

x12-837-fake-data-generator

For generating realistic but fake 837 files for learning purposes in healthcare, as well as parsing them into structured CSV files.

Project Hierarchy

- Generator and Parsers:
 - **Generator:** Generates 837 files with realistic but fake data
 - **Parser:** Parses 837 files into structured CSVs for analysis
 - The generator and parse each has:
 - **API:** Core logic for generating or parsing
 - **CLI:** Command-line interface for generating or parsing
 - **Web App:** Flask web app for generating or parsing 837 files
- **Web:** Combined web app for generation and parsing

```

x12-837-fake-data-generator/
|
└── generator_837/                               # Core directory for the
    ├── generator_code                           # API logic for generating 837
    │   ├── api/                                 # Generates X12 837 transactions
    │   │   ├── generator.py                   # Functions for generating X12
    │   │   └── segments.py                  segments
    │   ├── segments/                            # Utility functions
    │   │   ├── utils.py                      # Functions for loading reference
    │   │   └── data_loader.py               data
    │   ├── data/                                # Reference data (e.g., ICD
    │   │   ├── reference_data/             codes, payer data)
    │   │   │   ├── icd10.csv                # ICD-10 codes
    │   │   │   ├── cpt4.csv                 # CPT codes
    │   │   │   ├── payers.csv              # Payer data
    │   │   │   └── npi_orgs.csv            # NPI and organization data
    │   │   └── __init__.py
    │   └── cli/                                  # Command-line interface (CLI)
    │       └── main.py                      # CLI entry point for generating
    └── files/                                 # Flask web app for generating
        └── __init__.py
    └── web/                                    # Web app entry point for
        ├── app.py                             generation
        └── templates/                         # Web app front-end for
            └── index.html

```

```

generation
    └── __init__.py
    └── __init__.py

    ├── parser_837/
    │   code
    │   ├── api/
    │   │   ├── parser.py
    │   │   ├── claim_services/
    │   │   │   ├── cpt_hcpcs.py
    │   │   │   └── __init__.py
    │   │   ├── claim_diagnoses/
    │   │   │   ├── icd_diagnoses.py
    │   │   │   └── __init__.py
    │   │   ├── transaction_header/
    │   │   │   ├── header.py
    │   │   │   └── ...
    │   │   └── __init__.py
    │   └── __init__.py

    └── cli/
        for parsing
            └── main.py

files
    └── __init__.py

    ├── web/
    │   ├── app.py
    │   ├── templates/
    │   │   └── index.html
    │   └── __init__.py
    └── __init__.py

    └── web/
and parsing
    ├── app.py
    ├── templates/
    │   └── index.html
    generation and parsing
    └── __init__.py

    └── generator_837_output/
generated files
    ├── parser_837_output/
parsed files
    ├── .gitignore
in Git
    ├── README.md
    ├── requirements.txt
    └── venv/
included in repo)

```

Core directory for the parser

API logic for parsing 837 files
Main parser logic
Service-line-specific parsing
CPT/HCPCS parsing logic

Diagnosis-specific parsing
ICD parsing logic

Header parsing
Main header parser logic
Individual header parsers

Command-line interface (CLI)

CLI entry point for parsing

Flask web app for parsing files
Web app entry point for parsing

Web app front-end for parsing

Combined web app for generation

Unified web app entry point

Combined front-end for

Default output directory for

Default output directory for

Files and directories to ignore

Project documentation
Python dependencies
Virtual environment (not

The basic idea:

- (1) Being able to generate 'realish' profiles of patients with procedures and diagnosis
- (2) Push this data into a X12 837 form (.txt file) like what would be sent to a payer
- (3) Then also have a parser to read this data into csv/tabular structure for analyses that would be useful for healthcare providers and payers
- (4) Overall, this should promote training and understanding of the X12 837 form and how it is used in healthcare

Viewing example data:

1. Navigate to `837_generator_output` folder for the generated 837 files
2. Navigate to `837_parser_output` folder for the parsed csv files of the generated 837 files

1. CLI for generating 837 files

Creating your example data with the CLI:

1. Clone the repo
2. Create a virtual environment:
 - `python3 -m venv venv`
 - `source venv/bin/activate`
3. Install the requirements:
 - `pip install -r requirements.txt`
4. Run the script with the cli:
 - `python -m generator_837.cli.main -n 13 -o generator_837_output`
 - this will generate 20 files within the `837_generator_output` folder
 - The two current parameters:
 - `n` or `--number` - number of 837 files to generate
 - `o` or `--output` - directory to save the 837 files

