SEALS User Guide

Spatial Economic Allocation Landscape Simulator

Justin Andrew Johnson

2026-07-31

Table of contents

1 What SEALS is

SEALS, the Spatial Economic Allocation Landscape Simulator, is a land-use change model that downscales predictions of land-use change from aggregate (regional or coarse-gridded) inputs to a finer resolution, typically 10–300 meters.

Its primary comparative advantage is fast computation. It downscales to 300 meters globally — roughly 8.4 billion grid cells — in about an hour on a laptop. It is designed with parallelization in mind and so scales well to high-performance, cloud, or distributed computing. Performance-critical functions are written in C++ (via Cython); everything else is Python.

SEALS is one of three software products maintained by NatCap TEEMs (The Earth-Economy Modellers):

product	what it is
SEALS	this model — the downscaler
GTAP-InVEST	the coupled economy–ecosystem model that uses SEALS for its land-use step
Earth-Economy Devstack	the shared platform: hazelbean, ProjectFlow, and the conventions all three follow

1.1 How to read this guide

Start at Environment Setup and Quickstart — together they take you from nothing to a downscaled map. Then:

- **Walkthrough** — a guided first run, and the anatomy of a run file. Read this before you start a project of your own.
- **scenarios.csv format** — the column reference for the file that defines what SEALS runs.
- **MAgPIE** and the **MAgPIE tutorial** — coupling SEALS to an external land-use model, which is also the general pattern for coupling it to any coarse projection source.

This guide is rendered both as a website and as a single PDF (download). If you are reading the PDF, links to the other two products point at the published site.

1.2 Where SEALS ends and the platform begins

SEALS is a model. The machinery that runs it — project directories, task trees, skip-if-already-computed re-runs, automatic base-data download — is **ProjectFlow**, which lives in hazelbean and is shared across all three products. This guide covers SEALS and says only as much about ProjectFlow as you need to run it.

For the platform itself, see the Earth-Economy Devstack documentation:

- Conventions — the run-file spec, naming rules, and directory layout
- Project complexity — the three axes that describe how a run file is organized, and why
- ProjectFlow — the task-tree engine
- Installation — devstack-wide install options, including the developer install

1.3 Status and citation

SEALS is under active development and changes rapidly. Pull the latest `seals_dev` before reporting a problem.

The model was introduced alongside GTAP-InVEST in Johnson et al. 2023, PNAS. All code is available under a permissive open-source license.

2 SEALS Environment Setup

Two paths: a **user install** if you just want to run SEALS, or a **developer install** from a GitHub clone if you want to edit SEALS (or hazelbean) and have your edits take effect immediately.

SEALS is distributed as `sealsmodel` and imported as `seals`. It requires hazelbean 2.0.0 or later.

2.1 1. Install Miniforge/Conda

If not already installed, download and install Miniforge.

2.2 2. Create and activate an environment

```
conda create -n <your_env> python=3.10
conda activate <your_env>
```

Pick your own environment name — the devstack does not require a particular one. Always activate it before running SEALS scripts.

2.3 3a. User install

```
pip install hazelbean sealsmodel
```

Dependencies are installed automatically. This is enough to run `run_seals_standard.py` and to start your own project from it.

2.4 3b. Developer install from a GitHub clone

Use this if you cloned the repos and want your edits to take effect without reinstalling. SEALS compiles a Cython extension (`seals/seals_cython_functions.pyx`), so **a C/C++ compiler is required** — see Compiling C/C++ Code (`xcode-select --install` on macOS; Visual Studio Build Tools on Windows).

Clone into the devstack layout. Each repo lives in its own project folder under `~/Files/` (`C:\Users\<you>\Files\` on Windows). The layout matters: ProjectFlow infers where to put project directories from it, placing them at `~/Files/seals/projects/<project_name>/` — outside the repo, so outputs never land in your working tree.

```
mkdir -p ~/Files/hazelbean && cd ~/Files/hazelbean
git clone https://github.com/jandrewjohnson/hazelbean_dev.git

mkdir -p ~/Files/seals && cd ~/Files/seals
git clone https://github.com/jandrewjohnson/seals_dev.git
```

If you already installed hazelbean from conda-forge, remove it first so the editable install is the one that gets imported. Use `conda remove`, not `pip uninstall` — pip leaves behind directories that still shadow the source:

```
conda remove hazelbean --force
```

This drops the conda-forge build but keeps its dependencies.

Make the editable installs, hazelbean first — SEALS imports it:

```
cd ~/Files/hazelbean/hazelbean_dev
pip install -e . --no-deps

cd ~/Files/seals/seals_dev
pip install -e . --no-deps
```

`--no-deps` is deliberate: the dependencies came from conda in step 2, and letting pip resolve them again tends to replace conda's builds of the geospatial stack (GDAL and friends) with incompatible wheels.

Verify:

```
python -c "import hazelbean, seals; print(hazelbean.__file__); print(seals.__file__)"
```

Both paths should point inside `~/Files/...`, not into `site-packages`. If either points at `site-packages`, an installed copy is shadowing your clone — remove it and redo the editable install.

