## CONTENTS:

1 Using Prescient  
1.1 Installation .................................................. 3  
1.2 Running Prescient ........................................... 5  
1.3 Configuration Options ...................................... 8  
1.4 Input Data ..................................................... 14  
1.5 Results and Statistics Output ............................... 14  
1.6 Customizing Prescient with Plugins ....................... 14  

2 Modeling Concepts  
2.1 The Prescient Simulation Cycle .............................. 15  
2.2 Time Series Data Streams .................................. 17  
2.3 Reserves and Ancillary Services ............................ 18  
2.4 Energy Markets and Pricing ................................. 18  

3 Examples and Tutorials ......................................... 19  

4 Reference  
4.1 File Formats .................................................. 21  
4.2 Python Classes and Functions .............................. 21  

5 Indices and tables ............................................... 23
Prescient is a python library that provides production cost modeling capabilities for power generation and distribution networks.
1.1 Installation

The Prescient python package can be installed using pip, or it can be installed from source. Python and a linear solver are prerequisites for either installation method.

To install Prescient, follow these steps:

- **Install python**
- **Install a linear solver**
- **Install Using Pip**
- **Install From Source**
  - Get Prescient source code
  - Install Python Dependencies
  - Install Egret
  - Install the Prescient python package
  - Verify your installation

### 1.1.1 Install python

Prescient requires python 3.7 or later. We recommend installing Anaconda to manage python and other dependencies.

### 1.1.2 Install a linear solver

Prescient requires a mixed-integer linear programming (MILP) solver that is compatible with Pyomo. Options include open source solvers such as CBC or GLPK, and commercial solvers such as CPLEX, Gurobi, or Xpress.

The specific mechanics of installing a solver is specific to the solver and/or the platform. An easy way to install an open source solver on Windows, Linux, and Mac is to install the CBC Anaconda package into the current conda environment:

```
conda install -c conda-forge coincbc
```

**Tip:** Be sure to activate the correct python environment before running the command above.
Note that the CBC solver is used in most Prescient tests, so you may want to install it even if you intend to use another solver in your own runs.

### 1.1.3 Install Using Pip

Prescient is available as a python package that can be installed using pip. To install the latest release of Prescient use the following command:

```
    pip install gridx-prescient
```

Be sure the intended python environment is active before issuing the command above.

### 1.1.4 Install From Source

You may want to install from source if you want to use the latest pre-release version of the code, or if you want to modify/contribute to the code yourself. The steps required to install Prescient from source are described below:

**Get Prescient source code**

The latest version of Prescient can be acquired as source from the Prescient github project, either by downloading a zip file of the source code or by cloning the `main` branch of the github repository.

**Install Python Dependencies**

The python environment where you run Prescient must include a number of prerequisites. You may want to create a python environment specifically for Prescient. To create a new Anaconda environment and install Prescient’s prerequisites into the new environment, issue the following command from the root folder of the Prescient source code:

```
    conda env create -f environment.yml
```

The command above will create an environment named `prescient`. To use a different name for the environment, add the `-n` option to the command above:

```
    conda env create -n nameOfYourChoice -f environment.yml
```

Once you have create the new environment, make it the active environment:

```
    conda activate prescient
```

If you are using something other than Anaconda to manage your python environment, use the information in `environment.yml` to identify which packages to install.

**Install Egret**

When installing Prescient from the latest version of the source code, Egret may need to be installed manually because pre-release versions of Prescient sometimes depend on pre-release versions of EGRET. Install EGRET from source according to the instructions here <https://github.com/grid-parity-exchange/Egret/blob/main/README.md>.
**Install the Prescient python package**

The steps above configure a python environment with Prescient’s prerequisites. Now we must install Prescient itself. From the prescient python environment, issue the following command:

```
pip install -e .
```

This will update the active python environment to include Prescient’s source code. Any changes to Prescient source code will take affect each time Prescient is run.

This command will also install a few utilities that Prescient users may find useful, including `runner.py` (see Running Prescient).

**Verify your installation**

Prescient is packaged with tests to verify it has been set up correctly. To execute the tests, issue the following command:

```
python -m unittest tests/simulator_tests/test_sim_rts_mod.py
```

This command runs the tests using the CBC solver and will fail if you haven’t installed CBC. The tests can take as long as 30 minutes to run, depending on your machine. If Prescient was installed correctly then all tests should pass.

