PNNL-SA-192761

Energy Storage Participation Algorithm: Wholesale Electricity Market Pilot Competition

Participation Guide

January 2024

1 Matt Cornachione
2 Liping Li
3 Brent Eldridge

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;
ph: (865) 576-8401
fax: (865) 576-5728
email: reports@adonis.osti.gov

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

Abstract

This document describes the logistical details of how a team will participate in the Energy Storage Participation Algorithm Competition (ESPA-Comp). It includes an in-depth explanation of the algorithm requirements, including information on the computing environment, available input data (market intervals, forecast and historical data, resource status, and financial settlements), and offer format specifications required for the Wholesale Electricity Analysis via Simulation and Learning Experiments (WEASLE) simulation platform. This document covers the algorithm submission process, including the differences between sandbox trial and competition algorithm submissions. It includes a section on how scores will be computed and a section with tips and suggestions for troubleshooting algorithm submissions. A simple “dummy” algorithm and market shell are available in the WEASLE repository (github.com/pnnl/weasle) to facilitate algorithm development by competitors.

Contents

Abstract 1
1.0 Overview 3
2.0 Algorithm Requirements 6
2.1 Computing Environment 6
2.2 Computing Time Restrictions 6
2.3 Participant Algorithm Naming 7
2.4 Input Files 7
2.5 Output Files 12
2.6 Market Shell Repository 16
3.0 Algorithm Submission 18
3.1 Signing Up 18
3.2 Sandbox Submissions 20
3.3 Competition Submissions 21
3.4 Submission Results 22
4.0 Score Evaluation 23
4.1 Surplus Components 23
4.2 Settlement Calculation 23
4.3 Degradation Cost 25
5.0 Troubleshooting 26
5.1 Interfacing with Market Clearing Code 26
5.2 Website and Submission Issues 26
5.3 Sandbox Submissions 26
5.4 Competition Submissions 28
5.5 PNNL Support 28

1.0 Overview

This participation guide is intended to give participants guidance on the technical requirements of algorithm submissions for the Wholesale Electricity Analysis via Simulation and Learning Experiments (WEASLE) simulation platform in the Energy Storage Participation Algorithm Competition (ESPA-Comp). Within this guide, we include guidance on code language choice, solvers, available computing resources, file formats (forecasts, offers, etc.), and website submission. We also provide tips for troubleshooting to ensure participant algorithms perform as expected on the Pacific Northwest National Laboratory's (PNNL) high performance cluster (see Section 5 for details).

Participants may compete individually or as a team. Only one user registration is allowed per team. To register, go to www.espa-competition.pnnl.gov, click the 'Login' button at the top right, and then 'Sign up' under the login credentials (see Section 3.1 for more details). Once you have an account, you will receive an email confirmation from ESPACompSupport@pnnl.gov. Shortly after registering, ESPA-Comp Support will contact participants to request additional information: coding environment requirements, solvers, and storage resource selection from one of several locations in the electric topology. Storage resource characteristics will be pre-determined by PNNL, but participants will have freedom to choose how to offer these resources to maximize their profit. Participants may also make virtual offers at any nodes as part of their bidding strategy.

After registration, competitors can make two types of algorithm submissions through the ESPA-Comp website: sandbox and competition. Sandbox submissions allow competitors to develop and debug their algorithms by running tests on the competition's market clearing engine. Sandbox submissions will return the final settlement scores over the period of a simulated market, allowing participants to assess the performance of their algorithm and implement adjustments as needed. Competition submissions are used to specify the final algorithm version to be used in the competition. Configuration settings for the two submission types differ slightly and are described in more detail in Section 3.0. Scores for each division are given by the overall profits for a participant, including sales/purchases of energy discharged/charged, ancillary services, virtual offers, and battery degradation costs, as described in more detail in Section 4.0.

In addition to the ESPA-Comp Sandbox, code for a simple "dummy" algorithm and market shell are available to competitors in the WEASLE GitHub repository (github.com/pnnl/weasle). Prospective competitors can also contact ESPACompSupport@pnnl.gov for additional assistance.

In broad terms, the ESPA-Comp storage offer problem is an adversarial multi-stage profit maximization problem with incomplete information. Participants could, for example, formulate the offer problem as a bilevel optimization problem with the market clearing model in the lower level. They may decide to formulate an equilibrium problem that considers the strategic incentives of other competitors in the simulation. They may otherwise formulate it as a reinforcement learning problem due to the presence of incomplete information. Or they may take other approaches. The only obligation is that submitted algorithms are compatible with the requirements provided by this document. Figure 1 illustrates an abstract form of the ESPA-Comp storage offer problem.

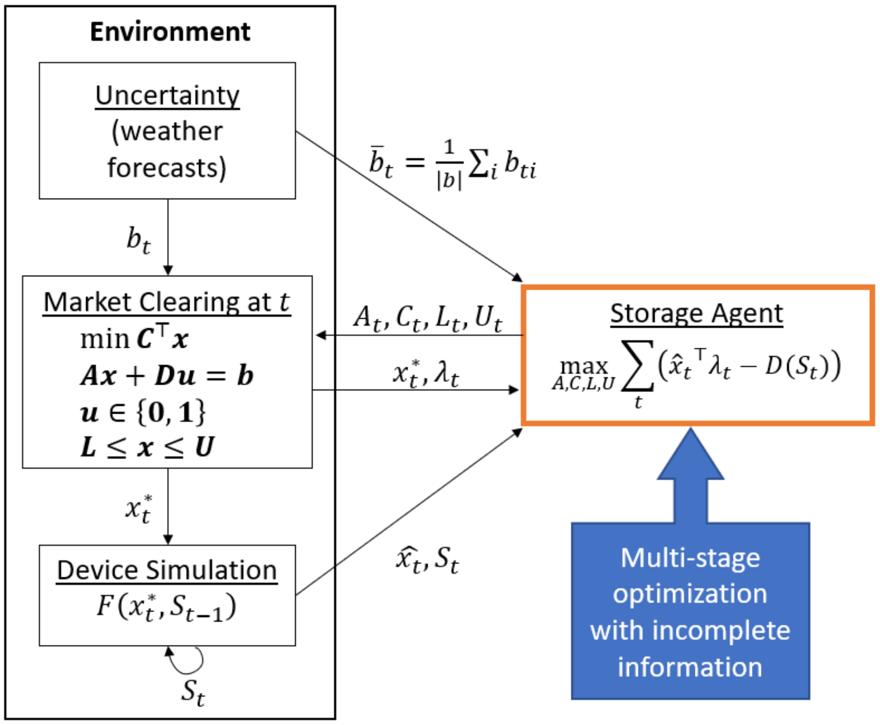

Environment
Uncertainty (weather forecasts)
b_t
Market Clearing at t
$\min \boldsymbol{C}^\top \boldsymbol{x}$
$\boldsymbol{Ax} + \boldsymbol{Du} = \boldsymbol{b}$
$\boldsymbol{u} \in \{\boldsymbol{0}, \boldsymbol{1}\}$
$\boldsymbol{L} \leq \boldsymbol{x} \leq \boldsymbol{U}$
x_t^*
Device Simulation
$F(x_t^*, S_{t-1})$
S_t
$\bar{b}_t = \frac{1}{|b|}\sum_i b_{ti}$
A_t, C_t, L_t, U_t
x_t^*, λ_t
$\hat{x}_t, S_t$
Storage Agent
$\max_{A,C,L,U} \sum_t (\hat{x}_t{}^\top \lambda_t - D(S_t))$
Multi-stage optimization with incomplete information

Figure 1: Abstract Diagram of the ESPA-Comp Offer Problem

The participant's problem is illustrated in the orange box. All participants share an environment that consists of weather uncertainty, the market clearing model, and the resource dispatch simulation. Throughout the course of the market simulation, participants will receive information from each portion of the environment, and they can decide how to process this information into market offers. Some of the information will be highly relevant to their market participation, but participants are free to ignore any information that they think is extraneous or irrelevant. A key for the symbols in Figure 1 is provided in Table 1.1.