2.5 4. Add other devstack repos (optional)

The same clone-and-editable-install pattern works for the rest of the stack, each in its own `~/Files/<name>/` project folder: gtap_invest_dev, global_invest_dev, gtappy_dev, gep_dev.

See the full installation guide for the devstack-wide version of these steps.

3 Quickstart

Get SEALS running, then make it your own. For the model itself see SEALS; for the full install options see Installation Steps.

3.1 1. Install

SEALS is distributed as `sealsmodel` and imported as `seals`. It needs hazelbean, which supplies `ProjectFlow`.

```
conda create -n <your_env> python=3.10
conda activate <your_env>
pip install hazelbean sealsmodel
```

Pick your own environment name. If you intend to edit SEALS rather than just run it, do a developer install from a clone instead — see environment_setup.md in the seals repo, which also covers the C/C++ compiler SEALS needs to build its Cython extension.

3.2 2. Run it

Two run files ship with SEALS, in `seals_dev/seals/`:

file	what it runs
`run_seals_standard.py`	the standard pipeline: three scenarios, 2030 and 2050
`run_seals_standard_test.py`	the same pipeline pared down — baseline plus one BAU, one year, Rwanda

Start with the test:

```
conda activate <your_env>
cd seals_dev/seals
python run_seals_standard_test.py
```

Required base data is downloaded at runtime from a public bucket; no credentials are needed for the default configuration.

3.3 3. Where the output goes

Not into the repo. Because the run file lives inside a cloned repo, ProjectFlow places the project directory just *outside* it:

```
~/Files/seals/projects/seals_standard_test/
   input/          seeded from the repo's input_template/
   intermediate/   one folder per task
   output/
```

Run the same command a second time and it finishes almost immediately — every task is behind an existence check, so completed work is skipped. That is the main thing ProjectFlow buys you: interrupt a long run, fix one input, re-run, and only what is missing recomputes.

3.4 4. Make it your own

Copy the run file into your own project repo as `run_<project>.py` and change the two lines in its `__main__` block — that is the whole of the configuration:

```
if __name__ == '__main__':
    p = hb.ProjectFlow(project_name='my_project', run_mode='check')
    p.scenario_definitions_filename = 'my_project_scenarios.csv'

    run_project(p)
```

`run_mode` controls how much prior work is reused: `'check'` (the default) resumes in a stable project dir; `'full'` timestamps a fresh one.

Put your scenarios CSV in an `input_template/` directory beside the run file. It is tracked in git, and ProjectFlow copies anything missing into the project's untracked `input/` on first run without ever overwriting your working copy. The columns are documented in scenarios_format.md.

To run the same pipeline a second way — a smoke test, a different AOI — write a *second run file* that imports the first rather than copying it. `run_seals_standard_test.py` is the worked example, and the pattern is explained in conventions.

3.5 Next

- Walkthrough — a guided tour with screenshots
- Conventions — the full run-file rules
- Project complexity — why run files are shaped this way

4 First SEALS run walkthrough

4.1 Getting set up

- Follow the installation page first, or the shorter quickstart if you just want to run the model
- Clone SEALS and Hazelbean into the devstack layout — each repo in its own project folder under `~/Files/`:
  - `~/Files/hazelbean/hazelbean_dev`
  - `~/Files/seals/seals_dev`
- The layout is not cosmetic: ProjectFlow reads it to decide where project directories go
- You know the install worked if this prints two paths inside `~/Files/`, not inside `site-packages`:

```
python -c "import hazelbean, seals; print(hazelbean.__file__); print(seals.__file__)"
```

4.2 Explore the SEALS code

- The repo root `seals_dev/` holds more than the library — docs, tests, scripts
- The library itself is the `seals_dev/seals` subdirectory. The nesting looks redundant but is how Python packaging works
- Inside it, `__init__.py` is what makes the directory importable as a package
- The modules are split by what they hold:
  - `seals_main.py`, `seals_tasks.py`, `seals_generate_base_data.py` — task functions, the actual model logic
  - `seals_initialize_project.py` — the task-tree builders
  - `seals_utils.py` — helpers
  - `run_seals_standard.py` — the run file

4.3 Run files

- You do not run `seals_main.py` directly
- A **run file** configures a project and executes a task tree built from the library's task functions
- Two ship with SEALS, both in `seals_dev/seals`:
  - `run_seals_standard.py` — three scenarios, 2030 and 2050
  - `run_seals_standard_test.py` — the same pipeline pared to a baseline, one BAU scenario, one year, Rwanda. Start here
- Open `run_seals_standard.py`. It is about 60 lines and has three parts

4.4 Part 1: the task tree

- `build_task_tree(p)` answers one question: *what is this pipeline?*
- It contains nothing but tree construction. For stock SEALS it is one line, delegating to the shared library builder
- Compose extra library subtrees or your own tasks here if your project diverges

```
def build_task_tree(p):
    seals_initialize_project.build_standard_task_tree(p)
```

