# Moody's Analytics Data Buffet API User Guide

*version 1.13.1*

## Introduction

Data Buffet is the Moody's Analytics repository of international and subnational economic and demographic time series data. We provide several means of manual and automatic access that you can integrate with your workflow, among them, the Data Buffet API (application program interface). The API uses HMAC and oAuth 2.0 authentication and JSON responses, and is agnostic regarding the client's operating system and programming language. The API is throttled (rate-limited) to 300 requests per minute and one gigabyte of data per month. The principal tutorial in this document is expressed in cURL notation, but the appendixes contain equivalent examples in C#, Java, Python and R.

## Overview

The API's core functionality is expressed by three groups of endpoints:

- */series* – An endpoint to retrieve a single time series.
- */multiseries* - An endpoint to return an array of series data formatted in JSON.
- */baskets* – A collection of endpoints pertaining to the baskets stored by a user on Data Buffet.
  - Get a list of baskets
  - Get a single basket by ID
  - Get the contents of a given basket – The content of a basket is a list of mnemonics or other expressions, not the time series data. To obtain the time series data and the associated metadata, a basket must be executed via the /orders endpoint.
- */orders* – A collection of endpoints to create and manage orders and to retrieve the basket output associated with a completed order.
  - Place an order
  - Delete an order
  - Get a list of orders
  - Check the status of a given order
  - Return the output of a completed order.

The API also provides some helper endpoints that are used for returning enumerations. For more information, please see https://github.com/moodysanalytics/databuffet.api.codesamples.

## Is the API right for me?

The Data Buffet API is best suited to programmatically retrieve a small number of individual series stored on Data Buffet, or execute and retrieve the results of a basket saved on your Data Buffet account. For other tasks, consider these mechanisms, documented elsewhere:

- Explore the contents of the repository: catalog and search features on DataBuffet.com
- Read long-form information about individual series or datasets: Mnemonic 411 and Data Buffet News at DataBuffet.com
- Transfer a large number of series: a basket created on DataBuffet.com
- Maintain a basket by a group of users: group and sharing features of DataBuffet.com
- Deliver a basket when one or more trigger series are updated: Scheduled baskets on DataBuffet.com
- Feed time series data into an Excel workbook calculation, chart or VBA process: Power Tools

- Display a visualization (chart, map) in a Microsoft Excel document with one-touch update: Power Tools

## Authentication

Databuffet API supports 2 forms of authentication:

1. HMAC Signature
2. OAuth 2.0 Token

Both methods require Databuffet API access key and encryption key. Every request to the API must contain either an HMAC signature or OAuth Token.

### Getting API Keys

Access to the API is controlled by the combination of an access key and an encryption key. These keys are issued to a single user. To obtain your keys, go to the "My Subscriptions" section of your Economy.com account: https://www.economy.com/myeconomy/api-key-info.

**Figure 1. Example access key and encryption key**

```
DB73FDF0-043C-4018-A7EB-CFB57356BA22
7C7C2FEA-6D18-49A1-BEC9-193B67EAE87D
```

## Using HMAC Authentication

### Authenticating each request

HMAC signature is generated from your access key, encryption key, and a time stamp. You must attach a signature to every request using HAMC authorization, and you must re-create the signature with every request; you will receive an HTTP 401 Unauthorized error otherwise.

The access key, time stamp and signature need to be passed in as HTTP headers (not as part of the query string). Do not transmit the encryption key in the request since it is a secret between you and the server. Specifically, the signature is a SHA256 hash of the access key, encryption key and time stamp. The time stamp must be formatted as yyyy-MM-ddTHH:mm:ssZ using UTC. For example, "July 30, 2018 5:03:28pm EST" must be represented as 2018-07-30T21:03:28Z.

**Figure 2. Example HTTP request header**

```
AccessKeyId: DB73FDF0-043C-4018-A7EB-CFB57356BA22
TimeStamp: 2012-08-02T14:25:20Z
Signature: A7808C5A67C422054364F195B161753083179308482 32C6A08A77224F1017E83
```

**Figure 3. Example signature creation in C**

This C# function creates a signature from an access key, encryption key, and time stamp. See the appendixes for equivalent examples in Java, Python and R.

```
using System;
using System.Text;
using System.Security.Cryptography;
public static string CreateHMACSignature
(string accKey, string encKey, string timeStamp)
{
    string signature = string.Empty;
    byte[] keyBytes = Encoding.UTF8.GetBytes(encKey);
    using (HMAC hmac = new HMACSHA256(keyBytes))
```

```
    {
        byte[] bytesToHash = Encoding.UTF8.GetBytes(accKey + timeStamp);
        byte[] hashedBytes = hmac.ComputeHash(bytesToHash);
        for (int i = 0; i < hashedBytes.Length; i++)
        {
            signature += hashedBytes[i].ToString("X2");
        }
    }
    return signature;
}
```

## Using OAuth authentication.

oAuth Token can be generated by calling an API endpoint, using API access key as *client_id* and API encryption key as *client_secret* and it will remain valid for 1 hour. You must include token in header of every request using OAuth authorization.

### Obtaining OAuth Token

The *oauth2/token* endpoint is used to generate oAuth Token using your *access key as client_id, encryption key as client_secret and grant_type as client_credentials*. Following cURL request can be used to obtain an OAuth token.

**Figure 4. Request**

```
curl -X POST \
  https://api.economy.com/data/v1/oauth2/token \
  -H 'Content-Type: application/x-www-form-urlencoded' \
  -d 'client_id=DB73FDF0-043C-4018-A7EB-CFB57356BA22' \
  -d 'client_secret=47C7C2FEA-6D18-49A1-BEC9-193B67EAE87D' \
  -d 'grant_type=client_credentials'
```

The response to the above request will have a new access token.

**Figure 5. Respone**

```
{
  "token_type": "bearer",
  "access_token": "SrZ5UkbzPn432zqMLgV3Ja",
  "expires_in": 3600
}
```

A request to API will have **Authorization: Bearer *token*** as header

**Figure 6. Call to API endpoint using oAuth Token**

```
curl -X GET \
  'https://api.economy.com/data/v1/series?m=ET.IUSA' \
  -H 'Authorization: Bearer SrZ5UkbzPn432zqMLgV3Ja'
```

## Step-by-step examples

All examples in this section use the access key, time stamp, and signature generated above. Replace these three header values with your newly generated values on every request. The following tutorial uses cURL syntax. See the appendixes for equivalent code examples in C#, Java, Python and R.

### Retrieve a single series

The */series?m={m}&freq={freq}&trans={trans}* endpoint is used to download a single series specified by a mnemonic or expression. The first parameter, m, is mandatory; The optional parameters freq, trans, conv perform a frequency conversion, apply a transformation and apply conversion type, respectively. startDate and endDate parameters are also optional and should be in YYYY-MM-dd format. If omitted, the API will use default values. For valid codes, see the Enumerations appendix.

**Figure 7a. Request using HAMC Signature**

```
curl -X GET \
'https://api.economy.com/data/v1/series?m=et.iusa&freq=0&trans=0' \
-H 'accesskeyid: DB73FDF0-043C-4018-A7EB-CFB57356BA22' \
-H 'signature: A7808C5A67C422054364F195B1617530831793084823 2C6A08A77224F1017E83' \
-H 'timestamp: 2012-08-02T14:25:20Z'
```

**Figure 7b. Request using oAuth Token**

```
curl -X GET \
'https://api.economy.com/data/v1/series?m=et.iusa&freq=0&trans=0' \
-H 'Authorization: Bearer SrZ5UkbzPn432zqMLgV3Ja'
```

**Figure 8. Response**


```
{
    "data": {
    "freq": "MONTH",
    "startDate": "1939-01-31T05:00:00Z",
    "freqCode": 128,
    "start": 1069,
    "periods": 946,
    "data": [
            29923,
            30101,
            30280,
            30094,
            30300,
            30502,
            30419,
            30663,
            ...
            ]
            },
    "mnemonic": "ET.IUSA",
    "concept": "ET",
    "geoCode": "IUSA",
    "geoTitle": "United States",
    "fipCode": "00",
    "description": "Employment: Total Nonfarm, (Ths. #, SA)",
    "source": "U.S. Bureau of Labor Statistics (BLS): Current Employment Statistics…"
    "databank": "IUSA_BLS_CES.db",
    "freqCode": "128",
    "observedAttribute": "AVERAGED",
    "startDate": "1939-01-31T05:00:00Z",
    "endDate": "2017-10-31T04:00:00Z",
    "lastHistory": "N/A",
    "dateCreated": "2013-08-20T19:28:32Z",
    "dateUpdated": "2017-11-03T12:34:06Z",
    "dateExecuted": "2017-11-27T15:26:23Z",
    "error": null,
    "status": "OK"
}
```


## Retrieve multi series

The */multi-series?m={m}&freq={freq}&trans={trans}&conv={conv}&startDate={startDate}&endDate={endDate}* endpoint is used to download multi series specified by a mnemonic or expression. The first parameter, m, is mandatory. m is a semicolon seperated list of mnemonics. The optional parameters freq, trans, conv perform a frequency conversion, apply a transformation and apply conversion type, respectively. startDate and endDate parameters are also optional and should be in YYYY-MM-dd format. If omitted, the API will use default values. For valid codes, see the Enumerations appendix.

