Reservoir Explorer

The reservoir data required for the tool has been scraped from <http://cdec.water.ca.gov/reservoir.html>. Parsekit is used to create two new parsers for the same. One parser is used to collect the historic data (2014 to present) about the reservoirs and the second parser collects the daily data. The data parsed is uploaded to a AWS S3 (Simple Storage Service) bucket.

(“Reservoir” and “Dam” are used interchangeably)

**Data Parsed by the parsers:**

* Dam Abbreviation (Dam\_Abbr)
* Dam Name
* Date
* Latitude
* Longitude
* Capacity
* Storage
* Elevation
* Percentage (Percentage of dam filled)

Currently, data about 78 dams is being collected. The list is shared in Appendix 1.

The two parsers are:

* historic\_parser
* daily\_parser

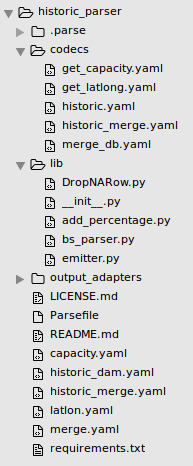
# **historic\_parser:**

The parsing is done in two parts – First the static data about dams like Dam\_Abbr, dam name, latitude, longitude and capacity are parsed using urls - <http://cdec.water.ca.gov/misc/daily_res.html> and <http://cdec.water.ca.gov/misc/resinfo.html>.

Then the remaining data is scraped from the individual dam websites using the url -

'http://cdec.water.ca.gov/cgi-progs/queryDaily?%s&d=%s&span=%s' % (dam\_id, date, span)

## **Parser Structure:**



l**atlon.yaml**

Step:

1. Download location data (Lat&Long) from URL: <http://cdec.water.ca.gov/misc/daily_res.html>  
2. Apply the codec ‘get\_latlong.yaml’ to return a database ‘latlong’, including Dam\_ABBR, Latitude, Longitude.

**capacity.yaml**

Steps:

1. Fetch capacity data from URL: <http://cdec.water.ca.gov/misc/resinfo.html>  
2. Apply the codec ‘get\_capacity.yaml’ to return a database ‘capacity’, including Dam\_ABBR, Dam\_Name, Capacity.

**merge.yaml**

Steps:

1. Emit SQL call to inner join two databases, ‘latlong’ and ‘capacity’.  
2. Apply the codec ‘merge\_db.yaml’ to return a database ‘capacity\_and\_latlong’, including Dam\_ABBR, Dam\_Name, Capacity, Latitude, Longitude.

notes: The primary keys of all databases above is Dam\_ABBR.

**Codec get\_capacity.yaml**

**Dependency:** "lib.DropNARow": "DropNARow"

Steps:

1. Set the table schema.  
2. Scrape Records off the Page by using extract.formats.html.ReadRecords  
3. Drop rows with 'Dam\_ABBR==NA' by using DropNARow.dropna.  
4. Get rid of garbage characters.  
5. Coerce empty strings to Null  
6. Specify Table Name: capacity  
7. Output Records to SQLite Table with following setting.

**yaml**  
 overwrite: no  
 upsert: yes  
 repository: sqlite:///.parse/static.db  
 primary keys:  
 capacity:  
 - "Dam\_ABBR"

**Codec get\_latlong.yaml**

**Dependency**: "lib.DropNARow": "DropNARow"

Steps:

1. Set the table schema  
2. Scrape Records off the Page by using extract.formats.html.ReadRecords.  
3. Drop rows with 'Dam\_ABBR==NA' by using DropNARow.dropna.  
4. Get rid of garbage characters.  
5. Coerce empty strings to Null  
6. Specify Table Name: latlong  
7. Output Records to SQLite Table with following setting.

**Yaml** overwrite: no  
 upsert: yes  
 repository: sqlite:///.parse/static.db  
 primary\_keys:  
 latlong:  
 - "Dam\_ABBR"

**Codec merge\_db.yaml**

**Dependency**: "lib.DropNARow": "DropNARow"

Steps:

1. Set the table schema  
2. Read Records from ‘sqlite:///.parse/static.db’  
3. Drop rows with 'Dam\_ABBR==NA' by using DropNARow.dropna.  
4. Get rid of garbage characters.  
5. Coerce empty strings to Null  
6. Specify Table Name: capacity\_and\_latlong  
7. Output Records to SQLite Table ‘cache’ and ‘static’ with following setting.

