

C-PAGE Manual

Version 1.0 Release

May 2026

Michael Abdelmalak,
Fernando Bereta dos Reis
Hisham Mahmood
Eran Schweitzer
Kostas Oikonomou

Pavel Etingov
Tony Nguyen
Bharat Vyakaranam
Nader Samaan



U.S. DEPARTMENT
of ENERGY

Prepared for the U.S. Department of Energy
under Contract DE-AC05-76RL01830

DISCLAIMER

This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor any agency thereof, nor Battelle Memorial Institute, nor any of their employees, makes **any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights.** Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof, or Battelle Memorial Institute. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof.

PACIFIC NORTHWEST NATIONAL LABORATORY
operated by
BATTELLE
for the
UNITED STATES DEPARTMENT OF ENERGY
under Contract DE-AC05-76RL01830

Printed in the United States of America

Available to DOE and DOE contractors from
the Office of Scientific and Technical Information,
P.O. Box 62, Oak Ridge, TN 37831-0062
www.osti.gov
ph: (865) 576-8401
fox: (865) 576-5728
email: reports@osti.gov

Available to the public from the National Technical Information Service
5301 Shawnee Rd., Alexandria, VA 22312
ph: (800) 553-NTIS (6847)
or (703) 605-6000
email: info@ntis.gov
Online ordering: <http://www.ntis.gov>

C-PAGE Manual

Version 1.0 Release

May 2026

Michael Abdelmalak,	Pavel Etingov
Fernando Bereta dos Reis	Tony Nguyen
Hisham Mahmood	Bharat Vyakaranam
Eran Schweitzer	Nader Samaan
Kostas Oikonomou	

Prepared for
the U.S. Department of Energy
under Contract DE-AC05-76RL01830

Pacific Northwest National Laboratory
Richland, Washington 99354

Abstract

This manual documents an end-to-end workflow for converting production cost modeling (PCM) outputs into alternating current (AC)-feasible power-flow base cases and geospatial visualizations using the C-PAGE, BPS-TUNE™, EGRET, and QGIS toolchain. Starting from GridView PCM results, the process standardizes data to HDF5, samples representative hours, and constructs direct current (DC) PowerWorld snapshots that are benchmarked against the original PCM flows. These DC cases are then transformed into EGRET-ready inputs, solved with a two-stage AC optimal power flow that adjusts reactive power and loss allocations to obtain AC-feasible solutions, and mapped back into converged AC PowerWorld.pwb cases. Dedicated benchmarking steps validate DC and AC consistency (PCM↔PowerWorld and EGRET↔PowerWorld) using comparison utilities available in BPS-TUNE™. The workflow is completed by database-backed and QGIS-based visualization capabilities that generate map projects and analytics outputs. The manual is written for power-system practitioners with limited software engineering background and provides tested commands, configuration templates, and practical guidance for running the C-PAGE process on Windows with Conda-managed Python environments.

Summary

This document translates the chronological alternating current (AC) power flow automated generation tool (C-PAGE) framework and associated software packages into a concrete, reproducible process for users who need to turn production cost model (PCM; GridView) runs into validated AC PowerWorld cases and visual products. The emphasis is on operational instructions rather than algorithms. Each step is presented with its purpose, required tools and branches, expected inputs and outputs, and the exact commands and Tom's Obvious Minimal Language (TOML) configuration patterns used on real project data.

Section 1 introduces an overview of the C-PAGE process and its role in bridging PCM outputs and AC power-flow models.

Section 2 is a preparation section that covers installation and configuration on Windows, including Conda environment setup, cloning and updating the relevant BPS-TUNE™ and EGRET repositories, branch management (e.g., `cpage`, `dataset_comparison`, `pwparser`, `pcm_acopf`, `qgis`), and access to local and HyperText Markup Language documentation.

Subsequent sections follow the numbered C-PAGE workflow steps: conversion of PCM .mdb results to .h5 (section 3), sampling and creation of direct current (DC) PowerWorld cases, DC benchmarking against PCM (section 4), construction of EGRET inputs (section 5), EGRET AC optimal power flow runs (section 6), conversion of EGRET AC solutions back to AC .pwb (section 7), and AC benchmarking between PowerWorld and EGRET (section 8). Each of these steps is backed by configuration file templates and examples that have been executed and verified on representative cases.

Section 9 documents the visualization and analytics phase using QGIS integration in BPS-TUNE™. It focuses on the Python-CLI pathway, using `make_qgis_data` and `build_qgz` to produce QGIS projects driven by database-backed views and configuration TOMLs. Throughout, the manual is structured for beginner users of these tools: detailed shell contexts (Git Bash vs. Command Prompt), explicit environment activation, and validation checkpoints are included to reduce friction and make the C-PAGE workflow repeatable across projects and users.

Acronyms and Abbreviations

.bin	binary files
AC	alternating current
ACOPF	AC optimal power flow
C-PAGE	Chronological AC power flow automated generation tool
CSV	comma-separated values (.csv)
DC	direct current
DOE	U.S. Department of Energy
EGRET	Electrical Grid Research and Engineering Tools
GIS	geographic information system
BPS-TUNE™	Bulk Power System - Transformation via Upgraded Network Evaluation
GV	GridView
HDF5	Hierarchical Data Format version 5 (.h5)
HTML	Hyper Text Markup Language
HVDC	high-voltage direct current
MDB	Microsoft Access Database format (.mdb)
MTDC	multi-terminal dc line
PCM	production cost model
PF	power flow
PNNL	Pacific Northwest National Laboratory
Q-V	reactive power vs. voltage analysis
QGIS	Quantum geographic information system
SSH	Secure Shell
TOML	Tom's Obvious Minimal Language
UniGrid	unified data-exchange module enabling tool interoperability
URL	Uniform Resource Locator
VS Code	Visual Studio Code
VSC	Voltage Source Converter

Contents

Abstract.....	ii
Summary.....	iii
Acronyms and Abbreviations	iv
1.0 Introduction	1
1.1 Purpose and Scope.....	1
1.1.1 High-Level Flow	1
1.1.2 AC Solution via EGRET (two stages)	2
1.1.3 From optimization Back to a PF Case	2
1.1.4 Benchmarking, Harmonization, and Planning.....	2
1.1.5 Outputs and Visuals.....	2
1.2 Process at a Glance	2
2.0 Preparation.....	5
2.1 Install Anaconda or Miniconda (Windows)	5
2.2 Create and Activate the Conda Environment	5
2.3 BPS-TUNE™ and C-PAGE Installation.....	6
2.3.1 Install BPS-TUNE™ (first-time setup).....	6
2.3.2 Enable C-PAGE (branch + extras).....	7
2.4 EGRET Installation.....	8
2.4.1 Copy the Clone URL (private repo)	9
2.4.2 Choose the Parent Folder (sibling to BPS-TUNE™) and Clone.....	9
2.4.3 If the Project Expects A Specific Branch	9
2.4.4 Install EGRET into the Conda Environment	10
2.5 Additional User Documentation.....	10
2.5.1 Local HTML Documentation (build and view).....	10
2.5.2 Offline HTML Documentation Package (html.zip).....	11
2.6 Purpose	11
2.7 Step 2 – Requirements	Error! Bookmark not defined.
2.7.1 General.....	Error! Bookmark not defined.
2.7.2 Pre-Run Update (BPS-TUNE™ only).....	11
2.8 Step 2 – Quick Path (recommended)	12
2.9 Step 2 – Options and Overrides	12
2.10 Step 2 – Expected Outputs.....	13
2.11 Step 2 – Where to Learn More	13
3.0 Step 3 – Build DC Power-Flow Snapshots From PCM	14
3.1 Purpose	14
3.2 Step 3 – Preparation of BPS-TUNE™ Repository.....	15

3.2.1	Option A: From the BPS-TUNE™ folder (File Explorer → Git Bash)	15
3.2.2	Option B: From Visual Studio Code (VS Code) with Git Bash	16
3.3	Step 3 – Install the Updated BPS-TUNE™ into Python	16
3.4	Step 3 – Get the Required BPS-TUNE™ Branch (if applicable)	17
3.5	Step 3 – Configuration Template for PCM to PowerWorld	17
3.6	Step 3 – Executing Automation Script cpageautomation.py	21
3.7	Step 3 – Output	21
4.0	Step 4 – DC Cases Benchmarking (PowerWorld .pwb vs PCM)	22
4.1	Step 4 – Purpose	22
4.2	Step 4 – Scope and General Capability	22
4.3	Step 4 – Requirements	23
4.3.1	Tools and Environment	23
4.3.2	Inputs	23
4.3.3	Entry Point	23
4.4	Step 4 – Pre-Run Update (BPS-TUNE™ → <branch> if applicable)	23
4.5	Step 4 – Configuration Inputs	24
4.5.1	Comparison Control Configuration	24
4.5.2	DC Parameters/Thresholds Configuration	26
4.6	Step 4 – Execution	28
4.7	Step 4 – Output	28
4.8	Step 4 – Summary	29
5.0	Step 5 – Create DC JSON Cases (PowerWorld → EGRET input)	30
5.1	Step 5 – Purpose	30
5.2	Step 5 – Pre-Run Update Checklist (required once per session)	30
5.2.1	Update BPS-TUNE™/C-PAGE (Git Bash, BPS-TUNE™ repo root)	30
5.2.2	Update EGRET (Git Bash, EGRET repo root)	30
5.3	Step 5 – Inputs	31
5.3.1	User Configuration (example: 2030ADS_to_acpf.toml)	31
5.3.2	Parser Overrides (pwdconfig.toml)	31
5.3.3	Templates (ready to paste)	31
5.4	Step 5 – Execution	33
5.5	Step 5 – Summary	33
6.0	Step 6 – Run EGRET ACOPF to produce AC-feasible cases	34
6.1	Step 6 – Purpose	34
6.2	Step 6 – Requirements	34
6.3	Step 6 – Pre-Run Update (EGRET and dependencies)	35
6.4	Step 6 – Directory and Configuration Layout	35

6.5	Step 6 – Basic Usage (single-case run)	36
6.6	Step 6 – Summary	38
7.0	Step 7 – Build AC Power Flow Cases from EGRET Output.....	39
7.1	Step 7 – Purpose.....	39
7.2	Step 7 – Preparation	39
7.3	Step 7 – Configuration Template for EGRET to PowerWorld.....	39
7.4	Step 7 – Executing Automation Script cpageautomation.py.....	42
7.5	Step 7 – Output	43
8.0	Step 8 – AC Cases Benchmarking (PowerWorld .pwb vs EGRET .json)	44
8.1	Step 8 – Purpose.....	44
8.2	Step 8 – How Step 8 Differs from Step 4 (reference).....	44
8.3	Step 8 – Requirements.....	44
8.3.1	Tools and Environment.....	44
8.3.2	Inputs.....	44
8.4	Step 8 – Pre-Run Update (BPS-TUNE™ and dependencies)	45
8.5	Step 8 – Configuration Inputs.....	45
8.5.1	Comparison Control Configuration	45
8.5.2	AC Parameters/Thresholds Configuration	46
8.6	Step 8 – Execution	48
8.7	Step 8 – Output	48
8.8	Step 8 – Summary.....	49
9.0	Step 11 – Visualization.....	50
9.1	Step 11 – Purpose.....	50
9.2	Step 11 – Requirements.....	50
9.3	Step 11 – Interfaces in This Step	50
9.4	Step 11 – Pre-Run Setup (BPS-TUNE™ qgis branch and environment).....	51
9.4.1	Ensure the Environment Uses a Compatible Python Version	51
9.4.2	Update BPS-TUNE™ to the qgis Branch	51
9.4.3	Install BPS-TUNE™ with QGIS Extras into this Env	51
9.5	Configuration Files and Where to Find Them.....	52
9.6	Database Configuration TOML (my_db_config.toml).....	52
9.7	Data Extraction: make_qgis_data	53
9.8	QGIS Project Configuration TOML (my_qgis_config.toml).....	53
9.9	Build the QGIS Project: build_qgz.....	55
9.10	Where to Find Detailed Help and Options	56
	Appendix A – Tools/Branch Mapping.....	A.1
	Appendix B – Create Mapping Files	B.1

Figures

Figure 1.1.	C-PAGE Workflow Steps.....	4
Figure 2.1.	PCM/Utilities Section Covering <code>gv_make_h5</code>	13
Figure 3.1.	Robust AC Convergence Workflow. Colors represent model types: green is PCM, blue is C-PAGE.....	15
Figure 3.2.	C-PAGE Module Workflow	15
Figure 3.3.	Step 3: Output Directory Layout and Results	21
Figure 4.1.	Step 4: Output Directory Layout and Results	29
Figure 6.1.	Step 6: Run Directory Layout and Path, Results Folder, and the README.md	37
Figure 7.1.	File Structure Created Inside the Designated Output Directory	43
Figure 9.1.	Voltage heatmap created using visualization tool, and displayed in QGIS software, for one operation instant of the WECC grid.	56

1.0 Introduction

1.1 Purpose and Scope.

C-PAGE, integrated within Bulk Power System - Transformation via Upgraded Network Evaluation (BPS-TUNE™), converts direct current (DC)-based production-cost outputs into converged alternating current (AC) power-flow cases and associated visual analytics.¹ BPS-TUNE™ is a collection of PNNL's grid planning capabilities. It is intended to be a tool (or library of tools) that support the simulation and analysis involved in various grid planning activities (e.g., national transmission planning study,² value of nuclear energy to the reliability of the North American power system,^{3,4} and west coast offshore wind transmission study⁵). The workflow automates hour selection⁶, DC→AC translation, reactive support decisions, and case generation in commercial power flow (PF) tools, reducing a months-long manual process to hours while preserving fidelity to the original production cost model (PCM) solution.

1.1.1 High-Level Flow

Starting from PCM time-series (e.g., GridView), BPS-TUNE™ leverages the unified data-exchange module enabling tool interoperability (UniGrid) to acquire the necessary PCM time-series data. Smart sampling is performed to select representative PCM hours and hands those hours to C-PAGE. C-PAGE first builds DC-consistent snapshots that match PCM dispatch and flows in the PF model, then uses the Electrical Grid Research and Engineering Tools (EGRET) AC optimal power flow (ACOPF) feature to obtain a feasible AC solution with targeted reactive support. EGRET⁷ is a package that offers tools and models for optimizing electric grid operations. The EGRET solution is mapped back into a commercial PF case (PowerWorld), benchmarked, harmonized across hours, and finalized through planning based on reactive power vs. voltage analysis. Analytics and visualization modules summarize results and prepare geospatial/ Hyper Text Markup Language (HTML) outputs, with DC snapshot as the bridge.

Because PCM is DC-linear and PF is AC-nonlinear, the first conversion step constructs a DC power-flow case in the PF tool that reproduces the PCM operating point. This reconciliation distributes region-level load to buses, disaggregates plant-level injections to units, aligns

¹ Bharat Vyakaranam et al., "Automated Tool to Create Chronological AC Power Flow Cases for Large Interconnected Systems," *IEEE Open Access Journal of Power and Energy* 8 (2021): 166–174, <https://doi.org/10.1109/OAJPE.2021.3075659>.

² National Renewable Energy Laboratory (NREL), *The National Transmission Planning Study: Chapter 4 – AC Power Flow Analysis for 2035 Scenarios* (Washington, D.C.: U.S. Department of Energy, 2024), program document, available at OSTI.