**Figure 9. Request using oAuth Token**

```
curl -X GET \
'https://api.economy.com/data/v1/multi-series?m=et.us;fet.us' \
-H 'Authorization: Bearer SrZ5UkbzPn432zqMLgV3Ja'
```

**Figure 10. Response**

For brevity, the date and value information of data is trimmed.


```
{
    "error": null,
    "data": [
        {
            "appliedFreq": 128,
            "appliedTransformation": 0,
            "databank": "IUSA_BLS_CES.db",
            "description": "Employment: Total Nonfarm, (Ths. #, SA)",
            "error": null,
            "freqCode": "MONTH",
            "mnemonic": "ET.IUSA",
            "observedAttribute": "AVERAGED",
            "source": "U.S. Bureau of Labor Statistics (BLS): Current Employment
                       Statistics (CE) [Series ID = CES0000000001]",
            "data": [
                {
                    "date": "1939-01-31",
                    "value": 29923
                },
                {
                    "date": "1939-02-28",
                    "value": 30101
                },
                {
                    "date": "1939-03-31",
                    "value": 30280
                },
                ...
                {
                    "date": "2018-11-30",
                    "value": 149951
                },
                {
                    "date": "2018-12-31",
                    "value": 150263
                }
            ]
        },
        {
            "appliedFreq": 172,
            "appliedTransformation": 0,
            "databank": "USFOR.db",
            "description": "Baseline Scenario (January 2019): Employment:
                            Total Nonagricultural, (Mil. #, SA)",
            "error": null,
            "freqCode": "QTRDEC",
            "mnemonic": "FET.IUSA",
            "observedAttribute": "AVERAGED",
```

```
            "source": "U.S. Bureau of Labor Statistics (BLS); Moody's Analytics Forecasted",
            "data": [
                {
                    "date": "1939-03-31",
                    "value": 30.1013333333333
                },
                {
                    "date": "1939-06-30",
                    "value": 30.2986666666667
                },
                {
                    "date": "1939-09-30",
                    "value": 30.7046666666667
                },
                ...
                "date": "2048-09-30",
                    "value": 176.234631805996
                },
                {
                    "date": "2048-12-31",
                    "value": 176.462734998586
                }
            ]
        }
    ]
}
```


## Executing a basket and downloading its output: Overview

Since the above endpoint *(/series)* allows for downloading only a single series at a time, we do not recommend it for pulling down large sets of data. Instead, it is more efficient to define a basket on Data Buffet, then execute it and download its output via the API. To do so, use the */baskets* and */orders* endpoints together:

1. Retrieve a list of your saved baskets using the */baskets* endpoint. Note: The baskets need to be created and managed on DataBuffet.com, as there are currently no endpoints in the API to perform these operations.
2. Locate the desired basket from the response of the previous step and make a note of its basketId.
3. Use the POST *orders?id={id}&type={type}&action={action}* endpoint to create a new basket execution order. Pass the *basketId* from Step 2 in the *{id}* parameter, set *{type}* to "baskets", and set *{action}* to "run."
4. This creates an order in the queue and returns some metadata about the order, including an orderId.
5. Since the execution of a basket takes time to complete, use this orderId value in a call to the GET *orders/{orderId}* endpoint to get the details on whether the process has completed. Whether the process has completed is indicated by the *dateFinished* response value. If null, the process is still executing; if not null, the process has completed.
6. Once the process is complete, the final step is to retrieve the output file by calling the *GET/orders?id={id}&type=baskets* endpoint. Like the POST request that executed the basket in Step 3, set the *{id}* parameter to the *basketId*.

**Pay attention to the distinction between the permanent ID of the basket that is being executed *(basketId)* and the *transient ID* assigned to the order that is performing this task *(orderId)*. Use the latter only when checking if an order has completed.**

## Retrieve a list of your saved baskets

**Figure 11. Request using oAuth Token**

```
curl -X GET \
   https://api.economy.com/data/v1/baskets/ \
   -H 'Authorization: Bearer SrZ5UkbzPn432zqMLgV3Ja'
```

**Figure 12. Response**

For brevity, only two baskets with a trimmed list of attributes are shown.

```
[
    {
        "basketId": "5DA8BDFD-8E8F-46F6-AC50-64C1814542EE",
        "name": "BuffetBasket",
        "dateCreated": "2017-09-20T13:03:42.77Z",
        "dateExecuted": "2017-09-23T00:06:53.273Z",
        "dateUpdated": "2017-09-20T13:04:02.957Z",
        ...
    },
    {
        "basketId": "BCBAE6AE-05DC-4EAF-BF58-06EC9E241792",
        "name": "Basket 2017-05-16 07h 57m",
        "dateCreated": "2017-05-16T11:57:43.583Z",
        "dateExecuted": "2017-09-12T18:31:53.14Z",
        "dateUpdated": "2017-05-16T11:57:46.48Z",
        ...
    },
    ...
]
```

## Execute a basket

We choose to execute the first basket returned from the prior request *(basketId 5DA8BDFD-8E8F-46F6-AC50-64C1814542EE)*.

**Figure 13. Request using oAuth Token**

```
curl -X POST \
  'https://api.economy.com/data/v1/orders?id=5DA8BDFD-8E8F-46F6-AC50-64C1814542EE&type=baskets&action=run' \
  -H 'Authorization: Bearer SrZ5UkbzPn432zqMLgV3Ja'
```

**Figure 14. Response**

```
{
  "orderId": "DF0AB8F3-19D3-4D7D-9AE5-CB410FC6B6E7",
  "dateOrdered": null,
  "dateStarted": null,
  "dateFinished": null,
  "failedAttempts": 0,
  "processing": false,
  "queueStatus": 0,
  "basketId": null,
  "orderType": 0,
  "enteredQueue": "0001-01-01T00:00:00",
  "updatedQueue": null
}
```

### Check the status of your order

Since the previous request returns an *orderId(DF0AB8F3-19D3-4D7D-9AE5-CB410FC6B6E7)*, we use that value to check if our basket execution order has finished running.

**Figure 15. Request using oAuth Token**

```
curl -X GET \
   'https://api.economy.com/data/v1/orders/DF0AB8F3-19D3-4D7D-9AE5-CB410FC6B6E7' \
   -H 'Authorization: Bearer SrZ5UkbzPn432zqMLgV3Ja'
```

**Figure 16. Response**

```
{
  "orderId": "DF0AB8F3-19D3-4D7D-9AE5-CB410FC6B6E7",
  "dateOrdered": "2017-11-27T19:36:39.927Z",
  "dateStarted": "2017-11-27T19:36:40.1Z",
  "dateFinished": "2017-11-27T19:36:40.513Z",
  "failedAttempts": 0,
  "processing": false,
  "queueStatus": 2,
  "basketId": "5DA8BDFD-8E8F-46F6-AC50-64C1814542EE",
  "orderType": 1,
  "enteredQueue": "2017-11-27T14:36:39.927",
  "updatedQueue": "2017-11-27T14:36:39.927"
}
```


If the order is not finished running, *dateFinished* is set to *null*. In that case, re-issue the request in a loop after introducing a pause of a second or more until *dateFinished* is *not* null or until a certain time-out period is hit to prevent an infinite loop. (You cannot check more often than once per second, because that is the API's throttle rate.)

## Download the output file

After making sure that your order is completed, it is now time to download the output file associated with the execution of the basket. The output file type is set as part of the basket's configuration and cannot be altered via the API. Note that the *id* used in this request is the *basketId*.

**Figure 17. Request using oAuth Token**

```
curl -X GET \
   'https://api.economy.com/data/v1/orders?type=baskets&id=5DA8BDFD-8E8F-46F6-AC50-64C1814542EE'\
   -H 'Authorization: Bearer SrZ5UkbzPn432zqMLgV3Ja'
```

**Response**

The response of this request is the binary of the output file associated with the basket. You will need to write this binary stream to a file using the same file name and extension as specified in your basket options. See the Appendix for samples in C#, Java, Python, and R.

