OpenFIDO
Open Framework for Integrated Data Operations

David P. Chassin, Principal Investigator
SLAC National Accelerator Laboratory
Menlo Park, California (USA)

This work was created with funding from the California Energy Commission under grant EPC-17-047.

SLAC National Accelerator Laboratory is operated by Stanford University for the US Department of Energy under Contract DE-AC02-67SF00515.

Contents

Part I

Developer Documents

Chapter 1

OpenFIDO

Openfido

Welcome to the Openfido Organization's repository hub! Here is where you can find the repositories used to build our open-source application, as well as the pipeline's used within the Openfido framework.

Getting Started

There are several options to help new users starting out with OpenFIDO, which is accessible through several distinct methods, or developers looking to help contribute to the project.

Sign up to shared app

You can email support@openfido.org to request to utilize the online application's resources. Once your request to use the application is approved, you will have access to all publicly available repositories that can be automatically searched in this organization, or you can manually designate your own repositories to run.

Host Locally

You can follow the instructions on the OpenFIDO App Service page or the OpenFIDO Command Line Interface to set up locally, and run OpenFIDO to your prefererred method.

Deploy to AWS

You can use this code as a base to deploy your own OpenFIDO AWS servers. While the majority of the terraform modules used to deploy this are available in the centralized OpenFIDO repository, you will need to provide your own code for the cloudfront, rds, ecs, and ses terraform modules, as those are proprietary and licensed to us, and therefore unavailable for use.

Application Status

App Service	Auth Service	Workflow Service	Client Service
Test-build passing	Test-build passing	Test-build passing	Test-build passing
Deploy Staging passing	Deploy Staging passing	Deploy Staging passing	Deploy Staging passing
Deploy Production passing	Deploy Production passing	Deploy Production passing	Deploy Production passing

Pipeline Status

Pipeline	Status
Tariff Design	
Loadshape	
Weather	
HiPAS GridLAB-D	
Census	
Resilience	
Hosting Capacity	
Electrification	
Address	

Chapter 2

OpenFIDO Application Service

OpenFIDO App Service

Summary: A service for the openfido-client, providing organizational access to workflows.

Vocabulary

Architecture Decision Records

- 1. Record architecture decisions
- 2. Project Structure
- 3. Deployment

Development

This service acts as a frontend to both the openfido-workflow-service and the openfido-auth-service, and cannot be usefully run without those services configured and setup locally as well. To do this:

- checkout this repository as well as openfido-workflow-service and openfido-auth-service
- Run all three docker-compose files to bring up the services.
- Reminder: `id_rsa` cannot include `passphrase` when it's generated.

A convenient way to do this is by setting environmental variables telling docker-compose which files to use, and where each project is:

```
export DOCKER_BUILDKIT=1
export COMPOSE_DOCKER_CLI_BUILD=1

# Configure the auth service admin account
cp ../openfido-auth-service/.env.example .auth-env
vi .auth-env

# Because these repositories make use of private github repositories, they
# need access to an SSH key that you have configured for github access:
touch .worker-env
touch ../openfido-auth-service/.env
docker-compose build --build-arg SSH_PRIVATE_KEY="$(cat ~/.ssh/id_rsa)"

# Initialize all the databases for all the services:
docker-compose run --rm auth-service flask db upgrade
docker-compose run --rm workflow-service flask db upgrade
docker-compose run --rm app-service flask db upgrade

# Configure the workflow service access tokens:
docker-compose run --rm workflow-service invoke create-application-key -n "local worker" -p
```

```
docker-compose run --rm workflow-service invoke create-application-key -n "local client" -p

# Obtain the React application key.
# COPY this to openfido-client/src/config/index.js to the API_TOKEN_DEVELOPMENT variable:
docker-compose run --rm app-service invoke create-application-key -n "react client" -p REAC


# Create a super admin user:
docker-compose run --rm auth-service flask shell
from app import models, services
u = services.create_user('admin@example.com', '1234567890', 'admin', 'user')
u.is_system_admin = True
models.db.session.commit()
exit()

# bring up all the services!
docker-compose up
```

Deployment

See openfido terraform docs.

Chapter 3

OpenFIDO Authentication Service

OpenFIDO Auth Service

A reference authentication server implementation, written in Flask.

Configuration

Flask's app.name is used as JWT the 'iss' issuer key. Be sure to configure an identifiable name for your application.

The following environmental variables are exposed:

- `SECRET_KEY`: The Flask secret key variable. Used as the JWT secret.
- `SQLALCHEMY_DATABASE_URI`: Database connection string.
- `SYSTEM_EMAIL`: Email address to use as the 'from address' when emails are sent to users.
- `EMAIL_DRIVER`: Mail server backend implementation. Each backend has its own settings, see the Mail section. Valid options: `null`, and `sendgrid`.

Email integrations

Null: a no-op email integrations. Sends any email to a python logger. To use this implemetation, set the following environmental variables: * `EMAIL_DRIVER`: `null`

Sendgrid: Sendgrid web API integrations. Uses Sendgrid's template system to send emails to users. Additional environmental variables required to use this implementation: * `EMAIL_DRIVER`: `sendgrid` * `SENDGRID_API_KEY`: SendGrid API key. * `SENDGRID_RESET_TEMPLATE_ID`: SendGrid 'reset email' template id of a dynamic template. * `SENDGRID_ORGANIZATION_INVITATION_TEMPLATE_ID`: SendGrid 'invite user' template id of a dynamic template.

Architecture Decision Records

- 1. Record architecture decisions
- 2. Authentication
- 3. Deployment

Development

The local development environment has been set up with docker compose. Once you have docker and docker-compose execute the following commands to setup your local development environment:

```
# Build the docker image, using the SSH private key you use for github
# access (to access other openslac private repositories)
export DOCKER_BUILDKIT=1
export COMPOSE_DOCKER_CLI_BUILD=1
```

```
# Copy the .env.example file into .env
cp .env.example .env

# Login to an docker instance of the flask app:
docker-compose run --rm auth-service bash

# Run database migrations
flask db upgrade

# Create an super admin user:
flask shell
from app import models, services
u = services.create_user('admin@example.com', '1234567890', 'admin', 'user')
u.is_system_admin = True
models.db.session.commit()
```

To start the server locally:

```
# start both the postgres database, and the flask app:
docker-compose up

# visit the app:
http://localhost:5000/
```

To connect to the database:

```
# while docker-compose is running
docker-compose exec db psql -d accountservices -U postgres
```

To connect to a shell in the auth-service container:

```
# while docker-compose is running:
docker-compose exec auth-service bash
```

To connect to a shell in the db container:

```
# while docker-compose is running:
docker-compose exec db bash
```

To run tests, use invoke:

```
# Run within the preconfigured docker instance:
docker-compose run --rm auth-service invoke test

# with code coverage
docker-compose run --rm auth-service invoke test --cov-report && open htmlcov/index.html

# Or if you'd rather run locally
pipenv install
pipenv run invoke test
```

```
# to run a lint test
docker-compose run --rm auth-service invoke lint

# to check code style
docker-compose run --rm auth-service invoke style

# to auto-correct code style errors
docker-compose run --rm auth-service invoke style --fix
```

Other tasks are available, in particular the `precommit` task, which mirrors the tests performed by CircleCI.