Table 1.1: Symbol Key

b_t	vector component of a larger vector b of all RHS parameters in the market clearing model.
$\bar{b}_t$	vector of aggregated (averaged) components of b.
A_t	vector component of a larger vector A of device parameters in the market clearing model.
L_t, U_t	vector components of larger vectors L,U of upper and lower limits in the market clearing model.
C_t	vector component of a larger vector C of all cost coefficients in the market clearing model.
λ_t	vector of Lagrange multipliers (prices) from the market clearing model.
x_t^*	vector of dispatch quantities from the market clearing model.
$\hat{x}_t$	vector of actual quantities from the device simulation.
$D(\cdot)$	nonlinear function of degradation costs.
$F(\cdot)$	nonlinear function and projection of x^* onto a feasible region (i.e., constrained nonlinear program).
S	state variable that is input into F(·).

The ESPA-Comp market simulation will include three market design configurations (called 'divisions'): Two-Settlement Market, Multi-Settlement Market, and Rolling Horizon Market. For each division, participants will submit an algorithm capable of taking inputs from the market and returning offers at the appropriate time steps. The market inputs include forecast data, settlement history, the storage resource's current status, and scoring information (see Section 2.0 for details). Participants may submit different algorithms for each division, or they can submit the same algorithm in all three divisions.

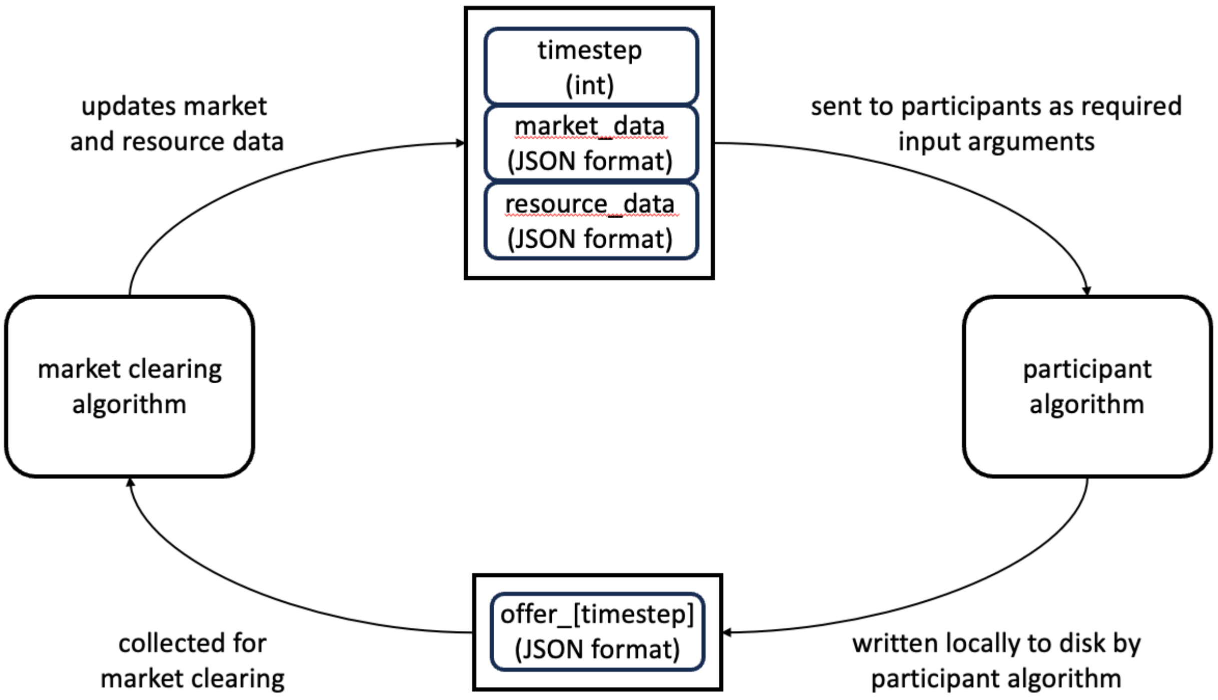

timestep
(int)
market_data
(JSON format)
resource_data
(JSON format)
updates market and resource data
sent to participants as required input arguments
market clearing algorithm
participant algorithm
offer_[timestep]
(JSON format)
collected for market clearing
written locally to disk by participant algorithm

Figure 2: Simplified Diagram of ESPA-Comp Inputs and Outputs to/from Participant Algorithm

Each participant's algorithm must be able to accept specific input data that is refreshed at every new market clearing interval. Offer data created by the algorithm must follow a particular format to be interpreted by the ESPA-Comp market clearing algorithm. Figure 2 shows how this information is passed between the ESPA market clearing algorithm and the participant algorithm.

2.0 Algorithm Requirements

This section contains the technical requirements for participant algorithms in ESPA-Comp. Subsections describe the computing environment (code languages, solvers, computing resources), computing time limitations, the input file formats (those generated by the market at each time step, the required offer format, and the PNNL-provided market shell.

A simple "dummy" algorithm is available to demonstrate the input/output data formats and syntax. This dummy algorithm and other helpful tools are located in the WEASLE GitHub repository (github.com/pnnl/weasle) in the "competitor-tools" directory. Competitors are encouraged to develop their code from this dummy algorithm or to use it to help debug their existing code.

2.1 Computing Environment

Participant algorithms will be run on PNNL's high-performance cluster. Algorithms will be allowed to use a single node on cluster. The node characteristics are given below:

CPU Type	Clock Rate	Cores per Node	Node Memory
AMD EPYC 7502 (Dual)	2.5GHz boost to 3.35GHz	64	256GB

We intend to provide participants as much freedom as possible in designing their algorithm. PNNL and participants will work together to ensure participants can access their required environment. This includes coding language support, compilers, libraries/packages, and solvers. PNNL will acquire any software or solver licenses as needed.

Participants may choose to use any coding language and any solver they wish. PNNL already has support for the following coding languages:

Coding Languages
- Python
- Executables
- Java
- Julia
- GAMS

PNNL will support any additional coding languages requested by participants. Please contact the ESPA-Comp support team at ESPACompSupport@pnnl.gov for computing environment needs.

2.2 Computing Time Restrictions

To simulate real-world time restrictions faced by electric grid independent system operators (ISOs), we will impose time limits on participant algorithms. Time will be measured as the wall clock time for algorithm execution. Time limits will vary based on the market configuration as shown in the following table. For each market, the total time allocation equates to 3600 seconds (one hour) per simulated day. Participants can expect the computational overhead to execute (call, load environment, and import/export data) requires about 5 seconds, which is included in the time limit (i.e., the 10-second Real Time Market limit can be expected to allow 5 seconds of algorithm run time due to the 5 seconds of overhead).

Two-Settlement Market	Multi-Settlement Market	Rolling Horizon Market
Day Ahead Market: 720 sec.	Day Ahead Market: 720 sec.	Hourly Market: 25 sec.
Real Time Market: 10 sec.	Real Time Market: 10 sec.	15-Minute Market: 15 sec.
		5-Minute Market: 10 sec.

Code running beyond the time limit will be aborted. If no offer has been generated, the market will use the offer supplied for the previous interval.

2.3 Participant Algorithm Naming

Participant code will be invoked automatically by the PNNL market clearing code. For this to process, participant algorithms will be required to have the fixed name `market_participant`.

For a coding language with extensions (such as Python or Julia), the extension will be added after the name. For example:

```
market_particpant.py
```

For an algorithm written as an executable, the algorithm must be named `market_participant`.

2.4 Input Files

The simulated market will provide two input files for participant use at each market clearing interval: `market_data` and `resource_data`. These may be used by the participant algorithm to help determine the next offer. Sample files are provided in the espa-market-shell repository (see Section 2.6). These, along with an integer timestep, will be sent to participant codes as input arguments. For example, if a participant code runs in Python and is named 'market_participant.py' it will be invoked with the following command:

```
python market_participant.py timestep market_data resource_data
```

An executable with the name 'market_participant' would be invoked as:

```
./market_participant timestep market_data resource_data
```

The `timestep` argument will start at 1 and increment by 1 at each market clearing interval. The `market` and `resources` arguments will be in JSON format with the information described below:

Market input argument (`market_data`):

Market information will be provided to all participants at each market interval. This includes the unique market identifier for the given time step and the start times and durations for the time intervals that will be cleared in the next market. For example, in the two-settlement division, the day-ahead market (DAM) will have 36 time stamps with 60-minute power forecasts while the real-time market (RTM) will have 36 time stamps with 5-minute power forecast. Time stamps will be a string formatted as YYYYmmddHHMM where Y=year, m=month, d=day, H=hour, and M=minute. For example, 2:15pm on October 12th, 2023, will be '202310121415'. Duration will be given units of minutes.