4.5 Part 2: run_project

- `run_project(p)` answers *how is it executed?* It takes only `p`
- The rule that decides what goes where:
  - **`run_project(p)` sets what no variant ever changes** — base data location, processing resolution
  - **the caller sets what a variant might** — project name, run mode, scenarios CSV
- When a constant starts varying, it moves one line up into the caller. There is no signature to edit and no default to keep in sync

```
def run_project(p):
    p.run_in_parallel = 1        # before the tree: it has parallel iterators
    build_task_tree(p)
    p.skip_tasks(p.tasks_to_skip)

    p.base_data_dir = os.path.join(p.user_dir, 'Files', 'base_data')
```

```
    p.processing_resolution = 1.0

    p.scenario_definitions_path = os.path.join(p.input_dir, p.scenario_definitions_filename)
    seals_initialize_project.initialize_scenario_definitions(p)
    seals_initialize_project.set_advanced_options(p)

    p.execute()
    return p
```

4.6 Part 3: the ___main___ guard

- This is where the project is configured — two lines of actual choice
- The guard is mandatory: importing a run file must never start a run. That is what lets a second run file import this pipeline instead of copying it

```
if __name__ == '__main__':
    p = hb.ProjectFlow(project_name='seals_standard', run_mode='check')
    p.scenario_definitions_filename = 'standard_scenarios.csv'

    run_project(p)
```

4.7 The p object

- `hb.ProjectFlow()` is a class — a recipe for an object. Calling it produces an object, which we assign to `p`
- `p` carries the project's **attributes**: `p.base_data_dir`, `p.scenario_definitions_filename`, and so on
- It also has **methods** that act on it: `p.add_task()`, `p.skip_tasks()`, `p.get_path()`, `p.execute()`
- Every task receives `p` and reads its configuration from it. That is how configuration set in `__main__` reaches code deep in the library

4.8 Where the project directory goes

- Not into the repo. Because the run file sits inside a cloned repo, ProjectFlow places the project dir just *outside* it, at `~/Files/seals/projects/<project_name>/`
- You do not compute this path yourself, and you should not save data in `seals_dev`
- `run_mode` decides how much prior work is reused:

– `'check'` — reuse the stable dir, recompute only what is missing (the default)
– `'fresh_intermediate'` — redo all computation, keep `input/` (test projects only)
– `'full'` — a fresh timestamped dir; everything re-runs

5 Scenario definitions

5.1 The scenarios CSV

- The scenarios CSV specifies the runs you want. **Each row is one scenario**
- As the model iterates, it re-hydrates `p` from the row it is on — which is why scenario-varying values belong in the CSV and not in the run file
- The tracked copy lives in `input_template/` beside the run file. ProjectFlow copies anything missing into the project's `input/` on first run and never overwrites your working copy
- If the named file is nowhere to be found, SEALS generates a default so a first run still works

```
p.scenario_definitions_filename = 'standard_scenarios.csv'
```

5.2 Scenario types

- `scenario_type` is a **column in the CSV**, not something a run file sets
- `baseline` — the observed anchor; not downscaled, supplies base-year LULC
- `policy` — a trajectory to downscale, compared against a counterfactual
- `bau` — business-as-usual, being deprecated in favour of `baseline`; it still appears in shipped CSVs
- Full column reference: scenarios_format.md

5.3 Automatically downloading data

- Paths in the CSV are **ref_paths** — relative locations resolved by `p.get_path()`, which searches the project `input/` dir, then `base_data`, then a cloud bucket, downloading what is missing
- `p.base_data_dir` is where those downloads land. Point it somewhere with room for very large files; the same base data serves every project
- The directory must be named `base_data` to match the bucket convention
- The default bucket is public — no credentials needed

```
p.base_data_dir = os.path.join(p.user_dir, 'Files', 'base_data')
```

6 Running it

6.1 Run the model

```
conda activate <your_env>
cd seals_dev/seals
python run_seals_standard_test.py
```

- On startup SEALS prints the **task tree** it is about to compute
- To understand the model in depth, read the task functions behind those names
- Run the same command a second time: it finishes almost immediately, because every task is behind an existence check and completed work is skipped

6.2 Find the output

- Go to `~/Files/seals/projects/seals_standard_test/intermediate/`
- There is one directory per task in the tree
- The downscaled maps are in `stitched_lulc_simplified_scenarios/`, for example:

`lulc_esa_seals7_ssp2_rcp45_luh2-message_bau_2030.tif`

- Open it in QGIS alongside the base-year LULC and compare — that is your high-resolution land-use change projection

6.3 Next

- Quickstart — the condensed version of this page
- Conventions — the full run-file rules
- Project complexity — why run files are shaped this way

7 scenarios.csv Format (v1)

The scenarios CSV defines your SEALS scenarios. Each row is one scenario. The tracked copies that ship with SEALS are `seals/input_template/standard_scenarios.csv` (three scenarios) and `standard_scenarios_test.csv` (a pared two-row version); this page documents their columns.

