Independent Statistics and Analysis ™

U.S. Energy Information Administration

Hydrocarbon Supply Model of the National Energy Modeling System: Model Documentation 2026

July 2026

www.eia.gov

U.S. Department of Energy

Washington, DC 20585

The U.S. Energy Information Administration (EIA), the statistical and analytical agency within the U.S. Department of Energy (DOE), prepared this report. By law, our data, analyses, and forecasts are independent of approval by any other officer or employee of the U.S. Government. The views in this report do not represent those of DOE or any other federal agencies.

OVERVIEW

HSM answers a question central to the U.S. energy outlook: given the prices set by energy markets, how much crude oil and natural gas will the United States produce over the coming decades, and from which regions and resources? It is a structural economic model of U.S. upstream oil and gas supply, written in Python and a component of the National Energy Modeling System (*NEMS*) at the U.S. Energy Information Administration (*EIA*). HSM projects domestic crude oil and natural gas production for the United States and natural gas available for import from Canada, and it serves as the upstream supply component of the *Annual Energy Outlook* (*AEO*).

HSM evaluates thousands of oil and gas projects each year using discounted cash flow (*DCF*) analysis, ranking them by profitability and drilling the most economic ones subject to rig, footage, and capital constraints. The model integrates with NEMS to receive price signals from the Liquid Fuels Market Module (*LFMM*) and Natural Gas Market Module (*NGMM*) and returns production volumes, well counts, and CO_2 capture economics.

HSM succeeded the Oil and Gas Supply Module (*OGSM*) in AEO2025; see *Historical Context* on the overview page for lineage and what changed for Outlook readers.

Geographic scope: 13 U.S. oil and gas regions — 7 Lower 48 onshore regions, three Lower 48 offshore regions (Gulf of America, Atlantic, Pacific), and three Alaska regions (onshore north slope, offshore north slope, and other Alaska) — represented at a more granular level as 84 oil and gas districts, plus two Canada regions (East and West) modeled separately.

Commodities: Crude oil, natural gas, natural gas plant liquids (*NGPL*), and CO_2 capture at natural gas processing plants.

Temporal scope: Long-term, annual projections, with the range varying by AEO publication year. For AEO2026, historical data run through 2024 with projections from 2025 through 2050.

> **Note**
>
> For non-technical readers, the *Overview* page provides a model summary without code. Developers should start at *Architecture* or *Model Run Sequence*.

CHAPTER

ONE

SUGGESTED PATHS

- **Analyst or policy reader:** *Overview* → *Model Run Sequence* → *NEMS Integration*
- **Developer:** *Getting Started* → *Model Run Sequence* → *Architecture* → *Running HSM — execution details*
- **Data maintainer:** *Model Run Sequence* → *Data Reference* → *Input Files Reference* → *Historical Data and STEO Overrides*

CHAPTER

TWO

SITE MAP

Section	Contents
Overview	Model purpose, geographic scope, commodity coverage, and NEMS context
Getting Started	Prerequisites, directory layout, and how to build and run HSM
Architecture	Module dependency graph, class hierarchy, and file layout
Submodules	Onshore, Offshore, Alaska, Canada, and NGP submodule details
Methodology	DCF economic framework, drilling equations, price calculations; shared mathematical notation and symbol lookup on every methodology page
Data Reference	Input files, NEMS restart inputs, output variables, and historical/STEO data sources
Running HSM — execution details	Running, debugging, and utility tools
API Reference	Auto-generated reference for the published Python modules
Glossary	Abbreviations and terms used throughout HSM documentation

2.1 Overview

Note

This page is for non-technical readers. Developers and analysts should start at *Architecture* or *Model Run Sequence*.

Every year, EIA publishes long-term projections of U.S. energy production, consumption, and prices in the *Annual Energy Outlook* (AEO). The Hydrocarbon Supply Model (HSM) produces the oil and gas supply side of those projections: it estimates how much domestic crude oil and natural gas the United States will produce through the projection horizon, and how much natural gas will be available for import from Canada.

HSM is a component of the National Energy Modeling System (NEMS) at the U.S. Energy Information Administration (EIA), and it serves as the upstream supply module for the AEO.

2.1.1 What HSM Does

HSM answers a fundamental question for each AEO projection cycle: **Given the oil and gas prices set by energy markets, how much domestic oil and gas will be produced, and from which geographic areas and resource types?** Prices come into HSM from the rest of NEMS; HSM decides how much production those prices call forth.

To answer this question, HSM:

1. **Receives oil and gas price signals** from NEMS market modules

2. **Evaluates thousands of candidate projects** using *discounted cash flow (DCF)* analysis — each project represents a drilling opportunity in a specific geographic area and resource category
3. **Drills the most economic projects** subject to physical constraints on drilling activity (rig availability, footage, capital)
4. **Returns production volumes** to NEMS — firm crude oil volumes to LFMM and expected natural gas volumes (associated-dissolved and non-associated) to NGMM for market clearing

This process repeats each model year across the full projection horizon, with prices and drilling activity evolving as supply and demand conditions change. In the final reporting iteration of each year, HSM receives realized natural gas production from NGMM and adjusts its drilling schedule to match those market-equilibrated volumes.

2.1.2 Historical Context

For many prior *Annual Energy Outlook* (AEO) cycles, the Oil and Gas Supply Module (OGSM) was NEMS's upstream supply module: it projected domestic oil and gas production from price signals and returned volumes to the liquid fuels and natural gas market modules. HSM replaced *OGSM* starting with AEO2025. EIA reimplemented upstream supply in Python to support project-level *DCF* economics, clearer maintenance, and integration with modern data and software workflows.

For readers comparing Outlooks across years, the role in NEMS is unchanged—HSM still models upstream supply only, still uses DCF ranking with drilling constraints, and still feeds the *Liquid Fuels Market Module (LFMM)* and *Natural Gas Market Module (NGMM)*. Methods, geography, and project definitions evolved with the new codebase; see the *Methodology* and *Submodules* pages for current detail rather than assuming one-to-one continuity with OGSM-era results.

2.1.3 Geographic Scope

HSM covers 13 U.S. oil and gas regions — represented at a more granular level as 84 oil and gas districts — plus two Canada regions modeled separately, organized across five submodules:

Submodule	Geography	Commodities	Regions
Lower 48 Onshore Submodule	Continental U.S. onshore	Crude oil, natural gas, *NGPL*, CO_2 (*EOR*)	7 regions (districts 1–66)
Lower 48 Offshore Submodule	U.S. *Outer Continental Shelf*	Crude oil, natural gas	3 federal *OCS* areas + state offshore
Alaska Crude Oil Submodule	Alaska	Crude oil	3 regions (offshore north, onshore north, other Alaska)
Canada Natural Gas Submodule	Canada	Natural gas (exports to U.S.)	2 regions (East, West Canada)
CO_2 Capture at Natural Gas Processing Plants Submodule (NGP)	U.S. natural gas processing plants	CO_2 capture (supply to *EOR*)	By HSM district

The district-to-region crosswalk is in `mapping.csv`; see *NEMS Integration* for how districts connect to LFMM and NGMM.

Temporal scope: Long-term, annual projections, with the range varying by AEO publication year. For AEO2026, historical data run through 2024 with projections from 2025 through 2050.

2.1.4 Commodity Coverage

Commodity	Submodules	NEMS consumer
Crude oil	Onshore, Offshore, Alaska	*LFMM*
Natural gas (*non-associated*)	Onshore, Offshore, Canada	*NGMM*
Natural gas (*associated-dissolved*)	Onshore, Offshore, Alaska, Canada	*NGMM*
Natural gas plant liquids (NGPL)	Onshore, Offshore, Alaska	*LFMM*
CO_2 (capture supply)	*NGP*	*CCATS* (Carbon Capture, Allocation, Transportation, and Sequestration)
CO_2 (*EOR* demand)	Onshore EOR	*CCATS*

Note

Alaska non-associated natural gas does not appear above because NGMM projects Alaska gas production internally, using local consumption projections, LNG export volumes, and the constraint that Alaska's gas market currently has no pipeline connection to Lower 48 or Canadian markets.

2.1.5 Resource Types

HSM models the full range of U.S. hydrocarbon resource types:

- **Tight oil and shale oil** — unconventional oil from low-permeability formations
- **Shale gas and tight gas** — unconventional natural gas
- **Conventional** — oil and gas from traditional reservoir formations
- ***Coalbed methane (CBM)*** — methane from coal seams
- **CO_2 injection and steam injection** — enhanced oil recovery from mature conventional fields (EOR in unconventional formations is not represented)

HSM tracks existing producing wells separately from new drilling activity. Production from active wells declines along empirical curves; the resource types above govern where and how HSM evaluates and projects new drilling. Alaska and Canada supply are modeled at the project or regional level rather than by resource type; see the *Alaska Crude Oil Submodule* and *Canada Natural Gas Submodule* pages for detail.

2.1.6 What HSM Does NOT Model

HSM models upstream crude oil and natural gas exploration, development, and production activities. It does not model the following:

- **Demand** — HSM models supply only; demand is modeled elsewhere in NEMS (*LFMM*, *NGMM*)
- **Price formation** — Prices are inputs to HSM, not outputs. HSM responds to prices; the supply-demand-price equilibrium that sets them emerges from NEMS iteration across all modules (see *Role in AEO*)
- **Pipeline infrastructure** — Pipeline capacity and flow constraints are handled by *NGMM*, not HSM
- **Refinery operations** — Crude quality allocation is done in LFMM, not HSM
- **LNG exports** — Governed by NGMM and trade modules in NEMS

2.1.7 Role in AEO

HSM runs within every NEMS iteration for every AEO projection year. Price signals from liquid fuels and natural gas markets feed into HSM's economic evaluations; HSM's production volumes feed back into those markets. This feedback loop iterates to convergence, producing the internally consistent supply-demand-price equilibrium that characterizes each AEO scenario.

2.1.8 How These Pages Relate

Read these overview pages in order:

1. **This page** — Purpose, scope, historical context, and what HSM does and does not model.
2. **Model run sequence** (*Model Run Sequence*) — Step-by-step execution inside each NEMS call.
3. **NEMS integration** (*NEMS Integration*) — How NEMS invokes HSM, restart files, iteration flags (`fcrl`, `ncrl`), and data flows to LFMM, NGMM, MAM, and CCATS.
4. **Next steps by audience** — *Architecture* and *Getting Started* for developers; *Submodules* and *Methodology* for economics and geography detail.

2.1.9 Related Pages

- *Model Run Sequence* — Step-by-step model execution
- *NEMS Integration* — Data flows between HSM and other NEMS modules
- *Architecture* — Technical architecture overview
- *Submodules* — Submodule descriptions

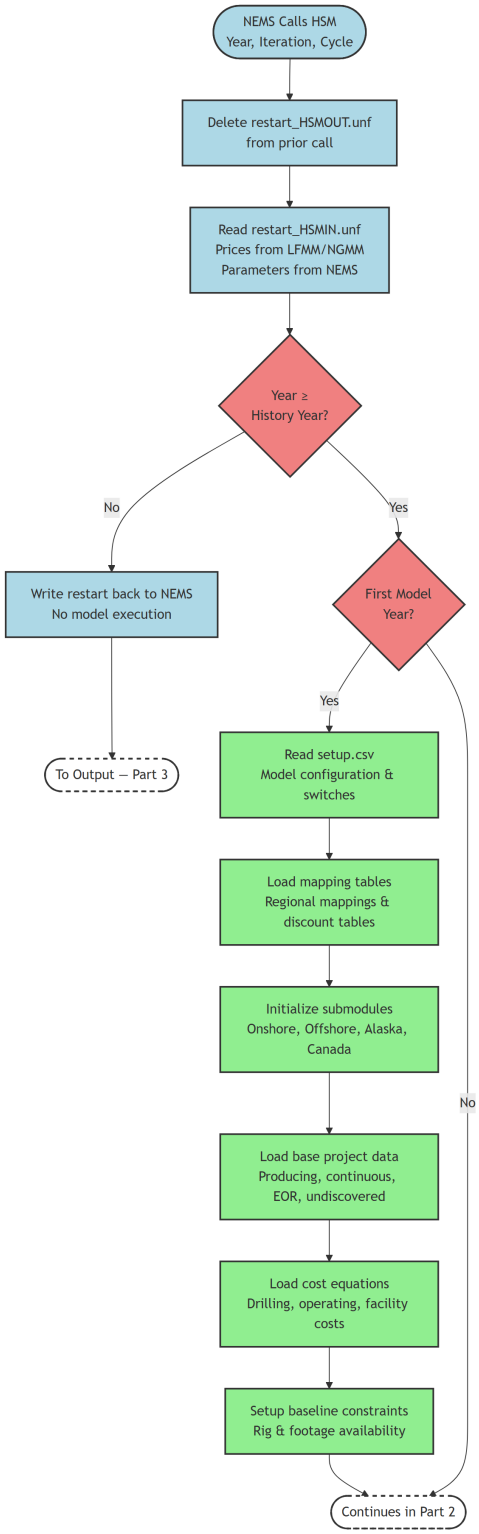
2.2 Model Run Sequence

This page describes the step-by-step execution sequence of *HSM* within a National Energy Modeling System (*NEMS*) run at the U.S. Energy Information Administration (*EIA*). The diagram below is the authoritative process flow. The diagram is described in further detail, step-by-step, below.

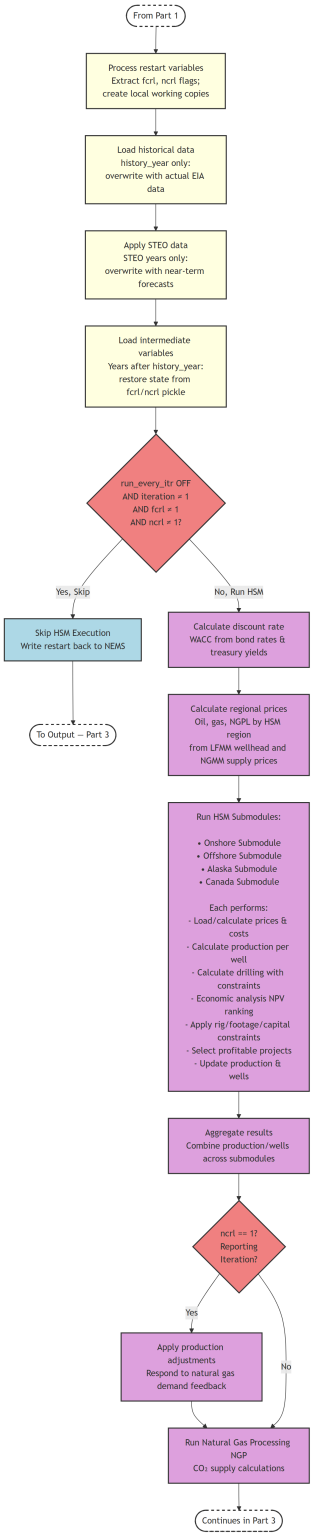
For how NEMS invokes HSM (command-line arguments, restart protocol, module handoffs), see *NEMS Integration*—that page is canonical for external integration; this page focuses on internal phases. Terms used in the diagram and walkthrough (`fcrl`, `ncrl`, STEO, restart files) are defined in *Key Terms* at the end of this page.

2.2.1 Detailed Process Flow

Part 1: Initialization and first-year setup



Part 2: Annual processing and HSM execution



Part 3: Output and NEMS convergence loop

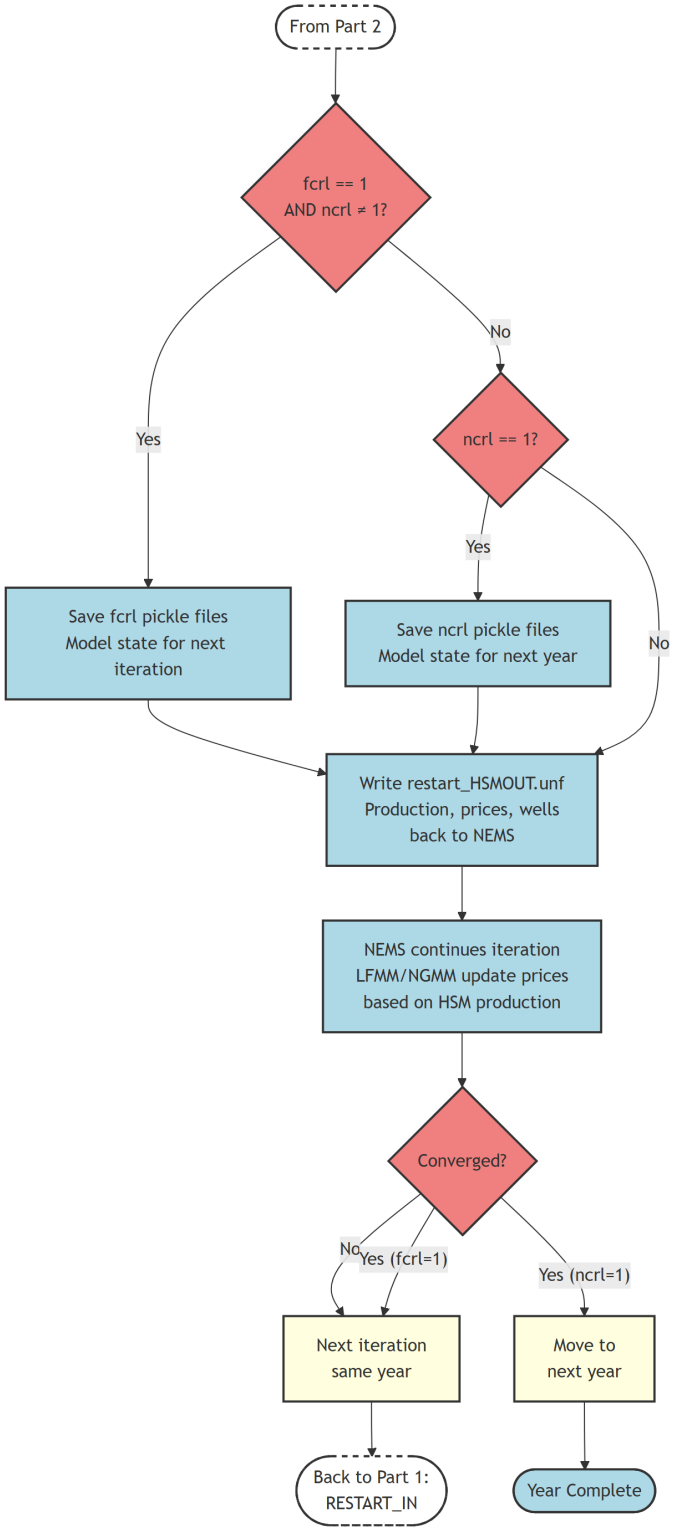
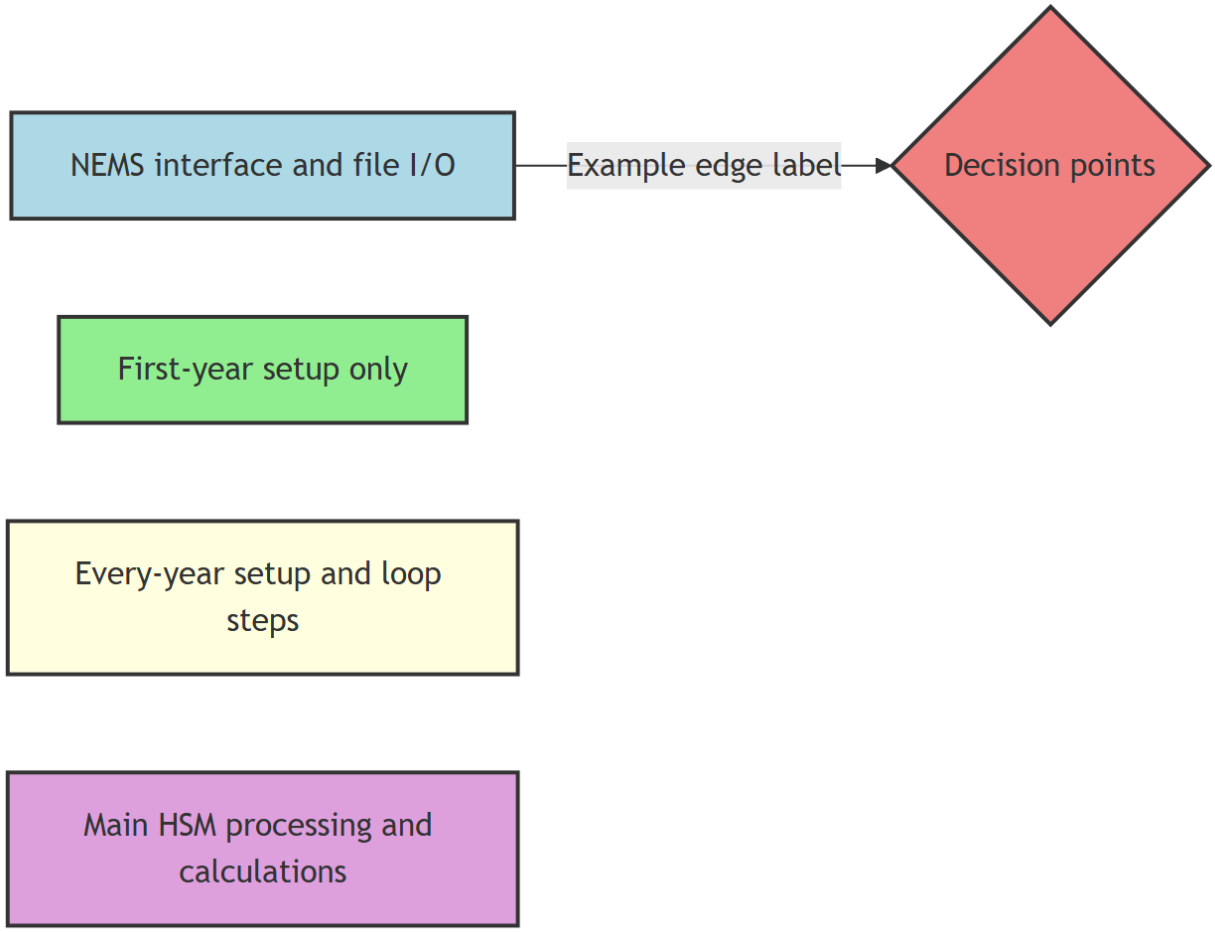


Diagram legend



Edge labels: Branch conditions that determine the next step in the flow.

2.2.2 Phase-by-Phase Walkthrough

1. NEMS Calls HSM

NEMS invokes HSM via `nexec.py`, which calls `run_hsm()` directly; see *NEMS Integration* for the full parameter table. In summary, three key values are passed:

- `-y / --curcalyr` — current calendar year
- `-t / --iteration` — iteration number within the current model year
- `-c / --cycle` — NEMS cycle number (outer loop index for the full NEMS solve; see *Cycle* in *NEMS Integration*)

NEMS begins calling HSM at `FIRSYR` (1990), the first execution year defined in the *SCEDES* file. For years before HSM's `history_year` (2024 for *AEO*2026, defined in `setup.csv`), HSM writes the restart file back to NEMS without executing any model logic. HSM first runs its full economic and drilling calculations at `history_year`.

HSM immediately deletes any existing `restart_HSMOUT.unf` from the prior call. This is intentional; if HSM crashes before writing output, NEMS will detect the missing file and stop cleanly.

2. Read Restart File

HSM reads `restart_HSMIN.unf`, which contains prices and parameters passed from NEMS:

- Crude oil wellhead prices by *LFMM* region (from LFMM)
- Natural gas supply prices from *NGMM* district and regional arrays
- Macroeconomic factors including bond rates and the GDP deflator (from *MAM*)
- CO_2 prices (from *CCATS*)
- Iteration control flags: `fcrl`, `ncrl`, and the current calendar year

3. Year 1 Initialization (first model year only)

On the first model year (determined by checking `history_year` in `setup.csv`), HSM performs a one-time setup:

1. **Read `setup.csv` —** loads all model configuration switches and filenames
2. **Load mapping tables —** district-to-region crosswalks (`mapping.csv`), LFMM price region mappings, and *NGPL* region assignments
3. **Load discount tables —** *WACC* inputs from `discounting.csv`
4. **Initialize submodules —** instantiates Onshore, Offshore, Alaska, Canada, and NGP submodule objects; each loads its own project data and cost equations
5. **Load base project data —** producing wells, continuous plays, *EOR* projects, undiscovered conventional resources
6. **Load cost equations —** drilling cost equations, operating cost tables, facility cost curves
7. **Set up baseline constraints —** rig availability, footage constraints, capital constraint equations

4. Every-Year Setup

HSM performs these steps every year at or after `history_year`. For years before `history_year`, HSM skips directly to writing the restart file back to NEMS (see *5. Skip Decision Logic*).

1. **Process restart variables —** extracts iteration control flags (`fcrl`, `ncrl`) and creates local working copies of restart arrays for use by the submodules. Step 2 reads the binary restart file; this step parses specific fields from it.
2. **Load historical data —** at `history_year` only (2024 for *AEO*2026), overwrites model-computed production and well counts with actual EIA data
3. **Apply STEO data —** for STEO years only (2025–2027 for AEO2026), overwrites results with *Short-Term Energy Outlook* benchmarks
4. **Load intermediate variables —** for years after `history_year`, reads the pickle file saved on the previous iteration (`fcrl_var.pkl` or `ncrl_var.pkl`) to restore model state

5. Skip Decision Logic

For years before `history_year`, HSM writes the restart file back to NEMS without executing any model logic (see step 1).

For years at or after `history_year`, HSM evaluates whether to run its full economic calculations or skip. HSM skips when `run_every_itr_switch` is `FALSE`, the iteration is not 1, `fcrl` $\neq$ `1`, and `ncrl` $\neq$ `1`. When skipping, HSM writes the existing output restart file and returns control to NEMS.

6. Preprocessing

When HSM runs, it first computes two things that are inputs to all submodules:

1. **Discount rate (WACC) —** calculated from bond rates and treasury yields using `discounting.csv` parameters; see *Price and Discount Rate Calculations* for the formula
2. **Regional prices —** LFMM crude wellhead prices are mapped from LFMM regions to HSM districts/regions, while NGMM natural gas supply prices are taken from district (`ogprcng`) and regional (`ogwprng`) arrays and aggregated to HSM regions using `mapping.csv`

7. Submodule Execution

HSM runs the following submodules in sequence: Onshore → Offshore → Alaska → Canada. Each submodule independently:

- Loads current prices and costs
- Calculates production per well using decline curves or production profiles
- Runs discounted cash flow (*DCF*) analysis on candidate projects
- Ranks projects by *NPV* or profitability index
- Applies rig, footage, and capital constraints
- Drills economically viable wells
- Updates production and well count arrays

See *Submodules* for submodule-specific documentation. The iteration context (`fcrl`/`ncrl`) introduces nuances in how submodules report results; see *NEMS Integration* for details on HSM's interaction with NGMM across iterations.

8. Post-Processing and Aggregation

After all submodules run, `module_unf.prepare_results()` aggregates production and well counts across submodules into the NEMS output arrays.

On the `ncrl` (reporting) iteration only, HSM adjusts its drilling schedule to match the realized non-associated natural gas production volumes returned by NGMM. This ensures that HSM's final reported drilling activity is consistent with the market-equilibrated gas volumes that NGMM determined.

9. Natural Gas Processing (NGP)

`module_unf.run_ngp()` runs the CO_2 Capture at Natural Gas Processing Plants submodule. This computes CO_2 supply cost curves and passes CO_2 demand and profitability data to the CCATS module. See *CO_2 Capture at Natural Gas Processing Plants Submodule (NGP)* for details.

10. Intermediate State Output

HSM saves model state to pickle files before writing the restart file:

- **`fcrl` iteration** (`fcrl == 1` and `ncrl ≠ 1`): saves `fcrl_var.pkl` for use on the next regular iteration
- **`ncrl` iteration** (`ncrl == 1`): saves `ncrl_var.pkl` for use at the start of the next model year

11. Write Restart File

`restart_unf.write_results()` serializes all NEMS output arrays to `restart_HSMOUT.unf`. NEMS reads this file and continues its iteration loop.

12. NEMS Iteration and Convergence

NEMS runs LFMM and NGMM with the new HSM production volumes to update prices. If the solution has not converged, NEMS calls HSM again with the updated prices (next iteration, same year). Once converged:

- If `fcrl == 1`, NEMS advances to the reporting iteration (`ncrl == 1`)
- If `ncrl == 1`, NEMS advances to the next model year

2.2.3 Key Terms

***fcrl* (Final Convergence Reporting Loop)**
: Flag set to 1 on the last regular iteration before reporting. HSM saves intermediate state (`fcrl_var.pkl`) for potential use in the next iteration.

***ncrl* (NEMS Convergence Reporting Loop)**
: Flag set to 1 on the reporting iteration. HSM applies natural gas production adjustments based on NGMM demand feedback and saves state (`ncrl_var.pkl`) for the next year.

STEO (*Short-Term Energy Outlook*)
: Near-term production forecasts published by EIA that override HSM model calculations for near-forecast years. For AEO2026, STEO years are 2025–2027.

LFMM (Liquid Fuels Market Module)
: NEMS module that provides crude oil wellhead prices to HSM and receives crude oil and NGPL production volumes from HSM.

NGMM (Natural Gas Market Module)
: NEMS module that provides natural gas supply prices and realized non-associated gas production volumes (ncrl) to HSM, and receives expected non-associated gas and associated-dissolved gas production from HSM.

CCATS (Carbon Capture, Allocation, Transportation, and Sequestration)
: NEMS module that provides CO_2 prices to HSM and receives CO_2 demand and supply cost curves from the NGP submodule.

MAM (Macroeconomic Activity Module)
: NEMS module that provides economic factors — including bond rates and the GDP deflator — used in WACC and price inflation calculations.

Restart File
: Binary file format (`.unf`) used for data exchange between NEMS modules. `restart_HSMIN.unf` carries inputs to HSM; `restart_HSMOUT.unf` carries HSM outputs back to NEMS.

2.2.4 Related Pages

- *NEMS Integration* — Data flows between HSM and other NEMS modules
- *Submodules* — Submodule execution details
- *Price and Discount Rate Calculations* — WACC and price calculation details
- *Running HSM — execution details* — How to run HSM standalone vs. integrated

2.3 NEMS Integration

HSM operates as a supply-side component within the National Energy Modeling System (*NEMS*) at the U.S. Energy Information Administration (*EIA*). This page describes the data flows between HSM and other NEMS modules, the restart-file-based communication protocol, and the iteration control flags that govern when HSM runs.

2.3.1 How NEMS Calls HSM

NEMS calls HSM through `nexec.py`, the model dispatch module that routes execution to each NEMS supply model by index. HSM is IMODEL 9. On each iteration, `nexec.py` imports `hsm.py` and calls `run_hsm()` directly — no subprocess or batch file is involved in integrated runs.

The call passes five parameters:

Parameter	Source	Description
`year`	`pyfiler1.ncntrl.curcalyr`	Current calendar year
`iteration`	`pyfiler1.ncntrl.curitr`	Iteration number within the current model year
`pyfiler1`	NEMS restart interface	Binary restart file read/write handle
`cycle`	`pyfiler1.cycleinfo.curirun`	NEMS cycle number (outer solve index; see *Cycle*)
`scedes`	`user.SCEDES`	Parsed SCEDES scenario configuration

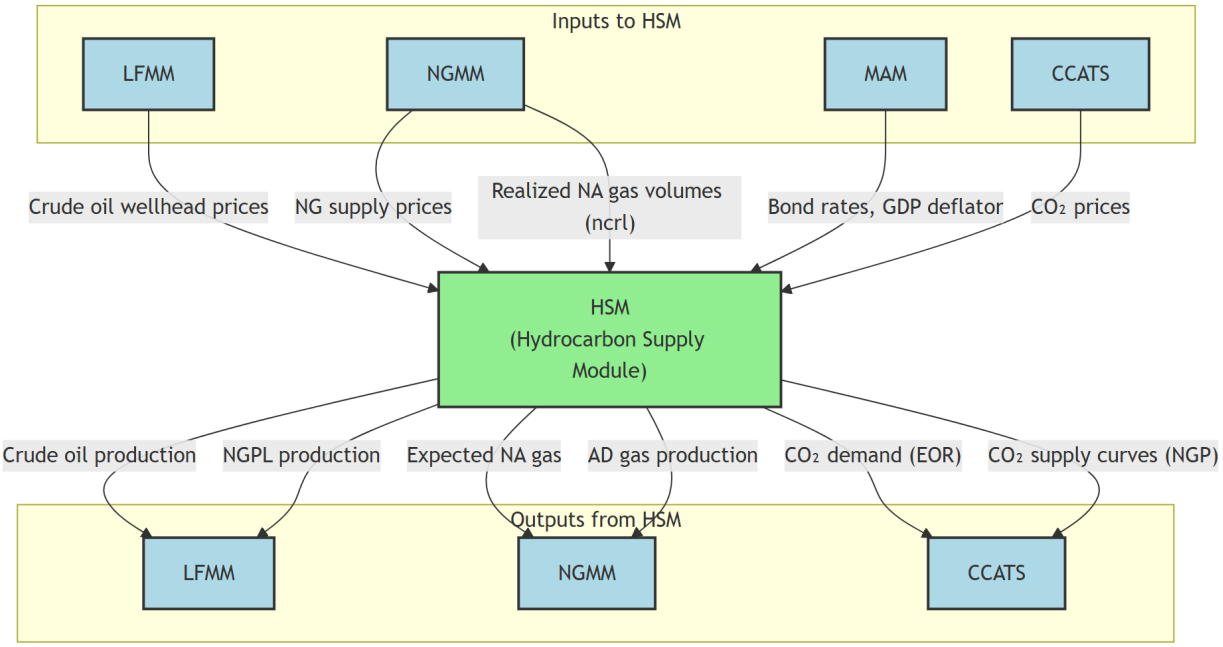
For **standalone (offline) runs**, `hsm.py` can also be invoked from the command line; see *Running HSM — execution details* for that usage.

Cycle

Cycle is NEMS's outer solve index for the full integrated model (distinct from **year**, the calendar projection year, and **iteration**, the convergence pass within that year). HSM receives all three values each call; most HSM logic keys off year and iteration flags (`fcrl`, `ncrl`) rather than cycle directly. For a full description of the NEMS solve structure, see the Integrating Module documentation.

2.3.2 NEMS Module Data Flows

HSM exchanges data with four NEMS modules through binary restart files and the pyfiler interface.



Liquid Fuels Market Module (LFMM)

- ***LFMM*** → **HSM:** Crude oil wellhead prices by LFMM region (`pmmout_dcrdwhp`). These are mapped to HSM districts via `mapping.csv` (`lfmm_region_number` column).
- **HSM → LFMM:** Domestic crude oil production by LFMM region (`pmmout_rfqdcrd`, `pmmout_rfqtdcrd`) and natural gas plant liquids (*NGPL*) production.

Natural Gas Market Module (NGMM)

The HSM–*NGMM* interaction has two phases per year:

1. **Iteration 1 through *fcrl*** — HSM updates **expected** non-associated gas production and associated-dissolved gas production in the restart file on every iteration it runs. NGMM reads these volumes as input to its market-clearing optimization.
2. ***ncrl* iteration** — NGMM returns **realized** non-associated gas demand volumes to HSM. HSM then adjusts its drilling schedule to match those realized volumes.

This two-phase design allows NEMS to achieve market equilibrium between gas supply and demand. Both U.S. (84 districts) and Canadian (2 regions) gas volumes participate in this exchange.

- **NGMM → HSM:** Natural gas supply prices via district-level (`ngtdmrep_ogprcng`) and regional (`ngtdmrep_ogwprng`) arrays. On the ncrl iteration, NGMM also returns realized non-associated gas production volumes (`ogsmout_ogrnagprd`) that HSM uses to adjust its drilling schedule.
- **HSM → NGMM:** Expected non-associated gas production for the U.S. (`ogsmout_ogenagprd`) and Canada (`ogsmout_cnenagprd`), plus associated-dissolved gas production for the U.S. (`ogsmout_ogadgprd`) and Canada (`ogsmout_cnadgprd`). HSM updates these in the restart file on every iteration it runs.

Macroeconomic Activity Module (MAM)

- ***MAM*** → **HSM:** Corporate bond rates and the GDP price deflator (`rest_mc_jpgdp`). The bond rates feed the *WACC* calculation; the deflator converts prices to constant 1987 dollars.

Carbon Capture, Allocation, Transportation, and Sequestration (CCATS)

- *CCATS* → **HSM:** CO_2 prices, used in the economic evaluation of CO_2 *EOR* projects.
- **HSM → CCATS:** CO_2 demand from EOR projects (`ccatsdat_dem_eor`, project-level volumes) and the price EOR projects are willing to pay for CO_2 (`ccatsdat_cst_eor`). The NGP submodule additionally provides CO_2 supply cost curves from natural gas processing plants.

2.3.3 Restart File Protocol

All HSM inputs and outputs are passed through binary restart files (`.unf` format) using the pyfiler interface.

> **Important**
>
> HSM deletes `restart_HSMOUT.unf` at the **start** of every call (`run_hsm()` in `hsm.py`). If HSM crashes before writing output, NEMS will detect the missing file and stop cleanly rather than reading stale data from a prior run.

File	Direction	Contents
`restart_HSMIN.unf`	NEMS → HSM	Prices, economic factors, iteration flags, CO_2 prices
`restart_HSMOUT.unf`	HSM → NEMS	Production volumes, prices, well counts, CO_2 data

For standalone runs, HSM writes restart outputs in the run directory after each call, with `restart_HSMOUT.unf` carrying the production and price arrays consumed by NEMS.

A tabular reference for arrays HSM reads from `restart_HSMIN.unf` but does not write back is on *NEMS restart input variables reference*.

2.3.4 Iteration Control Flags

Two flags in the restart file control HSM's behavior within each model year.

fcrl — Final Convergence Reporting Loop

`fcrl` is set to `1` on the last regular iteration before the reporting iteration. When `fcrl == 1`:

- HSM runs the full model
- Saves intermediate state to `fcrl_var.pkl` for use on the next iteration

HSM updates expected natural gas production in the restart file on every iteration it runs (iteration 1 through fcrl), not only when `fcrl == 1`.

ncrl — NEMS Convergence Reporting Loop

`ncrl` is set to `1` on the reporting iteration. When `ncrl == 1`:

- HSM runs the full model
- Applies natural gas production adjustments based on NGMM's realized demand volumes
- Saves state to `ncrl_var.pkl` for use at the start of the next model year
- Runs the NGP submodule to compute CO_2 supply

Run Decision Logic

HSM evaluates whether to execute or skip based on these conditions:

Condition	Action
Year < `history_year`	Skip — write prior restart file back
`run_every_itr_switch = FALSE` and iteration ≠ 1 and `fcrl ≠ 1` and `ncrl ≠ 1`	Skip
Otherwise	Run full HSM

The `run_every_itr_switch` in `setup.csv` controls whether HSM runs on every iteration or only on iteration 1 / fcrl / ncrl. Setting it to `FALSE` speeds up debugging at the cost of solution accuracy.

2.3.5 HSM and the Natural Gas Market Module (NGMM)

The Natural Gas Market Module (NGMM) represents North American natural gas transmission, distribution, and pricing within NEMS. It is a quadratic program that maximizes consumer plus producer surplus minus transportation costs, subject to mass balance and pipeline capacity constraints. For each month of a given year, NGMM determines the production, flows, and market-clearing prices of natural gas at the state level across the U.S. pipeline network and at the regional level for Canada and Mexico. NGMM replaced the earlier Natural Gas Transmission and Distribution Module (NGTDM) beginning with *AEO*2018.

Why HSM and NGMM Interact

HSM is the **supply side** of the natural gas market in NEMS: it projects how much gas can be profitably produced from wells across the United States and Canada, given prices, costs, and resource availability. NGMM is the **market-clearing side**: it takes that supply information, combines it with demand from all consuming sectors (residential, commercial, industrial, electric power, transportation), and solves for an equilibrium that balances supply and demand at each node in the pipeline network. The two modules iterate within each NEMS year until supply and demand converge within tolerance.

Two-Phase Design (fcrl / ncrl)

The HSM–NGMM iteration operates in two phases each model year:

1. **Iteration 1 through fcrl (convergence) —** HSM runs the full model and provides **expected** non-associated gas production by U.S. district (`ogsmout_ogenagprd`) and by Canadian region (`ogsmout_cnenagprd`), plus associated-dissolved gas production for both the U.S. (`ogsmout_ogadgprd`) and Canada (`ogsmout_cnadgprd`). These represent what HSM calculates can be profitably produced at current prices and costs. NGMM uses these volumes as input to its market-clearing optimization.
2. **ncrl (reporting) iteration —** NGMM has solved its equilibrium and returns **realized** natural gas demand volumes to HSM. HSM then adjusts its drilling activity so that reported production is consistent with market demand. This ensures that final AEO production numbers reflect what the market actually consumes, not just what could theoretically be produced.

During the convergence iterations, NGMM builds its supply curves from the expected production that HSM last wrote to the restart file. Specifically, NGMM uses `ogenagprd` as a reference quantity (`NGSCRV_Q0`) and `ogrnagprd` from the prior year as the current-year anchor (`NGSCRV_Q`) when constructing the price-quantity relationship it optimizes against. Each time HSM updates expected production (in response to NGMM's updated prices), NGMM re-solves its equilibrium with the revised supply — this iteration continues until prices and quantities converge within tolerance.

This two-phase design prevents a disconnect between supply-side profitability and demand-side market clearing — without it, HSM could project production volumes that exceed what pipelines can transport or consumers will purchase at equilibrium prices.

Supply Numbers Exchanged

The following data flow between HSM and NGMM each year:

HSM → NGMM (written every iteration HSM runs):

Variable	Description	Granularity
ogsmout_ogenagprd	Expected non-associated gas production	84 districts × year
ogsmout_ogadgprd	Associated-dissolved gas production	84 districts × year
ogsmout_cnenagprd	Canadian non-associated gas production	2 regions × year
ogsmout_cnadgprd	Canadian associated-dissolved gas production	2 regions × year

NGMM → HSM:

Variable	Description	Granularity
ngtdmrep_ogprcng	Natural gas supply prices by district	84 districts × year
ngtdmrep_ogwprng	Natural gas wellhead prices by region	13 regions × year
ogsmout_ogrnagprd	Realized non-associated gas production (district totals from NGMM market clearing); **written by NGMM on the ncrl iteration only**	84 districts × year

The 84-district detail allows NGMM to model state-level pipeline flows and regional price differentials while HSM operates at the project and county level.

Price Feedback Loop

NGMM returns natural gas supply prices to HSM, which feed directly into the discounted cash flow (*DCF*) analysis that determines project profitability:

- **Higher gas prices** → more gas-directed projects become profitable → HSM reports higher expected production on the next fcrl iteration
- **Lower gas prices** → fewer gas-directed projects pass the hurdle rate → HSM reports lower expected production

This iterative convergence between HSM supply curves and NGMM market clearing is how NEMS achieves supply-demand equilibrium for natural gas. Typically, convergence is reached within two to four NEMS iterations per model year.

Submodule Roles in the Gas Market

Each HSM submodule contributes differently to the NGMM data exchange:

- **Onshore** — the largest gas producer; provides expected and realized non-associated gas from shale, tight, conventional, and coalbed methane plays across seven Lower 48 regions
- **Offshore** — provides Gulf of America and Pacific gas production, primarily associated-dissolved gas from oil-directed fields
- **Alaska** — asymmetric role: Alaska natural gas reporting values are populated from NGMM-linked restart inputs rather than independently projected by HSM, because Alaska's gas market is modeled separately within NGMM
- **Canada** — provides Western Canadian non-associated and associated-dissolved gas available for export to the United States, a key input for NGMM's cross-border flow optimization
- **NGP** — uses NGMM-determined realized production volumes to calibrate how much natural gas flows through processing plants in each Census division

2.3.6 SCEDES File (Standalone Runs)

SCEDES (scenario description) is NEMS's text configuration format for case parameters. For standalone (offline) runs, HSM reads scenario-specific configuration from a SCEDES file instead of receiving it from NEMS. The file is located at:

```
<nems_root>/scedes/<standalone_scedes_file>
```

where `standalone_scedes_file` is set in `setup.csv` (for example, `scedes.cb2026.csv` for the AEO2026 reference case). The `pyscedes.py` module parses this file into a dictionary of key-value pairs.

2.3.7 Key Output Variables

The following table lists the primary output variables written to `restart_HSMOUT.unf`. For the complete list see *Output Variables Reference*.

NEMS Array	Description	Units	Key Dimension
`ogsmout_ogcoprd`	Crude oil production by L48 region	MMbbl/yr	`mnl48t` (11 regions)
`ogsmout_ogngprd`	Natural gas production by L48 region	BCF/yr	`mnl48t`
`ogsmout_ogngplprd`	NGPL production by district	Mbbl/yr	`ogdist` (84 districts)
`ogsmout_ogeorprd`	EOR production by region and type	Mbbl/yr	`mnlnp1`, `eor_types`
`ogsmout_cnenagprd`	Canada NA gas production	MMcf	`numcan` (2 regions)
`ccatsdat_dem_eor`	CO_2 demand from EOR (project-level)	metric tonnes	200 projects
`ccatsdat_cst_eor`	CO_2 price EOR will pay (project-level)	1987$/tonne	200 projects

---

2.3.8 Related Pages

- *Model Run Sequence* — Full run sequence diagram
- *Output Variables Reference* — Complete output variable reference
- *Running HSM — execution details* — Standalone vs. integrated run instructions
- *Price and Discount Rate Calculations* — WACC and regional price calculations
- NEMS Integrating Module documentation (AEO2025) — EIA's description of the NEMS solve structure, iteration, and convergence protocol

2.4 Getting Started

This page covers prerequisites, directory layout, and how to run *HSM* in standalone and integrated modes. For configuration file details, see *Configuration Reference — setup.csv*.

2.4.1 Prerequisites

Python Environment

Package	Minimum version	Purpose	Install
Python	3.10+	Runtime	—
numpy	1.26.2+	Array math	`pip install numpy`
pandas	2.2.3+	DataFrame operations	`pip install pandas`
pyfiler1	NEMS package	Binary restart file I/O	Internal NEMS install

For the full dependency list, see the NEMS requirements file at https://github.com/EIAgov/NEMS/blob/main/requirements.txt.

Note

`pyfiler1` is an internal package for the National Energy Modeling System (*NEMS*) at the U.S. Energy Information Administration (*EIA*) for reading and writing Fortran unformatted binary files (`.unf`). Standalone and integrated model runs require a working `pyfiler1` installation.

System Requirements

- Intel oneAPI Math Kernel Library (MKL) on the `PATH` for Fortran-linked numerical routines
- Sufficient memory for large NumPy arrays: a full *Annual Energy Outlook* (*AEO*) run requires approximately 4–8 GB

2.4.2 Directory Layout

Paths below are relative to the **hsm** package directory (the folder that contains `hsm.py`), for example `.../models/hsm` in the NEMS repository layout.

```
hsm/
├── __init__.py
├── hsm.py                          ← entry point
├── module_unf.py                   ← orchestrator, setup, restart wiring
├── restart_unf.py                  ← restart read and write helpers
├── input/                          ← model inputs
├── debug/                          ← optional debug outputs
├── intermediate_tables/            ← intermediate results
└── ...                             ← supporting model modules
```

In **integrated NEMS** mode the process working directory is the NEMS case folder, not this source tree. That folder holds `restart_HSMIN.unf` and `restart_HSMOUT.unf` next to an **`hsm/`** subdirectory whose `input/`, `debug/`, and `intermediate_tables/` paths mirror the layout above. See *NEMS Integration*.

Quick path guide

Path	Purpose	When you touch it
`hsm.py`	Main execution entry point and argument handling	Run model logic, add top-level flags, or adjust standalone flow
`module_unf.py`	Main run orchestration, setup loading, and restart wiring	Change integrated run flow or setup handling
`restart_unf.py`	Restart file read and write helpers	Update restart variable mapping or binary I/O behavior
`input/`	Input files used by HSM	Update assumptions, calibration values, or scenario inputs
`debug/`	Optional debug outputs generated during runs	Inspect diagnostics and trace run behavior
`intermediate_tables/`	Intermediate output tables generated during runs	Inspect year-by-year intermediate results
`restart_HSMIN.unf` / `restart_HSMOUT.unf`	Integrated-mode restart inputs and outputs in the NEMS case folder	Validate state handoff between NEMS and HSM

For first-time contributors, start with `hsm.py` and `module_unf.py`, and then review `input/` to understand model assumptions.

Source-controlled paths include Python modules and baseline files in `input/`. Runtime-generated paths include most files under `debug/`, `intermediate_tables/`, and restart files in the NEMS case folder.

2.4.3 Running HSM — Standalone Mode

Standalone mode runs HSM for a full projection period (2025–2050 for AEO2026) without NEMS. This is useful for development, testing, and scenario analysis.

```
cd models\hsm
python hsm.py --standalone
```

Or specify a year range:

```
python hsm.py --standalone -y 2025
```

In standalone mode:

- Scenario configuration comes from a ***SCEDES*** (scenario description) file named in `standalone_scedes_file` in `setup.csv`; see *SCEDES File (Standalone Runs)* in *NEMS Integration*
- The loop runs from `history_year + 1` through `DEBUG_STOP_YEAR` (set per AEO cycle in `hsm.py`; 2050 for AEO2026)
- Restart output is written to `restart_HSMOUT.unf` on each call

> **Warning**
>
> `DEBUG_STOP_YEAR` is set per AEO cycle in `hsm.py` (2050 for AEO2026). If you need a different end year for testing, change this constant.

2.4.4 Running HSM — Integrated (NEMS) Mode

In integrated mode, NEMS calls `hsm.py` once per iteration per year:

```
python hsm.py -y <year> -t <iteration> -c <cycle>
```

NEMS handles the year loop and provides restart files. HSM reads `restart_HSMIN.unf` and writes `restart_HSMOUT.unf`. See *NEMS Integration* for details.

2.4.5 Command-Line Arguments

Flag	Long form	Type	Default	Description
`-y`	`--curcalyr`	int	—	Current calendar year
`-t`	`--iteration`	int	1	Iteration number within the current year
`-c`	`--cycle`	int	1	NEMS cycle number
—	`--standalone`	flag	—	Run in standalone mode (year loop inside HSM)

2.4.6 Related Pages

- *Configuration Reference — setup.csv* — Complete `setup.csv` parameter reference
- *NEMS Integration* — Integrated run architecture
- *Running HSM — execution details* — Advanced run and debugging topics

2.5 Configuration Reference — `setup.csv`

`setup.csv` is HSM's primary configuration file. It is located at:

```
<run_directory>/input/setup.csv
```

Every HSM run reads this file on the first model year (`Module.setup()` in `module_unf.py`). The file uses a two-column CSV format: `table` (the key) and `filename` (the value or setting).

Note

In standalone runs, `standalone_scedes_file` points to the SCEDES (scenario description) file for scenario-specific NEMS variable values; see *SCEDES File (Standalone Runs)* in *NEMS Integration*. In integrated NEMS runs, the SCEDES path is provided by NEMS directly and this setting is ignored.

Warning

`run_every_itr_switch = TRUE` causes HSM to run on every NEMS iteration, which significantly increases runtime. Set to `FALSE` for test and debugging runs. The solution quality may differ slightly because HSM will only update on iteration 1, *fcrl — Final Convergence Reporting Loop* (`fcrl=1`), and *ncrl — NEMS Convergence Reporting Loop* (`ncrl=1`).

2.5.1 Parameter Reference

Submodule Setup Files

These keys point to the setup CSV files for each submodule and the historical/STEO data loader. Each setup file is located in the corresponding subdirectory of `input/`.

Key	Default filename	Description
`hist_setup`	`hist_setup.csv`	Setup file for the historical data loader (`history_steo.py`)
`steo_setup`	`steo_setup.csv`	Setup file for STEO data loader
`can_setup`	`can_setup.csv`	Setup file for Canada submodule
`off_setup`	`off_setup.csv`	Setup file for Offshore submodule
`on_setup`	`on_setup.csv`	Setup file for Onshore submodule
`ak_setup`	`ak_setup.csv`	Setup file for Alaska submodule
`ngp_setup`	`ngp_setup.csv`	Setup file for NGP submodule

Pickle Intermediate Variable Files

These keys define the filenames for the pickle files used to pass model state between iterations and between years.

Key	Default file-name	Description
fcrl_intermediate_vars	fcrl_var.pkl	Pickle file saved on the fcrl=1 iteration; restores state on the next iteration of the same year
ncrl_intermediate_vars	ncrl_var.pkl	Pickle file saved on the ncrl=1 (reporting) iteration; restores state at the start of the next model year

Submodule Enable/Disable Switches

Each submodule can be independently enabled or disabled. Disabling a submodule sets its production to zero and skips all economic calculations for that geography.

Key	Type	AEO2026 default	Description
onshore_switch	Boolean	TRUE	Enable/disable Lower 48 Onshore submodule
offshore_switch	Boolean	TRUE	Enable/disable Lower 48 Offshore submodule
alaska_switch	Boolean	TRUE	Enable/disable Alaska submodule
canada_switch	Boolean	TRUE	Enable/disable Canada submodule
ngp_switch	Boolean	TRUE	Enable/disable Natural Gas Processing (CO_2 capture) submodule

Behavioral Switches

Key	Type	AEO2026 default	Description
apply_ch4_emission_penalty_switch	Boolean	FALSE	When TRUE, apply CH4 methane emission cost penalties in cash flow calculations (Onshore, Offshore, Alaska). When FALSE, all CH4 emission costs are zero.
run_every_itr_switch	Boolean	TRUE	When TRUE, HSM runs on every NEMS iteration. When FALSE, HSM only runs on iteration 1, fcrl=1, and ncrl=1. Set FALSE for faster test runs.
side_case_steo_overwrite_switch	Boolean	TRUE	When TRUE, enable side-case STEO overwrites from reference-case preprocessed data for the STEO year. **Set FALSE for reference case runs. Set TRUE for side cases.**
side_case_steo_overwrite_year	Integer	2025	The year for which STEO overwrite data was extracted. Must match the year used when running preprocess_steo_overwrites.py. For AEO2026, this is 2025 (the last STEO-benchmarked year).
debug_switch	Boolean	FALSE	When TRUE, write debug output files (debug method outputs, STEO adjustment factor CSVs to debug/steo_debug/). Increases runtime.
hsm_var_debug_switch	Boolean	FALSE	When TRUE, write CSV copies of intermediate variable tables to intermediate_tables/ (for example, hsm_on_projects.csv, hsm_off_fields.csv). Increases runtime.
charts_switch	Boolean	TRUE	Enable/disable chart generation after standalone runs. Charts are written to the output/ directory.
suppress_warnings_switch	Boolean	FALSE	When TRUE, suppress Python warnings to reduce log clutter. Use prudently — warnings may indicate data quality issues.

Reference Table Files

These keys point to shared reference tables used across multiple submodules.

Key	Default filename	Description
standalone_scedes_file	scedes.cb2026.csv	SCEDES scenario configuration file for standalone (offline) runs, resolved under <nems_root>/scedes/ (not under models/hsm/input/). Format: scedes.<scenario>.<year>.csv.
discounting	discounting.csv	WACC input parameters: debt_ratio, fed_tax_rate, industry_beta, market_risk_premium, return_over_capital_cost.
mapping	mapping.csv	84-district × 13-region crosswalk with state, RRC, LFMM region, PADD, NGPL region, and census division mappings.
state_tax_table	state_tax.csv	State-level income tax rates and severance tax rates (revenue-based and production-based) by state abbreviation.
depreciation_schedules	depreciation_schedules.csv	MACRS depreciation schedule fractions by schedule type (for example, macrs_5yr, macrs_7yr). Used by CashFlow.calculate_depreciation().

Year Constants

These constants define the temporal scope of the model run.

Key	Type	AEO2026 value	Description
`base_year`	Integer	`1990`	Base year for indexed time arrays (year 0 = 1990).
`base_price_year`	Integer	`1987`	Dollar-year for all real price calculations. GDP deflator converts nominal prices to 1987 dollars.
`history_year`	Integer	`2024`	Last year of historical data. For years ≤ `history_year`, model outputs are overwritten with actual U.S. Energy Information Administration (EIA) data.
`technology_year`	Integer	`2025`	Technology improvement switch year. Technology cost improvements from `on_tech_levers.csv` apply for years > `technology_year`.
`aeo_year`	Integer	`2026`	Current AEO cycle year. Used in output labeling and scenario identification.

Output Control

Key	Type	AEO2026 default	Description
`restart_dump_start_year`	Integer	`2020`	Start year for restart file dump filtering. Only years ≥ this value are written to the output restart file. Leave empty to write all years. Setting this reduces output file size for long runs.

2.5.2 Example: Side Case Configuration

For a side case run, the typical changes to `setup.csv` are:

```
side_case_steo_overwrite_switch,TRUE
side_case_steo_overwrite_year,2025
standalone_scedes_file,scedes.mycase2026.csv
```

For a reference case run:

```
side_case_steo_overwrite_switch,FALSE
standalone_scedes_file,scedes.cb2026.csv
```

2.5.3 Related Pages

- *Getting Started* — Prerequisites and run instructions
- *Historical Data and STEO Overrides* — Historical data and STEO override logic
- *NEMS Integration* — SCEDES file role in standalone vs. integrated runs

2.6 Architecture

HSM is organized as a Python package of modules for the National Energy Modeling System (*NEMS*) at the U.S. Energy Information Administration (*EIA*). This page describes the module structure, dependency graph, class hierarchy, and file layout.

Read *Model Run Sequence* first for execution order; this page maps **code structure** to that sequence (`hsm.py` entry point → `module_unf.py` → geographic submodules). For restart and market handoffs, see *NEMS Integration*.

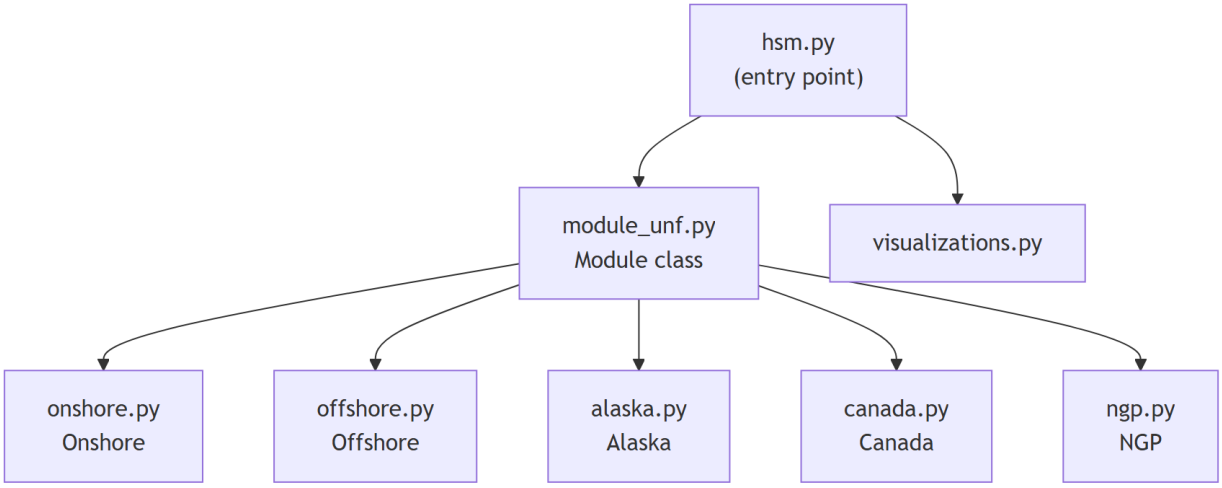
2.6.1 Module Dependency Graph

These diagrams show which modules import or instantiate which other modules. Arrows point from importer to imported. We split the graph into three figures so you can read the run path, orchestration helpers, and submodule-specific imports without crossing many parallel edges on one canvas.