## Frequently asked questions

- What programming languages does the API support?
- What response types are supported?
- Can I use the API from Linux?
- What kind of authentication does the API use?
- How often do I need to regenerate the signature?
- How often do I need to regenerate the token?
- Is the API throttled?
- What's the fastest way to retrieve a large number of series?
- Can I use the API to populate a data warehouse?
- What kind of Data Buffet objects can I retrieve?
- Which series can I retrieve?
- Can I create or alter a basket?
- If I alter the name of a basket on DataBuffet.com, do I need to change my code?
- Whom do I contact for assistance in using the API?
- Do other Moody's Analytics products have APIs?
- I don't understand this jargon—can you translate?

**What programming languages does the API support?**

The programming language used at your end is immaterial, so long as it (a) creates HTTP requests that the API can process, and (b) can interpret the JSON-formatted responses produced by the API. The examples in this document use cURL, C#, Java, Python and R.

**What response types are supported?**

JSON is the only response type returned by the API.

**Can I use the API from Linux?**

Yes, because the operating system is immaterial. Java, Python and R are commonly used on Linux machines; to run C#, you will need to install the .NET Core framework. Setting up your run-time environment is beyond the scope of this document.

**What kind of authentication does the API use?**

Our API uses HMAC and OAuth 2.0 authentication. See the Authentication section above for more info.

**How often do I need to regenerate the signature?**

You must re-create the signature prior to every request; otherwise you will receive the "HTTP 401 Unauthorized" error. You may find it useful to create a wrapper function that takes the time stamp, access key and encryption key as arguments, and generates a signature immediately before calling the endpoint.

**How often do I need to regenerate the token?**

Once generated the oAuth token is valid for 1 hour and can be used for multiple requests.

**Is the API throttled?**

Yes, in two ways. First, you can execute 300 requests per minute per account (but a single request can retrieve one series or a basket containing thousands of series). You will receive "HTTP 429 Too Many Requests" error. Second, you can retrieve only one gigabyte of data per month. This includes all of the metadata and HTTP headers, although these are insignificant relative to the data payload. The number of requests and series are not specifically limited.

**What's the fastest way to retrieve a large number of series?**

Because the API is throttled, do not retrieve the series individually; instead, execute a basket that contains all of the series.

**Can I use the API to populate a data warehouse?**

Yes. You may create a data warehouse for internal use, but the number of users who may have access to it is stipulated by your contract; please contact your Moody's Analytics sales representative if you have questions. To initially populate the warehouse, and to regularly or promptly update each time series with new periods, Data Buffet's "scheduled basket" mechanism is more efficient than the API. (See Is the API right for me? above.)

**What kind of Data Buffet objects can I retrieve?**

The API can return a single data series, a list of your saved baskets, the properties of a particular basket, or the output file of an execution of a basket. It also returns common enumerations such as frequency and file types.

**Which series can I retrieve?**

Your access via API is identical to that via Data Buffet and Power Tools; this includes E-model simulation aliases, our historical, estimated and forecast products, and your custom scenarios. Essentially any expression that a basket can run, you can submit via the API.

**Can I create or alter a basket?**

No. The API executes baskets you have created manually through DataBuffet.com; to alter their contents, go to DataBuffet.com.

**If I alter the name of a basket on DataBuffet.com, do I need to change my code?**

No. The /baskets/{id} endpoint identifies a basket by an immutable alphanumeric GUID that is assigned by our system, not the human-readable title assigned by you.

**Whom do I contact for assistance in using the API?**

Please go to the Economy.com Contact Us page for email, chat, and telephone options. If using the email form, set Topic to "Technical Issue."

**Do other Moody's Analytics products have APIs?**

Yes. We provide APIs for our AutoCycle and Précis products.

**I don't understand this jargon—can you translate?**

Please see if the glossary in this document helps. It lists terminology pertaining to web APIs, Data Buffet, and related Moody's Analytics products.

## Appendix 1: API endpoints

All API endpoints below are relative to the root URL https://api.economy.com/data/v1/.

| HTTP | Endpoint | Description |
|---|---|---|
| **Series** | | |
| GET | series?m={m}&freq={freq}&trans={trans}&conv={conv}&startDate={startDate}&endDate={endDate} | Return a series (metadata and numeric observations) formatted in JSON. |
| **Multi Series** | | |
| GET | multi-series?m={m}&freq={freq}&trans={trans}&conv={conv}&startDate={startDate}&endDate={endDate} | Returns an array of series data formatted in JSON. |
| **Baskets** | | |
| GET | baskets?filetype={filetype}&page={page}&size={size} | Return a JSON list describing baskets available for execution. Optionally paginated. |
| GET | baskets/output-file?id={id} | Returns the streamed file content that was generated the last time a basket was executed. |
| GET | baskets/{id} | Return a single basket. |
| GET | baskets/{id}/contents?page={page}&size={size} | Retrieve the contents of a basket. |
| **Frequency** | | |
| GET | frequencies | Return a list of frequency codes |
| **Orders** | | |
| GET | orders | Return a list of your orders. |
| GET | orders/{orderId} | Return the status of an execution order. |
| POST | orders?id={id}&type={type}&action={action} | Place an order to generate a data file; return a JSON object with the order ID. |
| GET | orders?id={id}&type={type} | Return the streamed file content generated the last time a basket was executed. |

| HTTP | Endpoint | Description |
|---|---|---|
| DELETE | orders/{orderId} | Delete an order from the queue. |
| **File types** | | |
| GET | filetypes?type={type} | Return the file types available for a given object type. |

## Appendix 2: Error messages

The error codes returned by the Data Buffet API are adaptations of standard HTTP server response codes.

| Error code | Diagnosis |
|---|---|
| 401 Unauthorized | The authenticating HMAC signature is outdated. You must generate a new signature with a fresh time stamp (see Authentication section). |
| 429 Too Many Requests | You have exceeded the one request per second rate limit. Throttling is access key-specific. |
| 500 Internal Server Error | Server error. |

## Appendix 3: Enumerations

**ConversionType**

Data Buffet provides for the conversion of a time series from its native frequency to a different output frequency (either higher or lower), but the appropriate mathematical process depends on the nature of the series. There are three options, of which Cubic is the default.

| Value | Name |
|---|---|
| 0 | Constant |
| 1 | Linear |
| 2 | Cubic (DEFAULT) |
| 3 | Discrete |

**DateFormat**

| Name | Value | Description |
|---|---|---|
| Default | 0 | |
| General | 1 | 4/26/99 |
| GeneralPadded | 2 | 04/26/99 |
| GeneralPaddedFullYear | 3 | 04/26/1999 |
| AbrevMonth | 4 | Apr-99 |
| AbrevMonthFullYear | 13 | Apr-1999 |
| NoYear | 5 | 26-Apr |
| LittleEndian | 6 | 26-Apr-99 |
| Year | 7 | 1999 |
| ISO8610 | 10 | 1999-04-26 |
| ShortYear | 11 | 99 |

| Name | Value | Description |
|---|---|---|
| YearMonth | 12 | 1999M1 |
| Quater | 14 | 99Q1 |

#### DateOption

| Value | Name |
|---|---|
| 0 | StartAndEnd |
| 1 | Start |
| 2 | EntireSeries |
| 3 | Period |

#### Filetype

Available file types can be retrieved by using the */filetypes?type=baskets* endpoint.

#### Frequencies

In each response with a field that is a numeric frequency code, there will be a paired field with a human-readable string.

| Value | Name |
|---|---|
| 0 | Default |
| 16 | INDEX |
| 49 | Daily |
| 50 | Business daily (Mon- Fri) |
| 65 | Weekly ending on Sunday |
| 66 | Weekly ending on Monday |
| 67 | Weekly ending on Tuesday |
| 68 | Weekly ending on Wednesday |
| 69 | Weekly ending on Thursday |
| 70 | Weekly ending on Friday |
| 71 | Weekly ending on Saturday |
| 80 | 3 Times a month 10th, 20th and end of month |
| 97 | Bi-Weekly, ending on alternating Sunday starting January, 27th 1850.<br>E.g. Jan 27 1850, Feb 10 1850, ... ,Dec 31 2017, Jan 14 2018, Jan 28 2018 |
| 98 | Bi-Weekly, ending on alternating Monday starting January, 14th 1850.<br>E.g. Jan 14 1850, Jan 28 1850, ... ,Dec 18 2017, Jan 1 2018, Jan 15 2018 |
| 99 | Bi-Weekly, ending on alternating Tuesday starting January, 15th 1850.<br>E.g. Jan 15 1850, Jan 29 1850, ... ,Dec 19 2017, Jan 2 2018, Jan 16 2018 |
| 100 | Bi-Weekly, ending on alternating Wednesday starting January, 16th 1850.<br>E.g. Jan 16 1850, Jan 30 1850, ... ,Dec 20 2017, Jan 3 2018, Jan 17 2018 |
| 101 | Bi-Weekly, ending on alternating Thursday starting January, 17th 1850.<br>E.g. Jan 17 1850, Jan 31 1850, ... ,Dec 21 2017, Jan 4 2018, Jan 18 2018 |
| 102 | Bi-Weekly, ending on alternating Friday starting January, 18th 1850.<br>E.g. Jan 18 1850, Feb 1 1850, ... ,Dec 22 2017, Jan 5 2018, Jan 19 2018 |