### 1.2 Running Prescient

There are three ways to launch and run Prescient:

- With a configuration file, using `runner.py`
- With command line options, using the `prescient.simulator` module
- From python code, using in-code configuration

In all three cases, the analyst supplies configuration values that identify input data and dictate which options to use during the Prescient simulation. Configuration options can be specified in a configuration file, on the command line, in-code, or a combination of these methods, depending on how Prescient is launched.

To see what configuration options are available, see Configuration Options.

#### 1.2.1 Launch with runner.py

Prescient can be run using `runner.py`, a utility which is installed along with Prescient (see Install Egret). Before executing `runner.py`, you must create a configuration file indicating how Prescient should be run. Here is an example of a configuration file that can be used with `runner.py`:

```
command/exec simulator.py
--data-directory=example_scenario_input
--output-directory=example_scenario_output
--input-format=rts-gmlc
--run-sced-with-persistent-forecast-errors
--start-date=07-11-2024
--num-days=7
--sced-horizon=1
--sced-frequency-minutes=10
--ruc-horizon=36
```
Because runner.py can potentially be used for more than launching Prescient, the first line of the configuration file must match the line shown in the example above. Otherwise runner.py won’t know that you intend to run Prescient.

All subsequent lines set the value of a configuration option. Configuration options are described in Configuration Options.

Once you have the configuration file prepared, you can launch Prescient using the following command:

```
runner.py config.txt
```

where `config.txt` should be replaced with the name of your configuration file.

### 1.2.2 Launch with the `prescient.simulator` module

Another way to run Prescient is to execute the `prescient.simulator` module:

```
python -m prescient.simulator <options>
```

where `options` specifies the configuration options for the run. An example might be something like this:

```
python -m prescient.simulator --data-directory=example_scenario_input --output-directory=example_scenario_output --input-format=rts-gmlc --run-sced-with-persistent-forecast-errors --start-date=07-11-2024 --num-days=7 --sced-horizon=1 --sced-frequency-minutes=10 --ruc-horizon=36
```

Configuration options can also be specified in a configuration file:

```
python -m prescient.simulator --config-file=config.txt
```

You can combine the `--config-file` option with other command line options. The contents of the configuration file are effectively inserted into the command line at the location of the `--config-file` option. You can override values in a configuration file by repeating the option at some point after the `--config-file` option.

Running the `prescient.simulator` module allows you to run Prescient without explicitly installing it, as long as Prescient is found in the python module search path.

### 1.2.3 Running Prescient from python code

Prescient can be configured and launched from python code:

```
from prescient.simulator import Prescient

Prescient().simulate(
    data_path='deterministic_scenarios',
    simulate_out_of_sample=True,
    run_sced_with_persistent_forecast_errors=True,
    output_directory='deterministic_simulation_output',
    start_date='07-10-2020',
    num_days=7,
    sced_horizon=4,
    reserve_factor=0.0,
    deterministic_ruc_solver='cbc',
    sced_solver='cbc',
    sced_frequency_minutes=60,
)
```

(continues on next page)
The code example above creates an instance of the Prescient class and passes configuration options to its `simulate()` method. An alternative is to set values on a configuration object, and then run the simulation after configuration is done:

```python
from prescient.simulator import Prescient

p = Prescient()
config = p.config
config.data_path = 'deterministic_scenarios'
config.simulate_out_of_sample = True
config.run_sced_with_persistent_forecast_errors = True
config.output_directory = 'deterministic_simulation_output'
config.start_date = '07-10-2020'
config.num_days = 7
config.sced_horizon = 4
config.reserve_factor = 0.0
config.deterministic_ruc_solver = 'cbc'
config.sced_solver = 'cbc'
config.sced_frequency_minutes = 60
config.ruc_horizon = 36
config.enforce_sced_shutdown_ramprate = True
config.no_startup_shutdown_curves = True

p.simulate()
```

A third option is to store configuration values in a `dict`, which can potentially be shared among multiple runs:

```python
from prescient.simulator import Prescient

options = {
    'data_path': 'deterministic_scenarios',
    'simulate_out_of_sample': True,
    'run_sced_with_persistent_forecast_errors': True,
    'output_directory': 'deterministic_simulation_output'
}

Prescient().simulate(**options)
```

