNIVERSE             object
Sector (SCR)         object
Price (SCR)         float64
Sales (FQL)         float64
Top 3 Holder Pos     object
dtype: object
```

And Now for Something Completely Different, Universal Screen Universe

- If you have a Universal Screen document you can use it to define the Universe
- You can pull out individual parameters using `referenceName`
- Remember that you can pull out *all* parameters using the `allUniversalScreenParameters` formula `source` (not shown here)
- If you reference a non-existent parameter, you will get an array of NaNs
- You can use the universe generated by the Universal Screen in regular screening or FQL expressions
- Let's get P_PRICE three times, from the Universal Screen, from an FQL expression, and from a screening expression
- Hopefully the numbers match (barring different rounding strategies)
- Note that Univeral Screens are *much* slower than getting universe or formula data using codes, avoid for large data sets!

```
{
  "data" :
  {
        "universe" :
        {
          "source" : "universalScreenUniverse",
          "screen" : "Personal:/Screens/TestScreen"
        },

        "dates" :
        {
               "source" : "fdsDate",
            "startdate" : "20050701",
              "enddate" : "20050701",
            "frequency" : "D",
             "calendar" : "FIVEDAY"
        },

        "formulas" :
        [
          {
                    "source" : "universalScreenParameter",
            "referenceName" : "P9",
                     "name" : "Price (USC)"
          },
          {
              "source" : "screeningExpression",
                "expr" : "P_PRICE",
                "name" : "Price (SCR)"
          },
          {
              "source" : "fqlExpression",
                "expr" : "P_PRICE(#DATE, #DATE, #FREQ)",
                "name" : "Price (FQL)"
          },
          {
                    "source" : "universalScreenParameter",
            "referenceName" : "AHA",
                     "name" : "Div Yield (USC)"
          },
          {
                    "source" : "universalScreenParameter",
            "referenceName" : "PARAM999",
                     "name" : "Bad Param"
          }
        ]
    }
}
```

We request the Quant Engine Service to return the data set as a Pandas Data Frame, and print it. When using Universal Screen documents, the columns are not necessarily in the same order as the formulas in the request. This will be fixed in future releases. Note that Pandas makes it easy to sort data frames, so it's easy to remedy.

```
         DATE UNIVERSE  Price (USC)  Div Yield (USC)  Bad Param  Price (SCR)  Price (FQL)
0    20050701      RTX        51.38         1.354620        NaN        51.38     51.38000
1    20050701      UNH        52.81         0.034079        NaN        52.81     52.81000
2    20050701      JNJ        64.95         1.726580        NaN        64.95     64.95000
3    20050701      MMM        73.00         1.754600        NaN        73.00     73.00000
4    20050701      CVX        56.97         2.913730        NaN        56.97     56.97000
5    20050701      IBM        74.67         0.710083        NaN        74.67     74.67000
6    20050701       PG        52.90         1.952610        NaN        52.90     52.90000
7    20050701       GS       102.71         0.953834        NaN       102.71    102.71000
8    20050701       BA        64.68         1.487350        NaN        64.68     64.68000
9    20050701      XOM        58.31         2.067890        NaN        58.31     58.31000
10   20050701      AXP        53.54         0.780557        NaN        53.54     53.54000
11   20050701       DD       134.07         2.706520        NaN       134.07    134.06999
DATE                object
UNIVERSE            object
Price (USC)        float64
```

```
Div Yield (USC)    float64
Bad Param          float64
Price (SCR)        float64
Price (FQL)        float64
dtype: object
```

Limits and Notes on Performance

The Quant Engine Service itself does not impose any limits on the number of dates, size of the universe, or number of formulas. Theoretically, you can construct very large data sets. However, FactSet's underlying screening, FQL, and universal screen infrastructures *do* have limits of their own. For example, screening expressions are optimized for very large universes, whereas FQL is not. The number of formulas that can be *reliably* processed in a single request varies depending on the particular content service and the complexity of the formulas. In practice, your request may fail because the data takes too long to generate and hits limits within these services.

The Quant Engine Service attempts to get around the performance limitations of various services, FQL in particular, by breaking up large requests into smaller ones, sending them out individually, and then reassembling the data. However, it is possible to construct very inefficient formulas that defeat this optimization. In such cases it is necessary to modify the formulas themselves to be more efficient.

To maximize chances of success try to limit the size of the input dimensions to

- 3000 identifer universe
- 2500 dates
- 50 formulas

These are not strict limits, you can trade-off the size of one dimension for another. For example, you can use a 30,000 universe (or larger) if you reduce the number of dates by a factor of 10.

If the amount of data you want to retrieve is larger than the above parameters, and you can't trade-off effectively, you should break up your data fetch into several requests to the API. The simplest method is to break up by the date dimension. This is recommended if the number of formulas is low, under 50. If the number of formulas is high and their performance is poor, then it is better to break up by formula. As a last resort, it may be necessary to break up along both the date and formula dimensions. It is usually not necessary to break up by the universe dimension.

In general, be aware that

- Screening formulas are genrally the fastest
- Only use FQL expressions if the data is not available via screening expressions
- Any request that uses a Universal Screen Universe will be an ***order of magnitude slower*** than simple screening or FQL codes
- When constructing data sets with hundreds of dates, a Universal Screen based request may take ***hours*** to generate

Notes on Missing Values

FactSet content databases use FactSet specific NA sentinels that are not recognized by other programming languages and environments. The Quant Engine Service understands these sentinels and marks missing data internally. However, when outputting the data set, representation of missing values depends on the output format used.

When the data set is returned as a Pandas Data Frame in Python

- numpy.nan is used to represent numeric missing values
- Python None is used to represent string missing values

Be aware that the following values represent FactSet's missing values. If you see them in your output it is likely that the output format does not make it possible to represent missing values correctly, or there was some quirk in the data returned by content services.