**yaml**

overwrite: no

upsert: yes

repository: sqlite:///cache.db

primary\_keys:

capacity\_and\_latlong:

- "Dam\_ABBR"

**yaml**

overwrite: no

upsert: yes

repository: sqlite:///static.db

primary\_keys:

capacity\_and\_latlong:

- "Dam\_ABBR"

**historic\_dam.yaml:**

Steps:

1) Emit custom urls - emitter.emit\_paths

2) Download the html pages

3) Pass the pages to 'historic' codec

**Dependencies:** emitter.emit\_paths

**emitter.emit\_paths:(emitter.py)**

A custom step is introduced to emit dynamic urls for each dam. Currently the attributes like Dam\_Abbr, date and time-span required to create the url are inside the custom code.

Current settings: date = today’s date, span = 2 years, Dam\_Abbr = extracted from table ‘capacity\_and\_latlong’.

If changes are to be made in the url and attributes, the custom library code requires changes accordingly.

Dam\_ABBRs are queried from the sqlite database created in the 'latlon' codec."sqlite3" library is used to query the database stored in parser directory.

**Improvements:** Attributes like Dam\_abbr, date, time\_span can be made as arguments and custom step can be made more reusable.

**Custom url: 'http://cdec.water.ca.gov/cgi-progs/queryDaily?%s&d=%s&span=%s' % (dam\_id, date, span)**

**historic Codec:**

Steps:

1) Set table schema

2) Set table name as ‘historic’

3) Parse the html pages downloaded in historic\_dam.yaml-step 2 to get dam attributes- date, storage, elevation. Custom step - parser.parser

4) Load the table into a local SQLite database called cache.db

**Dependencies:** parser.parser:(bs\_parser.py)

**parser.parser:(bs\_parser.py)**

A custom step to parse the html pages to get dam attributes – date, storage, elevation

The python code makes use of Beautiful Soup library to scrape the data. All the tags and attributes of the html page are hardcoded as per the structure of the html page to retrieve the required data.

**Improvements:** There maybe more than one table present in the html page (eg. <http://cdec.water.ca.gov/cgi-progs/queryDaily?ICH&d=2017011618:53&span=1day> has more than 1 html table) Parsing through such pages is currently handled manually, however new cases may arise eg. more than 3 tables in one page. This issue needs to be given some consideration.

**historic\_merge.yaml**

Steps:

1) Create a SQL query to join data from table ‘historic’ and ‘capacity\_and\_latlong’ saved in cache.db using the Dam\_Abbr, Date as the primary key between the two tables.

2) Call the codec ‘historic\_merge.yaml’.

**historic\_merge.yaml Codec**

Steps:

1) Set the initial table schema  
2) Execute the query to join the tables  
3) Add percentage to table schema  
4) Clean the NA rows  
5) Set table name as final\_historic  
6) Upload the data to AWS S3 bucket

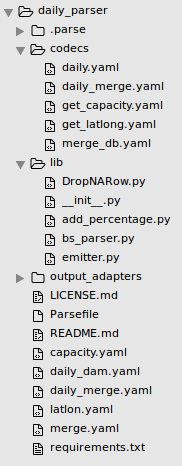
**daily\_parser:**

The parsing is done in two parts – First the static data about dams like Dam\_Abbr, dam name, latitude, longitude and capacity are parsed using urls - <http://cdec.water.ca.gov/misc/daily_res.html> and <http://cdec.water.ca.gov/misc/resinfo.html>.

Then the remaining data is scraped from the individual dam websites using the url -

'http://cdec.water.ca.gov/cgi-progs/queryDaily?%s&d=%s&span=%s' % (dam\_id, date, span)

**Parser Structure:**



l**atlon.yaml**

Step:

1. Download location data (Lat&Long) from URL: <http://cdec.water.ca.gov/misc/daily_res.html>  
2. Apply the codec ‘get\_latlong.yaml’ to return a database ‘latlong’, including Dam\_ABBR, Latitude, Longitude.