| Value | Name |
|---|---|
| 103 | Bi-Weekly, ending on alternating Saturday starting January, 19th 1850.<br>E.g. Jan 19 1850, Feb 2 1850, ... ,Dec 23 2017, Jan 6 2018, Jan 20 2018 |
| 104 | Bi-Weekly, ending on alternating Sunday starting January, 20th 1850.<br>E.g. Jan 20 1850, Feb 3 1850, ... ,Dec 24 2017, Jan 7 2018, Jan 21 2018 |
| 105 | Bi-Weekly, ending on alternating Monday starting January, 21st 1850.<br>E.g. Jan 21 1850, Feb 4 1850, ... ,Dec 25 2017, Jan 8 2018, Jan 22 2018 |
| 106 | Bi-Weekly, ending on alternating Tuesday starting January, 22nd 1850.<br>E.g. Jan 22 1850, Feb 5 1850, ... ,Dec 26 2017, Jan 9 2018, Jan 23 2018 |
| 107 | Bi-Weekly, ending on alternating Wednesday starting January, 23rd 1850.<br>E.g. Jan 23 1850, Feb 6 1850, ... ,Dec 27 2017, Jan 10 2018, Jan 24 2018 |
| 108 | Bi-Weekly, ending on alternating Thursday starting January, 24th 1850.<br>E.g. Jan 24 1850, Feb 7 1850, ... ,Dec 28 2017, Jan 11 2018, Jan 25 2018 |
| 109 | Bi-Weekly, ending on alternating Friday starting January, 25th 1850.<br>E.g. Jan 25 1850, Feb 8 1850, ... ,Dec 29 2017, Jan 12 2018, Jan 26 2018 |
| 110 | Bi-Weekly, ending on alternating Saturday starting January, 26th 1850.<br>E.g. Jan 26 1850, Feb 9 1850, ... ,Dec 30 2017, Jan 13 2018, Jan 27 2018 |
| 112 | Semi-Monthly, 15th and end of month |
| 128 | Monthly |
| 155 | Bi-Monthly, with year ending in November |
| 156 | Bi-Monthly, with year ending in December |
| 170 | Quarterly, with year ending in October |
| 171 | Quarterly, with year ending in November |
| 172 | Quarterly, with year ending in December |
| 183 | Semi-Annual, with year ending in July |
| 184 | Semi-Annual, with year ending in August |
| 185 | Semi-Annual, with year ending in September |
| 186 | Semi-Annual, with year ending in October |
| 187 | Semi-Annual, with year ending in November |
| 188 | Semi-Annual, with year ending in December |
| 193 | Annual, with year ending in January |
| 194 | Annual, with year ending in February |
| 195 | Annual, with year ending in March |
| 196 | Annual, with year ending in April |
| 197 | Annual, with year ending in May |
| 198 | Annual, with year ending in June |
| 199 | Annual, with year ending in July |
| 200 | Annual, with year ending in August |
| 201 | Annual, with year ending in September |
| 202 | Annual, with year ending in October |
| 203 | Annual, with year ending in November |

| Value | Name |
|---|---|
| 204 | Annual, with year ending in December |

TransformationType

| Value | Name |
|---|---|
| 0 | None (DEFAULT) |
| 1 | YearOverYearPctChange |
| 2 | SimpleDifference |
| 3 | AnnualizedGrowth |
| 4 | PctChange |
| 8 | YearOverYearDiff |

## Appendix 4: Examples in C

**Necessary libraries**

```
using System;
using System.IO;
using System.Net;
using System.Text;
using System.Security.Cryptography;
using Newtonsoft.Json;
```

**Main program**

```
namespace APICodeSample
{
    class Program
    {
        private const string URI_ENDPOINT = "https://api.economy.com/data/v1/";
        private const string ACC_KEY_HEADER = "AccessKeyId";
        private const string SIGNATURE_HEADER = "Signature";
        private const string TIME_STAMP_HEADER = "TimeStamp";
        private const string ACC_KEY = "XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX";
        private const string ENC_KEY = "XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX";
        static void Main(string[] args)
        {
            // All logs and basket files are currently saved to the Desktop.
            string DesktopLocation =
            Environment.GetFolderPath(Environment.SpecialFolder.DesktopDirectory);
            string OrderID = "XXXXX"; // OrderID you want to run.
            string FileName = "XXXXX"; // Filename of file you want to retrieve.
            string result = GetBaskets(ACC_KEY, ENC_KEY);
            var jsonObject = JsonConvert.DeserializeObject(result);
            File.WriteAllText(String.Format(@»{ 0}\Baskets.json», DesktopLocation),
            JsonConvert.SerializeObject(jsonObject, Formatting.Indented));
            System.Threading.Thread.Sleep(1000);
            PostOrders postObject = new PostOrders();
            string postResult = PostOrders(ACC_KEY, ENC_KEY, OrderID);
            postObject = JsonConvert.DeserializeObject<PostOrders>(postResult);
            File.WriteAllText(String.Format(@"{0}\Execute.json", DesktopLocation),
            JsonConvert.SerializeObject(postObject, Formatting.Indented));
            PostOrders orderStatus = new PostOrders();
            bool orderCompleted = false;
            while (!orderCompleted)
            {
                System.Threading.Thread.Sleep(1000);
```

```
                string getOrdersResult = GetOrderStatus(ACC_KEY, ENC_KEY, postObject.OrderID);
                orderStatus = JsonConvert.DeserializeObject<PostOrders>(getOrdersResult);
                JsonConvert.SerializeObject(orderStatus, Formatting.Indented));
                if (orderStatus.dateFinished != null)
                {
                    orderCompleted = true;
                }
            }
            System.Threading.Thread.Sleep(1000);
            Stream orderStream = GetOrderStream(ACC_KEY, ENC_KEY, OrderID);
            using (var fs = new FileStream(String.Format(@"{0}\{1}",
            DesktopLocation, FileName), FileMode.Create))
            {
                orderStream.CopyTo(fs);
            }
        }
        // Methods are listed below.
    }
}
```

### Create signature

```
public static string CreateHMACSig(string accKey, string encKey, string timeStamp)
{
    string signature = string.Empty;
    byte[] keyBytes = Encoding.UTF8.GetBytes(encKey);
    using (HMAC hmac = new HMACSHA256(keyBytes))
    {
        byte[] bytesToHash = Encoding.UTF8.GetBytes(accKey + timeStamp);
        byte[] hashedBytes = hmac.ComputeHash(bytesToHash);
        for (int i = 0; i < hashedBytes.Length; i++)
        {
            signature += hashedBytes[i].ToString("X2");
        }
    }
    return signature;
}
```

### Retrieve a basket

```
public static string GetBaskets(string accKey, string encKey)
{
    string timeStamp = DateTime.UtcNow.ToString("yyyy-MM-ddTHH:mm:ssZ");
    string signature = CreateHMACSig(accKey, encKey, timeStamp);
    Uri uri = new Uri(URI_ENDPOINT + "baskets");
    WebRequest webRequest = WebRequest.Create(uri);
    webRequest.Method = "GET";
    webRequest.Headers.Add(ACC_KEY_HEADER, accKey);
    webRequest.Headers.Add(TIME_STAMP_HEADER, timeStamp);
    webRequest.Headers.Add(SIGNATURE_HEADER, signature);
    WebResponse webResponse = webRequest.GetResponse();
    string json;
    using (Stream stream = webResponse.GetResponseStream())
    {
        using (StreamReader streamReader = new StreamReader(stream))
        {
            json = streamReader.ReadToEnd();
        }
    }
    return json;
}
```

### Execute a basket

```
public static string PostOrders(string accKey, string encKey, string basketId)
{
    string timeStamp = DateTime.UtcNow.ToString("yyyy-MM-ddTHH:mm:ssZ");
    string signature = CreateHMACSig(accKey, encKey, timeStamp);
    Uri uri = new Uri(URI_ENDPOINT + "orders" + "?id=" + basketId + "&type=baskets&action=run");
    WebRequest webRequest = WebRequest.Create(uri);
    webRequest.Method = "POST";
    webRequest.Headers.Add(ACC_KEY_HEADER, accKey);
    webRequest.Headers.Add(TIME_STAMP_HEADER, timeStamp);
    webRequest.Headers.Add(SIGNATURE_HEADER, signature);
    webRequest.ContentLength = 0;
    WebResponse webResponse = webRequest.GetResponse();
    string json;
    using (Stream stream = webResponse.GetResponseStream())
    {
        using (StreamReader streamReader = new StreamReader(stream))
        {
            json = streamReader.ReadToEnd();
        }
    }
    return json;
}
```