The module inventory table below remains the authoritative list of dependencies for every file.

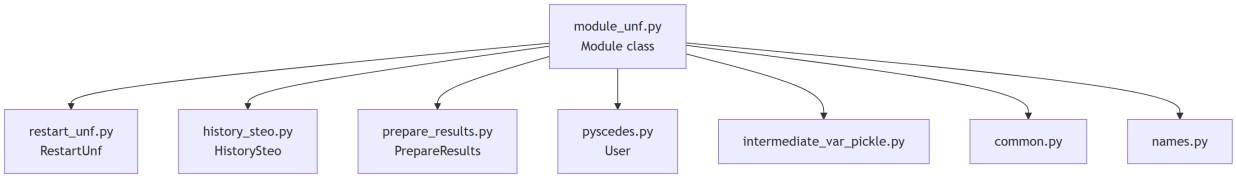
Run path

`hsm.py` is the entry point. It loads `module_unf.py`, which runs the five geographic or functional submodules.



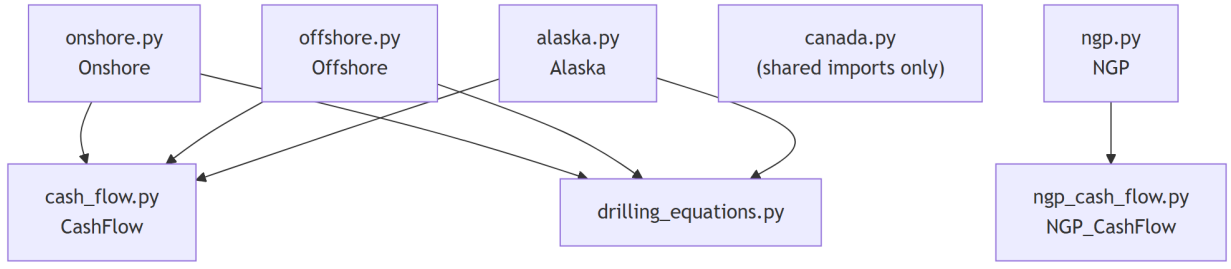
Orchestration and I/O helpers

`module_unf.py` coordinates restart files, historical and STEO data, result preparation, intermediate pickles, SCEDES parsing, and shared column or utility modules.

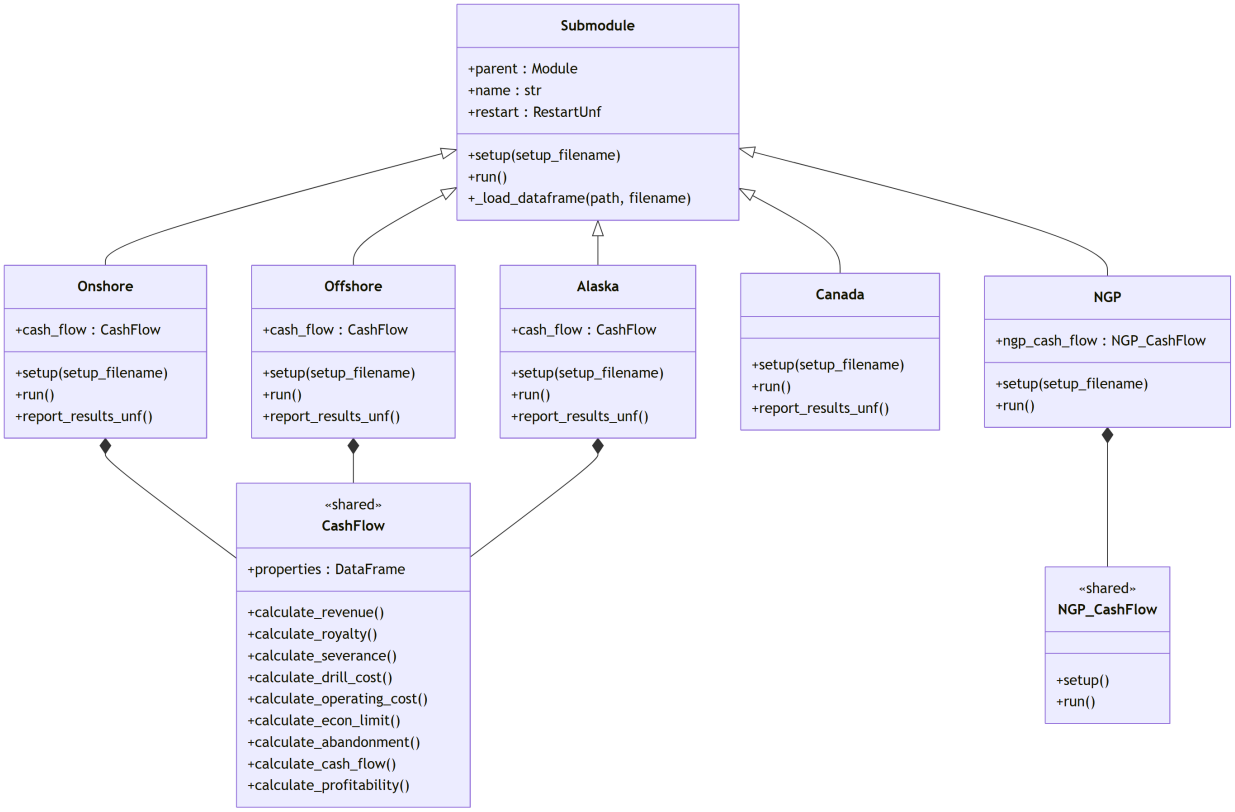


Submodule-specific imports

Onshore, Offshore, Alaska, Canada, and NGP each import `submodule.py`, `common.py`, and `names.py`. The diagram below shows **additional** modules beyond that shared stack. Canada uses a regression-based flow and does not import `cash_flow.py` or `drilling_equations.py`.



2.6.2 Class Hierarchy



`Submodule` is the base class for all five geographic/functional submodules. It provides shared infrastructure: parent module reference, path variables, `setup_table` loading, and the `_load_dataframe` helper. Subclasses override `setup()` and `run()`.

`CashFlow` is a **separate** class, not part of the `Submodule` hierarchy. It is instantiated as a composition member within Onshore, Offshore, and Alaska. Canada does not use `CashFlow` (it uses a regression-based model). NGP uses the separate `NGP_CashFlow` class.

2.6.3 Module Inventory

Module	LOC	Role	Primary class(es)	Key dependencies
hsm.py	310	Entry point; parses args, calls run_hsm	—	module_unf, visualizations
module_unf.py	1,785	Orchestrator; setup, annual run, price calc, aggregation	Module	All submodules, restart_unf, history_steo, prepare_results
submodule.py	108	Base class for all submodules	Submodule	common, names
onshore.py	8,290	Lower 48 onshore supply	Onshore	cash_flow, drilling_equations
offshore.py	3,473	*OCS* field-level supply	Offshore	cash_flow, drilling_equations
alaska.py	1,072	Alaska crude oil supply	Alaska	cash_flow, drilling_equations
canada.py	726	Canadian nat gas export supply	Canada	common, names
ngp.py	1,022	CO_2 capture at gas processing plants	NGP	ngp_cash_flow
cash_flow.py	941	Discounted cash flow engine	CashFlow	common, names
ngp_cash_flow.py	426	*DCF* engine for NGP (processing plant economics)	NGP_CashFlow	common, names
drilling_equations.py	947	Well count, production profiles, decline curves	— (functions)	numpy
restart_unf.py	886	Read/write NEMS binary restart files	RestartUnf	pyfiler1
history_steo.py	1,960	Load historical and STEO data; apply overwrites	HistorySteo	common, names
prepare_results.py	2,449	Aggregate submodule results into NEMS arrays	PrepareResults	common, names
common.py	274	Utility functions (read_dataframe, inflation, etc.)	— (functions)	pandas, numpy
names.py	947	String constants for all DataFrame column/index names	— (constants)	—
intermediate_var_pickle.py	191	Read/write *fcrl*/*ncrl* pickle files	— (functions)	pickle
visualizations.py	1,046	Chart generation for standalone debug output	— (functions)	matplotlib
pyscedes.py	73	Parse *SCEDES* scenario file for standalone runs	User	csv, numpy

2.6.4 Input/Output Directory Layout

```
models/hsm/
├── hsm.py                    ← entry point
├── module_unf.py             ← orchestrator
├── onshore.py, offshore.py, alaska.py, canada.py, ngp.py
├── cash_flow.py, drilling_equations.py
├── restart_unf.py, history_steo.py, prepare_results.py
├── common.py, names.py, submodule.py
├── intermediate_var_pickle.py, visualizations.py
├── pyscedes.py, ngp_cash_flow.py
└── hsm/                      ← run directory (created by NEMS or standalone runner)
    ├── input/                ← all model input files
    │   ├── setup.csv
    │   ├── mapping.csv
    │   ├── history/          ← preprocessed EIA historical data files
    │   ├── steo/             ← STEO forecast files
    │   ├── onshore/          ← onshore project, cost, and constraint files
    │   ├── offshore/         ← offshore field and cost files
    │   ├── alaska/           ← Alaska project and cost files
    │   ├── canada/           ← Canada natural gas files
```

(continues on next page)

(continued from previous page)

```
    │   ├── ngp/              ← NGP facility files
    │   ├── side_case_owrites/← `steo_overwrite_data.npz` (side-case STEO overwrites)
    │   └── variable_mapping/ ← output variable metadata (master_table_selection.csv)
    ├── output/               ← model output files
    ├── debug/                ← debug output files (when debug_switch=TRUE)
    ├── intermediate_tables/  ← intermediate variable CSVs (when hsm_var_debug_
↪switch=TRUE)
    ├── restart_HSMIN.unf     ← input restart file from NEMS
    └── restart_HSMOUT.unf    ← output restart file to NEMS
```

2.6.5 Related Pages

- *API Reference* — Auto-generated API reference for all published modules
- *Input Files Reference* — Complete input file reference
- *Submodules* — Submodule overview and comparison

2.7 Submodules

HSM partitions domestic and Canadian hydrocarbon supply into five geographic and functional submodules for the National Energy Modeling System (*NEMS*) at the U.S. Energy Information Administration (*EIA*). Each submodule has its own project database, cost structure, and economic model, but all share the same NEMS integration infrastructure and (for Onshore, Offshore, and Alaska) the same `CashFlow` *DCF* engine.

2.7.1 How These Pages Relate

Start with the comparison table below to see geography, commodities, and economic approach for each submodule. Read *Model Run Sequence* before diving into district-level detail so you know when each submodule runs in the annual loop. Onshore, Offshore, and Alaska share the *Economic Framework — Discounted Cash Flow Model* DCF engine; Canada and NGP use their own methods documented on their pages. Submodule pages are reference guides—tables and equations—not a substitute for the run sequence on the overview pages.

2.7.2 Submodule Comparison

Submodule	Code file	Geography	Commodities	Economic model
Lower 48 Onshore Submodule	`onshore.py`	L48 onshore, 7 regions, 84 districts	Crude, nat gas, *NGPL*, CO_2 (*EOR*)	DCF (`CashFlow`)
Lower 48 Offshore Submodule	`offshore.py`	*OCS* federal + state, 3 regions	Crude, nat gas, NGPL	DCF (`CashFlow`)
Alaska Crude Oil Submodule	`alaska.py`	Alaska, 3 regions	Crude (primary), assoc. gas	DCF (`CashFlow`)
Canada Natural Gas Submodule	`canada.py`	East/West Canada	Natural gas (exports)	Regression/decline curves
CO_2 Capture at Natural Gas Processing Plants Submodule (NGP)	`ngp.py`	U.S. gas processing plants	CO_2 capture (supply)	DCF (`NGP_CashFlow`)

Lower 48 Onshore Submodule

The Lower 48 Onshore submodule is the most comprehensive component of *HSM*. It projects crude oil and natural gas production from all onshore areas of the contiguous United States (excluding *OCS* offshore areas), organized around 84 districts and 7 supply regions.

Code file: `onshore.py` (~8,291 lines), class `Onshore`

For run order and *NEMS* handoffs, see *Model Run Sequence* and *Submodules*. Economics use the shared *DCF* engine in *Economic Framework — Discounted Cash Flow Model*. Districts are the geographic unit for project-level drilling and production; regions are rollups for reporting and price mapping via `mapping.csv`.

Geographic Scope

The onshore submodule covers districts 1–66 in `mapping.csv`. The 66 state-level districts (including multi-district Texas Railroad Commission regions) are aggregated into 7 HSM supply regions. Offshore state waters (districts 67–75) and federal offshore areas (districts 76–84) belong to the Offshore submodule.

7 HSM Onshore Supply Regions

Region number	Region name	States / areas
1	`east_coast`	ME, NH, VT, MA, RI, CT, NY, NJ, PA, MD, DE, DC, VA, WV, NC, SC, GA, FL, OH, IN, MI, IL, WI, KY, TN
2	`gulf_coast`	AL, LA, TX (RRC 1–3, 6), MS
3	`midcontinent`	AR, IA, MN, MO, NE, OK, TX (RRC 10), KS
4	`southwest`	NM East, TX (RRC 4–5, 7B, 7C, 8, 8A, 9)
5	`rocky_mountain`	AZ, CO, ID, MT (part), NV, NM West, UT, WY
6	`northern_great_plains`	MT, ND, SD
7	`west_coast`	CA, HI, OR, WA

84-District Reference Table

The table below is a **lookup reference** for district names and crosswalk columns (RRC = Railroad Commission of Texas district where applicable; PADD = Petroleum Administration for Defense Districts). For modeling narrative, continue to §Resource Categories.

#	District	State	RRC	HSM Region	*LFMM* Region	PADD
1	Alabama North	AL	0	gulf_coast	padd_3_interior	3
2	Alabama South	AL	1	gulf_coast	padd_3_gulf	3
3	Alaska	AK	0	alaska_onshore_north	padd_5_other	5
4	Arizona	AZ	0	rocky_mountain	padd_5_other	5
5	Arkansas	AR	0	midcontinent	padd_3_interior	3
6	California	CA	0	west_coast	padd_5_california	5
7	Colorado	CO	0	rocky_mountain	padd_4_mountain	4
8	Connecticut	CT	0	east_coast	padd_1_east_coast	1
9	Delaware	DE	0	east_coast	padd_1_east_coast	1
10	Washington DC	DC	0	east_coast	padd_1_east_coast	1
11	Florida	FL	0	gulf_coast	padd_1_east_coast	1
12	Georgia	GA	0	east_coast	padd_1_east_coast	1
13	Hawaii	HI	0	west_coast	padd_5_other	5
14	Idaho	ID	0	rocky_mountain	padd_4_mountain	4
15	Illinois	IL	0	east_coast	padd_2_great_lakes	2
16	Indiana	IN	0	east_coast	padd_2_great_lakes	2
17	Iowa	IA	0	midcontinent	padd_2_interior	2
18	Kansas	KS	0	midcontinent	padd_2_interior	2
19	Kentucky	KY	0	east_coast	padd_2_interior	2
20	Louisiana North	LA	0	gulf_coast	padd_3_interior	3
21	Louisiana South	LA	1	gulf_coast	padd_3_gulf	3
22	Maine	ME	0	east_coast	padd_1_east_coast	1
23	Maryland	MD	0	east_coast	padd_1_east_coast	1

continues on next page

Table 1 – continued from previous page

#	District	State	RRC	HSM Region	*LFMM* Region	PADD
24	Massachusetts	MA	0	east_coast	padd_1_east_coast	1
25	Michigan	MI	0	east_coast	padd_2_great_lakes	2
26	Minnesota	MN	0	midcontinent	padd_2_great_lakes	2
27	Mississippi North	MS	0	gulf_coast	padd_3_interior	3
28	Mississippi South	MS	1	gulf_coast	padd_3_gulf	3
29	Missouri	MO	0	midcontinent	padd_2_interior	2
30	Montana	MT	0	northern_great_plains	padd_4_mountain	4
31	Nebraska	NE	0	midcontinent	padd_2_interior	2
32	Nevada	NV	0	rocky_mountain	padd_5_other	5
33	New Hampshire	NH	0	east_coast	padd_1_east_coast	1
34	New Jersey	NJ	0	east_coast	padd_1_east_coast	1
35	New Mexico East	NM	4	southwest	padd_3_interior	3
36	New Mexico West	NM	5	rocky_mountain	padd_3_interior	3
37	New York	NY	0	east_coast	padd_1_east_coast	1
38	North Carolina	NC	0	east_coast	padd_1_east_coast	1
39	North Dakota	ND	0	northern_great_plains	padd_2_interior	2
40	Ohio	OH	0	east_coast	padd_2_great_lakes	2
41	Oklahoma	OK	0	midcontinent	padd_2_interior	2
42	Oregon	OR	0	west_coast	padd_5_other	5
43	Pennsylvania	PA	0	east_coast	padd_1_east_coast	1
44	Rhode Island	RI	0	east_coast	padd_1_east_coast	1
45	South Carolina	SC	0	east_coast	padd_1_east_coast	1
46	South Dakota	SD	0	northern_great_plains	padd_2_interior	2
47	Tennessee	TN	0	east_coast	padd_2_interior	2
48	Texas RRC 1	TX	1	gulf_coast	padd_3_interior	3
49	Texas RRC 2	TX	2	gulf_coast	padd_3_gulf	3
50	Texas RRC 3	TX	3	gulf_coast	padd_3_gulf	3
51	Texas RRC 4	TX	4	gulf_coast	padd_3_gulf	3
52	Texas RRC 5	TX	5	southwest	padd_3_interior	3
53	Texas RRC 6	TX	6	gulf_coast	padd_3_interior	3
54	Texas RRC 7B	TX	72	southwest	padd_3_interior	3
55	Texas RRC 7C	TX	73	southwest	padd_3_interior	3
56	Texas RRC 8	TX	8	southwest	padd_3_interior	3
57	Texas RRC 8A	TX	81	southwest	padd_3_interior	3
58	Texas RRC 9	TX	9	southwest	padd_3_interior	3
59	Texas RRC 10	TX	10	midcontinent	padd_3_interior	3
60	Utah	UT	0	rocky_mountain	padd_4_mountain	4
61	Vermont	VT	0	east_coast	padd_1_east_coast	1
62	Virginia	VA	0	east_coast	padd_1_east_coast	1
63	Washington	WA	0	west_coast	padd_5_other	5
64	West Virginia	WV	0	east_coast	padd_1_east_coast	1
65	Wisconsin	WI	0	east_coast	padd_2_great_lakes	2
66	Wyoming	WY	0	rocky_mountain	padd_4_mountain	4
67–75	State offshore (OCS)	OCS	—	atlantic / gulf_of_america / pacific	varies	varies
76–84	federal offshore (OCS)	OCS	—	atlantic / gulf_of_america / pacific / alaska	varies	varies

Resource Categories

The onshore submodule tracks four distinct resource categories, each with its own project database, cost structure, and economic model.

1. Producing Wells

Input files: `on_projects_producing_oil.csv`, `on_projects_producing_gas.csv`

These are existing active wells in known fields. Production is modeled using exponential decline curves:

Notation (decline curves)

Symbol	Description	Code / parameter
$q(t)$	Production rate	production array
q_i	Initial rate	`initial_rate`
d_i	Decline rate	`decline_rate`
b	Hyperbolic exponent	`b`
t	Producing-month index (0 at `qi_date`)	month index

See also *Drilling and Production Equations* for onshore well-count symbols (w_y, α_p).

$$q(t) = q_i \, e^{-d_i \, t}$$

Symbol	Description	Code / parameter
$q(t)$	Production rate	production
q_i	Initial rate	`initial_rate`
d_i	Decline rate	`decline_rate`
t	Time since start	month/year index

Decline rates are looked up from play-level or region-level tables:

- `on_producing_oil_decline_rates_plays.csv` (play-level, if available)
- `on_producing_oil_decline_rates_regions.csv` (region fallback)
- `on_producing_gas_decline_rates_play.csv` / `on_producing_gas_decline_rates_regions.csv` (same structure for gas)

Producing wells receive no new drilling investment; their economics are evaluated to determine economic limit (`calculate_econ_limit`) and abandonment timing.

See *Producing and Continuous Project Provenance (High-Level)* for how the `OP1..OP40` and `GP1..GP40` columns in the producing files are built from well-level data produced during preprocessing.

2. Continuous (Unconventional) Resources

Input file: `on_projects_continuous.csv`

Continuous resources include:

- **Tight oil** — low-permeability formations; horizontal drilling
- **Shale oil** — organic-rich source rocks (15 plays tracked in `ogsmout_ogqshloil`)
- **Shale gas** — 15 plays tracked in `ogsmout_ogqshlgas`
- **Tight gas** — low-permeability gas sands
- **Coalbed methane (*CBM*)** — methane desorbed from coal seams

Play boundaries are defined in `shale_gas_play_map.csv` and `tight_oil_play_map.csv`. Production per new well is a function of type curve parameters from `on_projects_continuous.csv`. Wells per year respond to oil and gas prices through price adjustment functions in `drilling_equations.py`.

Technology improvement rates from `on_tech_levers.csv` are applied for years after `technology_year = 2025` (*AEO*2026).

See *Producing and Continuous Project Provenance (High-Level)* for how the new-well type curves in the continuous file are built from well-level data produced during preprocessing.

3. EOR Projects (CO_2 Injection and Steam Injection)

Input files: `on_projects_co2_eor.csv`, `on_co2_pipe_seg_costs.csv`

HSM represents CO_2 EOR only for mature **conventional** oil reservoirs — the fields where CO_2 and steam flooding are the established recovery methods. CO_2 EOR in unconventional (tight and shale) formations is **not** represented in HSM.

Enhanced oil recovery (EOR) projects include:

- **CO_2 injection** — CO_2 injection to increase crude oil recovery from mature conventional reservoirs; CO_2 prices from *CCATS*
- **steam injection** — steam injection to increase crude oil recovery from mature conventional reservoirs

CO_2 EOR projects receive the §43 federal tax credit (`calculate_co2_eor_tax_credit`), subject to a price phaseout threshold (`eor_tc_phaseout`). CO_2 supply costs include pipeline transportation from source to injection site (`on_co2_pipe_seg_costs.csv`).

CO_2 demand from EOR projects is reported to CCATS via `ccatsdat_dem_eor` and `ccatsdat_cst_eor` (the price EOR is willing to pay for CO_2).

Natural gas classification rule: if gas-to-oil ratio > 6,000 cf/bbl, gas is classified as non-associated (NA); otherwise it is associated-dissolved (AD).

4. Undiscovered Conventional Resources

Input files: `on_projects_undiscovered.csv`, `on_discovery_order.csv`

Undiscovered conventional oil and gas resources are modeled using a discovery-order model: resources are evaluated in order of expected size (largest first), consistent with how the industry prioritizes exploration.

GG&LA (geological, geophysical, and lease acquisition) depletion is applied to undiscovered projects only, through `calculate_gg_la_depletion()`. This reflects the exploration costs sunk into finding the resource.

Producing and Continuous Project Provenance (High-Level)

The producing and continuous project CSVs (`on_projects_producing_oil.csv`, `on_projects_producing_gas.csv`, and `on_projects_continuous.csv`) are produced by an onshore preprocessing step before HSM runs. The preprocessor groups individual wells into HSM projects, fits a representative production profile to each project, and writes the per-year `OP1..OP40` and `GP1..GP40` columns that HSM consumes.

Preprocessing inputs

The preprocessor combines two sets of data produced during preprocessing:

- A third-party, well-level database that provides location, true vertical depth, lateral length, oil gravity, completion date, and monthly oil and gas production for individual U.S. onshore wells.
- Pre-fit decline-curve parameters from a prerun *EIA* curve-fitting pipeline. For each well, the pipeline supplies an initial rate (`qi`), a decline rate (`di`), a hyperbolic exponent (`b`), the date of the rate `qi_date` (a proxy for first production), and a model flag selecting one of three functional forms: hyperbolic, harmonic, or exponential.

Wells are also assigned a `GAS` or `OIL` type from the cumulative gas-to-oil ratio (GOR) using a 6 mcf/bbl threshold, and a maidenhead grid identifier from their latitude and longitude.

Project identifier (RESID) construction

Every well is rolled up into a project key called the **RESID** (reservoir ID). Producing-file *RESID*s are constructed by concatenating five components in this order:

- A 2-character prefix: `DO` for an oil project or `DG` for a gas project, taken from the well's GOR-based type.
- The 2-letter state postal abbreviation.
- The 4-digit EIA play code. Play codes are inherited from a legacy labeled well dataset using a k-nearest-neighbor match on a Gower distance over location, true vertical depth (adjusted for surface elevation), completion year, GOR-based well type, and state. The match returns the majority play among the k closest legacy neighbors of each new well.
- The 3-digit county FIPS code from a spatial join of the well point to U.S. Census county polygons.
- The 2-digit process code, assigned by configurable rules (process code by EIA well type, formation, and resource flags).

The continuous-file RESID is derived from the producing RESID by replacing the leading character with `C` and incrementing the process code by 8. The two files therefore use the same project geometry but distinguish wells already producing from new wells that could be drilled in the same project.

Producing-file curve development

For each well in the source database assembled during preprocessing, the preprocessor predicts monthly production from `qi_date` forward to the last year of the HSM projection horizon, using the fitted parameters and the model flag:

$$\text{(hyperbolic)} \quad q(t) = \frac{q_i}{(1 + b\, d_i\, t)^{1/b}}$$

$$\text{(harmonic)} \quad q(t) = \frac{q_i}{1 + d_i\, t}$$

$$\text{(exponential)} \quad q(t) = q_i\, e^{-d_i\, t}$$

Symbol	Description	Code / parameter
$q(t)$	Monthly production rate	predicted production
q_i	Initial rate (`qi`)	`qi`
d_i	Decline rate (`di`)	`di`
b	Hyperbolic exponent	`b`
t	Months from `qi_date`	producing-month index

where t is a normalized producing-month index that starts at 0 in `qi_date` and counts forward in months. Each well's monthly profile is aggregated to producing-year totals, **summed** across all wells in the same RESID, converted from bbl to Mbbl (crude oil) and from mcf to MMcf (natural gas), and pivoted into the `OP1..OP40` and `GP1..GP40` columns of the producing files. `OP1` and `GP1` correspond to the first HSM projection year; later columns are the same project's projected production in each subsequent year.

Continuous-file curve development

Continuous projects describe the productivity of a hypothetical new well that could be drilled in the project today. Construction differs from the producing files in two ways:

- Only wells whose `qi_date` year falls within a configurable recent window are kept (for AEO2026, 2021–2024). This restricts the type curve to wells drilled with current technology and completion practices.
- Each kept well's profile is **shifted** so that production starts on January 1 of the first HSM projection year, regardless of when the actual well came online.

The shifted, monthly profiles are aggregated to producing-year well totals, then collapsed to a representative project profile by taking a configurable quantile across all qualifying wells in the same RESID (the median by default). The result is one representative new-well type curve per project, written into the same `OP1..OP40` and `GP1..GP40` columns of the continuous file.

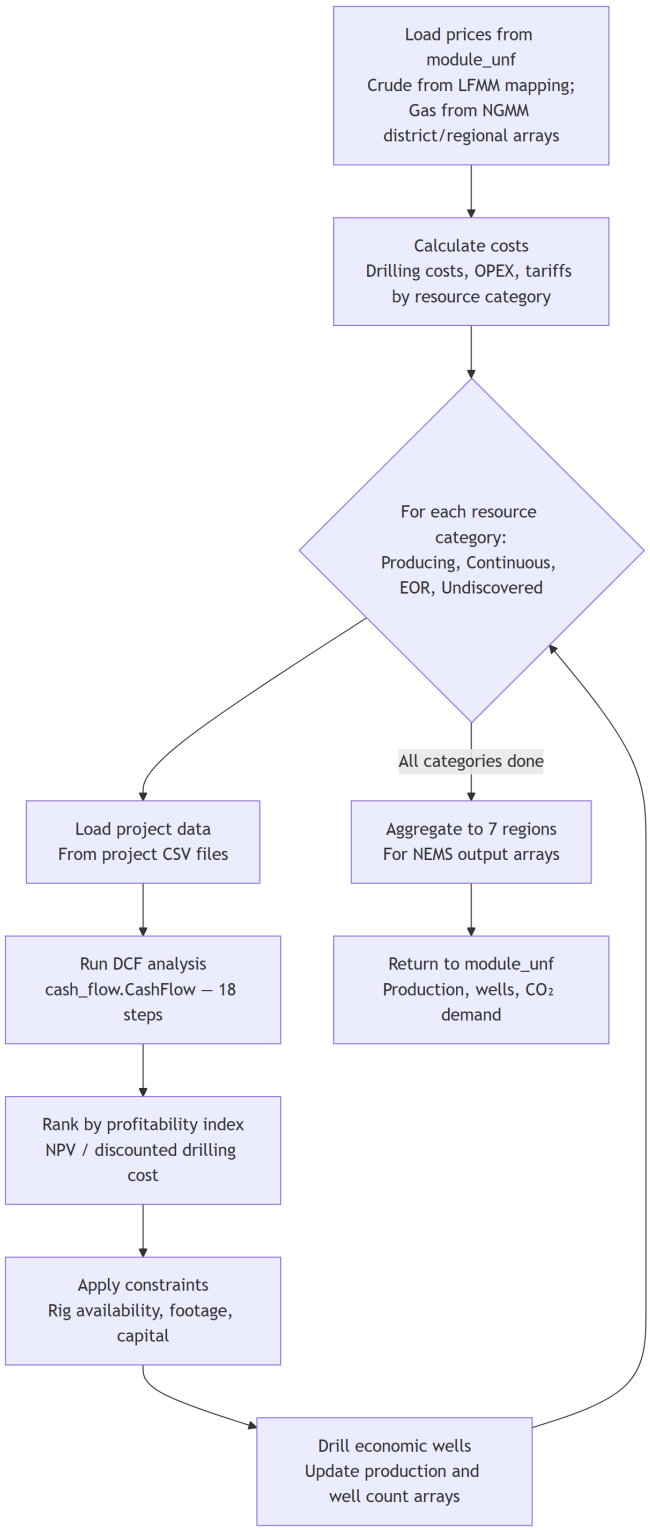
Project selection and attribute roll-up

Projects are filtered and decorated before they are written:

- A producing project is dropped if its first-year production falls below both `OP1 < 0.0001` Mbbl and `GP1 < 0.01` MMcf.
- A continuous project is kept only if at least one of `OP1` or `GP1` is non-negative (continuous wells with no first-year production add no information).
- Project-level attributes — average drill depth, lateral length, and oil API gravity — are computed as means across the wells assigned to the RESID. New-well counts (`Ninjwell`) and historical well counts (`past_wells`) are computed by filtering on the well's completion year and process code.
- Drillable acreage is computed from the maidenhead grid cells covered by the wells assigned to the project, capped by the total area of the underlying counties.
- County FIPS codes are mapped to HSM district number, supply region, and Texas Railroad Commission code through `mapping.csv`, a Texas county-to-RRC mapping, and a southern-state exception map for Alabama, Louisiana, Mississippi, and New Mexico.
- The federal-land share of each county is computed from the U.S. Geological Survey Protected Areas Database and attached to each project so that downstream HSM royalty calculations can apply.

The same prerun pipeline produces both the producing and continuous files in a single run; the only differences between them are the well filters, the time-shift step, and the cross-well aggregation method described above.

Submodule Workflow



Constraint System

The onshore submodule applies three types of supply constraints, all defined by polynomial equations calibrated to historical relationships.

Constraint type	Input file	Description
Rig constraints	`on_rig_constraint_eq.csv`	Maximum rigs available by region and price; polynomial of price
Footage constraints	`on_footage_constraint_eq.csv`	Maximum drilling footage per year
Capital constraints	`on_capital_constraint_eq.csv`	Maximum capital expenditure per year
Wells per rig	`on_wells_per_rig.csv`	Conversion from rig count to well count

Constraints are applied after economic ranking. When a constraint binds, the highest-profitability projects are drilled first until the constraint limit is reached.

Note

The onshore submodule is HSM's largest natural gas producer. For details on how onshore gas production feeds the Natural Gas Market Module (*NGMM*) and how NGMM demand feedback adjusts drilling, see *HSM and the Natural Gas Market Module (NGMM)*.

Key Output Variables

NEMS Array	Description	Units
`ogsmout_ogcoprd`	Crude oil production by L48 region	MMbbl/yr
`ogsmout_ogngprd`	Natural gas production by L48 region	BCF/yr
`ogsmout_ogqshloil`	Crude production from 15 shale oil plays	Mbbl/yr
`ogsmout_ogqshlgas`	Gas production from 15 shale gas plays	TCF
`ogsmout_ogoilprd`	Crude production by district and type	MMbbl/yr
`ogsmout_ogdngprd`	Dry natural gas by district and gas type	BCF/yr
`ogsmout_ogenagprd`	Expected non-assoc gas (updated every iteration; read by *NGMM*)	BCF/yr
`ogsmout_ogrnagprd`	Realized non-assoc gas (after *ncrl*)	BCF/yr
`ogsmout_ogadgprd`	Associated-dissolved gas by district	BCF/yr
`ogsmout_ogngplprd`	*NGPL* production by district	Mbbl/yr
`ogsmout_ogeorprd`	EOR production by region and type	Mbbl/yr
`ogsmout_ogco2inj`	CO_2 injected at EOR projects	MMcf
`ogsmout_ogco245q`	CO_2 purchased (§45Q credit)	MMcf
`ogsmout_ogwells148`	Well counts by region and type	count
`ogsmout_ogsrl48`	Drilling success rate	fraction
`ccatsdat_dem_eor`	CO_2 demand from EOR (project-level)	metric tonnes
`ccatsdat_cst_eor`	CO_2 price EOR willing to pay (project-level)	1987$/tonne

---

Input Files Summary

File	Subdirectory	Description
on_projects_producing_oil.csv	projects/	Existing producing oil wells
on_projects_producing_gas.csv	projects/	Existing producing gas wells
on_projects_continuous.csv	projects/	Unconventional (tight/shale/CBM) plays
on_projects_co2_eor.csv	projects/	CO_2 injection and steam injection EOR projects
on_projects_undiscovered.csv	projects/	Undiscovered conventional resources
on_discovery_order.csv	projects/	Discovery order for undiscovered projects
on_producing_oil_decline_rates_plays.csv	decline_rates/	Play-level oil decline rates
on_producing_oil_decline_rates_regions.csv	decline_rates/	Region-level oil decline rates
on_producing_gas_decline_rates_play.csv	decline_rates/	Play-level gas decline rates
on_producing_gas_decline_rates_regions.csv	decline_rates/	Region-level gas decline rates
on_drill_cost_eqs.csv	costs/	Drilling cost regression coefficients
on_region_avg_cost.csv	costs/	Region-average OPEX, transport, facility capex, and SGA
on_basin_avg_cost.csv	costs/	USGS-province-average cost fallback
on_ngpl_costs.csv	costs/	NGPL processing costs
on_co2_pipe_seg_costs.csv	costs/	CO_2 pipeline segment costs
on_legacy_co2_costs.csv	costs/	Legacy CO_2 cost parameters
on_eor_other_costs.csv	costs/	Other EOR supply costs (steam, polymers, and similar)
on_rig_constraint_eq.csv	constraints/	Rig constraint polynomial equations
on_footage_constraint_eq.csv	constraints/	Footage constraint equations
on_capital_constraint_eq.csv	constraints/	Capital constraint equations
on_tech_levers.csv	configuration/	Technology improvement rates
on_wells_per_rig.csv	configuration/	Wells drilled per rig
shale_gas_play_map.csv	configuration/	Shale gas play-to-district mapping
tight_oil_play_map.csv	configuration/	Tight oil play-to-district mapping

---

Related Pages

- *Economic Framework — Discounted Cash Flow Model* — DCF calculation used for all resource categories
- *Drilling and Production Equations* — Well count and production equations
- *Project Selection and Constraints* — *NPV* ranking and constraint application
- *Input Files Reference* — Full input file reference
- *onshore — Lower 48 Onshore Submodule* — Auto-generated API reference

Lower 48 Offshore Submodule

The Lower 48 Offshore submodule projects crude oil and natural gas production from the U.S. Outer Continental Shelf (*OCS*). It models fields through their complete lifecycle, from undiscovered resources through exploration, development, production, and abandonment.

Code file: `offshore.py` (~3,474 lines), class `Offshore`

See *Submodules* for how Offshore fits among submodules, *Model Run Sequence* for execution order, and *Economic Framework — Discounted Cash Flow Model* for the shared DCF engine.

Geographic Scope

The offshore submodule covers districts 67–84 in `mapping.csv`:

Districts	Area	Type
67–69	North, Mid, South Atlantic	State offshore
70–72	Alabama, Louisiana, Texas Gulf	State offshore
73–74	California, Northern Pacific	State offshore
75	Alaska State Offshore	State offshore
76–78	North, Mid, South Atlantic Federal	Federal OCS
79	Eastern GOA Federal	Federal OCS
80	Central GOA Federal	Federal OCS
81	Western GOA Federal	Federal OCS
82–83	California Federal, Northern Pacific Federal	Federal OCS
84	Alaska Federal Offshore	Federal OCS

The Gulf of America (GOA) — primarily Central and Western GOA — dominates offshore production. Deep-water GOA (>400m) and shallow-water fields have distinct cost and production profile assumptions.

Three-Component Structure

The offshore submodule uses a three-component model where fields move through distinct lifecycle stages.

Component 1: Undiscovered Fields

Key input files:

- `off_undiscovered_fields.csv` — USGS-based undiscovered resource base by area and field size class
- `off_discovery_coefficients.csv` — discovery rate coefficients for exploratory wells
- `off_nfw_coefficients.csv` — new field wildcat (NFW) well model coefficients
- `off_field_size_classes.csv` — field size class distribution probabilities

Undiscovered resources are estimated from USGS petroleum assessment data. Each year, the number of new field wildcat wells drilled (as a function of price) leads to new discoveries. Discovered field size is drawn from the field size class distribution.

Field-size class interpolation uses `common.df_interpolate_2` for smooth transitions between size classes.

Component 2: Discovered but Undeveloped Fields

Key input files:

- `off_announced_fields.csv` — fields announced but not yet in production; include operator, online year, resource size
- `off_delays.csv` — development delay distributions by area
- `off_platform_delays.csv` — platform installation lead times
- `off_platform_slots.csv` — available platform installation slots per year
- `off_platform_drill_per_year.csv` — maximum development wells drilled per platform per year

Once a field is discovered and the *DCF* analysis confirms economic viability, it enters the development queue. Development delays account for engineering, permitting, and platform installation lead times. Platform constraints limit the number of fields that can move to production in any given year.

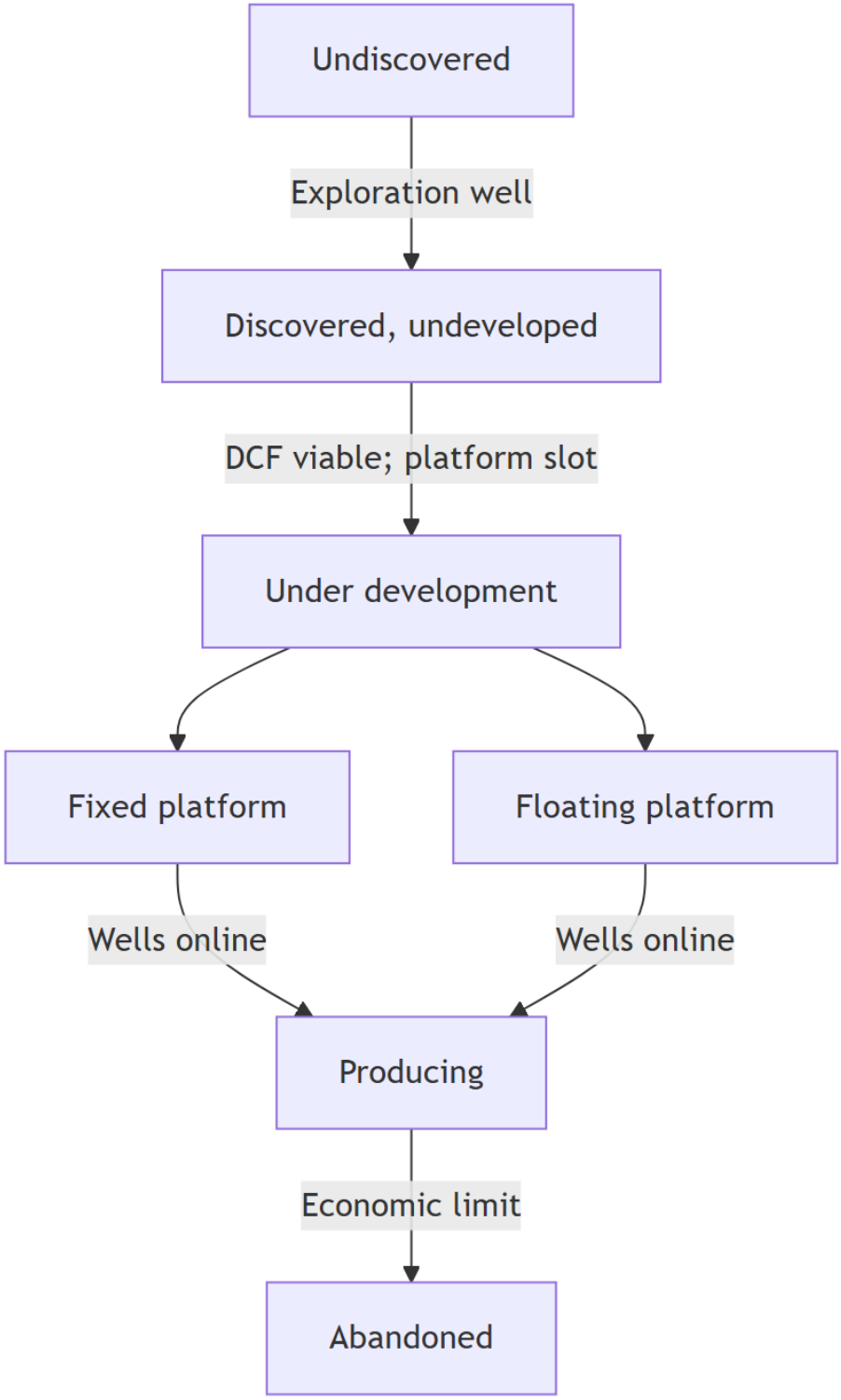
Component 3: Producing Fields

Key input files:

- `off_prod_profile.csv` — production profiles by field size class and water depth
- `off_operating_cost.csv` — OPEX by water depth category
- `off_platform_abandonment.csv` — abandonment cost schedules by platform type
- `off_gom_projections.csv` — GOA-specific near-term production projections
- `off_gom_projections_hogs.csv` / `off_gom_projections_logs.csv` — GOA high/low price projections
- `off_field_availability.csv` — technology improvement factors for field recovery

Producing fields follow type-curve production profiles from `off_prod_profile.csv`. Platform type (fixed vs. floating) is determined by water depth from `off_platform_structure_type.csv`. Operating costs vary by platform type and water depth.

Field Lifecycle



Economics

The offshore submodule uses the same `CashFlow` DCF engine as the Onshore submodule. Key differences in cost structure:

- Operating costs are loaded from `off_operating_cost.csv` (by field category and water depth) rather than computed from per-barrel rates
- Platform abandonment costs from `off_platform_abandonment.csv` are applied at the economic limit
- Exploration and development drilling costs from `off_exp_cost.csv` and `off_dev_cost.csv` differ significantly from onshore (offshore wells cost orders of magnitude more)
- *NGPL* revenues are not computed separately for offshore (NGPL is an associated product at offshore prices)

Note

Offshore natural gas production (primarily associated-dissolved gas from oil-directed fields) feeds the Natural Gas Market Module (*NGMM*). For the broader *HSM*–NGMM data exchange, see *HSM and the Natural Gas Market Module (NGMM)*.

Key Output Variables

NEMS Array	Description	Units
`ogsmout_ogprdoff`	Offshore production by federal/state category and type	Mbbl/yr
`ogsmout_ogcoprdgom`	GOA crude oil production by water depth	Mbbl/day
`ogsmout_ogngprdgom`	GOA natural gas production by water depth	BCF/yr
`ogsmout_ogprdadof`	Offshore associated-dissolved gas	BCF/yr
`ogsmout_ogcoprd`	Crude production (including offshore, in L48 regional format)	MMbbl/yr

---

Related Pages

- *Economic Framework — Discounted Cash Flow Model* — DCF calculation
- *Drilling and Production Equations* — Offshore drilling schedules and production profiles
- *Input Files Reference* — Full offshore input file reference
- *offshore — Lower 48 Offshore Submodule* — Auto-generated API reference

Alaska Crude Oil Submodule

The Alaska submodule projects crude oil production from three Alaskan regions. Unlike the Onshore submodule, Alaska uses **named projects** (known fields) from `ak_projects_known.csv` as the inputs that enter the annual project queue, plus a separate `ak_projects_undiscovered.csv` table that is loaded and matched to field-size metadata at setup. Production through `run()` is driven only by known-project rows queued by production start year, not by play- or district-level aggregates.

Code file: `alaska.py` (~1,073 lines), class `Alaska`

See *Submodules* for submodule comparison, *Model Run Sequence* for run order, and *Economic Framework — Discounted Cash Flow Model* for *DCF* methods used on named projects.

Note

The Alaska submodule models **crude oil only** as its primary output. Natural gas profiles are computed internally for project-level DCF economics, but Alaska does not independently project natural gas supply into gas reporting arrays. Alaska gas reporting values are populated during reporting from *NGMM*-linked restart inputs — Alaska's gas market is modeled separately within NGMM rather than by *HSM*. See *HSM and the Natural Gas Market Module (NGMM)* for the full HSM–NGMM data exchange design. `ogsmout_ogprcoak` is Alaska **crude oil** production by region. Alaska *NGPL* values are estimated from a fixed crude-production proxy relationship in `calculate_ngpls`.

Geographic Scope — Three Regions

Region code	Region name	Key areas
11	`alaska_offshore_north`	Beaufort Sea federal *OCS*, Endicott, North Star, Liberty
12	`alaska_onshore_north`	Prudhoe Bay, Kuparuk, Alpine/Colville, TAPS corridor
13	`alaska_other`	Cook Inlet, South Alaska, Gulf of Alaska state waters

Regional parameters including tariff rates and drilling costs are defined in `ak_mapping.csv`.

Tariff Rates (from `ak_mapping.csv`)

Region	Crude oil tariff	Natural gas tariff
Region 11 (Offshore North)	~$7/bbl	~$3/Mcf
Region 12 (Onshore North)	~$5.50/bbl	~$3/Mcf
Region 13 (Alaska Other)	~$2/bbl	~$3/Mcf

The Trans-Alaska Pipeline System (TAPS) tariff is the primary transportation cost for North Slope crude. Tariff rates are embedded in the economic evaluation and reduce wellhead netback prices.

Two Project Categories

1. Known Projects (`ak_projects_known.csv`)

Named fields with publicly available status and development parameters. The CSV contains:

- Oil resources and gas resources (volumes summed to `mean_resources_boe` for matching to `ak_field_size_classes.csv`)
- Initial production rate and maximum production rate (converted from Mbbl/day to annual volumes during setup)
- Production start year
- One annual historical production column per calendar year from 1990 through the latest historical year (`setup_production` backfills modeled years for legacy producers)
- Decline rate parameters (`oil_decline_rate`, `fc_decline`)
- Regional split (onshore/state offshore/federal offshore percentages)
- Alaska region number (11, 12, or 13)

The table below summarizes the major known projects:

Project	Region	Prod start	Oil resources (MMbbl)	Status notes
Prudhoe Bay (PB+PB/C)	12	1977	2,049	Largest U.S. oil field; in mature decline
Kuparuk	12	1981	717	Producing; satellite fields
Alpine (Colville)	12	2000	443	Producing; includes Nanuq, Fiord, Qannik
Greater Mooses Tooth-1	12	2018	190	Producing; NPRA
Greater Mooses Tooth-2	12	2021	40	Producing
Willow (NPRA)	12	2029	580	Sanctioned; under development
Pikka Early (Nanushuk)	12	2026	400	Under development
Pikka Full (Nanushuk)	12	2026	350	Under development
NUNA	12	2025	75	Near-term
Milne Point	12	1985	387	Producing; modest decline
Duck Island (Endicott)	11	1987	111	state offshore
McIntyre	11	1993	146	Producing
ANWR-1 through ANWR-12	12	2038–2060	1,370–360 each	Policy-sensitive; inclusion depends on scenario assumptions
Cook Inlet (SoAK)	13	—	—	South Alaska aggregate

Projects with `year_production_start = 9999` or `2570` represent speculative or policy-dependent developments that are typically outside the modeled projection window.

2. Undiscovered Projects (`ak_projects_undiscovered.csv`)

Probabilistic undiscovered resource estimates by region. At setup, rows are merged to `ak_field_size_classes.csv` on total oil-plus-gas resources (`mean_resources_boe`) using the same backward as-of logic as known projects. The resulting table is retained on the submodule object after setup; **`run()` still loads annual work only through known projects whose `year_production_start` matches the calendar year**, so undiscovered inputs are prepared but not iterated through `load_projects` or `select_projects` in the current Alaska code path.

Historical Data Provenance (High-Level)

Historical known-project production used in `ak_projects_known.csv` is prepared during preprocessing before HSM execution. The preprocessing step maps source Alaska `field_pool` combinations to HSM identifiers (`resid`, `hsm_field`) and provides the historical annual production series consumed by this submodule.

Cost Structure

Drilling Costs

Regional cost rates `exp_drill_cost_rate`, `dev_drill_cost_rate` in `ak_mapping.csv` multiply the exploratory and developmental well counts from `ak_exp_drill_schedule.csv` and `ak_dev_drill_schedule.csv` in `calculate_drilling`. Costs are multiplied by `(1 - tech_rate) ** (calendar_year - zero_year)`, where `tech_rate` comes from `ak_setup.csv`.

Region	`exp_drill_cost_rate`	`dev_drill_cost_rate`
11 (Offshore North)	$25,000,000	$20,010,000
12 (Onshore North)	$15,300,000	$11,550,000
13 (Alaska Other)	$12,000,000	$7,500,000

Facility Costs

ak_facility_costs.csv maps project oil resource size (oil_resources) to a capital cost rate (kap_cost_rate). The table uses size breakpoints, and the model applies the largest breakpoint less than or equal to each project's oil resources (a backward as-of match in calculate_kap_cost).

This structure creates a piecewise schedule with economies of scale: as project size increases, the assigned kap_cost_rate generally declines. In the current Alaska input table, rates range from about 12.38 for the smallest class to about 7.88 for the largest class.

Notation (facility and operating costs)

Symbol	Description	Code / parameter
K_{cap}	Total facility capital	total_kap_cost
κ	Capital cost rate per resource unit	kap_cost_rate
R_o	Oil resources	oil_resources
C_{ab}	Abandonment cost	abandon_rate
C_{op}	Operating cost	opex
b, m	OPEX intercept and slope	ak_opex.csv coefficients

Production profile symbols (q_{peak}, d_o) are in *Drilling and Production Equations*.

The model computes total facility capital as:

$$K_{\text{cap}} = \kappa \, R_o$$

Symbol	Description	Code / parameter
K_{cap}	Total facility capital	total_kap_cost
κ	Capital cost rate	kap_cost_rate
R_o	Oil resources	oil_resources

and then allocates that total evenly over the first four evaluation years (25% each year). Alaska also sets abandonment cost as 10% of the same total facility capital:

$$C_{\text{ab}} = 0.1 \, \kappa \, R_o$$

Symbol	Description	Code / parameter
C_{ab}	Abandonment cost	abandon_rate
κ	Capital cost rate	kap_cost_rate
R_o	Oil resources	oil_resources

Operating Costs

From ak_opex.csv: coefficients b and m vary by bracket. calculate_operating_cost picks rows index 0–3 using the **peak annual crude** volume in each project's evaluation profile: with peak = series.max() (barrels per year), row **index 0** applies when peak / 100,000 is less than **50** (about **5 million** bbl/year), **index 1** when less than **160** (about **16 million** bbl/year), **index 2** when less than **300** (about **30 million** bbl/year), and **index 3** otherwise.

For each project-year with production:

$$C_{\text{op}} = b + m \, Q$$

Symbol	Description	Code / parameter
C_{op}	Operating cost	`opex`
b	Fixed annual term (intercept)	`b` from `ak_opex.csv`
m	Production-linked slope	`m` from `ak_opex.csv`
Q	Annual production volume	`production`

where b is the fixed annual operating-cost term (intercept) and m is the production-linked slope for the selected row. Operating cost is set to zero in years with zero production and then multiplied by `(1 - tech_rate) ** (calendar_year - zero_year)`.

Production Modeling

Production profiles use Alaska-specific functions in `drilling_equations.py`:

- `ak_production_profile` — peak production as a function of resource size
- `ak_decline_profile` — decline rate by region

During `setup_production`, known projects that already started before `zero_year` with positive output there use `ak_decline_profile` when `year_production_start < zero_year - 4`, and `ak_production_profile` for oil when the start falls within the prior four calendar years through `zero_year`. The combined paths are anchored to observed annual volumes between `param_baseyr` (historical calendar base year from the parent module, usually `1990`) and `zero_year`, then extend in calendar-year columns. Each projection year, `calculate_production` assigns evaluation-window crude and natural gas with `ak_production_profile` for projects queued that year.

A 30-year evaluation horizon is used, with drilling schedule index years 0–29 in `ak_dev_drill_schedule.csv` and `ak_exp_drill_schedule.csv`.

Economics

Alaska uses the same `CashFlow` DCF engine as the Onshore and Offshore submodules; see *Economic Framework — Discounted Cash Flow Model* for depreciation, state tax formulas, and other shared mechanics.

Inputs that differentiate Alaska loading in `load_cash_flow()` and pricing in `load_prices()` include:

- Tariffs in `crude_tariff_price` and `natgas_tariff_price` from `ak_mapping.csv` (see above)
- Crude prices from *LFMM* and natural gas prices from NGMM, averaged over the trailing `averaging_years` calendar years set in `ak_setup.csv`
- Alaska property state code `AK` and severance **rates** `sev_rate_crude` and `sev_rate_natgas` from `ak_setup.csv` (applied before `calculate_state_tax`)
- Federal tax rate from the parent module; **discount rate = parent discount rate + 5 percentage points** for Alaska rows
- Royalty `royalty_rate` and tangible or intangible `exp_tang_frac`, `dev_tang_frac`, `kap_tang_frac`, plus amortization and depreciation schedules, from `ak_setup.csv`
- Facility capital and operating costs derived from Alaska cost tables (`calculate_kap_cost`, `calculate_operating_cost`)

Project selection

`select_projects` keeps rows with `profitability > 0` and `year_production_start <=` the current calendar year (`rest_curcalyr`).

- **Known projects with `resid <= 50`:** rows that satisfy `year_production_start <= rest_curcalyr` receive `profitability = 1` before other selection logic runs, retaining production from announced fields that already started.
- **`resid >= 51` projects (for example ANWR rows):** when more than one such row qualifies in the same iteration, extras are bumped to `rest_curcalyr + 2`. Rows in `ak_projects_known.csv` with `resid >= 51` and a production start on or after the current calendar year are shifted forward another 2 years when that rule fires, pacing development so roughly **one qualifying `resid >= 51` project can enter every two calendar years.**

Key output variables

All restart variables are written only when `rest_curcalyr >= steo_years[0]`.

Crude oil

NEMS array	Description	Units
`ogsmout_ogprcoak`	Alaska crude oil production by region (11–13)	MMbbl/yr
`ogsmout_ogqcrrep`	Crude oil production by oil category (category 4)	MMbbl/yr
`ogsmout_ogoilprd`	Crude oil production by oil type and HSM district (region 12 value from `pmmout_rfqdcrd`)	Mbbl/day
`ogsmout_ogcrdprd`	Crude oil production by HSM region and crude type (regions 11–13, crude type 3)	MMbbl/yr
`ogsmout_ogcruderef`	Crude oil production for LFMM by crude type and region (LFMM region 8)	MMbbl/yr
`pmmout_rfqtdcrd`	Total domestic crude production by HSM region for LFMM, including *EOR* (regions 11–13)	Mbbl/day
`pmmout_rfqdcrd`	Total domestic crude production by HSM region for LFMM, excluding EOR (regions 11–13)	Mbbl/day
`ogsmout_ogprdoff`	Offshore crude oil production — state (index 5) and federal (index 6); gas offshore indices set to zero	MMbbl/yr
`ogsmout_ogcoprd_fed`	Crude oil production on federal land (regions 11, 12)	Mbbl/day
`ogsmout_ogcoprd_nonfed`	Crude oil production on non-federal land (region 13)	Mbbl/day

Natural gas

NEMS array	Description	Units
`ogsmout_ogqngrep`	Alaska natural gas reporting entry populated from NGMM-linked restart values (category 8)	BCF/yr
`ogsmout_ogdngprd`	Dry natural gas production from NGMM restart (districts 3, 75, 84)	BCF/yr
`ogsmout_ogngprd_fed`	Natural gas production on federal land (regions 11, 12)	TCF
`ogsmout_ogngprd_nonfed`	Natural gas production on non-federal land (region 13)	TCF

NGPLs

NEMS array	Description	Units
`ogsmout_ogngplak`	Total Alaska NGPL production	Mbbl/day
`ogsmout_ogngplprd`	Total NGPL production by HSM district (district 3)	Mbbl/day
`ogsmout_ogngplet`	Ethane production by HSM district (hardcoded to zero)	Mbbl/day
`ogsmout_ogngplpr`	Propane production by HSM district	Mbbl/day
`ogsmout_ogngplbu`	Butane production by HSM district	Mbbl/day
`ogsmout_ogngplis`	Isobutane production by HSM district	Mbbl/day
`ogsmout_ogngplpp`	Pentanes-plus production by HSM district	Mbbl/day

Related Pages

- *Economic Framework — Discounted Cash Flow Model* — DCF calculation
- *Drilling and Production Equations* — Alaska production profiles and decline curves
- *Input Files Reference* — Alaska input file listing
- *alaska — Alaska Crude Oil Submodule* — Auto-generated API reference

Canada Natural Gas Submodule

The Canada submodule projects Canadian natural gas production available for export to the United States, disaggregated into East Canada and West Canada. It is calibrated to Canada Energy Regulator (*CER*) projections and responds to Henry Hub price signals.