This also includes a forecast for wind, solar, and demand, provided as a list of floating decimals in units of MW corresponding to the provided time intervals. Forecasted power will be provided as the aggregate of all renewable resources and demand across the entire system topology.

The last component will be a history of system wide wind, solar, demand, and price information along with a list of historic time steps. The history component will start as empty/null and build with each time step. Prices will be given in units of $/MWh.

The JSON dictionary structure is shown below to aid participants in setting up their algorithm to correctly gather and parse data provided by the ESPA-Comp simulation platform. Several sample market json files are provided in the WEASLE GitHub repository (github.com/pnnl/weasle), located in the competitor-tools/espa-market-shell directory. Section 2.5 provides further detail about the market shell.

```
market_data
{
"uid": "TSRTM202310121415",
"market_type": “TSRTM”,
"current_time": "202310121410"
"timestamps": [timestamp1, timestamp2, …],
"durations": [5, 5, 5, …],
"interval_type": [PHYS, FWD, FWD, …, ADVS, …],
"forecast_mw": {
     "wind": [ 453.4556, 478.9103, etc... (MW)],
     "solar": [ 125.6812, 117.0737, etc... (MW)],
     "load": [ 988.9316, 993.2208, etc... (MW)]
     },
"previous" : {
  "TSRTM": {
     "prev_uid": "TSRTM202310121410",
     "timestamp": [list of timestamps from previous market horizon],
     "prices": {
          "EN": {
               bus: [list of LMP ($/MWh)]
               },
          "RGU": [list of regulation up marginal prices ($/MWh)],
          "RGD": [list of regulation down marginal prices ($/MWh)],
          "SPR": [list of spinning reserve marginal prices ($/MWh)],
          "NSP": [list of non-spinning reserve marginal prices ($/MWh)]
          },
  "TSDAM": { … },
          },
"history": {
     "times": [list of historic times (YYYYMMDDHHMM - e.g. "201801280015")],
     "wind": [list of historic system-wide wind MW],
     "solar": [list of historic system-wide solar MW],
     "load": [list of historic system-wide demand MW],
     "prices": {
          "EN": {
               bus: [list of clearing-weighted LMP ($/MWh)],
               },
          "RGU": [list of clearing-weighted regulation up marginal prices
          ($/MWh)],
          "RGD": [list of clearing-weighted regulation down marginal prices
          ($/MWh)],
          "SPR": [list of clearing-weighted spinning reserve marginal prices
          ($/MWh)],
          "NSP": [list of clearing-weighted non-spinning reserve marginal prices
          ($/MWh)]
          }
     }
}
```

Information in the "previous" sub-dictionary reflects the previous market clearing of the current market type. For example, the market_data for a day ahead market will include the results from the previous day ahead market, and the market_data for a real time market will include the results from the previous real time market.

Resource input argument (`resource_data`):

Each participant will be given a unique `resource_data` input in JSON format relevant to their storage resource. This will contain four primary keys. The first is `status` and contains the latest physical dispatch information for the participant resource. This includes state-of-charge, temperature, and energy dispatched. There is also a key for the available cpu time for offer resources. It will also include degradation costs with the associated calculation times. Degradation costs will be computed once daily for each resource and include a timestamp of the calculation time. All values in `status` are incremental, that is, not cumulative.

The next key is `ledger` which provides a detailed list of every forward position cleared in the market. This is presented as a list of tuples of marginal quantities and marginal prices at each time step. While some time steps may only have a single forward position, in some market configurations each time step will have multiple forward positions. These price/quantity lists are divided by each of the five types of forward positions: physical energy (EN), regulation up/down (RGU/RGD), spinning/non-spinning reserves (SPR/NSP). All values in `ledger` are cumulative, including all cleared transactions to date.

The last key is `settlement`, which is derived at each time step from the `ledger`. The settlement value is a sum of the product of the marginal prices and quantities at each time step.

The final key is `score`, which is derived from the degradation cost information in "`status`" and resource revenue in "`settlement`." The total of all settlements across time, along with the degradation costs, will give each participant's profit at a given time. Keys for `revenue`, `cost`, and `profit` each provide a history of cumulative values at each timestamp. The resource's current score (i.e., profit), inclusive of all energy, ancillary services, and degradation revenues/costs, can be accessed under the key "`current`". All values in `score` are cumulative.

Both `ledger` and `settlement` will reflect any virtual bids made by the participant. These will be included with unique virtual identifiers (vid). All virtual bids will only contain energy (EN) positions since virtual bidding is not presently used for ancillary services.

The JSON dictionary structure is shown below. Several sample resource json files are provided in the espa-market-shell repo for further reference.

```
resource_data
{
"rid": your resources rid (used below and when submitting your offer),
"pid": your participant pid (used when submitting virtual offers),
"time_limit": available cpu time,
"status": {
     rid:  {
          "soc": latest_soc_value,
          "temp": latest_temp_value,
          "dispatch": latest_dispatch_value,
          "degradation": { timestamp: degradation_costs },
          },
     },
"ledger": {
     rid: {
          "EN": { timestamp: [(marginal quantity_1, marginal price_1), ….,
          (marginal quantity_n, marginal price_n)] },
          "RGU": { timestamp: [(marginal quantity_1, marginal price_1), ....,
          (marginal quantity_n, marginal price_n)] },
          "RGD": { timestamp: [(marginal quantity_1, marginal price_1), ....,
          (marginal quantity_n, marginal price_n)] },
          "SPR": { timestamp: [(marginal quantity_1, marginal price_1), ....,
          (marginal quantity_n, marginal price_n)] },
          "NSP": { timestamp: [(marginal quantity_1, marginal price_1), ....,
          (marginal quantity_n, marginal price_n)] }
          }
      vid: {
          "EN": { timestamp: [(marginal quantity_1, marginal price_1), ....,
          (marginal quantity_n, marginal price_n)] }
          },
      },
"settlement": {
     rid: {
          "EN": { timestamp: sum of P*Q for P,Q in prices/quantities from ledger
          },
          "RGU": { timestamp: sum of P*Q for P,Q in prices/quantities from
          ledger },
          "RGD": { timestamp: sum of P*Q for P,Q in prices/quantities from
          ledger },
          "SPR": { timestamp: sum of P*Q for P,Q in prices/quantities from
          ledger },
          "NSP": { timestamp: sum of P*Q for P,Q in prices/quantities from
          ledger },
          },
      vid: { "EN": { timestamp: sum of P*Q for P,Q in prices/quantities from
ledger } }
      },
"schedule": {
     rid: {
          "EN": { timestamp: sum of quantities from ledger },
          "RGU": { timestamp: sum of quantities from ledger },
          "RGD": { timestamp: sum of quantities from ledger },
          "SPR": { timestamp: sum of quantities from ledger },
          "NSP": { timestamp: sum of quantities from ledger },
          },
      vid: { "EN": { timestamp: sum of quantities from ledger } }
      },
"score": {
     "net_revenue": { timestamp: sum of settlement[rid][prod] for prod in [EN,
     RGU, RGD, SPR, NSP] + sum of settlement[vin]["EN"] for all vid },
     "degradation_cost": { timestamp: sum of status["degradation"] },
     "profit": { timestamp: score["net_revenue"][timestamp] –
     score["degradation_cost"][timestamp] },
     "current": latest profit value
     }
}
```

Information in the “status” sub-dictionary includes some data that is only provided to the competitor algorithm once. Competitors who wish to build algorithms that use a history of this data may save it to their local folder in json or other formats. An example of this is included in the dummy algorithm.

2.5 Output Files

Participant algorithms must submit offers in a format compatible with the market clearing code. Requirements of the offer format are described below. A sample participant offer format algorithm is embedded in ‘market_participant.py’ within the espa-market-shell. This may be used by participants or adapted to their chosen language. Sample offers are provided in the espa-market-shell and a few examples are shown at the end of this section.

Offer (`offer_[timestep].json`):

The offer must be created in a json file entitled `offer_[timestep].json`. The value `timestep` will be given as the first input argument passed to a participant algorithm (see Section 2.3) and will be an integer value. The time step will start at 1 and, for the two-settlement market, there will be 289 time steps per day. This comes from 288 5-minute physically dispatched intervals and one day-ahead market. The time step should not be padded with leading zeros, so offer files will be offer_1.json, offer_2.json, …, offer_10.json, offer_11.json, …, offer_100.json, etc.