Endpoints have been documented with swagger, which is configured to be easily explored in the default `run.py` configuration. When the flask server is running visit http://localhost:5000/apidocs to see documentation and interact with the API directly.

Deployment

See openfido terraform docs.

Chapter 4

OpenFIDO Workflow Service

Workflow Service

Summary: A Flask API server that offers an ability to execute GridLabD jobs and store the resulting artifacts on a file server.

Vocabulary

- Pipeline = a GridLabD job.
- Pipeline Run = An execution of a Pipeline.
- Workflow = A collection of interdependent Pipelines.

Architectural Decision Records

- 1. Record architecture decisions
- 2. Pipelines
- 3. Authentication
- 4. Deployment

Development

The local development environment has been set up with docker compose. Once you have docker and docker-compose execute the following commands to setup your local development environment:

```
# Set up worker configuration:
cp _worker_env.example .worker-env

# Build the docker image, using the SSH private key you use for github
# access (to access other openslac private repositories)
export DOCKER_BUILDKIT=1
export COMPOSE_DOCKER_CLI_BUILD=1
docker-compose build --build-arg SSH_PRIVATE_KEY="$(cat ~/.ssh/id_rsa)"
# TODO at some point docker-compose will support the "--ssh default" docker
# parameter - until then we need to pass the key manually :(

# Login to an docker instance of the flask app:
docker-compose run --rm workflow-service bash

# Run database migrations
flask db upgrade

# Create worker API token and client API token

invoke create-application-key -n "local worker" -p PIPELINES_WORKER | sed 's/^/WORKER_/' > .
invoke create-application-key -n "local worker" -p PIPELINES_CLIENT

# exit the docker instance
```

```
exit
```

To start the server locally:

```
# start both the postgres database, and the flask app:
docker-compose up

# visit the app:
http://localhost:5000/
```

To run tests, use invoke:

```
# Run within the preconfigured docker instance:
docker-compose run --rm workflow-service invoke test

# with code coverage
docker-compose run --rm workflow-service invoke --cov-report test

# Or if you'd rather run locally
pipenv install
pipenv run invoke test
```

Other tasks are available, in particular the `precommit` task, which mirrors the tests performed by CircleCI. See `invoke -l` for a full list of tasks.

The local docker worker will execute jobs, but requires an API key in order to update its status (generated in the instructions above).

Endpoints have been documented with swagger, which is configured to be easily explored in the default `run.py` configuration. When the flask server is running visit http://localhost:5000/apidocs to see documentation and interact with the API directly.

Workers

You can use the `run-worker` invoke task to test repositories. For instance, you can test the anticipation integration by uploading the 'inputs' of the run to an input directory and executing the following command:

```
invoke run-worker $PWD/inputs 'slacgrip/master:200527' https://github.com/PresencePG/grip-ar
```

Configuration

Common settings used by both server and workers:

- **CELERY_BROKER_URL** = Location of the celery broker.
- **S3_ACCESS_KEY_ID** = Access key for uploaded artifacts (optional).
- **S3_SECRET_ACCESS_KEY** = Secret key for uploaded artifacts (optional).
- **S3_ENDPOINT_URL** = Hostname of the S3 service.

- **S3_REGION_NAME** = S3 region (default: us-east-1).
- **S3_BUCKET** = Bucket where uploaded artifacts are kept.

See the constants.py for additional non-configurable options.

Server Configuration

Several environmental variables allow this server to be configured.

- **SECRET_KEY** = See Flask documentation.
- **SQLALCHEMY_DATABASE_URI** = Database connection string.
- **CELERY_BROKER_URL** = Location of the celery broker.
- **CELERY_ALWAYS_EAGER** = When True, execute celery jobs locally. Useful for development/testing purposes.
- **MAX_CONTENT_LENGTH** = Configures maximum upload file byte size.

Worker Configuration

Celery workers only require the following parameters: * **WORKER_API_SERVER** = The Workflow API server. Celery workers require access to this Workflow API in order to update pipeline run states, and upload artifacts. * **WORKER_API_TOKEN** = An application access token to access pipeline run endpoints.

To generate a token that a worker may use to interact with the API, use the following command:

```
invoke create-application-key -n "local worker" -p PIPELINES_WORKER
```

Deployment

See openfido terraform docs.

Chapter 5

OpenFIDO Client

Welcome to the Openfido-client wiki!

Here we will begin to gather all pertinent information on the development and implementation of the Openfido client. As things change, are added, removed, or updated, this wiki should also be appropriately updated to reflect these changes and ensure consistent and easy use of the application.

5.1 Primary Use Cases

5.1.1 Import data processing pipelines

This field will be updated with specifics as development progresses. Current understanding is that requisite docker images and code is pulled from a selected repository and utilized to run the pipeline and process the data. Repository selection is done when add-pipeline is selected, after navigating to the pipelines tab from the left-hand menu. Data can be input manually, or through an automated fill based on a repository's `manifest.json` file (more on that later) and adding a `Pipeline status:` badge to your repository's readme.

5.1.2 Upload configuration files

Different pipelines have different configuration and data file requirements in order to run properly. If you already know what is needed and have the files saved and on-hand, when you select a pipeline and then select start-a-run, you can drag and drop the files for upload into the pipeline. If you do not know what the pipeline needs, but the pipeline has a properly completed `manifest.json` file, a variety of forms will be generated based on the instructions from the manifest. You can then manually fill the required forms to generate designated `.csv`, `.json`, and `.rc` files. If the description field has been properly added, you can also click the question mark to get a quick description of the field and what it is intended to have.

This wiki page is dedicated to running through and building a pipeline, as well as testing out current pipeline features.

TODO

Everything

On this page, we will cover in-depth on how to configure your manifest.json file.

First, you can simply create a file in your pipeline's repository called `manifest.json`.

This file can enable autofill options for selecting your pipeline, if publicly available on github, and can also enable automatic generation of configuration forms for the pipeline, enabling a quick and easy reference to ensure proper use of the pipeline and that all required data is provided.

The structure of the manifest file is important to follow. For a complete example, go to the bottom of this section.

The basic structure

```
{
  "docker" : "debian:11",
  "git" : "https://github.com/openfido/feature-testing.git",
  "branch" : "main",
  "script" : "openfido.sh",
  etc...
}
```

This is the minimum core structure needed to enable complete autofill in the add-pipeline tab of the openfido client. Support will shortly be added to include/prioritize "name" and "description" fields provided in the manifest file, over utilizing the base fields provided by the repository. By properly adding these fields, and by including `Pipeline status:` in your public openfido repository's readme, the openfido client will be able to detect your repository and be able to import the provided pipeline data, ensuring easy and consistent addition of your pipeline.