These three methods can be used together quite flexibly. The example below demonstrates a combination of approaches to configuring a prescient run:

```python
from prescient.simulator import Prescient

simulator = Prescient()

# Set some configuration options using the simulator's config object
config = simulator.config
config.data_path = 'deterministic_scenarios'
```
```python
config.simulate_out_of_sample = True
config.run_sced_with_persistent_forecast_errors = True
config.output_directory = 'deterministic_simulation_output'

# Others will be stored in a dictionary that can
# potentially be shared among multiple prescient runs
options = {
    'start_date': '07-10-2020',
    'sced_horizon': 4,
    'reserve_factor': 0.0,
    'deterministic_ruc_solver': 'cbc',
    'sced_solver': 'cbc',
    'sced_frequency_minutes': 60,
    'ruc_horizon': 36,
    'enforce_sced_shutdown_ramp_rate': True,
    'no_startup_shutdown_curves': True,
}

# And finally, pass the dictionary to the simulate() method,
# along with an additional function argument.
simulator.simulate(**options, num_days=7)
```

### 1.3 Configuration Options

- **Overview**
- **Option Data Types**
- **List of Configuration Options**

#### 1.3.1 Overview

Prescient configuration options are used to indicate how the Prescient simulation should be run. Configuration options can be specified on the command line, in a text configuration file, or in code, depending on how Prescient is launched (see Running Prescient).

Each configuration option has a name, a data type, and a default value. The name used on the command line and the name used in code vary slightly. For example, the number of days to simulate is specified as `--num-days` on the command line, and `num_days` in code.
1.3.2 Option Data Types

Most options use self-explanatory data types like `String`, `Integer`, and `Float`, but some data types require more explanation and may be specified in code in ways that are unavailable on the command line:

<table>
<thead>
<tr>
<th>Data type</th>
<th>Command-line/config file usage</th>
<th>In-code usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Path</td>
<td>A text string that refers to a file or folder. Can be relative or absolute, and may include special characters such as ~.</td>
<td>Same as command-line</td>
</tr>
<tr>
<td>Date</td>
<td>A string that can be converted to a date, such as 1776-07-04.</td>
<td>Either a string or a datetime object.</td>
</tr>
<tr>
<td>Flag</td>
<td>Simply include the option to set it to true. For example, the command below sets simulate_out_of_sample to true: runner.py --simulate-out-of-sample</td>
<td>Set the option by assigning True or False: config.simulate_out_of_sample = True</td>
</tr>
<tr>
<td>Module</td>
<td>Refer to a python module in one of the following ways: - The name of a python module (such as prescient.simulator.prescient) - The path to a python file (such as prescient/simulator/prescient.py)</td>
<td>In addition to the two string options available to the command-line, code may also use a python module object. For example: import my_custom_data_provider config.data_provider = my_custom_data_provider</td>
</tr>
</tbody>
</table>

1.3.3 List of Configuration Options

The table below describes all available configuration options.

<table>
<thead>
<tr>
<th>Command-line Option</th>
<th>In-Code Configuration Property</th>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>--config-file</td>
<td>config_file</td>
<td>Path. Default=None.</td>
<td>Path to a file holding configuration options. Can be absolute or relative. Cannot be set in code directly on a configuration object, but can be passed to a configuration object's <code>parse_args()</code> function: p = Prescient() p.config.parse_args([&quot;--config-file&quot;, &quot;my-config.txt&quot;])</td>
</tr>
</tbody>
</table>

See `Launch with runner.py` for a description of configuration file syntax.

1.3. Configuration Options
Table 2 – continued from previous page

<table>
<thead>
<tr>
<th>Command-line Option</th>
<th>In-Code Configuration Property</th>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>--start-date</td>
<td>start_date</td>
<td>Date. Default=2020-01-01.</td>
<td>The start date for the simulation.</td>
</tr>
<tr>
<td>--num-days</td>
<td>num_days</td>
<td>Integer. Default=7</td>
<td>The number of days to simulate.</td>
</tr>
</tbody>
</table>

**Data Options**

<table>
<thead>
<tr>
<th>Option</th>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>--data-path or --data-directory</td>
<td>data_path</td>
<td>Path. Default=input_data.</td>
</tr>
<tr>
<td>--input-format</td>
<td>input_format</td>
<td>String. Default=dat.</td>
</tr>
<tr>
<td>--data-provider</td>
<td>data_provider</td>
<td>Module. Default=No custom data provider.</td>
</tr>
<tr>
<td>--output-directory</td>
<td>output_directory</td>
<td>Path. Default=oudir.</td>
</tr>
</tbody>
</table>