³ Samaan, Nader A., et al. "Value of Nuclear Energy to the Reliability of the North American Power System: Results for Western and Eastern Interconnections", Jul. 2024. <https://doi.org/10.2172/3013402>

⁴ Fernando Bereta dos Reis et al., "A Production Cost Modeling Approach to Analyze Nuclear Power Plant Availability during Normal and Extreme Weather Events," paper presented at the 2025 *IEEE Power & Energy Society General Meeting (PESGM)*, July 2025, 1–5, <https://doi.org/10.1109/PESGM52009.2025.11225012>.

⁵ Douville, Travis C., et al. "West Coast Offshore Wind Transmission Study." , Jan. 2025. <https://doi.org/10.2172/2500279>.

⁶ T. -C. Chen et al., "Smart Sampling of Representative Hourly Power Generation Scenario With High Renewable Penetration," in *IEEE Access*, vol. 12, pp. 115319-115328, 2024, doi: 10.1109/ACCESS.2024.3445409.

⁷ Sandia National Laboratories 2019

identifiers and topology, and imposes high-voltage direct current (HVDC) transfers at their AC terminals so DC flows match the PCM hour. This verified DC case is used as a seed for the AC stage.

1.1.2 AC Solution via EGRET (two stages)

EGRET solves an ACOPF in two stages. First, a feasibility phase adjusts for losses (i.e., area-coupled curtailment proxies) and provides broad reactive slack (proxy shunts) so a converged AC solution exists. Then, an optimization phase tightens criteria, penalizes deviations from PCM dispatch, and drives a sparser, more realistic reactive footprint. HVDC elements are modeled as coupled proxy generators during optimization and later mapped back to explicit DC equipment.

1.1.3 From optimization Back to a PF Case

C-PAGE translates the EGRET solution into a PowerWorld .pwb file by updating buses, generators, loads, phase shifters, and DC lines in a defined order. HVDC proxy results are converted into terminal injections that account for converter losses, and variable shunts from EGRET are discretized into actual step sizes. This produces an AC case that reflects the optimized solution and is ready for benchmarking.

1.1.4 Benchmarking, Harmonization, and Planning

The generated AC case is compared against the EGRET target (voltage magnitudes, slack accounting, HVDC transfers) and checked against solution-quality criteria, such as the absolute mismatch or mismatch percentage thresholds of branch flows, generator power, bus voltages, etc. Across multiple hours, C-PAGE consolidates added reactive devices, ranks candidates by frequency and local Q–V margins, and re-optimizes to minimize the device set while preserving performance. A reactive power vs. voltage analysis-based planning stage assesses reserves, tunes existing assets, and adds support only where sensitivity justifies it; an automated base-case selector improves robustness when large operating changes are present.

1.1.5 Outputs and Visuals

Final PF cases, metrics, and maps are written back through UniGrid for downstream use. The UniGrid model unifies data structure to standardize the reading and writing of grid related data in BPS-TUNE™. The Analytics and Visualization module produces utilization metrics, voltage/flow visuals, geographic information system (GIS)/Quantum geographic information system (QGIS) layers, and HTML reports, thus enabling transparent review and communication of results.

1.2 Process at a Glance

C-PAGE workflow steps are described below and shown graphically in Figure 1.1.

1. PCM production cost run (input): The process assumes a completed GridView PCM study that produces a yearly or multi-period database (e.g., an .mdb case directory) with unit commitment, dispatch, and hourly flows. This PCM run is treated as the upstream source and provides the sole input to Step 2.

2. Convert PCM output to HDF5: The user begins with a GridView PCM run (stored as an .mdb case directory) and uses BPS-TUNE™ (gv_make_h5) to extract and standardize the results to a UniGrid HDF5 file (.h5) for downstream processing.
3. PCM sampling and C-PAGE DC outputs: The user selects representative hours from the PCM run and, using BPS-TUNE™/C-PAGE (cpageautomation), builds DC PowerWorld snapshots (.pwb) that are consistent with PCM dispatch and flows in the selected power-flow topology.⁸
4. DC benchmarking: PowerWorld vs PCM: The user validates DC fidelity by comparing the DC PowerWorld cases from Step 3 against the PCM hour targets in the .h5 file, using dataset comparison utilities available in BPS-TUNE™. The comparison is performed on flows, injections, and bus angles, with thresholds controlled via a TOML (Tom's Obvious, Minimal Language) file configuration.
5. Create EGRET inputs from DC cases: The user converts each DC PowerWorld case to an EGRET-compatible DC JSON input using EGRET's PowerWorld parser (make_acopf_from_dcpf.py), capturing both the DC operating point and the associated AC network description for the subsequent ACOPF.
6. Run EGRET ACOPF (loss adjustment and reactive support): EGRET's pcm_acopf extension is used to solve an AC optimal PF for each DC snapshot, adjusting reactive power and, where configured, loss allocations and small active power/load corrections to obtain an AC-feasible operating point.
7. Convert EGRET AC results to AC PowerWorld cases: C-PAGE maps the EGRET AC solution back into converged AC PowerWorld cases (.pwb) using cpageautomation, including treatment of HVDC, switched shunts, and discretization of reactive devices where required.
8. AC benchmarking: PowerWorld vs EGRET: The user compares the AC PowerWorld cases from Step 7 against the EGRET AC solutions from Step 6 using the same dataset comparison framework, focusing on voltage magnitudes, angles, active/reactive flows, slack accounting, and HVDC transfers, with configurable AC-specific thresholds.
9. Reactive power harmonization across hours (advanced): Across multiple hours, C-PAGE is used to build an "Added Equipment Store," rank candidate reactive devices by selection frequency and Q–V margin contribution, and re-optimize to minimize the number of devices while meeting voltage and reserve criteria. This step is still under development for future release versions.
10. Fine-tuning with Q–V analysis (advanced): Q–V analysis is applied to quantify reactive reserves, identify weak locations, and apply targeted local corrections. Automated base-case selection is used where large operating changes would hinder convergence, and the analysis can be extended across seasons and consolidated. This step is covered in Reference no. 1.
11. Visualization and analytics: Final AC cases and comparison results are used to create maps, time-series plots, and HTML reports via BPS-TUNE™ analytics and QGIS integration. Outputs can be written through UniGrid for use in other tools and downstream workflows.

⁸ Tse-Chun Chen et al., "Smart Sampling of Representative Hourly Power Generation Scenario With High Renewable Penetration," *IEEE Access* 12 (2024): 115 319–115 328, <https://doi.org/10.1109/ACCESS.2024.3445409>.



Figure 1.1. C-PAGE Workflow Steps

2.0 Preparation

2.1 Install Anaconda or Miniconda (Windows)

Conda is chosen because it provides isolated, reproducible Python environments on Windows, makes it easy to pin the exact Python version required by BPS-TUNE™/C-PAGE, and simplifies installation of scientific packages with their required dependencies.

What to install. Miniconda (lighter) or Anaconda (full distribution). Either provides Conda functionality.

Installer choices that matter.

- Architecture: 64-bit
- Scope: “Just me” is sufficient.
- PATH: Leaving “Add to PATH” unchecked is typical; the installer adds an Anaconda Prompt that pre-configures PATH.

How to run Conda after installation.

- Easiest: Open Start → Anaconda Prompt (Miniconda3/Anaconda3).
- To use Conda in other shells (optional), initialize once and restart the shell:
 - Open **Anaconda Prompt** from the Start menu.
 - Run:
 - Command Prompt:

```
bat
conda init cmd.exe
```

- PowerShell:

```
bat
conda init powershell
```

- Git Bash (if desired for Git operations):

```
bash
conda init bash
```

2.2 Create and Activate the Conda Environment

Shell: Anaconda Prompt (or any shell where Conda is initialized).

1. Create the environment (example name “gridtune⁹”; pick Python 3.10 or 3.11; 3.10 is a safe default):

⁹ “gridtune” is the name of the main folder and repo name for the BPS-TUNE™. This alias can easily be changed.

```
bat
conda create --name environment_name python=3.11
```

For an example name “gridtune”

```
bat
conda create --name gridtune python=3.11
```

2. Activate the environment:

```
bat
conda activate gridtune
```

3. (Optional but helpful) Upgrade pip inside the env:

```
bat
python -m pip install --upgrade pip
```

4. Checks:

```
bat
python -V
where python
pip --version
```

2.3 BPS-TUNE™ and C-PAGE Installation

C-PAGE is implemented as a Python module within BPS-TUNE™ and relies on external simulators; the tested runtime is Windows with Python ≥ 3.9, and the currently implemented AC PF engine is PowerWorld.

2.3.1 Install BPS-TUNE™ (first-time setup)

2.3.1.1 Copy the Clone Uniform Resource Locator (URL)

- Browser → Open the project page: <https://github.com/pnnl/BPS-TUNE>¹⁰ public or the private <https://tanuki.pnnl.gov/gridtune/gridtune>.
- Click **Clone** and copy the **HTTPS** URL (e.g., <https://tanuki.pnnl.gov/gridtune/gridtune.git>) or secure shell (**SSH**) URL (e.g., <git@tanuki.pnnl.gov:gridtune/gridtune.git>).
 - HTTPS may prompt for a Personal Access Token (PAT) when Git asks for a password.
 - SSH requires a registered SSH key in Tanuki.

2.3.1.2 Choose the Parent Folder and Run the Clone

- **Shell: Git Bash.** In File Explorer, open the desired parent folder (e.g., C:\Work\Repos). Then right-click in a blank area → **Git Bash Here**.

¹⁰ The rest of the document utilizes the private link, since the public link is in the process of being made available.

- **Command (example, HTTPS):**

```
bash
git clone https://tanuki.pnnl.gov/gridtune/gridtune.git
```

A new subfolder “gridtune” is created under the current folder.

2.3.1.3 Verify the Working Copy

- **Shell:** Git Bash (inside the new folder).

```
bash
cd gridtune
git fetch
```

2.3.1.4 Install BPS-TUNE™ into the Python Environment (editable)

- **Shell:** Command Prompt
- **Folder:** the BPS-TUNE™ repository root.

```
bat
conda activate gridtune
pip install -e .
```

This makes the local BPS-TUNE™ source importable while allowing edits without re-installation.

2.3.2 Enable C-PAGE (branch + extras)

C-PAGE is integrated into BPS-TUNE™ as modules (e.g., **CPAGEDData**, **CPAGEMapping**, **CPAGESolver**) that handle PCM↔PF data linkage and case creation.

2.3.2.1 Update the Local Repository

- **Shell:** Git Bash (repo root).

```
bash
git fetch
git pull
```

- If git pull is blocked by “local changes,” clean the working files and pull again:

```
bash
git restore .
git pull
```

This discards uncommitted edits to tracked files in the repo, which is safe when the repository has not been intentionally modified.

2.3.2.2 Check Out the Required BPS-TUNE™ Branch (if applicable)

Note. During active development, functionality may temporarily reside on feature branches before being merged into main. The generic branch name `<branch>` used in the example commands is a placeholder. The actual branch to use for each tool is listed in Appendix A (Tool–Branch Mapping). Before running the Git commands, the user should replace `<branch>` with the branch indicated in Appendix A (for example `main`, `dataset_comparison`, `pcm_acopf-solver`, or `qgis`).

- **Shell:** Git Bash

```
bash
git checkout -t origin/<branch>
git pull
```

- If Git reports that `cpage` already exists locally, switch to it and pull updates:

```
bash
git checkout <branch>
git pull
```

2.3.2.3 Install with Branch Extras

- **Shell:** Command Prompt.
- **Folder:** BPS-TUNE™ repository root.

```
bat
conda activate gridtune
pip install -e ".[all]"
```

2.3.2.4 Quick CLI Check (optional)

- **Shell:** Command Prompt.

```
bat
cpageautomation --help
```

A help screen indicates the entry point is visible in the active environment.

2.4 EGRET Installation

Goal: Prepare a local EGRET working copy and install it into the existing Conda environment used by BPS-TUNE™, so later benchmarking and data utilities are available.

Shells used: Git Bash for Git; Command Prompt for Conda/Python (if `conda` is not recognized in Command Prompt, initialize it once with `conda init cmd.exe` and reopen).

The process used to obtain an EGRET working copy is described in the following sections.

2.4.1 Copy the Clone URL (private repo)

- Open: <https://github.com/pnnl-private/egret>
- Click **Code** → choose:
 - **HTTPS** URL (simple; will prompt for username + **Personal Access Token (PAT)** as password), or
 - **SSH** URL (requires an SSH key registered in GitHub).

2.4.2 Choose the Parent Folder (sibling to BPS-TUNE™) and Clone

- Shell: Git Bash
- Folder: the parent directory that already contains the gridtune folder (so EGRET will sit next to it)

```
bash

#Example (HTTPS)
git clone https://github.com/pnnl-private/egret.git
#Result: a new folder named "egret" under the current directory
cd egret
```

2.4.3 If the Project Expects A Specific Branch

Note. During active development, functionality may temporarily reside on feature branches before being merged into main. The generic branch name `<branch>` used in the example commands is a placeholder. The actual branch to use for each tool is listed in Appendix A (Tool/Branch Mapping). Before running the Git commands, the user should replace `<branch>` with the branch indicated in Appendix A (e.g., `main`, `dataset_comparison`, `pcm_acopf-solver`, or `qgis`).

- **First time on this machine** (create a local branch that tracks the remote):

```
bash

git fetch                # refresh local knowledge of server branches
git branch -r            # confirm the branch exists (optional check)
git checkout -t origin/<branch> # create local 'branch' tracking the server
git pull
```

- **Later updates** (branch already exists locally):

```
bash

git fetch
git checkout <branch>
git pull
```

If `git checkout -t origin/pwdparser` says the branch already exists, switch to it with `git checkout pwdparser` and then `git pull`.

2.4.4 Install EGRET into the Conda Environment

- **Shell:** Command Prompt (with Conda available)
- **Folder:** the **egret** repository root

```
bat
conda activate gridtune # gridtune is an environment name example
pip install -e .
```

2.5 Additional User Documentation

2.5.1 Local HTML Documentation (build and view)

Purpose. Build and open the same documentation locally from the checked-out source.

Where to run:

- **Shell:** Command Prompt
- **Folder:** <GridTUNE root>\docs

Prerequisite: Python package docs

```
bat
conda activate gridtune
pip install -e ".[docs]"
```

or

```
pip install -e ".[all]"
```

Steps:

1. In Windows File Explorer, navigate to <GridTUNE root>\docs.
2. Open a Command Prompt in this directory (e.g., click in the address bar, type `cmd`, press Enter).
3. Build the HTML (the project expects running the command **twice**):

```
bat
.\make.bat html
.\make.bat html
```

4. (Alternative) Auto-update the code reference page and build:

```
bat
.\auto_make.bat
.\auto_make.bat
```

This runs `docs\source\code_auto_module_rst.py`, updates `docs\source\files\auto_code.rst`, and then invokes `.\make.bat html` twice.

Open the result:

- The built site is under `docs\build\html\`.
- Open `docs\build\html\index.html` in a browser.

2.5.2 Offline HTML Documentation Package (html.zip)

This manual is distributed with an offline copy of the BPS-TUNE™ HTML documentation as `html.zip`.

Steps

1. Place `html.zip` in the same folder as the local.
2. Extract `html.zip`.
3. Confirm that the extracted folder contains the file: `html\index.html`
4. Open `html\index.html` by double-clicking it, or by opening it from a browser via `File → Open`.

Step 2 — Convert GridView run (`.mdb`) → HDF5 (`.h5`)

2.6 Purpose and Requirements

Convert a GridView PCM run stored in a single `.mdb` and multiple `.bin` (binary) files into an `.h5` file. The resulting `.h5` is the input used in Step 3 to create DC power-flow snapshots.