### Get order status

```
public static string GetOrderStatus(string accKey, string encKey, string orderId)
{
    string timeStamp = DateTime.UtcNow.ToString("yyyy-MM-ddTHH:mm:ssZ");
    string signature = CreateHMACSig(accKey, encKey, timeStamp);
    Uri uri = new Uri(URI_ENDPOINT + "orders/" + orderId);
    WebRequest webRequest = WebRequest.Create(uri);
    webRequest.Method = "GET";
    webRequest.Headers.Add(ACC_KEY_HEADER, accKey);
    webRequest.Headers.Add(TIME_STAMP_HEADER, timeStamp);
    webRequest.Headers.Add(SIGNATURE_HEADER, signature);
    WebResponse webResponse = webRequest.GetResponse();
    string json;
    using (Stream stream = webResponse.GetResponseStream())
    {
        using (StreamReader streamReader = new StreamReader(stream))
        {
            json = streamReader.ReadToEnd();
        }
    }
    return json;
}
```

### Get output file

```
public static Stream GetOrderStream(string accKey, string encKey, string basketId)
{
    string timeStamp = DateTime.UtcNow.ToString("yyyy-MM-ddTHH:mm:ssZ");
    string signature = CreateHMACSig(accKey, encKey, timeStamp);
    Uri uri = new Uri(URI_ENDPOINT + "orders" + "?id=" + basketId + "&type=baskets");
    WebRequest webRequest = WebRequest.Create(uri);
    webRequest.Method = "GET";
    webRequest.Headers.Add(ACC_KEY_HEADER, accKey);
    webRequest.Headers.Add(TIME_STAMP_HEADER, timeStamp);
    webRequest.Headers.Add(SIGNATURE_HEADER, signature);
    WebResponse webResponse = webRequest.GetResponse();
    Stream stream = webResponse.GetResponseStream();
    return stream;
}
```

## Appendix 5: Examples in Java

## Necessary libraries

```
import java.io.BufferedReader;
import java.io.DataOutputStream;
import java.io.File;
import java.io.FileOutputStream;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.io.OutputStream;
import java.net.HttpURLConnection;
import java.net.URL;
import java.nio.file.Path;
import java.nio.file.StandardCopyOption;
import java.time.ZoneOffset;
import java.time.ZonedDateTime;
import java.util.concurrent.TimeUnit;
import java.util.stream.Stream;
import javax.crypto.Mac;
import javax.crypto.spec.SecretKeySpec;
import com.google.gson.Gson;
```

### Main program

```
public static void main(String[] args) throws Exception
{
    String DeskTopLocation = System.getProperty("user.home") + "/Desktop".toString();
    String OrderID = "XXXXX";
    String FileName = "XXXXX";
    GetBaskets(ACC_KEY, ENC_KEY);
    TimeUnit.SECONDS.sleep(1);
    Gson gson = new Gson();
    String postResult = PostOrders(ACC_KEY, ENC_KEY, OrderID);
    PostOrders postOrders = gson.fromJson(postResult, PostOrders.class);
    System.out.println(postOrders.orderId);
    PostOrders orderStatus;
    boolean orderCompleted = false;
    while(!orderCompleted)
    {
        TimeUnit.SECONDS.sleep(1);
        String getOrdersResult = GetOrderStatus(ACC_KEY, ENC_KEY, postOrders.orderId);
        orderStatus = gson.fromJson(getOrdersResult, PostOrders.class);
        if(orderStatus.dateFinished != null)
        {
            orderCompleted = true;
        }
    }
    TimeUnit.SECONDS.sleep(1);
    InputStream orderStream = GetOrderStream(ACC_KEY, ENC_KEY, OrderID);
    File targetFile = new File(DeskTopLocation + "\\" + FileName);
    java.nio.file.Files.copy(orderStream,targetFile.toPath(),StandardCopyOption.REPLACE_EXISTING);
}
```

### Create signature

```
public static String CreateHMACSig(String accKey, String encKey, String timestamp) throws Exception
{
    byte[] hmacData = null;
    String signature = "";
    String combinedKey = accKey + timeStamp;
    // Create a new key from the encryption key parameter.
    SecretKeySpec secretKey = new SecretKeySpec(encKey.getBytes("UTF-8"), "HmacSHA256");
```

```
    // Create a new HMAC SHA-256 Mac instance.
    Mac mac = Mac.getInstance("HmacSHA256");
    // Initialize this Mac instance with the secret key.
    mac.init(secretKey);
    // Compute the digest of this MAC on the bytes specified.
    hmacData = mac.doFinal(combinedKey.getBytes("UTF-8"));
    // Take one byte at a time from the array, convert to hex, append to sig string.
    for(byte b : hmacData){
        signature += String.format("%02X", b);
    }
    return signature;
}
```

### Retrieve a basket

```
public static void GetBaskets(String accKey, String encKey) throws Exception
{
    String timeStamp = ZonedDateTime.now(ZoneOffset.UTC).toString();
    String signature = CreateHMACSig(accKey, encKey, timeStamp);
    URL url = new URL(URI_ENDPOINT + "baskets");
    HttpURLConnection httpConnection = (HttpURLConnection) url.openConnection();
    httpConnection.setRequestMethod("GET");
    httpConnection.setRequestProperty(ACC_KEY_HEADER, accKey);
    httpConnection.setRequestProperty(TIME_STAMP_HEADER, timeStamp);
    httpConnection.setRequestProperty(SIGNATURE_HEADER, signature);
    try
    {
    // Create a buffer for reading off the HTTP input stream.
    BufferedReader inputBuffet = new BufferedReader(new InputStreamReader(httpConnection.getInputStream()));
    String responseData = "";
    String inputLine;
    // Read one line at a time from the buffer and add it to the response string.
    while((inputLine = inputBuffet.readLine()) != null)
    {
    responseData += inputLine;
    }
    inputBuffet.close();
    // Print response data from API.
    System.out.println(responseData.toString());
    }
    catch(Exception ex)
    {
    System.out.println(ex.toString());
    }
}
```

### Execute a basket

```
public static String PostOrders(String accKey, String encKey, String basketId) throws Exception
{
    String timeStamp = ZonedDateTime.now(ZoneOffset.UTC).toString();
    String signature = CreateHMACSig(accKey, encKey, timeStamp);
    URL url = new URL(URI_ENDPOINT+"orders"+"?id="+basketId+"&type=baskets&action=run");
    HttpURLConnection httpConnection = (HttpURLConnection) url.openConnection();
    httpConnection.setRequestMethod("POST");
    httpConnection.setRequestProperty(ACC_KEY_HEADER, accKey);
    httpConnection.setRequestProperty(TIME_STAMP_HEADER, timeStamp);
    httpConnection.setRequestProperty(SIGNATURE_HEADER, signature);
    httpConnection.setRequestProperty("Content-Length", "0");
    httpConnection.setDoOutput(true);
    byte[] data = {};
    DataOutputStream wr = new DataOutputStream( httpConnection.getOutputStream());
    wr.write( data );
    wr.flush();
```

```
    try
    {
        BufferedReader inputBuffet = new BufferedReader(
        new InputStreamReader(httpConnection.getInputStream()));
        String responseData = "";
        String inputLine;
        while((inputLine = inputBuffet.readLine()) != null)
        {
            responseData += inputLine;
        }
        inputBuffet.close();
        System.out.println(responseData.toString());
        return responseData.toString();
    }
    catch(Exception ex)
    {
        System.out.println(ex.toString());
        return ex.toString();
    }
}
```

### Get order status

```
public static String GetOrderStatus(String accKey, String encKey, String orderID) throws Exception
{
    String timeStamp = ZonedDateTime.now(ZoneOffset.UTC).toString();
    String signature = CreateHMACSig(accKey, encKey, timeStamp);
    URL url = new URL(URI_ENDPOINT + "orders/" + orderID);
    HttpURLConnection httpConnection = (HttpURLConnection) url.openConnection();
    httpConnection.setRequestMethod("GET");
    httpConnection.setRequestProperty(ACC_KEY_HEADER, accKey);
    httpConnection.setRequestProperty(TIME_STAMP_HEADER, timeStamp);
    httpConnection.setRequestProperty(SIGNATURE_HEADER, signature);
    try
    {
        BufferedReader inputBuffet = new BufferedReader(
        new InputStreamReader(httpConnection.getInputStream()));
        String responseData = "";
        String inputLine;
        while((inputLine = inputBuffet.readLine()) != null)
        {
            responseData += inputLine;
        }
        inputBuffet.close();
        System.out.println(responseData.toString());
        return responseData.toString();
    }
    catch(Exception ex)
    {
        System.out.println(ex.toString());
        return ex.toString();
    }
}
```

### Get output file

```
public static InputStream GetOrderStream(String accKey, String encKey, String basketId) throws Exception
{
    String timeStamp = ZonedDateTime.now(ZoneOffset.UTC).toString();
    String signature = CreateHMACSig(accKey, encKey, timeStamp);
    URL url = new URL(URI_ENDPOINT + "orders"+ "?id=" + basketId + "&type=baskets");
    HttpURLConnection httpConnection = (HttpURLConnection) url.openConnection();
    httpConnection.setRequestMethod("GET");
    httpConnection.setRequestProperty(ACC_KEY_HEADER, accKey);
```

```
        httpConnection.setRequestProperty(TIME_STAMP_HEADER, timeStamp);
        httpConnection.setRequestProperty(SIGNATURE_HEADER, signature);
        try
        {
            InputStream stream = httpConnection.getInputStream();
            return stream;
        }
        catch(Exception ex)
        {
            System.out.println(ex.toString());
            InputStream emptyStream = null;
            return emptyStream;
        }
    }
```

## Appendix 6: Examples in Python

Note: Unlike most programming languages, Python is sensitive to whitespace, and the line breaks in these code samples have been distorted for clarity.