Code file: `canada.py` (~726 lines), class `Canada`

See *Submodules* for how Canada differs from *DCF* submodules and *NEMS Integration* for Natural Gas Market Module (*NGMM*) export flows.

> **Note**
>
> The Canada submodule does **not** use the `CashFlow` DCF engine. It is a supply-curve/regression-based model in which wells drilled per year are a polynomial function of Henry Hub prices, and production follows hyperbolic decline curves. This approach was chosen to be consistent with CER's own methodology for projecting Canadian gas supply.

Geographic Scope

Region code	Description
`numcan[0]`	West Canada (British Columbia, Alberta, Saskatchewan)
`numcan[1]`	East Canada (Nova Scotia, New Brunswick, Newfoundland, Quebec)

Production Model

Canadian natural gas production is split into two well-type categories:

Non-associated (NA) gas — the primary output; from gas wells not co-produced with oil.

Associated-dissolved (AD) gas — gas co-produced with oil; modeled separately using an elasticity-based relationship from `can_elasticity_ad_gas.csv`.

New vs. Legacy Production

Within NA gas:

- **New production** — from wells drilled in the current year; follow hyperbolic decline
- **Legacy (baseline) production** — pre-existing production declining through the projection period; from `can_natgas_baseline_prod.csv`

The split is approximately 70% new / 30% legacy, though this is calibrated per *AEO* cycle.

Notation

Symbol	Description	Code / parameter
$q(t)$	Production rate (new wells)	production array
q_i	Initial production rate (IPR)	`ipr`
b	Hyperbolic decline exponent	`b`
d_r	Initial decline rate	`dr`
w_y	Wells drilled in year y	`wells_drilled`
P	Henry Hub price	`price`
$c_0, \ldots, c_3$	Polynomial coefficients	`can_natgas_poly_eqs.csv`

Hyperbolic Decline Curve

New well production follows a hyperbolic decline:

$$q(t) = \frac{q_i}{(1 + b\, d_r\, t)^{1/b}}$$

Symbol	Description	Code / parameter
$q(t)$	Production rate	production
q_i	Initial production rate	`ipr`
b	Hyperbolic exponent	`b`
d_r	Initial decline rate	`dr`
t	Time since well start	year index

Parameters are from `can_natgas_dc_vars.csv`, updated per AEO cycle.

Price-Responsive Drilling

Wells drilled per year as a function of Henry Hub price follow a polynomial regression:

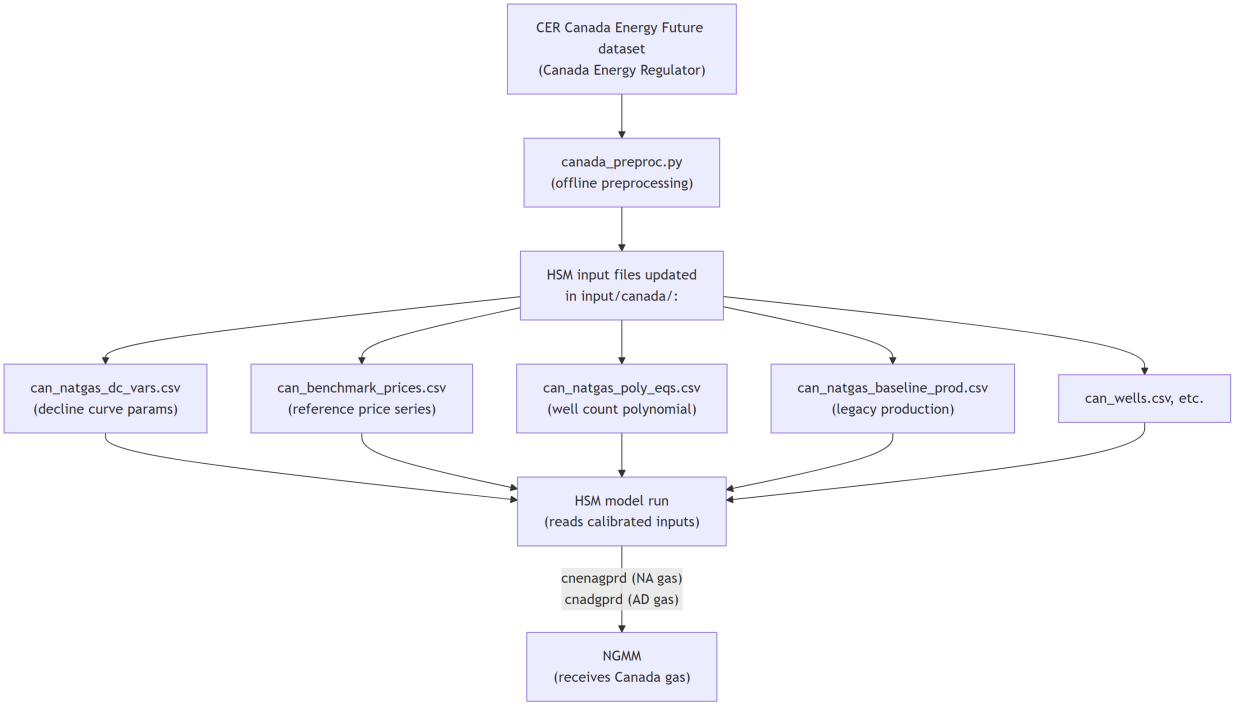
$$w_y = c_0 + c_1\, P + c_2\, P^2 + c_3\, P^3$$

Symbol	Description	Code / parameter
w_y	Wells drilled	`wells_drilled`
P	Henry Hub price	`price`
$c_0, \ldots, c_3$	Regression coefficients	`can_natgas_poly_eqs.csv`

Coefficients are from `can_natgas_poly_eqs.csv`. Benchmark prices from `can_benchmark_prices.csv` anchor the polynomial to CER reference case conditions.

Canada Preprocessing Workflow

The Canada input files are updated each AEO cycle through an **offline preprocessing step** run before the model is built. This is a separate Python script, not part of the *HSM* run itself.



Key inputs to the preprocessor:

- CER Canada Energy Future 2023 regional production scenarios
- Historical well counts and production from U.S. Energy Information Administration (*EIA*) survey data

Key outputs (written to `input/canada/`):

- `can_natgas_dc_vars.csv` — hyperbolic decline curve parameters
- `can_benchmark_prices.csv` — reference Henry Hub price series
- `can_natgas_poly_eqs.csv` — well count regression coefficients
- `can_natgas_baseline_prod.csv` — baseline (legacy) production by region
- `can_natgas_no_export_prod.csv` — Canadian gas not available for U.S. export

NEMS Integration

Canada gas exports to the U.S. are the key output. HSM provides:

- `ogsmout_cnenagprd` — non-associated natural gas production by Canada region (East/West)
- `ogsmout_cnadgprd` — associated-dissolved gas production
- `ogsmout_ogcnpprd` — Canada natural gas supply prices

NGMM receives these volumes and incorporates them into the North American natural gas market balance. Canadian gas is a key input for NGMM's cross-border flow optimization, as Western Canada accounts for most pipeline gas exports to the United States. For the full HSM–NGMM data exchange and iteration design, see *HSM and the Natural Gas Market Module (NGMM)*.

Input Files

File	Updated each AEO?	Description
`can_natgas_dc_vars.csv`	Yes	Hyperbolic decline parameters (`b`, `dr`, `ipr`) by region and well category
`can_benchmark_prices.csv`	Yes	CER-based reference Henry Hub price series for polynomial calibration
`can_natgas_poly_eqs.csv`	Yes	Polynomial regression coefficients for wells drilled vs. price
`can_natgas_baseline_prod.csv`	Yes	Baseline (legacy) production profile by region and year
`can_natgas_no_export_prod.csv`	Yes	Canadian domestic consumption (not available for U.S. export)
`can_wells.csv`	Yes	Historical and projected well count reference by region
`can_elasticity_ad_gas.csv`	No	AD gas elasticity to oil production (stable across AEO cycles)
`canada_setup.csv`	No	Submodule file path configuration

Related Pages

- *Input Files Reference* — Full Canada input file reference
- *NEMS Integration* — How Canada gas feeds NGMM
- *canada — Canada Natural Gas Submodule* — Auto-generated API reference

CO_2 Capture at Natural Gas Processing Plants Submodule (NGP)

Background

Natural gas processing plants separate raw natural gas into pipeline-quality methane and heavier hydrocarbon liquids. During this separation, CO_2 that naturally occurs in the raw gas stream is vented to the atmosphere. Because the CO_2 is already concentrated in the process stream — typically at 2% to 8% by volume — capturing it requires only compression and dehydration rather than the energy-intensive chemical absorption needed at power plants. This makes natural gas processing one of the lowest-cost industrial sources of captured CO_2 in the United States.

The NGP submodule evaluates whether retrofitting individual processing plants with CO_2 capture equipment is economically viable under projected CO_2 prices and federal tax incentives. Facilities that pass an economic screen begin capturing CO_2 and supplying it to the Carbon Capture, Allocation, Transportation, and Sequestration (*CCATS*) module, which clears a market between CO_2 suppliers (including NGP) and CO_2 consumers (primarily enhanced oil recovery projects).

Code file: `ngp.py` (~1,022 lines), class `NGP`; uses `ngp_cash_flow.py` (`NGP_CashFlow`)

How NGP Fits in the CO_2 Market

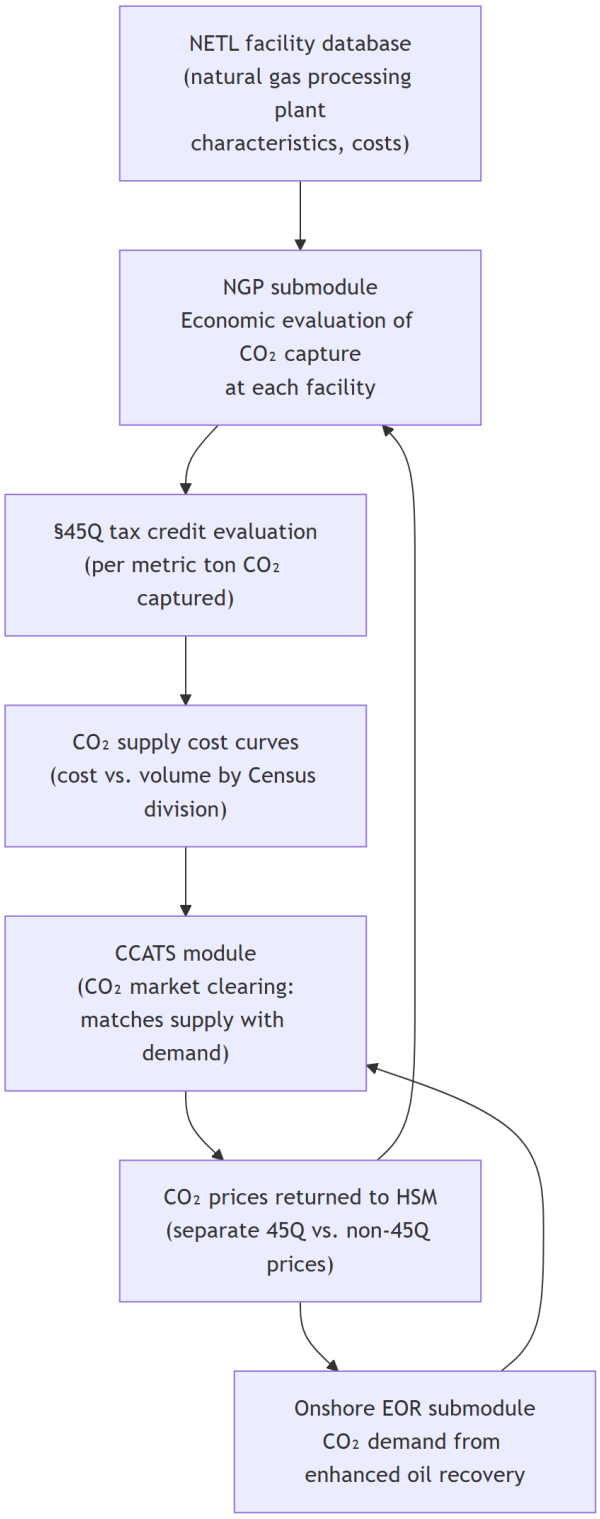
The CO_2 market in *NEMS* has two sides that connect through the CCATS module — not directly within HSM:

- **CO_2 supply** — *HSM*'s NGP submodule calculates how much CO_2 natural gas processing plants can profitably capture and at what cost. These regional supply cost curves are passed to CCATS.
- **CO_2 demand** — HSM's Onshore submodule calculates how much CO_2 enhanced oil recovery (*EOR*) projects need and the maximum price they are willing to pay for it. These demand curves are also passed to CCATS.

CCATS takes both curves, clears the market (finds the equilibrium price and volume for each Census division), and returns CO_2 prices to HSM for the next iteration. HSM then re-evaluates which NGP retrofits are profitable at those prices and which EOR projects can afford to purchase CO_2.

Note

"CO_2 capture supply" means the volume of CO_2 that natural gas processing plants can profitably capture and deliver to a pipeline at a given price. "EOR demand" means the volume of CO_2 that enhanced oil recovery projects are willing to purchase at that price for injection into depleted reservoirs to increase crude oil production.


NETL facility database (natural gas processing plant characteristics, costs)
NGP submodule Economic evaluation of CO_2 capture at each facility
§45Q tax credit evaluation (per metric ton CO_2 captured)
CO_2 supply cost curves (cost vs. volume by Census division)
CCATS module (CO_2 market clearing: matches supply with demand)
CO_2 prices returned to HSM (separate 45Q vs. non-45Q prices)
Onshore EOR submodule CO_2 demand from enhanced oil recovery

Historical Context and Data Source

The NGP facility database originates from the National Energy Technology Laboratory (NETL), which characterized the CO_2 capture potential of U.S. natural gas processing plants. NETL estimated the capital cost of adding compression equipment, the parasitic electricity load of that equipment, and the annual operating and maintenance costs for each plant based on its throughput and CO_2 concentration.

The cost of CO_2 from natural gas processing is significantly lower than from power plants because the gas stream already contains concentrated CO_2. For industrial sources of CO_2, the cost to the producer includes capture, compression to pipeline pressure, and transportation to the injection site via a regional pipeline. These transportation costs are modeled by CCATS rather than within HSM.

As of *AEO*2025, the facility database covers plants across all nine Census divisions. Facilities are classified by:

- **Retrofit eligibility** — whether the plant meets minimum size and CO_2 concentration thresholds for economic capture
- **Existing capture status** — plants already operating CO_2 capture equipment
- **Representative plants** — regional aggregates that represent new capacity growth after STEO years

---

§45Q Tax Credit

The Section 45Q sequestration tax credit, as amended by the Inflation Reduction Act of 2022 and subsequently modified by the One Big Beautiful Bill Act, provides a financial incentive for industrial facilities to capture and sequester CO_2. The credit is available for the first 12 years of operation for plants that begin construction before January 1, 2033.

Key provisions modeled by NGP:

- The tax credit is $85.00 per metric ton (2022 dollars) for both CO_2 used for EOR and CO_2 permanently stored in saline aquifers, rising with inflation after 2026
- Facilities lose 45Q eligibility after 12 years of operation (`evaluation_years`)
- After `final_45q_eligibility_year`, no new facilities qualify for the credit

The NGP submodule tracks 45Q eligibility separately from non-tax-credit (NTC) CO_2 and reports supply volumes in two streams — one for 45Q-eligible capture and one for NTC — because the effective supply cost differs by the value of the credit.

Model Architecture

Facility Types

The NGP submodule manages three categories of facilities:

1. **Operational facilities** (`facilities_df`) — real plants from the NETL database, each evaluated individually for retrofit economics
2. **Legacy facilities** (`legacy_facilities_df`) — representative regional plants that absorb "excess" natural gas throughput during STEO years (when production data are still being reconciled with historical values)
3. **Representative facilities** (`rep_facilities_df`) — regional templates used to create new plant entries after STEO years, when natural gas production growth in a Census division exceeds existing facility capacity

Flow Volume Reconciliation

Each year, the submodule compares realized natural gas production (from the NEMS restart file, variable `ogrnagprd`) against the combined throughput capacity of facilities in the database. The difference represents new processing capacity that must be assigned to either legacy or representative plants, depending on whether the model year falls within or after the STEO horizon.

Economic Evaluation (`NGP_CashFlow`)

The NGP submodule uses a specialized cash flow class (`NGP_CashFlow`) rather than the standard `CashFlow` used by Onshore, Offshore, and Alaska. The economics of retrofitting a processing plant differ from drilling a well:

Aspect	Standard `CashFlow` (wells)	`NGP_CashFlow` (plants)
Revenue source	Oil and gas production	CO_2 capture and sale
Capital costs	Well drilling and completion	Compressor and pipeline equipment
Operating costs	Lifting, gathering, transportation	Compression electricity + maintenance
Tax incentive	§43 EOR credit (price-phased)	§45Q credit (60–85/metric ton)
Key output metric	Net present value	Cost per metric tonne CO_2 + *NPV*

The cash flow evaluates each facility over a 12-year horizon (matching 45Q eligibility) at the discount rate used by the broader HSM model. Facilities with positive NPV are classified as profitable and moved to active capture status.

Technology Improvement

The model assumes that capture technology improves over time, reducing both investment costs and operating costs by 2% each year (`tech_rate_inv`, `tech_rate_om`). This represents advances in compressor design, process integration, and construction techniques that lower the cost of new retrofits over the projection period.

Retirement and Growth Constraints

- Facilities older than 60 years retire from the model
- Legacy facilities (representing aggregate production not tied to specific known plants) retire at a steady rate of 2% per year to reflect natural attrition
- To prevent unrealistic growth surges, the model caps annual CO_2 capture expansion in each Census division to twice the greater of 1,000,000 metric tons or the previous year's capture volume. This constraint reflects real-world limits on labor, equipment supply chains, and project development pipelines.

Electricity Demand

CO_2 compression is energy-intensive. The parasitic electrical load for each facility (in MW) is provided by the NETL database. The NGP submodule:

1. Converts the rated MW load to annual MWh using capacity factor assumptions (63% post-retrofit capacity factor) and hours-per-year conversion
2. Loads industrial electricity prices from the NEMS restart file (`pelin`) by Census division
3. Includes this electricity cost as an operating expense in the cash flow evaluation
4. Reports total electricity demand from active capture facilities back to the NEMS electricity module (via `qmore_qngpin`, converted from MWh to trillion Btu)

This feedback loop means that large-scale CO_2 capture deployment increases regional electricity demand, which can in turn raise electricity prices and affect the economics of marginal retrofit decisions.

When NGP Runs

The NGP submodule executes within `module_unf.run_ngp()` on the **reporting iteration only** (`ncrl == 1`), after the main submodules (Onshore, Offshore, Alaska, Canada) have completed. This sequencing matters because:

- Onshore must finish computing EOR CO_2 demand before NGP reports supply
- CCATS needs both supply and demand curves from the same iteration to clear the market
- CO_2 prices returned by CCATS on the next iteration then update both NGP supply decisions and EOR demand decisions simultaneously

---

Geographic Scope

NGP operates at the **Census division** level (nine divisions covering the Lower 48 states). Alaska is excluded because natural gas processing facilities there do not currently meet the economic thresholds for CO_2 capture retrofitting. Offshore facilities are excluded because they are not natural gas processing plants.

> **Note**
>
> Unlike the Canada submodule — which models Canadian natural gas supply because Canada is a significant exporter to the U.S. market — NGP does not model Mexican facilities. Mexico's natural gas processing infrastructure operates under a different regulatory and economic framework, Mexico is not a significant source of captured CO_2 for U.S. EOR projects, and cross-border CO_2 pipeline infrastructure does not exist at the scale needed to justify modeling.

Key Output Variables

NEMS Array	Direction	Description	Units
`ccatsdat_sup_ngp_45q`	HSM → CCATS	CO_2 supply from NGP (45Q-eligible facilities)	metric tons
`ccatsdat_sup_ngp_ntc`	HSM → CCATS	CO_2 supply from NGP (no tax credit)	metric tons
`ccatsdat_cst_ngp_inv`	HSM → CCATS	Average capital cost of CO_2 capture (by Census div.)	$/metric ton
`ccatsdat_cst_ngp_om`	HSM → CCATS	Average O&M cost of CO_2 capture (by Census div.)	$/metric ton
`ogsmout_ngpco2em`	HSM → NEMS	CO_2 emitted by plants where capture is unprofitable	metric tons
`qmore_qngpin`	HSM → EMM	Electricity demand from CO_2 compression	trillion Btu
`ccatsdat_co2_prc_dis_45q`	CCATS → HSM	CO_2 price for 45Q-eligible retrofit (by Census div.)	$/metric ton
`ccatsdat_co2_prc_dis_ntc`	CCATS → HSM	CO_2 price for non-tax-credit CO_2 (by Census div.)	$/metric ton

---

Input Files

File	Description
ngp_facility_input.csv	Natural gas processing facility database (NETL-based): location, throughput, CO_2 concentration, retrofit eligibility, existing capture status
netl_natural_gas_calculations.csv	CO_2 capture cost calculations by facility: compressor capital cost, O&M, parasitic electrical load
netl_purchased_power_cost.csv	Base-year electricity cost by state for CO_2 compression operations

---

Key Parameters

Parameter	Value	Description
evaluation_years	12	Cash flow evaluation horizon (matches 45Q eligibility window)
ng_thruput_capacity_factor	61%	Natural gas throughput capacity factor (from NETL)
electric_retrofit_cap	63%	Electricity capacity factor for compression equipment
ngp_new_plant_capacity_factor	70%	Capacity factor for new representative plants
ngpp_retire_rate	2%/year	Annual facility retirement rate
tech_rate_inv	2%/year	Annual investment cost technology improvement rate
tech_rate_om	2%/year	Annual O&M cost technology improvement rate
max_growth	2×	Maximum annual CO_2 capture growth (2× previous year or 1 million metric tons, whichever is larger)
census_divisions	9	Number of Census divisions modeled

Interaction with NGMM

The Natural Gas Market Module (*NGMM*) in NEMS determines natural gas production levels and prices. HSM uses these realized production volumes (from the restart file) to calibrate how much natural gas flows through processing plants in each Census division. When NGMM projects production growth in a region, the NGP submodule creates new representative plant entries to absorb the additional throughput. This means that natural gas supply growth directly expands the pool of facilities available for CO_2 capture retrofit evaluation.

Related Pages

- *Economic Framework — Discounted Cash Flow Model* — `NGP_CashFlow` vs. `CashFlow` distinction
- *Lower 48 Onshore Submodule* — EOR projects that demand CO_2 (the demand side of the CO_2 market)
- *NEMS Integration* — CCATS data flow between HSM and other NEMS modules
- *ngp — CO_2 Capture at Natural Gas Processing Plants* — Auto-generated API reference

2.8 Methodology

This section documents the cross-cutting methodologies used across *HSM* submodules in the National Energy Modeling System (*NEMS*) at the U.S. Energy Information Administration (*EIA*). Submodule-specific methods (for example, Alaska cost equations, offshore platform types) are documented in the respective *Submodules* pages.

2.8.1 Mathematical notation

Every page in this section follows the same reader pattern used in *Economic Framework — Discounted Cash Flow Model*:

- **Notation** section at the top of each page — collapsible symbol groups (`hsm-notation-group`) map math symbols to `CashFlow` / code parameter names.
- **Display equations** use standard symbols (NPV, P_c, w_y, …), not snake_case inside `{math}` blocks.
- **Symbols for this equation** panels (`hsm-step-symbols`) sit directly under each display equation for quick lookup while reading a step.

Shared symbols across pages (PI, C_K, r, $q(t)$, …) are defined once per topic and reused.

Page	Notation entry point
Economic Framework — Discounted Cash Flow Model	Full *DCF* legend (18 steps)
Price and Discount Rate Calculations	*WACC*, prices, 1987-dollar deflator
Project Selection and Constraints	PI, rig/footage/capex limits
Drilling and Production Equations	Well counts, decline curves, $q(t)$

Submodule pages with equations (*Lower 48 Onshore Submodule*, *Alaska Crude Oil Submodule*, *Canada Natural Gas Submodule*) use the same panels on a smaller scale. API reference pages link here for math; they do not repeat display equations.

2.8.2 Contents

Economic Framework — Discounted Cash Flow Model

HSM uses a project-level discounted cash flow (*DCF*) model to evaluate whether individual oil and gas drilling opportunities are economically viable. Every candidate project is assessed against current price expectations, cost assumptions, and a discount rate (*WACC*), and only projects with positive *NPV* (or, more precisely, positive profitability index) are drilled.

The DCF engine is implemented in `cash_flow.py` (`CashFlow` class) and is shared by the Onshore, Offshore, and Alaska submodules. Canada uses a regression-based model instead; NGP uses the separate `NGP_CashFlow` class.

> **Note**
>
> `CashFlow` uses class-level shared state — `state_tax_lookup`, `depreciation_schedule`, and `class_init_bool`. The `CashFlow.setup()` classmethod is idempotent: it loads data only on the first call across all submodule instances. Subsequent calls are no-ops. This means all submodules share the same tax tables and depreciation schedules loaded from `state_tax.csv` and `depreciation_schedules.csv`.

Conceptual Overview

Investment in hydrocarbon activities is driven by expected profitability. For each candidate project, HSM calculates the stream of future revenues and costs, discounts them at the project's WACC, and computes net present value. Projects are then ranked by profitability index (NPV divided by discounted drilling cost). The highest-ranked projects are drilled until rig, footage, or capital constraints bind.

The discount rate reflects the industry's weighted average cost of capital (WACC), computed from bond rates and macroeconomic parameters passed from *NEMS*'s Macroeconomic Activity Module. See *Price and Discount Rate Calculations* for the WACC formula.

All dollar flows in the equations below are in real **1987 dollars**, consistent with HSM price and cost inputs. NEMS supplies nominal values where needed; HSM converts them using the GDP deflator. See *Price and Discount Rate Calculations* for the inflation adjustment.

Notation

Equations below use mathematical symbols. Expand each group to map symbols to the corresponding `CashFlow` attribute or input parameter name in code. Symbols in the first column are typeset the same way as in the step equations (standard math subscripts).

Symbol conventions

Some letters appear with more than one meaning. Use subscripts and context to tell them apart:

Letter	Meaning	Where used
K	Capital cost basis (`kap`) in the tangible/intangible split	Step 8
K_{EOR}, K_f, K_i	*EOR* tax credit, federal credit, investment credit	Steps 13, 16
C_K	Present value of drilling costs (profitability denominator)	Step 18
T_c, T_g	Crude and gas transport cost dollars	Step 7
T_s, T_f	State and federal income tax	Steps 15–17
T, t	Projection horizon and year index in NPV sums	Steps 17–18
t_c, t_g	Transport tariff rates (not the year index t)	Step 7
R, R_c, R_g	Revenue	Steps 1–3, 11, 15–17
R_r	Remaining resources (depletion denominator)	Step 10
C_{dd}	Development dry component of drilling cost	Step 4
C_{DH}	Dry-hole cost in the state tax base	Step 15
T	Last year index in NPV sums ($0 \ldots T$); horizon length is `evaluation_years` (typically 30)	Steps 17–18

Symbol	Description	Code / parameter
Q_c, Q_g	Crude and natural gas production	`crude_production`, `natgas_production`
P_c, P_g	Crude and natural gas price	`crude_price`, `natgas_price`
τ_c, τ_g	Crude and natural gas tariff	`crude_tariff_price`, `natgas_tariff_price`
R_c, R_g, R	Crude revenue, gas revenue, total revenue	`crude_revenue`, `natgas_revenue`, `revenue`
ρ	Royalty rate	`royalty_rate`
F_c, F_g, F	Crude royalty, gas royalty, total royalty	`crude_royalty`, `natgas_royalty`, `royalty`

Symbol	Description	Code / parameter
S	Severance tax	`severance`
$s_{c,\text{rev}}$, $s_{g,\text{rev}}$	Severance rates on revenue (crude, gas)	`sev_rate_crude_rev`, `sev_rate_natgas_rev`
$s_{c,\text{prod}}$, $s_{g,\text{prod}}$	Severance rates on production (crude, gas)	`sev_rate_crude_prod`, `sev_rate_natgas_prod`
C_d	Total drilling cost	`drill_cost`
C_e, C_{ed}, C_{dv}, C_{dd}	Exploratory drill/dry, development drill/dry	`exp_drill_cost`, `exp_dry_cost`, `dev_drill_cost`, `dev_dry_cost`
c_n, C_n	*NGPL* net unit cost, NGPL operating cost	`ngpl_net_rate`, `ngpl_operating_cost`
c_N, P_n, V_n, κ, d_n	NGPL cost, price, volume, barrels/gal, divisor	`ngpl_cost`, `ngpl_price`, `ngpl_volume`, `barrels_per_gallon`, `ngpl_volume_divisor`

Symbol	Description	Code / parameter
C_o	Total operating cost	`operating_cost`
C_{oc}, C_{og}	Crude and gas operating cost	`crude_opex`, `natgas_opex`
c_o	Production opex per barrel	`production_opex_brl`
C_{CO_2}, u_{CO_2}, c_{CO_2}	CO_2 opex, CO_2 use, CO_2 unit cost	`co2_opex`, `co2_use`, `co2_cost`
C_{GA}, N_w, c_{GA}	G&A cost, G&A well count, G&A rate per well	`g&a_cost`, `g&a_wells`, `sga_opex_well`
b	BOE conversion (Mcf/BOE)	`boe_conversion`
C_t, T_c, T_g	Transport cost, crude/gas transport	`trans_cost`, `crude_trans`, `natgas_trans`
t_c, t_g	Crude and gas transport tariff	`crude_trans_price`, `natgas_trans_price`

Symbol	Description	Code / parameter
C_T, C_I	Tangible and intangible cost	`tang_cost`, `intang_cost`
D_e, D_d, K	Exploratory drill, development drill, capital (`kap`)	`exp_drill`, `dev_drill`, `kap`
f_e, f_d, f_k	Tangible fractions (exploratory, development, capital)	`exp_tang_frac`, `dev_tang_frac`, `kap_tang_frac`
C_{eq}	Equipment cost	`equip_cost`
E_I, A_b, a_I	Expensed intangibles, amortization base, amortization fraction	`expensed_intang`, `amor_base`, `intang_amor_frac`
A_{Ic}, D_T	Amortized intangible, depreciated tangible (EOR credit base)	`amor_intang_cost`, `deprec_tang_cost`
A_m	Amortized intangible expense (state tax)	`amor_intang`
C_{kap}	Capital (`kap`) cost in cash flow	`kap_cost`

Symbol	Description	Code / parameter
D_{pl}, C_{GG}, Q_a, R_r	Depletion, GG&LA cost, annual production, remaining resources	`depletion`, `gg_la_cost`, `annual_production`, `remaining_resources`
NOI	Net operating income	(economic limit calculation)
K_{EOR}, η_{EOR}	EOR tax credit, EOR credit rate	`eor_credit`, `eor_tc_rate`
E_{CH_4}, E_{mcf}, C_{CH_4}, γ_{CH_4}, p_{CH_4}	CH_4 emissions (mt), emissions (mcf), penalty cost, conversion, penalty rate	`ch4_emissions_mt`, `ch4_emissions_mcf`, `ch4_emission_cost`, `ch4_to_metric_tons`, `penalty_rate`
C_{ab}	Abandonment cost	`abandon_cost`

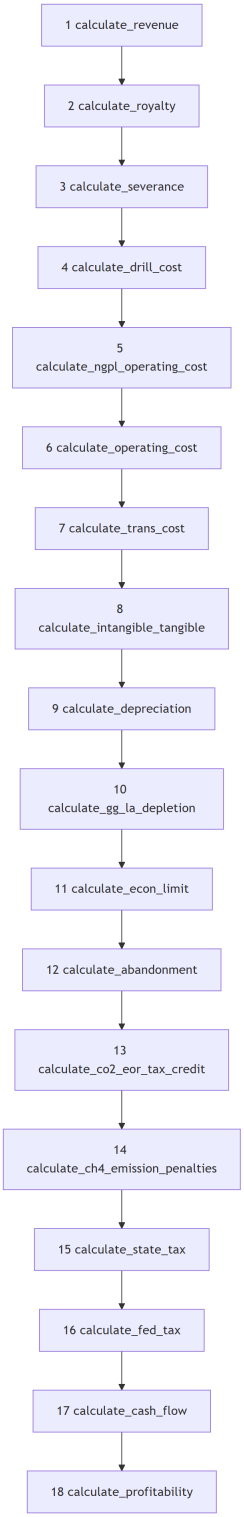
Symbol	Description	Code / parameter
B_s, B_f, T_s, T_f	State/federal tax base, state/federal tax	`tax_base_state`, `tax_base_fed`, `state_tax`, `fed_tax`
τ_s, τ_f	State and federal tax rates	`state_tax_rate`, `fed_tax_rate`
C_{DH}	Dry-hole cost (state tax base)	`dry_hole_cost`
K_f, K_i	Federal and investment credits	`fed_credit`, `invest_credit`
CF, CF_t	Cash flow (undiscounted), cash flow in year t	`cash_flow`
r, NPV, PI, C_K	Discount rate, NPV, profitability index, discounted drilling cost	`discount_rate`, `NPV`, `profitability`, `capital_cost`

Step-by-Step Reference

Every step includes a collapsible **Symbols for this step** list (symbols used in that equation only). The Notation section remains the full legend and code cross-reference.

Step numbers describe the **accounting sequence** in this document. The diagram below shows the order in which `CashFlow` methods run in the Onshore submodule (the fullest path). Offshore and Alaska omit some steps; see the applicability table.

CashFlow execution order



Abandonment (step 12) runs **before** taxes and cash flow: it zeroes production and cost arrays from the economic-limit year onward, then EOR and CH_4 adjustments and tax bases use the truncated series.

Submodule applicability

Doc step	Onshore	Offshore	Alaska	Notes
1–4	Yes	Yes	Yes	Revenue through drilling cost
5–6	Yes	No	No	Offshore/Alaska load aggregated `operating_cost` directly
7	Yes	Yes	Yes	Transportation
8–9	Yes	Yes	Yes	Tangible/intangible split and MACRS
10	Undiscovered conventional only	No	No	GG&LA depletion
11–12	Yes	Yes	Yes	Economic limit and abandonment
13	EOR projects only	No	No	§43 EOR tax credit
14	When `apply_ch4_emission_penalty_switch` is TRUE	No	No	Default switch is FALSE
15–17	Yes	Yes	Yes	State tax, federal tax, cash flow and NPV
18	Yes	No	Yes	Offshore ranks fields by NPV directly, not *PI*

In this section

Step 1 — Revenue (`calculate_revenue`)

$$
\begin{aligned}
R_c &= Q_c \left(P_c - \tau_c \right) \\
R_g &= Q_g \left(P_g - \tau_g \right) \\
R &= R_c + R_g
\end{aligned}
$$

Symbol	Description	Code / parameter
Q_c, Q_g	Crude and natural gas production	`crude_production`, `natgas_production`
P_c, P_g	Crude and natural gas price	`crude_price`, `natgas_price`
τ_c, τ_g	Crude and natural gas tariff	`crude_tariff_price`, `natgas_tariff_price`
R_c, R_g, R	Crude revenue, gas revenue, total revenue	`crude_revenue`, `natgas_revenue`, `revenue`

The tariff prices are typically set to zero by most submodules, so revenue equals price times volume. Revenue excludes NGPL (handled separately in step 5).

Step 2 — Royalty (`calculate_royalty`)

$$F_c = \rho\, R_c$$
$$F_g = \rho\, R_g$$
$$F = F_c + F_g$$

Symbol	Description	Code / parameter
ρ	Royalty rate	`royalty_rate`
R_c, R_g	Crude revenue, gas revenue	`crude_revenue`, `natgas_revenue`
F_c, F_g, F	Crude royalty, gas royalty, total royalty	`crude_royalty`, `natgas_royalty`, `royalty`

Royalty rates are the same for crude and natural gas (time-dependent, project-level). When `on_vented_gas_royalty` is True, royalty is also assessed on the imputed value of vented/flared gas (`vent_natgas_royalty` is added).

Step 3 — Severance Tax (`calculate_severance`)

Severance tax rates are looked up from `state_tax.csv` by state abbreviation. The tax combines revenue-based and production-based components:

$$S = (R_c - F_c)\, s_{c,\text{rev}} + (R_g - F_g)\, s_{g,\text{rev}} + Q_c\, s_{c,\text{prod}} + Q_g\, s_{g,\text{prod}}$$

Symbol	Description	Code / parameter
R_c, R_g	Crude revenue, gas revenue	`crude_revenue`, `natgas_revenue`
F_c, F_g	Crude royalty, gas royalty	`crude_royalty`, `natgas_royalty`
$s_{c,\text{rev}}$, $s_{g,\text{rev}}$	Severance rates on revenue (crude, gas)	`sev_rate_crude_rev`, `sev_rate_natgas_rev`
$s_{c,\text{prod}}$, $s_{g,\text{prod}}$	Severance rates on production (crude, gas)	`sev_rate_crude_prod`, `sev_rate_natgas_prod`
Q_c, Q_g	Crude and natural gas production	`crude_production`, `natgas_production`
S	Severance tax	`severance`

Step 4 — Drilling Cost (`calculate_drill_cost`)

$$C_d = C_e + C_{ed} + C_{dv} + C_{\text{dd}}$$

Symbol	Description	Code / parameter
C_d	Total drilling cost	`drill_cost`
C_e, C_{ed}, C_{dv}, C_{dd}	Exploratory drill/dry, development drill/dry	`exp_drill_cost`, `exp_dry_cost`, `dev_drill_cost`, `dev_dry_cost`

Exploratory and development costs are loaded by each submodule from their respective cost tables before calling `CashFlow`.

Step 5 — NGPL Operating Cost (`calculate_ngpl_operating_cost`) *(Onshore only)*

NGPL is treated as both a revenue offset and a cost:

$$c_n = c_N - \frac{P_n \, V_n \, \kappa}{d_n}$$
$$C_n = Q_g \, c_n$$

Symbol	Description	Code / parameter
c_n, C_n	NGPL net unit cost, NGPL operating cost	`ngpl_net_rate`, `ngpl_operating_cost`
c_N, P_n, V_n, κ, d_n	NGPL cost, price, volume, barrels/gal, divisor	`ngpl_cost`, `ngpl_price`, `ngpl_volume`, `barrels_per_gallon`, `ngpl_volume_divisor`
Q_g	Natural gas production	`natgas_production`

When NGPL price is high, this becomes negative (a revenue offset against opex).

Step 6 — Operating Cost (`calculate_operating_cost`) *(Onshore only)*

Offshore and Alaska load aggregated operating costs directly. For Onshore:

$$C_{oc} = Q_c \, c_o$$
$$C_{og} = \frac{Q_g}{b} \, c_o$$
$$C_{\mathrm{CO_2}} = u_{\mathrm{CO_2}} \, c_{\mathrm{CO_2}}$$
$$C_{\mathrm{GA}} = N_w \, c_{\mathrm{GA}}$$
$$C_o = C_{oc} + C_{og} + C_{\mathrm{CO_2}} + C_{\mathrm{GA}} + C_n$$

Symbol	Description	Code / parameter
C_{oc}, C_{og}	Crude and gas operating cost	`crude_opex`, `natgas_opex`
c_o	Production opex per barrel	`production_opex_brl`
$C_{\mathrm{CO_2}}$, $u_{\mathrm{CO_2}}$, $c_{\mathrm{CO_2}}$	CO_2 opex, CO_2 use, CO_2 unit cost	`co2_opex`, `co2_use`, `co2_cost`
C_{GA}, N_w, c_{GA}	G&A cost, G&A well count, G&A rate per well	`g&a_cost`, `g&a_wells`, `sga_opex_well`
C_o, C_n	Total operating cost, NGPL operating cost	`operating_cost`, `ngpl_operating_cost`
Q_c, Q_g	Crude and natural gas production	`crude_production`, `natgas_production`
b	BOE conversion (Mcf/BOE)	`boe_conversion`

`boe_conversion` = 5.62 Mcf/BOE.

Step 7 — Transportation Cost (`calculate_trans_cost`)

$$T_c = Q_c \, t_c$$
$$T_g = \frac{Q_g}{b} \, t_g$$
$$C_t = T_c + T_g$$

Symbol	Description	Code / parameter
T_c, T_g	Crude and gas transport cost dollars	`crude_trans`, `natgas_trans`
t_c, t_g	Crude and gas transport tariff	`crude_trans_price`, `natgas_trans_price`
C_t	Total transport cost	`trans_cost`
Q_c, Q_g	Crude and natural gas production	`crude_production`, `natgas_production`
b	BOE conversion (Mcf/BOE)	`boe_conversion`

Tariff rates by region are specified in the submodule input files (for example, Alaska uses region-specific tariffs from `ak_mapping.csv`).

Step 8 — Intangible/Tangible Split (`calculate_intangible_tangible`)

$$C_T = D_e\, f_e + D_d\, f_d + K\, f_k$$
$$C_I = D_e\,(1 - f_e) + D_d\,(1 - f_d) + K\,(1 - f_k) + C_{\text{eq}}$$

Symbol	Description	Code / parameter
C_T, C_I	Tangible and intangible cost	`tang_cost`, `intang_cost`
D_e, D_d, K	Exploratory drill, development drill, capital (`kap`)	`exp_drill`, `dev_drill`, `kap`
f_e, f_d, f_k	Tangible fractions (exploratory, development, capital)	`exp_tang_frac`, `dev_tang_frac`, `kap_tang_frac`
C_{eq}	Equipment cost	`equip_cost`

The fractions `exp_tang_frac`, `dev_tang_frac`, `kap_tang_frac` determine how much of each cost category is capitalized (tangible) versus expensed (intangible). See *Symbol conventions* for other uses of K.

Step 9 — Depreciation and Amortization (`calculate_depreciation`)

$$E_I = C_I\,(1 - a_I)$$
$$A_b = C_I\, a_I$$

Symbol	Description	Code / parameter
E_I	Expensed intangibles	`expensed_intang`
A_b	Amortization base	`amor_base`
a_I	Amortization fraction	`intang_amor_frac`
C_I	Intangible cost	`intang_cost`

Amortized intangibles and depreciated tangibles are spread forward according to MACRS schedules from `depreciation_schedules.csv`, which the code looks up by schedule name (`amor_schedule`, `deprec_schedule`) for each project.

Step 10 — GG&LA Depletion (`calculate_gg_la_depletion`) *(Onshore undiscovered only)*

Geological, geophysical, and lease acquisition (GG&LA) costs are depleted against production from undiscovered conventional resources:

$$D_{pl} = C_{\text{GG}}\,\frac{Q_a}{Q_a + R_r}$$

Symbol	Description	Code / parameter
D_{pl}	Depletion	`depletion`
C_{GG}	GG&LA cost	`gg_la_cost`
Q_a	Annual production	`annual_production`
R_r	Remaining resources (depletion denominator)	`remaining_resources`

This step applies only to the Onshore undiscovered conventional resource category. See *Symbol conventions* for R_r versus revenue R.

Step 11 — Economic Limit (`calculate_econ_limit`)

Net Operating Income (NOI) determines the project's economic life:

$$\text{NOI} = R - F - S - C_o - C_t + K_f$$

Symbol	Description	Code / parameter
NOI	Net operating income	(economic limit calculation)
R	Total revenue	`revenue`
F	Total royalty	`royalty`
S	Severance tax	`severance`
C_o	Total operating cost	`operating_cost`
C_t	Total transport cost	`trans_cost`
K_f	Federal credit	`fed_credit`

Net operating income (NOI) is the annual operating margin after royalties, severance, operating expense, and transport, plus federal credits (`fed_credit`) that the model treats as improving operating economics before the abandonment test. The first year with negative NOI marks when primary operations are no longer profitable; step 12 then truncates production and costs. `min_years_before_abandon` (default: 6) prevents early shutdown regardless of NOI, reflecting contractual or operational minimum project life.

Step 12 — Abandonment (`calculate_abandonment`)

Symbol	Description	Code / parameter
C_{ab}	Abandonment cost (`abandon_rate` in shutdown year)	`abandon_cost`
—	Economic life set from abandonment year (default 30 if still economic)	`econ_life`

After the economic limit year, all production, revenue, cost, and tax arrays are zeroed out. Abandonment costs (`abandon_rate`) are applied in the shutdown year.

The economic life (`econ_life`) property is set from the abandonment year. Projects that remain economical through the projection period receive `econ_life = 30` (default).

Step 13 — CO_2 EOR Tax Credit (`calculate_co2_eor_tax_credit`) *(EOR projects only)*

The U.S. Code §43 enhanced oil recovery tax credit is applied to EOR projects:

$$K_{\text{EOR}} = (A_{Ic} + D_T)\,\eta_{\text{EOR}}$$

Symbol	Description	Code / parameter
K_{EOR}	EOR tax credit	`eor_credit`
A_{Ic}	Amortized intangible (EOR credit base)	`amor_intang_cost`
D_T	Depreciated tangible (EOR credit base)	`deprec_tang_cost`
η_{EOR}	EOR credit rate	`eor_tc_rate`

The credit is set to zero when `crude_price > eor_tc_phaseout`.

Step 14 — CH_4 Emission Penalties (`calculate_ch4_emission_penalties`) *(when switch is ON)*

$$E_{\text{CH}_4} = \frac{E_{\text{mcf}}}{\gamma_{\text{CH}_4}}$$
$$C_{\text{CH}_4} = E_{\text{CH}_4}\, p_{\text{CH}_4}$$

Symbol	Description	Code / parameter
$E_{\mathrm{CH_4}}$	CH_4 emissions (metric tons)	`ch4_emissions_mt`
E_{mcf}	CH_4 emissions (mcf)	`ch4_emissions_mcf`
$\gamma_{\mathrm{CH_4}}$	MCF to metric ton conversion	`ch4_to_metric_tons`
$C_{\mathrm{CH_4}}$	CH_4 penalty cost	`ch4_emission_cost`
$p_{\mathrm{CH_4}}$	CH_4 penalty rate	`penalty_rate`

The code default for `ch4_to_metric_tons` is 51.921 Mcf per metric ton.

Controlled by `apply_ch4_emission_penalty_switch` in `setup.csv` — currently `FALSE` by default (no penalties applied).

Step 15 — State Tax (`calculate_state_tax`)

$$
\begin{aligned}
B_s &= R - F - S - C_o - C_{\mathrm{ab}} - E_I - A_m - D_{pl} - D_T \\
&\quad - C_{\mathrm{DH}} - C_t \\
T_s &= B_s \, \tau_s
\end{aligned}
$$

Symbol	Description	Code / parameter
B_s	State tax base	`tax_base_state`
T_s	State income tax	`state_tax`
τ_s	State tax rate	`state_tax_rate`
R	Total revenue	`revenue`
F	Total royalty	`royalty`
S	Severance tax	`severance`
C_o	Total operating cost	`operating_cost`
C_{ab}	Abandonment cost	`abandon_cost`
E_I	Expensed intangibles	`expensed_intang`
A_m	Amortized intangible expense (state tax)	`amor_intang`
D_{pl}	Depletion	`depletion`
D_T	Depreciated tangible	`deprec_tang_cost`
C_{DH}	Dry-hole cost (state tax base)	`dry_hole_cost`
C_t	Total transport cost	`trans_cost`

State tax rates are looked up from `state_tax.csv` by state abbreviation.

Step 16 — Federal Tax (`calculate_fed_tax`)

$$
\begin{aligned}
B_f &= B_s - T_s - K_{\mathrm{EOR}} \\
T_f &= B_f \, \tau_f - K_f - K_i
\end{aligned}
$$

Symbol	Description	Code / parameter
B_f	Federal tax base	`tax_base_fed`
T_f	Federal income tax	`fed_tax`
B_s	State tax base	`tax_base_state`
T_s	State income tax	`state_tax`
K_{EOR}	EOR tax credit	`eor_credit`
τ_f	Federal tax rate	`fed_tax_rate`
K_f	Federal credit	`fed_credit`
K_i	Investment credit	`invest_credit`

The federal tax rate (`fed_tax_rate`) is set from `discounting.csv` (0.21 for *AEO*2026).

Step 17 — Cash Flow and NPV (`calculate_cash_flow`)

$$CF = R - F - S - C_d - C_{\text{eq}} - C_{\text{kap}} - C_{\text{GG}} - C_o - C_{\text{ab}} - T_s - T_f - C_t - C_{\text{CH}_4}$$

$$\text{NPV} = \sum_{t=0}^{T} \frac{CF_t}{(1+r)^t}$$

Symbol	Description	Code / parameter
CF, CF_t	Cash flow (undiscounted), cash flow in year t	`cash_flow`
NPV	Net present value	`NPV`
r	Discount rate (WACC)	`discount_rate`
t, T	Year index, evaluation horizon (last year)	—
R	Total revenue	`revenue`
F	Total royalty	`royalty`
S	Severance tax	`severance`
C_d	Total drilling cost	`drill_cost`
C_{eq}	Equipment cost	`equip_cost`
C_{kap}	Capital (`kap`) cost in cash flow	`kap_cost`
C_{GG}	GG&LA cost	`gg_la_cost`
C_o	Total operating cost	`operating_cost`
C_{ab}	Abandonment cost	`abandon_cost`
T_s, T_f	State and federal income tax	`state_tax`, `fed_tax`
C_t	Total transport cost	`trans_cost`
C_{CH_4}	CH_4 penalty cost	`ch4_emission_cost`

t is the year index and T is the evaluation horizon — not transport cost C_t or tariffs t_c, t_g. See *Symbol conventions*.

CF is the undiscounted annual project cash flow after drilling, equipment, capital, GG&LA, operating, abandonment, taxes, transport, and CH_4 penalties. NPV sums discounted CF_t over the evaluation horizon at project WACC (`discount_rate`); a positive NPV means the project returns more than the cost of capital in present-value terms. NPV is the primary dollar metric before normalization in step 18.

Step 18 — Profitability Index (`calculate_profitability`)

$$C_K = \sum_{t=0}^{T} \frac{C_{d,t}}{(1+r)^t}$$

$$\text{PI} = \frac{\text{NPV}}{C_K}$$

Symbol	Description	Code / parameter
C_K	Present value of drilling costs (profitability denominator)	`capital_cost`
PI	Profitability index	`profitability`
NPV	Net present value	`NPV`
$C_{d,t}$	Drilling cost in year t	`drill_cost`
r	Discount rate (WACC)	`discount_rate`
t, T	Year index, evaluation horizon (last year)	—

t is the year index and T is the evaluation horizon — not transport cost C_t or tariffs t_c, t_g. See *Symbol conventions*.

C_K is the present value of drilling costs only—the capital put at risk upfront. Profitability index (PI) divides NPV by C_K so projects of different sizes rank on comparable terms in *Project Selection and Constraints*. PI > 0 is equivalent to NPV > 0 when drilling cost is positive; projects with PI ≤ 0 are not drilled.

Projects are ranked by profitability index in the project selection step. A profitability index > 0 indicates a positive NPV relative to drilling cost.

Cost Components Summary

Step	Method	Formula summary	Units	Applies to
1	`calculate_revenue`	$(P_c - \tau_c)\,Q_c + (P_g - \tau_g)\,Q_g$	$/yr	All
2	`calculate_royalty`	$\rho\,R_c + \rho\,R_g$	$/yr	All
3	`calculate_severance`	Revenue- and production-based state rates on $R - F$, Q	$/yr	All
4	`calculate_drill_cost`	$C_e + C_{ed} + C_{dv} + C_{\text{dd}}$	$	All
5	`calculate_ngpl_operating_cost`	$Q_g\,c_n$; NGPL net in c_n	$/yr	Onshore only
6	`calculate_operating_cost`	$C_{oc} + C_{og} + \text{CO}_2 + \text{GA} + C_n$	$/yr	Onshore only
7	`calculate_trans_cost`	$Q_c\,t_c + (Q_g/b)\,t_g$	$/yr	All
8	`calculate_intangible_tangible`	Split D_e, D_d, K into C_T, C_I	$	All
9	`calculate_depreciation`	E_I, A_b from C_I; MACRS schedules	$/yr	All
10	`calculate_gg_la_depletion`	$C_{\text{GG}}\,Q_a/(Q_a + R_r)$	$/yr	Onshore undiscovered only
11	`calculate_econ_limit`	NOI $= R - F - S - C_o - C_t + K_f$	$/yr	All
12	`calculate_abandonment`	Zero arrays after econ limit; `abandon_cost` from `abandon_rate`	$	All
13	`calculate_co2_eor_tax_credit`	$K_{\text{EOR}} = (A_{Ic} + D_T)\,\eta_{\text{EOR}}$	$/yr	EOR projects only
14	`calculate_ch4_emission_penalties`	$C_{\text{CH}_4} = E_{\text{CH}_4}\,p_{\text{CH}_4}$	$/yr	Onshore when switch ON
15	`calculate_state_tax`	$T_s = B_s\,\tau_s$ (see step equation)	$/yr	All
16	`calculate_fed_tax`	$T_f = B_f\,\tau_f - K_f - K_i$	$/yr	All
17	`calculate_cash_flow`	CF then NPV $= \sum(CF_t/(1+r)^t)$	,NPV	All
18	`calculate_profitability`	PI $=$ NPV$/C_K$	dimensionless	Onshore, Alaska

Key Input Parameters

Parameter	Source file	Description	AEO2026 value
`fed_tax_rate`	`discounting.csv`	Federal corporate income tax rate	0.21
`debt_ratio`	`discounting.csv`	Debt fraction of total capital	0.40
`industry_beta`	`discounting.csv`	Industry equity beta	1.5
`market_risk_premium`	`discounting.csv`	Equity risk premium	7.5%
`return_over_capital_cost`	`discounting.csv`	Return over cost of capital	5%
`min_years_before_abandon`	Code default	Minimum operating years before abandonment	6
Cash-flow horizon	`evaluation_years` in submodule setup (Offshore/Alaska) or derived from the AEO year range (Onshore)	Length of project cost/revenue arrays; NPV sums $t = 0 \ldots T$ where $T =$ evaluation_years $- 1$	30 (typical)
`ch4_to_metric_tons`	Code default	MCF → metric ton CH_4 conversion	51.921
`boe_conversion`	Code default	MCF per barrel of oil equivalent	5.62
State severance rates	`state_tax.csv`	Revenue-based and production-based rates	By state
Depreciation schedules	`depreciation_schedules.csv`	MACRS schedule fractions	By method

Debug Output

When `debug_switch = TRUE` in `setup.csv`, the Alaska submodule writes detailed cash flow output to two CSV files via `CashFlow.to_csv()` under `cashflow_debug/`:

- `ak_cash_flow_props.csv` — project-level properties (prices, rates, NPV, profitability)
- `ak_cash_flow_costs.csv` — time-series arrays for all 35+ cost and revenue components

These files append across iterations, with columns for year, iteration, and cycle. Onshore and offshore cash flow CSV export is currently disabled in code.

Related Pages

- *Price and Discount Rate Calculations* — WACC calculation and regional price mapping
- *Project Selection and Constraints* — How projects are ranked and drilled after DCF evaluation
- *Lower 48 Onshore Submodule* — Onshore resource category setup
- *cash_flow — Discounted Cash Flow Engine* — Auto-generated API reference

Drilling and Production Equations

The `drilling_equations.py` module contains the drilling schedule and production profile functions used by the Onshore, Offshore, and Alaska submodules. These functions translate economic decisions (which projects are profitable) into physical quantities (wells drilled and barrels/Mcf produced).

Conceptual Overview

After *Project Selection and Constraints* fills the drilling queue, `drilling_equations` computes well counts and production time series. Onshore links drilling to price through an adjustment factor α_p; offshore and Alaska use field-specific schedules and type curves. Production rates $q(t)$ feed back into *Economic Framework — Discounted Cash Flow Model* revenue and cost steps in subsequent years.

Notation

Symbol conventions

Letter	Meaning	Where used
w_y, $w_{\max}$	Wells drilled in year y, max drill rate	Onshore
$q(t)$, q_{peak}	Production rate, peak rate	Offshore, Alaska
α_p	Price adjustment factor	Onshore
t, y	Time index (years)	All profiles
δ	Post-saturation drill-rate decline fraction	Onshore

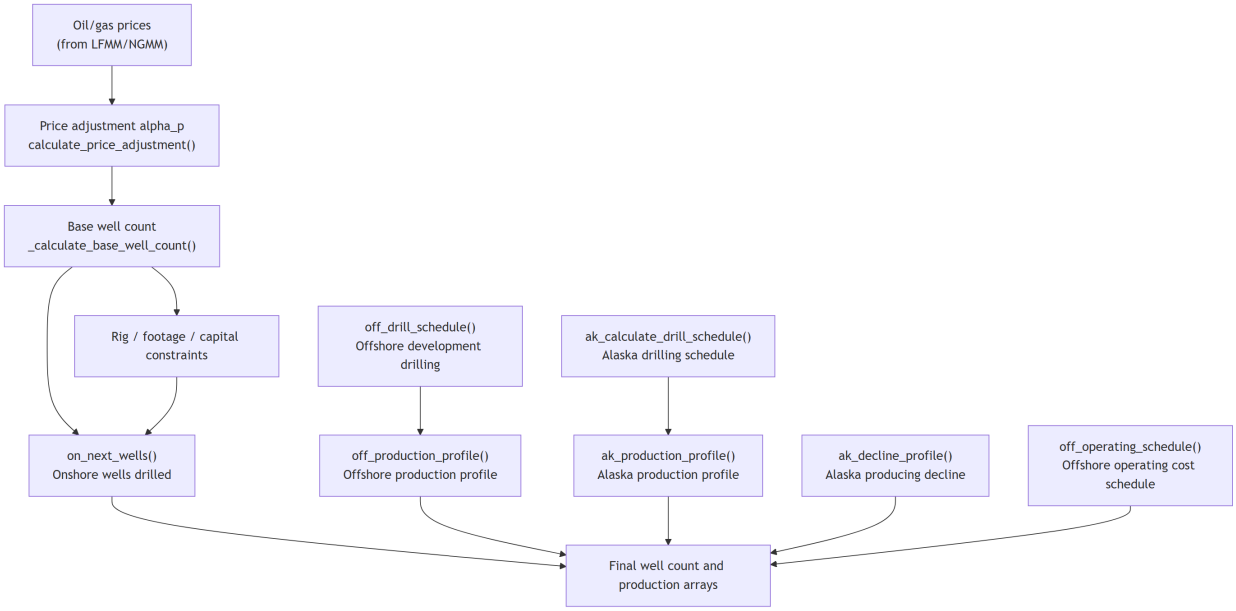
Symbol	Description	Code / parameter
w_y	Wells drilled in year y	wells per year output
$w_{\max}$	Maximum wells per year at capacity	`max_drill_rate`
δ	Annual decline fraction after saturation	`max_drill_rate_frac`
y_0	Year decline phase begins	`decline_start_year`
α_p	Price adjustment multiplier	`price_adj`
W_{lim}	Well decline limit (pattern size)	`well_decline_limit`

Symbol	Description	Code / parameter
$q_{\text{bu}}(t)$	Build-up phase rate	buildup segment
q_{pl}	Plateau rate	plateau segment
$q_{\text{dec}}(t)$	Decline phase rate	decline segment
q_{peak}	Peak production rate	`peak_rate`
T_{bu}	Build-up years	`buildup_years`
d	Decline rate	`decline_rate`

Symbol	Description	Code / parameter
q_{peak}	Peak production	`peak_production`
q_0	Initial production rate	`initial_prod_rate`
ϕ	Resource size factor	`resource_factor`
q_{cap}	Maximum production cap	`max_production_rate`
d_o	Oil decline rate	`oil_decline_rate`

In this section

Overview



Onshore Well Count — `on_next_wells`

`on_next_wells` is the primary onshore drilling equation. It determines how many wells are drilled in the current year for a given continuous or undiscovered project, as a function of:

- Past drilling (cumulative wells drilled in prior years)
- Capacity constraints (maximum drill rate, ramp-up period)
- Current oil and gas prices (price adjustment α_p)

Base Well Count (`_calculate_base_well_count`)

The base well count simulates drilling forward year-by-year from the project start date until cumulative drilling matches the actual past drilling. Key parameters:

Parameter	Description
`well_decline_limit`	Pattern size: total wells at which the play area reaches saturation (W_{lim})
`max_wells`	Absolute maximum wells possible in the project
`past_drilling`	Cumulative wells drilled in prior model years
`max_drill_rate`	Maximum wells per year at full capacity ($w_{\max}$)
`max_drill_rate_frac`	Annual exponential decline rate applied after saturation (δ)
`drill_predecline`	Fraction of `well_decline_limit` at which decline begins (default 0.70)
`ramp_up_years`	Years to reach `max_drill_rate` from zero

Ramp-up period: Drilling increases linearly from 0 to $w_{\max}$ over `ramp_up_years`. A minimum floor of 5 wells/year is applied.

