**LinkUp RAW 2 Enhanced – Gold Copy**

**Getting Started**

Table of Contents

[LinkUp RAW Jobs Data 1](#_Toc50093109)

[Summary 2](#_Toc50093110)

[Files Needed 3](#_Toc50093111)

[The Files 4](#_Toc50093112)

[jobs\_base 4](#_Toc50093113)

[jobs\_log 6](#_Toc50093114)

[scrapelog 7](#_Toc50093115)

[reference 8](#_Toc50093116)

[descriptions 9](#_Toc50093117)

[auxiliary 10](#_Toc50093118)

[analytics\_core 11](#_Toc50093119)

[The “Smart Code” 11](#_Toc50093120)

[Python Examples 14](#_Toc50093121)

[FAQs 18](#_Toc50093122)

# LinkUp RAW Jobs Data

The LinkUp Raw Jobs data is a database of company level job posts, gathered directly from company career portals like “ibm.com/careers”. We do not capture jobs from “aggregators” like Indeed. This makes our raw data clean, with no duplicates and no stale data – among other things.

The data consists of raw data files, as well as analytics. The RAW data consists of job level meta data, and descriptions.

The engine behind LinkUp’s Job Search platform is driven by a very sophisticated and complex technology infrastructure that systematically “spiders” corporate web sites to gather information about open job postings.

While scraping an individual website once is not a technological feat today, doing it daily for greater than 55,000 companies is. LinkUp has spent years developing technology to deliver the most reliable, accurate and timely jobs data in the industry.

The jobs data history extends back to 2007-08-03, and covers some 55,000+ companies. The data is captured, analytics processed, and delivered daily. We started capturing the descriptions data in 2014.

# Summary

The ***LinkUp RAW Enhanced Gold Copy (LUGC)*** is a job level, point in time version of the LinkUp RAW Jobs (LU) data set. The primary advantages of LUGC are:

1. **Greatly simplified ETL**
2. **Jobs, not company\_ids**. The “job” becomes the atomic level of the raw data and analytics in LUGC, versus the “company\_id’ in LU. This allows for extremely detailed, flexible, and accurate analytics.
3. **Pure point in time (PIT).** With LUGC, we have created a PURE point in time representation of the RAW data, ***as you would have seen it on each day***, back to the inception of the data on 2007-08-03. With LU, you only have RAW data starting in the middle of 2018, and analytics starting in November of 2019. Meaning you are seeing the data as you would have seen it in 2018/2019 respectively, and you would have to build your own “near” point in time representations on a day to day basis. LU is NOT point in time.
4. **Normalized and cleansed.** LUGC corrects:
   1. **inaccurate “deleted” dates in LU**, affecting some 20+million hash. These inaccurate dates created look ahead bias as well as analytics “mismatch” where the logical “truth” that active jobs “today” should = active jobs “yesterday” + created jobs “today” – deleted jobs “today”
   2. **removal of NULL “hash”**
   3. **correction of duplicate “hash” that have multiple company\_ids**
   4. **point in time removal of “bad” hash on the day LU defined them as “bad”**. Not waiting for the monthly ”archive” fix
   5. **point in time removal of jobs when a “company\_id scraper” is decommissioned**
   6. **correction/normalization of country and state values**
   7. **further cleansing of the “title” field**
   8. **normalized timestamps**
   9. **eliminate files with different encoding** (LU daily files shift from ‘utf8’ to ISO 8859-1 for 2 months in 2018)

***and LUGC “normalizes”:***

* **“created” dates.** LUGC has a “created\_pit” (this is the actual point in time production “created” date. LU assigns the created at the actual time the job was scraped, all good. BUT, they then assign the “last\_checked” date when the company\_id level scrape ended – which determines the date the information is published to users. This creates situations where in the daily scrape cycle, the job was created on 1 day, but “checked” on another. This creates many problems PIT, and creates a scenario where historical analytics are “changed” from day to day. With LUGC, jobs and analytics are perfectly synced with the production date – the day a client would have actually seen the information.