#### Necessary libraries

```
import requests
import hashlib
import hmac
import datetime
import json
import pandas as pd
from time import sleep
```

#### API wrapper function

```
#####
# Function: Make API request, including a freshly generated signature.
#
# Arguments:
# 1. Part of the endpoint, i.e., the URL after "https://api.economy.com/data/v1/"
# 2. Your access key.
# 3. Your personal encryption key.
# 4. Optional: default GET, but specify POST when requesting action from the API.
#
# Returns:
# HTTP response object.
def api_call(apiCommand, accKey, encKey, call_type="GET"):
    url = "https://api.economy.com/data/v1/" + apiCommand
    timeStamp = datetime.datetime.strftime(datetime.datetime.utcnow(), "%Y-%m-%dT%H:%M:%SZ")
    payload = bytes(accKey + timeStamp, "utf-8")
    signature = hmac.new(bytes(encKey, "utf-8"), payload, digestmod=hashlib.sha256)
    head = {"AccessKeyId":accKey,
            "Signature":signature.hexdigest(),
            "TimeStamp":timeStamp}
    sleep(1)
    if call_type == "POST":
        response = requests.post(url, headers=head)
    elif call_type =="DELETE":
        response = requests.delete(url, headers=head)
    else:
    response = requests.get(url, headers=head)
    return(response)
```

#### Retrieve a basket

```
#####
# Setup:
# 1. Store your access key, encryption key, and basket name.
# Get your keys at:
# https://www.economy.com/myeconomy/api-key-info
ENC_KEY = "XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX"
ACC_KEY = "XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX"
BASKET_NAME = "Test Basket"
#####
# Identify a basket to execute:
# 2. Get list of baskets.
# 3. Extract the basket with a given name, and save its ID for later.
baskets = pd.DataFrame(json.loads(api_call("baskets/", ACC_KEY, ENC_KEY).text))
basketId = baskets.loc[baskets["name"]==BASKET_NAME, "basketId"].item()
print("Basket ID: " + basketId)
print("Basket Name: " + BASKET_NAME)

# 4. Execute a particular basket using its ID.
# This requires that the optional argument "type" be set to "POST".
call = ("orders?type=baskets&action=run&id=" + basketId)
order = api_call(call, ACC_KEY, ENC_KEY, call_type="POST")
orderId = order.text[12:48]
print("Order ID: " + orderId)

#####
# Download the output:
# 5. Periodically check if the order has completed.
if order.status_code != 200:
    sleep(3)
    print("Failed! Status Code: "+ str(order.status_code))
else:
    sleep(3)
    print("Successful Order! Status Code: " + str(order.status_code))

# 6. Download completed output.
new_call = ("orders?type=baskets&id=" + basketId)
get_basket = api_call(new_call, ACC_KEY, ENC_KEY)
get_basket = (str(get_basket.content).split("\\r\\n"))

# 7. Format the data frame.
data_df= pd.DataFrame(get_basket)
data_df = data_df[0].str.split(',', expand=True)
headers = data_df.iloc[0]
headers[0] = "Mnemonic"
data_df.columns = headers
data_df = data_df.set_index(data_df["Mnemonic"])
data_df = data_df[:-1]
data_df.dropna(axis=1, how='all')
filter = data_df != ""
data_df = data_df[filter]

# 8. Summary of the data frame.
num_rows = str(len(data_df.index))
num_columns = str(len(data_df.columns))
print("Ready to use "+ BASKET_NAME + " DataFrame!")
print("DataFrame contains: " + num_columns + " columns & " + num_rows + " rows")
```

## Appendix 7: Examples in R

Necessary libraries

```
library(digest)
library(jsonlite)
library(httr)
```

### API wrapper function

```
#####
# Function: Make API request, including a freshly generated signature.
#
# Arguments:
# 1. Part of the endpoint, i.e., the URL after "https://api.economy.com/data/v1/"
# 2. Your access key.
# 3. Your personal encryption key.
# 4. Optional: default GET, but specify POST when requesting action from the API.
#
# Returns:
# httr content object
api.call <- function(apiCommand, accKey, encKey, type="GET"){
    url <- paste("https://api.economy.com/data/v1/", apiCommand, sep="")
    print(url)
    timeStamp <- format(as.POSIXct(Sys.time()), "%Y-%m-%dT%H:%M:%SZ", tz="UTC")
    hashMsg <- paste(accKey, timeStamp, sep="")
    signature <- hmac(encKey, hashMsg, "sha256")
    Sys.sleep(1)
    if (type == "POST") {
        req <- httr::POST(url, httr::add_headers("AccessKeyId" = accKey,
                                                 "Signature" = signature,
                                                 "TimeStamp" = timeStamp))
    } else {
        req <- httr::GET(url, httr::add_headers("AccessKeyId" = accKey,
                                                "Signature" = signature,
                                                "TimeStamp" = timeStamp))
    }
    return(req)
}
```

### Retrieve a basket

```
#####
# Setup:
# 1. Store your access key, encryption key, and basket name.
# Get your keys at:
# https://www.economy.com/myeconomy/api-key-info
ACC_KEY     <- "XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX"
ENC_KEY     <- "XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX"
BASKET_NAME <- "Test basket"

#####
# Identify a basket to execute:
# 2. Get list of baskets.
# 3. Extract the basket with a given name, and save its ID for later.
baskets.json <- api.call("baskets/", ACC_KEY, ENC_KEY)
baskets      <- fromJSON(httr::content(baskets.json, as="text"))
basketID     <- baskets$basketId[baskets$name==BASKET_NAME]

# 4. Execute a particular basket using its ID.
# This requires that the optional argument "type" be set to "POST".
call    <- paste("orders?type=baskets&action=run&id=", basketID, sep="")
order   <- api.call(call, ACC_KEY, ENC_KEY, type="POST")
orderID <- fromJSON(httr::content(order, as="text"))$orderId

#####
# Download the output:
# 5. Periodically check if the order has completed.
call <- paste("orders/", orderID, sep="")
processing.check <- TRUE
while(processing.check) {
    status <- api.call(call, ACC_KEY, ENC_KEY)
    processing.check <- fromJSON(httr::content(status, as="text"))$processing
    Sys.sleep(10)
```

```
}
rm(status)

# 6. Download completed output.
call    <- paste("orders?type=baskets&id=", basketID, sep="")
request <- api.call(call, ACC_KEY, ENC_KEY)

# 7. This works for CSV baskets:
cat(httr::content(request, as="text", type="text/html", encoding="UTF-8"),file="basket.data", sep="\n")
df_data <- as.data.frame(read.csv("basket.data"))

# 8. View the data.
View(df_data)

# 9. Clean up.
unlink("basket.data")
rm(ACC_KEY, ENC_KEY)
rm(baskets, basketID, baskets.json)
rm(order, orderID)
rm(processing.check, call, request)
```

## Further reading

#### API documentation and functionality

- API key management
- Technical user guide
- Code samples in C#, Java, Python, R
- How to authenticate (See Authentication section)

#### Background on Data Buffet

- Using Data Buffet: Which series can you download?
- Using Data Buffet: Basket wild card expressions
- Using Data Buffet: Basket fields
- Using Data Buffet: Basket formulas
- Using Data Buffet: Transformations
- Using Data Buffet: Dates in basket output
- Using Data Buffet: Automatic retrieval
- Using Power Tools: Introduction to v.8

## Glossary

**access key:** Part of the credentials used to access the Data Buffet API. A unique 36-character hexadecimal string, which is combined with the encryption key (qv) to produce the signature (qv).