The manual

```
{
  ...etc,
  "manual": {
    "config": "csv",
    "loadshapes": "csv",
    "gridlabd": "rc",
    "etc": "json"
  },
  ...etc
}
```

The "manual" property in your `manifest.json` file is important in enabling automatic form generation when you are looking to start a new run in the openfido client. This property tells the client what additional properties to look for in the manifest file, and it tells the client what kind of files to generate

from the form. Current supported filetypes are `.csv`, `.rc`, and basic `.json` files. Support may be added later for json files containing nested object trees. Current supported json usage is for a 'dictionary of lists' format with only the surface-level properties used.

Configuring a csv file

```
{
  ...etc
  "config" : {
    "VERBOSE":{
      "input_type": "boolean",
      "description": "Enables verbose output.",
      "default": "true",
      "prompt": "wordly"
    },
    "Input":{
      "input_type": "title"
    },
    "WORKDIR":{
      "input_type": "str",
      "description": "Specifies the working directory.",
      "default": "/tmp"
    },
    "INPUT_CSV":{
      "input_type": "str required",
      "description": "Specifies AMI input data file (REQUIRED).",
      "default": ""
    },
    "Analysis":{
      "input_type": "title"
    },
    "RESAMPLE":{
      "input_type": "str optional",
      "description": "Specifies resample method to use. Valid methods include all DataFrame aggre
      "default": ""
    },
    "GROUP_COUNT":{
      "input_type": "int required",
      "description": "Grouping count. Must be a positive number (REQUIRED).",
      "default": "0"
    },
    "Outputs":{
      "input_type": "title"
    },
```

```
    "LOADSHAPES_CSV":{
      "input_type": "str",
      "description": "Specifies the loadshape file to generate.",
      "default": "loadshapes.csv"
    },
    "LOAD_SCALE":{
      "input_type": "float",
      "description": "Specifies the scaling of the schedule data to load (e.g., 1kVA=1(
      "default": "1000"
    },
    "UPLOAD_COMMA_3":{
      "input_type": "upload",
      "description": "might be able to upload",
      "default": "upload.csv",
      "prompt": "testing upload comma 3",
      "upload_max": 3,
      "space_delimited": false
    }
  },
  ...etc
}
```

If you designated "config" in one of your manual properties, the openfido client will then look for and generate a form based on the "config" property located on the surface of the manifest.json file. The client builds all forms in **DIRECT** order, from top to bottom. **DO NOT** use numbers for the fields, as they do not follow the same pattern and can create their fields out-of-order. Each key within the config property is also the default field-name when the form is generated, unless a "prompt" field is included to overwrite using the key for the field name.

The "description" key, if included, will generate a tooltip that can be displayed by clicking the question-mark to the right of each form field. If left empty, an empty tooltip will be generated instead.

The "default" key, if included, will generate a default value in the form field. It is particularly useful if certain settings are consistently used, and can easily be deleted on the form if a different value is to be used for a particular run.

The "prompt" key, if included, is used for displaying a custom field-name. The default field-name is the property name. The property name needs to match the expected input name from the CSV. Using prompt will not change the generated CSV file, and is only for updating the display on the generated form.

The "input_type" field is required, as it designates what kind of input to expect for a given field, and can be used to enforce field validation (once implemented). Valid entries for "input_type" are:

```
{
```

```
'boolean'          ===> Generates a checkable field, where checked is true and unchecked is false
'str'              ===> Generates a field that takes any string as input.
'str optional'     ===> For use in optional fields. Can be helpful for maintaining the manifest f
'str required'     ===> Will enforce using the field, and highlights the field if left empty.
'float'            ===> Generates a field that takes only integers as inputs.
'int'              ===> Generates a field that takes only integers as inputs.
'int optional'     ===> For use in optional fields. Can be helpful for maintaining the manifest f
'int required'     ===> Will enforce using the field, and highlights the field if left empty. The
'enum'             ===> Generates a select-one field which will always have a value
'set'              ===> Generates a multi-select field, where you can select all values which app
'upload'           ===> Generates a direct upload field,  which can set the name manually, or upd
'upload required'  ===> Will enforce using the field, and highlights the field if left empty.
'title'            ===> Generates a title field
}
```

float and **int** are interchangeable. 'float' is currently included due to heavy use of python in a variety of projects, but means the same thing as 'int' in JavaScript.

Also worth noting is that upload fields can take two additional configuration properties, "upload_max" and "space_delimited". These properties can limit the total number of uploads a field will accept, and designate whether multiple-file uploads will be separated by a comma or space. If left out, the field will default to unlimited uploads and comma-separation.

Configuring a rc file

```
{
  ...etc
  "gridlabd": {
    "COMMAND_1":{
      "description": "Adds the -D command",
      "default": "-D csv_load_options=\"-f config\""
    }
    },
  ...etc
}
```

The `.rc` field is very simple and easy to configure, as a rc file is simply a specific kind of text file that contains an ordered list of single-line commands. "input_type" is only necessary for "upload" and "upload required" fields, as all other inputs will be text fields. Simply order the fields in the order you want the commands executed, from top to bottom, and it will generate a file in that same order.

Configuring a JSON file

```
{
```

```
    ...etc
    "etc": {
      "Json": {
        "input_type": "arr",
        "default": "item 1, item 2, item 3",
        "prompt": "hmm"
      },
      "application": {
        "input_type": "str",
        "default": "openfido"
      },
      "version": {
        "input_type": "int",
        "default": 0
      },
      "valid": {
        "input_type": "boolean",
        "default": true
      },
      "inputs": {
        "input_type": "int",
        "default": -1
      }
    }
}
```

Current JSON support is for surface-level keys in a 'dictionary of lists' style. It operates similarly to the csv form, with a couple key distinctions.

First, it actively preserves the input-type. For example, a boolean input of true retains its type as a boolean, without any additional work needing to be done.

Second, it can accept an array as an "input_type". Assigning "arr" to the "input_type" property will generate a text field that will split items into array indexes based on comma separation. For example, "item 1, item 2, item 3" will produce ["item 1", ""item 2,"item 3"]. See the complete example below.