- BPS-TUNE™ installed in the active Conda environment (Preparation Section 2.3).
- The GridView run folder (the directory that contains the `.mdb` and the multiple `.bin` files produced by GridView).
- Write access to that folder (the tool writes the `.h5` and a log file there).

2.6.1 Pre-Run Update (BPS-TUNE™ only)

Purpose. Ensure the `gv_make_h5` script exposed by BPS-TUNE™ is current in the active environment. (EGRET is not required for Step 2.)

First time on this machine (create local main if needed)

Shell: *Git Bash* (BPS-TUNE™ repo root)

```
bash
git fetch
git checkout -t origin/main    # only if 'main' does not exist locally
```

Later runs (most days)

Shell: *Git Bash* (BPS-TUNE™ repo root)

```
bash
git fetch
git checkout main
git pull
```

2.6.2 Quick Path (recommended)

Shell: Command Prompt

Folder: the GridView run directory that contains the `.mdb` and the multiple `.bin` files

```
bat

conda activate gridtune
gv_make_h5 .
```

What happens

The command scans the current folder (.) for the GridView run and writes a new `.h5` file in the same folder.

A log named `make_h5_GV_py.log` is also written to the folder. Review warnings/errors before using the file downstream.

Tip. The user may run from anywhere and pass the folder explicitly:

```
bat

conda activate gridtune
gv_make_h5 "C:\Path\To\GridViewRun"
```

2.6.3 Options and Overrides

Help output (for reference)

Command

```
bat

gv_make_h5 -help
```

```
usage: gv_make_h5 [-h] [--toml_ov_config TOML_OV_CONFIG] [--logfile LOGFILE]
gv_case_path

Make a h5 file for PCM GV run.

positional arguments:
  gv_case_path          The path to the GV case directory.

options:
  -h, --help            show this help message and exit
  --toml_ov_config TOML_OV_CONFIG
                        The path to the TOML file to overwrite default
                        configurations. Defaults to "None".
  --logfile LOGFILE     Path to logfile. Defaults to
                        "gv_case_path/make_h5_GV_py.log".
```

Examples

```
# Explicit folder + custom log location
gv_make_h5 "D:\GV\Runs\FY25_S4_R3" --logfile
"D:\GV\Runs\FY25_S4_R3\make_h5_GV_py.log"

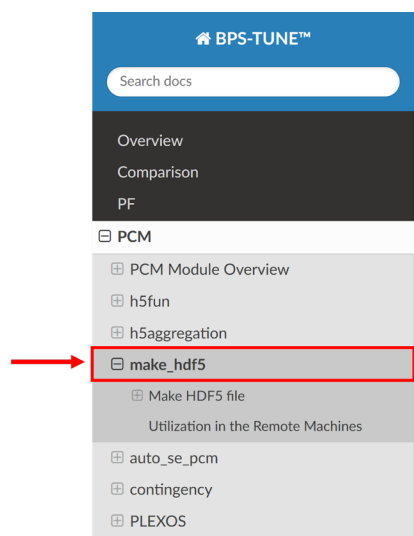
# Use an override TOML to change defaults
gv_make_h5 "D:\GV\Runs\FY25_S4_R3" --toml_ov_config
"D:\GV\Runs\FY25_S4_R3\gv_h5_overrides.toml"
```

2.6.4 Expected Outputs

- An `.h5` file written to the GridView run folder.
- A log file: `make_h5_GV_py.log` (or the path provided via `--logfile`). The log records any issues during extraction; confirm warnings/errors do not affect the analysis before proceeding to **Step 3**.

2.6.5 Where to Learn More

For advanced options (override TOML content, API notes, edge-case handling), see the interactive HTML documentation built with BPS-TUNE™ (Chapter 2 Section 2.5). In that site, navigate to the PCM/Utilities section covering `gv_make_h5` (Figure 2.1).



The screenshot shows the BPS-TUNE documentation interface. On the left is a navigation sidebar with a search bar and a list of sections: Overview, Comparison, PF, PCM, PCM Module Overview, h5fun, h5aggregation, **make_hdf5** (highlighted with a red box and a red arrow), Make HDF5 file, Utilization in the Remote Machines, auto_se_pcm, contingency, and PLEXOS. The main content area on the right is titled 'make_hdf5' and contains the following text:

Make HDF5 file

This module utilizes the GV API. Additional information on the API can be found on the PNNL share folder "`\PNL\Projects\Grid_Software\ABB GridView\PNNL Python Scripts\GVReader for Python 3.11\GV Python 3.11 setup files`".

With BPS-TUNE this creates the script call `gv_make_h5` to the python environment.

Utilized capability

The additional dependencies utilized are:

- "pyodbc"
- "sqlalchemy-access" Witch assist in collecting the information from the GridView `.mdb` case file. This is necessary to utilize the reminder of the automation in BPS-TUNE leveraging the HDF5 analytics.

Advanced User (altering default configuration)

Figure 2.1. PCM/Utilities Section Covering `gv_make_h5`

3.0 Step 3 – Build DC Power-Flow Snapshots From PCM

3.1 Purpose

Build a DC power-flow snapshots from PCM outputs to have a representative PF model of a particular state of the system from PCM. This ensures that components dispatch set points and PFs match the PCM solution before any AC physics are introduced. This is the “DC snapshot” in the robust-convergence workflow (see Section 4.1 and Figure 4.1). The output is a PF case that 1) solves DC and 2) reproduces the PCM hour’s dispatch and interface/branch flows within configured tolerances. This validated DC seed is the hand-off to ACOPF in the next stage. It is worth noting that several reconciliation tasks take place as follows:

- **Loads (regional → bus level):** PCM often reports load at region/area level and distributes to buses with factors; the PF model holds explicit bus loads. Step 3 applies the distribution so total P and Q at buses sum to the PCM hour and appear at the correct buses used by the PF case.
- **Generators (aggregated → unit level):** PCM may represent a plant or combined-cycle block as one injection, whereas the PF model has individual representation of each unit at specific buses. Step 3 disaggregates the PCM injection into unit setpoints using plant-specific ratios (or a mapping file) so each PF unit receives its share.
- **High-Voltage DC lines (bidirectional → unidirectional):** PCM treats HVDC as directed flows on a single branch. On the other hand, the PF model contains explicit DC elements (or will later use ACOPF proxies) or equivalent representation for DC. In Step 3, the DC snapshot sets HVDC transfers so that terminal injections/withdrawals match the PCM flow direction and magnitude at the connected buses. (Figure 3.1 shows the ACOPF proxy model used later; Step 3 simply honors the PCM transfer at the PF terminals.)
- **Component relocation:** PCM sometimes reduces the connectivity of generators by omitting collector systems, step-up transformers, or other fine-grain elements, that exist in PF, yielding an aggregated or simplified representation of some generators. Step 3 imprints the PCM operating point onto the PF topology that actually exists (bus numbers, kV levels, branch IDs), so the DC solution replicates PCM flows in that network.
- **Solution settings statuses and controls for a DC solve:** Upon updating setpoints in the PF model, a DC-solution is required. Specific solver settings are enforced to ensure a DC power-flow solve in the PF tool, without engaging AC voltage-control behavior yet. This includes transformers tap controls, shunts controls, or DC converters controls.

The CPAGEAutomation class is built to automate the process of using CPAGE tool. CPAGE module workflow is illustrated using the simplified diagram in Figure 3.2.

It provides:

- Automated creation of C-PAGE data and chronological PF models
- Flexible configuration through a TOML file
- Generates detailed logs and reports for analysis

Uses command-line arguments for ease of execution.

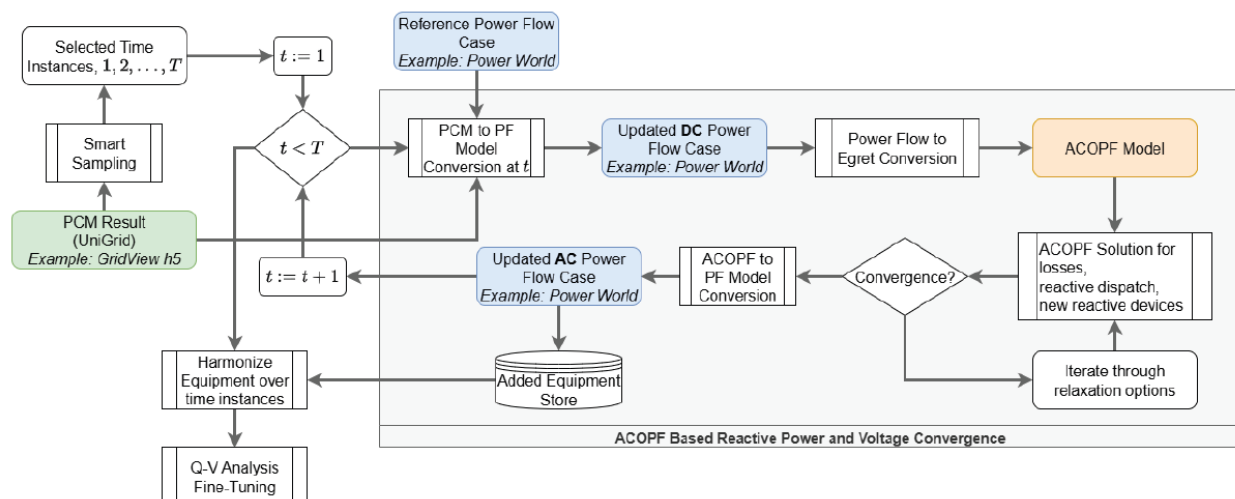


Figure 3.1. Robust AC Convergence Workflow. Colors represent model types: green is PCM, blue is C-PAGE.

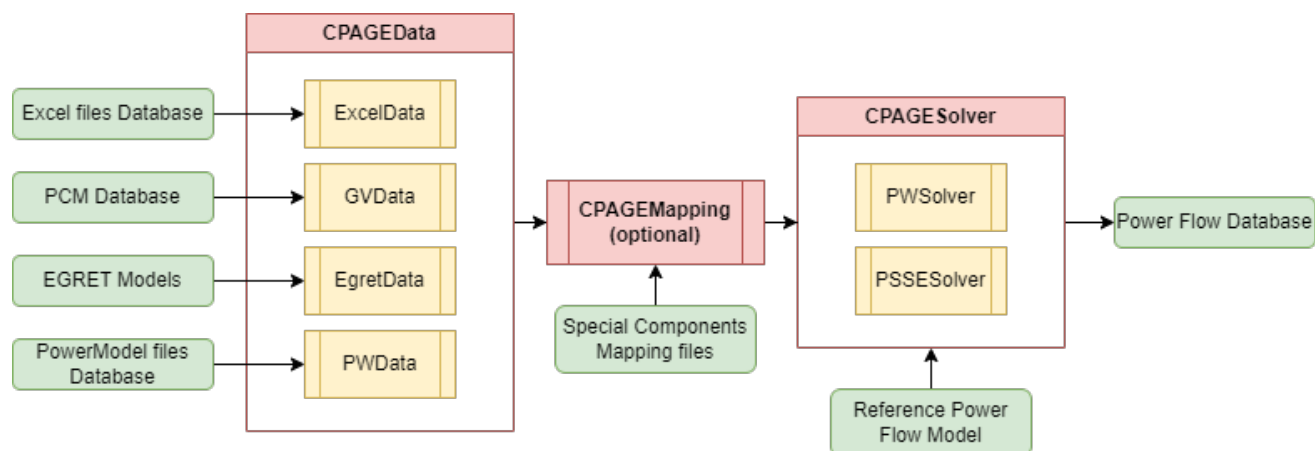


Figure 3.2. C-PAGE Module Workflow

3.2 Step 3 – Preparation of BPS-TUNE™ Repository

3.2.1 Option A: From the BPS-TUNE™ folder (File Explorer → Git Bash)

1. Open Git Bash in the repo
 - In File Explorer, go to your BPS-TUNE™ folder.
 - Right-click in a blank area → **Open Git Bash here.**
(This opens a terminal window already “inside” the BPS-TUNE™ repo.)
2. Get info about new changes

```
bash
git fetch
```

- *What it does:* contacts the remote server (Tanuki GitLab) to see what’s new.

- *What you might see:* prompts for **username and password** (enter them), then lines like From ... and a list of updated branches.

3. Bring those changes into your local files

```
bash
git pull
```

- *What it does:* updates your current branch's files to match the server.
- *If Git says there are "pending changes" and won't pull:* run

```
bash
git restore .
```

This discards your local edits in the working folder (not committed changes), then run:

```
bash
git pull
```

3.2.2 Option B: From Visual Studio Code (VS Code) with Git Bash

- Open VS Code in the BPS-TUNE™ folder.
- In the **bottom terminal** strip (the one with the + icon), click the small **arrow** next to + → choose **Git Bash**.
- Repeat the same two commands as above:

```
bash
git fetch
git pull
```

- If you see "pending changes" errors, use:

```
bash
git restore .
git pull
```

3.3 Step 3 – Install the Updated BPS-TUNE™ into Python

In VS Code, in the **bottom terminal**, click the arrow next to + and choose **cmd** (Command Prompt).

make sure you activate the proper python environment before running the following:

```
bat
pip install -e .
```

- What it does: "editable" (dev) install of BPS-TUNE™ as python package from your folder into your current active Python environment, so Python uses your local copy.

- Why now: you just pulled new code; this step ensures your environment points at the updated package.

It is worth noting that PowerWorld Simulator should be pre-installed prior to running any PF model creation. BPS-TUNE™ requires PowerWorld Simulator version 24 or later. Once the software is installed, the SimAuto¹¹ API plugin of PowerWorld Simulator must be registered. Detailed instructions on how to register SimAuto can be found at <https://www.powerworld.com/knowledge-base/registering-simauto>.

3.4 Step 3 – Get the Required BPS-TUNE™ Branch (if applicable)

Note. During active development, functionality may temporarily reside on feature branches before being merged into main. The generic branch name `<branch>` used in the example commands is a placeholder. The actual branch to use for each tool is listed in Appendix A (Tool/Branch Mapping). Before running the Git commands, the user should replace `<branch>` with the branch indicated in Appendix A (e.g., `main`, `dataset_comparison`, `pcm_acopf-solver`, or `qgis`).

1. In **Git Bash** (same place as before), run:

```
bash
git checkout -t origin/<branch>
```

- *What it does:* creates a **local branch named cpage** that **tracks** the server's origin/cpage branch, then **checks it out** (switches to it).
- *What you'll see:* a message like "Branch 'cpage' set up to track remote branch 'cpage' from 'origin'."

2. In **cmd** again in VS Code (bottom terminal → arrow → **cmd**), run:

```
bat
pip install -e ".[all]"
```

- *What it does:* another editable install, but with the all **extras** required by the C-PAGE functionality.
- *Why again:* the cpage branch can add dependencies; this makes sure they're available.

3.5 Step 3 – Configuration Template for PCM to PowerWorld

The CPAGEAutomation class is initialized with a TOML configuration file that has the following template color coding:

- **red:** specific for this choice
- **blue:** configurations must be modified based on the user inputs
- **green:** comments

¹¹ SimAuto is a separately licensed add-on to PowerWorld Simulator and is not included in base Simulator licenses. Users must ensure that their installation includes SimAuto access.