Post-decline period: Once cumulative drilling exceeds `drill_predecline × well_decline_limit`:

$$w_y = w_{\max}\,(1-\delta)^{y-y_0}$$

Symbol	Description	Code / parameter
w_y	Wells drilled in year y	year wells output
$w_{\max}$	Max drill rate	`max_drill_rate`
δ	Post-saturation decline fraction	`max_drill_rate_frac`
y	Model year	year index
y_0	Decline start year	`decline_start_year`

Price Adjustment

The base well count is then scaled by a price adjustment factor:

$$\alpha_p = f(P_{\text{cur}},\, P_{\text{base}},\, Q_o,\, Q_g)$$

Symbol	Description	Code / parameter
α_p	Price adjustment factor	`price_adj`
P_{cur}	Current price index	`current_price`
P_{base}	Calibration baseline price	`base_price`
Q_o, Q_g	Oil and gas production weights	`oil_prod`, `gas_prod`
f	Adjustment function	`calculate_price_adjustment()`

The price adjustment reflects the industry's response to price changes: higher prices lead to more drilling; lower prices reduce activity. The functional form uses a production-weighted price index comparing current prices to the baseline prices from the project's calibration period.

The `low_price_flag` prevents drilling from increasing above the base rate during low- price periods.

Offshore Drilling Schedule — `off_drill_schedule`

The offshore drilling schedule distributes development wells across time once a field has been approved for development. Parameters:

Parameter	Description
Field size class	Determines development well count from `off_undiscovered_fields.csv`
Platform drill rate	Maximum wells per platform per year (`off_platform_drill_per_year.csv`)
Development delays	Lead time from discovery to first production (`off_delays.csv`)

The schedule returns a time-indexed array of development wells per year for each project.

Offshore Operating Schedule — `off_operating_schedule`

`off_operating_schedule` generates the time-indexed operating cost schedule for an offshore field. Operating costs are loaded from `off_operating_cost.csv` by water depth category and scaled to the field's production volume.

Offshore Production Profile — `off_production_profile`

Production profiles for offshore fields use type curves from `off_prod_profile.csv`:

1. **Build-up phase** — production ramps up over several years after first oil
2. **Plateau phase** — production holds at peak for a number of years proportional to field size
3. **Decline phase** — production declines exponentially after plateau

The profile shape is parameterized by field size class and water depth. Fields in deeper water tend to have longer lead times but higher peak rates.

$$q_{\text{bu}}(t) = q_{\text{peak}} \frac{t}{T_{\text{bu}}} \quad \text{(linear buildup)}$$
$$q_{\text{pl}} = q_{\text{peak}} \quad \text{(flat plateau)}$$
$$q_{\text{dec}}(t) = q_{\text{peak}}\, e^{-d\,t} \quad \text{(exponential decline)}$$

Symbol	Description	Code / parameter
$q_{\text{bu}}(t)$	Build-up rate	buildup segment
q_{pl}	Plateau rate	plateau segment
$q_{\text{dec}}(t)$	Decline rate	decline segment
q_{peak}	Peak rate	`peak_rate`
T_{bu}	Build-up duration (years)	`buildup_years`
d	Decline rate	`decline_rate`
t	Years from phase start	time index

Alaska Drilling Schedule — `ak_calculate_drill_schedule`

The Alaska drilling schedule uses a 29-year template from `ak_dev_drill_schedule.csv` and `ak_exp_drill_schedule.csv`. The schedule is:

1. Fixed (deterministic) for **known projects** — based on publicly available operator development plans
2. Probabilistic for **undiscovered projects** — scheduled to begin after the discovery year

---

Alaska Production Profile — `ak_production_profile`

Alaska production profiles use a two-parameter model:

$$q_{\text{peak}} = \min(q_0\,\phi,\ q_{\text{cap}})$$

Symbol	Description	Code / parameter
q_{peak}	Peak production	`peak_production`
q_0	Initial production rate	`initial_prod_rate`
ϕ	Resource size factor	`resource_factor`
q_{cap}	Maximum production cap	`max_production_rate`

where ϕ scales the initial production rate based on resource size relative to the reference size. Peak production is capped at q_{cap} from `ak_projects_known.csv`.

Alaska Decline Profile — `ak_decline_profile`

Once production begins, Alaska projects decline according to:

$$q(t) = q_{\text{peak}}\, e^{-d_o\, t}$$

Symbol	Description	Code / parameter
$q(t)$	Production in year t	production array
q_{peak}	Peak production	`peak_production`
d_o	Oil decline rate	`oil_decline_rate`
t	Years from production start	time index

with a secondary decline rate (`fc_decline`) applied after the primary decline has reduced production to a threshold fraction. Both decline rates are defined per project in `ak_projects_known.csv`.

Submodule applicability

Function	Onshore	Offshore	Alaska	Notes
`on_next_wells`	Yes	No	No	Price adjustment α_p
`off_drill_schedule`	No	Yes	No	Development well timing
`off_operating_schedule`	No	Yes	No	OPEX by water depth
`off_production_profile`	No	Yes	No	Build-up / plateau / decline
`ak_calculate_drill_schedule`	No	No	Yes	29-year templates
`ak_production_profile`	No	No	Yes	Peak from resource size
`ak_decline_profile`	No	No	Yes	Primary + secondary decline

Related Pages

- *Economic Framework — Discounted Cash Flow Model* — Economic evaluation that selects which projects to drill
- *Project Selection and Constraints* — How wells from these equations are filtered by constraints
- *drilling_equations — Drilling and Production Functions* — Auto-generated API reference

Price and Discount Rate Calculations

HSM converts *NEMS* nominal prices into project-level inputs in real **1987 dollars** and computes the weighted average cost of capital (*WACC*) used as the discount rate r in *DCF* evaluations (*Economic Framework — Discounted Cash Flow Model*, steps 17–18).

Primary code: `module_unf.py` (`calculate_discount_rate`, price processing methods)

Conceptual Overview

Prices enter HSM from *LFMM* (crude) and *NGMM* (natural gas). Before any project economics run, `module_unf` maps those arrays to HSM districts and regions, deflates to 1987 dollars, and computes a time-varying WACC from *MAM* macro inputs and `discounting.csv`. The same real-dollar convention applies to all cost tables in `input/`.

Notation

Equations below use mathematical symbols. Expand each group to map symbols to code names. The Notation section is the full legend; each equation has a collapsible **Symbols for this equation** panel.

Symbol conventions

Letter	Meaning	Where used
r	Discount rate (WACC) in DCF	WACC section; *Economic Framework — Discounted Cash Flow Model*
r_e, r_d	Cost of equity, cost of debt	WACC
i	Region index in price-weighted sums	Regional mapping
d	HSM district index	Regional mapping
y	Calendar / model year	Inflation, technology switch
P, P_d, P_i	Price (generic, district, region i)	Mapping, inflation

Symbol	Description	Code / parameter
WACC, r	Weighted average cost of capital (discount rate)	`discount_rate`
w_d	Debt fraction of capital	`debt_ratio`
r_d	Cost of debt	`cost_of_debt`
r_e	Cost of equity	`cost_of_equity`
τ_f	Federal corporate income tax rate	`fed_tax_rate`
r_f	Risk-free rate (10-year treasury)	`risk_free_rate`
β	Industry equity beta	`industry_beta`
MRP	Market risk premium	`market_risk_premium`
r_b	Corporate bond rate (from MAM)	`bond_rate`
δ_c	Return over cost of capital	`return_over_capital_cost`

Symbol	Description	Code / parameter
P_d	District-level price	district price arrays in `module_unf`
P_i	LFMM region price	`lfmm_price` (conceptual)
$Q_{d,i}$	Production in district d from region i	`production` (weighted sum)

Symbol	Description	Code / parameter
P_{1987}	Real price in 1987 dollars	`real_price` after inflation
P^{nom}	Nominal price from NEMS	input before deflation
D_y	GDP price deflator (MAM)	`rest_mc_jpgdp` / `jpgdp`
y_0	Base price year (1987)	`base_price_year`
$C(y)$	Cost in year y after tech improvement	adjusted cost
C_0	Base calibrated cost	`base_cost`
γ	Technology improvement rate	`improvement_rate`
y_{tech}	First year tech lever applies	`technology_year`

In this section

- *Weighted Average Cost of Capital (WACC)* — Weighted average cost of capital
- *Regional Price Mapping* — LFMM/NGMM to HSM districts
- *Inflation Adjustment* — 1987-dollar deflator
- *Technology Year Switch* — Post-calibration cost reduction

Weighted Average Cost of Capital (WACC)

The WACC is the discount rate r used in all DCF calculations (`CashFlow.calculate_cash_flow`). It is computed in `module_unf.calculate_discount_rate()` using `discounting.csv` and MAM-supplied macroeconomic variables.

HSM uses a standard CAPM-based WACC:

$$r = \text{WACC} = w_d \, r_d \, (1 - \tau_f) + (1 - w_d) \, r_e$$

Symbol	Description	Code / parameter
r, WACC	Discount rate	`discount_rate`
w_d	Debt ratio	`debt_ratio`
r_d	Cost of debt	`cost_of_debt`
r_e	Cost of equity	`cost_of_equity`
τ_f	Federal tax rate	`fed_tax_rate`

where:

$$r_e = r_f + \beta \, \text{MRP}$$
$$r_d = r_b + \delta_c$$

Symbol	Description	Code / parameter
r_e	Cost of equity	`cost_of_equity`
r_f	Risk-free rate	`risk_free_rate`
β	Industry beta	`industry_beta`
MRP	Market risk premium	`market_risk_premium`
r_d	Cost of debt	`cost_of_debt`
r_b	Bond rate	`bond_rate`
δ_c	Return over cost of capital	`return_over_capital_cost`

r_f is the 10-year treasury yield from MAM; r_b is the corporate bond rate from MAM.

`discounting.csv` Parameters (AEO2026)

Parameter	Value	Description
`debt_ratio`	0.40	w_d: fraction of capital financed by debt
`fed_tax_rate`	0.21	τ_f: federal corporate income tax rate
`industry_beta`	1.5	β: oil and gas industry equity beta
`market_risk_premium`	7.5%	MRP: equity risk premium above risk-free rate
`return_over_capital_cost`	5.0%	δ_c: spread over cost of capital

The bond rate and treasury yield are provided dynamically by MAM each year, so r varies over the projection period as macroeconomic conditions change. That time path feeds directly into NPV and PI in *Economic Framework — Discounted Cash Flow Model*.

Regional Price Mapping

NEMS provides crude and gas prices through different pathways. Crude prices are provided at LFMM-region granularity, while natural gas prices come from NGMM district and regional arrays. HSM needs district-level (84 districts) and HSM-region-level prices for economic calculations.

Mapping Steps

1. **Crude only — LFMM region → HSM district:** `mapping.csv` column `lfmm_region_number` maps each of the 84 districts to an LFMM price region. Districts that span multiple LFMM regions receive a production-weighted average price.
2. **Crude only — price-weighted averaging:** `_calculate_production_weighted_price()` in `module_unf.py` computes:

$$P_d = \frac{\sum_i Q_{d,i} P_i}{\sum_i Q_{d,i}}$$

Symbol	Description	Code / parameter
P_d	District price	district price output
P_i	LFMM region price	`lfmm_price`
$Q_{d,i}$	Production attributed to district d, region i	`production`
i	LFMM region index	(loop over intersecting regions)
d	HSM district index	district

   where d is the district and i iterates over the LFMM regions that intersect it.
3. **Natural gas pathway:** `_process_natural_gas_prices()` loads district prices from `ngtdmrep_ogprcng` and regional prices from `ngtdmrep_ogwprng`, computes production-weighted HSM regional gas prices from district data, and uses restart-file regional values as fallback where district-based aggregation is unavailable.

Price Processing Methods (in `module_unf.py`)

Method	Description
`_process_crude_prices`	Converts LFMM crude prices to HSM district and region prices
`_process_natural_gas_prices`	Processes NGMM district/regional gas prices into HSM district/region prices
`_calculate_production_weighted_price`	Production-weighted price averaging

Inflation Adjustment

All prices in HSM are in real **1987 dollars**, regardless of the dollar year NEMS provides them in. The conversion uses the GDP price deflator from MAM (`rest_mc_jpgdp`):

$$P_{1987} = P^{\text{nom}} \frac{D_{y_0}}{D_y}$$

Symbol	Description	Code / parameter
P_{1987}	Real price in 1987 dollars	output of `calculate_inflation`
P^{nom}	Nominal price from NEMS	input price
D_y	GDP deflator in year y	`jpgdp` / `rest_mc_jpgdp`
y_0	Base price year (1987)	`base_price_year`

This matches the arithmetic in `common.calculate_inflation()`. Holding costs and revenues in a single real dollar year lets DCF steps in *Economic Framework — Discounted Cash Flow Model* compare cash flows across years without mixing nominal vintages.

`base_price_year = 1987` is set in `setup.csv`. All cost tables in the input files (drilling costs, operating costs, etc.) are in 1987 dollars to match.

Technology Year Switch

Technology improvement rates from `on_tech_levers.csv` are applied to reduce costs for years **after** `technology_year`:

$$C(y) = C_0 \, (1 - \gamma)^{y - y_{\text{tech}}}, \quad y > y_{\text{tech}}$$

Symbol	Description	Code / parameter
$C(y)$	Adjusted cost in year y	adjusted cost output
C_0	Base calibrated cost	`base_cost`
γ	Improvement rate	`improvement_rate`
y	Model year	year index
y_{tech}	Technology switch year	`technology_year`

For AEO2026, `technology_year = 2025`. This means AEO2026 is the first year with technology improvements relative to the base cost calibration.

Related Pages

- *Economic Framework — Discounted Cash Flow Model* — WACC as discount rate r in *NPV* and *PI*
- *NEMS Integration* — How MAM provides bond rates and treasury yields
- *module_unf — Core Orchestrator* — Auto-generated API reference for `module_unf.py`

Project Selection and Constraints

After the *DCF* model evaluates all candidate projects, *HSM* ranks them and applies physical and economic constraints to determine which projects actually receive investment in each model year.

Primary code: `onshore.py` (project selection methods)

Conceptual Overview

Selection is a two-stage process: rank by profitability index PI from *Economic Framework — Discounted Cash Flow Model* (step 18), then trim the drilling queue until rig, footage, or capital limits bind. Only onshore continuous, *EOR*, and undiscovered categories use the full constraint stack; producing wells are evaluated for economic limit but not re-drilled.

Notation

Equations use mathematical symbols aligned with *Economic Framework — Discounted Cash Flow Model*. Each display equation has a collapsible **Symbols for this equation** panel.

Symbol conventions

Letter	Meaning	Where used
PI, NPV, C_K	Profitability, *NPV*, discounted drilling cost	Ranking (same as step 18)
p	Oil/gas price index for constraint equations	Rig, footage, capital
N_{rigs}	Maximum rigs available	Rig constraint
$W_{\max}$	Maximum wells drillable	Rig → wells conversion
$L_{\max}$	Maximum drilling footage	Footage constraint
$K_{\max}$	Maximum capital expenditure	Capital constraint

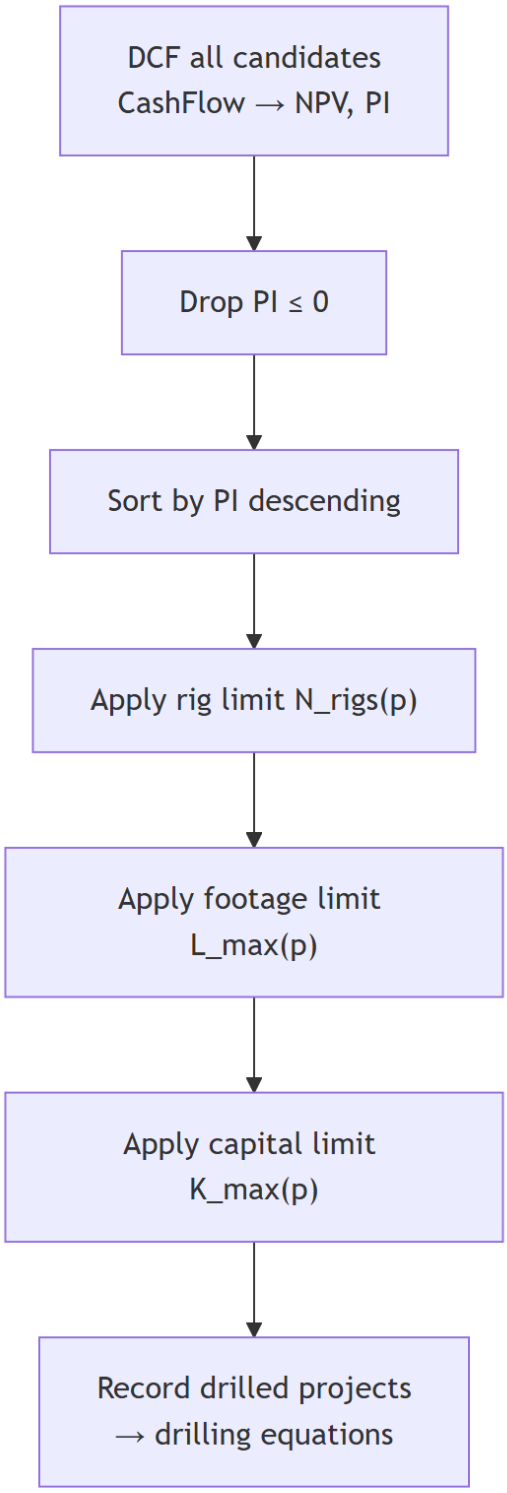
Symbol	Description	Code / parameter
PI	Profitability index	`profitability`
NPV	Net present value	`NPV`
C_K	Present value of drilling costs	`capital_cost`

Symbol	Description	Code / parameter
$N_{\text{rigs}}(p)$	Max rigs as function of price	`max_rigs` (polynomial)
ω	Wells per rig	`wells_per_rig`
$W_{\max}$	Maximum wells	derived well cap
$L_{\max}(p)$	Maximum footage	`max_footage`
$K_{\max}(p)$	Maximum capex	`max_capex`
a, b, c	Rig constraint polynomial coefficients	`on_rig_constraint_eq.csv`

In this section

- *Selection workflow* — Selection flow diagram
- *Selection Methodology* — Profitability index ranking
- *Constraint System* — Rig, footage, and capital limits
- *EOR Special Handling* — CO_2 EOR
- *Undiscovered Conventional Discovery Order* — Discovery order
- *Producing Wells* — Producing projects

Selection workflow



Each constraint is evaluated in sequence; the binding limit sets the final well count for the year. Projects already ranked by PI fill the queue until a cap is reached.

Selection Methodology

Ranking

Projects are ranked by **profitability index** (PI):

$$\text{PI} = \frac{\text{NPV}}{C_K}$$

Symbol	Description	Code / parameter
PI	Profitability index	`profitability`
NPV	Net present value	`NPV`
C_K	Discounted drilling cost (denominator)	`capital_cost`

C_K is the present value of all drilling costs (same as *Step 18 — Profitability Index (calculate_profitability)*). A PI > 0 indicates the project returns more than its drilling cost in NPV terms. Projects with PI ≤ 0 are uneconomic and not drilled.

Selection Process

1. Compute DCF for all candidate projects in the category (producing, continuous, EOR, undiscovered)
2. Filter out projects with PI ≤ 0
3. Sort remaining projects by PI (descending)
4. Apply constraints in order until the binding constraint is reached
5. Record which projects were drilled and update the project DataFrame

Constraint System

The onshore submodule applies three independent constraint types. When multiple constraints would bind simultaneously, each is applied sequentially and the most restrictive one determines the final well count.

Rig Constraints

Maximum rigs available by region is a polynomial function of oil/gas price p:

$$N_{\text{rigs}}(p) = a + b\,p + c\,p^2$$

Symbol	Description	Code / parameter
$N_{\text{rigs}}(p)$	Maximum rigs	rig constraint output
p	Price index for constraint	oil/gas price input
a, b, c	Polynomial coefficients	`on_rig_constraint_eq.csv`

Coefficients are from `on_rig_constraint_eq.csv`. Converting rigs to wells:

$$W_{\text{max}} = N_{\text{rigs}}(p)\,\omega$$

Symbol	Description	Code / parameter
$W_{\max}$	Maximum wells	well cap after rig limit
$N_{\text{rigs}}(p)$	Maximum rigs	from rig constraint
ω	Wells per rig by region/category	`wells_per_rig`

Wells per rig by region and resource category are in `on_wells_per_rig.csv`.

Higher prices expand rig availability and therefore $W_{\max}$, but the *PI*-ranked queue still determines which projects are funded first.

Footage Constraints

Total drilling footage per year is constrained similarly:

$$L_{\max}(p) = f(p)$$

Symbol	Description	Code / parameter
$L_{\max}(p)$	Maximum drilling footage	`max_footage`
p	Price index	oil/gas price
f	Footage constraint function	`on_footage_constraint_eq.csv`

(from `on_footage_constraint_eq.csv`)

Footage per well varies by project type (for example, horizontal wells for tight oil are longer than vertical conventional wells).

Capital Constraints

Total capital expenditure per year:

$$K_{\max}(p) = g(p)$$

Symbol	Description	Code / parameter
$K_{\max}(p)$	Maximum capital expenditure	`max_capex`
p	Price index	oil/gas price
g	Capital constraint function	`on_capital_constraint_eq.csv`

(from `on_capital_constraint_eq.csv`)

When capital is constrained, projects are still ranked by PI but total spending cannot exceed $K_{\max}$.

EOR Special Handling

CO_2 EOR projects have additional selection logic:

- CO_2 availability from *CCATS* constrains the number of active injection sites
- The 200 project slots (`ccatsdat_dem_eor`, `ccatsdat_cst_eor`) are tracked project-by-project
- `ogsmout_ns_start` records the year each EOR project was first selected

Projects that were previously drilled continue to produce (no re-selection needed); the selection logic applies to new projects starting injection.

Undiscovered Conventional Discovery Order

For the undiscovered conventional resource category, the discovery-order model (`on_discovery_order.csv`) controls which resources are available for selection in each year:

- Resources are evaluated from largest (highest expected value) to smallest
- Only resources "discoverable" by the current year (based on exploration effort in prior years) are in the candidate pool

---

Producing Wells

Producing well economics are evaluated but no new drilling decisions are made. The selection logic for producing wells is:

1. Compute DCF to determine economic limit (the year production becomes unprofitable)
2. Zero out production after the economic limit year
3. Apply abandonment costs

No new wells are drilled into producing well projects; they decline organically.

Notes

> **Note**
>
> The exact profitability index threshold and constraint-binding resolution details are implemented in `onshore.py`. The above describes the general design; consult the source code for precise parameter values and edge case handling.

Related Pages

- *Economic Framework — Discounted Cash Flow Model* — DCF evaluation that produces the PI values used here
- *Drilling and Production Equations* — Well count equations applied after selection
- *onshore — Lower 48 Onshore Submodule* — Auto-generated API reference

2.8.3 How These Pages Relate

The methodology pages follow the sequence in which methods are called during a model run:

1. **Price and Discount Rate Calculations** (*Price and Discount Rate Calculations*) — Prices are mapped from NEMS to HSM districts and the WACC is computed before any economic evaluation begins.
2. **Economic Framework** (*Economic Framework — Discounted Cash Flow Model*) — The `CashFlow` DCF engine evaluates each candidate project, computing *NPV* and profitability index.

3. **Project Selection and Constraints** (*Project Selection and Constraints*) — Projects are ranked by profitability index and the drilling queue is filled subject to rig, footage, and capital constraints.
4. **Drilling and Production Equations** (*Drilling and Production Equations*) — Once projects are selected, `drilling_equations.py` translates drilling decisions into physical well counts and production profiles.

2.9 Data Reference

This section documents Hydrocarbon Supply Model (*HSM*) input and output data, including CSV inputs under `input/`, *NEMS* restart arrays on *NEMS restart input variables reference*, and written restart outputs on *Output Variables Reference*.

> **Note**
>
> **Before this section:** Read *Model Run Sequence* for execution order and *Getting Started* for standalone vs. integrated runs. Integrated NEMS runs use `restart_HSMIN.unf` / `restart_HSMOUT.unf`; standalone runs use `restart.npz` in the run directory (see *NEMS Integration*).

2.9.1 How These Pages Relate

Suggested **reading order** for this section (see *Model Run Sequence* for the full run sequence). Each step notes **when in the run** the data applies:

1. **Inputs** (*Input Files Reference*) — **Year 1 and every year (setup).** CSV files and setup tables define districts, projects, and cost parameters before economics run.
2. **NEMS Restart Inputs** (*NEMS restart input variables reference*) — **Every call (start of setup).** Arrays read from `restart_HSMIN.unf` in integrated runs (prices from the Liquid Fuels Market Module (*LFMM*) and Natural Gas Market Module (*NGMM*), plus shared NEMS parameters).
3. **Historical and STEO Data** (*Historical Data and STEO Overrides*) — **Every-year setup, before submodules.** Actuals through the history year and *Short-Term Energy Outlook* (STEO) benchmarks written into restart variables.
4. **Outputs** (*Output Variables Reference*) — **After submodules complete.** Variables HSM writes to `restart_HSMOUT.unf` and optional debug files.

For **side-case** STEO behavior (reference vs. side scenarios), see *Historical Data and STEO Overrides* §Reference Case vs. Side Cases and *Debugging and Utilities* (STEO debug output).

Input Files Reference

How to use this page

This page is an **inventory** of CSV inputs under `input/`. **Maintainers** updating an *AEO* cycle should start at `setup.csv` and `mapping.csv`, then follow submodule setup files (`on_setup.csv`, etc.). **Developers** tracing a variable should find the filename here, then open the submodule page or *Configuration Reference — setup.csv* for switch semantics.

Integrated runs also pass prices and control flags through `restart_HSMIN.unf`—see *NEMS restart input variables reference* and *Restart variable registry*. When files load in the run, see *Model Run Sequence*.

All paths are relative to the model run directory (the folder containing `hsm.py`). Submodule-specific filenames are listed in setup files referenced from the master `setup.csv`.

The published input library on GitHub `main` contains about 175 CSV files across subdirectories (count changes with each AEO refresh).

Side cases

A **side case** is any non-reference AEO scenario. Side cases can share reference-case STEO benchmarks so all scenarios start from the same near-term production levels.

> **Note**
>
> **Side case runs** read preprocessed STEO overwrites from `side_case_owrites/steo_overwrite_data.npz` when `side_case_steo_overwrite_switch` is `TRUE` in `setup.csv`. The reference case sets this switch to `FALSE`. See *Historical Data and STEO Overrides* for generation with `preprocess_steo_overwrites.py`.

Root `input/` Files

These files are shared across all submodules and referenced directly from `setup.csv`.

Filename	Description	Used by
`setup.csv`	Master model configuration file	`module_unf.py`
`mapping.csv`	84-district × 13-region crosswalk (state, Railroad Commission of Texas (RRC) district, *LFMM* region, Petroleum Administration for Defense Districts (PADD), *NGPL* region, census)	All submodules
`discounting.csv`	Weighted average cost of capital (*WACC*) input parameters: debt ratio, fed tax rate, beta, risk premium	`module_unf.py`
`state_tax.csv`	State income tax and severance tax rates by state abbreviation	`cash_flow.py`
`depreciation_schedules.csv`	Modified Accelerated Cost Recovery System (MACRS) depreciation schedule fractions by schedule type	`cash_flow.py`
`chart_config.csv`	Chart generation configuration for standalone output	`visualizations.py`
`hist_setup.csv`	Historical data loader configuration	`history_steo.py`
`steo_setup.csv`	STEO data loader configuration	`history_steo.py`
`on_setup.csv`	Onshore submodule setup (filenames for all onshore input tables)	`onshore.py`
`off_setup.csv`	Offshore submodule setup	`offshore.py`
`ak_setup.csv`	Alaska submodule setup	`alaska.py`
`can_setup.csv`	Canada submodule setup	`canada.py`
`ngp_setup.csv`	NGP submodule setup	`ngp.py`

History Files (`input/history/` — 31 files)

Preprocessed from U.S. Energy Information Administration (*EIA*) source surveys (`HSM_Database.sas`). For years ≤ `history_year` (2024), model outputs are overwritten with these actuals. See *Historical Data and STEO Overrides* for data sources.

Production History

Filename	Description
hist_hcgasprd.csv	Historical conventional natural gas production by district
hist_hcbmprd.csv	Historical coalbed methane production by district
hist_hcadgprd.csv	Historical associated-dissolved gas production by district
hist_hcoilprd.csv	Historical conventional oil production by district
hist_hsgasprd.csv	Historical shale gas production by district
hist_htgasprd.csv	Historical tight gas production by district
hist_htadgprd.csv	Historical tight associated-dissolved gas production
hist_htoilprd.csv	Historical tight/shale oil production by district
hist_hsgwells.csv	Historical shale gas well counts by district
hist_eor_prod.csv	Historical *EOR* production by region and type
hist_eor_factors.csv	Historical EOR adjustment factors
hist_lfmm_prod.csv	Historical production in LFMM format
hist_rfqtdcrd.csv	Historical total domestic crude (including EOR)
hist_crude_lfmm.csv	Historical crude production in LFMM regional format
hist_ngplprd.csv	Historical NGPL production by district
hist_ogqshloil.csv	Historical shale oil play-level production
hist_ogqshlgas.csv	Historical shale gas play-level production
hist_ogngprd_fed.csv	Historical natural gas production — federal lands
hist_ogcoprd_fed.csv	Historical crude oil production — federal lands
hist_ognowell.csv	Historical total completed well counts

Well Count History

Filename	Description
hist_hcgwells.csv	Historical conventional gas well counts
hist_hcbmwells.csv	Historical coalbed methane well counts
hist_hcowells.csv	Historical conventional oil well counts
hist_htgwells.csv	Historical tight gas well counts
hist_htowells.csv	Historical tight oil well counts
hist_hdryholes.csv	Historical dry hole counts by district

Reserves and Prices History

Filename	Description
hist_hgasres.csv	Historical natural gas reserves by district (from EIA-23)
hist_hoilres.csv	Historical oil reserves by district (from EIA-23)
hist_dcrdwhp.csv	Historical crude oil wellhead prices
hist_EPL2.csv	Historical ethane prices
hist_EPLLBAI.csv	Historical isobutane prices
hist_EPLLBAN.csv	Historical butane prices
hist_EPLLEA.csv	Historical ethane price series
hist_EPLP.csv	Historical propane prices
hist_EPLLPA.csv	Historical pentane plus prices

STEO Files (`input/steo/` — 9 files)

Short-Term Energy Outlook (STEO) forecasts applied to the first projection year (2025 for AEO2026). See *Historical Data and STEO Overrides* for override logic.

Filename	*NEMS* Array	Description
steo_rfqtdcrd.csv	pmmout_rfqtdcrd	Total domestic crude production
steo_togqshloil.csv	ogsmout_ogqshloil	Shale oil play-level production
steo_togqshlgas.csv	ogsmout_ogqshlgas	Shale gas play-level production
steo_ogngplprd.csv	ogsmout_ogngplprd	Total NGPL production
steo_ogngplet.csv	ogsmout_ogngplet	Ethane production
steo_ogngplpr.csv	ogsmout_ogngplpr	Propane production
steo_ogngplbu.csv	ogsmout_ogngplbu	Butane production
steo_ogngplis.csv	ogsmout_ogngplis	Isobutane production
steo_ogngplpp.csv	ogsmout_ogngplpp	Pentanes plus production

Onshore Files (input/onshore/ — 27 files across 5 subdirectories)

Projects (onshore/projects/ — 5 files)

Filename	Description
on_projects_producing_oil.csv	Existing producing oil wells by district — initial volumes, decline parameters
on_projects_producing_gas.csv	Existing producing gas wells by district — initial volumes, decline parameters
on_projects_continuous.csv	Continuous (unconventional) plays — tight oil, shale oil, shale gas, *CBM*
on_projects_co2_eor.csv	CO_2 injection and steam injection EOR projects — resource size, cost, project parameters
on_projects_undiscovered.csv	Undiscovered conventional resources — discovery model parameters

Decline Rates (onshore/decline_rates/ — 4 files)

Filename	Description
on_producing_oil_decline_rates_plays.csv	Play-level oil well decline rates
on_producing_oil_decline_rates_regions.csv	Region-level oil well decline rates
on_producing_gas_decline_rates_play.csv	Play-level gas well decline rates
on_producing_gas_decline_rates_regions.csv	Region-level gas well decline rates

Costs (onshore/costs/ — 7 files)

Filename	Description
on_drill_cost_eqs.csv	Drilling cost regression coefficients by resource type (oil and gas)
on_region_avg_cost.csv	Region-average production OPEX, transport, facility capex, and SGA per well
on_basin_avg_cost.csv	USGS-province-average costs used when region costs are missing
on_ngpl_costs.csv	NGPL processing costs and revenue fractions
on_co2_pipe_seg_costs.csv	CO_2 pipeline segment transportation costs
on_legacy_co2_costs.csv	Legacy CO_2 cost parameters
on_eor_other_costs.csv	Other EOR supply costs (steam, polymers, and similar)

Constraints (`onshore/constraints/` — 3 files)

Filename	Description
`on_rig_constraint_eq.csv`	Rig constraint polynomial equations by region and price
`on_footage_constraint_eq.csv`	Footage constraint equations
`on_capital_constraint_eq.csv`	Capital constraint equations

Configuration (`onshore/configuration/` — 5 files)

Filename	Description
`on_tech_levers.csv`	Technology improvement rates by cost category and fuel type; applied for years > `technology_year`
`on_wells_per_rig.csv`	Wells drilled per rig by region and resource category
`on_discovery_order.csv`	Discovery order parameters for undiscovered conventional resources
`shale_gas_play_map.csv`	Mapping of shale gas plays to HSM districts and NEMS play indices
`tight_oil_play_map.csv`	Mapping of tight oil plays to HSM districts

Offshore Files (`input/offshore/` — ~30 files)

Undiscovered Resources

Filename	Description
`off_undiscovered_fields.csv`	USGS-based undiscovered resource base by area and field size class
`off_discovery_coefficients.csv`	Discovery rate coefficients for exploratory well model
`off_nfw_coefficients.csv`	New field wildcat well coefficients
`off_field_size_classes.csv`	Field size class distributions for undiscovered resources

Discovered/Announced Fields

Filename	Description
`off_announced_fields.csv`	Named announced fields not yet in production — development timeline
`off_delays.csv`	Development delay assumptions by area
`off_platform_delays.csv`	Platform installation delay assumptions
`off_platform_slots.csv`	Available platform slots by area and year
`off_platform_drill_per_year.csv`	Maximum wells drilled per platform per year

Producing Fields

Filename	Description
off_prod_profile.csv	Production profiles by field size class
off_operating_cost.csv	Operating costs by field category and water depth
off_platform_abandonment.csv	Abandonment cost schedules by platform type
off_platform_structure_type.csv	Platform type by water depth and field size
off_gom_projections.csv	GOA-specific production projections
off_gom_projections_hogs.csv	GOA high-oil-price projections
off_gom_projections_logs.csv	GOA low-oil-price projections
off_mapping.csv	Offshore district and region crosswalk
off_field_availability.csv	Advanced technology field availability factors
water_depth_class.csv	Water depth classification thresholds

Cost Files

Filename	Description
off_exp_cost.csv	Exploration drilling costs by water depth
off_dev_cost.csv	Development drilling costs by water depth and drill depth
off_prod_fac_cost_frac.csv	Production facility cost fraction by delay year

---

Alaska Files (input/alaska/ — 8 files)

Filename	Description
ak_projects_known.csv	Named known projects — oil resources, gas resources, peak production, decline rates, production start year, region, historical production (1990–2024)
ak_projects_undiscovered.csv	Undiscovered resource estimates — probabilistic resource base by region
ak_mapping.csv	Alaska regional parameters — tariff rates, exploration/development drilling costs, region codes (11=Offshore North, 12=Onshore North, 13=Other)
ak_facility_costs.csv	Facility costs as a function of oil resource size (step-function)
ak_opex.csv	Operating cost intercept and slope coefficients by bracket (selected from peak evaluation-year crude production)
ak_field_size_classes.csv	Field size class distributions for undiscovered resources
ak_dev_drill_schedule.csv	Development drilling schedule template (29-year horizon)
ak_exp_drill_schedule.csv	Exploration drilling schedule template

---

Canada Files (input/canada/ — 8 files)

Inputs are updated each AEO cycle by running the Canada preprocessor script (canada_preproc.py) against Canada Energy Regulator (*CER*) data. See *Canada Natural Gas Submodule* for the preprocessor workflow.

Filename	Updated per AEO?	Description
can_natgas_dc_vars.csv	Yes	Hyperbolic decline curve parameters (b, dr, ipr) by region and well type
can_benchmark_prices.csv	Yes	CER-derived benchmark Henry Hub price series
can_natgas_poly_eqs.csv	Yes	Polynomial regression coefficients: wells drilled vs. Henry Hub price
can_natgas_baseline_prod.csv	Yes	Baseline (legacy) production profile by region and year
can_natgas_no_export_prod.csv	Yes	Canada gas production not exported to U.S.
can_wells.csv	Yes	Historical and projected well count by region
can_elasticity_ad_gas.csv	No	Elasticity parameters for associated-dissolved gas
canada_setup.csv	No	Canada submodule file path configuration

NGP Files (input/ngp/ — 3 files)

Filename	Description
ngp_facility_input.csv	Natural gas processing facility database (from NETL) — facility-level characteristics
netl_natural_gas_calculations.csv	NETL-based CO_2 capture cost calculations by facility type
netl_purchased_power_cost.csv	Purchased power cost assumptions for CO_2 capture operations

Side Case Overwrite Files (input/side_case_owrites/ — 1 file)

Used when side_case_steo_overwrite_switch = TRUE. prepare_results.py loads a single preprocessed archive and overwrites selected restart variables for the first STEO year so side cases share reference-case STEO benchmarks.

Filename	Description
steo_overwrite_data.npz	Reference-case restart slices for the STEO year (side_case_steo_overwrite_year in setup.csv). Generated by running preprocess_steo_overwrites.py on the reference-case restart.npz.

Variable Mapping Files (input/variable_mapping/)

Filename	Description
master_table_selection.csv	Complete output variable list with NEMS array names, dimensions, units, and descriptions
parameter_ref_tables/*.csv	Dimension lookup tables (one file per dimension: ogdist.csv, mnl48t.csv, weltyp.csv, etc.)

Related Pages

- *NEMS restart input variables reference* — NEMS restart arrays read by HSM (restart_HSMIN.unf)
- *Output Variables Reference* — Output variable reference
- *Historical Data and STEO Overrides* — Historical and STEO data sources
- *Configuration Reference — setup.csv* — setup.csv parameter reference

NEMS restart input variables reference

In integrated mode, HSM reads restart_HSMIN.unf through the pyfiler interface (restart_unf.Restart). That file carries market prices, macroeconomic series, iteration control flags, and other arrays sized by NEMS index dimensions.

This page documents every restart quantity HSM registers with output=False in restart_unf.py (that is, arrays and integers HSM **reads** from the restart file but does **not** write back in Restart.write_results(), which updates only int_dict_out and df_dict_out). Arrays that HSM both reads and writes (for example production volumes and many ogsmout_* slots) are listed on *Output Variables Reference*.

Static CSV inputs under `input/` are described on *Input Files Reference*. Command-line arguments `-y`, `-t`, and `-c` mirror some NEMS control values; see *NEMS Integration* and *Getting Started*.

Note

Some variables use the `ogsmout_` pyfiler block name but are **inputs** to HSM because `output=False` (for example `ogsmout_ogcnpprd`). Conversely, many `ogsmout_*` arrays are HSM outputs; see *Output Variables Reference*.

Restart variable registry

NEMS passes data through binary restart files (`.unf` in integrated runs, `restart.npz` in standalone). Think of the registry in three layers:

Layer	Integrated run	Standalone run
Read from NEMS	`restart_HSMIN.unf` → arrays with `output=False` on this page	Prior `restart.npz` in the run directory
Read/write production state	Many `ogsmout_*` arrays (legacy Oil and Gas Supply Module naming)	Same pyfiler names in `restart.npz`
Write back to NEMS	`restart_HSMOUT.unf` — see *Output Variables Reference*	Updated `restart.npz` after each call

The `ogsmout_` prefix predates HSM; array names were kept for NEMS compatibility. An array listed here as an **input** may still use an `ogsmout_*` name because HSM reads it but does not write it in `write_results()`. Arrays on *Output Variables Reference* are those HSM updates each successful run.

For pyfiler setup and file layout, see *Getting Started* and *NEMS Integration*.

Dimension lookup

Index names and sizes for NEMS arrays match those used for HSM outputs. See *Dimension Lookup* on the output variables page.

NEMS control integers

Scalars from the `ncntrl` and `utils` / `other` blocks that govern the current solve and array shapes. Values are read in `restart_unf.Restart.__init__` and used throughout the model.

Name	Type	Description
ncntrl_curcalyr	int	Current calendar year in the NEMS solve
ncntrl_curitr	int	Current iteration index within the model year
ncntrl_curiyr	int	Current model-year index (reserved; not required for HSM logic)
ncntrl_lastyr	int	Last projection year in the NEMS run
parametr_fcrl	int	Final convergence reporting loop flag; when 1, last regular iteration before reporting. See *fcrl — Final Convergence Reporting Loop*
parametr_ncrl	int	NEMS convergence reporting loop flag; when 1, reporting iteration. See *ncrl — NEMS Convergence Reporting Loop*
parametr_baseyr	int	Base calendar year used to align restart indices with calendar years
parametr_gastypes	int	Count of natural gas type indices from pyfiler1.other (gastyp dimension)
parametr_mnumor	int	Count of OGSM regional indices from pyfiler1.other (includes Alaska components)
parameter_mnumor	int	Same dimension size from pyfiler1.utils (paired registration in restart_unf)
parametr_mnumcr	int	Count of census-style macro regions (mnumcr)
parametr_mnumyr	int	Count of model year slots (mnumyr)
parametr_ogdist	int	Count of oil and gas districts (ogdist)
parametr_oiltypes	int	Count of oil type indices (oiltyp)
parametr_mnl48n	int	Count of Lower 48 regional categories (mnl48n)
parametr_mnl48f	int	Count of Lower 48 offshore regions (mnl48f)
parametr_mnogcro	int	Count of crude reporting categories (nogcro)
parametr_mncrud	int	Count of LFMM crude stream categories (mncrud)
parametr_mnumpr	int	Count of PADD refinery regions (mnumpr)
parameter_gastypes	int	Gas type dimension from pyfiler1.utils (same role as parametr_gastypes; see restart_unf registration)
tcs45q_i_45q_duration	int	Number of years the §45Q credit applies
tcs45q_i_45q_lyr_ret	int	Last eligible year for retrofit credits
tcs45q_i_45q_syr	int	First eligible year for §45Q

Liquid Fuels Market Module (LFMM)

NEMS array	Units	Dimensions	Description
lfmmout_rfcrudewhp	1987$/bbl	(mnumpr, mncrud, mnumyr)	Reference crude wellhead prices by PADD refinery region and crude stream; mapped in module_unf to regional crude prices for supply economics (rest_rfcrudewhp).

Natural Gas Market Module (NGMM)

NEMS array	Units	Dimensions	Description
ngtdmrep_ogprcng	1987$/Mcf	(ogdist, mnumyr)	Natural gas supply price by oil and gas district (district-level price signal).
ngtdmrep_ogwprng	1987$/Mcf	(mnumor, mnumyr)	Natural gas wellhead price by OGSM region.
ngtdmrep_oghhprng	1987$/Mcf	mnumyr	Henry Hub benchmark price series; used for Canada and related benchmarks (rest_hh_price).
ngtdmrep_ogprdng	Bcf	(mnumor, mnumyr)	Dry natural gas production from NGMM by OGSM region and year; Alaska uses this to split regional gas production in reporting (alaska.py).

Macroeconomic Activity Module (MAM)

NEMS array	Units	Dimensions	Description
macout_mc_jpgdp	index (1987 = 1)	mnumyr	GDP price deflator; converts nominal values to constant 1987 dollars (rest_mc_jpgdp).
macout_mc_rmcorpbaa	percent	mnumyr	Corporate Baa bond yield; feeds weighted average cost of capital (rest_mc_rmcorpbaa).
macout_mc_rmtcm10y	percent	mnumyr	10-year Treasury rate; used with corporate rates in discounting (rest_mc_rmtcm10y).

CCATS and §45Q (CO_2 policy and pricing)

NEMS array	Units	Dimensions	Description
ccatsdat_co2_prc_dis_45q	from NEMS layout	(mnumcr, mnumyr)	CO_2 disposition price under §45Q; used by the NGP submodule for capture economics (rest_co2_price_45q).
ccatsdat_co2_prc_dis_ntc	from NEMS layout	(mnumcr, mnumyr)	CO_2 disposition price without §45Q credit; paired with the §45Q series above (rest_co2_price_ntc).
tcs45q_ccs_eor_45q	$/metric ton CO_2	mnumyr	§45Q credit level applicable to CO_2 EOR; drives rest_ccs_eor_45q in cash-flow logic.

International oil, NGPL, electricity, and conversion factors

NEMS array	Units	Dimensions	Description
intout_brent_price	1987$/bbl	mnumyr	International Brent price series for Canada and cross-checks (rest_brent_price).
intout_start_price	1987$/bbl	mnumyr	World oil price path specified for the scenario (rest_start_price).
pmore_plginpf	1987$/bbl	(mnumor, mnumyr)	Natural gas plant liquids price inputs (rest_plginpf).
mpblk_pelin	1987$/MWh	(mnumcr, mnumyr)	Industrial purchased electricity price by census region; used by the NGP submodule (rest_pelin).
convfact_cflgq	factor	mnumyr	Conversion factor for liquid petroleum gas / NGPL accounting (rest_cflgq).
emission_extrarisk	fraction	mnumyr	Additional risk adder from the emission block (rest_extrarisk).

Canada (read-only ogsmout slot)

NEMS array	Units	Dimensions	Description
ogsmout_ogcnpprd	from NEMS	(numcan, mnumyr)	Canada natural gas supply prices by East or West Canada; read for the Canada submodule (canada.py) and **not** written by write_results (output=False in restart_unf). Same economic concept as the "Canada wellhead price" row in variable_mapping/master_table_selection.csv.

Related pages

- *Output Variables Reference* — Arrays HSM writes to `restart_HSMOUT.unf`
- *Input Files Reference* — CSV input library under `input/`
- *NEMS Integration* — Module data flows and restart protocol

Output Variables Reference

How to use this page

HSM writes results to *NEMS* through `restart_HSMOUT.unf` (integrated runs) or `restart.npz` (standalone). Use this page to:

- **Report or validate production** — start with the production tables in §NEMS Output Variables; match array names to NEMS reporting docs.
- **Trace a restart handoff** — see *Restart variable registry* on *NEMS restart input variables reference* for read vs. write and `ogsmout_*` naming.
- **Debug a run** — enable switches on *Debugging and Utilities*; CSV debug files are separate from the 62 restart variables below.

Units in tables below: **MMbbl/yr** = million barrels per year; **Mbbl/yr** = thousand barrels per year; **Mbbl/day** = thousand barrels per day; **BCF/yr** = billion cubic feet per year. District-level rows use the 84-district list in `mapping.csv` (see *NEMS Integration*).

NEMS Output Variables

All 62 variables below are written to `restart_HSMOUT.unf` by `restart_unf.RestartUnf.write_results()`. The variable list is defined in `variable_mapping/master_table_selection.csv`.

Dimension Lookup

The index dimension names used below refer to the following NEMS array dimensions. For how districts map to regions, see `mapping.csv`.

Dimension	Size	Description
mnumyr	61	Model year index dimension used by NEMS arrays
ogdist	84	Oil and gas state and offshore districts
mnl48t	11	Lower 48 regions plus total (excluding Alaska)
mnl48a	10	Lower 48 supply regions excluding the total aggregate (East through Atlantic Offshore)
mnl48f	3	Lower 48 offshore regions (Gulf, Pacific, and Atlantic Offshore)
mnl48n	7	Lower 48 regional categories (East through West Coast)
mnumor	14	Regional index including Alaska components and total production
mnumor_2	16	Extended regional index including Alaska aggregate, onshore/offshore, and total production
mnumcr	11	Census-style regions plus California and U.S. total
mnumpr	10	PADD refinery regions plus maritime Canada/Caribbean and total
mnalas	3	Alaska Offshore North, Alaska Onshore North, and Alaska Other
numcan	2	East Canada and West Canada
mncrud	11	Crude stream categories (Brent through condensate)
mnlnp1	8	Regional index including total
eor_types	13	Thermal-Steam, Gas-CO_2, total, and placeholder entries
weltyp	7	Well categories (conventional oil/gas, tight oil/gas, shale gas, *CBM*, dry holes)
gastyp	5	Gas categories (all other, tight, shale, CBM, total)
oiltyp	5	Oil categories (all other, tight, CO_2 injection EOR, steam injection EOR, total)
nogcro	6	Crude reporting categories (EOR, onshore, offshore, Alaska, ANWR, total U.S.)
nogcat	12	Natural gas reporting categories (including shale, CBM, tight, conventional, offshore, Alaska, and total)
water_depth_class	2	Shallow and deep water depth classes
offshore_category	6	State and federal offshore groupings (Atlantic, Pacific, Gulf of America, and Alaska)
prodtyp	2	Crude oil and natural gas
ogprdugr_3	3	Tight gas, shale gas, and coalbed methane
ogregprd_7	7	Regional production categories for crude and natural gas reporting
soplay_ogqshloil	15	Select shale oil play slots (includes named plays and placeholders)
soplay_ogqshlgas	15	Select shale gas play slots (includes named plays and placeholders)

Crude Oil Production

NEMS Array	Units	Dimensions	Description
ogsmout_ogcoprd	MMbbl/yr	(mnl48t, mnumyr)	Crude oil production by Lower 48 supply region
ogsmout_ogoilprd	MMbbl/yr	(ogdist, oiltyp, mnumyr)	Crude oil production by district and oil type
ogsmout_ogcrdprd	Mbbl/yr	(mnumor, mncrud, mnumyr)	Crude oil production by *LFMM* region and crude type
ogsmout_ogqshloil	Mbbl/yr	(soplay_ogqshloil, mnumyr)	Crude oil production from select shale oil plays (15 plays)
ogsmout_ogeorprd	Mbbl/yr	(mnlnp1, eor_types, mnumyr)	*EOR* production from CO_2 injection and steam injection projects
ogsmout_ogcoprd_fed	MMbbl/yr	(mnumor, mnumyr)	Crude oil production — federal lands
ogsmout_ogcoprd_nonfed	MMbbl/yr	(mnumor, mnumyr)	Crude oil production — non-federal lands
ogsmout_ogcoprdgom	Mbbl/day	(water_depth_class, mnumyr)	GOA crude oil production by water depth (shallow/deep)
ogsmout_ogcruderef	Mbbl/day	(mnumpr, mncrud, mnumyr)	Crude oil production by refinery region and crude type
ogsmout_ogqcrrep	MMbbl	(nogcro, mnumyr)	Crude oil production by oil category (reporting)
pmmout_rfqdcrd	MMbbl/yr	(mnumor_2, mnumyr)	Total domestic crude production (LFMM format)
pmmout_rfqtdcrd	MMbbl/yr	(mnumor_2, mnumyr)	Total domestic crude including EOR (LFMM format)

For CO_2 injection, recycle, and purchased-volumes arrays used by EOR (ogsmout_ogco2inj, ogsmout_ogco2rec, and ogsmout_ogco245q), see the **CO_2 and NGP** section below.

Natural Gas Production

NEMS Array	Units	Dimensions	Description
ogsmout_ogngprd	BCF/yr	(mnl48t, mnumyr)	Natural gas production by Lower 48 supply region
ogsmout_ogdngprd	BCF/yr	(ogdist, gastyp, mnumyr)	Dry natural gas production by district and gas type
ogsmout_ogenagprd	BCF/yr	(ogdist, gastyp, mnumyr)	**Expected** non-associated natural gas production (updated every iteration; read by *NGMM*)
ogsmout_ogrnagprd	BCF/yr	(ogdist, gastyp, mnumyr)	**Realized** non-associated natural gas production (after *ncrl* adjustment)
ogsmout_ogadgprd	BCF/yr	(ogdist, oiltyp, mnumyr)	Associated-dissolved natural gas production by district
ogsmout_ogprdad	BCF/yr	(mnl48a, mnumyr)	Associated-dissolved gas production (L48 aggregate)
ogsmout_ogprdadof	BCF/yr	(mnl48f, mnumyr)	Offshore associated-dissolved gas production
ogsmout_ogqshlgas	TCF	(soplay_ogqshlgas, mnumyr)	Natural gas production from select shale gas plays (15 plays)
ogsmout_ogprdugr	BCF/yr	(mnl48n, ogprdugr_3, mnumyr)	Marketed unconventional Lower 48 natural gas production
ogsmout_ogqngrep	BCF/yr	(nogcat, mnumyr)	Natural gas production by gas category (reporting)
ogsmout_ogngprd_fed	TCF	(mnumor, mnumyr)	Natural gas production — federal lands
ogsmout_ogngprd_nonfed	TCF	(mnumor, mnumyr)	Natural gas production — non-federal lands
ogsmout_ogngprdgom	BCF/yr	(water_depth_class, mnumyr)	GOA natural gas production by water depth (shallow/deep)
ogsmout_ogprdoff	Mbbl/yr	(offshore_category, prodtyp, mnumyr)	Lower 48 offshore production by federal/state category
ogsmout_ogregprd	MMbbl/TCF	(mnl48n, ogregprd_7, mnumyr)	Regional crude oil and natural gas production by production type

Alaska and Canada

NEMS Array	Units	Dimensions	Description
ogsmout_ogprcoak	BCF/yr	(mnalas, mnumyr)	Alaska crude production by region (regions 11–13); populated from Alaska crude volumes (legacy unit label in NEMS metadata)
ogsmout_ogngla k	Mbbl/day	mnumyr	NGL production from Alaska
ogsmout_cnenagprd	MMcf	(numcan, mnumyr)	Canada non-associated natural gas production (East/West)
ogsmout_cnadgprd	MMcf	(numcan, mnumyr)	Canada associated-dissolved natural gas production
ogsmout_ogcnpprd	—	(numcan, mnumyr)	Canada natural gas supply prices

Natural Gas Plant Liquids (NGPL)

NEMS Array	Units	Dimensions	Description
ogsmout_ogngplprd	Mbbl/yr	(ogdist, mnumyr)	Total NGPL production by district
ogsmout_ogngplet	Mbbl/yr	(ogdist, mnumyr)	Ethane production from natural gas processing plants
ogsmout_ogngplpr	Mbbl/yr	(ogdist, mnumyr)	Propane production
ogsmout_ogngplbu	Mbbl/yr	(ogdist, mnumyr)	Butane production
ogsmout_ogngplis	Mbbl/yr	(ogdist, mnumyr)	Isobutane production
ogsmout_ogngplpp	Mbbl/yr	(ogdist, mnumyr)	Pentanes plus production

Prices

NEMS Array	Units	Dimensions	Description
pmmout_dcrdwhp	1987$/bbl	(mnumor, mnumyr)	Domestic crude wellhead price by LFMM region
ogsmout_ogcowhp	1987$/bbl	(mnl48t, mnumyr)	Crude wellhead price by oil category
ogsmout_ogngwhp	1987$/Mcf	(mnl48t, mnumyr)	Natural gas supply prices by gas category
ogsmout_ogpcrwhp	1987$/bbl	mnumyr	Average crude wellhead price
ogsmout_ogpngwhp	1987$/Mcf	mnumyr	Average natural gas supply price

---

Wells, Drilling, and Activity

NEMS Array	Units	Dimensions	Description
ogsmout_ognowell	count	mnumyr	Total completed wells across all districts, regions, and well types
ogsmout_ogogwells	count	(ogdist, weltyp, mnumyr)	Wells by district and type
ogsmout_ogwellsl48	count	(mnl48t, weltyp, mnumyr)	Total well count (successful + dry holes) by L48 region
ogsmout_ogsrl48	fraction	(mnl48t, weltyp, mnumyr)	Lower 48 drilling success rate (successful / total)
ogsmout_ogjobs	count	mnumyr	Number of jobs in oil and gas supply sector
ogsmout_ogtechon	—	(3, 6, mnumyr)	Technology factors by cost category, fuel, and year

---

CO_2 and NGP

NEMS Array	Units	Dimensions	Description
ccatsdat_dem_eor	metric tonnes	(200, mnumyr)	CO_2 demand from EOR volumes (project-level, 200 slots)
ccatsdat_cst_eor	1987$/tonne	(200, mnumyr)	CO_2 price EOR projects will offer (project-level)
ogsmout_ngpco2em	metric tons CO_2	(mnumcr, mnumyr)	CO_2 emitted by natural gas processing plants
ogsmout_ogco2inj	MMcf	(8, 13, mnumyr)	CO_2 injected at EOR projects
ogsmout_ogco2rec	MMcf	(8, 13, mnumyr)	CO_2 recycled at EOR projects
ogsmout_ogco245q	MMcf	(8, 13, mnumyr)	CO_2 purchased (§45Q credit)

Miscellaneous and Control

NEMS Array	Units	Dimensions	Description
qmore_qngpin	TBtu	(mnumcr, mnumyr)	Natural gas processing plant electricity consumption
ogsmout_ogcrdheat	BTU/bbl	(mncrud, mnumyr)	Heat rates by crude oil type
ogsmout_ns_start	year	(8, 200)	Year an EOR project was selected (project-level)
ogsmout_play_map	—	200	Map of geological formations (plays) — used to link to *CCATS*
ogsmout_ogprcexp	—	mnumyr	Price adjustment factor for expectations
ogsmout_oggrowfac	—	mnumyr	Factor reflecting expected future consumption growth

---

Debug and Standalone Output Files

Always Written (Standalone Mode)

File	Written by	Description
restart_HSMOUT.unf	restart_unf.py	Binary restart output written by HSM for downstream model use

Written When `debug_switch = TRUE`

File	Written by	Description
`debug/steo_debug/*.csv`	`history_steo.py`	STEO adjustment factor tables
`cashflow_debug/ak_cash_flow_props.csv`	`cash_flow.CashFlow.to_csv()` (Alaska only)	Project-level cash flow properties (prices, *NPV*, profitability)
`cashflow_debug/ak_cash_flow_costs.csv`	`cash_flow.CashFlow.to_csv()` (Alaska only)	Time-series arrays for all 35+ cost and revenue components

Written When `hsm_var_debug_switch = TRUE`

File	Description
`intermediate_tables/hsm_on_projects.csv`	Onshore project DataFrame mid-run
`intermediate_tables/hsm_off_fields.csv`	Offshore field DataFrame mid-run
Other `intermediate_tables/*.csv` files	Intermediate variable snapshots for each submodule

Pickle State Files

File	Created at	Description
`fcrl_var.pkl`	`fcrl=1` iteration	Intermediate variable state for recovery on next regular iteration
`ncrl_var.pkl`	`ncrl=1` iteration	Intermediate variable state for recovery at start of next year

---

Related Pages

- *Input Files Reference* — Input file reference
- *NEMS restart input variables reference* — NEMS restart inputs read but not written by HSM
- *NEMS Integration* — Restart file protocol and data flows
- *Debugging and Utilities* — Debug output tools

Historical Data and STEO Overrides

HSM benchmarks its projections against two types of observed data: historical production and well count actuals (from U.S. Energy Information Administration (EIA) surveys) and *Short-Term Energy Outlook* (STEO) forecasts for the near-term projection years. This page describes both data sources and the override logic.