1. **Fixed and Enhanced the scrapelog**. The LU scrapelog was incomplete. You had many company\_ids that you know were scraped on certain days, but no record of the scrape. In addition, the LU scrapelog did not capture when company\_id “scrapers” were decommissioned. The new LUGC scrapelog represents all possible known scrapes for ALL company\_id’s throughout the entire history back to 2007-08-03.
2. **Accurately captures job “reposts”.** When a company would “remove” a job, then sometime later “repost” it, LU would just change the “deleted” date from NULL to the date of the initial “removal”, then back to NULL when the company “reposted”. This is problematic in several ways. First, you lose any daily point in time information about the actual job posting. From your perspective today, the job was ALWAYS posted, when in fact we know it was removed for a period. In addition, the “deleted” date, like the “created” date, should be immutable. It should not change from NULL, to a date, to NULL again. Finally, the analytics created from this are continually restated upwards, overstating historical data, resulting in an “upward drift” of active job counts.
3. **Cutting edge usage of .parquet file format**. Smaller footprint, flexible and fast consumption, verified schemas, and “Smart” partitions.
4. **Greatly simplified file naming topology and organization**. No need to worry about daily or archive csv’s or xml’s., or multiple locations. LUGC has simple straightforward naming conventions.
5. **Greatly reduced file storage footprint**.
6. **Elegantly designed files allow for scaling in new analytics without changing schemas**
7. **Reference data consolidated into 1 file.**
8. **Greatly enhanced number of Aggregate Analytics (like core analytics by country, city, state, zip, all levels of SOC code, all levels of NAICS, and many more)**
9. **Job level enhanced analytics like “part time” and “full time” among many others**
10. **Addition of EnhancedAnalytics such as “Closed Duration” and “Normalized Active Jobs” among many others to be added**

# Files Needed

Before we continue, you can find the LU\_Enhnced\_Glossary.xlsx and the LU\_Enhanced\_Aggregate\_Analytics\_Codes.xlsx, as well as other documentation and information here:

<https://github.com/SmartMarketData/LinkUp_Enhanced_Documentation>

The files are simply designed and the topology has been simplified greatly. Basically, when you look at the data repository, this is what you see:



So ***what files do you need?*** Depends.

**If you want to build your own analytics, you will want:**

1. jobs\_base
2. jobs\_log
3. reference
4. scrapelog (not needed but can be used to verify data and look for scrape changes that may be responsible for quirky analytics)

**If you want to process the descriptions on your own with NLP, you will want:**

* jobs\_base
* descriptions
* jobs\_log (needed to further create analytics)
* reference (needed to create certain types of aggregates like “ticker” level)

*\*Note that SmartMarketData offers bespoke NLP work/analysis/analytics for any data set…*

**If you want just our analytics, you only need:**

* analyticsCore

# The Files

jobs\_base **– 7.22GB, adding <~10MB per day.**

This is an enhanced “jobs” table, which is just the immutable data for a given hash, with some important modifications to the LU standard data. This file is structured with the following schema:

* **jobid** – the jobid is a “smart” identifier. The first 8 digits are the created date, point in time, meaning the date that the client would have seen that the job was created the actual production date (see below “created\_pit”). The next 5 digits are the company\_id. The final 5 digits is the specific number of the job that was created that day for that company\_id. For example 202006090000100001 would be the jobid for Target for the first job that was created on 2020-06-09. While “hash” is random, this identifier has information and therefore value, flexibility and simplicity.
* **repostflag** – 0 or 1 based on if a repost or not
* **company\_id**
* **title**
* **city**
* **state**
* **zip**
* **country**
* **created** – we still give the original LU created as it is the “actual” scrape time vs. the “production” scrape time
* **parenthash** (this is the parent hash for the jobid. With this we can deal perfectly with all reposts).
* **created\_pit** (this is the actual PIT production created date, found also in the jobid, and fixes the “created” and the production mismatch issue.)



This and all the other files can be consumed directly from either your locally downloaded version, or from our S3 bucket, or your sync’ed AWS bucket.

To consume this file from a local source into a python dataframe for example, all you need to do is:



Where filedir is the local directory and parquet directory, for instance “c:\\lugc\\jobs\_base”

This will give you the entire jobs\_base file. No need to get monthly archives etc. And if you want to just grab a specific day, like the most current day, you can do this:



And, of course, you can grab multiple days, date ranges etc.