Complete manifest.json Example

```
{
  "name" : "Feature Test",
  "description" : "Feature testing faked pipeline",
  "docker" : "debian:11",
  "git" : "https://github.com/openfido/feature-testing.git",
  "branch" : "main",
  "script" : "openfido.sh",
  "manual": {
```

```
    "config": "csv",
    "loadshapes": "csv",
    "gridlabd": "rc",
    "etc": "json"
  },
  "config" : {
    "VERBOSE":{
      "input_type": "boolean",
      "description": "Enables verbose output.",
      "default": "true",
      "prompt": "wordly"
    },
    "Input":{
      "input_type": "title"
    },
    "INPUT_CSV":{
      "input_type": "str required",
      "description": "Specifies AMI input data file (REQUIRED).",
      "default": ""
    },
    "Analysis":{
      "input_type": "title"
    },
    "FILL_METHOD":{
      "input_type": "str optional",
      "description": "Specifies the fill method for missing data. Valid values are 'bfill', 'back
      "default": ""
    },
    "AGGREGATION":{
      "input_type": "str",
      "description": "Group aggregation method. Valid methods include all DataFrame aggregators.'
      "default": "median"
    },
    "GROUP_COUNT":{
      "input_type": "int required",
      "description": "Grouping count. Must be a positive number (REQUIRED).",
      "default": "0"
    },
    "GridLAB-D":{
      "input_type": "title"
    },
    "LOAD_SCALE":{
      "input_type": "float",
      "description": "Specifies the scaling of the schedule data to load (e.g., 1kVA=1000VA)",
      "default": "1000"
    },
```

```
    "ENUM_TEST":{
      "input_type": "enum",
      "description": "tests whether enum works",
      "default": "works",
      "choices": "tests, whether, enum, works"
    },
    "SET_TEST":{
      "input_type": "set",
      "description": "tests whether set works",
      "default": "whether, set",
      "choices": "tests, whether, set, works"
    }
  },
  "loadshapes": {
    "UPLOAD_1":{
      "input_type": "upload",
      "description": "might be able to upload",
      "default": "upload.csv",
      "prompt": "testing upload one",
      "upload_max": 1
    },
    "UPLOAD_SPACE":{
      "input_type": "upload",
      "description": "might be able to upload",
      "default": "upload.csv",
      "prompt": "testing upload space",
      "space_delimited": true
    },
    "UPLOAD_COMMA_3":{
      "input_type": "upload",
      "description": "might be able to upload",
      "default": "upload.csv",
      "prompt": "testing upload comma 3",
      "upload_max": 3,
      "space_delimited": false
    }
  },
  "gridlabd": {
    "COMMAND_1":{
      "description": "Adds the -D command",
      "default": "-D csv_load_options=\"-f config\""
    },
    "CONFIG":{
      "description": "Adds the config file requirement",
      "default": "config.csv"
    },
```

```
    "UPLOAD_1":{
      "description": "Requires the clock.glm file",
      "default": "clock.glm"
    },
    "UPLOAD_2":{
      "description": "Another way of requiring multiple files in one field",
      "default": "weather.glm test_R1-12.47-1.glm"
    },
    "DATA_1":{
      "description": "Adds the other -D command",
      "default": "-D minimum_timestep=3600",
      "prompt": "promptly"
    }
    },
    "etc": {
      "Json": {
        "input_type": "arr",
        "default": "item 1, item 2, item 3",
        "prompt": "hmm"
      },
      "application": {
        "input_type": "str",
        "default": "openfido"
      },
      "version": {
        "input_type": "int",
        "default": 0
      },
      "tooltype": {
        "input_type": "str",
        "default": "pipeline"
      },
      "valid": {
        "input_type": "boolean",
        "default": true
      },
      "inputs": {
        "input_type": "int",
        "default": -1
      },
      "outputs": {
        "input_type": "int",
        "default": 0
      }
    }
}
```

This project was bootstrapped with Create React App.

OpenFIDO Environments

Getting Started

`npm install`

For the first-time setup, make sure to run `npm install` to build the repo's dependencies.

Create the .env file

To protect sensitive information, the .env file is NOT provided for you. Rename the example.env.txt file to .env and fill in the appropriate variables for your use case.

Follow the step-by-step instructions in the (https://github.com/slacgismo/openfido-app-service) repository to get the REACT_APP_API_TOKEN, and follow the step-by-step instructions at (https://docs.github.com/en/developers/apps/building-oauth-apps/creating-an-oauth-app) to create a personal github Oauth app. Use your app's secret for the DEV_GAPP_SECRET.

Available Scripts

In the project directory, you can run:

`npm start`

Runs the app in the development mode. Open http://localhost:3000 to view it in the browser.

The page will reload if you make edits. You will also see any lint errors in the console.

`npm test`

Launches the test runner in the interactive watch mode. See the section about running tests for more information.

`npm run build`

Builds the app for production to the `build` folder. It correctly bundles React in production mode and optimizes the build for the best performance.

The build is minified and the filenames include the hashes. Your app is ready to be deployed!

See the section about deployment for more information.

`npm run eject`

Note: this is a one-way operation. Once you `eject`, you can't go back!

If you aren't satisfied with the build tool and configuration choices, you can `eject` at any time. This command will remove the single build dependency from your project.

Instead, it will copy all the configuration files and the transitive dependencies (webpack, Babel, ESLint, etc) right into your project so you have full control over them. All of the commands except `eject` will still work, but they will point to the copied scripts so you can tweak them. At this point you're on your own.

You don't have to ever use `eject`. The curated feature set is suitable for small and middle deployments, and you shouldn't feel obligated to use this feature. However we understand that this tool wouldn't be useful if you couldn't customize it when you are ready for it.

Learn More

You can learn more in the Create React App documentation.

To learn React, check out the React documentation.

Code Splitting

This section has moved here: https://facebook.github.io/create-react-app/docs/code-splitting

Analyzing the Bundle Size

This section has moved here: https://facebook.github.io/create-react-app/docs/analyzing-the-bundle-size

Making a Progressive Web App

This section has moved here: https://facebook.github.io/create-react-app/docs/making-a-progressive-web-app

Advanced Configuration

This section has moved here: https://facebook.github.io/create-react-app/docs/advanced-configuration

Deployment

This section has moved here: https://facebook.github.io/create-react-app/docs/deployment

`npm run build` fails to minify

This section has moved here: https://facebook.github.io/create-react-app/docs/troubleshooting#npm-run-build-fails-to-minify

Part II

Pipeline User Documents

Chapter 6

Tariff Design Pipeline

Pipeline status: 

OpenFIDO Tariff Design pipeline

The `tariff_design` pipeline runs GridLAB-D tariff_design template.