> **This is the v1 schema — the one the code reads today.** A **v2 schema** is specified in scenario_definitions.qmd and no code implements it yet. v2 fixes several structural problems in v1: it replaces `baseline_reference_label` / `comparison_counterfactual_labels` / per-model time columns with a `observed_reference_label` / `compared_to` / `depends_on` split and a single canonical clock, and it allows hindcasting. Write v1 for now; expect a migration when the hazelbean parser lands.

7.1 Column reference

Values in the Example column are taken from the shipped `standard_scenarios.csv`.

Column	Description	Example
`scenario_label`	Unique scenario identifier	`baseline_luh2-message`, `ssp2_rcp45_luh2-message_bau`
`scenario_type`	`baseline`, `bau`, or `policy` — see the note below	`baseline`
`aoi`	Area of interest: `global`, an ISO3 code, or a path to a vector	`RWA`
`exogenous_label`	Exogenous driver set	`baseline`, `ssp2`
`climate_label`	Climate scenario	`rcp45`
`model_label`	Coarse-projection model source	`luh2-message`
`counterfactual_label`	Counterfactual this row represents (blank on the baseline row)	`bau`, `bau_shift`
`years`	Space-separated projection years	`2030 2050`

Column	Description	Example
`baseline_reference_label`	`scenario_label` of the baseline row this one is measured against (blank on the baseline row)	`baseline_luh2-message`
`base_years`	Observed/base year(s)	`2017`
`key_base_year`	The base year used for LULC	`2017`
`comparison_counterfactual_labels`	Scenario(s) this row is compared against in reporting	`ssp2_rcp45_luh2-message_bau`
`time_dim_adjustment`	Offset applied to the source time dimension	`add2015`
`coarse_projections_input_path`	Coarse projections NetCDF	`luh2/raw_data/rcp45_ssp2/multiple-states_..._2015-2100.nc`
`lulc_src_label`	Fine LULC source	`esa`
`lulc_simplification_label`	Fine LULC class scheme	`seals7`
`lulc_correspondence_path`	Fine class correspondence	`seals/default_inputs/esa_seals7_corre`
`coarse_src_label`	Coarse data source	`luh2-14`
`coarse_simplification_label`	Coarse class scheme	`seals7`
`coarse_correspondence_path`	Coarse class correspondence	`seals/default_inputs/luh2-14_seals7_correspondence.csv`
`lc_class_varname`	Land-class variable in the NetCDF	`all_variables`
`dimensions`	Dimensions to read	`time`
`calibration_parameters_source`	Downscaling coefficients	`seals/default_inputs/default_global_c`
`base_year_lulc_path`	Base-year LULC raster	`lulc/esa/lulc_esa_2017.tif`
`regional_projections_input_path`	Regional projections CSV (optional; blank if coarse-only)	`seals/default_inputs/rwa_bdi_regional`
`regions_vector_path`	Regions vector	`cartographic/ee/ee_r264_correspondenc`
`regions_column_label`	Region column in that vector	`ee_r264_label`

Paths are **ref__paths**: relative locations resolved by `p.get_path()`, which searches the project `input/` dir, then `base_data`, then the cloud bucket, downloading if needed. Always write them with **forward slashes**, on every platform.

7.1.1 A note on `scenario_type`

Three values appear in current files:

- `baseline` — the observed anchor. Not downscaled; it supplies base-year LULC and calibration inputs.

- `bau` — a business-as-usual trajectory. **Deprecated in favour of `baseline`.** The migration is in progress and unevenly applied, so `bau` still appears in shipped CSVs and is still read by the code. Do not rely on it for new work, but do not be surprised to see it.
- `policy` — a policy trajectory, downscaled and typically compared against a BAU row via `comparison_counterfactual_labels`.

7.2 Example

The shipped `standard_scenarios.csv`: a Rwanda baseline, a BAU trajectory, and a policy row that shifts the BAU using regional projections.

```
scenario_label,scenario_type,aoi,exogenous_label,climate_label,model_label,counterfactual_lab
baseline_luh2-message,baseline,RWA,baseline,rcp45,luh2-message,,2017,,2017,2017,,add2015,luh
ssp2_rcp45_luh2-message_bau,bau,RWA,ssp2,rcp45,luh2-message,bau,2030 2050,baseline_luh2-messa
ssp2_rcp45_luh2-message_bau_shift,policy,RWA,ssp2,rcp45,luh2-message,bau_shift,2030 2050,base
```

> **Known wart:** the shipped CSVs currently write `base_year_lulc_path` as `lulc\esa\lulc_esa_2017.tif`, with backslashes. The devstack rule is forward slashes everywhere, on every platform; these should be normalized.

7.3 Where the file lives

Put your scenarios CSV in an **`input_template/`** directory beside your run file. It is tracked in git; ProjectFlow copies anything missing into the project's untracked `input/` on first run and never overwrites the working copy. Name it in your run file's `__main__` block:

```
p.scenario_definitions_filename = 'my_project_scenarios.csv'
```

If the named file is absent from both `input/` and `base_data`, SEALS generates a default one in the project's `input/` so a first run still works — edit that copy, or better, add your own to `input_template/`.

8 Release Notes

Release history for SEALS. Versions continue this repository's own tags (`v1.2.0` preceded `v2.0.0`).