Read *Model Run Sequence* §4 (every-year setup) for when these loads run relative to submodule execution. For side-case STEO sharing across scenarios, see §Reference Case vs. Side Cases below and *Debugging and Utilities* (STEO debug CSVs).

Historical Data

Override Logic

For every year ≤ `history_year` (2024 for AEO2026), HSM **overwrites** model-computed production and well counts with actual EIA data. This ensures that the model's starting conditions match reality before entering the projection period.

The overwrite runs during **every-year setup**, before submodule execution (see *Model Run Sequence* §4, steps 2–3). `module_unf.py` calls `_load_history_and_steo()` after the restart file is read and before submodules run. `history_steo.HistorySteo` loads all 31 files in `input/history/` and writes values into **restart variables** (for example `ogsmout_*` arrays) that submodules and NEMS use on that iteration—not into post-execution submodule outputs.

EIA Source Data

The historical files in `input/history/` are preprocessed from the following EIA surveys using `HSM_Database.sas` (a separate SAS program):

Survey	Full name	Data type	HSM files
Well-level production (commercial)	Commercial well-level production database (monthly well-level crude and gas production)	Monthly well-level crude and gas production by district	`hist_hcoilprd.csv`, `hist_hcgasprd.csv`, `hist_hcadgprd.csv`, `hist_hcbmprd.csv`, `hist_hsgasprd.csv`, `hist_htgasprd.csv`, `hist_htoilprd.csv`, `hist_htadgprd.csv`
EIA-23	Annual Survey of Domestic Oil and Gas Reserves	Oil and gas reserves by state and category	`hist_hoilres.csv`, `hist_hgasres.csv`
PSA	Petroleum Supply Annual	Crude oil production by state	`hist_rfqtdcrd.csv`, `hist_crude_lfmm.csv`, `hist_lfmm_prod.csv`
NGA	Natural Gas Annual	Natural gas production by state	`hist_hcgasprd.csv` (cross-check)
EIA Product Price	EIA petroleum product price surveys	NGPL component prices	`hist_EPL2.csv`, `hist_EPLLBAI.csv`, `hist_EPLLBAN.csv`, `hist_EPLLEA.csv`, `hist_EPLP.csv`, `hist_EPLLPA.csv`
EIA Well Completion	EIA completion data	Well counts by type and district	`hist_hcgwells.csv`, `hist_hcowells.csv`, `hist_hdryholes.csv`, `hist_hsgwells.csv`, `hist_htgwells.csv`, `hist_htowells.csv`, `hist_hcbmwells.csv`
EOR data	EIA-23 EOR supplement	EOR production by method	`hist_eor_prod.csv`, `hist_eor_factors.csv`
ONRR/BLM (federal lands)	ONRR and BLM federal land production records	Production on federal vs. non-federal lands	`hist_ogcoprd_fed.csv`, `hist_ogngprd_fed.csv`
Play-level	EIA tight oil/shale reporting	Play-level shale production	`hist_ogqshloil.csv`, `hist_ogqshlgas.csv`

Preprocessing

Historical files are **not** generated by HSM itself. They are produced by:

1. Running `HSM_Database.sas` (a SAS program in a separate repository) against the source surveys
2. Post-processing the SAS outputs into the CSV format expected by HSM
3. Copying the CSV files to `input/history/` before the model build

This preprocessing is done once per AEO cycle (typically in summer/fall before the model build season) and the resulting files are committed to version control.

STEO Override Logic

What STEO Is

The *Short-Term Energy Outlook* (STEO) is EIA's near-term energy forecast, updated monthly. For AEO purposes, STEO values are used to benchmark the first projection year.

STEO Year

The STEO applies to **the first projection year only**:

- AEO2026: STEO year = **2025** (set in `setup.csv: side_case_steo_overwrite_year = 2025`)
- AEO2025: STEO year = 2024
- In general: STEO year = `aeo_year - 1`

Reference Case vs. Side Cases

Important

The STEO override behaves differently for the reference case vs. side cases:

- **Reference case:** `side_case_steo_overwrite_switch = FALSE`. The reference case applies its own STEO calibration through the regular history/STEO files.
- **Side cases:** `side_case_steo_overwrite_switch = TRUE`. Side cases use reference-case preprocessed STEO values from `side_case_owrites/steo_overwrite_data.npz` to ensure all scenarios share the same STEO-benchmarked starting conditions.

STEO Source Files

The 9 STEO CSV files in `input/steo/` are sourced from EIA's published STEO tables:

File	Variable	Description
steo_rfqtdcrd.csv	pmmout_rfqtdcrd	Total domestic crude production
steo_togqshloil.csv	ogsmout_ogqshloil	Shale oil play-level production
steo_togqshlgas.csv	ogsmout_ogqshlgas	Shale gas play-level production
steo_ogngplprd.csv	ogsmout_ogngplprd	Total NGPL production
steo_ogngplet.csv	ogsmout_ogngplet	Ethane production
steo_ogngplpr.csv	ogsmout_ogngplpr	Propane production
steo_ogngplbu.csv	ogsmout_ogngplbu	Butane production
steo_ogngplis.csv	ogsmout_ogngplis	Isobutane production
steo_ogngplpp.csv	ogsmout_ogngplpp	Pentanes plus production

STEO Adjustment Factors

`history_steo.py` computes adjustment factors by comparing STEO values to the model's baseline for the STEO year. These factors scale the model's projection for the STEO year to match STEO. When `debug_switch = TRUE`, the adjustment factors are written to `debug/steo_debug/` as CSV files.

Side Case STEO Preprocessing

The `side_case_owrites/steo_overwrite_data.npz` file is generated by running `preprocess_steo_overwrites.py` against the reference case STEO output. This must be re-run each time the reference case STEO year values change. The parameter `side_case_steo_overwrite_year` must match the year used when running this script.

`history_steo.py` Responsibilities

`history_steo.HistorySteo` handles both historical and STEO data:

1. Loads all 31 historical CSV files from `input/history/`
2. For years ≤ `history_year`: overwrites model output arrays with historical actuals
3. For the STEO year: computes adjustment factors and applies STEO benchmarks
4. For side case runs: loads `steo_overwrite_data.npz` and applies reference-case STEO
5. Manages timing of which data applies in each year

Related Pages

- *Input Files Reference* — Complete input file reference including history/ and steo/ directories
- *Configuration Reference — setup.csv* — `side_case_steo_overwrite_switch` and related parameters
- *Debugging and Utilities* — STEO debug output files

2.10 Developer Guide

Resources for developers working on the HSM codebase.

2.10.1 How These Pages Relate

Complete *Getting Started* first (prerequisites, layout, and build). Then use this section in this order:

1. **Running HSM** (*Running HSM — execution details*) — Standalone and integrated NEMS execution, command lines, and environment setup.
2. **Debugging** (*Debugging and Utilities*) — Debug switches, intermediate outputs, and common failure modes after you can run the model.

Developers changing **inputs, history/STEO files, or restart mappings** should also use *Data Reference* (reading order and file inventories).

Running HSM — execution details

> **Note**
>
> **Prerequisites:** Complete *Getting Started* first (install, directory layout, and your first standalone or integrated command). This page adds timing, logging, pickle inspection, and iteration-loop detail.

This page covers execution mechanics after you can run HSM: timing, logging, intermediate state inspection, and differences between run modes. For debug switches and STEO tables, see *Debugging and Utilities*.

Standalone Mode — Full Year Loop

In standalone mode, `hsm.py` runs its own year loop from `history_year + 1` through `DEBUG_STOP_YEAR`:

```
# hsm.py — simplified
for year in range(history_year + 1, DEBUG_STOP_YEAR + 1):
    for iteration in [1, fcrl_iteration, ncrl_iteration]:
        run_hsm(year=year, iteration=iteration, cycle=1)
```

Key standalone-mode behaviors:

- SCEDES file provides scenario parameters (not received from NEMS); see *SCEDES File (Standalone Runs)*
- Each year runs at least three calls: iteration 1, *fcrl — Final Convergence Reporting Loop* (`fcrl=1`), *ncrl — NEMS Convergence Reporting Loop* (`ncrl=1`)
- `restart_HSMOUT.unf` is written on each model call
- Charts are generated if `charts_switch = TRUE`

Year Loop Configuration

Parameter	Source	Description
Start year	`setup.csv: history_year + 1`	First projection year (2025 for AEO2026)
End year	`hsm.py: DEBUG_STOP_YEAR = 2050`	Last projection year (set per AEO cycle)
Scenario	`setup.csv: standalone_scedes_file`	SCEDES file for scenario keys

Integrated (NEMS) Mode

In integrated mode, NEMS controls the year loop. HSM runs as a subprocess called once per NEMS iteration:

```
python hsm.py -y 2026 -t 1 -c 1
python hsm.py -y 2026 -t 2 -c 1
...
python hsm.py -y 2026 -t N -c 1   (fcrl=1)
python hsm.py -y 2026 -t N+1 -c 1 (ncrl=1)
python hsm.py -y 2027 -t 1 -c 1
...
```

NEMS passes `fcrl` and `ncrl` flags through `restart_HSMIN.unf`. HSM does not control iteration count — it responds to whatever iteration flags NEMS sends.

Timing and Logging

HSM writes to `hsm_debug.log` in the run directory. Log entries include:

- Year, iteration, cycle at each call
- Timing for each submodule
- Key decision points (skip vs. run, constraint binding)

Timing summaries are available through `module_unf.get_timing_summary()`. The `reset_timing()` function resets accumulated timers for performance benchmarking.

To reduce log verbosity, set `suppress_warnings_switch = TRUE` in `setup.csv`.

Inspecting Intermediate State

pickle files

The `fcrl_var.pkl` and `ncrl_var.pkl` files store the complete model state (all submodule DataFrames) at the end of each iteration type. Use `intermediate_var_pickle.py` to read them:

```
import intermediate_var_pickle as ivp

# Load fcrl state from a run directory
state = ivp.load_fcrl_vars(run_dir='path/to/hsm/')
# Inspect project DataFrame
print(state['on_projects_continuous'])
```

This is particularly useful for diagnosing why a specific project was or was not drilled.

hsm_var_debug_switch

Setting `hsm_var_debug_switch = TRUE` in `setup.csv` writes CSV snapshots of key DataFrames to `intermediate_tables/`:

- `hsm_on_projects.csv` — Onshore project table mid-run (includes NPV, PI, wells)
- `hsm_off_fields.csv` — Offshore field table
- Other submodule-specific tables

These are useful for verifying that prices, costs, and economic rankings are correct without writing custom debug code.

---

run_every_itr_switch

When `run_every_itr_switch = FALSE` in `setup.csv`, HSM only runs on:

- Iteration 1 of each year
- `fcrl = 1` iteration
- `ncrl = 1` iteration

For all other iterations, HSM writes the prior restart file unchanged. This is approximately 3–5× faster than `run_every_itr_switch = TRUE` and suitable for:

- Initial debugging of a code change
- Performance testing
- Verifying that iteration 1 output is reasonable

The reference case uses `run_every_itr_switch = TRUE` for production runs.

Related Pages

- *Getting Started* — Basic run setup and prerequisites
- *Configuration Reference — setup.csv* — `setup.csv` parameter reference
- *Debugging and Utilities* — Debug tools and output file reference

Debugging and Utilities

This page documents debug switches, diagnostic CSV output, and STEO debug tables. For year loops, logging, and pickle mechanics, see *Running HSM — execution details*.

> **Note**
>
> **Minimum repro path:** Run standalone or integrated once with default switches, then set `debug_switch` or `hsm_var_debug_switch` in `setup.csv` and rerun a single year if possible. Confirm `input/` paths from *Getting Started*.

Debug Switches (in `setup.csv`)

Switch	Effect when TRUE
`debug_switch`	Writes debug output files to `debug/`; STEO adjustment factors to `debug/steo_debug/`
`hsm_var_debug_switch`	Writes CSV copies of project/field DataFrames to `intermediate_tables/` at each call
`charts_switch`	Generates output charts after standalone runs (writes to `output/charts/`)
`suppress_warnings_switch`	Suppresses Python `warnings.warn()` messages (use cautiously)

Cash Flow Debug Output (`debug_switch`)

When `debug_switch = TRUE`, the Alaska submodule calls `CashFlow.to_csv()` after the economic evaluation. This writes two files under `cashflow_debug/`:

File	Contents
`ak_cash_flow_props.csv`	Project-level properties: prices, royalty rates, NPV, profitability index (PI)
`ak_cash_flow_costs.csv`	Time-series arrays for all ~35 cost/revenue components per project

Files append across iterations with columns `year`, `iteration`, `cycle`. Onshore and offshore cash flow CSV export is commented out in code. To examine a specific Alaska project in a specific year:

```
import pandas as pd
props = pd.read_csv('cashflow_debug/ak_cash_flow_props.csv')
row = props[(props.year == 2026) & (props.index == project_id)]
```

Intermediate Variable Tables (`hsm_var_debug_switch`)

Setting `hsm_var_debug_switch = TRUE` writes CSV snapshots of key DataFrames to `intermediate_tables/` on each HSM call:

File	Description
`hsm_on_projects.csv`	Onshore project table — includes all projects, NPV, PI, wells drilled
`hsm_off_fields.csv`	Offshore field table — lifecycle status, production, economics
`hsm_ak_projects.csv`	Alaska project table
Additional files	Other submodule-specific tables

These files are overwritten each call, so they always reflect the current iteration state. Useful for verifying economic ranking and constraint application.

`visualizations.py` — Chart Generation

After a standalone run completes, `visualizations.py` generates diagnostic charts from model output arrays when `charts_switch = TRUE`.

Charts are configured in `input/chart_config.csv`, which specifies:

- Which output variables to plot
- Chart type (line, bar, stacked bar)
- Geographic aggregation level

Charts are written to `output/charts/` as PNG or SVG files.

To generate charts manually after a run:

```
from visualizations import generate_charts
generate_charts(npz_path='path/to/output_snapshot.npz', config_path='input/chart_
↪config.csv')
```

`intermediate_var_pickle.py` — State Inspection

The pickle files `fcrl_var.pkl` and `ncrl_var.pkl` store all submodule DataFrames at the end of each iteration type. To inspect them:

```
import intermediate_var_pickle as ivp
import pickle

# Load pickle file
with open('fcrl_var.pkl', 'rb') as f:
    state = pickle.load(f)

# List available keys
print(list(state.keys()))

# Inspect a specific submodule table
projects = state['on_projects_continuous']
print(projects[projects.region == 'gulf_coast'].sort_values('profitability',␣
↪ascending=False))
```

This is useful for diagnosing why a specific play or project has unexpected production in a given year.

Logging

HSM uses Python's `logging` module. The log file `hsm_debug.log` is written to the run directory. Log levels:

- `DEBUG` — detailed per-step outputs within each submodule
- `INFO` — year/iteration entry, timing summaries, constraint binding notices

- `WARNING` — data quality issues, missing values, unexpected conditions

To increase log verbosity, set the logging level to `DEBUG` in `hsm.py`.

STEO Debug Output

When `debug_switch = TRUE`, `history_steo.py` writes STEO adjustment factor tables to `debug/steo_debug/`. These show:

- The raw STEO values
- The model's unadjusted values for the STEO year
- The adjustment factors applied

Useful for verifying STEO benchmarking is working correctly after updating STEO input files.

Related Pages

- *Running HSM — execution details* — Run modes and timing
- *Historical Data and STEO Overrides* — Historical and STEO data sources
- *visualizations — Chart Generation* — `visualizations` API reference

2.11 API Reference

Auto-generated reference for published HSM Python modules. Documentation is generated from NumPy-style module and function docstrings.

Mathematical notation for DCF, WACC, constraints, and drilling equations lives on the *Methodology* pages (not here). Key modules: *Economic Framework — Discounted Cash Flow Model* (`cash_flow`), *Price and Discount Rate Calculations* (`module_unf`), *Drilling and Production Equations* (`drilling_equations`).

> **Note**
>
> Module pages for large files (`onshore.py`, `offshore.py`, `history_steo.py`, `prepare_results.py`) are most useful when all public methods have NumPy-style docstrings. Where docstrings are sparse, the module-level docstring provides an overview.

2.11.1 How to use this section

- **Behavior and economics** — Read *Submodules* and methodology pages first; submodule narratives explain what the code does.
- **Signatures and class layout** — Use this API section and *Architecture* for module dependencies and entry points.
- **Run sequence** — *Model Run Sequence* describes when orchestration and I/O modules run relative to geographic submodules.

2.11.2 Orchestration

hsm — Entry Point

hsm.py is the command-line entry point for HSM. It parses arguments, sets up logging, and calls run_hsm(). In integrated NEMS runs, NEMS calls this script directly.

This reference page lists module symbols and docstrings. Main execution function for HSM.

Summary

hsm.py is the main execution function for HSM. It is called by NEMS via a .bat file, performs basic model setup functions, and then calls module_unf.py to run the main NEMS processes. The file operates as follows:

1. hsm.py is called by a .bat file from NEMS or run directly from the IDE.
2. File directory variables and model timer are instantiated.
3. *get_args* is called, pulling run information from the NEMS command line (i.e. iteration and model year).
4. *run_hsm* is called, kicking off the main HSM run operations:
    a. Module in module_unf.py is declared, main directories are declared, and main setup file is identified.
    b. hsm.setup in module_unf.py is called to perform year 1 model setup.
    c. HSM determines whether to run standalone vs. integrated based on run location.
    d. Main HSM functions are run via **module_unf.py** from history year through final AEO year.
    e. After main HSM model functions run, results are prepared and reported to the restart file.

Input files

None

HSM functions and class methods

- get_args - Parses the arguments passed in via command line of NEMS
- run_hsm - Executes main HSM functions

Output debug files

None

HSM class methods

hsm.**get_args**(*in_args=None*)

parses the arguments passed in via command line of NEMS. NOTE: NEMS calls Pyfiler with trailing options for PyFiler to parse through for the input file, output file, and direction of restart to hdf formatting.

Parameters
in_args (list) – "In arguments" variable first set equal to NONE, then creates a list of system arguments passed from NEMS via the command line

Returns
arguments – Command line arguments saved into namespace variable

Return type
Namespace

hsm.**run_hsm**(*year*, *iteration*, *pyfiler1*, *cycle*, *scedes*)

Executes main HSM functions.

Return type
None

`module_unf` — Core Orchestrator

`module_unf.py` implements the `Module` class that coordinates HSM setup, annual run flow, submodule execution, restart-file interaction, and final results preparation.

This reference page lists module symbols and docstrings. For run sequence and module relationships, see *Model Run Sequence* and *Architecture*. **Mathematical notation** for WACC (r), regional prices, and 1987-dollar deflation is in *Price and Discount Rate Calculations*. Parent module of HSM where all main processes are called. Parent class for all HSM submodules.

Summary

The module class performs all key model setup functions and runs all setup sub-classes (**restart.py**, **intermediate_var_pickle.py**, **history_steo.py**). It then runs each of the HSM submodules (**onshore.py**, **offshore.py**, **alaska.py**, and **canada.py**), aggregates model results, and performs reporting functions. The module operates as follows:

1. The module is initialized in the *__init__* method after being called by **hsm.py**, with class dataframes, variables, flags, and filepaths being declared here.
2. The *setup* method is called each model year, which in turn calls several additional setup functions. The *setup* method operates as follows:
    a. Model filepaths are initialized, and the main **setup.txt** table containing key model attributes (i.e. model switches, the HSM history year, etc.) is read in.
    b. Other global tables are read in (i.e. mapping and discount tables).
    c. *self.restart.setup* is called from child class in **Restart.py**, reading in the the NEMS Restart File and assigning all the restart variables to local HSM variables.
    d. *load_parameters* and *process_restart_variables* are called, loading in NEMS parameters (i.e. model year, iteration number, etc.) and creating an additiona layer of local variables for use by the HSM submodules to avoid data overwrites.
    e. The scedes "Scenario flag" is read in. The "Scenario flag" is used to determine if NEMS is running a side case, and if so, what model adjustments need to be made in HSM as part of this side case.
    f. Unlike OGSM, HSM is written in Python, meaning that its iterative, local variables cannot be stored in memory. Instead, these intermediate variables are stored in ".pkl" files each year and read in each model year. As a first step, the intermediate variable folder is cleaned out. In model year 1 the .pkl output directory is cleaned out entirely, in other model years "model year - 2" .pkl files are deleted. Then the relevant .pkl intermediate variables are read in from **intermediate_var_pickle.py**.
    g. Load historical data and apply to relevant restart variables.
    f. Calculate NGPL ratios for STEO calculations.
    h. Load in STEO overwrite data and apply to relevant restart variables.
    i. Setup the **Cashflow** class to be used by all HSM submodules.
    g. Setup the **Onshore**, **Offshore**, **Alaska**, and **Canada** submodules by calling their respective *setup* class methods.
3. The *run* method is called each model year, which operates as follows:
    a. Model year and iteration are declared.
    b. Model year discount rate is calculated in *calculate_discount_rate*.
    c. Model year prices are calculated in *calculate_prices* based on LFMM and NGMM regional wellhead prices.
    d. Run the **Onshore**, **Offshore**, **Alaska**, and **Canada** submodules by calling their respective *run* class methods.
4. Once all of the submodules have run, the *prepare_results* method is called, which operates as follows:

a. Run the **Onshore**, **Offshore**, **Alaska**, and **Canada** *report results* methods to write initial submodule results to the restart variables.

b. Run the *aggregate_results_across_submodules* method to combine production volumes across submodules for cumulative variables.

c. In the first STEO year (AEO year - 1), run *set_expected_production_to_steo*, which overwrites production volumes with STEO production values.

d. Run *adjust_production* to respond to actual natural gas demand volumes provided by NGMM.

f. Call *write_pkl_variables* from **intermediate_var_pickle.py** to write intermediate variables out for the next model year.

Input files

- setup.csv - setup file with key model inputs (model flags, model history year, etc.)
- mapping.csv - setup file with HSM regional mapping (mapping HSM districts to regions to LFMM regions, etc.)
- discount_tables.csv - setup file with key discount rate assumptions

Model functions and class methods

- __init__ - Constructor to initialize Module Class
- setup - HSM Setup function
- load_parameters - Loads in HSM Parameters
- process_restart_variables - Creates global variables from restart file for use in the submodules
- run - Method to run main HSM model functions
- calculate_discount_rate - "Calculates discount rate to be used in discounted cash flows (**CashFlow.py**)
- calculate_prices - Aggregates and shares out crude type and natural gas prices to regions and districts
- prepare_results - Prepare results for writing to the restart file
- aggregate_results_across_submodules - Calculate sum values in restart variable tables that are aggregated across submodules
- adjust_production - Adjust Restart variables so that they reflect final model year realized production

Output debug files

hsm_steo_nagas_adjustment_factor.csv - STEO adjustment factors for na natural gas production hsm_steo_adgas_adjustment_factor.csv - STEO adjustment factors for ad natural gas production

Output restart variables

Crude Oil Restart Variables

- pmmout_rfqtdcrd - Total crude production by HSM region
- pmmout_rfqdcrd - Total crude oil production by HSM region (not including EOR)
- ogsmout_ogqcrrep - Crude oil production by oil category
- ogsmout_ogcrdprd - Crude oil production by HSM region and crude type
- ogsmout_ogcruderef - Crude oil production by LFMM crude oil type and region

Natural Gas Restart Variable

- ogsmout_ogqngrep - Natural gas production by natural gas type

Well Restart Variables

- ogsmout_ogogwells - Total wells
- ogsmout_ognowell - Total completed wells

NGPL Production Restart Variables

- ogsmout_ogngplprd - NGPL production by HSM district
- ogsmout_ogngplet - Ethane production by HSM district
- ogsmout_ogngplpr - Propane production by HSM district
- ogsmout_ogngplbu - Butane production by HSM district
- ogsmout_ogngplis - Isobutane production by HSM district
- ogsmout_ogngplpp - Pentanes production by HSM district

Module class methods

class module_unf.**Module**

Bases: object

Head module for HSM

__init__()

Initializes Module object.

Return type
Module

setup(*standalone_directory*, *integrated_directory*, *integrated_input_path*, *setup_filename*, *year*, *pyfiler1*, *cycle*, *scedes*)

HSM Setup function. Orchestrates all per-year setup by delegating to private helpers:

1. `_init_logger_and_cycle` — instantiate logger; assign current cycle
2. `_setup_directories` — resolve standalone vs. integrated input/output paths
3. `_init_pyfiler` — assign pyfiler1 restart file interface
4. `_load_config` — read setup.csv; load scalar params, switches, discounting table, mapping
5. `_instantiate_submodules` — import and construct all submodule objects
6. `restart.setup` / `load_parameters` — read restart file; load NEMS parameters
7. `_load_scedes` — parse scedes flags; set side case adjustments and final_aeo_year
8. `process_restart_variables` — create local copies of restart variables
9. `_manage_intermediate_vars` — clean stale pickle files; load iterative state
10. `_load_history_and_steo` — load history and STEO data into restart variables
11. `_setup_cashflow_classes` — configure CashFlow and NGP_CashFlow class state
12. `_setup_submodules` — read submodule switches; run each submodule's setup method

Parameters

- **standalone_directory** (str) – Filepath for standalone HSM run main directory
- **integrated_directory** (str) – Filepath for integrated HSM run main directory
- **integrated_input_path** (str) – Filepath for integrated HSM run input directory
- **setup_filename** (str) – setup.csv filename

- **year** (int) – Current model year
- **pyfiler1** (module or None) – Pyfiler1 module instance passed from NEMS (integrated); None triggers local import (standalone)
- **cycle** (str) – Current NEMS cycle identifier; 'no load on cycle' for standalone
- **scedes** (dict or None) – Scedes scenario flags passed from NEMS (integrated); None triggers standalone file parse

Returns

- **self.main_directory** (str) – Main directory filepath
- **self.input_path** (str) – Inputs directory filepath
- **self.output_path** (str) – Outputs directory filepath
- **self.hsm_var_output_path** (str) – Intermediate variable file directory path
- **self.integrated_switch** (bool) – Flag indicating whether HSM is running in an integrated environment or as a standlone run
- **self.hist_input_path** (str) – Historical data input file directory filepath
- **self.steo_input_path** (str) – STEO input file directory filepath
- **self.onshore_input_path** (str) – Onshore submodule input file directory filepath
- **self.offshore_input_path** (str) – Offshore submodule input file directory filepath
- **self.alaska_input_path** (str) – Alaska submodule input file directory filepath
- **self.canada_input_path** (str) – Canada submodule input file directory filepath
- **self.setup_table** (pandas.DataFrame) – DataFrame of key model inputs (model flags, model history year, etc.)
- **self.base_price_year** (int) – Hardcoded base year for prices in NEMS (1987)
- **self.aeo_year** (int) – AEO publication year
- **self.history_year** (int) – Last year for which HSM contains historical data
- **self.technology_year** (int) – Base year for technology improvement in Submodules
- **self.final_aeo_year** (int) – Final AEO model year
- **self.steo_years** (list) – Current STEO forecast years
- **self.run_every_itr_switch** (bool) – Switch indicating whether HSM should run every iteration, or just itrs 1, fcrl = 1 and ncrl = 1
- **self.side_case_steo_overwrite_switch** (bool) – Switch indicating whether side case STEO overwrites from reference case CSV files should be enabled
- **self.debug_switch** (bool) – Switch indicating whether HSM-specific debug files should be printed out (increases runtime)
- **self.hsm_var_debug_switch** (bool) – Switch indicating whether .csv copies of hsm intermediate files should be printed out (increases runtime)
- **self.discounting_table** (pandas.DataFrame) – DataFrame with key discount rate assumptions
- **self.debt_ratio** (float) – Assumption of debt ratio used to calculated WACC
- **self.fed_tax_rate** (float) – Federal tax rate

- **self.industry_beta** (float) – Assumption of industry beta used to calculated WACC
- **self.market_risk_premium** (float) – Assumption of market risk premium used to calculated WACC
- **self.return_over_capital_cost** (float) – Assumption of return over capital cost used to calculated WACC
- **self.scedes** (dict) – Dictionary of key scedes values used by HSM
- **self.side_case_adj** (float) – Adjustment factor for side case (used for LOGS/HOGS side cases)
- **self.low_price_case** (bool) – Flag indicating that a low oil price side case is being run

load_parameters()

Loads in HSM Parameters.

Returns

- **self.param_baseyr** (int) – Base AEO model year for history (hardcoded as 1990)
- **self.param_fcrl** (bool) – Flag to indicate last regular model iteration
- **self.param_ncrl** (bool) – Flag to indicate reporting model iteration
- **self.param_num_ogsm_region** (int) – Number of HSM regions
- **self.param_num_yr** (int) – Number of model years
- **self.param_oiltypes** (int) – Number of HSM oil types
- **self.param_gastype** (int) – Number of HSM gas types
- **self.param_onshore_regions** (int) – Number of HSM onshore regions
- **self.param_offshore_regions** (int) – Number of HSM offshore regions

process_restart_variables()

Processes Restart Variables for HSM.

- Sets current model year variables
- Sets relevant iterative restart variable values to 0
- Creates local variables from restart file for use in the submodules

Returns

- **self.current_year** (int) – Current model run year
- **self.current_iteration** (int) – Current model run iteration
- **self.rest_mc_jpgdp** (pandas.DataFrame) – DataFrame of inflation multiplier relative to base hardcoded 1987 prices
- **self.rest_mc_rmcorpbaa** (pandas.DataFrame) – Corporate bond rate
- **self.rest_mc_rmtcm10y** (int) – 10-year treasury rate
- **self.rest_ogprcng** (pandas.DataFrame) – Natural gas wellhead-type price at district level (1987$/ thouand-cubic-feet)
- **self.rest_ogrnagprd** (pandas.DataFrame) – Realized non-associated natural gas production
- **self.rest_ogwprng** (pandas.DataFrame) – NG wellhead price (1987$)

- **self.rest_rfcrudewhp** (pandas.DataFrame) – Crude oil wellhead price by refinery region and type
- **self.rest_ogcruderef** (pandas.DataFrame) – Crude oil production by refinery region and crude type
- **self.rest_dcrdwhp** (pandas.DataFrame) – Domestic crude oil wellhead price by region (calculated from reg_crude_price, derived from rfcrudewhp)
- **self.rest_rfqtdcrd** (pandas.DataFrame) – Domestic crude oil production by region (including EOR)
- **self.rest_start_price** (pandas.DataFrame) – Model starting world oil price
- **self.rest_ogpngwhp** (pandas.DataFrame) – National natural gas average welhead price
- **self.rest_plginpf** (pandas.DataFrame) – NGPL prices from LFMM
- **self.rest_cflgq** (pandas.DataFrame) – NGPL conversion factor
- **self.rest_45q_lyr_ret** (pandas.DataFrame) – Year tax credit expire for retrofits
- **self.rest_45q_duration** (pandas.DataFrame) – Number of years tax credit apply to plant
- **self.rest_ccs_eor_45q** (pandas.DataFrame) – Tax credits for enhanced oil recovery in dollars per metric ton co2
- **self.rest_brent_price** (pandas.DataFrame) – Brent price path
- **self.rest_hh_price** (pandas.DataFrame) – Henry Hub price path

run(*year=None*, *iteration=None*)

Runs HSM Model.

- Sets master model year and iteration variables
- Performs global calculations for discount rate and prices
- Runs Submodules

Parameters

- **year** (int) – Current model year
- **iteration** (int) – Current model iteration

Returns

- **self.current_year** (pandas.DataFrame) – Current model year
- **self.current_iteration** (pandas.DataFrame) – Current model iteration

calculate_discount_rate()

Calculates discount rate to be used in discounted cash flows (**CashFlow.py**).

Returns

- **self.corp_bond_rate** (float) – Corporate bond rate
- **self.ten_year_treas_note_yield** (float) – Ten-year Treasury note yield
- **self.expected_inflation_rate** (float) – Expected inflation rate
- **self.discount_rate** (float) – Discount rate

calculate_prices()

Aggregates and shares out crude type and natural gas prices to regions and districts.

The calculation of regional prices should be aggregated up from the district, crude type level, currently crude production by type and district is not output by OGSM, so the LFMM regions are directly converted to OGSM/HSM regions.

Orchestrates price calculations via helper methods with NaN/zero handling.

Returns

- `self.dist_natgas_type_prod` – Natural gas production by HSM district and type
- `self.dist_natgas_prod` – Natural gas production by HSM district
- `self.dist_natgas_price` – Natural gas price by HSM district
- `self.reg_natgas_price` – Natural gas price by HSM region
- `self.lfmm_reg_crude_type_price` – Crude oil price by type and LFMM region
- `self.lfmm_reg_crude_type_prod` – Crude oil production by type and LFMM region
- `self.dist_crude_type_price` – Crude oil price by type and HSM district
- `self.reg_crude_price` – Crude oil price by HSM region

prepare_results()

Prepare results for writing to the restart file.

- Prepares results by submodule
- Then aggregates results across submodule
- Then benchmarks to STEO if model year in STEO years
- Then adjusts results to match realized natural gas production from NGMM if ncrl = 1
- Then writes out intermediate pickles and write results to the restart file

Returns

- **self.restart.ogsmout_ogprcexp** (`pandas.DataFrame`) – HSM price expectation adj (deprecated)
- **self.restart.ogsmout_oggrowfac** (`pandas.DataFrame`) – HSM growth factor (deprecated)

run_ngp()

Assess HSM results to produce Natural Gas Procesing Facility CO2 supply.

write_pkl()

Write local variables to PKL

reset_timing()

Reset all timing variables to zero.

Return type
`None`

format_time(*seconds*)

Format elapsed time in seconds to a human-readable string.

Parameters
seconds (`float`) – Elapsed time in seconds

Returns
Formatted time string

Return type
str

get_timing_summary()
Get a formatted summary of all timing data.

Returns
Formatted timing summary table

Return type
str

2.11.3 Submodules

onshore — Lower 48 Onshore Submodule

onshore.py implements the Onshore class. It is the largest module in HSM (~8,291 lines) and contains the complete logic for four resource categories: producing wells, continuous (unconventional), EOR, and undiscovered conventional.

This reference page lists module symbols and docstrings. For model design, assumptions, economics, data inputs, output interpretation, and the *onshore submodule workflow diagram*, see *Lower 48 Onshore Submodule*. **Mathematical notation** for DCF is in *Economic Framework — Discounted Cash Flow Model*; for selection constraints in *Project Selection and Constraints*; for decline curves on the submodule page (*Notation (decline curves)*). Submodule for calculating onshore well production.

Summary

This submodule projects production of crude oil and natural gas from the onshore Lower 48 states in response to price data received from the LFMM and the NGMM. The module operates as follows:

1. The module is initialized in the onshore class *__init__* method, with class dataframes and variables being declared here.
2. The *setup* method is called, which reads in the necessary input files via .csv or .pkl (see **intermediate_var_pickle.py**) format.
3. The *run* method is called, which kicks off the main model functions.
4. Year 1 operations are called from the *run* method, these operations are listed below. All other operations are run every model year unless otherwise indicated:

Setup Operations

a. *set_up_projects* - Setup for base project table (i.e. declare project resids, merge projects with process codes, etc.).

b. *setup_output_tables* - Setup output tables and set baseline crude and natural gas production from producing projects.

Legacy Producing Project Operations

d. *producing_projects_load_prices* - Get oil and natural gas prices for legacy producing projects using base international prices, so that cash flow can be calculated to determine producing projects economic life (i.e. when the project is no longer producing positive net revenue).

e. *producing_projects_load_costs* - Load operating costs for legacy producing projects so that cash flow can be calculated.

f. *producing_projects_calculate_drilling_capex* - Applies drilling cost equations for legacy producing projects so that cash flow can be calculated.

g. *producing_projects_load_cashflow* - Load legacy producing projects into the cashflow.

h. *run_producing_cash_flow* - Run cash flow for legacy producing projects to determine project economic life.

i. *shut_down_unprofitable_legacy_production* - Shut down legacy producing project production in year when project net income becomes negative.

j. *producing_projects_baseline_constraints* - Load baseline producing project rig and footage values for constraints.

Continuous, EOR and Undiscovered Projects Setup Operations

k. *load_continuous_projects* - Load continuous projects into the master self.projects dataframe for economic analysis.

l. *load_eor_projects* - Load EOR/ASR projects into the master self.projects dataframe for economic analysis.

m. *filter_undiscovered_projects* - Removes projects on restricted land from the undiscovered projects list and deletes duplicates.

Cost Setup Operations

n. *set_base_project_params* - Perform merges and calculations for base parameters commonly used in economic analysis.

o. *load_costs_setup* - Loads production opex, transportation opex, sga opex and facility capex into projects.

p. *drill_cost_eqs_assumptions* - Adjusts self.drill_cost_eq_coefs so that statistically insignificant coefficients are set to 0 (assumed to be indistinguishable from distribution).

q. *calculate_drilling_capex_setup* - Applies drilling cost equations derived from historical drilling cost data to projects.

r. *startup_cost_adjustments* - Reduces costs for projects with historical drilling to ensure they are selected for future drilling.

5. Calculate crude oil, natural gas, and ngpl prices by region based on price values from LFMM and NGMM in *load_prices*.

6. Calculate project constraints based on projected brent prices in *calculate_drilling_constraints* and *calculate capital constraint*.

7. Load in discovered projects based on load order generated from a Monte Carlo simulation. Discovered projects are merged with the main projects table for calculations:

   a. *set_undiscovered_drilling_params* - Set base undiscovered drilling assumptions (i.e. max drill rate, available wells, etc.).

   b. *calculate_undiscovered_drilling* - Calculate drilling required to explore and locate an undiscovered project (Same methodology as regular drilling equation, but used to determine number of wells needed to be drilled to discover a project).

   c. *load_undiscovered_projects* - Applies rig constraint to load in undiscovered projects.

8. Calculate production/well in *calculate_production* and apply technology improvement rate in *calculated_prod_tech_improvement*.

9. Set drilling paramaters for drilling (i.e. max wells, remaining wells, etc.) in *set_drilling params*, then call the *on_next_wells* drilling equation from **drilling_equations.py** in *calculate drilling* to determine model year project drilling. Dryholes are also calculated here.
10. Apply cost technology improvement rates in *apply_cost_tech_rate*.
11. Calculate remaining annualized costs based on brent_price in *calculate_exploration_costs* and *calculate_ngpl_costs*.
12. Setup CO2 EOR for integration with CTUS:
    a. *co2_eor_econ* - Calculates costs and tax credits related to CO2 EOR.
    b. *determine_co2_supply_prices* - Calculate Recycled CO2 supply and prices. Pull and sort CO2 supply and price curves from CTUS, LFMM, and EMM. Then merge CO2 supply and prices from all sources into a single table.
    c. *calculate_co2_project_costs* - Apply CO2 prices to CO2 required for each CO2 EOR project, using the lowest-cost prices first for all projects, and apply to the self.projects dataframe as a single weighted-average cost.
13. Load and run cash flow in *load_cashflow* and *run_cashflow*. Rank projects by project net present value.
14. Determine whether EOR projects are eligible to run (i.e. no other EOR project types have been selected for the same resid) in *determine_eor_eligibility*.
15. Apply drilling constraints in *apply_rig_constraints* and *apply_footage_constraints*, so that project drilling does not exceed projected available capacity to drill.
16. Run *select_projects* to mask for all drilling constraints and determine final project drilling. Assign production and well counts to the relevant output tables and update iterative calculation tables.
17. Run *write_intermediate_variables* to store iterative calculation tables for the next model run year. These tables are written out to .pkl files in **intermediate_var_pickle.py**.
18. *report_results_unf* is called from **module_unf.py** to report results to debug files and the restart file.

Input files

- on_projects_continuous - Continuous projects
- on_projects_producing_gas - Producing gas projects
- on_projects_producing_oil - Producing oil projects
- on_projects_co2_eor - CO2 eor projects
- on_projects_undiscovered - Undiscovered projects
- on_process_codes - Process codes
- on_old_process_code_conv - Convert OGSM proc codes to HSM codes
- on_rig_constraint_eq - Rig constraint coefficients
- on_footage_constraint_eq - Footage constraint coefficients
- on_capital_constraint_eq - Coefficient: natural log of Brent price
- on_dryhole_rate - Dryhole rate according to region number
- on_cost_eqs - Cost equations

Model functions and class methods

class Onshore(sub.Submodule) - Onshore submodule for HSM

- __init__ - Constructor to initialize Onshore submodule
- setup - Method to set up Onshore Submodule for HSM

- load_input_variables - Method to read variable data
- load_input_tables - Method to read table data from input files
- set_up_projects - Method to prepare tables for processing
- setup_output_tables - Method to set baseline crude and natural gas production from producing projects
- load_continuous_projects - Method to load continuous projects for economic analysis
- load_eor_projects - Method to load EOR projects for economic analysis
- cost_eq_setup - Method to maps, format, and clean data for cost equations
- run - Method to run Offshore Submodule for HSM
- calculate_production - Calculates production from setup table and lists paths of input files
- load_prices - Method to calculate average prices for Onshore regions; averages prices over years set in averaging_years (positive = forward-looking, negative = backward-looking)
- calculate_drilling - Method to calculate yearly drilling for continuous formations
- load_costs - Method to load operating expenses (Production and Transport) and assign to projects
- calculate_drilling_capex - Method to apply capex equations derived from commercial historical cost data to calculate project costs
- co2_eor_econ - Method to shut down ineligible projects
- calculate_geo_geo_and_lease_aq - Method N/A
- load_cashflow - Method to load cash flow
- run_cashflow - Method to run cash flow
- calculate_constraints - Method to calculate rig, footage, and capital constraints
- select_projects - Method to apply constraints and rank projects
- report_results_unf - Method to report results to restart variables

Output debug files

- on_crude_district_num.csv - Crude Oil Production By District Number
- on_crude_federal_land.csv - Crude Oil Production Grouped By Federal vs. Nonfederal Land
- on_crude_fields.csv - Crude Oil Production
- on_crude_proc_code.csv - Crude Oil Production Grouped By Process Code
- on_crude_region_num.csv - Crude Oil Production Grouped By Region Number
- on_natgas_district_num.csv - Natural Gas Production By District Number
- on_natgas_federal_land.csv - Natural Gas Production Grouped By Federal vs. Nonfederal Land
- on_natgas_fields.csv - Natural Gas Production
- on_natgas_proc_code.csv - Natural Gas Production Grouped By Process Code
- on_natgas_region_num.csv - Natural Gas Production Grouped By Region Number
- on_wells_district_num.csv - Wells By District Number
- on_wells_fields.csv - Wells Grouped By Resid, Play, Process Code, Region Number, District Number, Well Type Number, Well Limit
- on_wells_proc_code.csv - Wells Grouped By Process Code

- on_wells_region_num.csv - Wells Grouped By Region Number

Output restart variables

Crude Oil Restart Variables

- pmmout_rfqtdcrd - Total crude production by HSM region
- pmmout_rfqdcrd - Total crude oil production by HSM region (not including EOR)
- ogsmout_ogqcrrep - Crude oil production by oil category
- ogsmout_ogcoprd - Crude oil production by lower 48 region
- ogsmout_ogqshloil - Crude oil production by select tight oil play
- ogsmout_ogoilprd - Crude oil production by oil type and HSM district
- ogsmout_ogcrdprd - Crude oil production by HSM region and crude type
- ogsmout_ogcruderef - Crude oil production by LFMM crude oil type and region
- ogsmout_ogcrdheat - Heat rate by type of crude oil
- ogsmout_ogeorprd - CO2 EOR crude oil production

Natural Gas Restart Variables

- ogsmout_ogenagprd - Natural gas expected production by natural gas type and HSM district
- ogsmout_ogrnagprd - Natural gas realized production by natural gas type and HSM district
- ogsmout_ogadgprd - Natural gas associated dissolved production by oil type and HSM district
- ogsmout_ogprdad - Natural gas associated dissolved production by HSM region
- ogsmout_ogqshlgas - Natural gas production by select natural gas play
- ogsmout_ogqngrep - Natural gas production by natural gas type
- ogsmout_ogprdugr - Lower 48 unconventional natural gas production

Crude Oil and Natural Gas Production Restart Variable

- ogsmout_ogregprd - Total crude oil and natural gas production by production type

Crude Oil Price Restart Variables

- ogsmout_ogcowhp - Crude oil wellhead price by HSM region
- ogsmout_ogpcrwhp - Crude oil HSM average wellhead price

Natural Gas Price Restart Variable

- ogsmout_ogngwhp - Natural gas wellhead price by HSM region

Well Restart Variables

- ogsmout_ogogwells - Total wells
- ogsmout_ognowell - Total completed wells
- ogsmout_ogwellsl48 - Total lower 48 wells
- ogsmout_ogsrl48 - Lower 48 drilling success rates

NGPL Production Restart Variables

- ogsmout_ogngplprd - NGPL production by HSM district
- ogsmout_ogngplet - Ethane production by HSM district

- ogsmout_ogngplpr - Propane production by HSM district
- ogsmout_ogngplbu - Butane production by HSM district
- ogsmout_ogngplis - Isobutane production by HSM district
- ogsmout_ogngplpp - Pentanes production by HSM district

EOR Restart Variables

- ogsmout_ogco2rec - CO2 recycled by HSM region and CO2 type
- ogsmout_ogco2inj - CO2 injected by HSM region and CO2 type
- ogsmout_ogco2pur - CO2 purchased by HSM region and CO2 type
- ogsmout_ogco2avl - CO2 available by HSM region and CO2 type
- ogsmout_ogco2prc - CO2 price by HSM region and CO2 type

Technology Improvement Rate Restart Variable

- ogsmout_ogtechon - HSM technology improvement rate

Onshore submodule class methods

class onshore.**Onshore**(*parent*)

Bases: *Submodule*

Onshore submodule for HSM.

Parameters
parent (str) – Module_unf.Module (Pointer to parent module)

setup(*setup_filename*)

Setup Onshore submodule for HSM.

Parameters
setup_filename (str) – Path to offshore setup file.

Return type
None

load_input_variables()

Reads input variable data from self.setup_table (loaded in super().setup(setup_filename)) then adjusts base prices for inflation.

Returns

- **self.zero_year** (int) – First model year
- **self.final_year** (int) – Final model year
- **self.evaluation_years** (int) – Number of evaluation years (self.final_year - self.zero_year + 1)
- **self.averaging_years** (int) – Number of averaging years used to produce crude oil and natural gas prices
- **self.royalty_rate** (float) – U.S. Federal royalty rate
- **self.base_oil_prc** (float) – Baseline oil price used to determine drilling responsiveness (i.e. if model year oil price > baseline oil price, then drilling is adjusted higher)
- **self.base_gas_prc** (float) – Baseline natgas price used to determine drilling responsiveness (i.e. if model year natgas price > baseline natgas price, then drilling is adjusted higher)

- **self.oil_exp_tang_frac** (float) – Fraction of oil well exploration costs that are tangible
- **self.oil_dev_tang_frac** (float) – Fraction of oil well development costs that are tangible
- **self.gas_exp_tang_frac** (float) – Fraction of gas well exploration costs that are tangible
- **self.gas_dev_tang_frac** (float) – Fraction of gas well development costs that are tangible
- **self.kap_tang_frac** (float) – Fraction of well capital costs that are tangible
- **self.amor_schedule** (str) – Amortization schedule code (see **depreciation_schedules.csv**)
- **self.deprec_schedule** (str) – Depreciation schedule code (see **depreciation_schedules.csv**)
- **self.intang_amor_frac** (float) – Fraction of costs to be amortized
- **self.abandon_rate** (float) – Well abandonment cost rate
- **self.drill_ramp_up** (int) – Drilling ram-up years
- **self.drill_predecline** (float) – Fraction of total available wells drilled at which point drilling decline starts (i.e. 70% of available wells)
- **self.eor_tc_rate** (float) – EOR tax credit rate
- **self.eor_tc_phaseout** (int) – EOR tax credit phaseout year
- **self.co2_trans_cost** (float) – CO2 transportation cost

load_input_tables()

Reads the following table data from input files:

- Process Codes
- Undiscovered projects discovery order
- Projects by type (i.e. continuous, producing oil, etc.)
- Cost input
- Constraint input
- Select plays list for output tables
- technology levers
- CO2 supply and price input

self.setup_table (loaded in super().setup(setup_filename)) lists paths of input files.

Returns

- **self.process_codes** (pandas.DataFrame) – Table of process codes which describe projects (i.e. process code 0 = producing vertical oil project, process code 1 = producing vertical natural gas projects).
- **self.old_process_code_conv** (pandas.DataFrame) – Table that maps legacy OGSM process codes to HSM process codes.
- **self.discovery_order** (pandas.DataFrame) – Results of a Monte Carlo simulation to determine discovery order for undiscovered vertical projects in the *projects_undiscovered* df
- **self.projects_continuous** (pandas.DataFrame) – Table of horizontal projects
- **self.projects_producing_gas** (pandas.DataFrame) – Table of legacy producing vertical natural gas projects

- **self.projects_producing_oil** (pandas.DataFrame) – Table of legacy producing vertical oil projects
- **self.projects_co2_eor** (pandas.DataFrame) – Table of CO2 EOR projects
- **self.projects_undiscovered** (pandas.DataFrame) – Table of undiscovered projects
- **self.drill_eq_constraints** (pandas.DataFrame) – Table of drilling equation constraints (i.e. max available wells that can be drilled/year)
- **self.drill_cost_eq_coefs** (pandas.DataFrame) – Table of cost equations based on inputs from a commercial upstream cost database
- **self.region_costs** (pandas.DataFrame) – Table of average facility, operating, GA and transportation costs by HSM region based on inputs from a commercial upstream cost database
- **self.basin_costs** (pandas.DataFrame) – Table of average facility, operating, GA and transportation costs by USGS province based on inputs from a commercial upstream cost database
- **self.dryhole_rate** (pandas.DataFrame) – Table of dryhole rate by horizontal projects, undiscovered vertical projects, and discovered vertical projects
- **self.ngpl_costs** (pandas.DataFrame) – Table of legacy OGSM NGPL cost equations
- **self.rig_constraint_eq** (pandas.DataFrame) – Table of rig constraint equations by HSM region based on data from the DPR (Naser Ameen)
- **self.footage_constraint_eq** (pandas.DataFrame) – Table of footage constraint equations by HSM region based on data from the DPR (Naser Ameen)
- **self.capital_constraint_eq** (pandas.DataFrame) – Table of a capital constraint equation based on Evaluate Energy dataset (Jeff Barron)
- **self.constraint_params** (pandas.DataFrame) – Table containing rig constraint allocation to discovering vertical wells vs. drilling horizontal well
- **self.wells_per_rig** (pandas.DataFrame) – Table of wells/rig for constraint equations by HSM region based on data from the DPR (Naser Ameen)
- **self.tech_levers** (pandas.DataFrame) – Table of technology improvement rates
- **self.eor_other_costs** (pandas.DataFrame) – Table of legacy OGSM Other EOR project costs
- **self.base_co2_supply_price** (pandas.DataFrame) – Table of legacy OGSM CO2 EOR supply and price curves
- **self.co2_legacy_costs** (pandas.DataFrame) – Table of legacy CO2 EOR costs
- **self.co2_pipe_seg_costs** (pandas.DataFrame) – Table of legacy CO2 region-to-region transportation costs in OGSM
- **HSM Pickle Files** (pandas.DataFrame) – Collection of iterative intermediate model tables (i.e. tables the model updates and then needs again each year) that are input/output in **intermediate_var_pickle.py**

run()

Run Onshore Submodule for HSM.

1. In zero year setup project tables, run producing projects, and setup project costs
2. In zero year onward run main model processes

Year 1 Setup Methods

- self.set_up_projects
- self.setup_output_tables
- self.load_continuous_projects
- self.load_eor_projects
- self.filter_undiscovered_projects
- self.set_base_project_params

Year 1 Producing Project Methods

- self.producing_projects_load_prices
- self.producing_projects_load_costs
- self.producing_projects_calculate_drilling_capex
- self.producing_projects_load_cashflow
- self.run_producing_cash_flow
- self.shut_down_unprofitable_legacy_production
- self.producing_projects_baseline_constraints

Year 1 Cost Setup Methods

- self.load_costs_setup
- self.drill_cost_eqs_assumptions
- self.calculate_drilling_capex_setup
- self.startup_cost_adjustments

Annual Model Methods

- self.load_prices
- self.calculate_drilling_constraints
- self.calculate_capital_constraint
- self.set_undiscovered_drilling_params
- self.calculate_undiscovered_drilling
- self.load_undiscovered_projects
- self.calculate_production
- self.calculate_prod_tech_improvement
- self.set_drilling_params
- self.calculate_drilling
- self.apply_cost_tech_rate
- self.calculate_exploration_costs
- self.calculate_ngpl_costs
- self.co2_eor_econ
- self.determine_co2_supply_prices

- self.calculate_co2_project_costs
- self.load_cashflow
- self.run_cashflow
- self.determine_eor_eligibility
- self.apply_rig_constraints
- self.apply_footage_constraints
- self.select_projects
- self.write_intermediate_variables

Return type
None

set_up_projects()

Prepares tables for processing by:

- Creating a unique numerical id for each project
- Merging projects with process codes
- Creating id columns (i.e. state abbreviation and region numbers)

Returns

- **self.projects_continuous** (pandas.DataFrame) – Table of horizontal projects
- **self.projects_producing_gas** (pandas.DataFrame) – Table of legacy producing vertical natural gas projects
- **self.projects_producing_oil** (pandas.DataFrame) – Table of legacy producing vertical oil projects
- **self.projects_co2_eor** (pandas.DataFrame) – Table of CO2 EOR projects
- **self.projects_undiscovered** (pandas.DataFrame) – Table of undiscovered projects

setup_play_map()

Setup Restart file play map.

Returns
self.play_map – DataFrame containing mapping of CO2 EOR plays

Return type
pandas.DataFrame

apply_royalty_multiplier_overrides(*project_df*)

Apply royalty multiplier overrides to projects based on region and commodity type.

Parameters
project_df (pandas.DataFrame) – DataFrame containing projects with columns: region_number, region_name, resource_type, project_royalty_multiplier

Returns
project_df – DataFrame with updated project_royalty_multiplier values

Return type
pandas.DataFrame

setup_output_tables()

Setup output tables and set baseline production volumes and well counts from producing projects.

- Only legacy producing vertical wells are counted, horizontal wells are not
- These well counts are then adjusted based on oil price (higher oil price -> more wells)
- Current well counts are based on 2018 annual estimates, which require some degree of analyst judgement in the *Set Well Ratio* section of the code to smooth

Returns

- **self.crude_production** (pandas.DataFrame) – DataFrame of onshore crude oil production
- **self.natgas_production** (pandas.DataFrame) – DataFrame of onshore natural gas production
- **self.ngpl_production** (pandas.DataFrame) – DataFrame of onshore natural gas plant liquids production
- **self.water_production** (pandas.DataFrame) – DataFrame of onshore water production
- **self.wells** (pandas.DataFrame) – DataFrame of onshore wells

producing_projects_load_prices()

Get oil, natural gas and NGPL prices for legacy producing projects based on starting international oil price, regional natural gas wellhead prices, and regional NGPL prices, so that projects can be run through the cash flow to determine project economic life (i.e. when the project is no longer producing positive net revenue).

Returns

- **self.producing_price_df** (pandas.DataFrame) – Table of crude oil and natural gas prices for producing projects
- **self.producing_projects** (pandas.DataFrame) – Table of legacy producing projects

producing_projects_load_costs()

Apply operating and facility costs to legacy producing projects, so that they can be run through the cash flow to determine project economic life (i.e. when the project is no longer producing positive net income).

- Costs are based on average historic costs derived from a commercial upstream cost database
- Costs are mapped by production type and USGS Province (i.e. basin)
- If there are no reported historic costs at the USGS Province level, then costs are assigned as the HSM region level average with a cost adder to indicate that there is limited drilling in these areas
- Apply tech improvement rate (since cost average is derived from the past 5 years of data)

Returns

self.producing_projects – Table of legacy producing projects

Return type

pandas.DataFrame

producing_projects_load_ch4_costs()

Load methane venting/flaring costs/ton of ch4 vented/flared.