The offer file must contain top level keys for each resource, using each resource’s unique resource ID (rid). For batteries, resource IDs will be assigned to a storage unit and will have the format R00001, R00002, etc. Participants may also submit virtual bids with a unique virtual resource ID (vid). The virtual id will have the format [pid]_[bus] which includes both a unique participant ID (pid) and the bus name where participants are submitting the virtual offer.

To illustrate, suppose a participant selects resource R000004 at the bus CIPV and that the participant was assigned pid=p002 when they registered for ESPA-Comp. The participant offer includes the resource offer for R000004, and the participant also elects to include a virtual offer at the CIPV bus. The virtual offer will have vid “p002_CIPV”. This offer would look like:

```
offer.json:
{
“R00004”: storage offer
“p002_CIPV”: virtual offer
}
```

The offers themselves must contain a particular set of keys to be accepted by the market simulator. For a storage offer, the following keys (with the list of corresponding formats) must be included (in any order) in `offer`:

Key Name	Format	Description
`cost_rgu`	Time series	Cost ($) incurred per MWh of regulation up reserve
`cost_rgd`	Time series	Cost ($) incurred per MWh of regulation down reserve
`cost_spr`	Time series	Cost ($) incurred per MWh of spinning reserve
`cost_nsp`	Time series	Cost ($) incurred per MWh of non-spinning reserve
`soc_begin`	Float	State-of-charge at the beginning of this market clearing interval in MWh
`init_en`	Float	Initial dispatch power of this resource in MW
`init_status`	Float	Charging status of this resource 1=charging, 0=discharge/hold
`ramp_dn`	Float	Maximum decrease in dispatch level in MW/min
`ramp_up`	Float	Maximum increase in dispatch level in MW/min
`chmax`	Time series	Maximum power dispatch during charging in MW
`dcmax`	Time series	Maximum power dispatch when discharging in MW
`socmax`	Float	Maximum state-of-charge of this resource in MWh
`socmin`	Float	Minimum state-of-charge of this resource in MWh
`eff_ch`	Float	Battery efficiency during charging mode
`eff_dc`	Float	Battery efficiency during discharging mode
`block_ch_mc`	Time series	Block offer for marginal cost of MW dispatched in charging mode
`block_dc_mc`	Time series	Block offer for marginal cost of MW dispatched in discharging mode
`block_soc_mc`	Time series	Block offer for marginal cost of MWh when bidding battery state-of-charge
`block_ch_mq`	Time series	Block offer for marginal quantity of MW dispatched in charging mode
`block_dc_mq`	Time series	Block offer for marginal quantity of MW dispatched in discharging mode
`block_soc_mq`	Time series	Block offer for marginal quantity of MWh when bidding battery state-of-charge
soc_end	Float	Desired minimum state-of-charge at the end of the market clearing interval
`bid_soc`	Boolean	Boolean (True/False) whether to bid state-of-charge

For a virtual offer, the following keys must be included (in any order) in `offer`:

Key Name	Format	Description
`block_dec_mq`	Time series	Block offer for quantity of a virtual decrement (DEC)
`block_inc_mq`	Time series	Block offer for quantity of a virtual increment (INC)

`block_v_mv`	Time series	Block offer for marginal value of a virtual DEC
`block_v_mc`	Time series	Block offer for marginal cost of a virtual INC

The “Format” column indicates whether the quantity vary over time (t) or whether it is a single value for the given market clearing interval. Keys that start with `block` can also include up to 10 unique offer blocks at each time in the time series.

To help clarify, we provide examples of time series keys, both block offers and not, along with single value keys. A sample offer with all required keys is also provided in the ESPA-Comp GitHub repository in the espa-market-shell directory (github.com/pnnl/weasle).

Block Offer Example:

In existing wholesale energy markets, resources are allowed to enter an offer block for the quantity of power they can supply and the cost to supply that power at a particular time in the market clearing. These can be interpreted as supply curves for each resource.

For ESPA-Comp, participants will be allowed to submit blocks of up to ten elements. The sum of the power blocks (in the case of charging [ch] or discharging [dc]) or energy (in the case of state-of-charge [soc]) must equal the power or energy capacity of the battery.

To illustrate, suppose a battery has a maximum discharging capacity of 50MW. Perhaps a participant would like to submit its marginal cost as $25/MWh. The participant wishes to submit a single block offer. For a single timestep the single block offer would include:

Quantity (Power): `{“block_dc_mq”: { time: [50] }}`

Cost: `{“block_dc_mc”: { time: [25] }}`

Suppose the battery has the same maximum discharge capacity of 50MW but would like to limit discharge rates. Perhaps a participant would like to manage battery temperature and limit degradation cost, or perhaps they would like to save capacity for a later time interval. The participant decides, however, if the LMP (price of energy at the resource node) increases, the participant would be willing to discharge more. For the first 20MW the participant selects a marginal cost of $25/MWh, but for every 5MW thereafter they would like to increase their offer cost by $3/MWh, up to the discharge capacity. This results in seven blocks. This offer would include:

Quantity (Power): `{“block_dc_mq”: { time: [20, 5, 5, 5, 5, 5, 5] }}`

Cost: `{“block_dc_mc”: { time: [25, 28, 31, 34, 37, 40, 43] }}`

Note that the values in the power quantity blocks (`block_dc_mq`) are the incremental power quantities. The sum of all values is equal to the discharge capacity of 50MW. The cost values, however, are the absolute cost. Suppose that in the market clearing, the LMP is determined to be $36/MWh. The first four block offers have a cost less than $36/MWh and so would be dispatched for a total of (20+5+5+5) MW = 35MW. The entire 35MW will be settled at the LMP of $36/MWh rather than the block bid prices (see Section 4 for details on settlement calculations).

Finally, suppose the participant would like to submit slight changes in their offer block at different time steps. Here we will examine a charging offer with four blocks. We will include just the first three timesteps of the day-ahead market (for the day October 12th, 2023). The first is 202310120000 (recall this is year:2023, month:10, day:12, hour:00, minute:00). The second and third increment one hour and are therefore 202310120100 and 202310120200. In this case, the offer could contain:

```
{ “block_ch_mq”:   {“202310120000”: [20, 10, 10, 10]},
                   {“202310120100”: [20, 10, 10, 10]},
                   {“202310120200”: [20, 10, 10, 10]}
},

{ “block_ch_mc”:   {“202310120000”: [21, 18, 15, 12]},
                   {“202310120100”: [20.4, 17.4, 14.4, 11.4]},
                   {“202310120200”: [21.9, 18.9, 15.9, 12.9]}
}
```

In this example the participant has fixed their marginal quantity of MW at a 20 for the first block and 10 for the next three (again summing to the maximum charge rate of 50MW). For the marginal cost, the block offer changes at each time step. The first block bids in $21/MWh at the first time step, $20.4/MWh at the second time step, and $21.9/MWh at the third time step. These values are each dropped $3/MWh for each block 2-4.

Other Time Series Example:

For time series offers, the quantity of interest is allowed to vary over time during this market clearing period. Note that in some cases a participant may wish to keep the quantity constant over time, in which case they may submit identical values at each time step.

For this example, let us examine the cost of providing regulation up, `cost_rgu`. Suppose we adopt a low cost of $3/MWh at the beginning of the day, increasing it by $1.5/MWh each hour as the day progresses (this would effectively signal that we would rather provide other services later, unless regulation compensation is sufficiently high). Looking at just the first three timesteps (for the day-ahead market on October 12th, 2023) the offer would include:

```
{ "cost_rgu":      {"202310120000": 3 },
                   {"202310120100": 4.5},
                   {"202310120200": 6}
}
```

Note that if we instead bid into the real-time market at midnight, the first time step will still be 20231012000. After that, changes come in 5-minute increments so the second timestep would be 202310120005, followed by 202310120010 and so on.

Single Value Example:

In this case each storage resource has only one value of the given quantity. Many of these are storage attributes that will not change over the course of a market simulation, although a few will vary. We will show an example using the soc_end, which is the minimum requested state-of-charge at the end of an entire interval. For instance, if a participant may want to retain some battery charge to increase their options for the next day.

Let's say the minimum state-of-charge soc_min is 20MWh and the maximum state-of-charge soc_max is 95MWh. A participant wants to have a charge of at least 40.6 MWh at the end of the day. Their offer would include:

```
{ "soc_max": 95 },
{ "soc_min": 20 },
{ "soc_end": 40.6 }
```

2.6 Market Shell Repository

To aid with verifying correct input/output formats, and to provide example offers, we have included a basic version of the market simulation script, called 'espa-market-shell', in the competitor-tools directory in the ESPA-Comp GitHub repo (github.com/pnnl/weasle). This repo is intended to allow participants to test the compatibility of their code with the various input and output format requirements.