**capacity.yaml**

Steps:

1. Fetch capacity data from URL: <http://cdec.water.ca.gov/misc/resinfo.html>  
2. Apply the codec ‘get\_capacity.yaml’ to return a database ‘capacity’, including Dam\_ABBR, Dam\_Name, Capacity.

**merge.yaml**

Steps:

1. Emit SQL call to inner join two databases, ‘latlong’ and ‘capacity’.  
2. Apply the codec ‘merge\_db.yaml’ to return a database ‘capacity\_and\_latlong’, including Dam\_ABBR, Dam\_Name, Capacity, Latitude, Longitude.

notes: The primary keys of all databases above is Dam\_ABBR.

**get\_capacity.yaml**

**Dependency:** "lib.DropNARow": "DropNARow"

Steps:

1. Set the table schema.  
2. Scrape Records off the Page by using extract.formats.html.ReadRecords  
3. Drop rows with 'Dam\_ABBR==NA' by using DropNARow.dropna.  
4. Get rid of garbage characters.  
5. Coerce empty strings to Null  
6. Specify Table Name: capacity  
7. Output Records to SQLite Table with following setting.

**yaml**  
 overwrite: no  
 upsert: yes  
 repository: sqlite:///.parse/dam.db  
 primary keys:  
 capacity:  
 - "Dam\_ABBR"

**get\_latlong.yaml**

**Dependency**: "lib.DropNARow": "DropNARow"

Steps:

1. Set the table schema  
2. Scrape Records off the Page by using extract.formats.html.ReadRecords.  
3. Drop rows with 'Dam\_ABBR==NA' by using DropNARow.dropna.  
4. Get rid of garbage characters.  
5. Coerce empty strings to Null  
6. Specify Table Name: latlong  
7. Output Records to SQLite Table with following setting.

**Yaml** overwrite: no  
 upsert: yes  
 repository: sqlite:///.parse/dam.db  
 primary\_keys:  
 latlong:  
 - "Dam\_ABBR"

**merge\_db.yaml**

**Dependency**: "lib.DropNARow": "DropNARow"

Steps:

1. Set the table schema  
2. Read Records from ‘sqlite:///.parse/dam.db’  
3. Drop rows with 'Dam\_ABBR==NA' by using DropNARow.dropna.  
4. Get rid of garbage characters.  
5. Coerce empty strings to Null  
6. Specify Table Name: capacity\_and\_latlong  
7. Output Records to SQLite Table ‘dam’ and ‘cache’ with following setting.

**yaml**

overwrite: no

upsert: yes

repository: sqlite:///dam.db

primary\_keys:

capacity\_and\_latlong:

- "Dam\_ABBR"

**yaml**

overwrite: no

upsert: yes

repository: sqlite:///cache.db

primary\_keys:

capacity\_and\_latlong:

- "Dam\_ABBR"

**daily\_dam.yaml:**

Steps:

1) Emit custom urls - emitter.emit\_paths

2) Download the html pages

3) Pass the pages to 'daily' codec

**Dependencies:**

**emitter.emit\_paths:(emitter.py)**

A custom step is introduced to emit dynamic urls for each dam. Currently the attributes like Dam\_Abbr, date and time-span required to create the url are inside the custom code.

Current settings: date = today’s date, span = 1 day, Dam\_Abbr = extracted from table ‘capacity\_and\_latlong’.

If changes are to be made in the url and attributes, the custom library code requires changes accordingly.

Dam\_ABBRs are queried from the sqlite database created in the 'latlon' codec."sqlite3" library is used to query the database stored in parser directory.

**Improvements:** Attributes like Dam\_abbr, date, time\_span can be made as arguments and custom step can be made more reusable.

**Custom url: 'http://cdec.water.ca.gov/cgi-progs/queryDaily?%s&d=%s&span=%s' % (dam\_id, date, span)**

**daily Codec:**

Steps:

1) Set table schema

2) Set table name as ‘daily\_dam’

3) Parse the html pages downloaded in daily\_dam.yaml-step 2 to get dam attributes- date, storage, elevation. Custom step - parser.parser

4) Load the table into a local SQLite database called cache.db

**Dependencies:**

**parser.parser:(bs\_parser.py)**

A custom step to parse the html pages to get dam attributes – date, storage, elevation

The python code makes use of Beautiful Soup library to scrape the data. All the tags and attributes of the html page are hardcoded as per the structure of the html page to retrieve the required data.