Parsing your example data with the CLI:

1. Run the script with the cli:
 - `python -m parser_837.cli.main -i generator_837_output -o parser_837_output`
 - this will parse the 837 files in the `generator_837_output` folder and save the parsed data in the `parser_837_output` folder
 - The two current parameters:
 - `i` or `--input` - directory containing the 837 files to parse
 - `o` or `--output` - directory to save the parsed CSV files
2. Or if you want to parse an individual file:
 - `python -m parser_837.cli.main -i generator_837_output/837_example_1.txt -o parser_837_output`

2. Web App for Generation and Parsing:

Web App for Generation and Parsing:

Features:

- Generate X12 837 files with custom inputs and download them as a ZIP.
- Upload X12 837 files, parse them, and download the results as a ZIP of CSV files.
- API endpoints for generating and parsing X12 837 files.

Instructions:

1. Clone the repo
2. Create a virtual environment:
 - `python3 -m venv venv`
 - `source venv/bin/activate`
3. Install the requirements:
 - `pip install -r requirements.txt`
4. Run the Flask app:
 - `python web/app.py`
 - Then navigate to `http://localhost:5007/` or `0.0.0.0:5007` in your browser
 - You can then select the number of 837 files to generate and download them
5. If you don't want to do this, I have created a docker image that is deployed on GCP that you can use for testing found here:
 - [837 Generator Web App: https://form837-447631255961.us-central1.run.app](https://form837-447631255961.us-central1.run.app)

Web API Endpoints:

- `/api` - Documentation for the API
- `/api/generate` - GET or POST request to generate 837 files
- `/api/parse` - POST request to parse uploaded 837 files
- Example usage:

GET request for generating a single fake 837 file text file:

```
curl -X GET "http://localhost:5007/api/generate/" --output  
api_output/api_single_fake_claim_2.txt
```

Expected Result: `single_fake_claim.txt` is downloaded containing one valid 837 claim.

POST request for generating multiple fake 837 files that are returned in a .zip file:

```
curl -X POST "http://localhost:5007/api/generate/" \
-H "Content-Type: application/json" \
-d '{"number": 5}' --output
api_output/api_multiple_fake_claims_2.zip
```

Post request for parsing 837 files:

```
curl -X POST "http://localhost:5007/api/parse/" \
-F "file@api_output/api_single_fake_claim.txt" --output
api_output/api_parsed_837_files_2.zip
```

Testing of generated 837 files:

- For checking data structure/schema of generated 837 files from this repo with third party tools you can use:
 - <https://www.stedi.com/edi/inspector>
 - <https://datainsight.health/edi/viewer/>

Required Loops and Segments:

Heading Section:

- ST: Transaction Set Header (mandatory)
- BHT: Beginning of Hierarchical Transaction (mandatory)
- NM1 Loop 1000 for Submitter and Receiver, with optional N3 (address), N4 (geographic location), and PER (contact information).

Detail Section:

- HL (Hierarchical Level) in Loop 2000 (mandatory) to represent different levels (e.g., billing provider, subscriber).
- NM1 Loop 2010 for individual providers (billing, pay-to provider, etc.), with optional N3 (address), N4 (geographic), DMG (demographic), and PER segments.

Claim Level:

- CLM Loop 2300 contains the main CLM segment and can contain additional data such as DTP (Date or Time Period), HI (Health Care Diagnosis Codes), and REF (reference identifiers).

Service Line Level (Loop 2400):

- Each service line must contain LX (Line Number) and SV1 (Professional Service) or SV2 (Institutional Service).
- May contain DTP, QTY, and REF segments.

Common Missing Segments:

Loop 1000:

- Ensure that NM1-40 (Receiver) and NM1-41 (Submitter) both include necessary segments.

Loop 2010 for Billing Provider:

- REF segment after NM1-85 with the Tax ID (qualifier EI).
- Optional N3 and N4 segments for billing provider address details.

Loop 2000

- With HL for defining hierarchical levels (this might be required if not already included).

TO DO:

- need to replace some of the faker addresses with REAL ADDRESS that are randomly selected; or perhaps with real people names as well
- this will then help with the later merging of resources and services
- and potential identification of payers (?) or NPI locations (?) that are in X proximity

Interesting References and resources:

Parsing from Databricks:

- <https://github.com/databricks-industry-solutions/x12-edi-parser>

API based parsing from 3rd party:

- <https://datainsight.health/clinsight/swagger-ui/index.html#/File/fetchFiles>

Tutorials and basics:

- <https://datainsight.health/edi/intro/>