**API:** Application programming interface. Generically, a set of function signatures (input and output parameters) to perform documented behavior. See also: web API (qv).

**AutoCycle:** See: Moody's AutoCycle™ (qv).

**basket:** A facility of Data Buffet (qv). A basket consists of a list of time series (qv) mnemonics (qv) (or other expressions that evaluate to time series) and a set of formatting options. When executed, a basket retrieves the specified date range of the specified series, and specified metadata (qv) thereof, and emits an output file (qv) in the specified file format and layout. Analogous to a shopping list.

**basket formula:** An expression in a Data Buffet basket (qv) that operates on one or more native time series (qv) to synthesize a time series. May contain arithmetic, frequency conversion (qv) and/or transformation (qv) operations.

**basket output file:** A file emitted by the Data Buffet basket (qv) facility that contains the values of the specified series at a particular moment, in a particular specified format. When executed manually on Data Buffet, the output file is immediately delivered through the web browser; a basket can also be executed automatically by a periodic or triggered schedule. In both cases, a copy of the output file is cached on the user's account. The Data Buffet API (qv) enables access to basket output in JSON (qv) format.

**Coordinated Universal Time:** A civil time standard based on atomic clocks and astronomical measurements, and an associated representation using a 24-hour clock that includes year, month, day, hour, minute and second, and fixed punctuation. The format is yyyymm-ddThh:mm:ssZ, for example, 2018-07-30T21:03:28Z. This format is used when making requests to the Data Buffet API(qv). A.k.a. universal coordinated time, universal time coordinated, UTC.

**cURL:** Client for URLs. An open-source command-line software application to demonstrate HTTP (qv) requests and responses. Its syntax is often used to concisely document the behavior of web APIs (qv). See: curl.haxx.se

**CreditCycle:** see: Moody's CreditCycle™ (qv).

**CSV:** Comma-Separated Value. A file format that consists of plain text, where fields are separated by comma characters, and records are separated by line breaks.

**Data Buffet:** A web-based product of Moody's Analytics that contains a repository of economic and demographic time series (qv), organization mechanisms, and associated search, automatic retrieval, and visualization mechanisms, including baskets (qv). The database systems are shared between CreditForecast.com, DataBuffet.com, and other web products, and provide a back-end to Power Tools (qv) and the Data Buffet API (qv).

**Data Buffet API:** A web-based API (qv) that can access (a) the time series in Data Buffet (qv), and (b) baskets (qv) stored on a user's Data Buffet account.

**E-model:** A product of Moody's Analytics. A web-based tool that allows users to run scenarios with their own assumptions.

**encryption key:** Part of the credentials used to access the Data Buffet API. A unique 36-character hexadecimal string, which is combined with the access key (qv) to produce the signature (qv).

**end point:** In a web API (qv), a unique, static URL that represents an object or collection of objects; to interact with these resources, you point an HTTP client (qv) at the endpoint.

**frequency:** The characteristic fixed interval at which a time series (qv) is measured. Data Buffet (qv) contains time series of frequencies daily (i.e., one measurement per day), weekly, monthly, quarterly, semiannual and annual. Censal data are represented as an annual series with alternating ND values (qv).

**frequency conversion:** The mathematical translation of a time series (qv) from an input frequency (qv) to a different output frequency. Data Buffet provides GUI controls in its modules, and the CONVERT function for use in basket formulas (qv). Is impacted by the presence of ND values (qv).

**GUID:** Globally Unique Identifier. GUIDs are used in enterprise software development as database keys, component identifiers, and in COM programming; they are generated by individual users with an algorithm that virtually guarantees uniqueness. A GUID is a 128-bit integer, commonly expressed as a 32-character hexadecimal string delimited by hyphens. In the Data Buffet API, access and encryption keys, and basket and order identifiers, are GUIDs. A.k.a. Universally Unique Identifier, UUID.

**HMAC:** Hash-based Message Authentication Code. An international software standard (RFC2104 et seq) to verify the integrity of information transmitted over an unreliable medium such as the internet.

**HTTP:** HyperText Transfer Protocol. An international software standard (RFC2616 et seq) for an application-layer, client-server, stateless protocol for transmitting hypermedia documents and control information. See: https://www.w3.org/Protocols/, https://developer.mozilla.org/en-US/docs/Web/HTTP

**HTTP client:** Software that can communicate via HTTP (qv) with a server, for example, a web browser, cURL (qv), or a custom application that queries a web API (qv).

**JSON:** JavaScript Object Notation: An international software standard (ECMA-404), a lightweight data-interchange format that is easy for software to parse and generate, for humans to read and write, and is programming language-independent. JSON is the format in which the Data Buffet API (qv) delivers individual time series (qv) and basket output (qv). See: www.json.org.

**MIME:** Multipurpose Internet Mail Extension. An international software standard (RFC2045 et seq) that identifies how a file transmitted over the internet (as by email or HTTP) should be interpreted by the recipient.

**metadata:** Structured data that describes other data.

**mnemonic:** In Data Buffet (qv), an alphanumeric string that uniquely identifies a time series (qv). In its simplest form, a basket (qv) is a list of mnemonics.

**Moody's Analytics Power Tools for Microsoft Office:** A suite of Microsoft Office™ add-ins for Excel, PowerPoint and Word. The add-in for Excel allows direct retrieval of time series (qv) data and metadata into a spreadsheet; for all three applications, visualizations can be retrieved.

**Moody's AutoCycle™:** A software solution to forecast car prices, incorporating economic data and scenarios from Moody's Analytics. See: https://www.economy.com/products/data/autocycle

**Moody's CreditCycle™:** A software solution to model consumer credit risk; it combines customer data, economic data from Moody's Analytics, and consumer credit data from Equifax. See: https://www.economy.com/products/consumer-credit-analytics/moodys-creditcycle

**ND value:** No Data. Each observation (qv) in a time series (qv) on Data Buffet (qv) may be a specific numeric value or the "ND" marker. The specific meaning depends on the dataset, and may include "planned holiday interruption," "unplanned interruption," "value suppressed for confidentiality," "negligible," etc. The presence of ND values will impact the frequency conversion (qv) and transformation (qv) functions.

**OAuth:** An open software standard (RFC5849 et seq) for services over HTTP to provide "secure delegated access" whereby server owners authorize third-party access without the clients sharing their credentials.

**observation:** Each numeric measurement in a time series (qv).

**output file:** See: basket output file (qv).

**Power Tools:** See: Moody's Analytics Power Tools for Microsoft Office (qv).

**rate limiting:** With a web API (qv), a policy that controls how many requests from a given user will be processed per unit of time, typically for billing purposes or to promote adequate performance for all users.

**SHA256:** Secure Hash Algorithm. A cryptographic hash function that produces a 256-bit (32-byte) output.

**signature:** A cryptographic string generated from the access key (qv), encryption key (qv), and a time stamp and transmitted to a web API (qv) that uses HMAC (qv) authentication. See also: SHA256 (qv).

**throttling:** See: rate limiting.

**time series:** Generically, a vector of measurements (observations [qv]) at periodic intervals. In Data Buffet (qv), a data object that contains numeric values, metadata (qv) fields that explain how to interpret (frequency [qv] , etc.) and identify it (description, source), and one or more identifying mnemonics (qv).

**transformation:** In Data Buffet (qv), a mathematical translation of a time series (qv) for analysis. Transformations are provided by GUI controls in each module or can be expressed explicitly using a basket formula (qv). The main transformations are simple difference, year-over-year difference, percent change, year-over-year percent change, and annualized growth rate.

**UTC:** See: Coordinated Universal Time (qv).

**web API:** A programmatic, server-side interface consisting of one or more endpoints (qv), typically expressed in JSON (qv) or XML, and exposed to the web, typically by an HTTP server.

## About Moody's Analytics

Moody's Analytics helps capital markets and credit risk management professionals worldwide respond to an evolving marketplace with confidence. With its team of economists, the company offers unique tools and best practices for measuring and managing risk through expertise and experience in credit analysis, economic research, and financial risk management. By offering leading-edge software and advisory services, as well as the proprietary credit research produced by Moody's Investors Service, Moody's Analytics integrates and customizes its offerings to address specific business challenges.

Concise and timely economic research by Moody's Analytics supports fi rms and policymakers in strategic planning, product and sales forecasting, credit risk and sensitivity management, and investment research. Our economic research publications provide in-depth analysis of the global economy, including the U.S. and all of its state and metropolitan areas, all European countries and their subnational areas, Asia, and the Americas. We track and forecast economic growth and cover specialized topics such as labor markets, housing, consumer spending and credit, output and income, mortgage activity, demographics, central bank behavior, and prices. We also provide real-time monitoring of macroeconomic indicators and analysis on timely topics such as monetary policy and sovereign risk. Our clients include multinational corporations, governments at all levels, central banks, financial regulators, retailers, mutual funds, financial institutions, utilities, residential and commercial real estate firms, insurance companies, and professional investors.