**Improvements:** There maybe more than one table present in the html page (eg. <http://cdec.water.ca.gov/cgi-progs/queryDaily?ICH&d=2017011618:53&span=1day> has more than 1 html table) Parsing through such pages is currently handled manually, however new cases may arise eg. more than 3 tables in one page. This issue needs to be given some consideration.

**historic\_merge.yaml**

Steps:

1) Create a SQL query to join data from table ‘historic’ and ‘capacity\_and\_latlong’ saved in dam.db using the Dam\_Abbr as the primary key between the two tables.

2) Call the codec ‘daily\_merge.yaml’.

**daily\_merge.yaml Codec**

Steps:

1) Set the initial table schema  
2) Execute the query to join the tables  
3) Add percentage to table schema  
4) Clean the NA rows  
5) Set table name as dam\_data  
6) Upload the data to AWS S3 bucket

**Issues**

1. The data is not yet uploaded to the AWS S3 bucket because of a technical issue.
2. The code needs some peer review

**Future Improvements:**

1. Improvements in the code to make them reusable.
2. The attributes to be scraped can be increased. The attributes commented in the bs\_parser.py inside lib folder can be scraped by the existing code by uncommenting the attributes.
3. Not each dam has its daily data. Dams whose monthly or yearly data is available can be included in the tool.

**Appendix 1: List of dams whose data is parsed**

|  |  |
| --- | --- |
| **Dam\_Abbr** | **Dam Name** |
| CLE | Trinity |
| LEW | Lewiston |
| SCD | Pillsbury |
| COY | Coyote Valley |
| WRS | Warm Springs |
| SAT | San Antonio |
| CVE | Calaveras |
| CCH | Cachuma |
| PYM | Pyramid |
| CAS | Castaic |
| PRR | Perris |
| SVO | Seven Oaks |
| DMV | Diamond Valley |
| SHA | Shasta |
| KES | Keswick |
| WHI | Whiskeytown |
| BLB | Black Butte Rereg |
| MMW | Indian Ole |
| ALM | Canyon |
| BTV | Butt Valley |
| ANT | Antelope |
| BCL | Bucks |
| DAV | Grizzly Valley |
| FRD | Frenchman |
| LGV | Little Grass Valley |
| SLC | Sly Creek |
| ORO | Oroville |
| TAB | Thermalito |
| BUL | New Bullards Bar |
| ENG | Englebright Narrows |
| CFW | Camp Far West |
| LVY | Lake Valley |
| FMD | French Meadows |
| HHL | Lower Hell Hole |
| LON | Loon Lake |
| UNV | Union Valley |
| ICH | Ice House |
| SLB | Slab Creek |
| CPL | Caples |
| FOL | Folsom |
| SIV | Silver |
| NAT | Natoma |
| INV | Indian Valley |
| CLA | Clear Lake Imp |
| BER | Montecello |
| SLS | Salt Springs |
| PAR | Pardee Reservoir |
| CMN | Camanche |
| NHG | New Hogan |
| FRM | Farmington |
| SPM | Spicer Meadows |
| DON | Donnells |
| RLF | Relief |
| BRD | Beardsley |
| SWB | Main Strawberry |
| LYS | Lyons |
| NML | New Melones |
| TUL | Tulloch |
| CHV | Cherry Valley |
| HTH | O Shanessey |
| DNP | Don Pedro |
| EXC | New Exchequer |
| BUC | Buchanan |
| HID | Hidden |
| MIL | Friant |
| SNL | San Luis |
| CTG | Courtright |
| WSN | Wishon |
| PNF | Pine Flat |
| TRM | Terminus |
| SCC | Success |
| ISB | Isabella |
| STP | Stampede |
| INP | Independence |
| BOC | Boca |
| PRS | Prosser |
| DNN | Donner |
| MRT | Martis Creek Reservoir |