- black: other configurations that can be changed by the user)

Detailed explanation of each input parameter is provided in the ReadMe file inside the C-PAGE module.

```

toml

# Where outputs land
# Directory to save the output files
# Default Input: Must be provided.
output_directory = 'C:\Path\To\YourProject\output'

# Mode of operation
# Possible Input: "dc", "ac"
# Default Input: "dc"
# Run in DC mode for this step
mode = "dc"

# Which hours to build (list or "all" or CSV path)
# Index of timestamps
# Possible Input:
# selected hours: list of timestamps ["2035-08-05 22:00:00","2035-08-05
    23:00:00"]
# longer hours list: csv file path
# all hours: string "all"
# Default Input: Must be provided.
# timestamps = "all"

timestamps = ["2035-07-19 17:00:00"]

# PCM input (GridView)
# File path to CPAGE data input file
# Possible Input:
# EGRET data or Excel map file: .json or .json.gz file path
# GridView (GV) data: h5 file path
# Excel data: directory to excel cpage files
# Default Input: Must be provided.
# In this step provide GridView (GV) data: h5 file path

data = 'C:\Path\To\YourProject\Your_PCM_Run.h5'
[cpagesolver]

# Reference PF case with the right topology
# File path to PF case file
# Default Input: Must be provided.

pf = 'C:\Path\To\YourProject\Reference.pwb'

# Special mapping files folder
# Directory to special mapping files between cpage data and cpage solver power
    flow case
# Possible Input: folder path or empty string

maps_directory = 'C:\Path\To\YourProject\maps'

# CPAGE solver configuration settings
[cpagesolver.config.options]

# Flag to enable solving power flow after dispatch system components
solve_flag = true

# Flag to save power flow before solving power flow
save_pre_solve_flag = true          # save a pre-solve snapshot

```

```

# Flag to check if components are properly dispatched or not
validation_flag = true          # sanity checks

# Flag to turn off components before dispatching. This is NOT applied to
# 'shifter' and 'HVDC_VSC'
turn_off_flag = true

# Flag to save cpage xlsx data after loading and mapping
save_cpagedata_flag = true

# Flag to save cpage xlsx data used to solve power flow
save_solverdata_flag = true

# Flag to enforce line shunts limits using power flow data
enforce_lineshunt_limits_flag = true

# Flag to enforce switched shunts limits using power flow data
enforce_switchedshunt_limits_flag = true

# Flag to discretize switched shunts using power flow data
discretize_switchedshunt_flag = true

# Logger settings
[logger_options]
# Logging level
# Possible Input: Logging level as a string, such as 'DEBUG', 'INFO', 'WARNING',
# 'ERROR', 'CRITICAL'
level = 'DEBUG'

# Message format for logs
# Possible Input: A format string for log messages using Python's logging format
# variables.
# Example: '{asctime} - {name} - {levelname} - {message}'
msg_format = '{name}: {levelname}: {message}'

# Path to log file
# Default Input: can be left empty string.
logfile = ''

```

The mapping class CPAGEMapping provides a conversion process to handle special components modeling across different tools. For example, converting bi-directional HVDC lines model in Gridview into un-directional representation in PowerWorld. An HVDC direction map file is needed to remap the HVDC line flows properly. This class is only when mapping files are provided.

The class initializes proper subclasses to handle a specific component mapping based on provided mapping files. Each subclass has its own conversion procedure, which includes:

- **Hydro:** Handles mapping operations for hydro power plants
- **Gen:** Handles mapping operations for generation tables (includes PCM and PF maps)
- **HVDCGen:** Handles mapping operations to convert HVDC lines into equivalent Generators on AC-terminals

- **HVDCDir**: Handles mapping operations to convert bi-directional HVDC line into uni-directional HVDC line
- **HVDCVSC**: Handles mapping operations to convert HVDC line into Voltage source controlled (VSC) HVDC line
- **MTDC**: Handles mapping operations to convert HVDC line into Multi-terminal DC line (MTDC)
- **MTDCGen**: Handles mapping operations to convert MTDC converters into equivalent generators on AC-terminals

A detailed explanation of how to create these map files is provided in Appendix B.

3.6 Step 3 – Executing Automation Script cpageautomation.py

Follow these steps to run the automated cpage script:

- Create TOML configuration file as shown above.
- From Command Prompt in your project folder (with the env active), run the following:

```
bat

conda activate gridtune

cpageautomation "C:\Path\To\YourProject\cpageautomation-YYYY-MM-DD HH-MM-SS.toml"
```

3.7 Step 3 – Output

Figure 3.3 shows the output folder layout and contents.

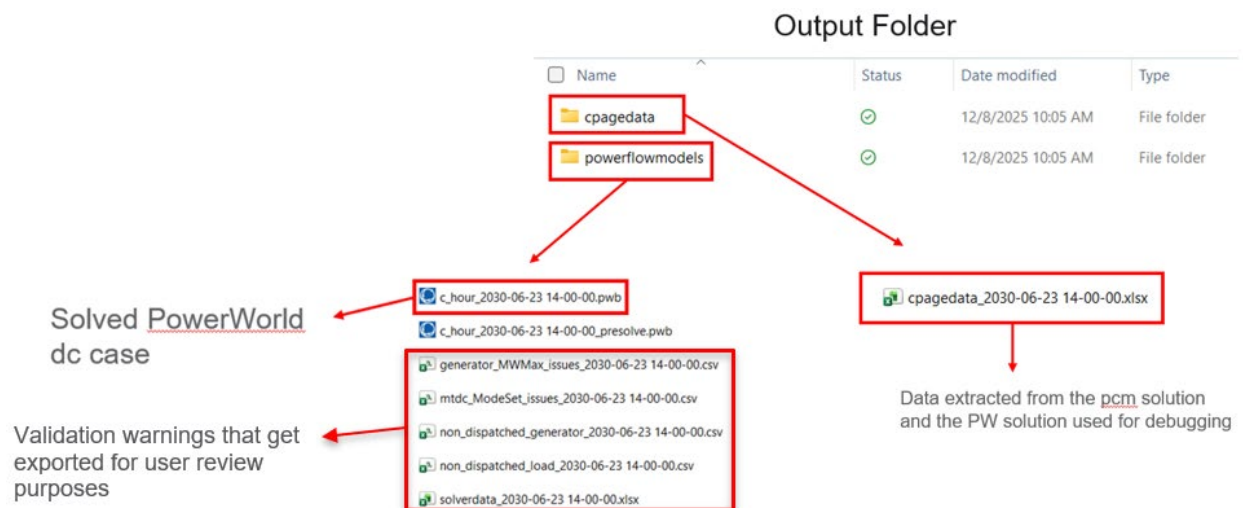


Figure 3.3. Step 3: Output Directory Layout and Results

4.0 Step 4 – DC Cases Benchmarking (PowerWorld .pwb vs PCM)

4.1 Step 4 – Purpose

This step validates that the generated DC power-flow cases in Step 3 (PowerWorld .pwb) are consistent with the PCM (GridView) DC results (stored in the .h5 file). The automation runs comparison on parameter level for each component category and summarizes results in Excel workbooks or h5 file. Each file shows a comparison of all common components between the defined datasets. In addition, a filtering threshold is used to report shortened list of mismatched components, which is fully adjustable by the user.

Outputs (per timestamp, unless aggregated is enabled):

- bus_all.xlsx / bus_filtered.xlsx — e.g., θ (angle); fixed fields like **kV** may appear as “fixed.”
- branch_all.xlsx / branch_filtered.xlsx — e.g., **P** flow comparisons.
- generator_all.xlsx / generator_filtered.xlsx — e.g., **P_g**.
- load_all.xlsx / load_filtered.xlsx — e.g., **P_{load}**.

“_filtered” files contain only rows where the absolute and/or percent error **exceeds** thresholds from the **parameters** configuration.

4.2 Step 4 – Scope and General Capability

The automation compares **results** across multiple dataset formats and for **either DC or AC** physics. Supported dataset types:

- pw — PowerWorld cases (.pwb; directory or file)
- gv — GridView PCM results (.h5; **DC only**)
- egret — EGRET JSON (.json/.json.gz; directory or file)

Supported pairings (examples):

- pw ↔ pw (DC or AC)
- pw ↔ egret (DC or AC)
- egret ↔ egret (DC or AC)
- pw ↔ gv (**DC only**, since GridView is DC)
- egret ↔ gv (**DC only**, when the EGRET export represents DC results)

In this step the focus is **DC benchmarking** (PowerWorld Step 3 outputs vs GridView .h5). The same driver and configuration pattern apply to AC comparisons by selecting appropriate datasets and thresholds.

4.3 Step 4 – Requirements

4.3.1 Tools and Environment

- BPS-TUNE™ installed in the active Conda environment, with the **required** branch checked out locally, if needed (see **Pre-run update** below).
- **EGRET** installed in the same environment **only if** the comparison involves EGRET JSON (e.g., `pw ↔ egret` or `egret ↔ egret`). For pure `pw ↔ gv` (DC) comparisons, EGRET is not required.
- **PowerWorld** is not invoked by this step, but the `.pwb` files being compared must already exist (from Step 3).

4.3.2 Inputs

- PowerWorld DC case(s): `.pwb` case(s) produced by Step 3 (stored in a directory).
- PCM: one `.h5` file produced by Step 2 and corresponding to the same timestamp(s).
- Two TOML files:
 1. **Control TOML** (example: `datasetcomparison_conus.toml`) — names datasets and timestamps and sets output paths.
 2. **Parameters TOML** (example: `datasetcomparison_parameters_config_dc.toml`) — thresholds and save controls.

4.3.3 Entry Point

The console script `datasetcomparisonautomation` is provided by BPS-TUNE™ (or BPS-TUNE™ branch if needed) and becomes available in the environment after `pip install -e .` from the BPS-TUNE™ repo root.

4.4 Step 4 – Pre-Run Update (BPS-TUNE™ → <branch> if applicable)

Note. During active development, functionality may temporarily reside on feature branches before being merged into main. The generic branch name `<branch>` used in the example commands is a placeholder. The actual branch to use for each tool is listed in Appendix A (Tool/Branch Mapping). Before running the Git commands, the user should replace `<branch>` with the branch indicated in Appendix A (e.g., `main`, `dataset_comparison`, `pcm_acopf-solver`, or `qgis`).

Shell: Git Bash (BPS-TUNE™ repo root)

```
bash

git fetch
git checkout <branch> || git checkout -t origin/<branch>
git pull
```

Shell: Command Prompt (BPS-TUNE™ repo root; Conda available)

```
bat  
  
conda activate gridtune  
pip install -e .
```

Optional quick check:

```
bat  
  
datasetcomparisonautomation --help
```

If the command is not found, ensure the environment is activated and the editable install succeeded.

4.5 Step 4 – Configuration Inputs

4.5.1 Comparison Control Configuration

Assuming a file named “datasetcomparison_inputs_configuration.toml”, below is the content of this file to define what to compare (datasets and timestamps) and where to save outputs. Detailed explanation is found on the README inside the datasetcomparison module.

```

# Directory to save the output files
# Default Input: Must be provided.

output_directory = 'C:/<Change this to your path>/Step4/output'

# Which hours to compare (list, "all", or CSV)
# Index of timestamps
# Possible Input:
# selected hours: list of timestamps ["2035-08-05 22:00:00","2035-08-05
    23:00:00"]
# longer hours list: csv file path
# all hours: string "all"
# Default Input: Must be provided.

timestamps = ["2035-07-19 17:00:00"]

# Directory to special mapping files between datasets
# Possible Input: folder path or empty string.
# Optional specialized mapping folder between datasets (leave empty if not used)
maps_directory = ''

# Path to user configuration file for comparison parameters and comparison
    control settings
# Possible Input: file path or empty string.
# Parameters file (thresholds and options)

configuration_file = 'C:/<Change this to your
    path>/Step4/input/datasetcomparison_parameters_config_dc.toml'

# Datasets to compare (keys are arbitrary labels; values define path and type)
[datasets]
# PowerWorld DC cases generated in Step 3 (directory containing .pwb files)
# dictionary of datasets to be compared. Each key represents a single dataset
# Possible Input:
# EGRET data: .json
# GV data: .h5
# PowerWorld data: directory to power flow model files
# Default Input: Must be provided.

pw = { directory = "C:/<Change this to your path>/Step4/input", type = "pw" }

# GridView PCM file (.h5) to benchmark against
gv = { directory = "C:/<Change this to your
    path>/Step4/input/S4_R3_FY25_bnd_pcm_2024_1119a_v0.h5", type = "gv" }

# (Optional) EGRET JSON input can also be compared by adding:
# egret = { directory = "C:/<Change this to your
    path>/Step4/input/egret_outputs", type = "egret" }

```

```

# Logger settings
[logger_options]

# Logging level
# Possible Input: Logging level as a string, such as 'DEBUG', 'INFO', 'WARNING',
#                 'ERROR', 'CRITICAL'
level = 'DEBUG'

# Message format for logs
# Possible Input: A format string for log messages using Python's logging format
#                 variables.
# Example: '{asctime} - {name} - {levelname} - {message}'
msg_format = '{name}: {levelname}: {message}'

# Path to log file
# Default Input: can be left empty string.
logfile = ''

```

Important remarks:

- pw with `type="pw"` expects a **directory**; the automation pairs each case for the selected timestamp(s) against the reference dataset.
- gv with `type="gv"` points to a single **.h5 file** (the PCM source).
- An EGRET dataset can be a directory (many `.json/.json.gz`) or a single file; use `type="egret"`.
- `maps_directory` remains **empty** unless a custom ID crosswalk is required; this step compares results and does not reconcile topology.

Pairing	DC	AC	Notes
pw ↔ pw	✓	✓	Compare PowerWorld cases, directory or single files.
pw ↔ egret	✓	✓	EGRET JSON may be per-hour files or a directory.
egret ↔ egret	✓	✓	JSON ↔ JSON.
pw ↔ gv	✓	—	GridView .h5 is DC-only .
egret ↔ gv	✓	—	Valid when the EGRET side represents DC results.

4.5.2 DC Parameters/Thresholds Configuration

Assuming a file named “datasetcomparison_parameters_config_dc.toml”, below is the content of this file to control what is considered a violation and how results are saved.

```

# comparison options

[options]

# Flag to save comparison for each independent timestamp in a single file
# Save one workbook per timestamp

save_independent_comparison_flag = true

# Flag to save comparison for all timestamps in a single file
# i.e., save one aggregated workbook across all timestamps
save_aggregated_comparison_flag = false

# Global thresholds (used when a component-specific override is absent)
error_absolute_threshold = 20      # error absolute threshold value
error_percentage_threshold = 20    # error absolute percentage threshold value

[components.bus]
fixed = ["kv"]
varying = [
    { name = "theta", error_absolute_threshold = 0.5, error_percentage_threshold
      = 5 }
]

[components.generator]
fixed = []
varying = [
    { name = "pg", error_absolute_threshold = 1, error_percentage_threshold = 10
    },
]

[components.load]
fixed = []
varying = [
    { name = "p_load", error_absolute_threshold = 1, error_percentage_threshold
      = 10 },
]

[components.phaseshifter]
fixed = []
varying = [
    # { name = "phase_angle", error_absolute_threshold = 0.5,
      error_percentage_threshold = 5 }
]

[components.dcline]
fixed = []
varying = [
    # { name = "p_to", error_absolute_threshold = 10, error_percentage_threshold
      = 10 }
]

```

```
[components.mtdc]
fixed = []
varying = [
    # { name = "p_to", error_absolute_threshold = 10, error_percentage_threshold
    = 10 },
]

[components.branch]
fixed = ["status", "from_bus_kv", "to_bus_kv"]
varying = [
    { name = "p", error_absolute_threshold = 10, error_percentage_threshold = 10
    },
]
```

Quick validation checklist:

- Timestamp coverage: Ensure each selected timestamp exists in both datasets (e.g., the `.pwb` directory and the `.h5` PCM file).
- Dataset types: `pw` → directory of `.pwb`; `gv` → one `.h5`; `egret` → `.json` files or directory.
- Paths and quoting: Use absolute paths. In TOML, wrap Windows paths in single quotes (`'C:\...\''`) or use forward slashes (`C:/...`) to avoid escaping backslashes.
- Maps directory: Leave empty unless a known crosswalk is required (for ID/name mismatches).
- PowerWorld version: `.pwb` files must open in the installed Simulator version.