Returns

self.ch4_emission_cost – DataFrame containing CH4 emission cost bases (costs are not consistent across years)

Return type
pandas.DataFrame

producing_projects_calculate_drilling_capex()

Applies drilling cost equations derived from historical drilling cost data.

Notes

1. **Separate cost equations for oil and gas; however, they have the same processes:**
   a. Set baseline cost to intercept value
   b. Apply basin and state coefficients, with missing data being filled in by an average of district, region, type production and a cost adder
   c. Apply lateral length and depth coefficients

2. If the cost equation produces outliers (costs that exceed the top and bottom historical percentile of costs, with some additional room for change with time), a warning is generated

3. **Calculate for dry hole cost:**
   a. Data source reports that 1/3 of capital costs are for drilling, 2/3 for completion
   b. Take drilling cost and multiply by dryhole rate to get dryhole costs

Returns
self.producing_projects – Table of legacy producing projects

Return type
pandas.DataFrame

producing_projects_load_cashflow()

Load legacy producing projects into the cashflow.

Returns

- **self.cash_flow.properties** (pandas.DataFrame) – DataFrame containing properties used in the cash flow (costs, tangible/intangible cost ratios, etc.)
- **self.cash_flow.crude_production** (pandas.DataFrame) – DataFrame of onshore crude oil production
- **self.cash_flow.natgas_production** (pandas.DataFrame) – DataFrame of onshore natural gas production
- **self.cash_flow.crude_price** (pandas.DataFrame) – DataFrame of crude prices
- **self.cash_flow.natgas_price** (pandas.DataFrame) – DataFrame of natural gas prices
- **self.cash_flow.general_admin_cost** (pandas.DataFrame) – DataFrame of GA costs
- **self.cash_flow.kap_cost** (pandas.DataFrame) – DataFrame pf capital costs

run_producing_cash_flow()

Run cash flow for legacy producing projects to determine project economic life.

- Project economic life is used to determine when EOR/ASR projects can begin production
- Secondary production can only turn on within 5 years of the primary production economic life
- Only calculate to abandonment when project economic life is determined

Returns
self.producing_projects – Table of legacy producing projects

Return type
pandas.DataFrame

shut_down_unprofitable_legacy_production()

Shut down legacy well production in the year when net income becomes negative.

Returns

- **self.crude_production** (pandas.DataFrame) – DataFrame of onshore crude oil production
- **self.natgas_production** (pandas.DataFrame) – DataFrame of onshore natural gas production
- **self.ngpl_production** (pandas.DataFrame) – DataFrame of onshore natural gas plant liquids production
- **self.water_production** (pandas.DataFrame) – DataFrame of onshore water production
- **self.wells** (pandas.DataFrame) – DataFrame of onshore wells

producing_projects_baseline_constraints()

Get baseline producing project rig and footage values for constraints.

Returns

- **self.producing_wells** (pandas.DataFrame) – DataFrame of producing projects wells
- **self.producing_footage** (pandas.DataFrame) – DataFrame of producing projects cumulative footage

load_continuous_projects()

Load continuous projects into the master self.projects dataframe for economic analysis.

- Assign last_year_drilling as NINJWELL value
- Use past_wells from input file and validate consistency

Returns
self.projects – Master DataFrame containing all projects to be processed in current model year

Return type
pandas.DataFrame

load_eor_projects()

Load EOR/ASR projects into the master self.projects dataframe for economic analysis.

- Load CO2 EOR projects
- Pull out resource-restricted projects
- Merge in economic life for producing projects to determine project eligibility (EOR projects can only move forward if associated producing project is approaching/just past end of economic life)

Returns
self.projects – Master DataFrame containing all projects to be processed in current model year

Return type
pandas.DataFrame

filter_undiscovered_projects()

Removes projects on restricted land from the undiscovered projects list and deletes duplicates.

Returns
self.projects_undiscovered – DataFrame of undiscovered projects

Return type
pandas.DataFrame

set_base_project_params()

Perform merges and calculations to self.projects for base parameters commonly used in economic analysis.

- Apply Side case adjustments
- In the Oil and Gas Supply Cases EURs and technology rates are adjusted by 50%
- Sidecase switch is pulled in from Scedes
- Merge technology improvement rates
- Apply drilling restrictions based on resource access flag

Returns
self.projects – Master DataFrame containing all projects to be processed in current model year

Return type
pandas.DataFrame

load_costs_setup()

Loads production opex, transportation opex, sga opex and facility capex into projects.

- All Costs are $/BOE
- Projects are each matched to cost by production type and basin
- If no basin-level costs are available, default to region-level costs with a cost adder
- if no region-level costs are available default to national-level average costs with a cost adder
- Additional cost adder is assigned for Other EOR projects, replacing the low-impact, EOR-type specific methodolgy in OGSM

Returns

- **self.projects** (pandas.DataFrame) – Master DataFrame containing all projects to be processed in current model year
- **self.projects_undiscovered** (pandas.DataFrame) – DataFrame of undiscovered projects

drill_cost_eqs_assumptions()

Adjusts self.drill_cost_eq_coefs so that statistically insignificant coefficients are set to mean of coefficients (assumed to be indistinguishable from distribution).

Returns
self.drill_cost_eq_coefs – Table of cost equations based on inputs from a commercial upstream cost database

Return type
pandas.DataFrame

calculate_drilling_capex_setup()

Applies drilling cost equations derived from historical drilling cost data.

Notes

1. **Separate cost equations for oil and gas; however, they have the same processes:**
    a. Set baseline cost to intercept value
    b. Apply basin and state coefficients, with missing data being filled in by an average of district, region, type production and a cost adder
    c. Apply lateral length and depth coefficients

2. If the cost equation produces outliers (costs that exceed the top and bottom historical percentile of costs, with some additional room for change with time), a warning is generated

3. **Calaculate for dry hole cost:**
    a. Data provider reports that 1/3 of capital costs are for drilling, 2/3 for completion
    b. Take drilling cost and multiply by dryhole rate to get dryhole costs

4. Additional cost adder is assigned for Other EOR projects, replacing the low-impact, EOR-type specific methodolgy in OGSM

Returns

- **self.projects** (`pandas.DataFrame`) – Master DataFrame containing all projects to be processed in current model year
- **self.projects_undiscovered** (`pandas.DataFrame`) – DataFrame of undiscovered projects

startup_cost_adjustments()

Reduces costs for projects with historical drilling to ensure they are selected for future drilling.

Returns
self.projects – Master DataFrame containing all projects to be processed in current model year

Return type
`pandas.DataFrame`

load_prices()

Calculate average crude, natural gas, and ngpl prices for Onshore variables by region.

- Crude prices are derived from LFMM regional wellhead prices
- Natural gas prices are derived from NGMM regional wellhead prices
- NGPL prices are derived from LFMM aggregate national price

Price averaging behavior:

- Positive averaging_years: forward-looking (current year + N future years) Example: averaging_years=2 averages current year + 2 future years (3 years total)
- Negative averaging_years: backward-looking (N past years + current year) Example: averaging_years=-2 averages 2 past years + current year (3 years total)

Returns

- **self.projects** (`pandas.DataFrame`) – Master DataFrame containing all projects to be processed in current model year
- **self.projects_undiscovered** (`pandas.DataFrame`) – DataFrame of undiscovered projects

calculate_drilling_constraints()

Calculate rig and footage constraints by year based on oil price.

Returns

- **self.exp_const_ratio** (float) – Ratio of drilling constraint capacity assigned to undiscovered production
- **self.dev_const_ratio** (float) – Ratio of drilling constraint capacity assigned to developing production
- **self.rig_constraint** (pandas.DataFrame) – DataFrame of regional rig constraints
- **self.footage_constraint** (pandas.DataFrame) – DataFrame of regional footage constraints

calculate_capital_constraint()

Calculate capital constraint by year based on oil price.

- Available dataset only accounts for public company capital investment, which is ~60% of total, so constraint value is adjusted to account for private company investment as well
- Available dataset doesn't split Offshore and Alaska production from domestic production; the commercial data source reports Offshore investment is approx 15% of US total, Alaska is approx 5%

Returns

- **self.projects** (pandas.DataFrame) – Master DataFrame containing all projects to be processed in current model year
- **self.projects_undiscovered** (pandas.DataFrame) – DataFrame of undiscovered projects

set_undiscovered_drilling_params()

Set base undiscovered drilling assumptions.

Calculates key drilling parameters for undiscovered projects:

- well_decline_limit - Factor of total acreage/average well spacing that determines when wells experience productivity decline
- max_wells - Maximum wells that can be drilled (total acreage/minimum well spacing)
- **max_drill_rate - Maximum annual drilling rate, calculated differently for unconventional vs conventional projects:**
  - Conventional projects: Use max_annual_wells from input data
  - Unconventional projects (process codes 20-22: Tight Oil, Shale Gas, Tight Gas): Use (total_acres / acres_per_section) * undiscovered_unconv_production_factor to limit drilling intensity
- totpat - Total patterns remaining (well_decline_limit - past_wells)
- Drilling constraint percentages based on min_annual_wells and max_annual_pct_dev

Returns

self.projects_undiscovered – DataFrame of undiscovered projects

Return type

pandas.DataFrame

calculate_undiscovered_drilling()

Calculate drilling required to explore and locate an undiscovered project.

Same methodology as regular drilling equation, but used to determine number of wells needed to be drilled to discover a project.

- Drilling limit is reduced in Colorado due to 2020 law increasing drilling setbacks
- *on_next_wells* from **drilling_equations.py** is called to determine number of wells that need to be drilled for a well discovery
- Calculate project dryhole rate for cost equations

Returns
self.projects_undiscovered – DataFrame of undiscovered projects

Return type
pandas.DataFrame

load_undiscovered_projects()

Applies rig constraint to load in undiscovered projects.

- Wells/rig and number of rigs available by region are calculated in the "onshore_constraints" preprocesser
- Projected wells are then split amongst the regions and loaded into the master projects list in order based upon the annual rig constraint tied to brent price
- A minimum of 15% of available rigs or 5 rigs is assigned to each region
- Once projects are discovered they are added to the master projects list and removed from the undiscovered projects list
- This function also reformats undiscovered projects to be added to the master projects list, and saves exploratory wells to track the cumulative annual well count

Returns

- **self.projects** (pandas.DataFrame) – Master DataFrame containing all projects to be processed in current model year
- **self.discovery_order** (pandas.DataFrame) – Results of a Monte Carlo simulation to determine discovery order for undiscovered vertical projects in the *projects_undiscovered* df
- **self.exploratory_wells** (pandas.DataFrame) – DataFrame of exploratory wells
- **self.parent.hsm_vars.on_projects_discovered** (pandas.DataFrame) – DataFrame of discovered projectst that have not yet been assigned to the master project df (duplicates)

calculate_production()

Assigns production of relevant resources for projects.

- Adjusts production units from input file
- Takes in input file production values and applies unit transformations and calculations for NGPLS and CO2 recycling
- Reassigns production df indices every model year to match self.projects index

Returns

- **self.project_crude_production** (pandas.DataFrame) – DataFrame of model year project crude oil production

- **self.project_natgas_production** (pandas.DataFrame) – DataFrame of model year project natural gas production
- **self.project_ngpl_production** (pandas.DataFrame) – DataFrame of model year project NGPL production
- **self.project_water_production** (pandas.DataFrame) – DataFrame of model year project water production
- **self.project_co2_inj** (pandas.DataFrame) – DataFrame of model year project CO2 injected

calculate_prod_tech_improvement()

Calculates project production technology improvement.

- Technology improvement rate is tiered with projects in areas without significant past production receiving a higher technology improvement rate
- For the first 5 model years the tier 2 tech rate is doubled for projects that don't have past drilling (assumes it takes time to figure out where the best resources are)
- Then assumed that productivity increases will be the same across project types
- For highly productive regions (i.e. the Haynesville) there is a manual function for assigning the tier-1 tech rate to "emerging" type projects

Returns

- **self.project_crude_production** (pandas.DataFrame) – DataFrame of model year project crude oil production
- **self.project_natgas_production** (pandas.DataFrame) – DataFrame of model year project natural gas production
- **self.project_ngpl_production** (pandas.DataFrame) – DataFrame of model year project NGPL production

set_drilling_params()

Set base drilling assumptions.

- Set undiscovered projects last year of drilling to 0
- **Two drilling limits are calculated:**
  1. well_decline_limit - A factor of total acreage/average well spacing, which determines when wells start to experience productivity decline (70% of well decline limit)
  2. max_wells - The maximum wells that can be drilling which is a factor of total acreage/minimum well spacing, which is the hard maximum on the number of wells that can be drilled in a project
- totpat (total patterns remaining) is calculated as well_decline_limit - past_wells
- **The project maximum drilling rates are calculated as a maximum of the following two equations:**
  1. Last year drilling * 1.3
  2. Total acreage / 640 * a production factor (this factor is reset every year as an analyst judgement of between 0.08 - 0.1)
- A hard ceiling on drilling is set at 200 wells (which can then be adjusted upward in the drilling equation based on price)
- The number of remaining wells to be drilled is calculated as a factor of max_wells / past_wells

- **A cap on the percentage of available wells that can be drilled is set based on a ceiling of 30%, and the maximum of the following two values:**
  # 1. hist_year_wells / max_wells 2. 3% - 6% depending on project type
- Validates consistency of drilling parameters (past_wells, totpat, well_decline_limit, pattern sizes)

Returns
self.projects – Master DataFrame containing all projects to be processed in current model year

Return type
pandas.DataFrame

calculate_drilling()

Calculates model year drilling for continuous and discovered formations.

- Contains code to increase Colorado well spacing per 2020 setback regulations
- Runs projects through the drilling_eq.py nexwells function to get annual project drilling
- Assigns dryhole values for projects

Returns

- **self.projects** (pandas.DataFrame) – Master DataFrame containing all projects to be processed in current model year
- **self.project_drilling** (pandas.DataFrame) – DataFrame of model year project drilling
- **self.project_dryholes** (pandas.DataFrame) – DataFrame of model year project dryholes

apply_cost_tech_rate()

Applies technology improvement rate to relevant costs.

Returns
self.projects – Master DataFrame containing all projects to be processed in current model year

Return type
pandas.DataFrame

calculate_exploration_costs()

Calculates geological, engineering and lease acquisition costs for discovered wells using legacy OGSM equations.

- We only calculate geological and acquisition costs for undiscovered projects (vs. horizontal projects) because these costs are tied to the discovery

Returns
self.gg_costs – DataFrame of geological, engineering and lease costs for discovered wells.

Return type
pandas.DataFrame

calculate_ngpl_costs()

Calculates NGPL Processing Plant Costs using legacy OGSM equations.

Returns
self.projects – Master DataFrame containing all projects to be processed in current model year

Return type
pandas.DataFrame

load_ch4_emission_costs()

Load methane venting/flaring costs/ton of ch4 vented/flared.

Returns

self.ch4_emission_cost – DataFrame containing CH4 emission cost bases (costs are not consistent across years)

Return type

pandas.DataFrame

co2_eor_econ()

Calculates costs and tax credits related to CO2 EOR using legacy OGSM equations.

- Calculate water handling plant cost
- Calculate recycling plant cost
- Get base EOR tax credit values including 45Q

Returns

- **self.projects** (pandas.DataFrame) – Master DataFrame containing all projects to be processed in current model year
- **self.co2_45q_eor_tax_credit** (pandas.DataFrame) – DataFrame of 45q tax credit values

determine_co2_supply_prices()

Calculate Recycled CO2 supply and prices.

- Calculate CO2 price from natural sources based on crude oil price (CO2 supply is determined by NETL input file)
- Apply Natural CO2 price as CO2 price

Returns

- **self.co2_supply_price** (pandas.DataFrame) – Table of legacy OGSM CO2 EOR supply and price curves
- **self.co2_price** (pandas.DataFrame) – DataFrame of co2 price by source and region

calculate_co2_project_costs()

Apply CO2 prices to CO2 required for each CO2 EOR project, using the lowest-cost prices first for all projects, and apply to the self.projects dataframe as a single weighted-average cost.

- Get CO2 Supply and Costs
- Apply transportation costs for CO2 to CO2 costs (not doing this for constraints because too granular and limited benefit)
- Take only the two most economical CO2 sources by project and apply co2 costs as a weighted average to reduce runtime (we only use the most economical sources because there is no way to determine which CO2 projects will be selected before cash flow, and thus no selection order for CO2 supply)
- We can default to the two lowest cost CO2 sources because CO2 required never exceeds top two sources of CO2 supply, but if this begins to happen with any frequency equation can be updated (at the cost of runtime as this calculation is slow)

Returns

self.projects – Master DataFrame containing all projects to be processed in current model year

Return type
pandas.DataFrame

load_cashflow()

Load developing projects into cash flow.

Returns

- **self.cash_flow.properties** (pandas.DataFrame) – DataFrame containing properties used in the cash flow (costs, tangible/intangible cost ratios, etc.)
- **self.cash_flow.crude_production** (pandas.DataFrame) – DataFrame of onshore crude oil production
- **self.cash_flow.natgas_production** (pandas.DataFrame) – DataFrame of onshore natural gas production
- **self.cash_flow.ngpl_production** (pandas.DataFrame) – DataFrame of onshore NGPL production
- **self.cash_flow.co2_use** (pandas.DataFrame) – DataFrame of onshore CO2 use
- **self.cash_flow.crude_price** (pandas.DataFrame) – DataFrame of crude prices
- **self.cash_flow.natgas_price** (pandas.DataFrame) – DataFrame of natural gas prices
- **self.cash_flow.exp_drill_cost** (pandas.DataFrame) – DataFrame of exploratory well drill costs
- **self.cash_flow.dev_drill_cost** (pandas.DataFrame) – DataFrame of development well drill costs
- **self.cash_flow.exp_dry_cost** (pandas.DataFrame) – DataFrame of exploratory well dryhole costs
- **self.cash_flow.dev_dry_cost** (pandas.DataFrame) – DataFrame of development well dryhole costs
- **self.cash_flow.gg_la_cost** (pandas.DataFrame) – DataFrame of geological, engineering and lease costs for undiscovered projects
- **self.cash_flow.general_admin_cost** (pandas.DataFrame) – DataFrame of GA costs
- **self.cash_flow.kap_cost** (pandas.DataFrame) – DataFrame pf capital costs

run_cashflow()

Run main projects cash flow.

- Run projects cashflow in **cash_flow.py**
- Rank projects by projects that had drilling in the previous model (or history) year, then NPV
- Make sure there are no duplicated undiscovered projects in the table

Returns
self.projects – Master DataFrame containing all projects to be processed in current model year

Return type
pandas.DataFrame

determine_eor_eligibility()

Determine which EOR/ASR projects are eligible to be run in each iteration based on economic life of legacy projects and profitability.

- CO2 EOR can only run within 10 years of legacy project abandonment, while other EOR projects can only run within 6 years
- Once we have a list of eligible projects based on economic life aggregate the most profitable, eligible, selected projects
- Set all other EOR/ASR projects tied to the same producing project (excluding infill) ineligible (there can't be multiple EOR processes operating on the same well)

Returns
self.projects – Master DataFrame containing all projects to be processed in current model year

Return type
pandas.DataFrame

apply_rig_constraints()

Apply rig constraints to projects based on project npv rank.

- Map project well counts against regional wells/rig calculated in preprocessors
- Assign two rig constraints, total and regional
- There's some flexibility in the regional constraints to represent rigs moving around, but no flexibility in the total constraint
- Select projects until rig count > rig constraint
- Only 85% of total rigs go to developing projects, the rest goes to exploration

Returns
self.projects – Master DataFrame containing all projects to be processed in current model year

Return type
pandas.DataFrame

apply_footage_constraints()

Apply footage constraints to projects based on project npv rank.

- Assign two footage constraints, total and regional
- There's some flexibility in the regional constraints to represent rigs moving around, but no flexibility in the total constraint
- Select projects until footage > footage constraint
- Only 85% of total rigs go to developing projects, the rest goes to exploration

Returns
self.projects – Master DataFrame containing all projects to be processed in current model year

Return type
pandas.DataFrame

select_projects()

Select projects that are economical to produce in the current run iteration.

1. **Create all the relevant constraint and run masks:**
    a. npv_mask: If a project has positive npv it is eligible to be selected
    b. past_drilling_mask: If a project started drilling, keep drilling

c. max_wells_mask: If a project hits 70% of max wells, revert to calculating based on economics (overrides past_drilling_mask)

d. proc_code_mask: Only apply past drilling mask to developing primary production projects

e. reg_rig_mask: Regional rig constraint mask

f. nat_rig_mask: National rig constraint mask

g. reg_footage_mask: Regional footage constraint mask

h. nat_footage_mask: National footage constraint mask

2. Apply capital constraint and apply capital constraint mask
3. Apply selected mask to relevant production to get production and well values
4. Generate annual project output tables
5. Update iterative values (i.e. past_wells)
6. Remove projects that are no longer eligible from production (i.e. exhausted discovered vertical projects) from the master projects table

Returns

- **self.projects** (pandas.DataFrame) – Master DataFrame containing all projects to be processed in current model year
- **self.crude_production** (pandas.DataFrame) – DataFrame of onshore crude oil production
- **self.natgas_production** (pandas.DataFrame) – DataFrame of onshore natural gas production
- **self.ngpl_production** (pandas.DataFrame) – DataFrame of onshore natural gas plant liquids production
- **self.ngpl_ethane_production** (pandas.DataFrame) – DataFrame of onshore ethane production
- **self.ngpl_propane_production** (pandas.DataFrame) – DataFrame of onshore propane production
- **self.ngpl_butane_production** (pandas.DataFrame) – DataFrame of onshore butane production
- **self.ngpl_isobutane_production** (pandas.DataFrame) – DataFrame of onshore isobutane production
- **self.ngpl_proplus_production** (pandas.DataFrame) – DataFrame of onshore pentanes production
- **self.wells** (pandas.DataFrame) – DataFrame of onshore wells
- **self.dryholes** (pandas.DataFrame) – DataFrame of onshore wells

debug_onshore()

Produce debug outputs for Onshore Submodule.

Return type

None

write_intermediate_variables()

Write local variables to restart file to be read back in each iteration.

Returns

- **self.parent.hsm_vars.on_projects** (pandas.DataFrame) – Master DataFrame containing all projects to be processed in current model year
- **self.parent.hsm_vars.on_projects_undiscovered** (pandas.DataFrame) – DataFrame of undiscovered projects
- **self.parent.hsm_vars.on_projects_discovered** (pandas.DataFrame) – DataFrame of discovered projects that have not yet been assigned to the master project df (duplicates)
- **self.parent.hsm_vars.on_crude_production** (pandas.DataFrame) – DataFrame of onshore crude oil production
- **self.parent.hsm_vars.on_natgas_production** (pandas.DataFrame) – DataFrame of onshore natural gas production
- **self.parent.hsm_vars.on_ngpl_production** (pandas.DataFrame) – DataFrame of onshore NGPL production
- **self.parent.hsm_vars.on_ngpl_ethane_production** (pandas.DataFrame) – DataFrame of onshore ethane production
- **self.parent.hsm_vars.on_ngpl_propane_production** (pandas.DataFrame) – DataFrame of onshore propane production
- **self.parent.hsm_vars.on_ngpl_butane_production** (pandas.DataFrame) – DataFrame of onshore butane production
- **self.parent.hsm_vars.on_ngpl_isobutane_production** (pandas.DataFrame) – DataFrame of onshore isobutane production
- **self.parent.hsm_vars.on_ngpl_proplus_production** (pandas.DataFrame) – DataFrame of onshore pentanes production
- **self.parent.hsm_vars.on_wells** (pandas.DataFrame) – DataFrame of onshore wells
- **self.parent.hsm_vars.on_producing_wells** (pandas.DataFrame) – DataFrame of onshore legacy producing project wells
- **self.parent.hsm_vars.on_producing_footage** (pandas.DataFrame) – DataFrame of osnhore legacy producing project footage
- **self.parent.hsm_vars.on_dryholes** (pandas.DataFrame) – DataFrame of onshore dryholes
- **self.parent.hsm_vars.on_exploratory_wells** (pandas.DataFrame) – DataFrame of onshore exploratory wells
- **self.parent.hsm_vars.on_co2_injected** (pandas.DataFrame) – DataFrame of onshore CO2 used
- **self.parent.hsm_vars.on_co2_recycled** (pandas.DataFrame) – DataFrame of onshore CO2 recycled

report_results_unf()

Report results to restart variables and produce debug files.

Returns

- **self.restart.pmmout_rfqtdcrd** (pandas.DataFrame) – Total crude production by HSM region
- **self.restart.pmmout_rfqdcrd** (pandas.DataFrame) – Total crude oil production by HSM region (not including EOR)
- **self.restart.ogsmout_ogqcrrep** (pandas.DataFrame) – Crude oil production by oil category
- **self.restart.ogsmout_ogcoprd** (pandas.DataFrame) – Crude oil production by lower 48 region
- **self.restart.ogsmout_ogqshloil** (pandas.DataFrame) – Crude oil production by select tight oil play
- **self.restart.ogsmout_ogoilprd** (pandas.DataFrame) – Crude oil production by oil type and HSM district
- **self.restart.ogsmout_ogcrdprd** (pandas.DataFrame) – Crude oil production by HSM region and crude type
- **self.restart.ogsmout_ogcruderef** (pandas.DataFrame) – Crude oil production by LFMM crude oil type and region
- **self.restart.ogsmout_ogcrdheat** (pandas.DataFrame) – Heat rate by type of crude oil
- **self.restart.ogsmout_ogeorprd** (pandas.DataFrame) – CO2 EOR crude oil production
- **self.restart.ogsmout_ogenagprd** (pandas.DataFrame) – Natural gas expected production by natural gas type and HSM district
- **self.restart.ogsmout_ogrnagprd** (pandas.DataFrame) – Natural gas realized production by natural gas type and HSM district
- **self.restart.ogsmout_ogadgprd** (pandas.DataFrame) – Natural gas associated dissolved production by oil type and HSM district
- **self.restart.ogsmout_ogprdad** (pandas.DataFrame) – Natural gas associated dissolved production by HSM region
- **self.restart.ogsmout_ogqshlgas** (pandas.DataFrame) – Natural gas production by select natural gas play
- **self.restart.ogsmout_ogqngrep** (pandas.DataFrame) – Natural gas production by natural gas type
- **self.restart.ogsmout_ogprdugr** (pandas.DataFrame) – Lower 48 unconventional natural gas production
- **self.restart.ogsmout_ogregprd** (pandas.DataFrame) – Total crude oil and natural gas production by production type
- **self.restart.ogsmout_ogcowhp** (pandas.DataFrame) – Crude oil wellhead price by HSM region
- **self.restart.ogsmout_ogpcrwhp** (pandas.DataFrame) – Crude oil HSM average wellhead price
- **self.restart.ogsmout_ogngwhp** (pandas.DataFrame) – Natural gas wellhead price by HSM region
- **self.restart.ogsmout_ogogwells** (pandas.DataFrame) – Total wells
- **self.restart.ogsmout_ognowell** (pandas.DataFrame) – Total completed wells

- **self.restart.ogsmout_ogwellsl48** (pandas.DataFrame) – Total lower 48 wells
- **self.restart.ogsmout_ogsrl48** (pandas.DataFrame) – Lower 48 drilling success rates
- **self.restart.ogsmout_ogngplprd** (pandas.DataFrame) – NGPL production by HSM district
- **self.restart.ogsmout_ogngplet** (pandas.DataFrame) – Ethane production by HSM district
- **self.restart.ogsmout_ogngplpr** (pandas.DataFrame) – Propane production by HSM district
- **self.restart.ogsmout_ogngplbu** (pandas.DataFrame) – Butane production by HSM district
- **self.restart.ogsmout_ogngplis** (pandas.DataFrame) – Isobutane production by HSM district
- **self.restart.ogsmout_ogngplpp** (pandas.DataFrame) – Pentanes production by HSM district
- **self.restart.ogsmout_ogco2rec** (pandas.DataFrame) – CO2 recycled by HSM region and CO2 type
- **self.restart.ogsmout_ogco2inj** (pandas.DataFrame) – CO2 injected by HSM region and CO2 type
- **self.restart.ogsmout_ogco2pur** (pandas.DataFrame) – CO2 purchased by HSM region and CO2 type
- **self.restart.ogsmout_ogco2avl** (pandas.DataFrame) – CO2 available by HSM region and CO2 type
- **self.restart.ogsmout_ogco2prc** (pandas.DataFrame) – CO2 price by HSM region and CO2 type
- **self.restart.ogsmout_ogtechon** (pandas.DataFrame) – HSM technology improvement rate

offshore — Lower 48 Offshore Submodule

offshore.py implements the Offshore class. It models OCS fields through their complete lifecycle using a three-component structure: undiscovered fields, discovered undeveloped fields, and producing fields.

This reference page lists module symbols and docstrings. For model design, assumptions, economics, data inputs, and output interpretation, see *Lower 48 Offshore Submodule*. Submodule for calculating offshore well production.

Summary

This submodule provides a field-level micro-economic model for characterizing the oil and gas resource potential in the United States Outer Continental Shelf. The model currently handles the Western and Central Gulf of America, Eastern Gulf of America, Atlantic OCS, and Pacific OCS. The module operates as follows:

1. The module is initialized in the onshore class *__init__* method, with class dataframes and variables being declared here.
2. The *setup* method is called, which reads in the necessary input files via .csv or .pkl (see **intermediate_var_pickle.py**) format. From *setup* the following methods are called to load in different input variable classes:
   a. *load_input_variables* - Reads variable data from self.setup_table
   b. *load_input_tables* - Reads Offshore input tables in as DataFrames

c. *load_announced* - Performs one-time setup for announced fields

d. *run_producing_resources* - Performs one-time calculation for producing field resources

e. *run_producing_properties* - Performs one-time calculation for producing field properties

f. *run_producing_production* - Performs one-time production calculation for producing fields

g. *determine_undiscovered_fields* - Calculates distribution of undiscovered fields

h. *calculate_drilling_capacity* - Determine model drilling capacity by rig type

i. *side_case_adjustments* - Adjust U=undiscovered project EURs for Oil and Gas Supply Cases

3. The *run* method is called, which kicks off the main model functions.
4. Technically recoverable resources are calculated for discovered fields in *calculate_resources*.
5. New field wildcats are calculated for each evaluation unit in *determine_nfws*.
6. Newly discovered fields and their properties are calculated.
7. In *load_fields* newly discovered fields and announced fields are added to the main self.fields DataFrame for processing.
8. Calculate crude oil and natural gas by region based on price values from LFMM and NGMM in *load_prices*.
9. For the loaded fields, calculate annual drilling in *calculate_drilling*.
10. Calculate exploration wells and costs for loaded fields in *calculate_exploration_cost*.
11. For loaded fields, calculate the number of delay years between drilling being discovered and being eligible for production in *calculate_delay_years*.
12. Calculate various costs in *calculate_platform_costs*, *calculate_development_cost*, and *calculate_operating_cost*, and load these costs into the cashflow.
13. Load cashflow in *load_cash_flow_properties* and *load_cash_flow_production*.
14. Run the cashflow
15. Rank fields based on profitability, apply drilling constraints and select fields in *rank_fields* and *select_fields*.
16. Expand drilling constraints if they are exceeded in *expand_drilling_capacity*
17. Calculate ngpl volumes as proportion of natural gas production
18. Run *write_intermediate_variables* to store iterative calculation tables for the next model run year. These tables are written out to .pkl files in **intermediate_var_pickle.py**.
19. *report_results_unf* is called from **module_unf.py** to report results to debug files and the restart file.

Input files

- off_mapping.csv - Offshore Mapping
- off_water_depth.csv - Water Depth Mapping
- off_drill_depth.csv - Drill Depth Mapping
- off_delineation_wells.csv - Delineation Drilling Mapping
- off_exp_cost.csv - Exploration Cost
- off_dev_cost.csv - Development Cost
- off_development_wells.csv - Development Drilling Mapping

- off_delays.csv - Delays Based On Depth And Wells
- off_platform_delays.csv - Delays Based On Depth And Platform Type
- off_platform_slots.csv - Slots By Water Depth And FSC
- off_platform_structure_type.csv - Platform Type By Water Depth And FSC
- off_platform_abandonment.csv - Abandonment Fraction By Facility Type
- off_announced_fields.csv - Announced Fields
- off_drill_per_year.csv - Drilling Per Year
- off_platform_drill_per_year.csv - Drilling Per Year And Platform Type
- off_field_size_classes.csv - Standard Field Size Class Resources
- off_operating_cost.csv - Operating Costs
- off_gas_ratio_and_cond.csv - Gas Ratios And Condensate Ratios
- off_transportation.csv - Gas And Oil Transportation Costs/Unit
- off_royalty - self.royalty_rates = pd.DataFrame() #: DataFrame of royalty rates
- off_prod_fac_cost_frac.csv - Fraction Of Distributed Facility Costs
- off_producing_fields.csv - Producing Fields
- off_prod_profile.csv - Production Profile Params For Producing Fields
- off_1990_fields.csv - Fields Discovered Before 1990
- off_cumulative_nfws.csv - New Field Wildcats At Zero Year
- off_undiscovered_fields.csv - Undiscovered Fields After 1990
- off_field_availability.csv - Fields Availability By Evaluation Unit
- off_nfw_coefficients.csv - NFW Drilling Rate Coefficients
- off_discovery_coefficients.csv - Discovery Coefficients
- off_undiscovered_production.csv - Undiscovered Production Parameters
- off_exp_success_rate.csv - Exploration Success Rate

Model functions and class methods

class Offshore(sub.Submodule) - Offshore submodule for HSM

- __init__ - Constructor to initialize Offshore submodule
- setup - Method to set up Offshore Submodule for HSM
- load_input_variables - Method to read input variable data
- load_input_tables - Method to read table data from input files
- load_announced - Method to perform one-time setup for announced fields
- run_producing_resources - Method to perform one-time calculation for producing field resources
- run_producing_properties - Method to perform one-time calculation for producing field properties
- run_producing_production - Method to perform one-time production calculation for producing fields
- determine_undiscovered_fields - Method to calculate distribution of undiscovered fields
- run - Method to run Offshore module for HSM

- load_prices - Method to calculate average prices for Offshore regions
- calculate_drilling_capacity - Method to determine drilling capacity by rig type
- calculate_resources - Method to calculate technically recoverable resources based on available undiscovered fields
- determine_nfws - Method to calculate number of new field wildcats drilling in each evaluation unit
- calculate_new_fields - Method to calculate number of new fields discovered based on new field wildcat drilling
- calculate_new_field_properties - Method to calculate properties (e.g. drilling depth, water depth) for discovered fields
- load_fields - Method to load available announced and discovered fields for economic analysis
- calculate_exploration_cost - Method, for the loaded fields, to calculate exploration cost
- calculate_delay_years - Method, for the loaded fields, to calculate the number of delay years
- calculate_platform_cost - Method, for the loaded fields, to calculate platform, flowline, and abandonment costs
- calculate_drilling - Method, for the loaded fields, to calculate yearly drilling
- calculate_development_cost - Method, for the loaded fields, to calculate development drilling costs
- calculate_operating_cost - Method, for the loaded fields, to calculate operating costs
- load_cash_flow_properties - Method, for the loaded fields, to load CashFlow properties in preparation for NPV/DCF calculation
- load_cash_flow_production - Method, for the loaded fields, to load CashFlow object production in preparation for NPV/DCF calculation
- run_cash_flow - Method, for the loaded fields, to run through CashFlow
- rank_fields - Method to rank Fields by net present value
- select_fields - Method to select fields based on rank, net present value and rig capacity
- expand_drilling_capacity - Method to develop new drilling capacity based on rig utilization
- calculate_ngpls - Method to calculate production of NGPLs
- report_results_hdf/unf - Method to report results to restart variables

Output debug files

- off_discovery_disc.csv - Offshore discovered field distribution by evaluation unit per year
- fields.csv - Multiple resources and rates
- off_drilling_rig_constraint.csv - Constraints by type of drilling rig
- off_fields.csv - Discovered and producing fields by evaluation unit per year
- off_fields_selected.csv - Multiple resources and rates
- off_cumulative_nfws.csv - Offshore cumulative new field wildcats by evaluation unit per year

Output restart variables

REAL OGPRDOFF(6,2,MNUMYR) ! Lower 48 offshore production split into Federal/State categories REAL OGQCRREP(MNOGCRO,MNUMYR) ! CRUDE PRODUCTION BY OIL CAT REAL OGQNGREP(MNOGCAT,MNUMYR) ! NG PRODUCTION BY GAS CAT REAL OGNGPLBU(OGDIST,MNUMYR) ! Butane production REAL OGNGPLET(OGDIST,MNUMYR) ! Ethane production REAL OGNGPLIS(OGDIST,MNUMYR) ! Isobutane production REAL OGNGPLPP(OGDIST,MNUMYR) ! Pentanes plus production REAL OGNGPLPR(OGDIST,MNUMYR) ! Propane production REAL OGNGPLPRD(OGDIST,MNUMYR)

! Natural gas plant output by NGPL region REAL OGNOWELL(MNUMYR) ! WELLS COMPLETED REAL OGOILPRD(OGDIST,OILTYPES,MNUMYR) ! crude oil production

Offshore submodule class methods

class offshore.**Offshore**(*parent*)

Bases: *Submodule*

Offshore submodule for HSM.

Parameters
parent (*module_unf.Module*) – Pointer to head module

mapping
DataFrame of mapping

water_depth
DataFrame of water depth mapping

drill_depth
DataFrame of drill depth mapping

delineation_wells
DataFrame of delineation drilling mapping

exp_cost
DataFrame of exploration cost

dev_cost
DataFrame of development cost

development_wells
DataFrame of development drilling mapping

delays
DataFrame of delays based on depth and wells

platform_delays
DataFrame of delays based on depth and platform type

platform_slots
DataFrame of slots by water depth and FSC

platform_structure_type
DataFrame of platform type by water depth and FSC

platform_abandonment
DataFrame of abandonment fraction by facility type

announced_fields
DataFrame of announced fields

drill_per_year
DataFrame of drilling per year

platform_drill_per_year
DataFrame of drilling per year and platform type

drill_rig_constraint
DataFrame of drilling rig constraints

field_size_classes
DataFrame of standard field size class resources

operating_cost
DataFrame of operating costs

gas_ratio_and_cond
DataFrame of gas ratios and condensate ratios

transportation
DataFrame of gas and oil transportation costs/unit

royalty_rates
DataFrame of royalty rates

prod_fac_cost_frac
DataFrame of fraction of distributed facility costs

producing_fields
DataFrame of producing fields

prod_profile
DataFrame of production profile params for producing fields

off_1990_fields
DataFrame of fields Discovered before 1990

cumulative_nfws
DataFrame of new field wildcats at zero year

undiscovered_fields
DataFrame of fields undiscovered fields after 1990

field_availability
DataFrame of fields availability by evaluation unit

nfw_coefficients
DataFrame of nfw drilling rate coefficients

discovery_coefficients
DataFrame of discovery coefficients

undiscovered_production
DataFrame of undiscovered production parameters

exp_success_rate
DataFrame of exploration success rate

off_reg_crude_price
DataFrame of fixed model iteration 1 crude prices to stop oscillations in NEMS

off_reg_natgas_price
DataFrame of fixed model iteration 1 natural gas prices to stop oscillations in NEMS

fields
DataFrame of active fields

fields_schedule
DataFrame of active fields

total_fields
DataFrame of total count of fields by fsc

discovered_field_dist
DataFrame of total count of discovered fields by fsc (distribution)

discovered_fields
DataFrame of discovered fields in queue for cash flow

technically_recoverable
DataFrame technically recoverable resources

fields_selected
DataFrame of selected fields

crude_production
DataFrame of all crude production

natgas_production
DataFrame of all natural gas production

nagas_production
DataFrame of non-associated natural gas production

adgas_production
DataFrame of associated-dissolved natural gas production

field_crude_production
DataFrame of developing field crude production

field_natgas_production
DataFrame of developing field natgas production

wells
DataFrame of all producing wells

exploratory_wells
DataFrame of all exploratory wells

ad_fraction
DataFrame of ad gas fraction of total gas production

ngpl_production
DataFrame of all ngpl production

ethane_production
DataFrame of all ethane (ngpl) production

propane_production
DataFrame of all propane (ngpl) production

butane_production
DataFrame of all butane (ngpl) production

isobutane_production
DataFrame of all isobutane (ngpl) production

proplus_production

DataFrame of all natural gasoline (ngpl) production

setup(*setup_filename*)

Setup Offshore submodule for HSM.

Parameters

setup_filename (str) – Path to offshore setup file.

Return type

None

load_input_variables()

Reads variable data from self.setup_table (loaded in super().setup(setup_filename))

Returns

- **self.zero_year** (int) – First model year
- **self.averaging_years** (int) – Number of averaging years used to produce crude oil and natural gas prices
- **self.exp_success_improve_rate** (float) – Technology improvement rate for exploration success rate
- **self.exp_delay_improve_rate** (float) – Technology improvement rate for exploration delay period
- **self.drill_cost_improve_rate** (float) – Technology improvement rate for drilling costs
- **self.delay_improve_rate** (float) – Technology improvement rate for nfw delays
- **self.platform_cost_improve_rate** (float) – Technology improvement rate for platform costs
- **self.operating_cost_improve_rate** (float) – Technology improvement rate for operating costs
- **self.drill_price_ratio** (float) – Price adjustment ratio for drill prices
- **self.pipeline_diameter** (float) – Assumption for pipeline diameter
- **self.wells_per_flowline** (int) – Assumption for wells/flowline
- **self.evaluation_years** (int) – Number of evaluation years (self.final_year - self.zero_year + 1)
- **self.boe_to_mcf** (float) – BOE to MCF conversion factor
- **self.exp_tang_frac** (float) – Fraction of well costs that are tangible
- **self.dev_tang_frac** (float) – Fraction of well costs that are tangible
- **self.kap_tang_frac** (float) – Fraction of capital costs that are tangible
- **self.intang_amor_frac** (float) – Fraction of costs to be amortized
- **self.fed_tax_rate** (float) – Federal tax rate
- **self.amor_schedule** (str) – Amortization schedule code (see **depreciation_schedules.csv**)
- **self.deprec_schedule** (str) – Depreciation schedule code (see **depreciation_schedules.csv**)
- **self.discount_rate** (float) – Discount rate assumption

- **self.boem_assessment_year** (int) – BOEM field size class assessment year (for determining discovered projects)
- **self.platform_cost_year** (int) – Basis year for platform costs
- **self.drill_cost_base** (int) – Drill cost base value used for cost adjustments
- **self.oil_price_cost_base** (float) – Oil price base value used for cost adjustments

load_input_tables()

Reads Offshore input tables in as DataFrames.

Returns

- **self.water_depth** (pandas.DataFrame) – DataFrame of water depth by evaluation unit and field size class
- **self.drill_depth** (pandas.DataFrame) – DataFrame of drill depth by evaluation unit and field size class
- **self.development_wells** (pandas.DataFrame) – DataFrame of development wells by water depth and field size class
- **self.delays** (pandas.DataFrame) – DataFrame of delays between well discovery and drilling by water depth and total number of wells
- **self.platform_delays** (pandas.DataFrame) – DataFrame of delays between project start and drilling based on platform type by platform type and water depth
- **self.platform_slots** (pandas.DataFrame) – DataFrame of platform slots per well by water depth and field size class
- **self.platform_structure_type** (pandas.DataFrame) – DataFrame of well platform structure type by water depth and field size class
- **self.platform_drill_per_year** (pandas.DataFrame) – DataFrame of platform annual drilling by platform type and water depth
- **self.exp_success_rate** (pandas.DataFrame) – DataFrame of exploratory wells success rates by evaluation unit and field size class
- **self.royalty_rates** (pandas.DataFrame) – DataFrame of royalty rates by evaluation unit and field size class
- **self.transportation** (pandas.DataFrame) – DataFrame of transportation costs by evaluation unit
- **self.mapping** (pandas.DataFrame) – DataFrame of HSM mapping by evaluation unit
- **self.prod_fac_cost_frac** (pandas.DataFrame) – DataFrame of capital cost annualized distribution by delay years
- **self.prod_profile** (pandas.DataFrame) – DataFrame of well production profile by evaluation unit and field size class
- **self.undiscovered_production** (pandas.DataFrame) – DataFrame of undiscovered project production attributes by evaluation unit
- **self.producing_fields** (pandas.DataFrame) – DataFrame of producing offshore fields
- **self.off_1990_fields** (pandas.DataFrame) – DataFrame of fields discovered before 1990 by evaluation unit
- **self.cumulative_nfws** (pandas.DataFrame) – DataFrame of cumulative new field wildcats by evaluation unit

- **self.undiscovered_fields** (pandas.DataFrame) – DataFrame of number of undiscovered fields by evaluation unit
- **self.field_availability** (pandas.DataFrame) – DataFrame of when fields are available for drilling by evaluation unit
- **self.nfw_coefficients** (pandas.DataFrame) – DataFrame of new field wildcat drilling coefficients by evaluation unit
- **self.discovery_coefficients** (pandas.DataFrame) – DataFrame of discovery coefficients for discovery algorithim by evaluation unit
- **self.dev_cost** (pandas.DataFrame) – DataFrame of development cost equation coefficients by water depth and drill depth
- **self.field_size_classes** (pandas.DataFrame) – DataFrame of field size class characteristics
- **self.operating_cost** (pandas.DataFrame) – DataFrame of operating cost equation coefficients
- **self.gas_ratio_and_cond** (pandas.DataFrame) – DataFrame of gas to oil ratio for mixed wells by evaluation unit
- **self.drill_rig_constraint** (pandas.DataFrame) – DataFrame of drill rig constraint eq coefficients by rig type and water depth
- **self.exp_cost** (pandas.DataFrame) – DataFrame of exploration cost coefficients by rig type
- **self.platform_abandonment** (pandas.DataFrame) – DataFrame of platform abandonment captial cost ratios by rig type
- **self.announced_fields** (pandas.DataFrame) – DataFrame of announced projects

load_announced()

Performs one-time setup for announced fields.

- Merge to HSM mapping, drilling depths etc.
- Convert water depth from feet to metres
- Declare project type as "announced"

Returns
self.announced_fields – DataFrame of announced projects

Return type
pandas.DataFrame

load_intermediate_variables()

Load intermediate variables for integrated runs from .pkl files.

Returns

- **self.announced_fields** (pandas.DataFrame) – DataFrame of announced projects
- **self.discovered_fields** (pandas.DataFrame) – DataFrame of discovered fields
- **self.discovered_field_dist** (pandas.DataFrame) – DataFrane if discovered fields distribution across evaluation units
- **self.undiscovered_fields** (pandas.DataFrame) – DataFrame of number of undiscovered fields by evaluation unit

- **self.cumulative_nfws** (pandas.DataFrame) – DataFrame of cumulative new field wildcats by evaluation unit
- **self.total_fields** (pandas.DataFrame) – DataFrame of total fields by evaluation unit and field size class
- **self.fields** (pandas.DataFrame) – Main DataFrame of fields with field-level attributes
- **self.fields_selected** (pandas.DataFrame) – DataFrame of fields that have been selected to produce
- **self.crude_production** (pandas.DataFrame) – DataFrame of crude production by field
- **self.natgas_production** (pandas.DataFrame) – DataFrame of total natural gas production by field
- **self.nagas_production** (pandas.DataFrame) – DataFrame of non-associated natural gas production by field
- **self.adgas_production** (pandas.DataFrame) – DataFrame of associated dissolved natural gas production by field
- **self.wells** (pandas.DataFrame) – DataFrame of wells by field
- **self.exploratory_wells** (pandas.DataFrame) – DataFrame of exploratory wells only by field
- **self.drill_rig_constraint** (pandas.DataFrame) – DataFrame of rig constaints by rig type
- **self.field_crude_production** (pandas.DataFrame) – DataFrame of crude production by developing field
- **self.field_natgas_production** (pandas.DataFrame) – DataFrame of natural gas production by developing field

run_producing_resources()

Performs one-time calculation for producing field resources.

- Gets producing project max production and initial production based on history or field characteristics
- Merge field attributes (i.e. evaluation unit, field size class resources)

Returns
self.producing_fields – DataFrame of producing offshore fields

Return type
pandas.DataFrame

run_producing_properties()

Performs one-time calculation for producing field properties.

- Merge producing projects to mapping, production properties, decline rates, etc.

Returns
self.producing_fields – DataFrame of producing offshore fields

Return type
pandas.DataFrame

run_producing_production()

Performs one-time production calculations for producing fields.

- Apply drilling equations to producing projects to project future production

- Split production between oil and natural gas and write to self.crude_production, self.natgas_production self.adgas_production, and self.nagas_production

Returns
self.producing_fields – DataFrame of producing offshore fields

Return type
pandas.DataFrame

determine_undiscovered_fields()

Calculates distribution of undiscovered fields.

- Accounts for discoveries before 1990, and after the BOEM assessment year by subtracting them from BOEM assessment
- Appends all producing and announced wells to the relevant field dfs

Returns

- **self.undiscovered_fields** (pandas.DataFrame) – DataFrame of number of undiscovered fields by evaluation unit
- **self.total_fields** (pandas.DataFrame) – DataFrame of total fields by evaluation unit and field size class
- **self.discovered_field_dist** (pandas.DataFrame) – DataFrame of discovered fields distribution across evaluation units

calculate_drilling_capacity()

Determine model drilling capacity by rig type.

Returns
self.drill_rig_constraint – DataFrame of rig constaints by rig type

Return type
pandas.DataFrame

side_case_adjustments()

Adjust undiscovered project EURs for Oil and Gas Supply Cases.

Returns
self.undiscovered_production – DataFrame of undiscovered project production attributes by evaluation unit

Return type
pandas.DataFrame

run()

Run Offshore Submodule for HSM.

Return type
None

fix_prices()

Fix natural gas prices for the offshore submodule in the first model iteration to stop oscillations with NGMM

Returns

- **self.off_reg_natgas_price** (pandas.DataFrame) – Fixed iteration 1 natural gas prices
- **self.off_reg_crude_price** (pandas.DataFrame) – Fixed iteration 1 crude prices

calculate_resources()

Calculate technically recoverable resources based on available undiscovered fields.

- Sum undiscovered project resource volumes by field area

Returns
self.technically_recoverable – DataFrame of technically recoverable resources by field

Return type
pandas.DataFrame

determine_nfws()

Calculate number of new fields wildcats drilling in each evaluation unit.

The new field wildcat drilling volume is based on the lagged prices of crude oil and natural gas.

Returns
self.cumulative_nfws – DataFrame of cumulative new field wildcats by evaluation unit

Return type
pandas.DataFrame

calculate_new_fields()

Calculate number of new fields discovered based on new field wildcat drilling.

Using Arps-Roberts discovery algorithm. The discovery coefficient (self.discovery_coefficients) is scaled by the number of total fields (self.total_fields).

Returns

- **self.discovered_field_dist** (pandas.DataFrame) – DataFrame of discovered fields distribution across evaluation units
- **self.discovered_fields** (pandas.DataFrame) – DataFrame of discovered fields

calculate_new_field_properties()

Assigns properties (e.g. drilling depth, water depth) for discovered fields.

Properties are assigned based upon field evaluation unit.

Returns
self.discovered_fields – DataFrame of discovered fields

Return type
pandas.DataFrame

load_fields()

Load available announced and discovered fields for economic analysis.

Returns
self.fields – Main DataFrame of fields with field-level attributes

Return type
pandas.DataFrame

load_prices()

Calculate average prices for Offshore regions.

Averages prices over the years set in averaging_years. Currently set to only use current year data.

Returns
self.fields – Main DataFrame of fields with field-level attributes

Return type
pandas.DataFrame

calculate_drilling()

For the loaded fields, calculate annual drilling.

- Fields are merged to platform slots, platform delays, and drilling/year
- Drilling/year is multiplied by available platform slots

Returns
self.fields – Main DataFrame of fields with field-level attributes

Return type
pandas.DataFrame

calculate_exploration_cost()

Calculate exploration wells and associated costs.

- Well-level exploration costs are calculated
- Delineation wells and exploration success rates are merged to the cost calculation DataFrame
- Number of exploratory wells is calculated
- Exploration costs are distributed based on delineation and exploratory wells drilled

Returns
self.fields – Main DataFrame of fields with field-level attributes

Return type
pandas.DataFrame

calculate_delay_years()

For the loaded fields, calculate the number of delay years between drilling being discovered and being eligible for production.

Returns
self.fields – Main DataFrame of fields with field-level attributes

Return type
pandas.DataFrame

calculate_platform_cost()

For the loaded fields, calculate platform, abandonment, and flowline costs.

Cost equations transferred from hard-coded cost equations in OGSM.

Returns
self.fields – Main DataFrame of fields with field-level attributes

Return type
pandas.DataFrame

calculate_development_cost()

For the loaded fields, calculate development drilling costs.

Returns
self.fields – Main DataFrame of fields with field-level attributes

Return type
pandas.DataFrame

calculate_operating_cost()

For the loaded fields, calculate operating costs.

Returns
self.fields – Main DataFrame of fields with field-level attributes

Return type
pandas.DataFrame

load_cash_flow_properties()

For the loaded fields, load CashFlow properties in preparation for NPV/DCF calculation.

Returns

- **self.cash_flow.properties** (pandas.DataFrame) – DataFrame containing properties used in the cash flow (costs, tangible/intangible cost ratios, etc.)
- **self.cash_flow.exp_drill_cost** (pandas.DataFrame) – DataFrame of exploratory well drill costs
- **self.cash_flow.dev_drill_cost** (pandas.DataFrame) – DataFrame of development well drill costs
- **self.cash_flow.operating_cost** (pandas.DataFrame) – DataFrame of operating costs
- **self.cash_flow.kap_cost** (pandas.DataFrame) – DataFrame pf capital costs
- **self.fields** (pandas.DataFrame) – Main DataFrame of fields with field-level attributes

load_cash_flow_production()

For the loaded fields, load CashFlow object production in preparation for NPV/DCF calculation.

Returns

- self.cash_flow.crude_production – DataFrame of onshore crude oil production
- **self.cash_flow.natgas_production** (pandas.DataFrame) – DataFrame of onshore natural gas production
- **self.fields** (pandas.DataFrame) – Main DataFrame of fields with field-level attributes

run_cash_flow()

For the loaded fields, run through CashFlow.

Returns
self.fields – Main DataFrame of fields with field-level attributes

Return type
pandas.DataFrame

rank_fields()

Rank Fields by net present value.

Returns
self.fields – Main DataFrame of fields with field-level attributes

Return type
pandas.DataFrame

select_fields()

Select fields based on rank, net present value and rig capacity.

- Run through drilling constraints to determine if there are sufficient rigs available to drill all profitable projects

- Mask for projects that fall under rig constratints, and apply production volumes and well counts to relevant aggregate dfs (i.e. self.crude_production, self.natgas_production)

Returns

- **self.fields** (pandas.DataFrame) – Main DataFrame of fields with field-level attributes
- **self.crude_production** (pandas.DataFrame) – DataFrame of crude production by field
- **self.natgas_production** (pandas.DataFrame) – DataFrame of onshore natural gas production
- **self.adgas_production** (pandas.DataFrame) – DataFrame of associated dissolved natural gas production by field
- **self.nagas_production** (pandas.DataFrame) – DataFrame of non-associated natural gas production by field
- **self.wells** (pandas.DataFrame) – DataFrame of wells by field
- **self.fields_selected** (pandas.DataFrame) – DataFrame of fields that have been selected to produce

expand_drilling_capacity()

Develop new drilling capacity based on rig utilization.

- If profitable fields exceed rig constraints, add more rigs for next model year

Returns

self.drill_rig_constraint – DataFrame of rig constaints by rig type

Return type

pandas.DataFrame

calculate_ngpls()

Calculate NGPL production volumes based on natural gas production factor.

Returns

- **self.ngpl_production** (pandas.DataFrame) – DataFrame of onshore natural gas plant liquids production
- **self.ngpl_ethane_production** (pandas.DataFrame) – DataFrame of onshore ethane production
- **self.ngpl_propane_production** (pandas.DataFrame) – DataFrame of onshore propane production
- **self.ngpl_butane_production** (pandas.DataFrame) – DataFrame of onshore butane production
- **self.ngpl_isobutane_production** (pandas.DataFrame) – DataFrame of onshore isobutane production
- **self.ngpl_proplus_production** (pandas.DataFrame) – DataFrame of onshore pentanes production

debug_offshore()

Produce debug outputs for Offshore Submodule.

Return type

None

write_intermediate_variables()

Write local variables to restart file to be read back in each iteration.

Returns

- **self.parent.hsm_vars.off_announced_fields** (pandas.DataFrame) – DataFrame of announced projects
- **self.parent.hsm_vars.off_discovered_fields** (pandas.DataFrame) – DataFrame of discovered fields
- **self.parent.hsm_vars.off_discovered_field_dist** (pandas.DataFrame) – DataFrame of discovered fields distribution across evaluation units
- **self.parent.hsm_vars.off_undiscovered_fields** (pandas.DataFrame) – DataFrame of number of undiscovered fields by evaluation unit
- **self.parent.hsm_vars.off_cumulative_nfws** (pandas.DataFrame) – DataFrame of cumulative new field wildcats by evaluation unit
- **self.parent.hsm_vars.off_fields** (pandas.DataFrame) – Main DataFrame of fields with field-level attributes
- **self.parent.hsm_vars.off_fields_selected** (pandas.DataFrame) – DataFrame of fields that have been selected to produce
- **self.parent.hsm_vars.off_total_fields** (pandas.DataFrame) – DataFrame of total fields by evaluation unit and field size class
- **self.parent.hsm_vars.off_crude_production** (pandas.DataFrame) – DataFrame of crude production by field
- **self.parent.hsm_vars.off_natgas_production** (pandas.DataFrame) – DataFrame of onshore natural gas production
- **self.parent.hsm_vars.off_nagas_production** (pandas.DataFrame) – DataFrame of non-associated natural gas production by field
- **self.parent.hsm_vars.off_adgas_production** (pandas.DataFrame) – DataFrame of associated dissolved natural gas production by field
- **self.parent.hsm_vars.off_wells** (pandas.DataFrame) – DataFrame of wells by field
- **self.parent.hsm_vars.off_exploratory_wells** (pandas.DataFrame) – DataFrame of exploratory wells only by field
- **self.parent.hsm_vars.off_drill_rig_constraint** (pandas.DataFrame) – DataFrame of rig constaints by rig type
- **self.field_crude_production** (pandas.DataFrame) – DataFrame of crude production by developing field
- **self.field_natgas_production** (pandas.DataFrame) – DataFrame of natural gas production by developing field

report_results_unf()

Report results to restart variables.