PIPELINE

Recommended pipeline settings:

Setting	Recommended value
Pipeline name	Tariff_Design
Description	Run GridLAB-D
DockerHub Repository	slacgismo/gridlabd:develop
Git Clone URL (https)	https://github.com/openfido/tariff_design
Repository Branch	main
Entrypoint Script (.sh)	openfido.sh

INPUTS

The configuration file `config.csv` must be uploaded with the first row as `Header, Value`. The following parameters are recognized in `config.csv`: * `WEATHER_STATION`: Specifies the weather station for the tariff simulation environment. No default. * `STARTTIME`: Specifies the starting time for the tariff simulation. Recommended to use ISO8601 format. No default (subject to change). * `STOPTIME`: Specifies the ending time for the tariff simulation. Recommended to use ISO8601 format. No default (subject to change). * `TIMEZONE`: Specifies time zone of `STARTTIME` and `STOPTIME`. Recommended to use ISO8601 format. No default (subject to change). * `MODEL`: Specifies name of model for tariff simulation. File of the same name must be provided as input. Optional, default is `model.glm`. * `OUTPUT`: Specifies name of output file to store results of tariff simulation. Optional, default is `output.csv`. * `TARIFF_UTILITY`: Specifies utility company name. Values must be `Pacific Gas & Electric Co`, `San Diego Gas & Electric Co`, or `Southern California Edison Co`. If value not provided, will attempt simulation using `TARIFF_NAME` and `TARIFF_REGION`. * `TARIFF_NAME`: Specifies tariff name. Values must be `E-TOU-C3`, `E-7 Residential Time of Use Baseline`, `E-1`, `E-7`, `E-6`, `DR`, `EV-TOU-2`, `TOU-D-B`, or `TOU-D-TEV`. If value not provided, will attempt simulation using `TARIFF_UTILITY` and `TARIFF_REGION`. * `TARIFF_REGION`: Specifies tariff region. Values must be `Region R`, `REGION P`, `REGION T`, `REGION Z`, `MOUTAIN BASELINE REGION`, `REGION 15`. * `TARIFF_INDEX_SPECIFIC`: Some tariffs need extra information to simulate. When encountered, provide this field with a corresponding value specified by the error message.

The `model.glm` file also requires various definitions and module declarations currently:

```
module powerflow;
module residential;

#input "config.csv" -f config -t config

#define tariff_index=${TARIFF_INDEX}

clock {
    timezone ${TIMEZONE};
    starttime ${STARTTIME};
    stoptime ${STOPTIME};
}

#input "${WEATHER_STATION}.tmy3"
```

An example of a complete `model.glm` file is shown below:

```
module powerflow;
module residential;

#input "config.csv" -f config -t config

#define tariff_index=${TARIFF_INDEX}

clock {
    timezone ${TIMEZONE};
    starttime ${STARTTIME};
    stoptime ${STOPTIME};
}

#input "${WEATHER_STATION}.tmy3"

#define PRIMARY_VOLTAGE=4800V
#define POWER_RATING=500
#define RESISTANCE=0.011
#define REACTANCE=0.02

class meter
{
    string monthly_charges;
    string monthly_usage;
    string monthly_power;
    double monthly_updated_charges[$];
    double monthly_updated_usage[kWh];
```

```
    double monthly_updated_power[W];
}

object meter
{
    bustype "SWING";
    name "meter_1";
    nominal_voltage "4800V";
    phases "ABCN";
}

object transformer_configuration {
    name "transformer_type1";
    connect_type "SINGLE_PHASE_CENTER_TAPPED";
    install_type "PADMOUNT";
    power_rating ${POWER_RATING};
    primary_voltage ${PRIMARY_VOLTAGE};
    secondary_voltage "120V";
    resistance ${RESISTANCE};
    reactance ${REACTANCE};
}

#for ID in ${RANGE 1, 20}

object transformer {
    name transformer_${ID};
    phases "AS";
    from "meter_1";
    to "submeter_${ID}";
    configuration "transformer_type1";
}

object triplex_meter
{
    name "submeter_${ID}";
    nominal_voltage "120V";
    phases "AS";
    object house
    {
        floor_area random.triangle(1000,2000);
        thermal_integrity_level "NORMAL";
        gas_enduses "WATERHEATER|DRYER|RANGE";
        heating_system_type "HEAT_PUMP";
    };
}
```

`#done`

Below is an example of `config.csv`:

Header	Value
WEATHER_STATION	CA-San_Francisco_Intl_Ap
STARTTIME	2020-01-01T00:00:00-00:00
STOPTIME	2021-01-15T00:00:00-12:00
TIMEZONE	PST+8PDT
MODEL	model.glm
OUTPUT	output.csv
TARIFF_UTILITY	Pacific Gas & Electric Co
TARIFF_NAME	E-TOU-C3
TARIFF_REGION	Region R

An optional `clock.glm` file can also be uploaded containing a clock object. The clock object must have the following properties: * `STARTTIME`: Specifies the starting time for the tariff simulation. Recommended to use ISO8601 format. No default (subject to change). * `STOPTIME`: Specifies the ending time for the tariff simulation. Recommended to use ISO8601 format. No default (subject to change). * `TIMEZONE`: Specifies time zone of `STARTTIME` and `STOPTIME`. Recommended to use ISO8601 format. No default (subject to change). Note that the same values in `config.csv` must still be provided. However, the `clock.glm` values will be used. Below is an example `clock.glm`:

```
clock {
    timezone "PST+8PDT";
    starttime "2020-12-08 16:00:00 PST";
    stoptime "2021-1-09 12:00:00 PST";
}
```

OUTPUTS

`output.csv` or the name specified in `OUTPUT` of `config.csv` is generated in the output folder. It will contain the following data by column: * `Meter_ID`: The name of the meter as the index. * `Date`: The date that row results are sampled. * `Days`: The number of days the row results accumulated. * `Cost ($)`: The amount incurred based on the configured simulation. * `Energy (kWh)`: The electricty consumption. * `Peak Power (W)`: The measured demand during simulation duration.

Three bargraphs (.png) are generated in the output folder for each meter: one for `Cost ($)`, `Energy (kWh)`, and `Peak Power (W)`. The values of each meter for each month during the simulation duration will be plotted.

Three histograms (.png) are generated in the output folder, plotting the dis-

tribution of `Cost ($)`, `Energy (kWh)`, and `Peak Power (W)` across all triplex meters.

SEE ALSO

- https://docs.gridlabd.us/_page.html?&doc=/GLM/Directive/Clock.md
- https://github.com/slacgismo/gridlabd-template/edit/add-tariff-design/US/CA/SLAC/tariff_design/tariff_design.md

Chapter 7

Loadshape Pipeline

Pipeline status: 

OpenFIDO loadshape pipeline

The loadshape pipeline analyses AMI data and generates the most common loadshapes present. Hourly loadshapes are generated for each season weekday and weekend. The AMI data is then grouped using the specified group method (by default K-Means Clustering).

The loadshape data may be optionally output to GLM files so that loads can be attached to network models.

PIPELINE

Recommended pipeline settings:

Setting	Recommended value
Pipeline name	Loadshape
Description	AMI loadshape analysis and generation
DockerHub Repository	debian:11
Git Clone URL (https)	https://github.com/openfido/loadshape
Repository Branch	main
Entrypoint Script (.sh)	openfido.sh

INPUTS

** Required inputs **

`config.csv` - The run configuration file is required (see CONFIGURATION below).