For the shared platform beneath SEALS — hazelbean, ProjectFlow and the conventions — see the Earth-Economy Devstack release notes.

8.1 SEALS v2.0.0 (2026-07-30)

Run-file spec: three complexity axes, and `run_project(p)`.

- A run file's complexity is now described by three *independent* axes rather than one ladder. **Configuration** (how a run varies) keeps the numbered levels 1–4: one task with inline constants; a task tree; + a scenarios CSV; + a parameters CSV. **Code layout** (where the code lives) runs single file → split (`<project>_tasks.py`, `_functions.py`, `_utils.py`, `_initialize_project.py`) → library package. **Ownership** (who owns the code you run) runs self-contained → devstack developer → downstream user. Moving along one axis never requires moving along another, so *"level" unqualified always means the configuration axis.*
- `run_project` now takes only `p`. The caller builds and configures the ProjectFlow in the `__main__` guard; `run_project(p)` only runs it. The rule that replaces a signature full of keyword defaults: **`run_project(p)` sets what no variant ever changes; the caller sets what a variant might.** A constant that starts varying moves one line up instead of becoming a new keyword argument, so the entry point stops growing as a project accumulates definition CSVs. Cost: bare `run_project()` no longer works, and a variant wrapper is four lines rather than three.
- `<project>_utils.py` is defined as a **promotion queue** into hazelbean — science-unaware helpers used by more than one project belong upstream — while `<project>_functions.py` holds the model's domain knowledge and is terminal.
- `examples/` reorganized: `run_templates/` (copy-me) and `run_templates_annotated/` (the same code with the reasoning written out), plus `input_research_scripts/` for real run files and the raw scripts they replace. Two new templates hold configuration fixed at level 4 and move one other axis each: `run_template_split_layout.py` and `run_template_downstream_user.py`.

- The six numbered sections of *project_complexity.qmd* are renamed to "Stage N, the old way" / "Stage N, redone with ProjectFlow" so they no longer collide with the configuration levels. They are a historical narrative, not the spec.
- All 46 `*_old_spec.py` archives are deleted across the stack. Git history is the archive for pre-conversion forms. The empty `docs/run_file.qmd` stub is removed; `examples/index.qmd` is the landing page for "what should my run file look like?".

Part I

Coupling SEALS to other models

9 Magpie

9.1 Installation

git clone -b develop https://github.com/magpiemodel/magpie.git magpie_develop

Has input and output folders in gitignore

MAgpie has a config folder

default.cfg

is something huge, so use a start script which takes lines from default and overwrites.

Then in the non-ignored starts folder, has a test_runs.R

10 To run gams

see start/start_functions.R

11 Applying SEALS to downscale MAgPIE outputs

11.1 Overview

The Model of Agricultural Production and Its Impact on the Environment (MAgPIE) is a global food and land system modelling framework. It uses a wide range of spatially-explicit biophysical and socioeconomic information to model global land-use dynamics throughout the 21st century. MAgPIE operates at two spatial scales that include a coarse scale of twelve model regions with similar socioeconomic characteristics and a subregional level that combines biophysical information, such as suitability for agricultural production (crop yield potential, irrigation water requirements, travel time to urban markets etc.), to determine the cost-effectiveness of different land-use activities. Both these resolutions, however, are typically too coarse to assess how global land-use dynamics could drive landscape-scale changes and associated changes in biodiversity and ecosystems service supply, which is a key sustainability challenge.

The Spatial Economic Allocation Landscape Simulator (SEALS) model can therefore be applied to spatially allocate projected land-use dynamics to spatial scale that is relevant for assessing landscape and biodiversity change. SEALS uses empirically calibrated adjacency relationships, physical suitability and conversion eligibility information at a spatial resolution of 300 x 300 Meter to allocate land use activities across space.

In the following this tutorial describes how both the MAgPIE and SEALS models can be linked using some of the functionalities provided by MAgPIE and SEALS to processes model outputs (MAgPIE) or inputs (SEALS) respectively. This tutorial therefore assumes some familiarity with the MAgPIE model - in particular how to set up model runs. Basic tutorials on how to run the MAgPIE model can be found here https://magpiemodel.github.io/tutorials/. A full documentation of the MAgPIE model (release version 4.7.2) can be accessed under https://rse.pik-potsdam.de/doc/magpie/4.7.2/ (substitute the version number 4.7.2 with any newer release version number to get the latest documentation).

11.2 Create MAgPIE outputs for SEALS by using output scripts

The MAgPIE model provides several output scripts that can be executed after a model run and are stored under `scripts/output` in the MAgPIE folder. Two scripts are particularly

relevant for providing MAgPIE outputs to SEALS: the `extra/disaggregation.R` script and the `extra/reportMAgPIE2SEALS.R` script. The `extra/disaggregation.R` script disaggregates coarse-resolution land-use projections to a spatial resolution of 0.5 degrees and is run by default after each model run. The `extra/reportMAgPIE2SEALS.R` script, on the other hand, generates a NetCDF file that can readily be used by SEALS but is *not* run by default.