Ideally, you would have your “version” of LUGC synced with our AWS bucket. If you want to consume the data directly though (not download it locally) from either our bucket or yours, you can do this:



Where you would add your AWS credentials to the ‘key’ and ‘secret’ args, and the bucket…

The jobs\_base file becomes the “base” to the whole data set. It is point in time, and efficiently partitioned by the jobid’s “created\_pit”. You can consume it all, or just the dates you want, and never need to duplicate the storage or fields in your data repository. There is no monthly archive. No need for “deltas”. No need for separate point in time files as they are by nature PIT. This file becomes the base “jobid” reference table, replacing the LU “jobs” file.

jobs\_log **– 14.1GB, adding <~10MB per day.**

With the corrected/enhanced company level scrapelog (see below), and the jobs\_base as a foundation, we have created a true hash level point in time log for the entire history of the data. You can now look at 2007-08-03, and see the “daily” view on that date. The result though is a perfect PIT dataset, at the atomic “hash” level. This dataset is approaching 5 BILLION records! BUT, because of how we have constructed it, it requires only ~14GB of storage space, and is flexibly and quickly consumable vs. LinkUp’s current .csv and .xml methods. No archives, dailies, pit, or deltas needed. Pull the whole file up to date, or just select the scrape\_date (s) you want. Here are the fields:

* **jobid** – see above in jobs\_base
* **scrape\_date** – the PIT scrape date for the job
* **addremoveflag** - The addremoveflag is SMART!!! It can be either 1,0,-1, where 1 means the jobid was created, 0 means it was “last\_checked”, -1 means it was deleted.

This file is partitioned by “scrape\_date”, so you can pull all the jobs that were scraped or deleted on a specific date. It is also partitioned by the “addremoveflag”. So you can pull all “created” jobs where addremoveflag=1, or all “checked” jobs for addremoveflag=0, or all “deleted” jobs for addremoveflag=-1.

Like jobs\_base above, the parquet files can be consumed directly and efficiently.



scrapelog **– 121MB, adding <~50KB per day**

The scrapelog completes and improves the current LU Raw\_company\_scrape\_log\_daily\_yyyy-mm-dd.csv.gz.

We have created an enhanced “scrapelog”, which eliminates ALL of the gaps in the previous log and provides a complete historical base of company level scrapes. The scrapelog adjusts the deleted dates to point in time, so they reflect reality as you would have known it on any given day. We have also incorporated the dates that company level scrapes were decommissioned.

The scrapelog is quite efficient with fields:

* **company\_id**
* **actionDate**
* **actionType**. (which now can be “scrape”, “inferred scrape”, “scrapeadded”, “scrapedeleted”, “scrapechanged”)



reference **– ~12MB**

This is the consolidated reference data file. It contains LinkUp reference data, like the “company\_name” associated with the “company\_id”. It also contains the market data reference items like tickers, sedols, etc., as well as broader identifiers like LEI. The file is also parquet, and has the following schema:

* **company\_id**
* **refsource** – the source of the data
* **refsource\_id** – the primary identifier used by the refsource
* **reftype** – the type of reference data – i.e. URL, or company\_name, sedol, or ticker etc.
* **start\_date** – the start date of the mapping
* **end\_date** – the end date of the mapping
* **value** – the value (i.e. ‘AAPL’)
* **thrudate** – the date that the data was added to the file

This parquet is partitioned by ***“refsource”*** (i.e. Refinitiv/Factset/LinkUp/SmartMarketData etc.) as well as ***“reftype”*** (i.e. ticker, cusip, sedol, LEI, PermID etc.) for a fast and easy way to grab all the data at once, or just what you want (like the Factset stock\_ticker for instance).

The currently available refsource/reftype’s are:

* **Factset**
  + stock\_ticker\_primary
  + stock\_ticker
  + sedol\_primary
  + sedol
  + isin\_primary
  + isin
  + factset\_entity\_id
  + cusip\_primary
  + cusip
* **LinkUp**
  + open\_perm\_id
  + naics\_code
  + lei
  + company\_url
  + company\_name
* **SMD**
  + URL
  + TRBC\_IG (requires authorization)
  + TRBC\_I (requires authorization)
  + TRBC\_ES (requires authorization)
  + TRBC\_BS (requires authorization)
  + ticker\_ric (requires authorization)
  + OrgID
  + OpenPermID
  + NAICS1
  + MatchScoreFlag
  + Match Score
  + ManualCheckFlag
  + LEI
  + CIK



descriptions **– 92GB, growing ~80MB per day**

These are the job descriptions from each job post. They are unstructured text. LU started to capture these in 2014, although jobs that were present before 2014 and active when we started capturing the descriptions will show up as well. You will see descriptions all the way back to 2007 for evergreen type jobs. In general, starting in 2014, most jobs have an associated description.

