PQ strEEm Exploratory Data Analysis

'phasefront' Python package

Version 0.2.5

Table of contents

1. PhaseFront

PQ strEEm Exploratory Data Analysis tool

1.1 Installation

Use pip to automatically download and install the `phasefront` package from its hosted location on PyPI.

```
pip install phasefront
```

1.2 Configuration

Configuration is through a TOML file named `phasefront.toml` and placed in an accessible location. The software will check the following locations in priority order: 1. Path specified by `--config` argument 2. Path in `PHASEFRONT_CONFIG` environment variable 3. `./phasefront.toml` (current directory) 4. `~/.config/phasefront.toml` 5. `/etc/phasefront.toml`

If you do not have `phasefront.toml` file, you can create one in a text editor using the following entries:

```
[host]
address = "192.168.1.2"
username = "admin"
password = "secret"
```

1.3 Connection

The

with your host credentials. Typically the host is a PQ strEEm gateway but it can be any computer that supports SSH access and hosts PQ strEEM waveform and power monitor files 'wave' and 'pmon' folders respectively.

with SSH acc that possible hosts include but are not limited to PQ strEEm gateways.

1.4 Usage

1.4.1 Basic Commands

List remote files:

```
wave-list --config phasefront.toml '*.parquet'
```

Download files:

```
wave-fetch --config phasefront.toml '202411*.parquet'
```

Create plots:

```
wave-plot data.parquet
```

1.4.2 Windows PowerShell Usage

When using patterns on Windows PowerShell, be sure to wrap them in single quotes to prevent glob expansion:

```
wave-list '*.parquet'
wave-list '20241113_*.parquet'
```

1.4.3 Remote File Patterns

The `wave-list` command supports flexible file matching patterns:

```
# List all Parquet files in a directory
wave-list '*.parquet'
# List files for a specific date
wave-list '20241113_*.parquet'
# List files for a specific hour
wave-list '20241113_12*.parquet'
# List files within a time range
wave-list '20241113_{12,13,14}*.parquet'
```

The pattern matching uses Python's `Path.match()` function, which supports: - `*` : Matches any number of characters except slashes - `?` : Matches any single character except slashes - `[seq]` : Matches any character in seq - `[!seq]` : Matches any character not in seq - `{a,b,c}` : Matches any of the comma-separated alternatives

1.4.4 Pipeline Usage

Chain commands together:

```
wave-list '24111*.parquet' | wave-fetch | wave-plot
```

1.4.5 Plot Options

Wave Plot Options

Wave Plot supports multiple plot types that can be combined

```
# All plots (default)
wave-plot data.parquet

# Combine any plot types
wave-plot --wave --vhist data.parquet # Show waveforms and voltage histograms
wave-plot --vhist --ihist data.parquet # Show both histogram types
wave-plot --vderiv --ideriv data.parquet # Show both derivative plots

# Available plot types:
--wave # Voltage and current waveforms
--vhist # Voltage histograms
--ihist # Current histograms
--vderiv # Voltage derivative histograms
--ideriv # Current derivative histograms

# Show metadata for each file
wave-plot --verbose data.parquet
```

Power Monitoring Plot Options

Power monitoring plots support different metrics that can be combined:

```
# All plots (default)
pmon-plot data.parquet

# Combine any metrics
pmon-plot --freq --voltage data.parquet # Show frequency and voltage
pmon-plot --current --power data.parquet # Show current and power

# Available metrics:
--freq Frequency over time
--voltage RMS voltage
--current RMS current
--power Active and reactive power

# Show metadata for each file
pmon-plot --verbose data.parquet
```

1.4.6 Time Slicing

```
# First 100ms (default for wave plots)
wave-plot data.parquet

# Start at 2.0 seconds, show 500ms
wave-plot --start 2.0 --duration-millis 500 data.parquet

# Show 2 hours of power monitoring data starting at 14:00 UTC
pmon-plot --start 14:00 --duration-hrs 2 data.parquet
```

1.4.7 File Handling

When downloading files that already exist: - `wave-fetch` will prompt for action: - `[s]kip once`: Skip this file - `[S]kip all`: Skip all existing files - `[f]orce once`: Overwrite this file - `[F]orce all`: Overwrite all existing files - Use `-f` or `--force` to always overwrite without prompting

1.5 Exit Codes

All commands: - `0`: Success - `1`: Complete failure - `2`: Partial success (some files processed)

2. Development Guide

2.1 Getting Started

Clone and install in development mode:

```
git clone https://github.com/edge-energy/streem.git
cd phasefront
pip install -e .
```

2.2 Version Management

The package version is managed automatically using git tags. Here's how it works:

2.2.1 Development Mode

When installed with `pip install -e .`: - Version is dynamically fetched from git tags - Changes to code take effect immediately - Supports version formats: `v0.1.2       # Release version`
`   v0.1.2rc1   # Release candidate`
`   v0.2.2_ti   # Variant (adds +ti suffix)` - Development versions show commits since last tag: `0.1.2+dev5.gabc123f  # 5 commits after v0.1.2`
`   0.1.2+dev5.gabc123f.dirty  # Uncommitted changes`

2.2.2 Release Mode

For building releases:

```
# Tag the version you want to release
git tag v0.2.2 # or v0.2.2_xx for variants

#Generate version file and build package
./build.sh
```

Important notes: - Working directory must be clean (no uncommitted changes) - Version is frozen in .build_version.txt during build - Dirty working directory will cause build errors

2.3 Common Issues

1. Build fails with version error:
2. Ensure you've run `./build.sh`
3. Check that git tags are correct
4. Commit or stash changes
5. Version shows as "dirty":
6. Only allowed in development mode
7. Commit or stash changes before release
8. Wrong version appearing:
9. Check current git tag (`git describe --tags`)
10. Ensure you're on the right branch