Returns

- **self.restart.ogsmout_ogprdoff** (pandas.DataFrame) – Lower 48 offshore production split into Federal/State categories

- **self.restart.ogsmout_ogqcrrep** (pandas.DataFrame) – Crude oil production by oil category
- **self.restart.ogsmout_ogcrdprd** (pandas.DataFrame) – Crude oil production by HSM region and crude type
- **self.restart.pmmout_rfqtdcrd** (pandas.DataFrame) – Total crude production by HSM region
- **self.restart.pmmout_rfqdcrd** (pandas.DataFrame) – Total crude oil production by HSM region (not including EOR)
- **self.restart.ogsmout_ogcoprd** (pandas.DataFrame) – Crude oil production by lower 48 region
- **self.restart.ogsmout_ogcruderef** (pandas.DataFrame) – Crude oil production by LFMM crude oil type and region
- **self.restart.ogsmout_ogngprd** (pandas.DataFrame) – Natural Gas production by natural gas category
- **self.restart.ogsmout_ogqngrep** (pandas.DataFrame) – Natural gas production by natural gas type
- **self.restart.ogsmout_ogdngprd** (pandas.DataFrame) – Dry natural gas production by state/district and gas type
- **self.restart.ogsmout_ogenagprd** (pandas.DataFrame) – Natural gas expected production by natural gas type and HSM district
- **self.restart.ogsmout_ogrnagprd** (pandas.DataFrame) – Natural gas realized production by natural gas type and HSM district
- **self.restart.ogsmout_ogadgprd** (pandas.DataFrame) – Natural gas associated dissolved production by oil type and HSM district
- **self.restart.ogsmout_ogprdad** (pandas.DataFrame) – Natural gas associated dissolved production by HSM region
- **self.restart.ogsmout_ogprdadof** (pandas.DataFrame) – Offshore AD natural gas production (BCF)
- **self.restart.ogsmout_ogcoprdgom** (pandas.DataFrame) – Crude oil production Gulf of America
- **self.restart.ogsmout_ogngprdgom** (pandas.DataFrame) – Natural gas production Gulf of America
- **self.restart.ogsmout_ogcowhp** (pandas.DataFrame) – Crude oil wellhead price by HSM region
- **self.restart.ogsmout_ogngwhp** (pandas.DataFrame) – Natural gas wellhead price by HSM region
- **self.restart.ogsmout_ogngplprd** (pandas.DataFrame) – NGPL production by HSM district
- **self.restart.ogsmout_ogngplet** (pandas.DataFrame) – Ethane production by HSM district
- **self.restart.ogsmout_ogngplpr** (pandas.DataFrame) – Propane production by HSM district

- **self.restart.ogsmout_ogngplbu** (pandas.DataFrame) – Butane production by HSM district
- **self.restart.ogsmout_ogngplis** (pandas.DataFrame) – Isobutane production by HSM district
- **self.restart.ogsmout_ogngplpp** (pandas.DataFrame) – Pentanes production by HSM district
- **self.restart.ogsmout_ognowell** (pandas.DataFrame) – Total completed wells
- **self.restart.ogsmout_ogoilprd** (pandas.DataFrame) – Crude oil production by oil type and HSM district

alaska — Alaska Crude Oil Submodule

alaska.py implements the Alaska class. It projects crude oil production from three Alaskan regions using named known projects and an undiscovered resource pool.

This reference page lists module symbols and docstrings. For model design, assumptions, economics, data inputs, and output interpretation, see *Alaska Crude Oil Submodule*. Submodule for calculating Alaskan well production.

Summary

This submodule takes field size classes, known projects and undiscovered projects and uses these datasets and equations to project Alaskan production. The Module operates as follows:

1. The module is initialized in the alaska class *__init__* method, with class dataframes and variables being declared here.
2. The *setup* method is called, which reads in the necessary input files via .csv or .pkl (see **intermediate_var_pickle.py**) format.
3. The *run* method is called, which kicks off the main model functions.
4. In the first model run year the *setup_production* method is run, which determines the initial production rate for currently producing projects, and applies the drilling algorithm located in **drilling_equations.py** to project future production for these projects.
5. Run *load_projects*, which loads announced projects into the main project dataframe by model year. Producing projects are concatenated to the master "projects" dataframe in the first model year.
6. Calculate crude oil, natural gas, and ngpl prices by region based on price values from LFMM and NGMM in *load_prices*.
7. Run *calculate_drilling*, where project drilling is determined based on a static drilling schedule
8. Run *calculate production*, which calculates Alaska production based on production profile equations in **drilling_equations.py**.
9. Calculate capital costs, abandon rate and operating costs by project size based on static cost tables.
10. Load and run cash flow in *load_cashflow* and *run_cashflow*. Rank projects by project net present value.
11. Select profitable projects in *select_projects* and write production volumes to output tables.
12. Calculate NGPL volumes based on a ratio of crude production.
13. Run *write_intermediate_variables* to store iterative calculation tables for the next model run year. These tables are written out to .pkl files in **intermediate_var_pickle.py**.
14. *report_results_unf* is called from **module_unf.py** to report results to debug files and the restart file.

Input files

- ak_dev_drill_schedule.csv - Development wells drilled by year assigned to region number

- ak_exp_drill_schedule.csv - Exploratory wells drilled by year assigned to region number
- ak_facility_costs.csv - Capital expenses cost rate by oil resources
- ak_field_size_classes.csv - Field Size Class, Minimum Resources BOE, Production Rate Factors, Ramp Up Years, Decline Rates, Predecline Production Factors, Hyper Decline Coefficients
- ak_mapping.csv - Matches crude tariff price, natural gas tariff price, exploratory drill cost rate, and developmental drill cost rate to region numbers
- ak_opex.csv - Operating costs (e.g., fees for oil and water hauling, facility electricity, etc.)
- ak_projects_known.csv - Oil & Gas Resources, Production Rates, Region Numbers, Production Start Years, Oil Decline Rate, Onshore%, OffSTAT%, OffFED%, Historical Production
- ak_projects_undiscovered.csv - Region Numbers, Oil & Gas Resources

Model functions and class methods

class Alaska(sub.Submodule) - Alaska submodule for HSM

- __init__ - Constructor to initialize Alaska submodule
- setup - Method to set up Alaska submodule for HSM; parameter “setup_filename” - path to offshore setup file
- _setup_production - Method to determine NA drilling and production
- run - Method to run Alaska Submodule for HSM
- _load_projects - Method to load known and unknown projects
- _load_prices - Method to calculate average prices for Offshore regions; averages prices over past years set in averaging_years
- _calculate_drilling - Method to calculate drilling
- _calculate_production - Method to calculate production
- _calculate_kap_cost - Method to calculate other expected cap. costs
- _calculate_operating_cost - Method to calculate operating cost
- _load_cash_flow - Method to load cash flow
- _run_cash_flow - Method for running through CashFlow
- _select_projects - Method to get production from cash flow; selection mask based on profitability
- report_results_unf/hdf - Report results to restart variables

Output debug files

- hsm_ak_crude_projects.csv - Crude Oil production by project
- hsm_ak_ngpls_projects.csv - NGPL production by project

Output restart variables

Crude Oil Restart Variables

- pmmout_rfqtdcrd - Total crude production by HSM region
- pmmout_rfqdcrd - Total crude oil production by HSM region (not including EOR)
- ogsmout_ogqcrrep - Crude oil production by oil category
- ogsmout_ogcrdprd - Crude oil production by HSM region and crude type
- ogsmout_ogcruderef - Crude oil production by LFMM crude oil type and region

Natural Gas Restart Variable

- ogsmout_ogqngrep - Natural gas production by natural gas type

Well Restart Variables

- ogsmout_ogogwells - Total wells
- ogsmout_ognowell - Total completed wells

NGPL Production Restart Variables

- ogsmout_ogngplprd - NGPL production by HSM district
- ogsmout_ogngplet - Ethane production by HSM district
- ogsmout_ogngplpr - Propane production by HSM district
- ogsmout_ogngplbu - Butane production by HSM district
- ogsmout_ogngplis - Isobutane production by HSM district
- ogsmout_ogngplpp - Pentanes production by HSM district

Alaska submodule class methods

class alaska.**Alaska**(*parent*)

Bases: *Submodule*

Alaska submodule for HSM.

Parameters
parent (str) – Module_unf.Module (Pointer to parent module)

setup(*setup_filename*)

Setup Alaska submodule for HSM.

Parameters
setup_filename (str) – Path to offshore setup file.

Return type
None

run()

Run Alaska Submodule for HSM.

Return type
None

setup_production()

Setup production for Alaska Submodule.

Returns

- **self.projects_known** (pandas.DataFrame) – Table of announced Alaska projects
- self.crude_production – DataFrame of alaska crude oil production

load_projects()

Load Alaska projects based on production start date.

Returns
self.projects – Master DataFrame containing all projects to be processed in current model year

Return type
pandas.DataFrame

load_prices()

Calculate average crude oil and natural gas prices for Alaksa wells.

- Oil prices are based on regional wellhead prices from LFMM
- Natural Gas Prices are based on regional wellhead prices from NGMM

Returns

self.projects – Master DataFrame containing all projects to be processed in current model year

Return type

pandas.DataFrame

calculate_drilling()

Calculate Alaska drilling.

- Calculates exploration and development drilling based on static drill schedule
- Assigns drilling costs to cash flow
- Applies technology improvement rate

Returns

- **self.projects** (pandas.DataFrame) – Master DataFrame containing all projects to be processed in current model year
- **self.cash_flow.exp_drill_cost** (pandas.DataFrame) – Table of exploration well drill costs
- **self.cash_flow.exp_dry_cost** (pandas.DataFrame) – Table of exploration well dryhole costs
- **self.cash_flow.dev_drill_cost** (pandas.DataFrame) – Table of development well drill costs
- **self.cash_flow.dev_dry_cost** (pandas.DataFrame) – Table of development well dryhole costs

calculate_production()

Calculates Alaska production based on production profile equations in drilling_equations.py.

Returns

- **self.projects** (pandas.DataFrame) – Master DataFrame containing all projects to be processed in current model year
- **self.cash_flow.crude_production** (pandas.DataFrame) – DataFrame of alaska crude oil production
- **self.cash_flow.natgas_production** (pandas.DataFrame) – DataFrame of alaska natural gas production

calculate_kap_cost()

Calculates capital costs and abandon rate based on a static cost table of capital cost rates by project oil resources.

Returns

- **self.projects** (pandas.DataFrame) – Master DataFrame containing all projects to be processed in current model year
- **self.cash_flow.kap_cost** (pandas.DataFrame) – Table of Alaska project capital costs

calculate_operating_cost()

Calculates project operating costs based on project crude production.

Returns
self.cash_flow.operating_cost – Table of Alaska project operating costs

Return type
pandas.DataFrame

load_cash_flow()

Loads Alaska submodule cash flow.

Returns
self.cash_flow.properties – DataFrame containing properties used in the cash flow (costs, tangible/intangible cost ratios, etc.)

Return type
pandas.DataFrame

run_cash_flow()

Run Alaska Submodule Cash Flow.

Returns
self.projects – Master DataFrame containing all projects to be processed in current model year

Return type
pandas.DataFrame

select_projects()

Select Alaska projects based on project NPV.

Returns
self.projects – Master DataFrame containing all projects to be processed in current model year

Return type
pandas.DataFrame

calculate_ngpls()

Calculate NGPL Production based on a ratio relative to crude oil production.

Returns

- **self.ngpl_production** (pandas.DataFrame) – DataFrame of onshore natural gas plant liquids production
- **self.ngpl_ethane_production** (pandas.DataFrame) – DataFrame of onshore ethane production
- **self.ngpl_propane_production** (pandas.DataFrame) – DataFrame of onshore propane production
- **self.ngpl_butane_production** (pandas.DataFrame) – DataFrame of onshore butane production
- **self.ngpl_isobutane_production** (pandas.DataFrame) – DataFrame of onshore isobutane production
- **self.ngpl_proplus_production** (pandas.DataFrame) – DataFrame of onshore pentanes production

debug_alaska()

Produce debug outputs for Alaska Submodule.

Return type
None

write_intermediate_variables()

Write local variables to restart file to be read back in each iteration.

Returns

- **self.projects** (pandas.DataFrame) – Master DataFrame containing all projects to be processed in current model year
- **self.projects_known** (pandas.DataFrame) – Table of announced Alaska projects
- self.crude_production – DataFrame of alaska crude oil production

report_results_unf()

Report results to restart variables.

Returns

- **self.restart.pmmout_rfqtdcrd** (pandas.DataFrame) – Total crude production by HSM region
- **self.restart.pmmout_rfqdcrd** (pandas.DataFrame) – Total crude oil production by HSM region (not including EOR)
- **self.restart.ogsmout_ogqcrrep** (pandas.DataFrame) – Crude oil production by oil category
- **self.restart.ogsmout_ogoilprd** (pandas.DataFrame) – Crude oil production by oil type and HSM district
- **self.restart.ogsmout_ogcrdprd** (pandas.DataFrame) – Crude oil production by HSM region and crude type
- **self.restart.ogsmout_ogcruderef** (pandas.DataFrame) – Crude oil production by LFMM crude oil type and region
- **self.restart.ogsmout_ogngplprd** (pandas.DataFrame) – NGPL production by HSM district
- **self.restart.ogsmout_ogngplet** (pandas.DataFrame) – Ethane production by HSM district
- **self.restart.ogsmout_ogngplpr** (pandas.DataFrame) – Propane production by HSM district
- **self.restart.ogsmout_ogngplbu** (pandas.DataFrame) – Butane production by HSM district
- **self.restart.ogsmout_ogngplis** (pandas.DataFrame) – Isobutane production by HSM district
- **self.restart.ogsmout_ogngplpp** (pandas.DataFrame) – Pentanes production by HSM district

canada — Canada Natural Gas Submodule

canada.py implements the Canada class. It projects Canadian natural gas production using a regression/decline-curve model calibrated to Canada Energy Regulator (CER) projections.

This reference page lists module symbols and docstrings. For model design, assumptions, economics, data inputs, and output interpretation, see *Canada Natural Gas Submodule*. Submodule for calculating Canadian well production.

Summary

This submodule takes baseline production values, decline curve estimations and well/price sensitivity estimations derived from the Canada Energy Regulator's "Canada Energy Future" report, and uses these datasets and equations to project Canadian production available for export to the United States. The Module operates as follows:

1. The module is initialized in the canada class *__init__* method, with class dataframes and variables being declared here.
2. The *setup* method is called, which reads in the necessary input files via .csv or .pkl (see **intermediate_var_pickle.py**) format.
3. The *run* method is called, which kicks off the main model functions.
4. Year 1 operations are called from the *run* method, these operations are listed below. All other operations are run every model year unless otherwise indicated:
    a. *canada_base_prod_setup* - Splits AD and NA gas fixed production into separate tables, and assigns production to regions (East/West).
    b. *calibrate_ad_gas* - Calibrates AD natural gas fixed production based on brent price.
    c. *wells_setup* - Reset Canada well counts to 0 for all model years.
    d. *prod_profile_setup* - Solves for NA production profiles and creates net-new production dataframe.
5. Run *calculate_wells*, where wells are calculated by region and fuel type based on Henry Hub price.
6. Calculate regional na_prod based on a function of pre-calculated production profiles, and number of wells in *calculate_na_prod*.
7. Aggregate data into outputs files in *sum_and_merge_production*
8. **Run *write_intermediate_variables* to store iterative calculation tables for the next model run year. These tables are written out to .pkl files in intermediate_var_pickle.py**.
9. *report_results_unf* is called from **module_unf.py** to report results to debug files and the restart file.

Input files

- can_natgas_baseline_prod.csv - Canada legacy production
- can_natgas_dc_vars.csv - Decline Curve Variables
- can_benchmark_prices.csv - Canada Energy Regulator Prices
- can_natgas_no_export_prod.csv - Canadian Production for Domestic Use
- can_wells.csv - Canadian well counts
- can_natgas_poly_eqs.csv - Equations for calculating Canadian well drilling based on HH price
- can_elasticity_ad_gas.csv - Static assumptions regarding AD natural gas production elasticities

Model functions and class methods

- setup - Loads in relevant variables and processes.

- canada_base_prod_setup - Splits AD and NA gas fixed production into separate tables, and assigns production to regions (East/West).
- calibrate_ad_gas - Calibrates AD natural gas fixed production based on brent price.
- wells_setup - Reset Canada well counts to 0 for all model years.
- prod_profile_setup - Solves for NA production profiles and creates net-new production dataframe.
- calculate_wells - Solves for number of producing wells by region, gas type, and year.
- calculate_na_prod - Determines production from new drilling based on production profiles calculated in model year 1.
- sum_and_merge_production - Aggregate production data.
- write_intermediate_variables - Write local variables to restart file to be read back in each iteration.
- report_results_unf - Report results to restart variables

Debug files

- hsm_can_na_prod.csv - Expected NA Gas Production
- hsm_can_ad_prod.csv - Associated Dissolved Gas Production

Output restart variables

- ogsmout.cnenagprd - Expected NA gas production in Canada (1=east, 2=west)
- ogsmout.cnrnagprd - Realized NA gas production in Canada (1=east, 2=west)
- ogsmout.cnadagprd - AD gas production in Canada (1=east, 2=west)

class canada.**Canada**(*parent*)

Bases: *Submodule*

Canada submodule for HSM.

Parameters
parent (str) – Module_unf.Module (Pointer to parent module)

benchmark_prices
DataFrame of benchmark NG and crude prices

natgas_production
DataFrame of natural gas production

natgas_baseline_prod
DataFrame of total baseline production

natgas_dc_vars
DataFrame of decline curve variable input

natgas_poly_eqs
DataFrame of WCSB natural gas polynomial drilling equations

elasticity_ad_gas
DataFrame of elasticity for AD gas

natgas_no_export_prod
Dataframe containing NA gas produced by Canada not available for export

zero_year
Zero year for offshore submodule (equal to AEO year)

input_years
Index years of Canadian Energy Regulator Reporting

base_year
Base model year

bench_brent
Benchmark Brent Price

bench_ng_prod
Benchmark natural gas production by area & fuel category (bcf/yr)

b1
Drilling equation parameter

b0
Drilling equation parameter

bench_hen_hub
Benchmark Henry Hub Price

can_well_v_HH
Canadian wellhead to Henry Hub price differential

fixed_vols
DataFrame of Canadian gas fixed volumes

na_prod
DataFrame of total NA gas production

ad_prod
DataFrame of total AD production

na_prod_new
DataFrame of new Canadian NA Production

na_prod_sum
DataFrame of cumulative Canadian NA Production

can_wells
DataFrame of Canadian Model Outputs

cnenagprd
AIMMS output of expected NA gas production in Canada (1=east, 2=west)

cnadgprd
AIMMS output of AD gas production in Canada (1=east, 2=west)

cnrnagprd
AIMMS output of historical NA gas production in Canada (1=east, 2=west)

setup(*setup_filename*)
Setup for Canada Submodule.

Parameters
setup_filename (str) – Path to offshore setup file.

Return type
None

run()

Run Canada Submodule for HSM.

Return type
None

canada_base_prod_setup()

Splits AD and NA gas fixed production into separate tables, and assigns production to regions (East/West).

Returns

- **self.na_prod** (pandas.DataFrame) – DataFrame of non-associated natural gas production by region and type.
- **self.ad_prod** (pandas.DataFrame) – DataFrame of associated dissolved natural gas production by region.

calibrate_ad_gas()

Calibrates AD natural gas fixed production based on brent price.

Returns
self.ad_prod – DataFrame of associated dissolved natural gas production by region.

Return type
pandas.DataFrame

wells_setup()

Reset Canada well counts to 0 for all model years.

Returns
self.can_wells – Canada natural gas well counts by region and type.

Return type
pandas.DataFrame

prod_profile_setup()

Solves for NA production profiles and creates net-new production dataframe.

Returns
self.prod_profile – DataFrame of static production profiles for Canadian Natural Gas well production by region and type.

Return type
pandas.DataFrame

calculate_wells()

Solves for number of producing wells by region, gas type, and year.

- Calibrates Henry Hub Price
- Determines the number of wells drilled by region based on polynomial drilling equations calculated in pre-processors

Returns
self.can_wells – Canada natural gas well counts by region and type.

Return type
pandas.DataFrame

calculate_na_prod()

Determines production from net new drilling based on production profiles calculated in model year 1.

Returns

self.na_prod_new – DataFrame of net new non-associated natural gas production.

Return type

pandas.DataFrame

sum_and_merge_production()

Aggregate production data.

- Sum and merge production into NA and AD gas by region, type, and year
- Assign production to NEMS regions by year and set to restart file format

Returns

- **self.na_prod** (pandas.DataFrame) – DataFrame of non-associated natural gas production by region and type
- **self.na_prod_new** (pandas.DataFrame) – DataFrame of net new non-associated natural gas production
- **self.ad_prod** (pandas.DataFrame) – DataFrame of associated dissolved natural gas production by region
- self.na_prod_sum – DataFrame of total non-associated natural gas production
- self.ad_prod_sum – DataFrame of total associated dissolved natural gas production

write_intermediate_variables()

Write local variables to restart file to be read back in each iteration.

Returns

- **self.na_prod** (pandas.DataFrame) – DataFrame of non-associated natural gas production by region and type
- **self.na_prod_new** (pandas.DataFrame) – DataFrame of net new non-associated natural gas production
- self.na_prod_sum – DataFrame of total non-associated natural gas production
- **self.ad_prod** (pandas.DataFrame) – DataFrame of associated dissolved natural gas production by region
- self.ad_prod_sum – DataFrame of total associated dissolved natural gas production
- **self.can_wells** (pandas.DataFrame) – Canada natural gas well counts by region and type.
- **self.prod_profile** (pandas.DataFrame) – DataFrame of static production profiles for Canadian Natural Gas well production by region and type.

report_results_unf()

Report results to restart variables.

Returns

- **self.restart.ogsmout.cnenagprd** (pandas.DataFrame) – Expected NA gas production in Canada (1=east, 2=west)
- **self.restart.ogsmout.cnrnagprd** (pandas.DataFrame) – Realized NA gas production in Canada (1=east, 2=west)

- **self.restart.ogsmout.cnadagprd** (pandas.DataFrame) – AD gas production in Canada (1=east, 2=west)

ngp — CO_2 Capture at Natural Gas Processing Plants

ngp.py implements the NGP class. It evaluates the economic viability of CO_2 capture at natural gas processing facilities and generates CO_2 supply cost curves for CCATS.

This reference page lists module symbols and docstrings. For model design, assumptions, economics, data inputs, and output interpretation, see *CO_2 Capture at Natural Gas Processing Plants Submodule (NGP)*. Submodule for calculating natural gas processing facility carbon capture.

Summary

This submodule produces captured CO2 volumes and reports them to CCATS in response to the CCATS CO2 price path. The module operates as follows:

1. The module is initialized in the ngp class *__init__* method, with class dataframes and variables being declared here.
2. The *setup* method is called, which reads in the necessary input files via .csv or .pkl (see **intermediate_var_pickle.py**) format.
    a. *load_input_tables* - Load NGP submodule input tables.
    b. *adjust_inflation* - Adjust cost and price variables for inflation.
    c. *setup_input_tables* - Setup NGP submodule input tables.
    d. *setup_process_tables* - Setup NGP submodule process tables.
    e. *assign_operating_cc_plants* - Assign operational cc plants to active_cc_facilities_df.
    f. *load_intermediate_tables* - Load pickled tables.
3. The *run* method is called, which kicks off the main model functions.
    a. *calculate_flow_volumes* - Determine difference between natural gas processing facility database volumes and restart file natural gas flow.
    b. *assign_legacy_plant_volumes* - Assign excess CO2 volume through STEO years to representative "legacy" facilities, and add these to the NGP facility database.
    c. *assign_representative_plant_volumes* - Assign CO2 volumes after STEO years to representative "legacy" facilities, and add these to the NGP facility database.
    d. *retire_legacy_plants* - Retire legacy plants that are not CO2 retrofit eligible.
    e. *retrofit_eligibility* - Determine plant retrofit eligibility.
    f. *assign_electricity_costs* - Load electricity costs from the restart file and calculate electricity opex/tonne of CO2.
6. Load and run cash flow in *load_cashflow* and *run_cashflow*. Rank projects by project net present value.
7. Setup to send CO2 capture volumes to CCATS:
    a. *calculate_co2_capture_volumes* - Calculates CO2 capture volumes. Put into two carbon capture tables: one for 45Q eligible CO2 (retrofits 12 years old or less), and one for 45Q ineligible CO2.
    b. *calculate_co2_capture_costs* - Calculate average CO2 capital and opex (including electricity) capture costs, by Census division, and put in relevant table.
    c. *calculate_electricity_demand* - Calculate total electricity demand from carbon capture.
    d. *assign_active_status* - Assign profitable (NPV > 0) retrofits to active status.

9. Run *write_intermediate_variables* to store iterative calculation tables for the next model run year. These tables are written out to .pkl files in **intermediate_var_pickle.py**.
10. *report_results_unf* is called from **module_unf.py** to report results to debug files and the restart file.

Input files

- ngpp_input_eia_throughput???
- ngpp_input_netl_throughput???
- ngp_facility_input
- netl_purchased_power_cost
- netl_natural_gas_calculations
- mapping

Model functions and class methods

class NGP(sub.Submodule) - NGP submodule for HSM

Output debug files

Output restart variables

NGP submodule class methods

class ngp.**NGP**(*parent*)

Bases: *Submodule*

Natural Gas Processing Submodule for HSM.

Parameters
parent (str) – Module_unf.Module (Pointer to parent module)

setup(*setup_filename*)

Setup Natural Gas Processing Submodule for HSM.

Parameters
setup_filename (str) – Path to ngp setup file.

Return type
None

load_input_tables()

Load NGP submodule input tables.

Return type
None

adjust_inflation()

Adjust cost and price variables for inflation.

setup_input_tables()

Setup NGP submodule input tables

setup_process_tables()

Setup NGP submodule process tables

assign_operating_cc_plants()

Assign operational cc plants to active_cc_facilities_df.

load_intermediate_tables()

Load pickled tables.

Return type

None

run()

Run Natural Gas Processing Submodule for HSM.

Returns

- **self.ngpp_cash_flow** (pandas.DataFrame) – DataFrame of ngp project cashflows
- **self.ngpp_properties** (pandas.DataFrame) – DataFrame of key ngp project properties for cashflow (i.e. project cost bases)

calculate_flow_volumes()

Determine difference between natural gas processing facility database volumes and restart file natural gas flow (ogrnagprd).

tech_improvement()

Apply technology improvement rate to INV and O&M costs.

assign_legacy_plant_volumes()

Assign excess CO2 volume through STEO years to representative "legacy" facilities, and add these to the NGP facility database.

assign_representative_plant_volumes()

Assign CO2 volumes after STEO years to representative "legacy" facilities, and add these to the NGP facility database.

retire_legacy_plants()

Retire legacy plants that are not CO2 retrofit eligible

retrofit_eligibility()

Determine plant retrofit eligibility

Return type

None

assign_co2_prices()

Assign CO2 Price from CCATS.

assign_electricity_costs()

Load electricity costs from the restart file and calculate electricity opex/tonne of CO2.

load_cashflow()

load Cashflow

run_cashflow()

Run Cash Flow.

assign_active_status()

Assign profitable retrofits to active status

calculate_co2_capture_volumes()

Calculate capture volumes

calculate_co2_capture_costs()

Calculate average cost of capture

calculate_electricity_demand()

Calculate total electricity demand from carbon capture.

write_intermediate_variables()

Pickle local variables.

Return type
None

report_results()

Report results to relevant restart variables

- Report CO2 supply by 45Q eligibility and opex/capex costs respectively

CCATSDATA CST_NGP_INV - Investment cost for CO2 capture from natural gas processing facilities CCATSDATA CST_NGP_OM - O&M cost for CO2 capture from natural gas processing facilities CCATSDATA SUP_NGP_45Q - CO2 volumes from natural gas processing facilities that are 45Q eligible CCATSDATA SUP_NGP_NTC - CO2 volumes from natural gas processing facilities, no tax credit

submodule — Shared Base Class

submodule.py defines the Submodule base class shared by Onshore, Offshore, Alaska, Canada, and NGP implementations. It provides common setup and data-loading behaviors that child classes extend.

This reference page lists module symbols and docstrings. For submodule overview and inheritance context, see *Submodules* and *Architecture*. Generic submodule class of HSM

Example

import submodule as sub

class submodule.**Submodule**(*parent*, *submodule_name*)

Bases: object

Generic submodule for HSM.

Parameters

- **parent** (*module_unf.Module*) – Pointer to head module
- **submodule_name** (str) – Name of submodule (e.g. 'Offshore')

setup(*setup_filename*)

Setup General Submodule for HSM

Parameters
setup_filename (str) – Path to submodule setup file.

Examples

def setup(self):
super().setup(setup_filename)

Return type
None

run()

Run General Submodule for HSM

Examples

def run(self):
com.print_out('running offshore submodule') super().run()

Return type
None

2.11.4 Shared Economics

`cash_flow` — Discounted Cash Flow Engine

`cash_flow.py` implements the `CashFlow` class, the shared DCF engine used by the Onshore, Offshore, and Alaska submodules. It performs the 18-step sequential calculation from revenue to profitability index.

This reference page lists module symbols and docstrings. For full methodology details, see *Economic Framework — Discounted Cash Flow Model*. **Mathematical notation** (NPV, PI, C_K, step equations) is defined there, not on this API page. Submodule for calculating the discounted cash flow.

Summary

The cashflow class is a universal cashflow calculation used by the **Onshore**, **Offshore** and **Alaska** submodule to calculate project economics. The class operates as follows:

1. The module is initialized in the onshore class *__init__* method, with class dataframes and variables being declared here.
2. The *setup* function is called, which loads in base input file assumptions (i.e. tax rates), all other setup is done in the submodules (i.e. **Onshore.py**).
3. Project revenue from crude oil and natural gas, is calculated in *calculate revenue*.
4. Calculate project royalties and severance taxes for crude oil and natural gas production
5. Calculate project drill cost (combination of dryhole and drilling costs)
6. For **Onshore** submodule developing projects run *calculate_ngpl_operating_cost* and *calculate_operating_cost* to aggregate multiple cost streams. *calculate_ngpl_operating_cost* also includes NGPL revenues, which are allocated as a negative cost. In **Alaska** and **Offshore** submodules ngpl operating costs aren't calculated and aggregated operating costs are loaded in directly from the submodule
7. Calculate project transportation costs.
8. Split costs into intangible and tangible costs based on HSM input assumptions in *calculate_intangible_tangible*.
9. Run *calculate_depreciation* to calculate amortization and depreciation based on intangible/tangible breakdown. For **Onshore** submodule only run *calculate_gg_la_depletion* to apply depletion for geological and lease costs for undiscovered conventional projects.
10. Run *calculate_econ_limit* to determine the project operation year at which primary project operations are no longer profitable.
11. Run *calculate_abandonment* to apply abandonment costs and zero out production for projects beyond their economic limit.
12. Apply taxes and credits in *apply_co2_eor_tax_credit*, *calculate_state_tax* and *calculate_fed_tax*.
13. Calculate project NPV in *calculate_cash_flow* and calculate project profitability in *calculate_profitability*

Input files

None

Model functions and class methods

- __init__ - Initializes CashFlow object
- setup - Initializes CashFlow input variables
- calculate_revenue - Calculates revenue from crude and natural gas production
- calculate_royalty - Calculates crude and natural gas royalty
- calculate_severance - Calculates crude and natural gas severance tax
- calculate_drill_cost - Calculates drilling cost
- calculate_ngpl_operating_cost - Calculates NGPL operating cost. Usually a negative value to be applied against opex
- calculate_operating_cost - Calculates operating expenses for **Onshore** submodule (**Offshore** and **Alaska** operating costs are loaded in directly)
- calculate_trans_cost - Calculates crude and natural gas transportation costs
- calculate_intangible_tangible - Calculates all tangible and intangible cost
- calculate_depreciation - Calculates all tangible and intangible cost
- calculate_gg_la_depletion - Calculates depletion from geological, geophysical and least acquisition expenses
- calculate_econ_limit - Calculates project economic limit (Net Operating Income) to determine well lifetime
- calculate_state_tax - Calculates the state tax base and state tax
- calculate_fed_tax - Calculates the federal tax base and federal tax
- calculate_cash_flow - Calculates the discounted cash flow (NPV)
- calculate_profitability - Calculates project profitability
- to_csv - Prints cash flow data and properties to csv

Output debug files

- cashflow_props.csv - Output of cashflow "properties" table
- cashflow_costs.csv - Output of cashflow costs

Output restart variables

None

Example

Time-dependent DataFrames should have the following format:

```
index   0       1       2       ...
0       val00   val01   val02   ...
1       val10   val11   val12   ...
2       val20   val21   val22   ...
...     ...     ...     ...     ...
```

Row or index values represent well, project, or field id numbers; columns are zero-indexed projection years. Columns can be set up using `list(range(year_range))` where `year_range` is the last year of the evaluation period.

One DataFrame, the `self.properties` DataFrame, should have the following form:

```
index   nam.crude_price nam.natgas_price nam.crude_tariff_price ...
0       val00           val01            val02                  ...
1       val10           val11            val12                  ...
2       val20           val21            val22                  ...
...     ...             ...              ...                    ...
```

Where the row or index values represent well, project, or field id numbers, and columns are non-time-dependent variables for each project.

The recommended usage:

```
import cash_flow as cf
cash_flow = cf.CashFlow()
cash_flow.calculation_revenue()
...
```

Notes

- Calculations are currently unit-agnostic. Rates, e.g. state tax, are assumed to be fractional rather than percentile.
- Todo: Add checking/warning for required properties (setup so that when a property not included in DCF model will fail model tells us LOW/MED)
- Todo: Need to tie federal tax rate in wacc/discount rate calculation to cashflow federal tax rate (wasn't comfortable with how we were using fed tax rate, better way to do this? LOW/MED)

Cashflow class methods

class cash_flow.**CashFlow**

Bases: object

DataFrame-based, discounted cash flow calculator.

state_tax_filename = ''

filename of state_tax_lookup

state_tax_lookup

state tax rate lookup table

alias of DataFrame

depreciation_schedule_filename = ''

filename of depreciation_schedule

depreciation_schedule

table of depreciation schedules

alias of DataFrame

class_init_bool = False

check if CashFlow class is init'ed

__init__()

Initializes CashFlow object.

Return type

CashFlow

classmethod setup()

Initializes CashFlow input variables.

Will only load input data once. Must load in all input filenames into the relevant variables (e.g. state_tax_filename)

Returns

- **cls.state_tax_lookup** (pandas.DataFrame) – DataFrame of state tax rates
- **cls.depreciation_schedule** (pandas.DataFrame) – DataFrame of depreciation schedule assumptions
- **cls.class_init_bool** (bool) – Flag for whether cashflow class has been initialized

calculate_revenue()

Calculates revenue from crude and natural gas production.

Revenue is real revenue only: crude_revenue + natgas_revenue. Vented-gas royalty is computed in calculate_royalty. Crude and natgas revenue use (price - tariff) per unit; submodules set tariff (crude_tariff_price, natgas_tariff_price) and typically use 0, so revenue is effectively price x volume when tariff is zero.

Returns

- **self.crude_revenue** (pandas.DataFrame) – DataFrame of project revenue from crude oil production
- **self.natgas_revenue** (pandas.DataFrame) – DataFrame of project revenue from natural gas production
- **self.revenue** (pandas.DataFrame) – DataFrame of project revenue from crude oil and natural gas production (real revenue only)

calculate_royalty()

Calculates crude and natural gas royalty.

> **Note**
>
> Currently crude and natural gas have the same, time-dependent royalty rate. Royalties are applied to all projects because we don't have the means to differentiate between federal and non-federal land. The imputed value of vented/flared gas (vent_natgas_revenue) is computed in this method when on_vented_gas_royalty is True and used as the base for vent_natgas_royalty; royalty thus includes royalty on that imputed value when the flag is True.

Returns

- **self.crude_royalty** (pandas.DataFrame) – Royalty paid on project crude oil production
- **self.natgas_royalty** (pandas.DataFrame) – Royalty paid on project natural gas production
- **self.royalty** (pandas.DataFrame) – Royalty paid on project crude oil and natural gas production

calculate_severance()

Calculates crude and natural gas severance tax.

Returns

- **self.properties** (pandas.DataFrame) – DataFrame of key project properties for cashflow (i.e. project cost bases)
- **self.severance_tax** (pandas.DataFrame) – DataFrame of project severance taxes

calculate_drill_cost()

Calculates drilling cost.

Returns

- **self.drill_cost** (pandas.DataFrame) – DataFrame of overall project drilling costs
- **self.dry_hole_cost** (pandas.DataFrame) – DataFrame of just project dryhole costs

calculate_ngpl_operating_cost()

Calculates NGPL operating cost. Usually a negative value to be applied against opex.

Returns
self.ngpl_operating_cost – DataFrame of NGPL operating costs (or revenues when negative)

Return type
pandas.DataFrame

calculate_operating_cost()

Calculates operating expenses for **Onshore** submodule (**Offshore** and **Alaska** operating costs are loaded in directly).

Returns

- **self.crude_operating_cost** (pandas.DataFrame) – DataFrame of crude oil production operating cost
- **self.natgas_operating_cost** (pandas.DataFrame) – DataFrame of natural gas production operating cost
- **self.co2_operating_cost** (pandas.DataFrame) – DataFrame of CO2 EOR operating costs
- **self.general_admin_cost** (pandas.DataFrame) – DataFrame of general well admin costs
- **self.operating_cost** (pandas.DataFrame) – DataFrame of total operating costs

calculate_trans_cost()

Calculates crude and natural gas transportation costs.

Returns

- **self.crude_trans** (pandas.DataFrame) – DataFrame of project crude oil transportation costs
- **self.natgas_trans** (pandas.DataFrame) – DataFrame of project natural gas transportation costs
- **self.trans_cost** (pandas.DataFrame) – DataFrame of total project transportation costs

calculate_intangible_tangible()

Calculates all tangible and intangible cost.

Returns

- **self.tang_cost** (pandas.DataFrame) – DataFrame of project tangible costs
- **self.intang_cost** (pandas.DataFrame) – DataFrame of project intangible costs

calculate_depreciation()

Calculates all tangible and intangible cost.

Returns

- **self.expn_intang_cost** (pandas.DataFrame) – DataFrame of expensed intangible costs
- **self.amor_intang_cost** (pandas.DataFrame) – DataFrame of amortized intangible costs
- **self.deprec_tang_cost** (pandas.DataFrame) – DataFrame of depreciated tangible costs

calculate_gg_la_depletion()

Calculates depletion from geological, geophysical and least acquisition expenses.

Returns

- **self.depletion** (pandas.DataFrame) – DataFrame of depletion from project geological, geophysical and lease costs
- **self.gg_la_cost_dep** (pandas.DataFrame) – DataFrame of depreciated project geological, geophysical and lease costs

calculate_econ_limit()

Calculates project economic limit (Net Operating Income) to determine well lifetime.

Returns

self.econ_limit – DataFrame of project economic limits

Return type

pandas.DataFrame

calculate_abandonment()

Calculates abandonment and zeros out properties beyond the economic limit.

> **Note**
>
> Assumes minimum years before abandonment configurable via setup file

Returns

- **self.abandon_cost** (pandas.DataFrame) – DataFrame of project abandonment costs
- **self.properties** (pandas.DataFrame) – DataFrame of key project properties for cashflow (i.e. project cost bases)
- **self.revenue** (pandas.DataFrame) – DataFrame of project revenue from crude oil and natural gas production
- **self.royalty** (pandas.DataFrame) – Royalty paid on project crude oil and natural gas production
- **self.severance_tax** (pandas.DataFrame) – DataFrame of project severance taxes
- **self.operating_cost** (pandas.DataFrame) – DataFrame of total operating costs
- **self.expn_intang_cost** (pandas.DataFrame) – DataFrame of expensed intangible costs
- **self.amor_intang_cost** (pandas.DataFrame) – DataFrame of amortized intangible costs
- **self.deprec_tang_cost** (pandas.DataFrame) – DataFrame of depreciated tangible costs
- **self.dry_hole_cost** (pandas.DataFrame) – DataFrame of just project dryhole costs

- **self.drill_cost** (pandas.DataFrame) – DataFrame of overall project drilling costs
- **self.crude_production** (pandas.DataFrame) – DataFrame of project crude oil production
- **self.natgas_production** (pandas.DataFrame) – DataFrame of project natural gas production
- **self.crude_revenue** (pandas.DataFrame) – DataFrame of project revenue from crude oil production
- **self.natgas_revenue** (pandas.DataFrame) – DataFrame of project revenue from natural gas production
- **self.crude_trans** (pandas.DataFrame) – DataFrame of crude oil transportation costs
- **self.natgas_trans** (pandas.DataFrame) – DataFrame of natural gas transportation costs
- **self.trans_cost** (pandas.DataFrame) – DataFrame of crude oil and natural gas transportation costs
- **self.crude_royalty** (pandas.DataFrame) – DataFrame of project crude oil royalties
- **self.natgas_royalty** (pandas.DataFrame) – DataFrame of project natural gas royalties

calculate_co2_eor_tax_credit()

Calculates CO2 EOR tax Credit.

Returns

- **self.properties** (pandas.DataFrame) – DataFrame of key project properties for cashflow (i.e. project cost bases)
- **self.eor_tax_credit** (pandas.DataFrame) – DataFrame of project CO2 EOR tax credits

calculate_ch4_emission_penalties()

Calculates CH4 emission penalties.

- Converts Emissions from mcf to Metric tons
- Calculates Emission Penalties

Returns

- self.ch4_emissions – DataFrame of CH4 emissions/project in metric tons
- **self.ch4_emission_cost** (pandas.DataFrame) – DataFrame of CH4 emission costs/project

calculate_state_tax()

Calculates the state tax base and state tax.

Returns

- **self.properties** (pandas.DataFrame) – DataFrame of key project properties for cashflow (i.e. project cost bases)
- **self.tax_base_state** (pandas.DataFrame) – DataFrame of taxable project revenues for state taxes
- self.state_tax – DataFrame of project state taxes

calculate_fed_tax()

Calculates the federal tax base and federal tax.

Returns

- **self.tax_base_fed** (pandas.DataFrame) – DataFrame of taxable project revenues for federal taxes
- **self.fed_tax** (pandas.DataFrame) – DataFrame of project federal taxes

calculate_cash_flow()

Calculates the discounted cash flow (NPV).

Returns

- **self.cash_flow** (pandas.DataFrame) – DataFrame of project cashflows
- **self.properties** (pandas.DataFrame) – DataFrame of key project properties for cashflow (i.e. project cost bases)

calculate_profitability()

Calculates project profitability.

Returns
self.properties – DataFrame of key project properties for cashflow (i.e. project cost bases)

Return type
pandas.DataFrame

to_csv(*output_path*, *filename*, *year*, *iteration*, *cycle='nan'*, *mode='a'*, *header=True*)

Prints cash flow data and properties to csv.

Return type
None

ngp_cash_flow — NGP Cash Flow Engine

ngp_cash_flow.py implements the NGP_CashFlow class, the DCF engine specialized for natural gas processing plant economics. It differs from the standard CashFlow class in that revenue comes from CO_2 capture and sale rather than hydrocarbon production.

This reference page lists module symbols and docstrings. Submodule for calculating the natural gas processing facility discounted cash flow.

Summary

The ngp cashflow class is a cashflow calculation specific to natural gas processing facility economics. The class operates as follows:

1. The module is initialized in the ngp class *__init__* method, with class dataframes and variables being declared here.
2. The *setup* function is called, which loads in base input file assumptions (i.e. tax rates), all other setup is done in ngp.py.
3. Revenue from captured co2 is calculated in *calculate revenue*.
4. Calculate energy costs.
5. Calculate total project capital costs and operating costs.
6. Split costs into intangible and tangible costs based on HSM input assumptions in *calculate_intangible_tangible*.
7. Run *calculate_depreciation* to calculate amortization and depreciation based on intangible/tangible breakdown.

8. Run *calculate_econ_limit* to determine the project operation year at which primary project operations are no longer profitable.
9. Apply taxes.
10. Calculate project NPV in *calculate_cash_flow* and calculate project profitability in *calculate_profitability*

Input files

None

Model functions and class methods

Output debug files

Output restart variables

None

Example

Time-dependent DataFrames should have the following format:

```
index   0       1       2       ...
0       val00   val01   val02   ...
1       val10   val11   val12   ...
2       val20   val21   val22   ...
...     ...     ...     ...     ...
```

Row or index values represent well, project, or field id numbers; columns are zero-indexed projection years. Columns can be set up using `list(range(year_range))` where `year_range` is the last year of the evaluation period.

One DataFrame, the `self.properties` DataFrame, should have the following form:

```
index   nam.crude_price nam.natgas_price nam.crude_tariff_price ...
0       val00           val01            val02                   ...
1       val10           val11            val12                   ...
2       val20           val21            val22                   ...
...     ...             ...              ...                     ...
```

Where the row or index values represent well, project, or field id numbers, and columns are non-time-dependent variables for each project.

The recommended usage:

```
import cash_flow as cf
cash_flow = cf.CashFlow()
cash_flow.calculation_revenue()
...
```

Notes

- Calculations are currently unit-agnostic. Rates, e.g. state tax, are assumed to be fractional rather than percentile.

NGP cashflow class methods

class ngp_cash_flow.**NGP_CashFlow**

Bases: `object`

DataFrame-based, discounted cash flow calculator.

depreciation_schedule_filename = ''

filename of depreciation_schedule

depreciation_schedule

table of depreciation schedules

alias of DataFrame

class_init_bool = False

check if CashFlow class is init'ed

__init__()

Initializes NGP CashFlow object.

Return type
CashFlow

classmethod setup()

Initializes CashFlow input variables.

Will only load input data once. Must load in all input filenames into the relevent variables (e.g. deprec_schedule filename)

Returns

- **cls.depreciation_schedule** (pandas.DataFrame) – DataFrame of depreciation schedule assumptions
- **cls.class_init_bool** (bool) – Flag for whether cashflow class has been initialized

calculate_revenue()

Calculates revenue from crude and natural gas production.

Returns
self.co2_cost – DataFrame of facility cost from CO2 flows

Return type
pandas.DataFrame

calculate_electricity_cost()

Calculates electricity cost associated with CO2 capture at NGP facilities.

calculate_total_operating_costs()

Calculates total costs

calculate_intangible_tangible()

Split capital cost into tangible and intangible drilling cost DataFrames.

Tangible drilling costs (TDC) are expenses tied to physical drilling equipment; intangible drilling costs (IDC) cover the remainder of drilling-related expenses.

Returns
Sets self.tang_cost and self.intang_cost from capital_cost and exp_tang_frac in self.properties.

Return type
None

calculate_depreciation()

Calculates all tangible and intangible cost.

Returns

- **self.expn_intang_cost** (pandas.DataFrame) – DataFrame of expensed intangible costs
- **self.amor_intang_cost** (pandas.DataFrame) – DataFrame of amortized intangible costs
- **self.deprec_tang_cost** (pandas.DataFrame) – DataFrame of depreciated tangible costs

calculate_econ_limit()

Calculates project economic limit (Net Operating Income) to determine facility lifetime.

Returns
self.econ_limit – DataFrame of project economic limits

Return type
pandas.DataFrame

calculate_fed_tax()

Calculates the federal tax base and federal tax.

Returns

- **self.tax_base_fed** (pandas.DataFrame) – DataFrame of taxable project revenues for federal taxes
- **self.fed_tax** (pandas.DataFrame) – DataFrame of project federal taxes

calculate_cash_flow()

Calculates the discounted cash flow (NPV).

Returns

- **self.cash_flow** (pandas.DataFrame) – DataFrame of project cashflows
- **self.properties** (pandas.DataFrame) – DataFrame of key project properties for cashflow (i.e. project cost bases)

calculate_profitability()

Calculates project profitability.

Returns
self.properties – DataFrame of key project properties for cashflow (i.e. project cost bases)

Return type
pandas.DataFrame

calculate_cost_tonne()

Calculates facility costs in $_tonne

Returns
self.properties – DataFrame of key project properties for cashflow (i.e. project cost bases)

Return type
pandas.DataFrame

to_csv(*output_path*, *filename*, *year*, *iteration*, *mode='a'*, *header=True*)

Prints cash flow data and properties to csv.

Return type
None

`drilling_equations` — Drilling and Production Functions

`drilling_equations.py` contains the drilling schedule and production profile functions used by Onshore, Offshore, and Alaska. It is a module of functions (no classes).

This reference page lists module symbols and docstrings. For methodology details, see *Drilling and Production Equations*. **Mathematical notation** (w_y, α_p, $q(t)$) is defined there, not on this API page. File containing drilling equations used in HSM.

Summary

This file contains the drilling functions used by the **Onshore**, **Offshore** and **Alaska** submodules. Each drilling equation is accessed from the relevant submodule file, with some submodules containing multiple drilling equations. The drilling equations below are:

1. on_next_wells - Generates Onshore submodule wells drilled by project for the current model year
2. off_drill_schedule - Generates Offshore Submodule developmental drilling schedule by project
3. off_operating_schedule - Generates Offshore Submodule operating cost schedule
4. off_production_profile - Generates Offshore submodule project production profiles
5. ak_calculate_drill_schedule - Generates Alaska submodule project drilling schedules
6. ak_production_profile - Generates Alaska Submodule project production profiles
7. ak_decline_profile - Decline schedule for Alaska Submodule producing projects

Model functions and class methods

- on_next_wells - Generates Onshore submodule wells drilled by project for the current
- off_drill_schedule - Generates Offshore Submodule developmental drilling schedule by
- off_operating_schedule - Generates Offshore Submodule operating cost schedule
- off_production_profile - Generates Offshore submodule project production profiles
- ak_calculate_drill_schedule - Generates Alaska submodule project drilling schedules
- ak_production_profile - Generates Alaska Submodule project production profiles
- ak_decline_profile - Decline schedule for Alaska Submodule producing projects

Input files

None

Output debug files

None

Output restart variables

None

Notes

Convention for import alias is, import drilling_equations as drill_eq

Drilling equations class methods

`drilling_equations.`**`on_next_wells`**(*well_decline_limit*, *max_wells*, *past_drilling*, *last_year_drilling*, *max_drill_rate*, *max_drill_rate_frac*, *drill_predecline*, *ramp_up_years*, *oil_price*, *ng_price*, *well_type_num*, *base_oil_prc*, *base_gas_prc*, *oil_prod*, *gas_prod*, *max_year_drill_percent*, *current_model_year*, *low_price_flag*, *undiscovered_drill_flag=False*)

Generates Onshore submodule wells drilled by project for current model year.

The drilling equation operates as follows:

1. Function variables are instantiated and ramp-up year status is declared (whether drilling is sitll ramping up)
2. **Base drilling is determined in a loop, where drilling is continuously recalculated as long as total drilling < past drilling:**
    a. If ramp-up year, base drilling = max drilling * (production year/total ramp-up year)
    b. If normal drilling year, base drilling = max drilling
    c. If past drilling > well decline limit (project area well saturation limit), base drilling = max drilling * (1. - drilling fraction) ** (year - decline start)
3. Set a floor on base drilling of 5 wells/year and add 1 to the current modeled year
4. **Once base drilling is determined when total drilling > past drilling:**
    a. Set max drilling at double last year's drilling (or min of 10)
    b. Set max year drilling as maximum of current calculated drilling and calculated maximum drilling percentage based on total remaining available patterns
    c. Adjust max drilling for ramp-up years
    d. Set a floor on drilling based on these constraints
5. Adjust drilling based on crude oil and natural gas prices (i.e. higher oil price means more drilling)
6. Apply the maximum drilling constraint
7. Round drilling to be an integer

Parameters

- **well_decline_limit** (int) – Project pattern / standard project pattern size, used to determine point at which well starts experiencing diminishing returns
- **max_wells** (int) – Project pattern / min well spacing, used to determine absolute maximum number of wells possible in a given project
- **past_drilling** (int) – Past drilling during model years
- **last_year_drilling** (int) – Drilling in last model year
- **max_drill_rate** (float) – Maximum drill rate allowed by the model
- **max_drill_rate_frac** (float) – Fraction of maximum drill rate applied in given year (i.e. well will not operate at maximum drill rate if in decline)
- **drill_predecline** (float) – At 70% of project well_decline_limit, project EURs start to decline
- **ramp_up_years** (float) – Number of years a project needs to be drilling before reaching max_drill_rate
- **oil_price** (int) – Model year's oil price
- **ng_price** (int) – Model year's natural gas price
- **well_type_num** (int) – Well type number used for price benchmarking
- **base_oil_prc** (float) – Base oil price set manually in the input file for price at which drilling accelerates

- **base_gas_prc** (float) – Base natural gas price set manually in the input file for price at which drilling accelerates
- **oil_prod** (float) – Project oil production
- **gas_prod** (float) – Project oil production
- **max_year_drill_percent** (float) – Percentage of remaining produciton that can be drilled in model year
- **low_price_flag** (bool) – Flag to throttle drilling responsiveness in low price case
- **undiscovered_drill_flag** (bool, *optional*) – Flag indicating if this is an undiscovered project

Returns
cur_year_drill – Current model year drilling

Return type
int

drilling_equations.**calculate_price_adjustment**(*oil_price*, *ng_price*, *well_type_num*, *base_oil_prc*, *base_gas_prc*, *oil_prod*, *gas_prod*, *low_price_flag*)

Calculate price adjustment factor for drilling based on commodity prices.

This function adjusts drilling activity based on current oil/gas prices relative to base prices. Higher prices increase drilling (price_adj > 1), lower prices decrease drilling (price_adj < 1).

Parameters

- **oil_price** (float) – Current model year's oil price
- **ng_price** (float) – Current model year's natural gas price
- **well_type_num** (int) – Well type number (<=2 for oil wells, >=3 for gas wells)
- **base_oil_prc** (float) – Base oil price threshold for drilling acceleration
- **base_gas_prc** (float) – Base natural gas price threshold for drilling acceleration
- **oil_prod** (float) – Project oil production level
- **gas_prod** (float) – Project gas production level
- **low_price_flag** (int) – Flag to throttle drilling responsiveness in low price scenarios (0 or 1)

Returns
Price adjustment factor (multiplier for drilling count)

Return type
float

drilling_equations.**off_drill_schedule**(*series*, *evaluation_years*)

Generates Offshore Submodule developmental drilling schedule by project.

- Map drilling/year against total wells that need to be drilled

Parameters

- **series** (pandas.Series) – Series containing drilling variables, specifically, nam.delay_years, nam.drill_per_year, and nam.development_wells
- **evaluation_years** (int) – Number of years for economic or production evaluation

Returns
ser_out – Project drilling schedule

Return type
pandas.Series

drilling_equations.**off_operating_schedule**(*series*, *evaluation_years*)

Generates Offshore Submodule operating cost schedule.

- Map drilling/year against total wells that need to be drilled

Parameters

- **series** (pandas.Series) – Series containing drilling variables, specifically, nam.delay_years, nam.drill_per_year, nam.development_wells, and nam.delineation_wells
- **evaluation_years** (int) – Number of years for economic or production evaluation

Returns
ser_out – Project operating cost schedule

Return type
pandas.Series

drilling_equations.**off_production_profile**(*series*, *evaluation_years*, *hist_year*, *oil_or_gas='oil'*, *past_prod_flag=False*, *decline_flag=False*)

Generates Offshore submodule project production profiles.

The Offshore production profile equation operates as follows:

1. Instantiate variables
2. **Apply decline flags for producing projects**
    a. 0 = new field, apply undeveloped prod forecast even if past production
    b. 1 = field declining, apply field decline profile
    c. 2 = field production increasing, hold production constant for the following year, then start declining
    d. 3 = standard field, apply normal production profile
3. **Base drilling is determined in a loop, where drilling is continuously recalculated as long as total drilling < past drilling:**
    a. If ramp-up year, base drilling = max drilling * (production year/total ramp-up year)
    b. If normal drilling year, base drilling = max drilling
    c. If past drilling > well decline limit (project area well saturation limit), base drilling = max drilling * (1. - drilling fraction) ** (year - decline start)

Parameters

- **series** (pandas.Series) – Series containing production variables, specifically, nam.oil_resources, nam.predecline_prod_fraction_oil, nam.initial_oil_prod_rate, nam.max_oil_production_rate, nam.oil_decline_rate, nam.gas_resources, nam.predecline_prod_fraction_gas, nam.initial_gas_prod_rate, nam.max_gas_production_rate, nam.gas_decline_rate, nam.past_production, nam.ramp_up_years, and nam.hyper_decline_coef
- **evaluation_years** (int) – Number of years for economic or production evaluation
- **history_year** (int) – Last year of produciton history

- **oil_or_gas** (str) – nam.oil or nam.gas
- **past_prod_flag** (bool) – If the field has past production (i.e. producing fields) use past production, otherwise use 0
- **decline_flag** (bool) – Apply decline flags for historical production

Returns
ser_out – Project production profile

Return type
pandas.Series

drilling_equations.**ak_calculate_drill_schedule**(*series*, *evaluation_years*)

Generates Alaska submodule project drilling schedules.

Parameters

- **series** (pandas.Series) – Series containing production variables, specifically, nam.oil_resources, nam.predecline_prod_fraction_oil, nam.initial_oil_prod_rate, nam.max_oil_production_rate, nam.oil_decline_rate, nam.gas_resources, nam.predecline_prod_fraction_gas, nam.initial_gas_prod_rate, nam.max_gas_production_rate, nam.gas_decline_rate, nam.past_production, nam.ramp_up_years, and nam.hyper_decline_coef
- **evaluation_years** (int) – Number of years for economic or production evaluation

Returns
ser_out – Alaska project drill schedule

Return type
pandas.Series

drilling_equations.**ak_production_profile**(*series*, *evaluation_years*, *oil_or_gas='oil'*, *past_prod_flag=False*)

Generates Alaska Submodule project production profiles.

The Alaska production profile equation operates as follows:

1. Instantiate variables
2. **Base drilling is determined in a loop, where drilling is continuously recalculated as long as total drilling < past drilling:**
   a. If ramp-up year, base drilling = max drilling * (production year/total ramp-up year)
   b. If normal drilling year, base drilling = max drilling
   c. If past drilling > well decline limit (project area well saturation limit), base drilling = max drilling * (1. - drilling fraction) ** (year - decline start))

Parameters

- **series** (pandas.Series) – Series containing prodcution variables, specifically, nam.oil_resources, nam.predecline_prod_fraction_oil, nam.initial_oil_prod_rate, nam.max_oil_production_rate, nam.oil_decline_rate, nam.gas_resources, nam.predecline_prod_fraction_gas, nam.initial_gas_prod_rate, nam.max_gas_production_rate, nam.gas_decline_rate, nam.past_production, nam.ramp_up_years, and nam.hyper_decline_coef
- **evaluation_years** (int) – Number of years for economic or production evaluation
- **oil_or_gas** (str) – nam.oil or nam.gas
- **past_prod_flag** (bool) – If the field has past production (i.e. producing fields) use past production, otherwise use 0

Returns
ser_out – Alaska project production profile

Return type
pandas.Series

drilling_equations.**ak_decline_profile**(*series*, *evaluation_years*)

Decline schedule for Alaska Submodule producing projects.