The espa-market-shell is written in Python 3. We recommend installing the latest version of Python. The espa-market-shell requires the 'strenum' Python package as well as some packages that are standard with many installations such as Anaconda.

Once Python is installed and the repository is cloned, a participant can verify that market shell is working by invoking market simulator from the command line, or through an IDE.

```
python market_simulator.py
```

This should print out lines indicating that the simulation is running and that the offer format is valid at each time step. We provide examples of the various input and output files (described in sections 2.3 and 2.4) in the market shell.

In the system_data folder are:

`system_data/market[0-2].json` and `system_data/resources[0-2].json`

These contain input market and resources files for varying market conditions. The first files, market0.json and resources0.json, are for the initial run and have no history information or settlement values. The market1.json and resources1.json are several timesteps in and have a short history. Finally, market2.json and resources2.json are from a day into the run, meaning resources2.json also includes the first computed degradation cost. The market simulator will test that the algorithm returns acceptable offers in each of these cases.

In the folder 'offer_data/participant_p00001' are placeholder offers made at each of the three time steps:

```
offer_data/offer_[1-3].json
```

These offers include a single storage resource with resource id 'R000001' and a virtual resource for participant 'p00001' at bus 'CIPV' giving a virtual id of 'p00001_CIPV'.

To test a participant algorithm formatting, take the following steps:

- Within offer_data/participant_p00001, clone or link the participant GitHub repository, including the algorithm and any supporting scripts.
  (Note, if the code language is Python and the algorithm name is 'market_participant.py' we recommend copying the existing 'market_participant.py' to a backup folder before beginning.)
- Run from a command line (or within an IDE):
  ```
  python market_simulator.py -a [participant_algorithm_name] -l
  [participant_code_language]
  ```

If a participant needs a new language added to the market_shell, please contact PNNL (see Section 5.5).

When run, the simulator will first check to make sure that the algorithm has written an offer in offer_data/participants_p00001 with the name `offer_[time_step].json` (e.g. offer_1.json).

If an offer is found, the simulator will check to make sure it contains all required keys and that the data values are in the expected formats (such as lists for block offers). It will also send a warning if the offer includes extra keys. These will not affect the market clearing, but any extra keys will be ignored.

Participants may also test the creation of virtual offers. Within an offer, participants may add a key with the virtual id and the corresponding virtual offer. The vid format is [pid]_[bus] where the market shell uses a default pid of 'p00001'. The bus number will be selected from the list of balancing authorities in the WECC model (see the Competition Overview, available on the ESPA-Comp website, for details). This list includes:

Balancing Authority (Bus) List
AVA, AZPS, BANC, BPAT, CHPD, CIPB, CIPV, CISC, CISD, DOPD, EPE, GCPD, IID, IPFE, IPMV, IPTV, LDWP, NEVP, NWMT, PACW, PAID, PAUT, PAWY, PGE, PNM, PSCO, PSEI, SCL, SPPC, SRP, TEPC, TH_Malin, TH_Mead, TH_PV, TIDC, TPWR, VEA, WACM, WALC, WAUW

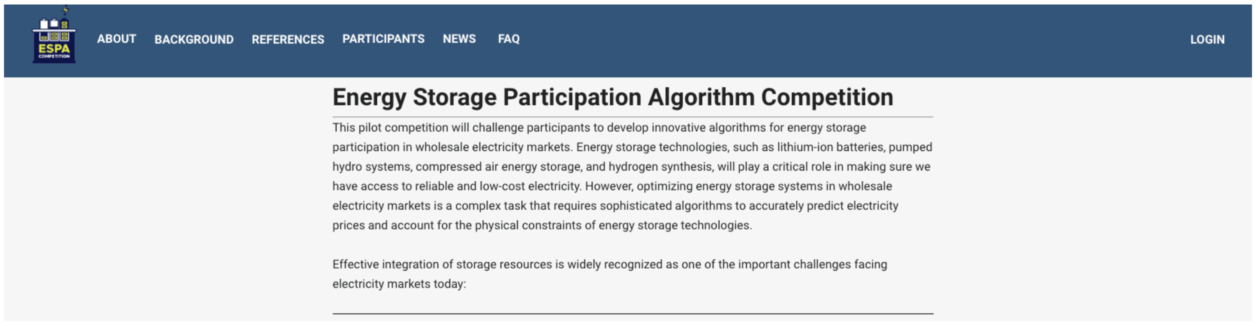
In the market shell, the bus number will not be validated, but in sandbox trials and in the competition the bus number must be in the range of available busses provided above, otherwise the virtual offer will be ignored.

3.0 Algorithm Submission

Before submitting an algorithm, participants must email ESPACompSupport@pnnl.gov to register and select a storage unit. We also recommend testing your algorithm on your own system and checking formatting with the espa-market-shell (Section 2.5). Once participants are ready, they can proceed to algorithm submissions. There are two submission modes: sandbox and competition.

3.1 Signing Up

To register for the contest, participants can go to the ESPA-Comp website to sign up. Visit https://espa-competition.pnnl.gov/:

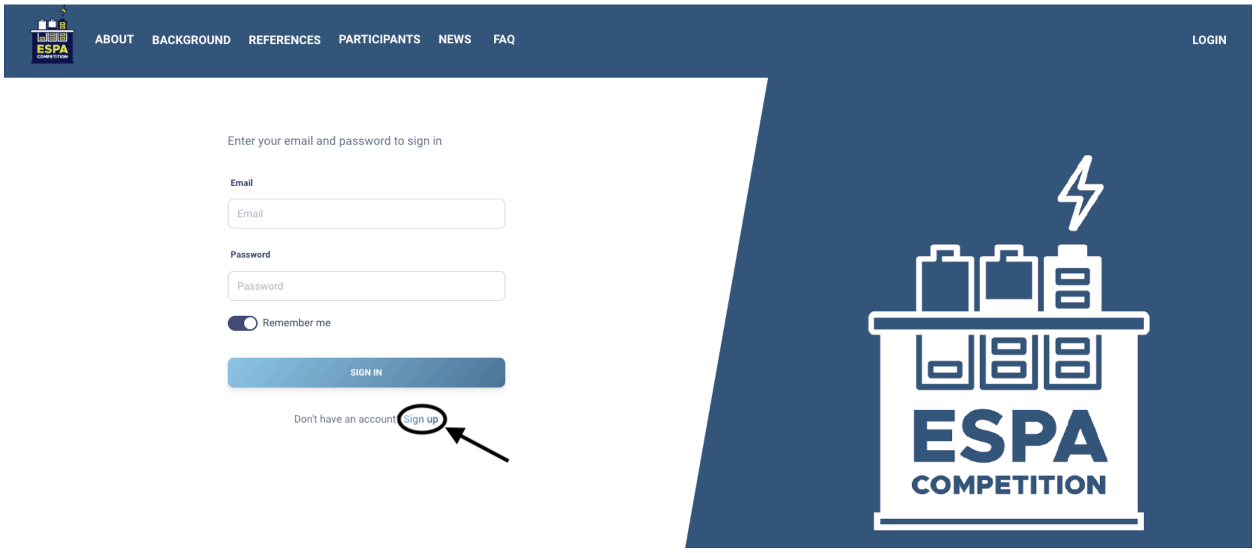


Energy Storage Participation Algorithm Competition

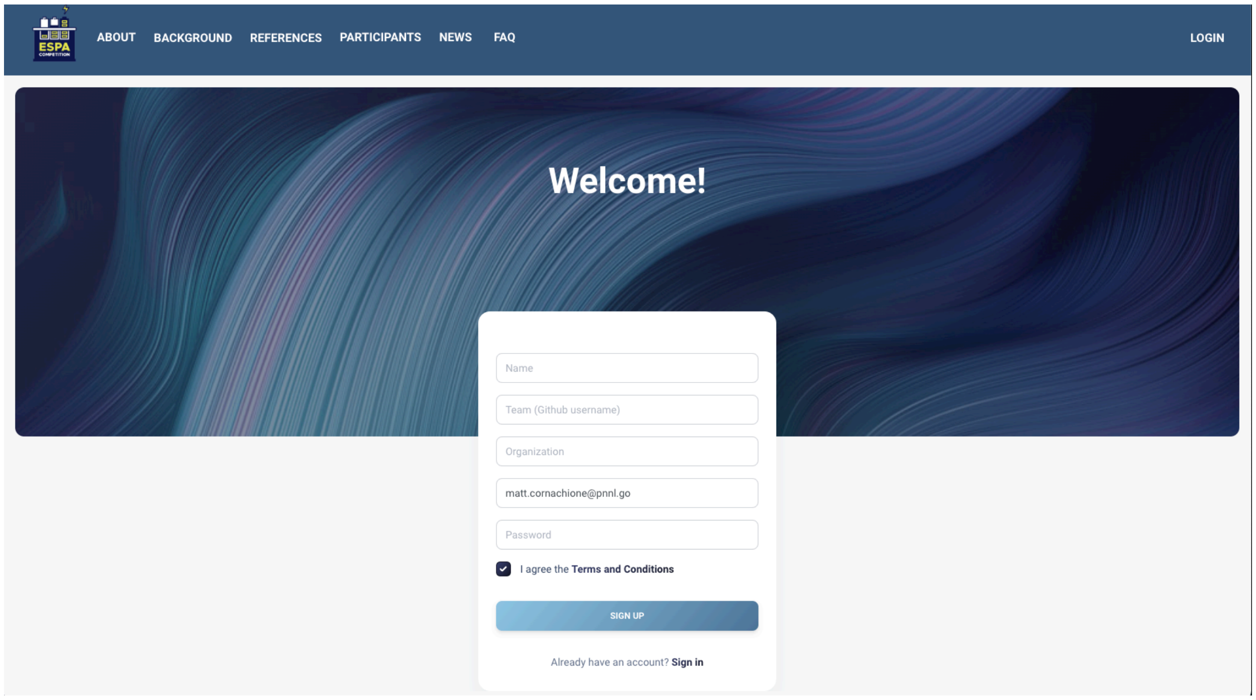
This pilot competition will challenge participants to develop innovative algorithms for energy storage participation in wholesale electricity markets. Energy storage technologies, such as lithium-ion batteries, pumped hydro systems, compressed air energy storage, and hydrogen synthesis, will play a critical role in making sure we have access to reliable and low-cost electricity. However, optimizing energy storage systems in wholesale electricity markets is a complex task that requires sophisticated algorithms to accurately predict electricity prices and account for the physical constraints of energy storage technologies.

Effective integration of storage resources is widely recognized as one of the important challenges facing electricity markets today:

Click LOGIN in the upper right corner to be taken to the sign in page:

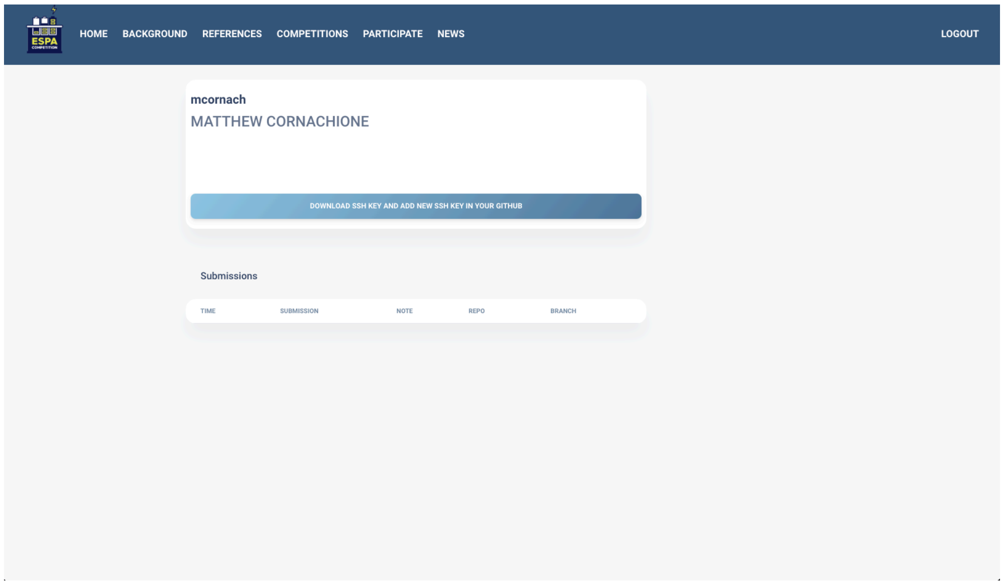


Under the SIGN IN button click Sign up (after ‘Don’t have an account?).



Choose a name (this can be an individual or a team name). Under Team, enter the team/individual Github username as prompted. Fill out the organization and email then choose a password and agree to the terms and conditions.

After signing up, the website will redirect to the Team Dashboard.

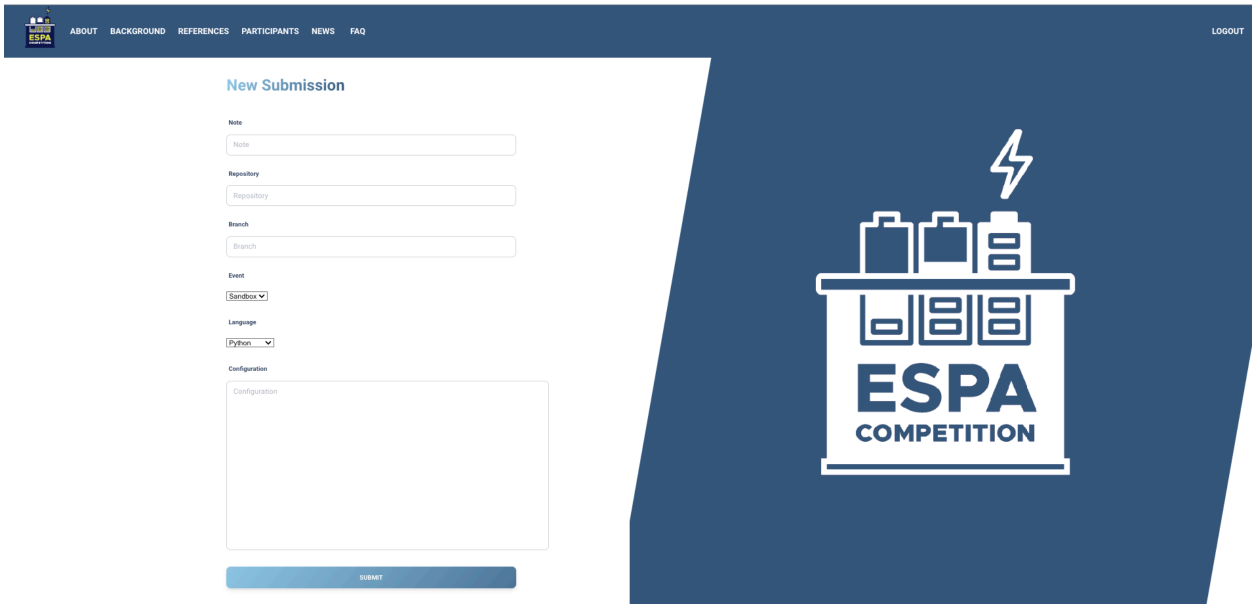


From the Dashboard, click the button “Download SSH key and add new SSH key in your Github” to download a text file with the PNNL cluster’s public ssh key. Add this to the profile in the team Github account to allow PNNL to clone your repository onto the PNNL cluster. (See https://docs.github.com/en/authentication/connecting-to-github-with-ssh/adding-a-new-ssh-key-to-your-github-account for instructions on adding SSH keys). Note also that your repository must be made public to allow PNNL to clone it onto our cluster.

3.2 Sandbox Submissions

The ESPA-Comp sandbox allows participants to submit algorithms for a trial test in the full market clearing engine. Participants can select custom time windows and market configurations with each submission. Sandbox submissions will return the final settlement scores over the period of a simulated market, allowing participants to assess the performance of their algorithm and make adjustments as needed.

Once the sign-in process is complete, including adding the SSH key to GitHub, click “Participants” , then click “New Team Submission” to reach the submissions page:



In the first line, enter a note about the submission (such as First Trial). In repository, enter the name of the GitHub repository only (not including the github username). List the branch of this repo, such as main. Make sure the Event dropdown is on Sandbox. In the language dropdown select your code language.