AMI data - The AMI data (required) as a CSV file (may be compressed). The name of this file must be specified in the `config.csv` using the `INPUT_CSV` parameter. The required columns include:

Column	Content
0	Date and time
1	Meter ID
2	Interval energy measurement
3	Timezone specification

The date/time column may be specified in UTC or local date/time. The name of the date/time column may be specified using the `DATETIME_COLUMN` configuration parameter in `config.csv`. The format of the date/time column is given by the

DATETIME_FORMAT configuration parameter in config.csv.

The meter id column may contain any valid unique string identifier. The name of the meter id column may be specified using the ID_COLUMN configuration parameter in config.csv.

The interval energy is measured in units of energy per hour in kWh/h. The name of the interval energy measurement may be changed using the DATA_COLUMN configuration parameter in config.csv.

The timezone specification is given in hours offset relative to UTC, i.e., east is positive and west is negative. The name of the timezone specification column may be changed using the TIMEZONE_COLUMN configuraiton parameter in config.csv. If this parameter is set to an empty string, the timezone is set to UTC. If UTC is used, then the timezone must be specified as UTC offset, with DST shifts, if any, e.g., -8 for PST, and -7 for PDT. If local time is used, then the timezone should specify only the DST shift, i.e., 0 for standard, and 1 for summer time. If DST is not use and the date/time data is local, the timezone column may be omitted. At this time only Atlantic (AST/ADT), Eastern (EST/EDT), Central (CST/CDT), Mountain (MST/MDT), Pacific (PST/PDT), Alaska (AKST/AKDT), and Hawaii (HST/HDT) timezones are supported.

** Optional Inputs **

Load map - An optional CSV file containing the mapping of loads to the network model. The name of this file may be specified using the LOADS_CSV parameter in config.csv. Required columns correspond to GridLAB-D load object properties:

Property	Description
meter_id	The meter id from the AMI data
class	The object class (load or triplex_load
parent	The parent object ID (a valid network node name)
phases	The load phases (must match network node)
nominal_voltage	The load nominal voltage (must match network node)

Property	Description
`{power,current,impedance}_fraction_[ABC]`	The ZIP powerflow `load` fractions (only for 1, 2, or 3-phase non-split loads)
`{power,current,impedance}_fraction_{1,2,12}`	The ZIP powerflow `triplex_load` fractions (only for single-phase split-tap loads)

If the fractions are omitted, the ZIP load is set to a unitary constant power fraction. All other columns are copied to the loads verbatim.

OUTPUTS

Data files (always output)

Loadshapes - The loadshapes are saved to the CSV file specified by the `LOADSHAPES_CSV` configuration parameter in `config.csv`. The default filename if the parameter is not specified is `loadshapes.csv`. The rows identify each loadshape group, and the columns provide the load for each hour, daytype, and season in that loadshape. The column names use abbreviations for season and day type, i.e., season in {`win`,`spr`,`sum`,`fal`} and day type in {`wd`,`we`}, which a concatenated with the hour of day, e.g., `win_wd_0` for hour 0 of a winter weekday.

Groups - The groups are saved to the CSV file specifies by the `GROUPS_CSV` configuration parameter in `config.csv`. The default filename if the parameter is not specified is `groups.csv`. The rows identify each meter specified by the `ID_COLUMN` field in the input AMI data.

GridLAB-D model (optional output)

Clock - This model fragement contains the data range and timezone specification based on input the AMI data. This file is generated only when `CLOCK_GLM` is specified in `config.csv`.

Schedules - This model fragment contains the loadshape data generated as GridLAB-D schedules. This file is generated only when `SCHEDULES_GLM` is specified in `config.csv`.

Loads - This model fragment contains the load objects generated using scaled references to schedules. This file is generated only `LOADS_GLM` is specified in `config.csv`. In addition, the file specified by `INPUT_MAP` is required to identify how loads are mapped to the network model.

Plots (optional output)

If `OUTPUT_PNG` is specified in `config.csv`, then a plot containing the load-shapes and underlying AMI data is generated. The `config.csv` parameters `PNG_FIGSIZE` and `PNG_FONTSIZE` control figure size (in inch) and font size (in points), respectively. If omitted the defaults `10x7` and `14`, respectively.

CONFIGURATION

The following configuration parameters are supported

Parameter	Default	Description
`VERBOSE`	`True`	Enables verbose output.
`DEBUG`	`True`	Enables debug output.
`QUIET`	`False`	Disables all output.
`WARNING`	`True`	Enables warning output.
Input		
`WORKDIR`	`/tmp`	Specifies the working directory.
`INPUT_CSV`	None	Specifies AMI input data file (REQUIRED).
`DATETIME_COLUMN`	`0`	Specifies the date/time column in the AMI file.
`ID_COLUMN`	`1`	Specifies the id column in the AMI file.
`DATA_COLUMN`	`2`	Specifies the data column in the AMI file.
`TIMEZONE_COLUMN`	`3`	Specifies the timezone column in the AMI file.

Parameter	Default	Description
DATETIME_FORMAT	%Y-%m-%d %H:%M:%S	Specifies the input date/time format.
Analysis		
GROUP_METHOD	kmeans	Grouping method. Valid method is 'kmeans'.
GROUP_COUNT	0	Grouping count. Must be a positive number (REQUIRED).
Outputs		
LOADSHAPES_CSV	loadshapes.csv	Specifies the loadshape file to generate.
GROUPS_CSV	groups.csv	Specifies the group file to generate.
FLOAT_FORMAT	%.4g	Specifies float data format.
ARCHIVE_FILE	None	Specifies the name TAR file to use (trailing z for compressed)
Plotting		
OUTPUT_PNG	None	Specifies the output PNG file name.
PNG_FIGSIZE	10x7	Specifies the output PNG image size (in inches)
PNG_FONTSIZE	14	Specifies the output PNG image font size (in points)
GridLAB-D		
LOADS_CSV	None	Specifies the load mapping file.

Parameter	Default	Description
CLOCK_GLM	None	Specifies the output GLM clock model fragment.
LOADS_GLM	None	Specifies the output GLM load model fragment.
SCHEDULES_GLM	None	Specifies the output GLM schedule model fragment.
LOAD_SCALE	1000	Specifies the scaling of the schedule data to load (e.g., 1kVA=1000VA)

ENVIRONMENT

- OPENFIDO_INPUT specifies the input folder.
- OPENFIDO_OUTPUT specifies the output folder.
- PWD specifies the default working folder.

EXIT CODES

The following exit codes are used:

Exit code	Condition
0	Load shape analysis completed ok
1	A fatal exception was detected
2	An invalid input was received
3	The loadshape analysis failed

In the event of a non-zero exit code, the DEBUG and VERBOSE configuration options in config.csv may be used to obtain additional information on where and why the condition occured.

Chapter 8

Weather Pipeline

Pipeline status: 

OpenFIDO weather pipeline

The weather pipeline collates weather data for a location and set of years.

If only the *CSVFILE* is specified, then the CSV output includes a header row. If the *GLMFILE* is also specified, then the CSV output does not include a header row and the column names are identified in the GLM weather object.