Parameters

- **series** (pandas.Series) – Series containing production variables, specifically, nam.oil_decline_rate, nam.max_oil_production_rate, nam.oil_resources, nam.avg_3_prod
- **evaluation_years** (int) – Number of years for economic or production evaluation

Returns
ser_out – Alaska project decline profile

Return type
pandas.Series

2.11.5 Input/Output

restart_unf — NEMS Restart File I/O

restart_unf.py defines the RestartUnf class used to read and write NEMS unformatted binary restart arrays (.unf) for HSM inputs, control flags, and outputs.

This reference page lists module symbols and docstrings. For variable-level details, see *NEMS restart input variables reference* and *Output Variables Reference*. Class for handling the Restart File in HSM.

Summary

The **Restart** module reads in the variables from the restart file, stores these variables in a dictionary, and writes these HSM output variables back to the restart file after HSM processes have been run. These processes are performed using **self.parent.pyfiler1.pyd**, which is maintained by the Integration team. The module operates as follows:

1. **restart_unf** is called from the module_unf parent class to read in the restart file.
2. The *setup* function is called, which in turn calls run function. The run function calls the read_filer method in self.parent.pyfiler1 and loads all the variables into a class dictionary from restart_HSMIN.unf. From this class dictionary HSM is able to identify which variables need to be written back out to the restart file at the end of each HSM run.
3. Relevant restart variables are instantiated, read into appropriately indexed dataframes, and offset to the current calendar years in the relevant indices. Once this is done, HSM operations move back to **module_unf**.
5. Restart variables are updated in the various HSM modules.
6. In the *prepare_results* method of **module_unf** the *write_results* **restart_unf** function is called.
7. Each instantiated variable is called from the **restart_unf** class dictionary, re-sized to match restart file formatting, and then written to the appropriate restart variable.

Input files

- restart.npz - Restart file from NEMS in npz format

Model functions and class methods

class (sub.Submodule) - Restart File submodule for HSM

- __init__ - Constructor to initialize Restart submodule
- setup - Setup function for reading the restart file into HSM
- run - Calls the read_filer method from self.parent.pyfiler1 and loads the restart file into HSM
- add_int - Function for adding an integer from the restart file into HSM as an integer
- add_df - Function for adding a dataframe from the restart file into NEMS
- write_results - Repacks local restart file variables into multi-dimensional arrays, and writes to restart file
- dump_restart - Dumps restart file to .xlsx debug file

Output restart variables

Crude Oil Restart Variables

- pmmout_rfqtdcrd - Total crude production by HSM region
- pmmout_rfqdcrd - Total crude oil production by HSM region (not including EOR)
- ogsmout_ogqcrrep - Crude oil production by oil category
- ogsmout_ogcoprd - Crude oil production by lower 48 region
- ogsmout_ogqshloil - Crude oil production by select tight oil play
- ogsmout_ogoilprd - Crude oil production by oil type and HSM district
- ogsmout_ogcrdprd - Crude oil production by HSM region and crude type
- ogsmout_ogcruderef - Crude oil production by LFMM crude oil type and region
- ogsmout_ogcrdheat - Heat rate by type of crude oil
- ogsmout_ogeorprd - CO2 EOR crude oil production
- ogsmout_ogcoprdgom - Crude oil production Gulf of America
- ogsmout_ogprcoak - Crude oil production by Alaska region

Natural Gas Restart Variables

- ogsmout_ogenagprd - Natural gas expected production by natural gas type and HSM district
- ogsmout_ogrnagprd - Natural gas realized production by natural gas type and HSM district
- ogsmout_ogadgprd - Natural gas associated dissolved production by oil type and HSM district
- ogsmout_ogprdad - Natural gas associated dissolved production by HSM region
- ogsmout_ogqshlgas - Natural gas production by select natural gas play
- ogsmout_ogqngrep - Natural gas production by natural gas type
- ogsmout_ogprdugr - Lower 48 unconventional natural gas production
- ogsmout_ogdngprd - Dry natural gas production by state/district and gas type
- ogsmout_ogngprd - Natural Gas production by natural gas category
- ogsmout_ogngprdgom - Natural gas production Gulf of America
- ogsmout_ogprdadof - Offshore AD natural gas production (BCF)
- ogsmout_ogprdoff - Lower 48 offshore production split into Federal/State categories

Crude Oil and Natural Gas Production Restart Variable

- ogsmout_ogregprd - Total crude oil and natural gas production by production type

Crude Oil Price Restart Variables

- ogsmout_ogcowhp - Crude oil wellhead price by HSM region
- ogsmout_ogpcrwhp - Crude oil HSM average wellhead price

Natural Gas Price Restart Variable

- ogsmout_ogngwhp - Natural gas wellhead price by HSM region
- ogsmout_ogpngwhp - Average national wellhead price

Well Restart Variables

- ogsmout_ogogwells - Total wells
- ogsmout_ognowell - Total completed wells
- ogsmout_ogwellsl48 - Total lower 48 wells
- ogsmout_ogsrl48 - Lower 48 drilling success rates

NGPL Production Restart Variables

- ogsmout_ogngplprd - NGPL production by HSM district
- ogsmout_ogngplet - Ethane production by HSM district
- ogsmout_ogngplpr - Propane production by HSM district
- ogsmout_ogngplbu - Butane production by HSM district
- ogsmout_ogngplis - Isobutane production by HSM district
- ogsmout_ogngplpp - Pentanes production by HSM district
- ogsmout_ognglak - Alaska ngpl production

EOR Restart Variables

- ogsmout_ogco2rec - CO2 recycled by HSM region and CO2 type
- ogsmout_ogco2inj - CO2 injected by HSM region and CO2 type
- ogsmout_ogco2pur - CO2 purchased by HSM region and CO2 type
- ogsmout_ogco2avl - CO2 available by HSM region and CO2 type
- ogsmout_ogco2prc - CO2 price by HSM region and CO2 type
- ogsmout_ns_start -
- ogsmout_ogco245q -

Canada Restart Variables

- ogsmout.cnenagprd - Expected NA gas production in Canada (1=east, 2=west)
- ogsmout.cnrnagprd - Realized NA gas production in Canada (1=east, 2=west)
- ogsmout.cnadagprd - AD gas production in Canada (1=east, 2=west)

NGP Restart Variables

- qmore.qngpin - Natural Gas Processing Plant Electricity Consumption during Carbon Capture
- ogsmout.ngpco2em - CO2 emissions from Natural Gas Processing Plants

Technology Improvement Rate Restart Variable

- ogsmout_ogtechon - HSM technology improvement rate

Miscellanous Restart Variables

- ogsmout_oggrowfac - HSM growth factor (deprecated)
- ogsmout_ogjobs - Oil and gas industry jobs
- ogsmout_ogprcexp - HSM price expectation adj (deprecated)

Output debug files

rest_all.xlsx - Debug of HSM output restart variables

Restart submodule class methods

class restart_unf.**Restart**(*parent*)

Bases: object

parent

module.Module head module

output_path

output path

df_dict_in

df_dict_out

int_dict_in

int_dict_out

param_mappings

Cached parameter mapping dictionaries

setup(*temp_rest*)

Setup function for reading the restart file into HSM.

Parameters

temp_rest (str) – Filename for main NEMS restart file

Return type

None

run(*temp_rest*)

Calls the read_filer method from self.parent.pyfiler1 and loads the restart file into HSM.

- If main NEMS restart file exist, read abbreviated restart file
- If not, write abbreviated restart file as main restart file
- Read in all restart variables and create dictionary storing all restart variable keys

Parameters

temp_rest

Returns

- Restart File
- self.__dict__

add_int(*dict_name*, *restart_ref*, *output*)

Function for adding an integer from the restart file into HSM as an integer.

Parameters

- **dict_name** (str) – Name of restart variable or parameter in self.parent.pyfiler1
- **restart_ref** (int) – Integer data value from self.parent.pyfiler1
- **output** (bool) – Boolean of whether the restart variable is an output value of HSM

Returns

restart_ref – Integer value from the restart file

Return type

int

add_df(*dict_name*, *restart_ref*, *output*, *offset=None*, *dtype=None*)

Function for adding a dataframe from the restart file into NEMS as a local DataFrame.

Parameters

- **dict_name** (str) – Name of restart variable or parameter in self.parent.pyfiler1
- **restart_ref** (int) – Integer data value from self.parent.pyfiler1
- **output** (bool) – Boolean of whether the restart variable is an output value of HSM
- **offset** (list) – List indicating output df index value offset
- **dtype** (numpy.dtype, *optional*) – If provided, cast the array to this dtype before creating the DataFrame. Use np.float64 for variables that pyfiler returns as REAL*4 (float32) but require float64 precision for downstream .update() calls.

Returns

df – Restart variable df

Return type

pandas.DataFrame

write_results(*temp_filename*)

Repacks local restart file variables into multi-dimensional arrays, and writes to restart file.

- Loop through the variable dictionaries
- Split out each of the method levels into different variables
- Use getattr to link two method levels together
- Link the third method level with setattr, and set to value

Parameters

temp_filename (str) – Restart File name

Return type

None

write_restart_npz(*filepath*)

Write the current restart state to an NPZ file (standalone runs).

The output uses the same key convention as the input restart.npz consumed by restart_load.read_restart_file: keys are of the form “block/variable”, e.g. “ogsmout/ogenagprd”. All variables known to the Restart module (both inputs and outputs) are included.

Parameters
filepath (str) – Path to the output .npz file.

dump_restart()

Processes and reformats internal DataFrames by substituting index values with human-readable labels from reference CSVs, then writes each resulting table to a separate CSV file for inspection or debugging.

This method performs the following: - Loads a mapping file (master_table_selection.csv) to get metadata for each variable (description, dimensions, index types). - Reads substitution tables (CSV files named after index types) to map numeric codes to names. - Applies necessary *unstack* or transpose operations on data. - Substitutes index codes with readable names using the lookup dictionaries. - Writes each formatted DataFrame to its own CSV file, with a description in the first row.

history_steo — Historical and STEO Data Integration

history_steo.py implements the HistorySteo class for loading historical inputs, applying STEO adjustments, and preparing time-series inputs used during model runs.

This reference page lists module symbols and docstrings. For data provenance and debug outputs, see *Historical Data and STEO Overrides* and *Debugging and Utilities*. Module for reading in historical data and STEO forecasts, and applying this data to Restart variables.

This module reads in historical data and STEO forecasts, reformats the data to match the restart file, and updates the restart variables. The module operates as follows:

1. Load in historical data in *load_history* and reformat to match the restart variables. NGPLS are converted from OES NGPL regional totals to HSM regional totals
2. Overwrite restart variables with history in *overwrite_restart_var_history*.
3. Get historical shares of NGPLs from natural gas production in *calculate_ngpl_region_shares* to be used in STEO NGPL calculations.
4. Load in STEO forecast data in *load_steo_tables* and reformat to match the restart variables, apply NGPL shares calculated in *calculate_ngpl_region_shares* here.
5. Overwrite select restart variables with history in *overwrite_restart_var_steo*.
6. Aggregate updates to history and steo in "total" or "national" value rows for PMM output tables in *aggregate_pmm_output_totals*.

Input files

History input files are mostly derived from the OGSM_Database SAS preprocessor input file, while STEO Input files are derived from Python preprocessors.

History Input Files

- hist_rfqtdcrd.csv - Historical crude oil production by HSM region
- hist_ogqshloil.csv - Historical crude oil production by play
- hist_ogqshlgas.csv - Historical natural gas production by play
- hist_ngplprd.csv - Historical NGPL production by US state
- hist_htowells.csv - Historical successful tight oil wells drilled by HSM district
- hist_htoilprd.csv - Historical tight oil production by HSM district
- hist_htgwells.csv - Historical successful tight natural gas wells drilled by HSM district
- hist_htgasprd.csv - Historical tight natural gas production by HSM district
- hist_htadgprd.csv - Historical tight associated-dissolved natrual gas production by HSM district

- hist_hsgwells.csv - Historical successful shale gas wells drilled by HSM district
- hist_hsgasprd.csv - Historical successful shale natural gas wells drilled by HSM district
- hist_hdryholes.csv - Historical dryholes by HSM district
- hist_hcowells.csv - Historical successful non-tight oil wells drilled by HSM district
- hist_hcoilprd.csv - Historical non-tight oil production by HSM district
- hist_hcgwells.csv - Historical successful non-tight natural gas wells drilled by HSM district
- hist_hcgasprd.csv - Historical non-tight natural gas production by HSM district
- hist_hcbmwells.csv - Historical successful coalbed methane natural gas wells drilled by HSM district
- hist_hcbmprd.csv - Historical coalbed methane natural gas production by HSM district
- hist_hcadgprd.csv - Historical conventional associated dissolved natural gas production by HSM district
- hist_ognowell.csv - Historical wells drilled
- hist_eplp.csv - Historical pentanes production
- hist_epllpa.csv - Historical propane production
- hist_epllea.csv - Historical ethane production
- hist_epllban.csv - Historical butane production
- hist_epllbai.csv - Historical isobutane production
- hist_epl2.csv - Historical NGPL production
- hist_dcrdwhp.csv - Historical crude oil wellhead prices
- hist_eor_prod.csv - Historical EOR crude oil production
- hist_lfmm_prod.csv - Historical LFMM region and crude type crude oil production

STEO Input Files

- steo_dcrdwhp.csv - STEO year domestic crude oil wellhead price by region
- steo_ogngplbu.csv - STEO year butane production by HSM district
- steo_ogngplet.csv - STEO year ethane production by HSM district
- steo_ogngplis.csv - STEO year isobutane production by HSM district
- steo_ogngplpp.csv - STEO year pentanes production by HSM district
- steo_ogngplpr.csv - STEO year propane production by HSM district
- steo_ogngplprd.csv - STEO year NGPL production by HSM district
- steo_rfqtdcrd.csv - STEO year total crude production by HSM region
- steo_togqshlgas.csv - STEO year natural gas production by select tight oil play
- steo_togqshloil.csv - STEO year crude oil production by select tight oil play
- steo_ogenagprd.csv - STEO year natural gas expected production by natural gas type and HSM district
- steo_ogadgprd.csv - STEO year natural gas associated dissolved production by oil type and HSM district
- steo_can.csv - STEO year natural gas production by HSM Canada region and type

Model functions and class methods

class (sub.Submodule) - History and STEO submodule for HSM

- __init__ - Constructor to initialize History and STEO submodule
- load_history - Reads in history data from DI benchmarked to EIA reporting and reformats data for write to restart file
- overwrite_restart_var_history - Overwrites Restart variables history
- calculate_ngpl_region_shares - Breakout EIA History NGPL reporting for STEO years
- load_steo_tables - Load in STEO benchmark values
- overwrite_restart_var_steo - Overwrites Restart variables with STEO variables for PMMOUT variables used to calculate price
- load_steo_adjustments - Placeholder method (kept for backward compatibility, no longer loads data)
- aggregate_pmm_output_totals - Aggregate updates to history and steo in "total" or "national" value rows for PMM output tables

Output debug files

None

Output restart variables

Crude Oil Restart Variables

- pmmout_rfqtdcrd - Total crude production by HSM region
- pmmout_rfqdcrd - Total crude oil production by HSM region (not including EOR)
- ogsmout_ogqcrrep - Crude oil production by oil category
- ogsmout_ogcoprd - Crude oil production by lower 48 region
- ogsmout_ogqshloil - Crude oil production by select tight oil play
- ogsmout_ogoilprd - Crude oil production by oil type and HSM district
- ogsmout_ogcrdprd - Crude oil production by HSM region and crude type
- ogsmout_ogcruderef - Crude oil production by LFMM crude oil type and region
- ogsmout_ogprcoak - Crude oil production by Alaska region
- ogsmout_ogeorprd - CO2 EOR crude oil production

Natural Gas Restart Variables

- self.restart.ogsmout_ogenagprd - Natural gas expected production by natural gas type and HSM district
- self.restart.ogsmout_ogrnagprd - Natural gas realized production by natural gas type and HSM district
- self.restart.ogsmout_ogadgprd - Natural gas associated dissolved production by oil type and HSM district
- self.restart.ogsmout_ogqshlgas - Natural gas production by select natural gas play
- self.restart.ogsmout_ogqngrep - Natural gas production by natural gas type

Crude Oil and Natural Gas Production Restart Variable

- self.restart.ogsmout_ogregprd - Total crude oil and natural gas production by production type

Well Restart Variables

- self.restart.ogsmout_ogogwells - Total wells
- self.restart.ogsmout_ognowell - Total completed wells

NGPL Production Restart Variables

- self.restart.ogsmout_ogngplprd - NGPL production by HSM district
- self.restart.ogsmout_ogngplet - Ethane production by HSM district
- self.restart.ogsmout_ogngplpr - Propane production by HSM district
- self.restart.ogsmout_ogngplbu - Butane production by HSM district
- self.restart.ogsmout_ogngplis - Isobutane production by HSM district
- self.restart.ogsmout_ogngplpp - Pentanes production by HSM district

History and STEO submodule class methods

class history_steo.**hist_steo**(*parent*)

Bases: *Submodule*

ngpl_shares

dataframe of ngpl shares by region based on historical production

load_history()

Reads in history data from a commercial well-level production database benchmarked to EIA reporting and reformats data for write to restart file.

Returns

- **self.hist_rfqtdcrd** (pandas.DataFrame) – DataFrame of historical total crude production by HSM region
- **self.hist_ogqshloil** (pandas.DataFrame) – DataFrame of crude oil production by select tight oil play
- **self.hist_ogqshlgas** (pandas.DataFrame) – DataFrame of natural gas production by select natural gas play
- **self.hist_ngplprd** (pandas.DataFrame) – DataFrame of historical NGPL production by state
- **self.hist_htowells** (pandas.DataFrame) – DataFrame of historical successful tight oil wells drilled by HSM district
- **self.hist_htoilprd** (pandas.DataFrame) – DataFrame of historical tight oil production by HSM district
- **self.hist_htgwells** (pandas.DataFrame) – DataFrame of historical successful tight natural gas wells drilled by HSM district
- **self.hist_htgasprd** (pandas.DataFrame) – DataFrame of historical tight natrual gas production by HSM district
- **self.hist_htadgprd** (pandas.DataFrame) – DataFrame of historical tight associated-dissolved natrual gas production by HSM district
- **self.hist_hsgwells** (pandas.DataFrame) – DataFrame of historical successful shale gas wells drilled by HSM district
- **self.hist_hsgasprd** (pandas.DataFrame) – DataFrame of historical successful shale natural gas wells drilled by HSM district
- **self.hist_hdryholes** (pandas.DataFrame) – DataFrame of historical dryholes by HSM district
- **self.hist_hcowells** (pandas.DataFrame) – DataFrame of historical successful non-tight oil wells drilled by HSM district

- **self.hist_hcoilprd** (pandas.DataFrame) – DataFrame of historical non-tight oil production by HSM district
- **self.hist_hcgwells** (pandas.DataFrame) – DataFrame of historical successful non-tight natural gas wells drilled by HSM district
- **self.hist_hcgasprd** (pandas.DataFrame) – DataFrame of historical non-tight natural gas production by HSM district
- **self.hist_hcbmwells** (pandas.DataFrame) – DataFrame of historical successful coalbed methane natural gas wells drilled by HSM district
- **self.hist_hcbmprd** (pandas.DataFrame) – DataFrame of historical coalbed methane natural gas production by HSM district
- **self.hist_hcadgprd** (pandas.DataFrame) – DataFrame of historical conventional associated dissolved natural gas production by HSM district
- **self.hist_ognowell** (pandas.DataFrame) – DataFrame of historical wells drilled
- **self.hist_eplp** (pandas.DataFrame) – DataFrame of historical pentanes production
- **self.hist_epllpa** (pandas.DataFrame) – DataFrame of historical propane production
- **self.hist_epllea** (pandas.DataFrame) – DataFrame of historical ethane production
- **self.hist_epllban** (pandas.DataFrame) – DataFrame of historical butane production
- **self.hist_epllbai** (pandas.DataFrame) – DataFrame of historical isobutane production
- **self.hist_epl2** (pandas.DataFrame) – DataFrame of total historical NGPL production
- **self.hist_eor_prod** (pandas.DataFrame) – DataFrame of historical EOR crude oil production
- **self.hist_lfmm_prod** (pandas.DataFrame) – DataFrame of historical LFMM region and crude type crude oil production
- **self.hist_dcrdwhp** (pandas.DataFrame) – DataFrame of historical crude oil wellhead prices

Notes

overwrite_restart_var_history()

Overwrites Restart variables history.

Returns

- **self.restart.pmmout_rfqtdcrd** (pandas.DataFrame) – Total crude production by HSM region
- **self.restart.pmmout_rfqdcrd** (pandas.DataFrame) – Total crude oil production by HSM region (not including EOR)
- **self.restart.ogsmout_ogcrdprd** (pandas.DataFrame) – Crude oil production by HSM region and LFMM crude oil type
- **self.restart.ogsmout_ogcruderef** (pandas.DataFrame) – Crude oil production by LFMM crude oil type and region
- **self.restart.ogsmout_ogoilprd** (pandas.DataFrame) – Crude oil production by oil type and HSM district
- **self.restart.ogsmout_ogqshloil** (pandas.DataFrame) – Crude oil production by select tight oil play

- **self.restart.ogsmout_ogprcoak** (pandas.DataFrame) – Crude oil production by Alaska region
- **self.restart.ogsmout_ogeorprd** (pandas.DataFrame) – CO2 EOR crude oil production
- **self.restart.pmmout_dcrdwhp** (pandas.DataFrame) – Domestic crude oil wellhead price by region
- **self.restart.ogsmout_ogenagprd** (pandas.DataFrame) – Natural gas expected production by natural gas type and HSM district
- **self.restart.ogsmout_ogrnagprd** (pandas.DataFrame) – Natural gas realized production by natural gas type and HSM district
- **self.restart.ogsmout_ogadgprd** (pandas.DataFrame) – Natural gas associated dissolved production by oil type and HSM district
- **self.restart.ogsmout_ogngprd** (pandas.DataFrame) – Natural gas production by type
- **self.restart.ogsmout_ogqngrep** (pandas.DataFrame) – Natural gas production by natural gas type
- **self.restart.ogsmout_ogqshlgas** (pandas.DataFrame) – Natural gas production by select natural gas play
- **self.restart.ogsmout_ogregprd** (pandas.DataFrame) – Total crude oil and natural gas production by production type
- **self.restart.ogsmout_ogogwells** (pandas.DataFrame) – Total wells
- **self.restart.ogsmout_ogngplprd** (pandas.DataFrame) – NGPL production by HSM district
- **self.restart.ogsmout_ogngplpp** (pandas.DataFrame) – Pentanes production by HSM district
- **self.restart.ogsmout_ogngplpr** (pandas.DataFrame) – Propane production by HSM district
- **self.restart.ogsmout_ogngplet** (pandas.DataFrame) – Ethane production by HSM district
- **self.restart.ogsmout_ogngplbu** (pandas.DataFrame) – Butane production by HSM district
- **self.restart.ogsmout_ogngplis** (pandas.DataFrame) – Isobutane production by HSM district
- **self.restart.ogsmout_ogngplprd** (pandas.DataFrame) – NGPL production by HSM district
- **self.restart.ogsmout_ogqcrrep** (pandas.DataFrame) – Crude production by oil category
- **self.restart.ogsmout_ogcoprd_nonfed** (pandas.DataFrame) – Crude production on non-federal lands
- **self.restart.ogsmout_ogcoprd_fed** (pandas.DataFrame) – Crude production on federal lands
- **self.restart.ogsmout_ogngprd_nonfed** (pandas.DataFrame) – Natural Gas production on non-federal lands
- **self.restart.ogsmout_ogngprd_fed** (pandas.DataFrame) – Natural Gas production on federal lands

calculate_ngpl_region_shares()

EIA STEO reporting for NGPL values has different regions than HSM, this equation splits out the regions
proportionally so that STEO NGPL regions are matched to HSM regions.

Returns
self.ngpl_shares – DataFrame of historical ngpl shares by HSM regions

Return type
pandas.DataFrame

load_steo_tables()

Load in STEO benchmark values.

Returns

- **self.steo_dcrdwhp** (pandas.DataFrame) – DataFrame of STEO year domestic crude oil wellhead price by region
- **self.steo_rfqtdcrd** (pandas.DataFrame) – DataFrame of STEO year total crude production by HSM region
- **self.steo_ogenagprd** (pandas.DataFrame) – DataFrame of STEO year natural gas expected production by natural gas type and HSM district
- **self.steo_ogadgprd** (pandas.DataFrame) – DataFrame of STEO year natural gas associated dissolved production by oil type and HSM district
- **self.steo_can** (pandas.DataFrame) – DataFrame of STEO year natural gas production by HSM Canada region and type
- **self.steo_togqshloil** (pandas.DataFrame) – DataFrame of STEO year crude oil production by select tight oil play
- **self.steo_togqshlgas** (pandas.DataFrame) – DataFrame of STEO year natural gas production by select tight oil play
- **self.steo_ogngplprd** (pandas.DataFrame) – DataFrame of STEO year NGPL production by HSM district
- **self.steo_ogngplet** (pandas.DataFrame) – DataFrame of STEO year ethane production by HSM district
- **self.steo_ogngplpr** (pandas.DataFrame) – DataFrame of STEO year propane production by HSM district
- **self.steo_ogngplbu** (pandas.DataFrame) – DataFrame of STEO year butane production by HSM district
- **self.steo_ogngplis** (pandas.DataFrame) – DataFrame of STEO year isobutane production by HSM district
- **self.steo_ogngplpp** (pandas.DataFrame) – DataFrame of STEO year pentanes production by HSM district

overwrite_restart_var_steo()

Overwrites Restart variables with STEO variables for PMMOUT variables used to calculate price.

Returns

- **self.restart.pmmout_dcrdwhp** (pandas.DataFrame) – Domestic crude oil wellhead price by region

- **self.restart.pmmout_rfqtdcrd** (pandas.DataFrame) – Total crude production by HSM region

load_steo_adjustments()

Load in STEO overwrite values.

This method is kept for backward compatibility but no longer loads any data. STEO overwrites for side cases are now handled by apply_side_case_steo_overwrites() in prepare_results.py, which uses files from input/side_case_owrites/ directory.

Return type
None

aggregate_pmm_output_totals()

Aggregate updates to history and steo in "total" or "national" value rows for PMM output tables.

- pmmout_rfqtdcrd row 14 is for Alaska production
- pmmout_rfqtdcrd row 15 is for Onshore/Offshore production
- pmmout_rfqtdcrd row 16 is for total production
- pmmout_rfqdcrd rows 14, 15, 16 are aggregated the same way (Alaska, Onshore/Offshore, total)
- pmmout_dcrdwhp row 14 is for national weighted average price

Returns

- **self.restart.pmmout_rfqtdcrd** (pandas.DataFrame) – DataFrame of STEO overwrite values for crude oil production
- **self.restart.pmmout_dcrdwhp** (pandas.DataFrame) – Oh Domestic crude oil wellhead price by region

prepare_results — Output Aggregation and Mapping

prepare_results.py defines the PrepareResults class that aggregates submodule outputs and maps them into NEMS output array structures before restart-file writeback.

This reference page lists module symbols and docstrings. For output catalog and data-flow context, see *Output Variables Reference* and *NEMS Integration*. File for preparing results prior to transmission to the restart file.

Summary

This file contains the operations required to prepare results for transmission to the restart file, these methods include:

1. aggregate_results_across_submodules -combines production volumes across submodules for cumulative variables.
2. set_expected_production_to_steo - overwrites calculated production volumes with STEO production values.
3. adjust_production - respond to actual natural gas demand volumes provided by NGMM.

Model functions and class methods

- aggregate_results_across_submodules -combines production volumes across submodules for cumulative variables.
- set_expected_production_to_steo - overwrites calculated production volumes with STEO production values.
- adjust_production - respond to actual natural gas demand volumes provided by NGMM.

Input files

None

Output debug files

None

Output restart variables

None

Notes

None

Prepare results methods

class prepare_results.**Prep_Results**(*parent*)

Bases: *Submodule*

Class for preparing results prior to transmission to the restart file.

ONSHORE_REGIONS = [1, 2, 3, 4, 5, 6, 7]

OFFSHORE_REGIONS = [8, 9, 10]

ONSHORE_LFMM_REGIONS = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13]

ALASKA_REGION = 14

PRODUCTION_CATEGORIES = {1: 'EOR', 2: 'L48_Onshore', 3: 'Offshore', 4: 'Alaska'}

ONSHORE_MAX_DISTRICT = 66

OFFSHORE_MIN_DISTRICT = 67

ALASKA_ONSHORE_DISTRICT = 3

ALASKA_STATE_OFFSHORE_DISTRICT = 75

ALASKA_FEDERAL_OFFSHORE_DISTRICT = 84

QNGREP_SHALE = 1

QNGREP_CBM = 2

QNGREP_TIGHT = 3

QNGREP_CONV = 4

COLUMN_ALIASES = {'gastypes': 'gastyp', 'oiltypes': 'oiltyp', 'region_number': 'mnumor'}

QNGREP_OTHER_AD = 5

QNGREP_OFFSHORE_NA = 6

QNGREP_OFFSHORE_AD = 7

QNGREP_ALASKA = 8

QNGREP_TOTAL = 9

NA_GAS_CONV = 1

NA_GAS_TIGHT = 2

NA_GAS_SHALE = 3

NA_GAS_CBM = 4

NA_GAS_TOTAL = 5

AD_GAS_CONV = 1

AD_GAS_TIGHT = 2

AD_GAS_CO2_EOR = 3

AD_GAS_OTHER_EOR = 4

AD_GAS_TOTAL = 5

REGPRD_CONV = 4

REGPRD_TIGHT = 5

REGPRD_SHALE = 6

REGPRD_CBM = 7

SHALE_TIGHT_WELL_TYPES = [4, 5]

SHALE_AD_WELL_TYPES = [2, 5]

CONV_WELL_TYPES = [1, 3]

CBM_WELL_TYPE = 6

BCF_TO_TCF = 1000000000

OGQSHLGAS_OTHER = 15

aggregate_results_across_submodules()

Calculate sum values in restart variable tables that are aggregated across submodules.

Returns

- **self.parent.restart.ogsmout_ogoilprd** (pandas.DataFrame) – Crude oil production by HSM district
- **self.parent.restart.ogsmout_ogenagprd** (pandas.DataFrame) – Expected non-associated natural gas production by HSM district and type
- **self.parent.restart.ogsmout_ogqngrep** (pandas.DataFrame) – Natural gas production by natural gas type
- **self.parent.restart.ogsmout_ogadgprd** (pandas.DataFrame) – Associated-dissolved natural gas production by HSM district and type
- **self.parent.restart.pmmout_rfqtdcrd** (pandas.DataFrame) – Total crude production by region and type (including EOR)
- **self.parent.restart.pmmout_rfqdcrd** (pandas.DataFrame) – Total crude production by region and type (not including EOR)
- **self.parent.restart.ogsmout_ogjobs** (pandas.DataFrame) – Oil and gas industry jobs (based on legacy equation developed by Dana Van Wagener)

- **self.parent.restart.qmore_qngpin** (pandas.DataFrame) – Electricity consumed by natural gas processing plants during carbon capture
- **self.parent.restart.ogsmout_ogcrdprd** (pandas.DataFrame) – Crude production by LFMM region and LFMM crude type
- **self.parent.restart.ogsmout_ngpco2em** (pandas.DataFrame) – Emissions from natural gas processing plants
- **self.parent.restart.ogsmout_ogqcrrep** (pandas.DataFrame) – Crude oil production by oil category

adjust_fed_nonfed_gas()

Calculates and updates federal and non-federal split adjusted for realized natural gas.

- Adjusts total natural gas production for realized natural gas feedback
- Maps production to HSM regions
- Calculates fed/nonfed gas ratio
- Applies fed/nonfed ratio to regional production

Returns

- **self.restart.ogsmout_ogngprd_nonfed** (pandas.DataFrame) – Natural Gas production on non-federal lands
- **self.restart.ogsmout_ogngprd_fed** (pandas.DataFrame) – Natural Gas production on federal lands

adjust_production(*mode*)

Adjust restart variables for natural gas production.

This unified method combines non-associated gas (from gas wells) and associated-dissolved gas (from oil wells) to update various production tables.

Replaces the three separate methods: - adjust_vars_for_year_one_prod() -> mode='year_one' - adjust_vars_for_realized_prod() -> mode='realized' - adjust_regional_natgas_for_standalone() -> mode='standalone'

Ratio Adjustment Behavior: For all integrated years (mode='year_one' or 'realized'), applies ratio adjustment to scale expected production by type to match NGMM realized totals. All years >= steo_years[0] are treated the same with this ratio adjustment.

Shale Gas Plays Adjustment: For integrated years when current_year > steo_years[0], also calls adjust_onshore_shale_gas_plays_for_realized_prod() to update play-level production, well counts, and project drilling parameters.

Parameters

mode (str) –

'year_one' - Integrated mode for all years >= steo_years[0]. Applies ratio adjustment to realized non-associated gas, then updates all production variables. All integrated years use the same ratio adjustment approach.

'realized' - Same as 'year_one' (kept for backward compatibility). Both modes now apply ratio adjustment to all integrated years.

'standalone' - Standalone mode. Runs the same updates as integrated (ogqngrep, ogdngprd, fed/nonfed, ogngprd, ogregprd, and first-STEO-year OGQSHLGAS "other" adjustment) using expected non-associated gas (ogenagprd). No ratio adjustment applied.

Returns

- Updates the following restart variables depending on mode
- **- ogsmout_ogqngrep** (NG production by category (all modes))
- **- ogsmout_ogngprd** (NG production by region (all modes))
- **- ogsmout_ogdngprd** (Dry NG by district/type (all modes))
- **- ogsmout_ogregprd** (Regional production by type (all modes))
- **- ogsmout_ogrnagprd** (Realized non-associated gas (integrated modes, via ratio adjustment))
- **- ogsmout_ogngprd_fed/nonfed** (Fed/nonfed split (all modes))
- **- ogsmout_ogqshlgas** (First STEO year "other" adjustment (all modes); play-level) – scaling (integrated only, years > steo_years[0])
- **- ogsmout_ogogwells** (Total wells (integrated only, years > steo_years[0]))
- **- ogsmout_ognowell** (Completed wells (integrated only, years > steo_years[0]))
- **- ogsmout_ogwellsl48** (Lower 48 wells (integrated only, years > steo_years[0]))
- **- ogsmout_ogsrl48** (Drilling success rates (integrated only, years > steo_years[0]))

adjust_onshore_shale_gas_plays_for_realized_prod()

Scales onshore natural gas play production to NGMM realized demand.

This method applies realized production ratios to onshore shale/tight gas production, well counts, and project drilling parameters to match NGMM feedback. It updates play-level production variables and restart variables.

Returns

- Updates the following restart variables
- **- ogsmout_ogqshlgas** (Natural gas production by select natural gas play)
- **- ogsmout_ogogwells** (Total wells)
- **- ogsmout_ognowell** (Total completed wells)
- **- ogsmout_ogwellsl48** (Total lower 48 wells)
- **- ogsmout_ogsrl48** (Lower 48 drilling success rates)
- **- ogsmout_ogregprd** (Total crude oil and natural gas production by production type)
- Also updates onshore internal data structures
- **- natgas_production** (Natural gas production by project)
- **- wells** (DataFrame of onshore wells)
- **- projects** (Master DataFrame containing all projects)

apply_steo_benchmarks_fed_nonfed()

Applies STEO benchmarks to federal and non-federal oil production data.

- Extracts relevant data for the current year, including total proved reserves (rfqtdcrd) and federal/non-federal oil production (ogcoprd_fed, ogcoprd_nonfed).

- Calculates the ratio of federal to total (federal + non-federal) oil production for each region.
- Merges the total proved reserves data with the calculated federal/non-federal ratio.
- Calculates the federal and non-federal benchmarks by applying the federal ratio to the total proved reserves.
- Updates the parent's restart data with the calculated federal and non-federal benchmarks for oil production.

Returns

- **self.restart.ogsmout_ogcoprd_fed** (pandas.DataFrame) – Federal oil production with updated benchmarks
- **self.restart.ogsmout_ogcoprd_nonfed** (pandas.DataFrame) – Non-federal oil production with updated benchmarks

apply_steo_fixed_benchmarks()

Overwrites model production with STEO values for first two STEO years.

When This Runs: - Always called for years in *steo_years[0:2]* (first two STEO years) - No flag required - this is the primary STEO benchmarking method - Called in *module_unf.py* line 1506 when *current_year in steo_years[0:2]*

What This Does: - Directly overwrites model-calculated production with STEO forecast values - Updates crude oil, natural gas, and NGPL production variables to match STEO - This is the first step in STEO benchmarking - sets the baseline STEO values

This method overwrites restart variables with reference case values from CSV files.

Returns

Updates multiple restart variables in place: - pmmout_rfqtdcrd: Total crude production (including EOR) - pmmout_rfqdcrd: Non-EOR crude production - ogsmout_ogcoprd: Crude oil production by lower 48 region (onshore regions 1-7 and offshore regions 9-10) - ogsmout_ogqshloil: Tight oil production from select tight oil plays - ogsmout_ogqshlgas: Shale natural gas production from select shale natural gas plays - ogsmout_ogenagprd: Expected non-associated natural gas (STEO benchmarked) - ogsmout_ogrnagprd: Realized natural gas (updated to match ogenagprd) - ogsmout_ogngplprd: Total NGPL production by HSM district - ogsmout_ogngplet: Ethane production by HSM district - ogsmout_ogngplpr: Propane production by HSM district - ogsmout_ogngplbu: Butane production by HSM district - ogsmout_ogngplis: Isobutane production by HSM district - ogsmout_ogngplpp: Pentanes production by HSM district - ogsmout_ogregprd: Regional production by type (crude oil types only) - ogsmout_ogeorprd: EOR production by region and type

Return type

None

apply_side_case_steo_overwrites()

Overwrite restart variables with reference case values from preprocessed NPZ file.

Only runs for side cases in first STEO year when side_case_steo_overwrite_switch == True. Variables that are already handled by apply_steo_fixed_benchmarks() are skipped.

This method loads preprocessed data from input/side_case_owrites/steo_overwrite_data.npz and directly overwrites restart variables for the first STEO year only.

The NPZ file is generated by running preprocess_steo_overwrites.py on the cb case restart.npz file. This preprocessing step extracts single-year data slices, eliminating the need for complex dimension handling at runtime.

2.11.6 Utilities

common — Utility Functions

common.py contains utility functions shared across all HSM modules. Key functions include read_dataframe (CSV loading with error handling), calculate_inflation (nominal-to-real price conversion), and df_interpolate_2 (field-size class interpolation used in the offshore submodule).

This reference page lists module symbols and docstrings. Module containing miscellaneous common functions used in hsm.

Notes

Convention for import alias is import common as com

common.**print_out**(*text_in*)

Prints text to output with tabs denoting stack depth.

Parameters
text_in (str, int, float)

Return type
None

common.**read_dataframe**(*filename*, *sheet_name=0*, *index_col=None*, *skiprows=None*, *to_int=True*)

Prints text to output with tabs denoting stack depth.

Parameters

- **filename** (str) – Filename, including file type extension (e.g. .csv)
- **sheet_name** (str) – Name of the sheet if using excel or hdf
- **index_col** (int, str, list, or False) – Column number to use as row labels
- **skiprows** (int) – Column number to use as row labels

Return type
pandas.DataFrame

common.**calculate_inflation**(*rest_mc_jpgdp*, *from_year*, *to_year=None*)

Calculates inflation.

Return type
self.rest_mc_jpgdp

common.**assign_ngpls_to_district**(*df*, *ngpl_shares*, *steo_years*)

Assigns historic ngpl volumes to a district number based on ngpl shares.

Parameters

- **df** (pandas.DataFrame) – Dataframe of historic ngpl production
- **ngpl_shares** (pandas.DataFrame) – DataFrame of NGPL shares by HSM district
- **steo_years** (list) – List of STEO years

Return type
pandas.DataFrame

common.**df_interpolate_2**(*left*, *right*, *xcol*, *ycol*, *xdir='nearest'*, *ydir='nearest'*, *xmerge='merge_asof'*)

Merges field size classes in the Offshore submodule.

Parameters

- **left** (pandas.DataFrame) – Left merge df
- **right** (pandas.DataFrame) – Right merge df
- **xcol** – Left merge column
- **ycol** – Right merge column
- **xdir** – Left direction for "merge_asof"
- **ydir** – Right direction for "merge_asof"
- **xmerge** (str) – Type of merge

Return type

pandas.DataFrame

common.**leg_cost_conversion**(*cost*, *capacity*, *discount_rate*, *plant_life*, *operating_factor*)

! calculate costs in 10**6 dollars (see reference (2)).

- Then convert to $/MCF via subroutine calc_ucc, which assumes that the capacity is in MMCF/day
- Costs for each impurity will be generated only if impurity %'s are above associated limits
- input cost = capital costs ($10**6)
- ac = annual capital charge ($10**6/year)
- output cost = unitized capital cost ($/MCF)
- avt = average volume thoughput ($10**6 cubic feet/day) (calendar basis)
- Note: capacity is expressed in MMCF/day)

common.**array_to_df**(*array*)

Create df from array.

- Creates a multi-index DataFrame from a multi-dimensional array
- Each array dimension is stored as a binary index column in the DataFrame multi-index
- Multi-index columns are currently numbered to reflect array dimension level

Parameters

series (array) – A multi-dimensional array containing restart file data

Return type

pandas.DataFrame

common.**df_to_array**(*df*)

Turns multi-index DataFrame into multi-dimensional array.

- Creates a multi-dimensional array from a multi-index dataframe
- Each DataFrame multi-index is stored as an array dimension

Parameters

series (pandas.DataFrame) – A multi-index DataFrame containing restart file data

Return type

array

`names` — String Constants

`names.py` defines string constants for all DataFrame column names, index names, and variable identifiers used throughout HSM. Importing `names as nam` is the standard pattern. Using constants from this module rather than literal strings prevents typo bugs and enables IDE autocomplete.

This reference page lists module symbols and docstrings. File containing strings used in hsm.

> **Note**
>
> Convention for import alias is, import names as nam

`intermediate_var_pickle` — Pickle State Management

`intermediate_var_pickle.py` provides functions for reading and writing the `fcrl_var.pkl` and `ncrl_var.pkl` files that preserve model state between iterations and between years.

This reference page lists module symbols and docstrings. For debugging workflow usage, see *Debugging and Utilities*. Class for handling intermediate variables in HSM.

Notes

- In OGSM we kept cumulative variables in Fortran working memory (i.e. projects each year)
- In HSM this isn't possible, so instead these tables are written to Pickle (PKL) files which are stored in the hsm directory between model years
- In year 1 the PKLs are created, and then in subsequent years they are written to and read
- The PKL files are only written out when fcrl = 1 (final regular iteration) or ncrl = 1 (report iteration) to avoid duplication during other NEMS iterations
- There are therefore two sets of .pkl files, one "fcrl" set for regular iterations, and one "ncrl" set for reporting iterations
- All .pkl filenames and corresponding df names match the relevant corresponding submodule variable names (i.e. on_projects_xxxx.pkl = self.projects in the onshore submodule).

The Pickle methods operate as follows:

1. The class is initialized in the intermediate_vars *__init__* method, with class being assigned as a child to **module_unf** here.
2. *read_pkl_vars* is called from **module_unf.py** in all model years excluding year 1. In this method all required .pkl local, iterative variable files are read into HSM as class dataframes accessible by all other modules in HSM.
3. *write_pkl_vars* is called from **module_unf.py** in all model years. In this method all required local, iterative dataframes are written to .pkl files.

class intermediate_var_pickle.**intermediate_vars**(*parent*)

Bases: `object`

parent

module.Module head module

read_pkl_vars(*itr_type*, *current_year*, *temp_filepath*)

Read in Pickle Intermediate Tables.

Return type

PKL Dataframes

write_pkl_variables(*itr_type*, *current_year*, *temp_filepath*)

Write out Pickle Intermediate Tables.

Return type
PKL Files

visualizations — Chart Generation

visualizations.py generates diagnostic charts after standalone model runs when charts_switch = TRUE in setup.csv. Charts are configured via input/chart_config.csv.

This reference page lists module symbols and docstrings. Chart visualization module for HSM debugging.

This module provides a flexible system for generating charts from HSM restart variables based on CSV configuration files. It supports multiple chart types (line, bar, stacked_area) and various aggregation methods.

Summary

The visualization system allows users to configure which charts are displayed after a run via a CSV configuration file. Charts can be displayed in windows following an offline HSM run, saved to files, or both.

Functions

- generate_charts - Main entry point that reads config and generates charts
- load_chart_config - Loads and validates chart configuration CSV
- create_chart - Creates a single chart based on config
- get_variable_data - Extracts and filters data from restart object
- get_variable_metadata - Loads dimension info from variable_mapping
- parse_filter_string - Parses filter strings like "1:4" or "1,2,3,4"
- parse_year_range - Parses year range strings
- create_line_chart - Creates a line chart
- create_bar_chart - Creates a bar chart
- create_stacked_area_chart - Creates a stacked area chart
- add_value_annotations - Adds value annotations to charts

visualizations.**generate_charts**(*hsm*, *config_path*)

Main entry point that reads config and generates charts.

Parameters

- **hsm** (Module) – HSM module object containing restart data
- **config_path** (str) – Path to chart configuration CSV file

Return type
None

visualizations.**load_chart_config**(*config_path*)

Loads and validates chart configuration CSV.

Parameters
config_path (str) – Path to chart configuration CSV file

Returns
List of chart configuration dictionaries

Return type
List[Dict]

visualizations.**get_variable_metadata**(*var_name*, *var_mapping_path*)

Loads dimension info from variable_mapping.

Parameters

- **var_name** (str) – Variable name (e.g., 'ogsmout_ogqcrrep')
- **var_mapping_path** (str) – Path to variable_mapping directory

Returns
Dictionary containing variable metadata: - dimension: Number of dimensions - index_names: List of index names - units: Units string - description: Variable description

Return type
Dict

visualizations.**get_dimension_labels**(*index_name*, *var_mapping_path*)

Get labels for dimension values from parameter reference tables.

Parameters

- **index_name** (str) – Name of the index (e.g., 'nogcro')
- **var_mapping_path** (str) – Path to variable_mapping directory

Returns
Dictionary mapping index values to labels

Return type
Dict[int, str]

visualizations.**parse_filter_string**(*filter_str*)

Parses filter strings like "1:4" or "1,2,3,4" or "1:10,15".

Parameters
filter_str (str) – Filter string (e.g., "1:4" or "1,2,3,4" or "1:10,15")

Returns
List of integer values

Return type
List[int]

visualizations.**parse_year_range**(*year_range_str*)

Parses year range strings.

Parameters
year_range_str (str) – Year range string (e.g., "2020:2050" or "2020,2025,2030,2040,2050")

Returns
List of years

Return type
List[int]

visualizations.**get_variable_data**(*hsm*, *var_name*, *filters*, *year_range*)

Extracts and filters data from restart object.

Parameters

- **hsm** (Module) – HSM module object

- **var_name** (str) – Variable name (e.g., 'ogsmout_ogqcrrep')
- **filters** (Dict) – Dictionary with keys 'dim0', 'dim1', 'dim2' containing filter lists
- **year_range** (List[int]) – List of years to include

Returns
Filtered and processed data

Return type
pandas.DataFrame

visualizations.**create_chart**(*hsm*, *chart_config*, *var_metadata*, *var_mapping_path*)

Creates a single chart based on config.

Parameters

- **hsm** (Module) – HSM module object
- **chart_config** (Dict) – Chart configuration dictionary
- **var_metadata** (Dict) – Variable metadata dictionary
- **var_mapping_path** (str) – Path to variable_mapping directory

visualizations.**prepare_chart_data**(*value_series*, *aggregation*, *var_metadata*, *dim_labels*)

Prepares data for charting based on aggregation method.

Parameters

- **value_series** (pandas.Series) – Series with MultiIndex and values
- **aggregation** (str) – Aggregation method: 'total', 'by_dim0', 'by_dim1', 'by_dim2'
- **var_metadata** (Dict) – Variable metadata
- **dim_labels** (Dict) – Dictionary of dimension labels

Returns
(years, data, series_labels)

Return type
Tuple[List[int], Union[pd.Series, pd.DataFrame], Dict]

visualizations.**build_chart_title**(*chart_config*, *var_metadata*, *dim_labels*)

Builds a comprehensive chart title with filter information and other details. Returns multi-line title if too long, keeping base title on first line.

Parameters

- **chart_config** (Dict) – Chart configuration dictionary
- **var_metadata** (Dict) – Variable metadata dictionary
- **dim_labels** (Dict) – Dictionary of dimension labels (maps dimension index to {value: label} dict)

Returns
Complete chart title; may include newline characters for multi-line titles.

Return type
str

visualizations.**add_value_annotations**(*ax*, *years_list*, *values_series*, *label_years=[2025, 2030, 2040, 2050]*)

Add text annotations showing values for specific years.

Parameters

- **ax** (matplotlib.axes.Axes) – Axes object to annotate
- **years_list** (List[int]) – List of years
- **values_series** (Union[pd.Series, np.ndarray]) – Series or array of values
- **label_years** (List[int]) – Years to annotate

visualizations.**create_line_chart**(*var_df*, *value_series*, *chart_config*, *var_metadata*, *dim_labels*, *aggregation*)

Creates a line chart.

Parameters

- **var_df** (pandas.DataFrame) – Variable DataFrame
- **value_series** (pandas.Series) – Value series
- **chart_config** (Dict) – Chart configuration
- **var_metadata** (Dict) – Variable metadata
- **dim_labels** (Dict) – Dimension labels
- **aggregation** (str) – Aggregation method

visualizations.**create_bar_chart**(*var_df*, *value_series*, *chart_config*, *var_metadata*, *dim_labels*, *aggregation*)

Creates a bar chart.

Parameters

- **var_df** (pandas.DataFrame) – Variable DataFrame
- **value_series** (pandas.Series) – Value series
- **chart_config** (Dict) – Chart configuration
- **var_metadata** (Dict) – Variable metadata
- **dim_labels** (Dict) – Dimension labels
- **aggregation** (str) – Aggregation method

visualizations.**create_stacked_area_chart**(*var_df*, *value_series*, *chart_config*, *var_metadata*, *dim_labels*, *aggregation*)

Creates a stacked area chart.

Parameters

- **var_df** (pandas.DataFrame) – Variable DataFrame
- **value_series** (pandas.Series) – Value series
- **chart_config** (Dict) – Chart configuration
- **var_metadata** (Dict) – Variable metadata
- **dim_labels** (Dict) – Dimension labels
- **aggregation** (str) – Aggregation method

pyscedes — SCEDES File Parser

pyscedes.py parses the SCEDES scenario configuration file used in standalone runs. The User class reads the CSV-format SCEDES file and exposes scenario key-value pairs as attributes.

This reference page lists module symbols and docstrings. PyScedesAll Created on April 4 2023 @author: jmw updated on May 21 2025: ads

PyScedesAll is a Python function for parsing the scedes.all file into a user class that can be passed between Python programs for use in NEMS. This code sets a dictionary of scedes keys in the user class.

class pyscedes.**User**(*scedes_dict*)

Bases: object

pyscedes.**find_scedes**(*scedes_name*)

Locates the absolute path to a SCEDES file in a predefined directory.

Constructs a path to the SCEDES file based on *scedes_name* and a relative path structure (two levels above the current script, under the 'scedes' folder).

Args:
scedes_name (str): The name of the SCEDES file or directory.

Returns:
str: The absolute path to the SCEDES file if found.

Raises:
SystemExit: If the SCEDES file is not found, the program exits with an error message.

pyscedes.**parse_scedes**(*filename*)

Parses a scedes file into a dictionary.

Args:
filename (str): The path to the scedes file as defined in module_unf.py

Returns:
dict: A dictionary where keys are from the first column and values are from the second column in scedes CSV.

2.12 Glossary

Abbreviations and terms used throughout HSM documentation, listed alphabetically by abbreviation or short form. Linked wherever a :term: cross-reference appears in the prose.

AEO
Annual Energy Outlook
The *Annual Energy Outlook*, published annually by EIA. HSM serves as the upstream oil and gas supply module for every AEO projection. AEO projections run from the current history year through a multi-decade horizon; the end year varies by publication cycle (2050 for AEO2026).

AD gas
Associated-dissolved gas
Natural gas co-produced with crude oil from oil-bearing formations. Reported separately from non-associated gas because production volume depends on oil drilling decisions, not gas prices. See *NA gas*.

CBM
Coalbed methane
Methane absorbed in coal seams, recovered as a natural gas resource. Modeled within the onshore submodule.

CAPM
Capital Asset Pricing Model
A standard finance model used to estimate the cost of equity (r_e) as a function of the risk-free rate, beta, and the market risk premium. HSM uses CAPM inside the *WACC* calculation; see *Price and Discount Rate Calculations*.

CCATS
Carbon Capture Allocation Transportation and Sequestration
The NEMS module for carbon capture, allocation, transportation, and sequestration. HSM sends CO_2 demand from *EOR* projects and CO_2 supply cost curves from the *NGP* submodule to CCATS each model year.

CER
Canada Energy Regulator
The federal Canadian regulator whose *Canada's Energy Future* scenarios provide the baseline production projections used to calibrate the HSM Canada submodule.

DCF
Discounted cash flow
The economic evaluation method used to value drilling projects. HSM discounts projected revenues and costs at the *WACC* to compute *NPV* and *PI* for each candidate project. See *Economic Framework — Discounted Cash Flow Model*.

EIA
U.S. Energy Information Administration
The statistical and analytical agency of the U.S. Department of Energy. HSM is developed and maintained by EIA as a component of *NEMS*.

EOR
Enhanced oil recovery
Techniques — principally CO_2 flooding and steam injection — used to recover oil from mature fields beyond primary and secondary recovery. HSM models CO_2 EOR projects for mature conventional reservoirs only within the onshore submodule and exchanges CO_2 demand data with *CCATS*; EOR in unconventional (tight and shale) formations is not represented.

fcrl
Final Convergence Reporting Loop
A NEMS iteration flag (value 1 on the last regular iteration before reporting). When `fcrl == 1`, HSM saves intermediate state to `fcrl_var.pkl` for use on the next iteration. HSM updates expected natural gas production in the restart file on every iteration it runs (not only fcrl). See *fcrl — Final Convergence Reporting Loop*.

HSM
Hydrocarbon Supply Model
The Python-based upstream oil and gas supply module within *NEMS*, developed at EIA. HSM projects domestic crude oil and natural gas production for the United States and natural gas available for import from Canada, using *DCF* economics. Replaced *OGSM* beginning with AEO2025.

IPR
Initial production rate
The production rate of a new well at the start of its decline curve. Used in the Canada submodule's hyperbolic decline equations to compute the production profile of new Canadian gas wells.

L48
Lower 48
Lower 48 states
The contiguous 48 U.S. states, excluding Alaska and Hawaii. The onshore and offshore submodules cover L48 geography; Alaska is modeled separately.

LFMM
Liquid Fuels Market Module
The NEMS module that models domestic liquid fuels supply, demand, refining, and pricing. LFMM sends crude

oil wellhead prices to HSM and receives crude oil production volumes and *NGPL* production in return.

MAM
Macroeconomic Activity Module
: The NEMS module that models macroeconomic conditions. MAM provides the GDP price deflator and corporate bond and treasury yields to HSM each model year; these inputs drive price deflation and the *WACC* calculation.

MRP
Market risk premium
: The expected excess return of equities over the risk-free rate; denoted MRP in the *CAPM*-based cost-of-equity formula. Set in `discounting.csv`.

NA gas
Non-associated gas
: Natural gas produced from gas-only wells, not co-produced with crude oil. The primary gas output of the onshore and Canada submodules. Reported separately from *AD gas* because production responds directly to gas prices.

ncrl
NEMS Convergence Reporting Loop
: A NEMS iteration flag (value 1 on the reporting iteration). When `ncrl == 1`, HSM applies natural gas production adjustments based on realized demand volumes from *NGMM* and runs the *NGP* submodule. See *ncrl — NEMS Convergence Reporting Loop*.

NEMS
National Energy Modeling System
: EIA's integrated energy model of the U.S. energy economy. NEMS couples supply, demand, conversion, and macroeconomic modules that iterate to a supply-demand-price equilibrium each model year. HSM is NEMS's upstream oil and gas supply module.

NGP
Natural Gas Processing
: The HSM submodule that models CO_2 capture from U.S. natural gas processing plants. NGP runs on the *ncrl* iteration and supplies CO_2 cost curves to *CCATS*.

NGMM
Natural Gas Market Module
: The NEMS module that models North American natural gas transmission, distribution, and pricing. NGMM sends natural gas supply prices to HSM and receives expected (*fcrl*) and realized (*ncrl*) non-associated gas production volumes.

NGPL
Natural gas plant liquids
: Hydrocarbon liquids — ethane, propane, butane, pentane, and natural gasoline — separated from natural gas at processing plants. Produced by the onshore, offshore, and Alaska submodules and reported to *LFMM*.

NPV
Net present value
: The discounted sum of a project's cash flows over its life, computed at the *WACC* discount rate. Projects with NPV > 0 are economically viable. See *Economic Framework — Discounted Cash Flow Model*.

OCS
Outer Continental Shelf
: The submerged offshore lands and waters beyond state jurisdiction (generally beyond three nautical miles), administered by the Bureau of Ocean Energy Management. The HSM offshore submodule covers federal OCS areas in the Gulf of America, Atlantic, and Pacific.

OGSM
Oil and Gas Supply Module
: The Fortran-based upstream supply module that HSM replaced beginning with AEO2025. Readers comparing Outlooks across years should see *Historical Context* for what changed.

PI
Profitability index
: The ratio of a project's *NPV* to its discounted drilling cost (C_K). HSM ranks candidate projects by PI descending and drills the highest-PI projects subject to rig, footage, and capital constraints. See *Project Selection and Constraints*.

RESID
Reservoir ID
: The project key that identifies an onshore HSM project. Individual wells are rolled up into a project sharing a RESID, a concatenated code built from the project's oil or gas type, state, EIA play code, county FIPS code, and process code. See *Project identifier (RESID) construction*.

SCEDES
Scenario description
: NEMS's text-based scenario configuration format. For standalone HSM runs, `pyscedes.py` parses a SCEDES file to supply scenario parameters that NEMS would otherwise provide at runtime. See *SCEDES File (Standalone Runs)*.

WACC
Weighted average cost of capital
: The blended after-tax discount rate used in HSM's *DCF* analysis, combining the cost of equity (from *CAPM*) and the after-tax cost of debt. Computed annually from *MAM*-supplied macro inputs and parameters in `discounting.csv`. See *Price and Discount Rate Calculations*.

PYTHON MODULE INDEX

INDEX

Symbols

A

D

E

F

G

H

I

L

M

N

O

P

Q

R

S

T

U

V

W

Z