In the configuration box you can enter optional market specifications to choose the characteristics of your run. The following keywords are available:

- mktSpec = (choose from TS, MS, or RHF)
- horizon = (integer with the length of time, max 30 days, min 5 minutes)
- horizon_type = (units of the duration, either ‘days’, ‘hours’, or ‘minutes’)
- start_date = (select the start date in YYYYmmddHHMM. This will choose times from the forecast. Any year may be selected, but valid start dates must be between YYYY01010000 and YYYY12010000)
- str_bus = (bus name where the storage resource will be located – choose from available buses in the Competition Overview. If None, will randomly select.)

Defaults are given below:

```
mktSpec = TS
horizon = 25
horizon_type = minutes
start_date = 202302010000
str_bus = None
```

These keywords allow participants to try different market types or longer/shorter trials. For initial testing we suggest using a short horizon of 10-60 minutes. This will ensure you get prompt results and can troubleshoot any errors. You can later extend to hours or days (competition simulations will run for 30 days).

Here you must also enter configuration keywords describing the computing environment you need. This includes packages, libraries, and solvers. For assistance in setting up the configuration, please contact ESPA-Comp support (see Section 5.5) to discuss your needs. The ESPA team will create a configuration file and email it to you.

Once the submission is made, PNNL will immediately start processing it on the PNNL system. To view any dates click the unique submission id on the Team Dashboard. You should see Updates for submission progress steps. When the submission is complete, the Dashboard will show a status of “Results” with a note: Submission complete. This will include the address of a tar.gz zipped file with log and results data from the Submission. Copy this link into a browser to download the results. Sandbox scores are non-binding but can help determine how well an algorithm is performing.

3.3 Competition Submissions

Competition submissions will be similar to sandbox submissions. Participants will select “Competition” from the event dropdown. Furthermore, participants will not include the keywords mktSpec, duration, duration_type, start_date, and str_bus (if included, they will be ignored). PNNL will set the market configuration for each competition trial. Competitions will be run in the following three divisions:

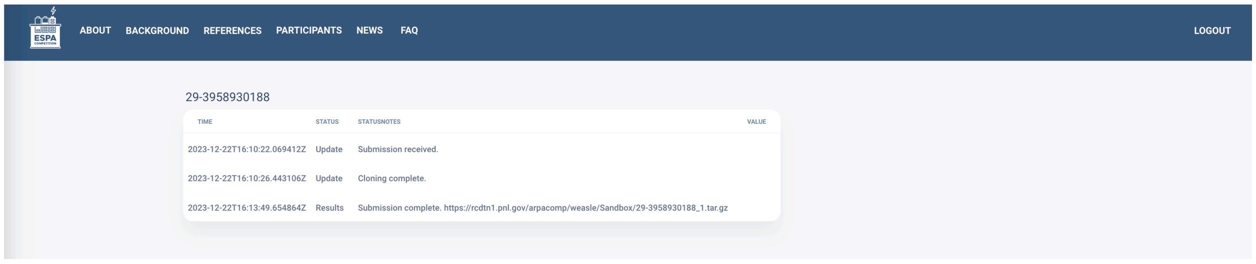
- 1: Two-settlement market
- 2: Multi-settlement market
- 3: Rolling horizon market

During each competition, participant code will not be run immediately upon submission. Instead, PNNL will provide the start dates of each competition. Prior to the start date, participants will be given a window during which to submit an algorithm for a particular trial. Participants may submit multiple times during that window, but once the window is closed, the last submitted algorithm will be used in the competition.

There will be a separate algorithm submission for each division (two-settlement, multi-settlement, and rolling horizon market types). Participants may submit an updated algorithm for each division. At the close of each division’s market simulation, all participants will be provided their score via the web interface (see Section 4).

3.4 Submission Results

After the submission's simulations are completed, an archive file can be downloaded from the team dashboard. Click on the submission number and you will see the submission message page (see below). To find the archive, copy and paste the address into a new tab (starting with 'https,' ending with 'tar.gz').



29-3958930188

TIME	STATUS	STATUSNOTES	VALUE
2023-12-22T16:10:22.069412Z	Update	Submission received.	
2023-12-22T16:10:26.443106Z	Update	Cloning complete.	
2023-12-22T16:13:49.654864Z	Results	Submission complete. https://rcdtn1.pnl.gov/arpacomp/weasle/Sandbox/29-3958930188_1.tar.gz	

The archive includes all log files produced by the competitor's code, all JSON offer files produced by the competitor's code, and the final JSON "resource" file that contains the storage resource's status, ledger, settlement, and score data. Participants can use this information to understand how their resource was cleared by the market and how their total profits were determined.

4.0 Score Evaluation

Participants will be provided equivalent batteries with identical storage capacity and operational characteristics. Scores will be the market surplus of the storage resource at the conclusion of the competition. Separate scores will be assigned for each market configuration: Two-settlement, Multi-Settlement, and Rolling Horizon.

4.1 Surplus Components

Within each market configuration the profit will consist of the following components:

- z_e = Real energy settlements (charging and discharging)
- z_r = Settlements for ancillary services (regulation up/down, reserves)
- z_v = Settlements for virtual bids (DECs and INCs)
- z_d = Battery degradation cost (loss in value of your battery over time)

This will be computed with a simple summation:

$$z_{\text{score}} = z_{\text{e}} + z_{\text{r}} + z_v + z_d$$

where the degradation cost, z_d, will always be a negative number. Ancillary services, z_r, will always be a positive number. Real energy, z_e, includes positive components (income from discharging) and negative components (expenses from charging). Virtual energy, z_v, may also include positive and negative components from INCs and DECs.

4.2 Settlement Calculation

A complete description of the market settlement calculation is provided in the Model Formulations document, Section 2.4, provided under the References menu on the ESPA-Comp webpage. Here we provide a brief summary of the key points, including examples.

For every market type a participant will have at least one and perhaps many forward positions at a particular time increment in the market. The settlement at a particular time step will be computed as the sum of forward positions times the LMP (locational marginal price) at the bus, as determined by the market clearing code. Note that both physical and forward positions are updated only as the change in energy from a previous forward position. Thus for a particular resource, at a time step and a market (m) the change in energy is found by

$$p_m(t) = \delta p_m(t) + P_{fwd}(t)$$

or

$$\delta p_m(t) = p_m(t) - P_{fwd}(t)$$

Here $p_m(t)$ is the *dispatched* power at time t and $\delta p_m(t)$ is the *settled* power. The forward settled power, $P_{fwd}(t)$, is the sum of the settled power in all existing market positions (m):

$$P_{fwd}(t) = \sum_m \delta p_m(t)$$

Thus the settlement for a given resource n at a particular time t is given by the sum over all markets (where the settled power $\delta p_m(t)$ is also only that from resource n) :

$$z_{n(t)} = \sum_m \delta p_m(t)\, LMP_m(t)\, \delta t_m$$

This is the settlement for a single resource at a single time. Notice that, as in real energy wholesale markets, the participant's energy price is set by the LMP, not by the bid price. Each participant's energy settlement will be summed over all resources and all times:

$$z_e = \sum_{n,t} z_n(t)$$

To illustrate we consider a two-settlement market and look at the settlement at midnight (time=00:00) on October 12, 2023 (we will call this t1=20231012000). Let us suppose that in the DAM, which clears on October 11, 2023 at 12:00, the market clearing provides a forward dispatch of $p_{DAM}(t_1)$ = 27MW discharged (supplied) at $LMP(t_1)$ = \$85/MWh. The change in power is given by:

$$\delta p_{DAM}(t_1) = p_{DAM}(t_1) + P_{fwd}(t_1)$$

Since there are not yet any forward periods set in previous markets $P_{fwd}(t_1)$ = 0. This means that:

$$\delta p_{DAM}(t_1) = p_{DAM}(t_1) = 27MW$$

The initial settlement by is therefore:

$$z(t) = \delta p_{DAM}(t_1)\, LMP_{DAM}(t_1)\, \delta t_m = 27MW \cdot \$85/MWh \cdot 1\,hr = \$2{,}295$$

Note that the duration is 1hr since the day-ahead market clears at hourly intervals.