The output scripts can be selected and executed via the command line. Therefore, on the command prompt, navigate to the MAgPIE model folder and use the following command:

```
Rscript output.R
```

On the command line you are now asked for which of the model runs you would like to do the postprocessing. You can choose `all`, select a specific run, or run the postprocessing for a selection of runs by choosing `search by the pattern.`

```
CLUSTER2015 - vjeetze@login01:/p/projects/magpie/users/vjeetze/magpie/other/f_magpie-seals_tutorial> Rscript output.R
Global .Rprofile loaded! (R version 4.1.2 (2021-11-01))
Hi Pat,

Your .Rprofile was loaded!

Checking for updates... Update check done.
Checking package version requirements... Requirements check done.

Choose runs:
1: all
2: weeklyTests_SSP2EU-NDC/
3: weeklyTests_SSP2EU-PkBudg650/
4: weeklyTests_SSP2EU-Ref/
5: Search by the pattern.
Number:
```

After this you can select the output script that you want to execute. Choose `extra`.

```
Main selection of MAgPIE output scripts
 ------------------------------------------------------------------------
 -> Scripts in this selection are actively managed and work out of  <-
 -> the box. Please provide a yml header to your script specifying  <-
 -> "description" and "comparison script", otherwise it may not be  <-
 ->                      processed correctly.                       <-
 ------------------------------------------------------------------------
 1:          output check | check output for known problems
 2:            rds report | extract report in rds and mif format from
                          | run
 3:          merge report | Merges report.mif files from several runs
                          | into a single mif file
 4:        rds report iso | extract report in rds format from run
 5:            validation | creates a validation pdf file for a run
                          | (long version - single crop types)
 6:      validation short | creates a validation pdf file for a run
                          | (short version - aggregated crop types)
 7: comparison validation | Creates an validation pdf out of several
                          | runs
 8:        runBlackmagicc | Append MAGICC7 warming pathways to a
                          | MAgPIE run's report.mif
 9:       validation cell | Comparison at cellular level of land and
                          | crop types with LUH2 and MAPSPAM

Alternatively, choose a output script from another selection:
10:                 extra | Additional MAgPIE output scripts
11:              projects | Project-specific MAgPIE output scripts
12:            deprecated | Deprecated scripts
Choose a output script or folder: 10
```

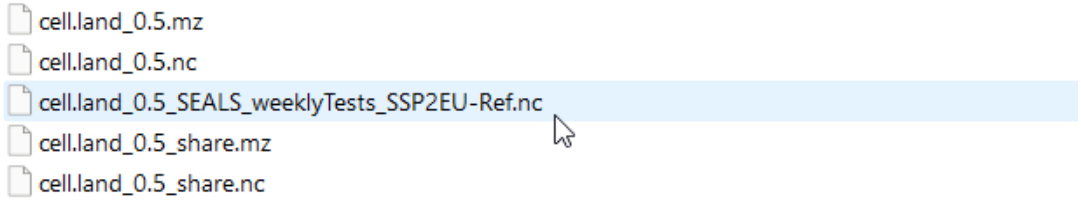
Next select `reportMAgPIE2SEALS`.

```
Additional MAgPIE output scripts
 ------------------------------------------------------------------------
 ->      Scripts in this selection might require some manual        <-
 -> adjustments before they work with the given MAgPIE version as   <-
 ->   they are not necessarily updated along with changes in the    <-
 ->     model code. Please provide a yml header to your script      <-
 -> specifying "description" and "comparison script", otherwise it  <-
 ->                  may not be processed correctly.                <-
 ------------------------------------------------------------------------
 1:                    aff area | extracts afforstation area from
                                | multiple runs and creates a PDF
                                | (useful template for other
                                | variables)
 2:         disaggregation LUH2 | Downscale MAgPIE results to 0.25
                                | degree resolution in LUH2 format for
                                | ISIMIP 3b
 3:              disaggregation | Interpolates land pools to 0.5
                                | degree resolution
 4:                    emulator |
 5:         force runstatistics | Forces runstatistics submission in
                                | central repo
 6:         ForestChangeCluster | Compares Forest Area Change at
                                | Cluster Level
 7:                     highres | starts a run with higher resolution
                                | in parallel mode (each region is
                                | solved individually) using trade
                                | patterns from an existing run
 8:                land cluster | Write land use information on
                                | cluster level to land_cluster.gpkg
                                | (GeoPackage)
 9:                    modelstat | checks the modelstat of several runs
10:         reportMAgPIE2REMIND | Write only those variables into a
                                | report that are relevant for the
                                | REMIND coupling
11:          reportMAgPIE2SEALS | Modifies gridded MAgPIE land use
                                | patterns so that they can be read in
                                | by the Spatial Economic Allocation
                                | Landscape Simulator (SEALS)
12:                     resubmit | re-submits runs to different queues
                                | (e.g. priority)
13:                      runtime | Checks the runtime of several runs
14:           timestep duration | Makes a png file containing solve
                                | time for each step
15: validation existing report | Creates a validation pdf out of an
                                | existing mif file, speeding up the
                                | process.
Choose a script: 11
```