PIPELINE

Recommended pipeline settings:

Setting	Recommended value
Pipeline name	Weather
Description	NSRDB historical weather data downloader
DockerHub Repository	debian:11
Git Clone URL (https)	https://github.com/openfido/weather.
Repository Branch	main
Entrypoint Script (.sh)	openfido.sh

INPUTS

`config.csv` - The run configuration file is required (see CONFIGURATION below).

OUTPUTS

CSVFILE - Must be specified in `config.csv`. The following columns are provided:

Column name	Description
datetime	`YYYY-MM-DD HH:MM:SS` if CSV only, seconds in epoch w/GLM
solar_global[W/sf]	Global solar irradiance
solar_horizontal[W/sf]	Horizontal surface solar irradiance
solar_direct[W/sf]	Direct normal solar irradiance
clouds	Cloud type from NOAA PATMOS-X (see below)
dewpoint[degF]	Wet bulb temperature
temperature[degF]	Dry bulb temperature
ground_reflectivity[pu]	Ground albedo
wind_speed[m/s]	Wind speed
wind_dir[rad]	Wind direction (compass heading in radians

Column name	Description
solar_altitude[deg]	Solar altitude.
humidity[%]	Relative humidity
pressure[mbar]	Air pressure
heat_index[degF]	Heat index temperature (NOAA method)

Cloud type	Definition.
0	Clear
1	Probably clear
2	Fog
3	Water
4	Super-cooled water
5	Mixed
6	Opaque ice
7	Cirrus
8	Overlapping
9	Overshooting
10	Unknown
11	Dust
12	Smoke
-15	Not available

GLMFILE - Only if specified in `config.csv`. The model file includes the global "WEATHER", which enumerates the weather object name included. The model will always include the class definition for a weather object with the weather properties defined above.

CONFIGURATION

The following is a summary of parameters that are supported by the config.csv file.

Template for `config.csv`:

```
CSVFILE,filename.csv
GLMFILE,filename.glm
NAME,objectname
EMAIL,your.email@your.org
APIKEY,your-api-key
YEARS,year1,year2,...
LATLON,latitude,longitude
```

CSVFILE - Specifies the weather CSV output file name. Required.

GLMFILE - Specifies the gridlabd GLM output model file. Optional. If omitted, the CSV file is formatted for standalone use (with a header row). If included, the CSV file is formatted for GridLAB-D player input (without a header row).

NAME - Specifies the GLM weather object name to use. Optional. If omitted, the object will be given a geocoded name based on the latitude and longitude of the weather location.

EMAIL - Specifies the email address used to register with the NREL NSRDB API. See https://nsrdb.nrel.gov/data-sets/api-instructions.html for details.

APIKEY - Provides the API key provided by NREL when the EMAIL was registered. See EMAIL for details.

YEARS - Specifies the years for which weather data is downloaded. Multiple years are permitted by adding additional columns

LATLON - Specifies the latitude and longitude for the weather data.

You may use the http://openfido.gridlabd.us/weather.html to generate the configuration file.

Chapter 9

GridLAB-D Pipeline

Pipeline status: 

OpenFIDO gridlabd pipeline

The `gridlabd` pipeline runs GridLAB-D.

PIPELINE

Recommended pipeline settings:

Setting	Recommended value
Pipeline name	GridLAB-D
Description	Run GridLAB-D
DockerHub Repository	slacgismo/gridlabd:latest
Git Clone URL (https)	https://github.com/openfido/gridlabd
Repository Branch	main
Entrypoint Script (.sh)	openfido.sh

INPUTS

The gridlabd run file `gridlabd.rc` must be uploaded. The format of the RC file is one line per gridlabd command option, e.g., the command `gridlabd --verbose model.glm` should be entered as

```
--verbose
model.glm
```

Options may be provided together, e.g., `gridlabd -D csv_load_options="-f config" config.csv network.glm equipment.glm` should be entered as

```
-D csv_load_options="-f config"
config.csv
network.glm equipment.glm
```

All files referenced by the RC file, or by any file contained in the RC file must be uploaded as well.

OUTPUTS

All files at the end of the run will be copied to the output folder, including the original input files.

CAVEATS

1. Unlike most OpenFIDO pipelines, this pipeline runs in the input folder.

2. All normal output is copied to `stdout` and errors to `stderr`. If you wish to capture the output stream is file, add the desired `--redirect` options in `gridlabd.rc`, e.g.,

```
--redirect all
```

to redirect all output to files names `gridlabd.*`.

3. A number of `gridlabd` capabilities require credentials. These may not work unless the credentials are uploaded to the input files.

Chapter 10

Census Pipeline

Pipeline status: 

Census TIGER geographic data retrieval.

CONFIG

The following parameters are recognized in `config.csv`:

- `INPUT_FILENAME`: Specifies the input filename. Optional, default is `input.csv`.
- `OUTPUT_FILENAME`: Specifies the output filename. Optional, default is `output.csv`.
- `STATE_FIELDS`: Specifies the state fields to retrieve. Valid fields are `REGION`, `DIVISION`, `STATEFP`, `STATENS`, `GEOID`, `STUSPS`, `NAME`, `LSAD`, `MTFCC`, `FUNCSTAT`, `ALAND`, `AWATER`, `INTPTLAT`, `INTPTLON`, `geometry`. Optional, default is `*`.
- `ZIPCODE_FIELDS`: Specifies the zipcode fields to retrieve. `ZCTA5CE10`, `GEOID10`, `CLASSFP10`, `MTFCC10`, `FUNCSTAT10`, `ALAND10`, `AWATER10`, `INTPTLAT10`, `INTPTLON10`, `geometry`. Optional, default is `*`.

INPUT

The following files are accepted as input folder.

- `input.csv`: The input file must provide the `latitude` and `longitude` fields.

OUTPUT

The following files are generated in the output folder.

- `output.csv`: The output file will include the requested state and zipcode fields, as well as all the input fields.

SEE ALSO

- https://www.census.gov/programs-surveys/geography/guidance/tiger-data-products-guide.html

Chapter 11

Resilience Pipeline

Pipeline status: 

OpenFIDO Grid Resilience Anticipation pipeline

The `resilience` pipeline runs GridLAB-D resilience analysis template.

PIPELINE

Recommended pipeline settings:

Setting	Recommended value
Pipeline name	GridLAB-D
Description	Run GridLAB-D
DockerHub Repository	slacgismo/gridlabd:latest
Git Clone URL (https)	https://github.com/openfido/resilience
Repository Branch	main
Entrypoint Script (.sh)	openfido.sh

INPUTS

The gridlabd run file `gridlabd.rc` must be uploaded. The format of the RC file is one line per gridlabd command option, e.g., the command `gridlabd --verbose model.glm` should be entered as

```
--verbose
model.glm
```

Options may be provided together, e.g., `gridlabd -D csv_load_options="-f config" config.csv network.glm equipment.glm` should be entered as

```
-D csv_load_options="-f config"
config.csv
network.glm equipment.glm
```

All files referenced by the RC file, or by any file contained in the RC file must be uploaded as well.