The descriptions parquet schema is:

* **hash** – joins to the parenthash in the jobs\_base
* **description** – the unstructured text
* **hash1** – this is so you can iterate through this large data set easily in the file/partition. Simplifies processing
* **created\_pit** – this is so you can pull specific days or just add a new day. Simplifies processing

Again, you can grab the entire set of descriptions, or you can grab by the first character of the hash (0-9,a-f), or by the “created\_pit” that the ‘hash’ was originally created.



auxiliary – **6MB – grows slightly periodically**

This directory contains auxiliary tables, typically from external sources as “helper” tables. For instance, the ‘/smd-lu/auxiliary/soccode/soc\_2010’ directory parquet contains all of the 2010 soc code descriptions. The available files are:

* /smd-lu/auxiliary/soccode/soc\_2010 – the soc code description, partitioned by “code”. Pull the whole file, or a specific code(s).
* /smd-lu/auxiliary/soccode/soc\_2018 – the soc code description, partitioned by “code”. Pull the whole file, or a specific code(s). (2018 SOC codes are coming in the future)
* /smd-lu/auxiliary/naics – naics descriptions coming soon
* /smd-lu/auxiliary/trbc – trbc descriptions coming soon

The schema for the soccode tables is:

* title
* definition
* code

analytics\_core – **each Aggregate can be from 20-1GB depending on the aggregate**

The analytics\_core is our “core” analytics, created for any “aggregate”. An aggregate is a subset of jobs. For instance “Macro” is all jobs. “USMacro” is just jobs with country=’USA’. An infinite number of aggregates can exist.

### The “Smart Code”

The “aggregates” are organized by a “smart code”. You simply look at the [table description](https://github.com/lgreen4/LinkUp_Enhanced_Documentation/blob/master/LU_Enhanced_Aggregate_Analytics_Codes.xlsx), determine the aggregate you want and it’s “smart code”, and extract that “smart code” to get all the available aggregate data (download locally, sync to your AWS S3 bucket, or extract directly from our source AWS S3 bucket at /smd-lu/analyticsCore/{smart code}/analytics\_core . The code is a 14 digit code such as:

00011010000001

The first four digits are the “main aggregate id”, such as:

* Macro=0001
* USMacro10000=0002
* Word\_Parttime=1001
* Many more to come!

The next digit is the “level”. This can be:

* Point in time = 1,
* Non-Point in time = 2
* Point in time – cleansed = 3
* Non-Point in time cleansed = 4
* Others possible

The next 2 digits are the “bucket” code. This is what the “blevel” field is in the aggregate panel. This can be:

* macro=00
* company\_id=01
* ticker=02
* ticker\_fs=03
* sedol\_fs=04
* country=10
* state=11
* city=12
* zip=13
* naics\_s=30
* naics\_ss=31
* naics\_ig=32
* naics\_i=33
* naics\_si=34
* trbc\_es=35
* trbc\_bs=36
* trbc\_ig=37
* trbc\_i=38
* soc\_major=70
* soc\_minor=71
* soc\_broad=72
* soc\_detailed=73

The next 2 digits are the “sub-bucket” code. For instance if you want to see all soc\_major(s) by company\_id, you would have 7001. These are be the same codes as the “bucket” code above.

The next digit is the “top” code. This is where we can assign the number of “top” we want in the aggregate. For instance, the “top 10,000”, or “top 50”. These codes are:

* 1=1
* 2=10
* 3=50
* 4=100
* 5=500
* 6=1000
* 7=2000
* 8=3000
* 9=5000
* 0=10000

The next 3 digits are the “client code”, and are assigned for specific clients who want to keep proprietary analytics. The default is ‘000’ which means the aggregate is not client specific.

The final digit is the “period” code. These can be:

* 1=daily
* 2=weekly
* 3=monthly
* 4=quarterly
* 5=semi-annually
* 6=annually

Some common aggregates:

* Macro
* Country
* State
* Zip
* Company\_id
* Ticker
* Ticker\_fs
* Sedol\_fs
* XYZ Hedgefund’s universe
* Detailed SOC Code
* GICS Sector
* TRBC Industry
* NAICS Sub Industry

You name it, we can create it. Just drop a note to [operations@smartmarketdata.com](mailto:operations@smartmarketdata.com) if you have an idea for an aggregate we are missing!