```
Float: -1.0e+22
Float: -1.5e+21
Float: -9.999999778196308e+21
Integer: -2147483648
String: @NA
String: $$FDS_US_@NA$$
```

Notes on Array Valued Screening Formulas

Some FactSet formulas produce an array of scalar values for each (date, identifier) combination. Common examples are screening iteration codes such as `ARRAY5(P_PRICE)`, or FQL formulas such as `OS_TOP_HLDR_POS`. Array valued FQL formulas are always allowed. However, Screening codes that return array values are not common. To use such codes you must turn on this behavior by setting the `allowArrayData` attribute to `1` in the `meta` section of the request. If you do not, formulas such as `ARRAY5(P_PRICE)` will return the first value as a scalar.

Here is an example request.

```
{
    "data" :
    {
          "dates" :
          {
               "source" : "fdsDate",
                "startdate" : "0",
                  "enddate" : "-5D",
            "frequency" : "D",
             "calendar" : "FIVEDAY"
          },
```

```
        "universe":
        {
                 "source" : "identifierUniverse",
          "universeType" : "EQUITY",
            "identifiers" : [
              "AAPL",
              "MS",
              "GE"
            ]
        },

        "formulas" :
        [
          {
            "source" : "screeningExpression",
              "expr" : "ARRAY5(P_PRICE)",
              "name" : "Price (SCR) x5"
            }
          ]
       }
    },

    "meta" :
    {
      "allowArrayData" : 1
    }
}
```

And the data set.

```
        DATE UNIVERSE                            Price (SCR) x5
0   20210610     AAPL   [126.11, 127.13, 126.74, 125.9, 125.89]
1   20210610       MS       [91.68, 92.67, 92.79, 93.21, 93.96]
2   20210610       GE        [13.63, 13.73, 13.9, 13.91, 13.96]
3   20210609     AAPL   [127.13, 126.74, 125.9, 125.89, 123.54]
4   20210609       MS       [92.67, 92.79, 93.21, 93.96, 93.35]
5   20210609       GE        [13.73, 13.9, 13.91, 13.96, 14.09]
6   20210614     AAPL  [130.48, 127.35, 126.11, 127.13, 126.74]
7   20210614       MS       [90.72, 92.05, 91.68, 92.67, 92.79]
8   20210614       GE        [13.47, 13.69, 13.63, 13.73, 13.9]
9   20210611     AAPL   [127.35, 126.11, 127.13, 126.74, 125.9]
10  20210611       MS       [92.05, 91.68, 92.67, 92.79, 93.21]
11  20210611       GE        [13.69, 13.63, 13.73, 13.9, 13.91]
12  20210608     AAPL   [126.74, 125.9, 125.89, 123.54, 125.06]
13  20210608       MS       [92.79, 93.21, 93.96, 93.35, 92.76]
14  20210608       GE        [13.9, 13.91, 13.96, 14.09, 14.09]
15  20210607     AAPL   [125.9, 125.89, 123.54, 125.06, 124.28]
16  20210607       MS       [93.21, 93.96, 93.35, 92.76, 92.11]
17  20210607       GE       [13.91, 13.96, 14.09, 14.09, 14.15]
DATE              object
UNIVERSE          object
Price (SCR) x5    object
dtype: object
```

Output Format(s)

The primary output format supported by the Quant Engine Service data "cube" endpoint is the Apache Feather format. The data set is converted to a Data Frame, serialized using the Feather format, and returned as a binary response. Data Frames encoded in the Feather format can be read by any programming environment that supports this format such as Python Pandas, and R Data Frames.

Note that support for FactSet's STACH format has been dropped