Finally, choose the run submission type, e.g. `Direct execution`.

```
Choose submission type:
1: SLURM standby
2: SLURM standby maxMem
3: SLURM priority
4: SLURM priority maxMem
5: Direct execution
6: Background execution
7: Debug mode
Number: 5
```

All outputs are written to a subfolder in the `output` folder within the MAgPIE directory. This subfolder (or run folder) is created automatically at the beginning of a model run and its name is a combination of the run title and date (For further information see https://magpiemodel.github.io/tutorials/t06-changingconfig). Before you run `reportMAgPIE2SEALS`, please make sure that the script `extra/disaggregation.R` was executed without error. If the script `extra/disaggregation.R` was completed successfully, the run folder should contain the file `cell.land_0.5_share.mz`. Once the script `extra/reportMAgPIE2SEALS.R` has finished the run folder should contain a file named `cell.land_0.5_SEALS_<run title>.nc`. This file contains spatially-explicit information on the share of total land per grid cell used for different land-use activities (cropland, pasture, forest, urban etc.) projected until 2050, which can now be used as an input for SEALS.



11.2.1 Change your start script to generate MAgPIE outputs for SEALS

If you are familiar with setting up a start script for MAgPIE runs (see https://magpiemodel.github.io/tutorials/t07-startscript), an alternative way of generating MAgPIE outputs for SEALS is by adding `"extra/reportMAgPIE2SEALS"` to the setting

```
cfg$output <- c("output_check", "extra/disaggregation", "rds_report")
```

so that it reads

```
cfg$output <- c("output_check", "extra/disaggregation", "rds_report", "extra/reportMAgPIE2SEA
```

As the order of the vector determines the order of execution, please make sure that "extra/reportMAgPIE2SEALS" is called after "extra/disaggregation".

11.3 Setting up SEALS to downscale MAgPIE outputs

After generating the output cell.land_0.5_SEALS_<run title>.nc, the following describes how this information can be used as input for the SEALS model.

First, we need to relate the land use types of the MAgPIE model to the land cover types used by the SEALS model. While MAgPIE separates seven different main land types (cropland, grassland, primary forest, secondary forest, forestry, non-forest vegetation & urban), the SEALS model only allocates changes for 5 main land types (urban, cropland, grassland, forest & other natural vegetation). Below a mapping between the different land types is shown.

```
src_id,dst_id,src_label,dst_label,src_description,dst_description
6,1,urban,urban,urban land,
1,2,crop,cropland,cropland,
2,3,past,grassland,grassland,
3,4,primforest,forest,primary forest,
4,4,secdforest,forest,secondary forest,
5,4,forestry,forest,forestry,
7,5,other,othernat,non-forest vegetation,
```

Copy and paste this mapping into a CSV file and save it, e.g. under base_data/seals/default_inputs/magpie_s
The folder base_data/seals/default_inputs also contains other mappings such as a mapping betweehn LUH2v2 land types and SEALS. This mapping can now be used to set up the **scenarios CSV**. The scenarios CSV provides SEALS with all necessary information on the input data (or MAgPIE outputs), including the path to the MAgPIE outputs. It also allows the user to set up multiple SEALS runs (e.g. in the case of different scenario runs) in one go. The file is named per project — the one shipped with SEALS is standard_scenarios.csv — and your run file points at it with p.scenario_definitions_filename. One way to better understand it is to run SEALS on the test data without further modification, which generates a default in the project folder that you can then tailor to your needs. The columns are documented in scenarios_format.md.

The following shows how an example scenario_definitions_tutorial.csv is created that provides relevant information for a set of different MAgPIE test runs. Assume we have created land use projections for three different test scenarios: a reference 'business-as-usual' scenario (weeklyTests_SSP2EU-Ref), a low ambition mitigation scenario, in which current nationally determined contributions (NDCs) under the Paris agreement for the land sector are fulfilled (weeklyTests_SSP2EU-NDC) and a highly ambitious mitigation scenario in line with the 1.5 degree target (weeklyTests_SSP2EU-PkBudg650). For each of the scenarios, we have created

gridded land-use projections in a format that can be read by SEALS using the output script `extra/reportMAgPIE2SEALS.R` (see above).

Here, the `scenario_definitions_tutorial.csv` contains a line for each scenario specified under `scenario_label`. Note that the `Baseline` always needs to be defined as well. In the years column you can specify the years for which SEALS should do a downscaling. Multiple years can be separated by a space.

	column 1	column 2	column 3	column 4	column 5	column 6	column 7	column 8
1	scenario_label	scenario_type	aoi	exogenous_label	climate_label	model_label	counterfactual_label	years
2	Baseline	baseline	global	baseline		magpie		2020
3	weeklyTests_SSP2EU-Ref	bau	global	ssp2	rcp26	magpie	weeklyTests_SSP2EU-Ref	2030 2050
4	weeklyTests_SSP2EU-NDC	policy	global	ssp2	rcp26	magpie	weeklyTests_SSP2EU-NDC	2030 2050
5	weeklyTests_SSP2EU-PkBudg650	policy	global	ssp2	rcp26	magpie	weeklyTests_SSP2EU-PkBudg650	2030 2050