4.6 Step 4 – Execution

Shell: Command Prompt (Conda available)

```
bat

conda activate gridtune
datasetcomparisonautomation "C:\Users\Path\to\Step4\input\datasetcomparison-
conus.toml"
```

4.7 Step 4 – Output

Excel files are written under `output_directory`.

With `save_independent_comparison_flag = true`, a set of `*_all.xlsx` and `*_filtered.xlsx` appears per timestamp.

If `save_aggregated_comparison_flag = true`, an additional consolidated workbook is produced.

Figure 4.1 shows a typical output directory layout and a look at the results.

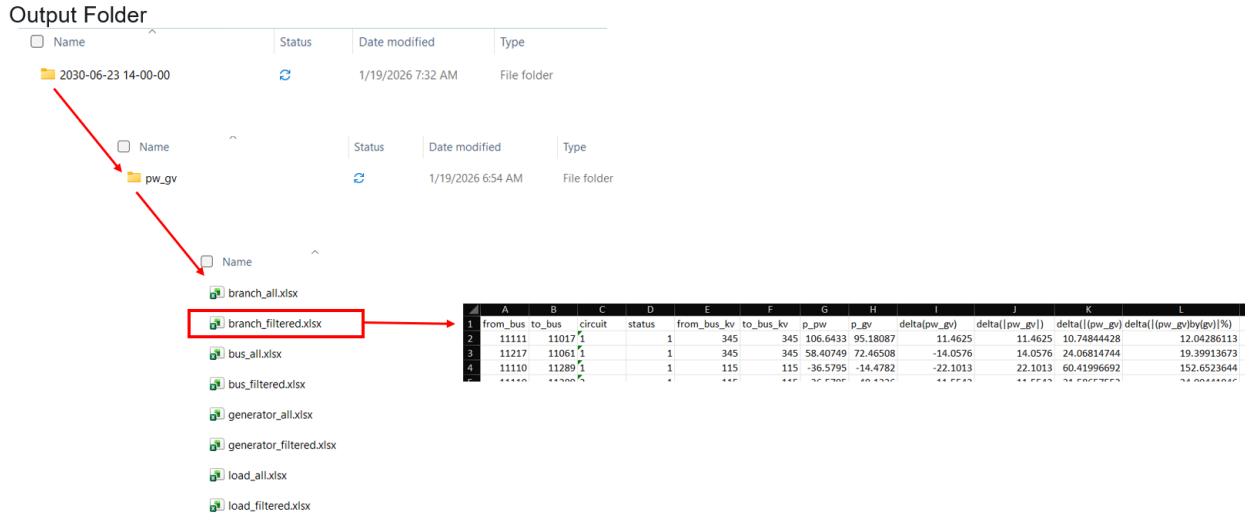


Figure 4.1. Step 4: Output Directory Layout and Results

In an ideal matching, no files will be generated. However, in most of the models some files are exported, particularly bus and generator files. Generator file will show how much difference on the slack bus is between two models in addition to other generators, if any. In an instance in which several files are exported and each file shows many mismatches, detailed debugging on topological and setpoint compatibility is needed. Also, note that these files report the mismatches above the user predefined thresholds. If no files are generated with small threshold tolerance, then the results are acceptable.

4.8 Step 4 – Summary

The user updates BPS-TUNE™ to the required branch if needed, refreshes the editable install, and runs `datasetcomparisonautomation "<control-toml>"`. The tool compares DC results from PowerWorld cases (Step 3 outputs) against the GridView .h5 PCM baseline (and optionally against EGRET JSON), writing `*_all.xlsx` and `*_filtered.xlsx` to the configured output directory. Thresholds and save options are controlled by the parameters TOML.

5.0 Step 5 – Create DC JSON Cases (PowerWorld → EGRET input)

5.1 Step 5 – Purpose

This step converts DC PowerWorld cases (.pwb, one per timestamp) created in Step 3 into DC PF JSON files. These JSON files are inputs EGRET uses in Step 6 to run the AC optimization.

The converter is EGRET's PowerWorld parser package. Although the script is named `make_acopf_from_dcpf.py`, the output here is DC JSON, not an AC case.

5.2 Step 5 – Pre-Run Update Checklist (required once per session)

5.2.1 Update BPS-TUNE™/C-PAGE (Git Bash, BPS-TUNE™ repo root)

```
bash

git fetch
git pull
```

In Command Prompt (BPS-TUNE™ repo root; Conda available)

```
bat

conda activate gridtune
pip install -e .
```

5.2.2 Update EGRET (Git Bash, EGRET repo root)

Note. During active development, functionality may temporarily reside on feature branches before being merged into main. The generic branch name `<branch>` used in the example commands is a placeholder. The actual branch to use for each tool is listed in Appendix A (Tool/Branch Mapping). Before running the Git commands, the user should replace `<branch>` with the branch indicated in Appendix A (for example `main`, `dataset_comparison`, `pcm_acopf-solver`, or `qgis`).

First time on this machine

```
bash

# Git Bash, inside the EGRET repo
git fetch
git checkout -t origin/<branch>
# optional right after creation:
git pull
```

Branch already created earlier (most runs)

```
# Git Bash, inside the EGRET repo
git fetch
git checkout <branch>
git pull
```

5.3 Step 5 – Inputs

5.3.1 User Configuration (example: 2030ADS_to_acpf.toml)

Tells the parser which DC `.pwb` to read, the timestamps, and where to write JSON and log:

- `pwbpath = [ ... ]` — list of absolute paths to DC `.pwb` cases (one per timestamp).
- `time_keys = [ ... ]` — list of timestamps matching `pwbpath` in count and order.
- `pwdconfig = "..."` — path to `pwdconfig.toml` (overrides).
- `savepath = "..."` — output folder for the JSON files.
- `logfile = "..."` — path to a run log (typically inside `savepath`).

Paths with spaces must be quoted. Use forward slashes (`C:/...`) in TOML to avoid escaping.

5.3.2 Parser Overrides (pwdconfig.toml)

Overrides a subset of EGRET's defaults in:

```
<EGRET root>\egret\parsers\pwparser\pwdefaults.toml
```

Typical fields used for DC parsing:

- `mode = "parse"` — create a model from the `.pwb`.
- `[shunts].initial_bs_key = "MvarNom"` — initial shunt susceptance sourced from nominal MVAR (because Mvar is zero in a DC-solved case).
- `[generation].min_voltage_tolerance` — tolerance for regulated bus checks while parsing.
- `[bus].vm_update_key = "EPCVSched", update_va = false` — read scheduled V from EPCV; do not adjust angles in DC parsing.

5.3.3 Templates (ready to paste)

5.3.3.1 2030ADS_to_acpf.toml

```
toml

# DC .pwb case(s) (one path per case)
pwbpath = [
  "C:/Path/to/Step_5/input/pf_case_hour_YYYY-MM-DD HH-00-00.pwb "
  # "C:/Path/to/Step_5/input/pf_case_hour_YYYY-MM-DD HH-00-01.pwb "
  # "C:/Path/to/Step_5/input/pf_case_hour_YYYY-MM-DD HH-00-02.pwb "
]

# Timestamps aligned to pwbpath (same count & order)
time_keys = [
  "2030-01-21 08-00-00",
  # "YYYY-MM-DD HH-00-00",
```

```

]

# Parser overrides file
pwdconfig = "C:/ Path/to/Step_5/input/pwdconfig.toml"

# Output folder and log
savepath = "C:/Path/to/Step_5/output/"
logfile = "C:/Path/to/Step_5/output/make_case.log"

```

5.3.3.2 pwdconfig.toml

```

mode = "parse" # create a new model from DC .pwb

#####
# Component information
#####
[shunts]
initial_bs_key = "MvarNom" # "Mvar" or "MvarNom"; DC uses nominal MVar

[generation]
min_voltage_tolerance = 0.01 # min tolerance for regulated bus voltage

[bus]
vm_update_key = "EPCVSched" # "Vpu" or "EPCVSched"
update_va = false # do not update angles during DC parse

```

5.4 Step 5 – Execution

Shell: Command Prompt (with Conda available)

Run folder: EGRET PowerWorld parser directory :

```
<EGRET root>\egret\parsers\pwparser
```

```
bat
```

```
conda activate gridtune
cd C:\Path\to\egret\egret\parsers\pwparser
python make_acopf_from_dcpf.py
    "C:\Path\to\Step_5\input\2030ADS_to_acpf.toml"
```

5.5 Step 5 – Summary

Step 5 — PowerWorld DC → EGRET JSON.

The user updates BPS-TUNE™/C-PAGE and EGRET, ensures the EGRET required branch is active if needed, then runs `python make_acopf_from_dcpf.py "<config>"` from `egret\parsers\pwparser`. The user-supplied TOML (`2030ADS_to_acpf.toml`) lists DC `.pwb` paths and `time_keys`, points to `pwdconfig.toml` for parser overrides, and specifies `savepath` and `logfile`. The script writes one DC JSON per timestamp to `savepath`, which feeds Step 6.

6.0 Step 6 – Run EGRET ACOPF to produce AC-feasible cases

6.1 Step 6 – Purpose

Step 6 takes the DC power-flow snapshots (JSON files) produced in Step 5 and uses EGRET's `pcm_acopf` extension to compute an AC-feasible operating point for each case.

At a high level, this step:

- Builds an AC network model for each timestamp, using:
 - The network topology and equipment data from the PowerWorld / reference model.
 - The real-power dispatch and flows from the DC JSON (Step 5 output).
- Solves an ACOPF using Pyomo + IPOPT, where the decision variables include:
 - Bus voltage magnitudes and angles.
 - Generator reactive power injections (and, within configured limits, small adjustments to real power if allowed).
 - Switched shunt and other reactive controls, where enabled in the configuration.
 - Load adjustments to account for losses or enforce feasibility, when permitted by the configuration.
- Enforces AC constraints:
 - Power-balance at each bus (P and Q).
 - Voltage magnitude limits.
 - Generator and shunt reactive power limits.
 - Thermal limits and, where modeled, HVDC constraints.
- Produces a converged AC solution written as a JSON results file, structurally similar to the DC JSON but enriched with AC quantities (voltages, reactive power, loss-consistent flows, etc.).

In other words, Step 6 optimizes reactive power and, where configured, small active power/load adjustments so that each DC snapshot becomes a physically consistent AC case that can later be mapped back into a PowerWorld `.pwb` in Step 7.

6.2 Step 6 – Requirements

- **EGRET** installed in the active Conda environment.
- **IPOPT** and core Python packages available in that environment:


```
numpy, pandas, networkx, pyomo, ipopt, and tomli, tomli_w.
```
- Access to the EGRET installation directory, e.g.:


```
C:\Path\to\egret\egret\model_library\extensions\pcm_acopf\
```
- DC JSON case(s) produced in Step 5 (one per timestamp).

- A configuration TOML file specifying:
 - The input JSON path.
 - The output directory for results.
 - Any other settings not overridden here are taken from config/default.toml.

6.3 Step 6 – Pre-Run Update (EGRET and dependencies)

Note. During active development, functionality may temporarily reside on feature branches before being merged into main. The generic branch name `<branch>` used in the example commands is a placeholder. The actual branch to use for each tool is listed in Appendix A (Tool/Branch Mapping). Before running the Git commands, the user should replace `<branch>` with the branch indicated in Appendix A (e.g., `main`, `dataset_comparison`, `pcm_acopf-solver`, or `qgis`).

Shell: Git Bash (EGRET repo root)

```
bash
git fetch
git checkout <branch> || git checkout -t origin/<branch>
git pull
```

Shell: Command Prompt (EGRET repo root; Conda available)

```
bat
conda activate gridtune
pip install -e .
```

Install/refresh required packages using the requirement.txt file to ensure the required packages are up to date:

```
bat
conda activate gridtune
cd /path/to/egret/model_library/extensions/pcm_acopf
pip install -r requirements.txt
```

This ensures:

- All needed Python modules import correctly.
- The **IPOPT** solver binary is available on PATH.

6.4 Step 6 – Directory and Configuration Layout

Within the EGRET installation:

directory layout

```
C:\Path\to\egret\egret\model_library\extensions\pcm_acopf\
  solve.py
  config\
    default.toml
    (user configs can go here, but do not have to)
  input\
    (user DC JSON files for Step 6; can be placed here)
  results\
    (EGRET ACOPF outputs are written here)
```

- `solve.py` is the main script.
- `config/default.toml` contains the default configuration: model type, solver options, and default paths.
- The user can create a simple override config (e.g. `my_config.toml`) that only sets `input_path` and `output_directory`; all other settings are inherited from `default.toml`.

6.5 Step 6 – Basic Usage (single-case run)

Shell: Command Prompt

Folder: `C:\Path\to\egret\egret\model_library\extensions\pcm_acopf\`

1. Test with the default configuration (sanity check)

```
bat

conda activate <ENV_USED_FOR_STEP6>
cd C:\Path\to\egret\egret\model_library\extensions\pcm_acopf
python solve.py config/default.toml
```

Here, `default.toml` already defines some input and output paths. This verifies that EGRET and IPOPT are wired correctly.

2. Create a simple project-specific configuration (example: `my_config.toml`)

The user placed this file directly in the `pcm_acopf` directory:

```
toml

input_path      = "input/pf_case_hour_2030-01-21 08-00-00.json"
output_directory = "results"
```

- `input_path` is the relative path (from the working directory) to the DC JSON exported in Step 5.
- `output_directory` is where EGRET writes the AC results JSON.
- All other settings (solver, model type, tolerances, allowed controls) are taken from `config/default.toml`.

Note: The user copied the DC JSON (e.g. `pf_case_hour_2030-01-21 08-00-00.json`) into the input subdirectory:

```
C:\Path\to\egret\egret\model_library\extensions\pcm_acopf\input\pf_case_hour_2030-01-21 08-00-00.json
```

3. Run EGRET ACOPF with this configuration

```
bat

conda activate <ENV_USED_FOR_STEP6>
cd C:\Path\to\egret\egret\model_library\extensions\pcm_acopf
python solve.py my_config.toml
```

Expected behavior

- EGRET reads the DC JSON at `input_path`.
- It builds and solves the ACOPF model.
- It writes an **AC-feasible JSON** to `output_directory`, typically preserving the base name (e.g. `results\pf_case_hour_2030-01-21 08-00-00.json`).
- Logs and console output indicate solver status (e.g. IPOPT convergence, objective value, possible constraint violations).

Figure 6.1 shows the run directory layout and path, results folder, and the README.md. The README file provides more details about the PCM-ACOPF process, settings, configurations, etc. For a better view, open the file in the VS Code → right-click on the README file tab → Open Preview.