**RUC Options**

<table>
<thead>
<tr>
<th>Option</th>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>--ruc_every-hours</td>
<td>ruc_every_hours</td>
<td>Integer. Default=24</td>
</tr>
<tr>
<td>--ruc-execution-hour</td>
<td>ruc_execution_hour</td>
<td>Integer. Default=16</td>
</tr>
<tr>
<td>--ruc-horizon</td>
<td>ruc_horizon</td>
<td>Integer. Default=48</td>
</tr>
<tr>
<td>--ruc-prescience-hour</td>
<td>ruc_prescience_hour</td>
<td>Integer. Default=0.</td>
</tr>
<tr>
<td>--run-ruc-with-next-day-data</td>
<td>run_ruc_with_next_day_data</td>
<td>Default=false.</td>
</tr>
</tbody>
</table>

continues on next page
<table>
<thead>
<tr>
<th>Command-line Option</th>
<th>In-Code Configuration Property</th>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>--simulate-out-of-sample</td>
<td>simulate_out_of_sample</td>
<td>Flag. Default=false.</td>
<td>If false, use forecast input data as both forecasts and actual values; the actual value input data is ignored. If true, values for the current simulation time are taken from the actual value input, and actual values are used to blend near-term values if ruc_prescience_hour is non-zero.</td>
</tr>
<tr>
<td>--ruc-network-type</td>
<td>ruc_network_type</td>
<td>String. Default=ptdf.</td>
<td>Specifies how the network is represented in RUC models. Choices are: * ptdf – power transfer distribution factor representation * btheta – b-theta representation</td>
</tr>
<tr>
<td>--ruc-slack-type</td>
<td>ruc_slack_type</td>
<td>String. Default=every-bus.</td>
<td>Specifies the type of slack variables to use in the RUC model formulation. Choices are: * every-bus – slack variables at every system bus * ref-bus-and-branches – slack variables at only reference bus and each system branch</td>
</tr>
<tr>
<td>--deterministic-ruc-solver</td>
<td>deterministic_ruc_solver</td>
<td>String. Default=cbc.</td>
<td>The name of the solver to use for RUCs.</td>
</tr>
<tr>
<td>--deterministic-ruc-solver-options</td>
<td>deterministic_ruc_solver_options</td>
<td>String. Default=None.</td>
<td>Solver options applied to all RUC solves.</td>
</tr>
<tr>
<td>--ruc-mipgap</td>
<td>ruc_mipgap</td>
<td>Float. Default=0.01.</td>
<td>The mipgap for all deterministic RUC solves.</td>
</tr>
<tr>
<td>--output-ruc-initial-conditions</td>
<td>output_ruc_initial_conditions</td>
<td>Flag. Default=false.</td>
<td>Print initial conditions to stdout prior to each RUC solve.</td>
</tr>
<tr>
<td>--output-ruc-solutions</td>
<td>output_ruc_solutions</td>
<td>Flag. Default=false.</td>
<td>Print RUC solution to stdout after each RUC solve.</td>
</tr>
<tr>
<td>--write-deterministic-ruc-instances</td>
<td>write_deterministic_ruc_instances</td>
<td>Flag. Default=false.</td>
<td>Save each individual RUC model to a file. The date and time the RUC was executed is indicated in the file name.</td>
</tr>
</tbody>
</table>
| --deterministic-ruc-solver-plugin | deterministic_ruc_solver_plugin | Module. Default=None. | If the user has an alternative method to solve RUCs, it should be specified here, e.g., my_special_plugin.py.  
**Note:** This option is ignored if --simulator-plugin is used. |