OUTPUTS

All files at the end of the run will be copied to the output folder, including the original input files.

CAVEATS

1. Unlike most OpenFIDO pipelines, this pipeline runs in the input folder.

2. All normal output is copied to `stdout` and errors to `stderr`. If you wish to capture the output stream is file, add the desired `--redirect` options in `gridlabd.rc`, e.g.,

```
--redirect all
```

to redirect all output to files names `gridlabd.*`.

3. A number of `gridlabd` capabilities require credentials. These may not work unless the credentials are uploaded to the input files.

Chapter 12

Hosting Capacity Pipeline

Pipeline status: 

OpenFIDO Hosting Capacity Analysis pipeline

The `hosting_capacity` pipeline runs GridLAB-D ICA analysis template.

PIPELINE

Recommended pipeline settings:

Setting	Recommended value
Pipeline name	GridLAB-D
Description	Run GridLAB-D
DockerHub Repository	slacgismo/gridlabd:latest
Git Clone URL (https)	https://github.com/openfido/hosting_capacity
Repository Branch	main
Entrypoint Script (.sh)	openfido.sh

INPUTS

The gridlabd run file `gridlabd.rc` must be uploaded. The format of the RC file is one line per gridlabd command option, e.g., the command `gridlabd --verbose model.glm` should be entered as

```
--verbose
model.glm
```

Options may be provided together, e.g., `gridlabd -D csv_load_options="-f config" config.csv network.glm equipment.glm` should be entered as

```
-D csv_load_options="-f config"
config.csv
network.glm equipment.glm
```

All files referenced by the RC file, or by any file contained in the RC file must be uploaded as well.

OUTPUTS

All files at the end of the run will be copied to the output folder, including the original input files.

CAVEATS

1. Unlike most OpenFIDO pipelines, this pipeline runs in the input folder.

2. All normal output is copied to `stdout` and errors to `stderr`. If you wish to capture the output stream is file, add the desired `--redirect` options in `gridlabd.rc`, e.g.,

```
--redirect all
```

to redirect all output to files names `gridlabd.*`.

3. A number of `gridlabd` capabilities require credentials. These may not work unless the credentials are uploaded to the input files.

Chapter 13

Electrification Pipeline

Pipeline status: 

OpenFIDO Electrification Analysis pipeline

The `electrification` pipeline runs GridLAB-D ICA analysis template.

PIPELINE

Recommended pipeline settings:

Setting	Recommended value
Pipeline name	GridLAB-D
Description	Run GridLAB-D
DockerHub Repository	slacgismo/gridlabd:latest
Git Clone URL (https)	https://github.com/openfido/electrification
Repository Branch	main
Entrypoint Script (.sh)	openfido.sh

INPUTS

The gridlabd run file `gridlabd.rc` must be uploaded. The format of the RC file is one line per gridlabd command option, e.g., the command `gridlabd --verbose model.glm` should be entered as

```
--verbose
model.glm
```

Options may be provided together, e.g., `gridlabd -D csv_load_options="-f config" config.csv network.glm equipment.glm` should be entered as

```
-D csv_load_options="-f config"
config.csv
network.glm equipment.glm
```

All files referenced by the RC file, or by any file contained in the RC file must be uploaded as well.

OUTPUTS

All files at the end of the run will be copied to the output folder, including the original input files.

CAVEATS

1. Unlike most OpenFIDO pipelines, this pipeline runs in the input folder.

2. All normal output is copied to `stdout` and errors to `stderr`. If you wish to capture the output stream is file, add the desired `--redirect` options in `gridlabd.rc`, e.g.,

```
--redirect all
```

to redirect all output to files names `gridlabd.*`.

3. A number of `gridlabd` capabilities require credentials. These may not work unless the credentials are uploaded to the input files.

Chapter 14

Address Pipeline

Pipeline status: 

The address resolution pipeline resolves addresses and locations.

CONFIG

The following parameters are recognized in `config.csv`:

- `DATA`: The CSV data file name contain address or location (required)
- `REVERSE`: Boolean value indicating whether the data contains locations (False, default) or addresses (True)
- `PROVIDER`: Resolver provider (default is `nominatim`)
- `USER_AGENT`: Resolver user agent (default is `csv_user_ht`)
- `TIMEOUT`: Resolver timeout in seconds (default is `5`)
- `RETRIES`: Resolver retry limit (default is `5`)
- `SLEEP`: Resolver sleep between retries in seconds (default `1`)

INPUT

The format of the input depends on the value of REVERSE in `config.csv`.

- `REVERSE=False`

The CSV file must include a `latitude` and `longitude` column.

- `REVERSE=True`

The CSV file must include an `address` column.

OUTPUT

The format of the output depends on the value of `REVERSE` in config.csv.

- `REVERSE=False`

The CSV file will include an `address` column.

- `REVERSE=True`

The CSV file will include a `latitude` and `longitude` column.

Chapter 15

Cyme Extract

Pipeline status: 

CYME Model Data Extractor

Extract a CYME MDB file to its constituent tables in CSV format.

Input

The input folder must contain one or more MDB files, with the extension `.mdb`.

File `config.csv`:

Parameter name	Default value	Remarks
`FILES`	`*.mdb`	Supports patterns. Single filename example: `my-network.mdb`
`TABLES`	`*`	Supports patterns. Most CYME tables match `CYM*`
`EXTRACT`	`non-empty`	Allowed values are `all` or `non-empty`
`TIMEZONE`	`US/CA`	General format is `<country>/city`
`POSTPROC`	`network_graph`	Allowed post-processors are list in `postproc` folder. Current valid values are `network_graph`, `voltage_profile`, and `write_glm`
`OUTPUT`	`zip csv json`	File extensions to copy to the output folder.

Examples

Example 1 is based on the autotest/input_1.

config.csv

config.glm

modify.csv

settings.csv

Output

The output folder will contain a folder for each input MDB file, with CSV files corresponding to each of the tables in the input MDB file. CSV file names correspond to the MDB table name, with the `CYM` prefix removed and using lowercase letters.

An index file named `index.csv` is output containing information about each CSV file created, with the following structure

database	table	csvname	size	rows
mdbname	*CYMTABLENAME*	*tablename*	*n-chars*	*n-rows*

Docker Usage

You can run this pipeline on a supported docker container (e.g., `ubuntu:20.04`) using the following command:

```
host% mkdir input
host% cp my-database.mdb input
host% git clone https://github.com/openfido/cyme-extract --depth 1
host% mkdir output
host% docker run -it -v $PWD:$PWD -e OPENFIDO_INPUT=$PWD/input -e OPENFIDO_OUTPUT=$PWD/outpu
```