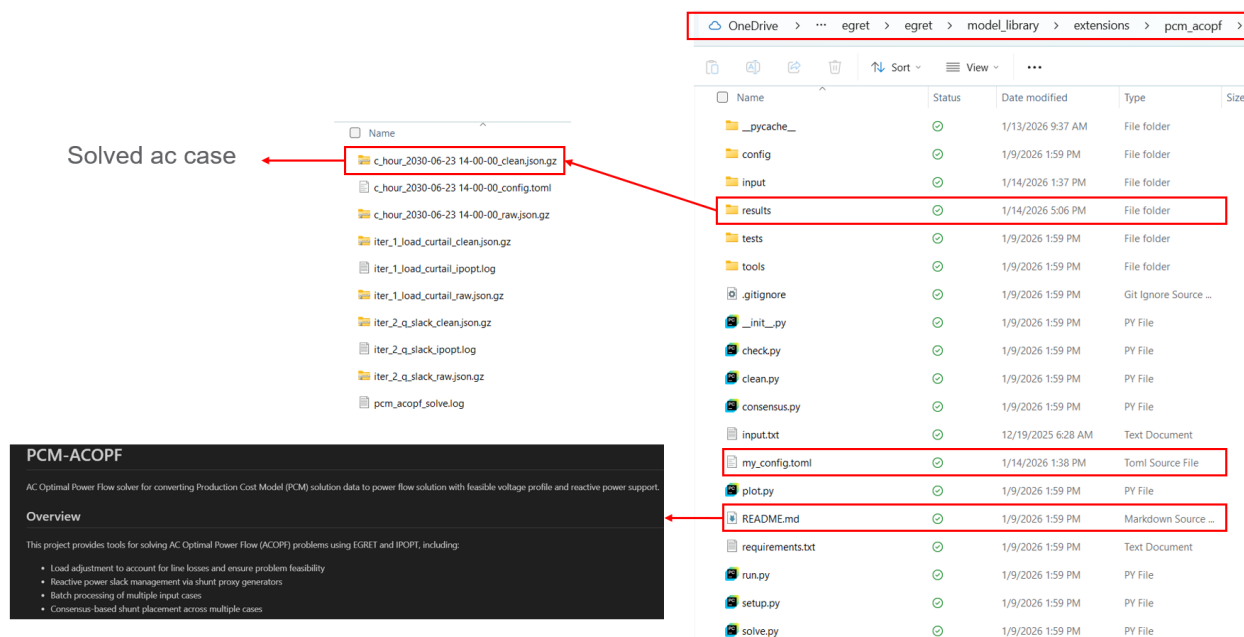


Figure 6.1. Step 6: Run Directory Layout and Path, Results Folder, and the README.md

6.6 Step 6 – Summary

Step 6 — DC JSON → AC-feasible JSON (EGRET ACOPF).

The user updates EGRET to the `pcm_acopf-solver` branch, installs EGRET into the active Conda environment, and ensures the required packages are present (`numpy`, `pandas`, `networkx`, `pyomo`, `ipopt`, `tomli`, `tomli_w`).

From `...\egret\egret\model_library\extensions\pcm_acopf\`, the user runs:

```
bat

conda activate gridtune
cd C:\Path\to\egret\egret\model_library\extensions\pcm_acopf
python solve.py my_config.toml
```

where `my_config.toml` can either:

```
toml

# Single-case mode
input_path      = "input/pf_case_hour_YYYY-MM-DD HH-00-00.json"
output_directory = "results"
```

or

```
toml

# Multi-case mode (directory)
input_path      = "input"          # directory containing multiple DC JSON cases
output_directory = "results"
```

EGRET reads one DC JSON from Step 5 and writes an AC-feasible JSON with the same base name under `results\`. EGRET runs an ACOPF, adjusting **reactive power** (and small load/dispatch changes if configured) to obtain AC-feasible operating points, which are then used in **Step 7** to construct converged PowerWorld AC `.pwb` cases.

7.0 Step 7 – Build AC Power Flow Cases from EGRET Output

7.1 Step 7 – Purpose

Build an AC power-flow snapshots from EGRET ACOPF solution ensuring that components dispatch set points, PFs, and voltage profile match the EGRET solution (see Section 4.1 and Figure 4.1). In this step, cpage module will use the EGRETData parser to prepare data received from EGRET (.json) file properly. The process involves updating bus; voltages and angles, generator; P,Q, and Vreg, load; P,Q, phase-shifter angles, HVDC; real power setpoint and converter reactive power compensation, and shunt reactive setpoints onto a reference PF model of the same topology. The cpagemapping will not be needed because there is no special component mapping from EGRET to PowerWorld. The output is a solved AC .pwb case for each requested hour. (Report Sec. 4.2.6, pp. 29–30.). It is worth noting that several reconciliation tasks take place as follows:

- High-Voltage DC lines (proxy generator → unidirectional branch): EGRET treats HVDC as two independent generators at the AC-bus terminals. On the other hand, the PF model contains explicit DC converters and branches. It is essential to map each two equivalent generator back to their corresponding DC branch flow. no special mapping files are needed since all information reside the egret (.json) file.
- High-Voltage DC converter losses (proxy generator → switched shunt): EGRET assumes that each DC converter can supply reactive power up to 0.9 power factor of their real power capacity. This reactive power has to be converted into new shunt components in the PF model with the proper setpoints.
- Switched shunts (continuous → discrete): EGRET assumes all reactive shunts have continuous nature to reduce the complexity of the ACOPF problem into linear formulation, and hence proper discretization is applied to revert the shunt setpoints to actual steps.

7.2 Step 7 – Preparation

Before running, ensure BPS-TUNE™/C-PAGE is current and installed into the active Conda environment. Refer to Step 3's "Update & Install" instructions in Sections 3.2 and 3.3 (Git Bash: `git fetch`, `git pull`, `git checkout -t origin/<branch>`; Command Prompt: `pip install -e .`). No additional setup is required specifically for Step 7.

7.3 Step 7 – Configuration Template for EGRET to PowerWorld

What changes relative to Step 3:

- `mode = "ac"` (not `"dc"`).
- `data = '...EGRET_solution.json'` (or `.json.gz`).
- `maps_directory = ""` (left **empty**; mapping module is **not** called for this path).

Important configuration changes from Step 3 are marked in red as shown in the .toml template below.

Key assumption: the reference .pwb used here shares the same network topology as the EGRET solution (ideally the DC snapshot's case for that hour, but any topologically identical PF case is acceptable).

```

toml

# Where outputs land
# Directory to save the output files
# Default Input: Must be provided.
output_directory = 'C:\ <Change this to your path>\output'

# Mode of operation
# Possible Input: "dc", "ac"
# Default Input: "dc"
# Run in AC mode for this step
mode = "ac"

# Which hours to build (list or "all" or CSV path)
# Index of timestamps
# Possible Input:
# selected hours: list of timestamps ["2035-08-05 22:00:00","2035-08-05
    23:00:00"]
# longer hours list: csv file path
# all hours: string "all"
# Default Input: Must be provided.
# timestamps = "all"

timestamps = ["2035-07-19 17:00:00"]

# PCM input (GridView)
# File path to CPAGE data input file
# Possible Input:
# EGRET data or Excel map file: .json or .json.gz file path
# GridView (GV) data: h5 file path
# Excel data: directory to excel cpage files
# Default Input: Must be provided.
# In this step provide GridView (GV) data: h5 file path

data = 'C:\<Change this to your path>\EGRET_Solution.json'
[cpagesolver]

# Reference PF case with the right topology
# File path to PF case file
# Default Input: Must be provided.

pf = 'C:\<Change this to your path>\Reference.pwb'

# Special mapping files folder
# Directory to special mapping files between cpage data and cpage solver power
    flow case
# Possible Input: folder path or empty string

maps_directory = ' ' # Empty in this Step

# CPAGE solver configuration settings
[cpagesolver.config.options]

# Flag to enable solving power flow after dispatch system components
solve_flag = true

# Flag to save power flow before solving power flow
save_pre_solve_flag = true          # save a pre-solve snapshot

```

```

# Flag to check if components are properly dispatched or not
validation_flag = true          # sanity checks

# Flag to turn off components before dispatching. This is NOT applied to
# 'shifter' and 'HVDC_VSC'
turn_off_flag = true

# Flag to save cpage.xlsx data after loading and mapping
save_cpagedata_flag = true

# Flag to save cpage.xlsx data used to solve power flow
save_solverdata_flag = true

# Flag to enforce line shunts limits using power flow data
enforce_lineshunt_limits_flag = true

# Flag to enforce switched shunts limits using power flow data
enforce_switchedshunt_limits_flag = true

# Flag to discretize switched shunts using power flow data
discretize_switchedshunt_flag = true

# Logger settings
[logger_options]
# Logging level
# Possible Input: Logging level as a string, such as 'DEBUG', 'INFO', 'WARNING',
# 'ERROR', 'CRITICAL'
level = 'DEBUG'

# Message format for logs
# Possible Input: A format string for log messages using Python's logging format
# variables.
# Example: '{asctime} - {name} - {levelname} - {message}'
msg_format = '{name}: {levelname}: {message}'

# Path to log file
# Default Input: can be left empty string.
logfile = ''

```

7.4 Step 7 – Executing Automation Script cpageautomation.py

The `cpageautomation` is executed in the command prompt as below

```

bat

conda activate gridtune
cpageautomation "C:\Path\To\Project\cpageautomation-YYYY-MM-DD HH-MM-SS.toml"

```

where “cpageautomation-YYYY-MM-DD HH-MM-SS.toml” is the TOML configuration file name. cpageautomation loads the EGRET file, selects EGRETData, skips cpagemapping, and uses PWSolver to imprint values on the reference case. Buses, generators, loads, phase shifters, HVDC, and shunts are updated in order; HVDC terminal Q is computed considering converter losses; shunt setpoints are discretized to available steps. The resulting .pwb is saved under output_directory.

7.5 Step 7 – Output

Inside the designated output directory, two subdirectories will be created “cpagedata” and “powerflowmodels”, as shown in Figure 7.1.

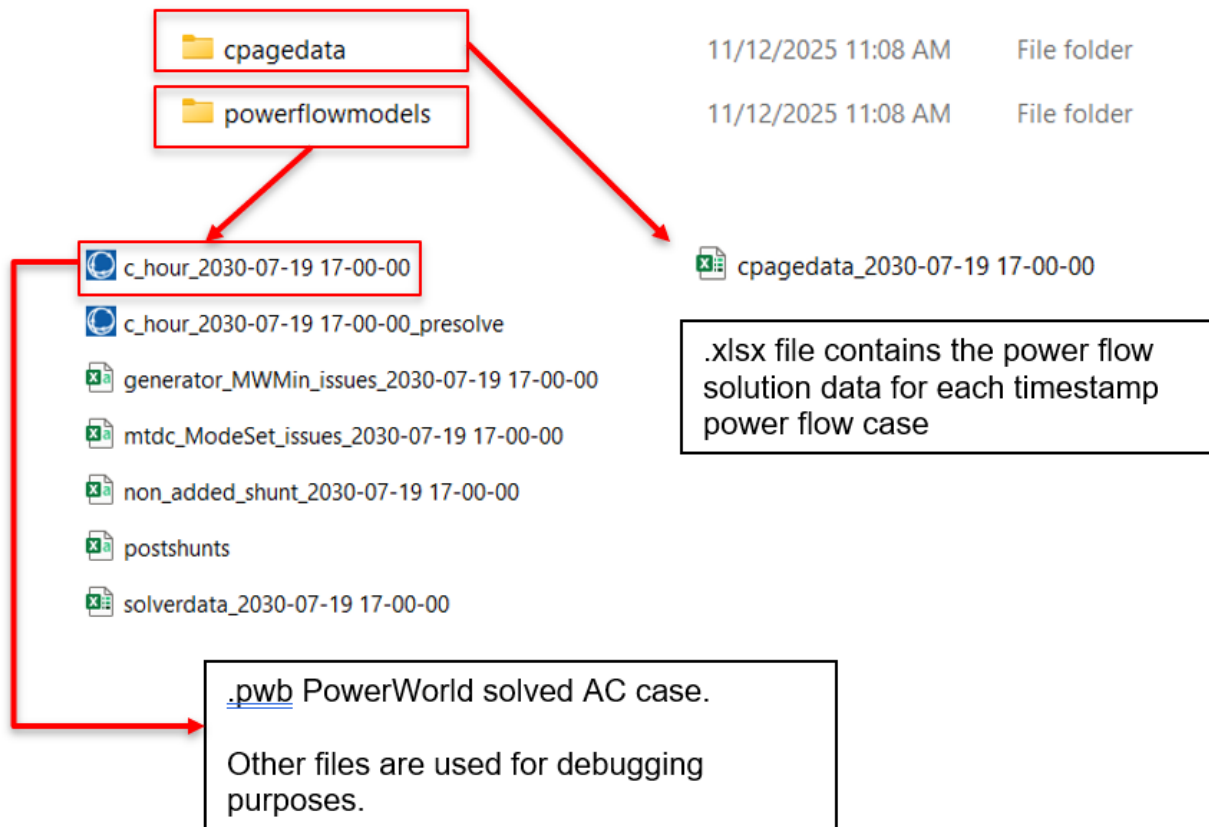


Figure 7.1. File Structure Created Inside the Designated Output Directory

8.0 Step 8 – AC Cases Benchmarking (PowerWorld .pwb vs EGRET .json)

8.1 Step 8 – Purpose

This step verifies that the converged AC PowerWorld case produced in Step 7 (.pwb) is consistent with the EGRET AC solution (.json) produced in Step 6 for the same timestamp(s). The comparison focuses on AC results consistency, including (as configured):

- Bus voltage magnitude (**vpu**) and angle (**theta**)
- Generator active/reactive output (**pg**, **qg**)
- Load active/reactive demand (**p_load**, **q_load**)
- Shunt status and reactive injection (when modeled)
- Branch active/reactive flow (**p**, **q**)

The process generates Excel workbooks for **all elements** and a **filtered subset** of elements whose differences exceed user-defined thresholds.

Important Note: **This step reuses Step 4 (Section 4.0) and only changes dataset types and thresholds**

8.2 Step 8 – How Step 8 Differs from Step 4 (reference)

Step 8 uses the same automation driver as Step 4 (datasetcomparisonautomation) and the same two-TOML design, but changes:

- **Datasets:** pw ↔ egret (instead of pw ↔ gv)
- **Physics:** AC comparison (instead of DC)
- **Threshold file:** an AC **parameters config** (includes vpu, q, qg, q_load, etc.)

Everything else (e.g., outputs, filtering concept, execution style) remains the same as Step 4 (Section 4.0).

8.3 Step 8 – Requirements

8.3.1 Tools and Environment

- BPS-TUNE™ installed in the active Conda environment, with the dataset_comparison branch available (driver is provided by BPS-TUNE™).
- EGRET installed in the same environment (required because dataset type egret is used to read .json cases).

8.3.2 Inputs

- PowerWorld AC case(s): .pwb case(s) produced by Step 7 (stored in a directory).

- EGRET AC solution JSON: one `.json` file (or a directory of `.json` files) produced by Step 6 and corresponding to the same timestamp(s).
- Two TOML files:
 1. Main comparison configuration (Step 8 control file)
 2. AC thresholds/parameters configuration

8.4 Step 8 – Pre-Run Update (BPS-TUNE™ and dependencies)

Note. During active development, functionality may temporarily reside on feature branches before being merged into main. The generic branch name `<branch>` used in the example commands is a placeholder. The actual branch to use for each tool is listed in Appendix A (Tool/Branch Mapping). Before running the Git commands, the user should replace `<branch>` with the branch indicated in Appendix A (e.g., `main`, `dataset_comparison`, `pcm_acopf-solver`, or `qgis`).