**SCED Options**

<table>
<thead>
<tr>
<th>Command-line Option</th>
<th>In-Code Configuration Property</th>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>--sced-frequency-minutes</td>
<td>sced_frequency_minutes</td>
<td>Integer. Default=60.</td>
<td>How often a SCED will be run, in minutes. Must divide evenly into 60, or be a multiple of 60.</td>
</tr>
<tr>
<td>--sced-horizon</td>
<td>sced_horizon</td>
<td>Integer. Default=1</td>
<td>The number of time periods to include in each SCED. Must be at least 1.</td>
</tr>
</tbody>
</table>

continues on next page
Table 2 – continued from previous page

<table>
<thead>
<tr>
<th>Command-line Option</th>
<th>In-Code Configuration Property</th>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>--run-sced-with-persistent-forecast-errors</td>
<td>run_sced_with_persistent_forecast_errors</td>
<td>Flag. Default=false.</td>
<td>If true, then values in SCEDs use persistent forecast errors. If false, all values in SCEDs use actual values for all time periods, including future time periods. See Forecast Smoothing.</td>
</tr>
<tr>
<td>--enforce-sced-shutdown-ramp-rate</td>
<td>enforce_sced_shutdown_ramprate</td>
<td>Flag. Default=false.</td>
<td>Enforces shutdown ramp-rate constraints in the SCED. Enabling this option requires a long SCED look-ahead (at least an hour) to ensure the shutdown ramp-rate constraints can be satisfied.</td>
</tr>
<tr>
<td>--sced-network-type</td>
<td>sced_network_type</td>
<td>String. Default=ptdf.</td>
<td>Specifies how the network is represented in SCED models. Choices are: * ptdf – power transfer distribution factor representation * btheta – b-theta representation</td>
</tr>
<tr>
<td>--sced-slack-type</td>
<td>sced_slack_type</td>
<td>String. Default=every-bus.</td>
<td>Specifies the type of slack variables to use in SCED models. Choices are: * every-bus – slack variables at every system bus * ref-bus-and-branches – slack variables at only reference bus and each system branch</td>
</tr>
<tr>
<td>--sced-solver</td>
<td>sced_solver</td>
<td>String. Default=cbc.</td>
<td>The name of the solver to use for SCEDs.</td>
</tr>
<tr>
<td>--sced-solver-options</td>
<td>sced_solver_options</td>
<td>String. Default=None.</td>
<td>Solver options applied to all SCED solves.</td>
</tr>
<tr>
<td>--print-sced</td>
<td>print_sced</td>
<td>Flag. Default=false.</td>
<td>Print results from SCED solves to stdout.</td>
</tr>
<tr>
<td>--output-sced-initial-conditions</td>
<td>output_sced_initial_conditions</td>
<td>Flag. Default=false.</td>
<td>Print SCED initial conditions to stdout prior to each solve.</td>
</tr>
<tr>
<td>--output-sced-loads</td>
<td>output_sced_loads</td>
<td>Flag. Default=false.</td>
<td>Print SCED loads to stdout prior to each solve.</td>
</tr>
<tr>
<td>--write-sced-instances</td>
<td>write_sced_instances</td>
<td>Flag. Default=false.</td>
<td>Save each individual SCED model to a file. The date and time the SCED was executed is indicated in the file name.</td>
</tr>
</tbody>
</table>

**Output Options**

<table>
<thead>
<tr>
<th>Command-line Option</th>
<th>In-Code Configuration Property</th>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>--output-max-decimal-places</td>
<td>output_max_decimal_places</td>
<td>Integer. Default=6.</td>
<td>The number of decimal places to output to summary files. Output is rounded to the specified accuracy.</td>
</tr>
<tr>
<td>--output-solver-logs</td>
<td>output_solver_logs</td>
<td>Flag. Default=false.</td>
<td>Whether to print solver logs to stdout during execution.</td>
</tr>
</tbody>
</table>

**Miscellaneous Options**

<table>
<thead>
<tr>
<th>Command-line Option</th>
<th>In-Code Configuration Property</th>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