Now we advance in time to the real time market. Suppose now at t1 = 00:00 on October 12, 2023 the RTM clearing provides an updated dispatch of $p_{RTM}(t_1)$ = 24MW discharged at $LMP_{RTM}(t_1)$ = \$83/MWh. We must now compute the adjustment to the forward settlement. We can compute $P_{fwd}(t_1)$ as

$$P_{fwd}(t_1) = \sum_m p_m(t_1) = 27MW$$

since the only cleared market at time t1 was our DAM. Thus, we can find the change in power

$$\delta p_{RTM}(t_1) = p_{RTM}(t_1) - P_{fwd}(t_1) = 24 - 27 = -3MW$$

This indicates that we are dispatching 3MW less than previously committed. The full settlement at time t1 is now the sum of the DAM settlement and the RTM market settlement. For the RTM we clear on five-minute intervals so the associated δt_m is 5/60 hr.

$$z(t_1) = \sum_m \delta p_m(t_1)\, LMP_m(t_1)\, \delta t_m =$$

$$\underbrace{(27MW \cdot \$85/MWh \cdot 1hr)}_{\text{DAM}} + \underbrace{(-3MW \cdot \$83/MWh \cdot 5/60\,hr)}_{\text{RTM}} = \$2{,}295 - \$20.75 = \$2{,}274.25$$

The settlement at t1 has decreased because the resource supplied less power than the forward position. Only the difference, however, is settled at the new LMP of $83/MWh. The original DAM settlement is unchanged. As the real-time market advances, there will be 11 settlements within the hour (since there are a total of 12 RTM clearings in an hour). As we have formulated it, the DAM settlement will only be counted once, then each RTM interval will increment or decrement the settlement depending on the actual dispatched power and LMP relative to the DAM forward position.

While the provided example only covered the energy surplus, z_e, identical computations apply to the ancillary services, z_r. The same equations also apply to virtual offers, the only difference being that no virtual offers are cleared in the physical dispatch period of the RTM. The degradation component is described in the following section.

4.3 Degradation Cost

For the ESPA-Comp, we will implement a physics-based degradation model described in the Model Formulations document, Section 4.1.3, provided under the References menu on the ESPA-Comp webpage. Here we provide guidance on the key factors affecting degradation. Participants may choose whether to address these in their algorithm development.

The cost of degradation can be divided into four components as follows:

$$c_{\text{degradation}} = c_{\text{cyc}} + c_{\text{therm}} + c_{\text{SoC}} + c_{\text{DoD}}$$

First is c_{cyc} or cycling degradation. This is primarily affected by degradation over time, but also by how much current flows in and out of the battery. As a battery cycles, its performance degrades and value decreases.

Second is c_{therm}or thermal degradation. Batteries tend to retain performance better when operated at lower temperatures. In the resources input (see Section 2.3), we will provide participants with their battery temperature (temp). This will allow participants to monitor temperature and restrict charging/discharging when temperatures get high, if they so wish.

The third and fourth components c_{SoC} and c_{DoD} measure costs associated with the battery's state-of-charge and depth of discharge. These are both related in that a battery tends to degrade less when operated near its maximal state-of-charge. Cycles with deep depth of discharge (for example, draining the battery to the minimum state-of-charge) tend to degrade performance more quickly than shallower discharges.

Participants may choose to monitor their battery characteristics and consider bidding strategies that balance degradation costs with energy and ancillary settlement surplus.

5.0 Troubleshooting

Because the ESPA-Comp requires frequent back-and-forth communication between the market clearing code and participant algorithms and because participant algorithms must run on the PNNL high-performance cluster, we expect challenges to arise in the development process. While the following is not a complete guide to all problems, it should help guide participants through the troubleshooting process to get algorithms running as expected.

5.1 Interfacing with Market Clearing Code

As described in Section 2.0, participant algorithms must be able to accept inputs and provide outputs in a format compatible with the market clearing code. The espa-market-shell described in Section 2.5 is designed to aid this process. Please follow the setup instructions provided in section, reaching out to ESPA-Comp Support if any issues arise.

The espa-market-shell provides the same directory structure where participant algorithms will be placed on the PNNL cluster. It also provides the same input file format and the same time intervals that will be provided during the competition. Participants can examine these to ensure their algorithm can interpret the market clearing code's output files.

The espa-market-shell also has a limited offer validation scheme. While this will not measure the performance of an offer, it will ensure that an offer is in the correct directory, correct format, and has all appropriate keys and values. If the espa-market-shell accepts participant algorithm offers, then a participant is ready to proceed to sandbox trials.

The espa-market-shell also allows participants to test their algorithms in all three markets, TS, MS, and RHF. The validation scheme will primarily check that the offer returned includes the appropriate time steps for a given market configuration. This can be especially complicated in the RHF market which has several different time step variations, including non-uniform time steps.

5.2 Website and Submission Issues

If participants run into problems with the ESPA-Comp website (such as problems signing up or signing in, difficultly downloading ssh keys, or problems when trying to make a submission), please try the following steps before contacting ESPA-Comp support.

- Restart web browser
- Try a different browser (Edge, Chrome, Safari, Firefox, etc.)
- Clear your browser cache

If a problem persists, please contact ESPA-Comp support for troubleshooting assistance.

5.3 Sandbox Submissions

Sandbox submissions are likely to require the most troubleshooting steps. We recommend submitting initial submissions for the minimum time step of 5 minutes. This will help ensure a prompt reporting of any errors.

Once participants make a sandbox submission (see Section 3.2), their Github repository will be cloned on the PNNL cluster, the market clearing code will initiate a simulation, then it will attempt to call your code and request an offer. When the market clearing code is completed, or forced to exit, the results of your submission will be posted on the ESPA-Comp website under the Team Dashboard. This will include a log file and, if the simulation is successful, a scoring report.

The log file will contain any error messages encountered and will be a valuable tool for troubleshooting. Possible errors include:

Trouble cloning repository

- Ensure that the ESPA-Comp ssh keys have been added to the correct Github profile.
- Ensure the team has entered the correct Github username, repo name, and branch name.
- Ensure your repository is public.

Missing required packages, libraries, or solvers

- Ensure that the configuration box, provided for submissions, includes all package names in the custom configuration file sent by PNNL.
- If your code is compiled and/or using parallel processing, ensure you have requested the necessary compilers and MPI (message passing interface).
- If you have requested all necessary configuration files and still have errors, please contact ESPA-Comp support and we will help verify all required versions and licences are available on the PNNL system.

Runtime Errors

- A number of runtime errors can be raised, either by the market clearing code or by a participant algorithm.
- Participants may choose to include logging in their algorithm. This log will be returned and may help diagnose what is causing the error. Participants may then adjust the algorithm and make a new submission, iterating until all errors are resolved.
- In the case of segmentation faults, if participants can help identify the line(s) in your algorithm that are causing the fault before contacting ESPA-Comp support, this may help us more quickly resolve the problem.
- Errors returned by the market clearing code likely indicate that either files are not found in the correct place or formats are incorrect. If you have not already verified formats with the espa-market-shell, we recommend using it to help troubleshoot.
- If your results show an error you cannot interpret, please contact ESPA-Comp support.

Unexpected Results

- Once the algorithm successfully runs, participants are advised to validate that the settlement figures make sense and match expectations.
- In the case of unexpected results, please revisit documentation on settlement calculations (see Section 4 and the Model Formulations document) to ensure the algorithm behavior aligns with the settlement formulation. Participants may wish to adjust their algorithm and try a new submission.

- If a participant cannot determine the source of any unexpected results, please contact ESPA-Comp support and we may be able to provide additional information on the market clearing output to help determine the source of the unexpected behavior.

5.4 Competition Submissions

We recommend participants ensure their algorithm works without errors in the sandbox submissions before the competition begins. If an algorithm works in the sandbox mode, it is highly likely to work for the competition. Once the competition submissions window is closed, PNNL will simultaneously process algorithms from all participants in a simulated wholesale market. PNNL will attempt to troubleshoot any issues with the algorithms. If we are unable to resolve an issue, we may exclude an algorithm from the competition trial, which will result in a score of zero. Similar to sandbox submissions, the results from the competition simulation will be posted to the team dashboard.

After each competition division (the TS, MS, or RHF market) is run, participants will have an opportunity to make a new submission for the next market. During the open window, participants will be able to validate that an algorithm is working using the sandbox mode. If participants choose to submit a new algorithm, please verify functionality before the start of the competition to help the simulation proceed in a timely manner and ensure that a valid score is returned.

5.5 PNNL Support

For any questions or inquiries, please contact ESPACompSupport@pnnl.gov.

Pacific Northwest National Laboratory

902 Battelle Boulevard
P.O. Box 999
Richland, WA 99354
1-888-375-PNNL (7665)

www.pnnl.gov