- **Shell:** Git Bash (BPS-TUNE™ repository root)

```
bash

git fetch
git checkout <branch> || git checkout -t origin/<branch>
git pull
```

- **Shell:** Command Prompt (BPS-TUNE™ repository root)

```
bat

conda activate gridtune
pip install -e .
```

If comparing with EGRET JSON, ensure EGRET is installed in the same environment (from the EGRET repo root).

8.5 Step 8 – Configuration Inputs

8.5.1 Comparison Control Configuration

Assuming a file named “datasetcomparison_inputs_configuration.toml”, below is the content of this file to define what to compare (datasets and timestamps) and where to save outputs. This file is the same `toml` structure used in Step 4. The only required changes are:

- Replace the `gv` dataset block with an egret dataset block.
- Point `configuration_file` to the **AC** parameters TOML.

Example dataset block (Step 8):

```
toml

[datasets]
pw    = { directory = 'C:\Users\mahm135\OneDrive -
PNNL\Projects\POWER_WORLD_SIMULATOR\GridTune_Projects\NTP_1\Step8\input',
type = "pw" }
egret = { directory = 'C:\Users\mahm135\OneDrive -
PNNL\Projects\POWER_WORLD_SIMULATOR\GridTune_Projects\NTP_1\Step8\input\pf
_case_hour_2030-07-19 17-00-00_clean.json', type = "egret" }
```

Notes:

- pw typically points to a directory of `.pwb` files (one per timestamp).
- EGRET can point to a single `.json` file (as shown) or a directory containing multiple `.json` files.
- Ensure the `timestamps = [...]` value in the control TOML matches the case(s) being compared.

8.5.2 AC Parameters/Thresholds Configuration

Assuming a file named “datasetcomparison_parameters_config_ac.toml,” below is the content of this file to control what is considered a violation and how results are saved. This is an AC analogue of the Step 4 DC thresholds file.

Template Example used for AC comparisons:

```

toml

[options]
save_independent_comparison_flag = true
save_aggregated_comparison_flag = false
error_absolute_threshold = 20
error_percentage_threshold = 20

[components.bus]
fixed = ["kv"]
varying = [
    { name = "vpu", error_absolute_threshold = 0.1, error_percentage_threshold =
      10 },
    { name = "theta", error_absolute_threshold = 0.5, error_percentage_threshold =
      5 }
]

[components.generator]
fixed = []
varying = [
    { name = "pg", error_absolute_threshold = 1, error_percentage_threshold = 10
    },
    { name = "qg", error_absolute_threshold = 1, error_percentage_threshold = 10 }
]

[components.load]
fixed = []
varying = [
    { name = "p_load", error_absolute_threshold = 1, error_percentage_threshold =
      10 },
    { name = "q_load", error_absolute_threshold = 1, error_percentage_threshold =
      10 }
]

[components.phaseshifter]
fixed = []
varying = [
    # { name = "phase_angle", error_absolute_threshold = 0.5,
      error_percentage_threshold = 5 }
]

[components.dcline]
fixed = []
varying = [
    # { name = "p_to", error_absolute_threshold = 10, error_percentage_threshold =
      10 }
]

[components.mtdc]
fixed = []
varying = [
    # { name = "p_to", error_absolute_threshold = 10, error_percentage_threshold =
      10 }
]

```

```
[components.shunt]
fixed = []
varying = [
  { name = "status", error_absolute_threshold = 1, error_percentage_threshold =
    10 },
  { name = "qg",      error_absolute_threshold = 1, error_percentage_threshold =
    10 }
]

[components.branch]
fixed = ["status", "from_bus_kv", "to_bus_kv"]
varying = [
  { name = "p", error_absolute_threshold = 10, error_percentage_threshold = 10
    },
  { name = "q", error_absolute_threshold = 10, error_percentage_threshold = 10 }
]
```

Quick validation checklist:

- The PowerWorld `.pwb` case corresponds to the same timestamp as the EGRET `.json` case.
- The control TOML points to the correct `pw` directory and `egret` file (or directory).
- The control TOML points to the AC parameters file (not the DC thresholds file).
- If most rows are flagged:
 - Thresholds may be tight for the study case.
 - The `.pwb` and `.json` are not from the same timestamp/case name.

8.6 Step 8 – Execution

Shell: Command Prompt (Conda available)

```
bat

conda activate gridtune
datasetcomparisonautomation "C:\path\to\Step8\input\datasetcomparison-
step8.toml"
```

(Replace `datasetcomparison-step8.toml` with the actual Step 8 control TOML name.)

8.7 Step 8 – Output

The automation writes Excel workbooks under `output_directory` (from the Step 8 control TOML), typically including:

- `bus_all.xlsx`, `bus_filtered.xlsx`
- `branch_all.xlsx`, `branch_filtered.xlsx`
- `generator_all.xlsx`, `generator_filtered.xlsx`
- `load_all.xlsx`, `load_filtered.xlsx`
- (If enabled/available) shunt-related workbooks

Filtered workbooks contain only rows exceeding the configured AC thresholds.

8.8 Step 8 – Summary

Step 8 — AC Benchmarking (PowerWorld vs EGRET)

The user runs `datasetcomparisonautomation` with the Step 8 control TOML pointing to `pw` (PowerWorld `.pwb`) and `egret` (EGRET AC solution `.json`). An AC threshold file (`datasetcomparison_parameters_config_ac.toml`) enables comparison of voltage magnitude/angle, P/Q flows, generator P/Q, load P/Q, and shunt behavior. The tool writes Excel reports for all elements and those exceeding thresholds, supporting validation of Step 7's `.pwb` conversion versus the EGRET AC solution.

9.0 Step 11 – Visualization

9.1 Step 11 – Purpose

This step creates **QGIS map projects** that visualize power-flow data (buses, branches, heatmaps, and supporting layers) from **PowerWorld / PSSE** cases processed by BPS-TUNE™. The workflow uses:

1. A **data extraction stage** (`make_qgis_data`) to:
 - Read power-flow cases (`.pwb`, etc.).
 - Push standardized bus/branch data into a **PostgreSQL/PostGIS database** and/or export local **CSVs**.
2. A **project builder stage** (`build_qgz`) to:
 - Read a **TOML configuration** (`my_qgis_config.toml`).
 - Build a QGIS project (`.qgz`) with pre-configured layers, symbology, base maps, and optional voltage heatmaps.

The user then opens the generated `.qgz` in QGIS for interactive exploration.

Path covered in this manual:

This step documents **only the Python CLI scripts** (the “Python CLI (scripts)” workflow). The QGIS plugin (graphical user interface) is out of scope here.

9.2 Step 11 – Requirements

- **QGIS**: Recommended QGIS 3.40 LTR (or the project-standard LTR build). QGIS is a third-party tool (not part of BPS-TUNE™) and must be installed separately from <https://qgis.org/>
- **Python**: Recommended Python 3.12 (64-bit) in the Conda environment used for BPS-TUNE™ QGIS integration. The Python version of the active environment must be **compatible with QGIS’s embedded Python**. For QGIS 3.40 LTR, this is Python 3.12.
- **BPS-TUNE™**: The **qgis branch** of the BPS-TUNE™ repository must be checked out and installed with **extras** (the “all” extra set):

```
pip install -e ".[all]"
```
- **Database**:
 - A PostgreSQL database is used as the repository for GIS data and must be installed and configured according to its manual. The following extensions must be installed: `postgis`, `postgis_raster`.
 - Access credentials to the **internal PostgreSQL/PostGIS database** used for this project (host, port, database, username, password, schema).
 - Standard network/VPN access, if required.

9.3 Step 11 – Interfaces in This Step

BPS-TUNE™ for QGIS exposes two main interfaces:

1. **QGIS plugin (graphical user interface)** – for GIS users working inside QGIS (not covered here).
2. **Python CLI (scripts)** – for power-system engineers and automation pipelines:
 - `make_qgis_data` – process power-flow cases and write to DB and/or CSV.
 - `build_qgz` – build QGIS projects from TOML configuration.

This step documents the **Python CLI** path only.

9.4 Step 11 – Pre-Run Setup (BPS-TUNE™ qgis branch and environment)

9.4.1 Ensure the Environment Uses a Compatible Python Version

If not already done, create and use with Python 3.12 (example name):

Shell: Command Prompt

```
bat

conda create -n gridtune python=3.12 -y
conda activate gridtune
```

9.4.2 Update BPS-TUNE™ to the qgis Branch

Note. During active development, functionality may temporarily reside on feature branches before being merged into main. The generic branch name `<branch>` used in the example commands is a placeholder. The actual branch to use for each tool is listed in Appendix A (Tool–Branch Mapping). Before running the Git commands, the user should replace `<branch>` with the branch indicated in Appendix A (e.g., `main`, `dataset_comparison`, `pcm_acopf-solver`, or `qgis`).

Shell: Git Bash (BPS-TUNE™ repo root)

```
bash

git fetch
git checkout <branch> || git checkout -t origin/<branch>
git pull
```

9.4.3 Install BPS-TUNE™ with QGIS Extras into this Env

Shell: Command Prompt (BPS-TUNE™ repo root)

```
bat

conda activate gridtune
pip install -e "[all]"
```

9.5 Configuration Files and Where to Find Them

BPS-TUNE™ ships example and default configuration files inside the repository:

- **QGIS project config template:**
src/gridtune/qgis/example/my_qgis_config.toml
(basic QGIS project configuration structure).
- **Database default configuration:**
src/gridtune/qgis/example/my_db_config.toml
- **Comprehensive example with all options:**
tests/tests_database/test_user_config.toml

In addition, the **HTML documentation** (see Chapter 2, Additional Documentation) includes:

- A “**Config Files**” section with:
 - Parameter descriptions for QGIS config TOMLs (project builder).
 - Parameter descriptions for database config TOMLs.
- Links or paths to downloadable example TOML files.

In the manual, a screenshot of the HTML docs main page with the QGIS/Config Files section circled can be inserted here for orientation.

9.6 Database Configuration TOML (my_db_config.toml)

The database config TOML describes how BPS-TUNE™ should connect to the PostgreSQL/PostGIS database and which **mapping region** to use.

Example:

```
toml

[[database]]
name      = "ntp"          # ties to mapping entries in /support/db/_mapping (e.g.
                        ntp, pr)
host      = ""             # database host
port      = 5432
database  = ""             # database name
username  = ""
password  = ""
schema    = ""

[[view_settings]]
mapping_region = "wecc"    # region mapping (e.g., "wecc", "ei", "conus")
```

Key points for the user:

- name selects a **mapping profile** (e.g. 'ntp' or 'pr') that determines how power-flow fields map into DB columns (see repo's `_mapping` folder).
- mapping_region controls which region mapping is used (e.g. 'wecc', 'ei', 'conus'); this affects view creation and geometry columns.

- All connection fields (host, database, username, password, schema) must match the internal database configuration provided by the project.

If the user only wants **local CSV output** and no database writes, this file is **not required** (see Section 9.7 notes).

9.7 Data Extraction: make_qgis_data

This script reads PowerWorld/PSSE power-flow cases and writes:

- To a **database** (using `my_db_config.toml`), and/or
- To **local CSVs** (`-l`)

Usage (conceptual):

- `-f` ⇒ single case file path.
- `-d` ⇒ directory of multiple case files.
- `-c` ⇒ path to DB config TOML.
- `-l` ⇒ also write local CSVs.

Example: single case, DB + local CSVs

```
bat
```

```
conda activate gridtune_qgis
cd C:\Path\To\CaseDirectory
```

```
bat
```

```
make_qgis_data -f .\my_power_flow_case.pwb -c .\my_db_config.toml -l
```

Notes:

- `-f` and `-d` are mutually exclusive.
- `-c` and `-l` can be combined:
 - `-c` → write to DB.
 - `-l` → write local CSVs under the case directory
- CSVs written by `-l` do **not** contain geometry columns; geometry is added via DB views when using the database path.

9.8 QGIS Project Configuration TOML (my_qgis_config.toml)

This file controls how the QGIS project is built: base maps, bus/branch layers, heatmaps, group structure, and database bindings.

Example (from the README):

```

toml

original_qgz_file = ''          # existing .qgz to modify (optional)
output_qgz_file   = 'test.qgz' # full path preferred
crs               = "EPSG:4326" # WGS84
database_config   = 'my_db_config.toml' # DB connection info

[[add]]
  layer_type      = 'QgsRasterLayer'
  layer_name      = 'ESRI Gray (light)'
  visible         = true
  base_map        = true

[[add]]
  layer_type      = 'QgsVectorLayer'
  layer_name      = 'bus'
  layer_data_provider_name = 'postgres'
  layer_source    = 'table'
  key             = 'bus'
  geom            = "geom"
  table           = 'dcat.<case_name>_bus_view' # replace <case_name>
  database        = 'ntp'
  layer_load_options = ''
  group_name      = 'bus'
  split_field     = ''
  build_heatmap   = true # enable heatmap creation
  z_field         = 'vm_pu'
  heatmap_filter  = 'kv>200 and status=1'
  visible         = true
  show_feature_count = true

[[add]]
  layer_type      = 'QgsVectorLayer'
  layer_name      = 'branch'
  layer_data_provider_name = 'postgres'
  layer_source    = 'table'
  key             = '_uid_'
  geom            = "geom"
  table           = 'dcat.<case_name>_branch_view' # replace <case_name>
  database        = 'ntp'
  layer_load_options = ''
  group_name      = 'branch'
  split_field     = ''
  filter          = 'from_kv>200 and to_kv>200'
  style           = 'branch_rate_a.qml'
  show_feature_count = true

[[add]]
  layer_type      = 'DefinitionLayers'
  layer_name      = 'general'
  layer_source    = 'general.qlr'
  layer_data_provider_name = 'gridtune'
  group           = 'general'
  visible         = true

```

```
[[group]]
  name = '<case_name>'      # root group for this case

[[group]]
  name = 'branch'
  parent = '<case_name>'

[[group]]
  name = 'bus'
  parent = '<case_name>'
```

Key items the user must adjust:

- `output_qgz_file` – where the new QGIS project is written.
- `database_config` – path to the DB config TOML file.
- `<case_name>` placeholders — replaced with the actual power-flow case name used in `make_qgis_data` (so that table names match, e.g. `dcat.case123_bus_view`).
- Layer styling and filters:
 - `build_heatmap`, `z_field`, `heatmap_filter` for voltage heatmaps.
 - style for branch symbology (e.g. `branch_rate_a.qml`).

9.9 Build the QGIS Project: `build_qgz`

Once the DB and config TOMLs are prepared, and data is present in the database (and optionally CSVs), the user builds the QGIS project:

Shell: Command Prompt

```
bat

conda activate gridtune_qgis
cd C:\Path\To\WorkingDirectory

build_qgz -c .\my_qgis_config.toml
```

On success:

- A QGIS project file (e.g. `test.qgz`) is created in the working directory.
- The user opens `test.qgz` in QGIS 3.40 LTR and sees:
 - Base map(s).
 - Bus and branch layers (drawn from database views).
 - Definition layers and groups, including optional voltage heatmaps.

An example for a voltage heatmap created using this tool, and displayed in QGIS, for one operation instant of the WECC grid is shown in Figure 9.1.