continues on next page
Table 2 – continued from previous page

<table>
<thead>
<tr>
<th>Command-line Option</th>
<th>In-Code Configuration Property</th>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>--reserve-factor</td>
<td>reserve_factor</td>
<td>Float. Default=0.0.</td>
<td>The reserve factor, expressed as a constant fraction of demand, for spinning reserves at each time period of the simulation. Applies to both RUC and SCED models.</td>
</tr>
<tr>
<td>--no-startup-shutdown-curves</td>
<td>no_startup_shutdown_curves</td>
<td>Flag. Default=False.</td>
<td>If true, then do not infer startup/shutdown ramping curves when starting-up and shutting-down thermal generators.</td>
</tr>
<tr>
<td>--symbolic-solver-labels</td>
<td>symbolic_solver_labels</td>
<td>Flag. Default=False.</td>
<td>Whether to use symbol names derived from the model when interfacing with the solver.</td>
</tr>
<tr>
<td>--enable-quick-start-generator-commitment</td>
<td>enable_quick_start_generator_commitment</td>
<td>Flag. Default=False.</td>
<td>Whether to allow quick start generators to be committed if load shedding would otherwise occur.</td>
</tr>
<tr>
<td><strong>Market and Pricing Options</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>--compute-market-settlements</td>
<td>compute_market_settlements</td>
<td>Flag. Default=False.</td>
<td>Whether to solve a day-ahead market as well as real-time market and report the daily profit for each generator based on the computed prices.</td>
</tr>
<tr>
<td>--day-ahead-pricing</td>
<td>day_ahead_pricing</td>
<td>String. Default=aCHP.</td>
<td>The pricing mechanism to use for the day-ahead market. Choices are: * LMP – locational marginal price * ELMP – enhanced locational marginal price * aCHP – approximated convex hull price.</td>
</tr>
<tr>
<td>--price-threshold</td>
<td>price_threshold</td>
<td>Float. Default=10000.0.</td>
<td>Maximum possible value the price can take. If the price exceeds this value due to Load Mismatch, then it is set to this value.</td>
</tr>
<tr>
<td>--reserve-price-threshold</td>
<td>reserve_price_threshold</td>
<td>Float. Default=10000.0.</td>
<td>Maximum possible value the reserve price can take. If the reserve price exceeds this value, then it is set to this value.</td>
</tr>
<tr>
<td><strong>Plugin Options</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>--plugin</td>
<td>plugin</td>
<td>Module. Default=None.</td>
<td>Python plugins are analyst-provided code that Prescient calls at various points in the simulation process. See Customizing Prescient with Plugins for details. After Prescient has been initialized, the configuration object’s plugin property holds plugin-specific setting values.</td>
</tr>
<tr>
<td>--simulator-plugin</td>
<td>simulator_plugin</td>
<td>Module. Default=None.</td>
<td>A module that implements the engine interface. Use this option to replace methods that setup and solve RUC and SCED models with custom implementations.</td>
</tr>
</tbody>
</table>
1.4 Input Data

1.4.1 Custom Data Providers

1.5 Results and Statistics Output

Under Construction
Documentation coming soon

1.6 Customizing Prescient with Plugins

Under Construction
Documentation coming soon
2.1 The Prescient Simulation Cycle

RUC
Daily – Generator Commitments

SCED
Hourly – Dispatch Levels

Prescient simulates the operation of a power generation network throughout a study horizon, finding the set of opera-
tional choices that satisfy demand at the lowest possible cost.

A Prescient simulation consists of two repeating cycles, one nested in the other. The outer cycle is the Reliability Unit Commitment (RUC) planning cycle, which schedules changes in dispatchable generators’ online status during the cycle’s period. The inner, more frequent cycle is the Security Constrained Economic Dispatch (SCED) cycle, which determines dispatch levels for dispatchable generators.

### 2.1.1 The RUC Cycle

The RUC cycle periodically generates a RUC plan. A RUC plan consists of two types of data: a unit commitment schedule and optionally a pricing schedule (when `compute-market-settlements` is True). The unit commitment schedule indicates which dispatchable generators should be activated or deactivated during upcoming time periods. The pricing schedule sets the contract price for expected power delivery and for reserves (ancillary service products). The RUC plan reflects the least expensive way to satisfy predicted loads while honoring system constraints.

A new RUC plan is generated at regular intervals, at least once per day. A new RUC plan always goes into effect at midnight of each day. If more than one RUC plan is generated each day, then additional RUC plans take effect at equally spaced intervals. For example, if 3 RUC plans are generated each day, then one will go into effect at midnight, one at 8:00 a.m., and one at 4:00 p.m. Each RUC plan covers the time period that starts when it goes into effect and ends just as the next RUC plan becomes active.

A RUC plan is based on the current state of the system at the time the plan is generated (particularly the current dispatch and up- or down-time for dispatchable generators), and on forecasts for a number of upcoming time periods. The forecasts considered when forming a RUC plan must extend at least to the end of the RUC’s planning period, but typically extend further into the future in order to avoid poor choices at the end of the plan (“end effects”). The amount of time to consider when generating a RUC plan is known as the RUC horizon. A commonly used RUC horizon is 48 hours.