The analyticsCore/analytics\_core schema:

* **bucketid** – the “smart code” described above
* **blevel** – the bucket “level” breakout – for instance company\_id, state, ticker, country, soc code, naics code etc.
* **jobsactive** – the number of active jobs on the given date
* **jobscreated** – the number of created jobs on the given date
* **jobsremoved** – the number of removed jobs on the given date
* **durationActive** – the average length of time that the current active jobs have been posted for the aggregate
* **durationClosed** – the average length of time that the “removed” jobs were posted
* **jobsscraped** – ignore this for now. Will be the actual number of jobs actually scraped on the given date.
* **processed** – the datetime the analytic was actually calculated. This is just metadata, not to be associated with the actual “jobsdate” below.
* **jobsdate** – the date the analytic references. If you have 20 active jobs with jobsdate of 2020-01-01, this means “there were 20 active jobs on 2020-01-01”



You can find a list of all production aggregate core analytics here: <https://github.com/SmartMarketData/LinkUp_Enhanced_Documentation/blob/master/LU_Enhanced_Aggregate_Analytics_Codes.xlsx>

### Python Examples

With the analyticsCore files, you can get core job analytics for virtually any “bucket”, like tickers, company\_ids, etc. For instance, to get the point in time analytics for Deloitte & Touche (company\_id=’134’), you would pull the file from your local repository or S3 like this:

|  |
| --- |
| getblevel=’134  linkupfiledir=yourdrive+'LinkUp\\bucket\\'  bucketid='00011010000001'  getdir=linkupfiledir+bucketid+'\\analytics\_core\\'  analytics=pqc.ParquetDataset(str(getdir))#,filters=[(jobsdate', '=', '2020-08-28'),])  analytics=analytics.read()  analytics=analytics.to\_pandas()  coidpanel=analytics.loc[analytics.blevel==getblevel,:] |



To get Factset stock ticker for Accenture, you would replace ‘getblevel’ and ‘bucketid’ above.

|  |
| --- |
| getblevel=’ACN|NYS|US’  bucketid='00011030000001' |



Or for the state of California, again just replace the ‘getblevel’ and the ‘bucketid’.

|  |
| --- |
| getblevel=’CA’  bucketid='00011030000001' |



And we are adding more all the time.

# FAQs

1. **How do I create my own analytics?**

You can create the analytics using the jobs\_log. At its base, if you just want a “Macro” look, you would query the addremoveflags 1=created\_pit, and -1=deleted. This gives you the bookends that each jobid (job) was active. There are any number of ways to finish the job based on where you hold your table, and the programming language you use. In python, you could unstack the table so you had a “jobid”, “created\_pit”,”deleted” in individual rows for each jobid. You could then loop or all at once count/sum/size the jobs where your target date is >= the created\_pit, and < the deleted.

To do this for company\_id’s (a “blevel” described above), you would first join/merge the jobs\_log with the jobs\_base, attaching the appropriate company\_id to the jobs\_log. Of course you can do this for any of the job specific items in the jobs\_base, like country, city, state, zip…

If you want ticker, or other reference data, you can join the company\_id as above, then join/merge the “reference” table, and attach the appropriate ticker, or other identifier you would like (like NAICS, or TRBC(if permissioned), or SEDOL, or PermID… You name it!).

WOnce attached, you would do the same exercise as above, but group by the ‘blevel’ you want (i.e. company\_id,ticker, sedol, etc…)

But don’t forget! We have already done most of these for you in the analyticsCore described above. And if we havn’t put in production yet something you want, let us know!

1. **The old LinkUp data had a deleted date for each “hash”. Where is it?**

The deleted date is a part of the “jobs\_log”. Each “jobid”, when deleted, is assigned a “-1” on the addremoveflag.

The jobs\_log also contains the created\_pit date(the pit corrected version of the old “created”) (“1”), as well as each and every scrape (“0”) which aligns with every “last\_checked” point in time from the old data set. Here we give you a full record. In the old LinkUp, you lost that hash level scrape information every day.

To capture all of the deleted dates, you can:

1. In the jobs\_log, query only for addremoveflag=-1. You will then have the deleted dates for each jobid.
2. You can extract directly from the parquet, as the file is partitioned by not only the scrape\_date, but also the addremoveflag. So in python, this would get you just the deleted dates:

**deleteddates=pqc.ParquetDataset(str(Path({locationofjobs\_log.parquet})), filters=[('addremoveflag', '=', '-1'),])**

1. **How about the created date?**

The created “timestamp is in the jobs\_base file. It is not used directly in the analytics, the “created\_pit” is. Many times they are the same. Both of these are immutable, and therefore in the jobs\_base file. You can also find the “created\_pit” as the first 8 digits of the jobid for each job. You can also capture the created\_pit from the job\_log with the same process as above for “deleted”, by using 1 instead of -1.