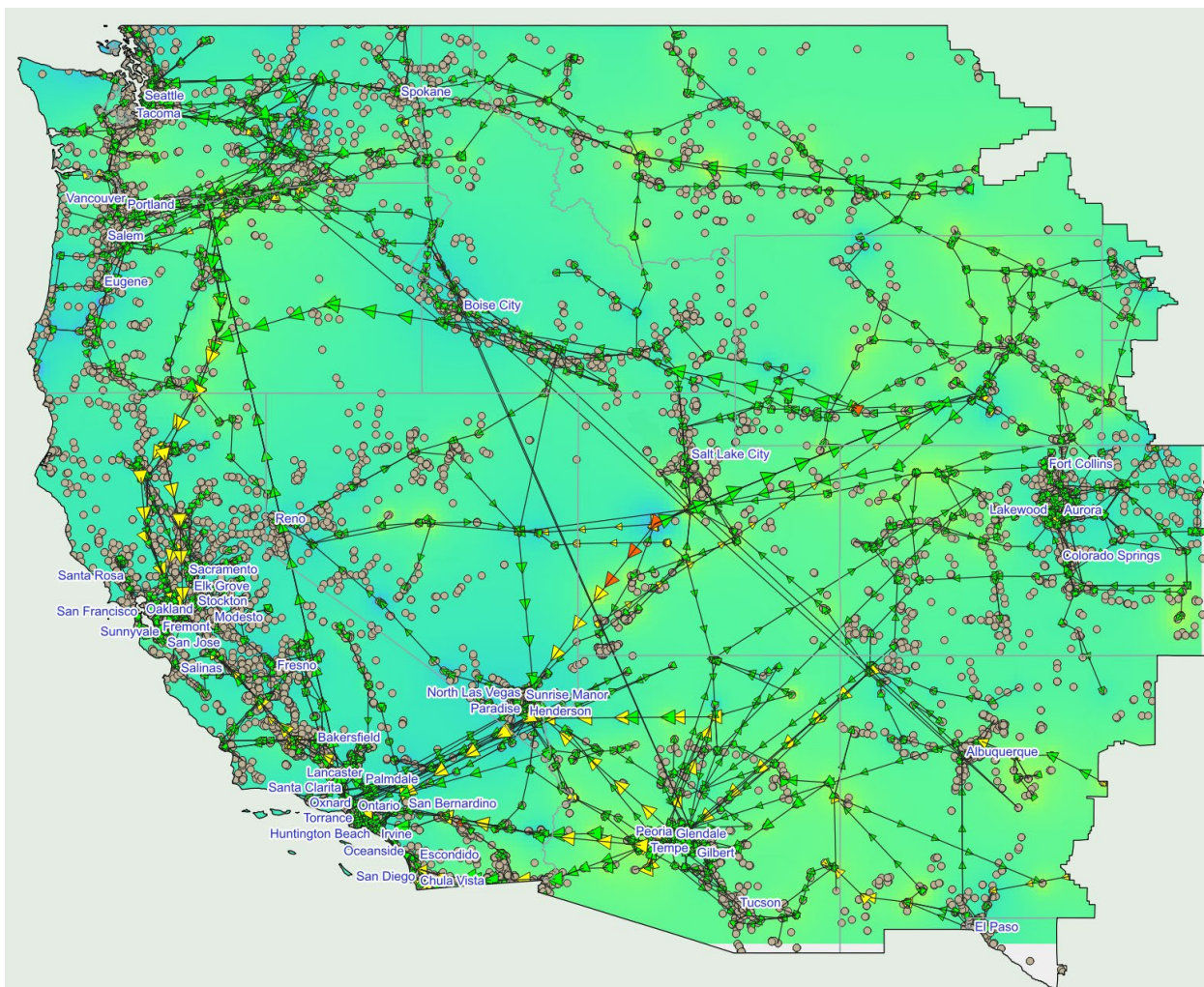


Figure 9.1. Voltage heatmap created using visualization tool, and displayed in QGIS software, for one operation instant of the WECC grid.

9.10 Where to Find Detailed Help and Options

For **full parameter lists** and advanced options:

- Open the BPS-TUNE™ **HTML documentation** (see Chapter 2, Additional Documentation).
- In the left-hand navigation tree, locate the QGIS section and the **Config Files** subsection for:
 - QGIS project configuration TOMLs.
 - Database configuration TOMLs.
 - CLI usage (`make_qgis_data`, `build_qgz`), including flags and edge cases.

The user can also examine the following as concrete, working examples:

- `src/gridtune/qgis/config_files/example_config.toml`
- `src/gridtune/database/config_files/default_db_config.toml`
- `tests/tests_database/test_user_config.toml`

In the manual, the author may include screenshots of the HTML docs with relevant sections circled to guide the user.

Appendix A – Tools/Branch Mapping

Table A.1: Tools/Branch Mapping

Step	Tool	Command	Branch	Python Version
Step 2: Convert PCM to H5	BPS-TUNE™	gv_make_h5	main	3.10, 3.11
Step 3: DC Powerflow	BPS-TUNE™	cpageautomation	main	3.10, 3.11
Step 4: DC Benchmarking	BPS-TUNE™	datasetcomparisonautomation	main	3.10, 3.11
Step 5: EGRET Inputs	EGRET	make_acopf_from_dcpf	pwdparser	3.10, 3.11
Step 6: EGRET ACOPF	EGRET	solve.py	pcm-acopf-solver	3.10, 3.11
Step 7: AC PW Cases	BPS-TUNE™	cpageautomation	main	3.10, 3.11
Step 8: AC Benchmarking	BPS-TUNE™	datasetcomparisonautomation	main	3.10, 3.11
Step 11: Visualization	BPS-TUNE™	make_qgis_data, build_qgz	qgis	3.12

Appendix B – Create Mapping Files

This appendix provides guidance for creating mapping files that define the special relationships needed to translate power system components from a production cost model representation into a power flow (PF) model representation. The purpose of this guidance is to help users understand how to build or update the required mapping files when working with new power system models. In particular, it focuses on how to represent component correspondences, naming differences, and any model-specific exceptions so that the automated mapping process can be applied correctly and consistently.

The mapping process is organized into the following categories:

1. **Gen**: Handles mapping operations for generation tables, including both PCM and PF mappings.
2. **HVDCGen**: Handles mapping operations to convert HVDC lines into equivalent generators at AC terminals.
3. **HVDCDir**: Handles mapping operations to convert bi-directional HVDC lines into uni-directional HVDC lines.
4. **HVDCVSC**: Handles mapping operations to convert HVDC lines into voltage source controlled (VSC) HVDC lines.
5. **MTDC**: Handles mapping operations to convert HVDC lines into multi-terminal DC (MTDC) lines.

Detailed explanations of the needed files and context of each file for proper mapping are provided here:

1. **Hydro**: The Hydro category is used to map a hydro plant in the production cost model (PCM) to the corresponding individual hydro generating units in the PF model. This mapping is needed because hydro resources are often represented differently in the two models. In the PCM, a hydro plant may be modeled as a single aggregated resource at the plant level, while in the PF model the same plant may be represented by many separate generating units connected at one or more buses. Also, in the PF model, hydro generators are dispatched based on a pre-defined order list based on their ratings and availability. The hydro mapping file defines how the aggregated plant should be associated with its detailed PF generator records.

The file contains the following columns:

- **plant_id**: A unique identifier for the hydro plant.
- **plant_name**: The name of the hydro plant.
- **bus**: The bus number of the hydro generator in the PF model.
- **name**: The generator name in the PF model.
- **genid**: The generator ID in the PF model.
- **cap**: The generating capacity of the individual hydro unit.
- **res_abs**: The absolute reserve value assigned to the hydro unit.
- **res_per**: The reserve percentage assigned to the hydro unit.
- **load_order**: The loading priority or dispatch order for the hydro unit within the plant.

Example:

plant_id	plant_name	bus	name	genid	cap	res_abs	res_per	load_order
1	Plant1	1	C1	1	200	50	0	1
1	Plant1	2	C2	1	200	50	0	3
1	Plant1	3	C3	1	200	50	0	4
1	Plant1	4	C4	1	200	50	0	2

2. Generation

The Gen category is used to map **generator representations between the PCM and the PF model**. This category is needed to address two main issues:

1. **Generators that exist at different bus locations between the PCM and the PF model**
2. **Generators that are aggregated or disaggregated at different levels in each model**

In both cases, the same physical plant may not appear identically in the two models. The Gen mapping files provide a consistent way to link the PCM generator records to the corresponding PF generator records.

This category requires **two files**:

- **gen_pcm_map.csv**
- **gen_pf_map.csv**

The generators are matched using the **first three columns** in each file:

- **poi**
- **shape**
- **type**

These three columns together represent the **identification of the power plant** and are used as the common key between the PCM and PF mapping files.

The first file “gen_pcm_map.csv” defines how the generator or plant appears in the production cost model. The columns are:

- **poi**: The point-of-interconnection or plant location identifier.
- **shape**: A plant grouping or shape identifier used to distinguish units or configurations within the same plant.
- **type**: The generator technology or plant type.
- **gbus**: The generator bus number in the PCM model.
- **id**: The generator identifier in the PCM model.
- **pmax**: The maximum active power capacity of the generator in the PCM model.

Example:

poi	shape	type	gbus	id	pmax
1	1	gas	1	a	100
2	10	solar	2	1	10
2	10	solar	2	2	10
3	30	wind	3	1	35

The second file “gen_pf_map.csv” defines how the same generator or plant appears in the PF model. The columns are:

- **poi**: The point-of-interconnection or plant location identifier.
- **shape**: A plant grouping or shape identifier.
- **type**: The generator technology or plant type.
- **gbus**: The generator bus number in the PF model.
- **id**: The generator identifier in the PF model.
- **pmax**: The maximum active power capacity of the generator in the PF model.

Example:

poi	shape	type	gbus	id	pmax
1	1	gas	1	b	100
2	10	solar	2	a	20
3	30	wind	3	1	10
3	30	wind	3	2	25

3. HVDC to Generators

The HVDCGen category is used to map an HVDC line in the PCM to equivalent generators at the AC terminals in the PF model. This type of mapping is needed when the PF representation does not model the HVDC link directly as an HVDC element but instead represents its behavior using generator records at the connected AC buses. To ensure the mapping is recognized by the code, the file must be named “hvdc_gen_map.csv”.

The file contains the following columns:

- **from_bus**: The sending-end bus number of the HVDC line in the PCM model.
- **to_bus**: The receiving-end bus number of the HVDC line in the PCM model.
- **ID**: The identifier of the HVDC line in the PCM model.

These first three columns uniquely identify the HVDC line in the production cost model.

- **from_gbus**: The generator bus number in the PF model that represents the from-side AC terminal of the HVDC line.
- **from_id**: The generator identifier in the PF model for the from-side equivalent generator.
- **to_gbus**: The generator bus number in the PF model that represents the to-side AC terminal of the HVDC line.
- **to_id**: The generator identifier in the PF model for the to-side equivalent generator.
- **pmax**: The maximum transfer capability or maximum active power associated with the HVDC link, typically used as the capacity of the equivalent representation.

Example:

from_bus	to_bus	ID	from_gbus	from_id	to_gbus	to_id	pmax
1	2	a	1	G1	2	G1	400

4. HVDC to bi-directional HVDC

The HVDCDir category is used to map a bi-directional HVDC line in the PCM to its corresponding unidirectional HVDC representations in the PF model. In some cases, a single HVDC connection in the PCM is modeled as one logical line that can transfer power in either direction, while in the PF model the same connection is represented using two separate directional records—one for the forward direction and one for the reverse direction. The file name must be “hvdc_direction_map.csv”.

The file contains the following columns:

- **from_bus**: The sending-end bus number of the HVDC line in the PCM model.
- **to_bus**: The receiving-end bus number of the HVDC line in the PCM model.
- **ID**: The circuit identifier of the HVDC line in the PCM model.

These first three columns uniquely identify the HVDC line in the **PCM**.

- **from_bus_pf**: The sending-end bus number of the corresponding HVDC line in the PF model for the **forward direction**.
- **to_bus_pf**: The receiving-end bus number of the corresponding HVDC line in the PF model for the **forward direction**.
- **ID_pf**: The circuit identifier of the corresponding HVDC line in the PF model for the **forward direction**.

These three columns identify the HVDC record in the **PF model** that represents flow in the same direction as the PCM line definition.

- **from_bus_pf_reverse**: The sending-end bus number of the corresponding HVDC line in the PF model for the **reverse direction**.
- **to_bus_pf_reverse**: The receiving-end bus number of the corresponding HVDC line in the PF model for the **reverse direction**.
- **ID_pf_reverse**: The circuit identifier of the corresponding HVDC line in the PF model for the **reverse direction**.

These final three columns identify the HVDC record in the **PF model** that represents flow in the opposite direction.

Example:

from_bus	to_bus	ID	from_bus_pf	to_bus_pf	ID_pf	from_bus_pf_reverse	to_bus_pf_reverse	ID_pf_reverse
1	2	a	1	2	f	2	1	r

5. HVDC to VSC-HVDC

The HVDCVSC category is used to map an **HVDC line in the PCM** to its corresponding **VSC HVDC representation in the PF model**. This is needed when an HVDC connection that is modeled as a standard DC line in the PCM must be represented in the PF model using a VSC-based HVDC element. The “hvdc_vsc_map.csv” file defines the relationship between the PCM HVDC line identifier and the corresponding VSC HVDC record in the PF model.

The file contains the following columns:

- **from_bus**: The sending-end bus number of the HVDC line in the PCM model.
- **to_bus**: The receiving-end bus number of the HVDC line in the PCM model.
- **ID**: The identifier of the HVDC line in the PCM model.

These first three columns uniquely identify the HVDC line in the **production cost model**.

- **from_bus_pf**: The sending-end bus number of the corresponding VSC HVDC line in the PF model.
- **to_bus_pf**: The receiving-end bus number of the corresponding VSC HVDC line in the PF model.
- **name_pf**: The name or identifier of the corresponding VSC HVDC record in the PF model.

Example:

from_bus	to_bus	ID	from_bus_pf	to_bus_pf	name_pf
1	2	dc	1	2	1_2

6. HVDC to MTDC

The MTDC category is used to map an HVDC line in the PCM to its corresponding multi-terminal DC (MTDC) representation in the PF model. This is needed when a point-to-point HVDC connection in the PCM is represented in the PF model as part of an MTDC system with associated AC and DC bus definitions. The mtdc_map.csv file defines how the PCM HVDC line corresponds to the MTDC configuration in the PF model. The file must be named “mtdc_map.csv”.

The file contains the following columns:

- **from_bus**: The sending-end bus number of the HVDC line in the PCM model.
- **to_bus**: The receiving-end bus number of the HVDC line in the PCM model.
- **ID**: The identifier of the HVDC line in the PCM model.

These first three columns uniquely identify the HVDC line in the **production cost model**.

- **from_AC_bus_number**: The AC bus number in the PF model associated with the sending-end converter station.
- **from_DC_bus_number**: The DC bus number in the PF model associated with the sending-end side of the MTDC system.
- **to_AC_bus_number**: The AC bus number in the PF model associated with the receiving-end converter station.

- **to_DC_bus_number**: The DC bus number in the PF model associated with the receiving-end side of the MTDC system.
- **mtdc_number**: The identifier of the MTDC system in the PF model.

Example:

from_bus	to_bus	ID	from_AC_bus_number	from_DC_bus_number	to_AC_bus_number	to_DC_bus_number	mtdc_number
1	2	dc	1	11	2	12	1

Pacific Northwest National Laboratory

902 Battelle Boulevard
P.O. Box 999
Richland, WA 99354

1-888-375-PNNL (7665)

www.pnnl.gov