The simulation can be configured to generate RUC plans some number of hours before they take effect. This is done by specifying a time of day for one of the plans to be generated. The gap between the specified generation time and the next time a RUC plan is scheduled to take effect is called the RUC gap. Each RUC plan still covers the expected time period, from the time the plan takes effect until the next RUC plan takes effect, but its decisions will be based on what is known at the time the RUC plan is generated.

### 2.1.2 The SCED Cycle

The SCED process selects dispatch levels for all active dispatchable generators in the current simulation time period. Dispatch levels are determined using a process that is very similar to that used to build a RUC plan. The current state of the system, together with forecasts for a number of future time periods, are examined to select dispatch levels that satisfy current loads and forecasted future loads at the lowest possible cost.

The SCED cycle is more frequent than the RUC cycle, with new dispatch levels selected at least once an hour. The SCED honors unit commitment decisions made in the RUC plan; whether each generator is committed or not is dictated by the RUC schedule currently in effect.

Costs are also determined with each SCED, based on dispatchable generation selected by the SCED process, the commitment and start-up costs as selected by the associated RUC process, as well as current actual demands and non-dispatchable generation levels.
2.2 Time Series Data Streams

Prescient uses time series data from two data streams, the real-time stream (i.e., actuals) and the forecast stream. As their names imply, the real-time stream includes data that the simulation should treat as actual values that occur at specific times in the simulation, and the forecast stream includes forecasts for time periods that have not yet occurred in the simulation.

Both streams consist of time-stamped values for loads and non-dispatchable generation data.

2.2.1 Real-Time Data (Actuals)

The real-time data stream provides data that the simulation should treat as actual values. Real-time values are typically used only when the simulation reaches the corresponding simulation time.

Real-time data can be provided at any time interval. The real-time data interval generally matches the SCED interval (see sced-frequency-minutes), but this is not a requirement. If the SCED interval does not match the real-time interval then real-time data will be interpolated or discarded as needed to match the SCED interval.

2.2.2 Forecasts

Forecast data are provided by the forecast data stream. The frequency of data provided through the forecast stream must be hourly.

New forecasts are retrieved each time a new RUC plan is generated. The forecasts retrieved in a given batch are those required to satisfy the RUC horizon (see ruc-horizon), starting with the RUC activation time.

Forecast Smoothing

As forecasts are retrieved from the forecast data stream, they may be adjusted so that near-term forecasts are more accurate than forecasts further into the future. This serves two purposes: first, to avoid large jumps in timeseries values due to inaccurate forecasts; and second, to model how forecasts become more accurate as their time approaches.

The number of forecasts to be smoothed is determined by the ruc-prescience-hour configuration option. Values for the current simulation time are set equal to their actual value, ignoring data read from the forecast stream. Values for ruc-prescience-hour hours after the current simulation time are set equal to data read from the forecast stream. Between these two times, values are a weighted average of the values provided by the actuals and forecast data streams. The weights vary linearly with where the time falls between the current time and the ruc prescience hour. For example, if ruc-prescience-hour is 8, then the adjusted forecast for 2 hours after the current simulation time will be $0.25 \times \text{forecast} + 0.75 \times \text{actual}$.

Note that blending weights are determined relative to the current simulation time when the RUC is generated, not relative to the time the RUC goes into effect.
Real-Time Forecast Adjustments

Forecasts are adjusted further each time a SCED is run. This is done by comparing the forecast for the current time with the actual value for the current time. The ratio of these two values is calculated, then used as a scaling factor for forecast values. For example, if the forecast for a value was 10% too high, all future forecasts for the same value are reduced by 10%.

Note: If `run-sced-with-persistent-forecast-errors` is false, then SCEDs will use actual values for all time periods. Forecasts will still be used for RUCs, but SCEDs will be based entirely on actual values, even for future time periods.

2.3 Reserves and Ancillary Services

2.4 Energy Markets and Pricing
CHAPTER
THREE

EXAMPLES AND TUTORIALS
4.1 File Formats

4.1.1 RTS-GMLC
This is the main input format.

4.1.2 Pyomo DAT Files
Old way to do it.

4.2 Python Classes and Functions
• genindex
• modindex
• search