Moody's Analytics added the economic forecasting firm Economy.com to its portfolio in 2005. This unit is based in West Chester PA, a suburb of Philadelphia, with offi ces in London, Prague and Sydney. More information is available at www.economy.com.

Moody's Analytics is a subsidiary of Moody's Corporation (NYSE: MCO). Further information is available at www.moodysanalytics.com.

DISCLAIMER: Moody's Analytics, a unit of Moody's Corporation, provides economic analysis, credit risk data and insight, as well as risk management solutions. Research authored by Moody's Analytics does not reflect the opinions of Moody's Investors Service, the credit rating agency. To avoid confusion, please use the full company name "Moody's Analytics", when citing views from Moody's Analytics.

## About Moody's Corporation

Moody's is an essential component of the global capital markets, providing credit ratings, research, tools and analysis that contribute to transparent and integrated financial markets. **Moody's Corporation** (NYSE: MCO) is the parent company of Moody's Investors Service, which provides credit ratings and research covering debt instruments and securities, and **Moody's Analytics**, which encompasses the growing array of Moody's nonratings businesses, including risk management software for financial institutions, quantitative credit analysis tools, economic research and data services, data and analytical tools for the structured finance market, and training and other professional services. The corporation, which reported revenue of $3.6 billion in 2016, employs approximately 11,500 people worldwide and maintains a presence in 41 countries.

**SECURITIES. NEITHER CREDIT RATINGS NOR MOODY'S PUBLICATIONS COMMENT ON THE SUITABILITY OF AN INVESTMENT FOR ANY PARTICULAR INVESTOR. MOODY'S ISSUES ITS CREDIT RATINGS AND PUB-LISHES MOODY'S PUBLICATIONS WITH THE EXPECTATION AND UNDERSTANDING THAT EACH INVESTOR WILL, WITH DUE CARE, MAKE ITS OWN STUDY AND EVALUATION OF EACH SECURITY THAT IS UNDER CONSIDERATION FOR PURCHASE, HOLDING, OR SALE.**

MOODY'S CREDIT RATINGS AND MOODY'S PUBLICATIONS ARE NOT INTENDED FOR USE BY RETAIL INVESTORS AND IT WOULD BE RECKLESS AND INAPPROPRIATE FOR RETAIL INVESTORS TO USE MOODY'S CREDIT RATINGS OR MOODY'S PUBLICATIONS WHEN MAKING AN INVESTMENT DECISION. IF IN DOUBT YOU SHOULD CONTACT YOUR FINANCIAL OR OTHER PROFESSIONAL ADVISER.

ALL INFORMATION CONTAINED HEREIN IS PROTECTED BY LAW, INCLUDING BUT NOT LIMITED TO, COPYRIGHT LAW, AND NONE OF SUCH IN-FORMATION MAY BE COPIED OR OTHERWISE REPRODUCED, REPACKAGED, FURTHER TRANSMITTED, TRANSFERRED, DISSEMINATED, REDISTRIB-UTED OR RESOLD, OR STORED FOR SUBSEQUENT USE FOR ANY SUCH PURPOSE, IN WHOLE OR IN PART, IN ANY FORM OR MANNER OR BY ANY MEANS WHATSOEVER, BY ANY PERSON WITHOUT MOODY'S PRIOR WRITTEN CONSENT.

All information contained herein is obtained by MOODY'S from sources believed by it to be accurate and reliable. Because of the possibility of human or mechanical error as well as other factors, however, all information contained herein is provided "AS IS" without warranty of any kind. MOODY'S adopts all necessary measures so that the information it uses in assigning a credit rating is of sufficient quality and from sources MOODY'S considers to be reliable including, when appropriate, independent third-party sources. However, MOODY'S is not an auditor and cannot in every instance independently verify or validate information received in the rating process or in preparing the Moody's publications.

To the extent permitted by law, MOODY'S and its directors, officers, employees, agents, representatives, licensors and suppliers disclaim liability to any person or entity for any indirect, special, consequential, or incidental losses or damages whatsoever arising from or in connection with the information contained herein or the use of or inability to use any such information, even if MOODY'S or any of its directors, officers, employees, agents, representatives, licensors or suppliers is advised in advance of the possibility of such losses or damages, including but not limited to: (a) any loss of present or prospective profi ts or (b) any loss or damage arising where the relevant financial instrument is not the subject of a particular credit rating assigned by MOODY'S.

To the extent permitted by law, MOODY'S and its directors, offi cers, employees, agents, representatives, licensors and suppliers disclaim liability for any direct or compensatory losses or damages caused to any person or entity, including but not limited to by any negligence (but excluding fraud, willful misconduct or any other type of liability that, for the avoidance of doubt, by law cannot be excluded) on the part of, or any contingency within or beyond the control of, MOODY'S or any of its directors, officers, employees, agents, representatives, licensors or suppliers, arising from or in connection with the information contained herein or the use of or inability to use any such information.

NO WARRANTY, EXPRESS OR IMPLIED, AS TO THE ACCURACY, TIMELINESS, COMPLETENESS, MERCHANTABILITY OR FITNESS FOR ANY PARTICULAR PURPOSE OF ANY SUCH RATING OR OTHER OPINION OR INFORMATION IS GIVEN OR MADE BY MOODY'S IN ANY FORM OR MANNER WHATSOEVER.

Moody's Investors Service, Inc., a wholly-owned credit rating agency subsidiary of Moody's Corporation ("MCO"), hereby discloses that most issuers of debt securities (including corporate and municipal bonds, debentures, notes and commercial paper) and preferred stock rated by Moody's Investors Service, Inc. have, prior to assignment of any rating, agreed to pay to Moody's Investors Service, Inc. for appraisal and rating services rendered by it fees ranging from $1,500 to approximately $2,500,000. MCO and MIS also maintain policies and procedures to address the independence of MIS's ratings and rating processes. Information regarding certain affi liations that may exist between directors of MCO and rated entities, and between entities who hold ratings from MIS and have also publicly reported to the SEC an ownership interest in MCO of more than 5%, is posted annually at www.moodys.com under the heading "Investor Relations — Corporate Governance — Director and Shareholder Affiliation Policy."

Additional terms for Australia only: Any publication into Australia of this document is pursuant to the Australian Financial Services License of MOODY'S affiliate, Moody's Investors Service Pty Limited ABN 61 003 399 657AFSL 336969 and/or Moody's Analytics Australia Pty Ltd ABN 94 105 136 972 AFSL 383569 (as applicable). This document is intended to be

provided only to “wholesale clients” within the meaning of section 761G of the Corpora-tions Act 2001. By continuing to access this document from within Australia, you represent to MOODY’S that you are, or are accessing the document as a representative of, a “wholesale client” and that neither you nor the entity you represent will directly or indirectly disseminate this document or its contents to “retail clients” within the meaning of section 761G of the Corporations Act 2001. MOODY’S credit rating is an opinion as to the creditwor-thiness of a debt obligation of the issuer, not on the equity securities of the issuer or any form of security that is available to retail investors. It would be reckless and inappropriate for retail investors to use MOODY’S credit ratings or publications when making an investment decision. If in doubt you should contact your fi nancial or other professional adviser.

Additional terms for Japan only: Moody’s Japan K.K. (“MJKK”) is a wholly-owned credit rating agency subsidiary of Moody’s Group Japan G.K., which is wholly-owned by Moody’s Overseas Holdings Inc., a wholly-owned subsidiary of MCO. Moody’s SF Japan K.K. (“MSFJ”) is a wholly-owned credit rating agency subsidiary of MJKK. MSFJ is not a Nationally Recognized Statistical Rating Organization (“NRSRO”). Therefore, credit ratings assigned by MSFJ are Non-NRSRO Credit Ratings. Non-NRSRO Credit Ratings are assigned by an entity that is not a NRSRO and, consequently, the rated obligation will not qualify for certain types of treatment under U.S. laws. MJKK and MSFJ are credit rating agencies registered with the Japan Financial Services Agency and their registration numbers are FSA Commissioner (Ratings) No. 2 and 3 respectively.

MJKK or MSFJ (as applicable) hereby disclose that most issuers of debt securities (including corporate and municipal bonds, debentures, notes and commercial paper) and preferred stock rated by MJKK or MSFJ (as applicable) have, prior to assignment of any rating, agreed to pay to MJKK or MSFJ (as applicable) for appraisal and rating services rendered by it fees ranging from JPY200,000 to approximately JPY350,000,000.

MJKK and MSFJ also maintain policies and procedures to address Japanese regulatory requirements.