Set the path to the gridded MAgPIE outputs in the column `coarse_projections_input_path`

	column 1		column 14	column 15
1	scenario_label	ent	coarse_projections_input_path	lulc_src_label
2	Baseline		C:/Users/vjeetze/PIK/MAgPIE/other/magpie_seals_tutorial/cell.land_0.5_SEALS_weeklyTests_SSP2EU-Ref.nc	esa
3	weeklyTests_SSP2EU-Ref		C:/Users/vjeetze/PIK/MAgPIE/other/magpie_seals_tutorial/cell.land_0.5_SEALS_weeklyTests_SSP2EU-Ref.nc	esa
4	weeklyTests_SSP2EU-NDC		C:/Users/vjeetze/PIK/MAgPIE/other/magpie_seals_tutorial/cell.land_0.5_SEALS_weeklyTests_SSP2EU-NDC.nc	esa
5	weeklyTests_SSP2EU-PkBudg650		C:/Users/vjeetze/PIK/MAgPIE/other/magpie_seals_tutorial/cell.land_0.5_SEALS_weeklyTests_SSP2EU-PkBudg650.nc	esa

The `lulc_correspondence_path` gives the path to a mapping between ESA CCI land cover classes and the aggregated land cover types used in SEALS.

	column 1	column 17	column 18	column 19
1	scenario_label	lulc_correspondence_path	coarse_src_label	coarse_simplification_label
2	Baseline	seals/default_inputs/esa_seals7_correspondence_mosaic-natural.csv	magpie	seals7
3	weeklyTests_SSP2EU-Ref	seals/default_inputs/esa_seals7_correspondence_mosaic-natural.csv	magpie	seals7
4	weeklyTests_SSP2EU-NDC	seals/default_inputs/esa_seals7_correspondence_mosaic-natural.csv	magpie	seals7
5	weeklyTests_SSP2EU-PkBudg650	seals/default_inputs/esa_seals7_correspondence_mosaic-natural.csv	magpie	seals7

The default mapping in SEALS is stored under `base_data\seals\default_inputs\esa_seals7_correspondenc`
For MAgPIE-SEALS couplings, however, it is recommended to use a mapping that classifies "crop_natural_mosaic" as "othernat" and "natural_crop_mosaic" as "forest" to better align the total cropland extent of the ESA map with the cropland initialisation in MAgPIE. That there is an overestimation of the cropland extent in the ESA CCI data is a well-know issue. Thus, the lines in `esa_seals7_correspondence.csv`

```
6,,30,2,crop_natural_mosaic,cropland
7,,40,2,natural_crop_mosaic,cropland
```

are changed to

```
6,,30,2,crop_natural_mosaic,othernat
7,,40,2,natural_crop_mosaic,forest
```

in `esa_seals7_correspondence_mosaic-natural.csv`.

Next, we set the path to the mapping between MAgPIE land use types to SEALS land cover types, as defined above

	column 1	column 20	column 21	column 22
1	scenario_label	coarse_correspondence_path	lc_class_varname	dimensions
2	Baseline	seals/default_inputs/magpie_seals7_correspondence.csv	all_variables	time
3	weeklyTests_SSP2EU-Ref	seals/default_inputs/magpie_seals7_correspondence.csv	all_variables	time
4	weeklyTests_SSP2EU-NDC	seals/default_inputs/magpie_seals7_correspondence.csv	all_variables	time
5	weeklyTests_SSP2EU-PkBudg650	seals/default_inputs/magpie_seals7_correspondence.csv	all_variables	time

We also specify a set of regressors used during downscaling. Tables of trained model coefficients used by SEALS can be found under `base_data/seals/default_inputs`. Here we can also align some model constraints between MAgPIE and SEALS. If, for example, compliance with currently implemented land policies such as protected areas is assumed in MAgPIE, these constraints also need to be reflected during the downscaling. We here therefore add an additional multiplicative regressor that precludes expansion of agricultural land (cropland and grassland) in legally protected areas as reported by the WDPA database, which is in line with the assumption in MAgPIE.

	column 1	column 2	column 3	column 4	column 5	column 6	column 7	column 8
1	spatial_regressor_name	data_location	type	urban	cropland	grassland	forest	othernat
11	grassland_binary	lulc/esa/seals7/binaries/2014/binary_esa_seals7_2014_grassland.tif	additive	0.005555556	0.018888889	0	0.041666667	-0.026111111
12	forest_binary	lulc/esa/seals7/binaries/2014/binary_esa_seals7_2014_forest.tif	additive	-0.019444444	-0.016666667	-0.002666667	0	0.033444444
13	othernat_binary	lulc/esa/seals7/binaries/2014/binary_esa_seals7_2014_othernat.tif	additive	0.01	0.144444444	0.060111111	0.02	0
14	water_binary	lulc/esa/seals7/binaries/2014/binary_esa_seals7_2014_water.tif	additive	0	0	0	0	0
15	other_binary	lulc/esa/seals7/binaries/2014/binary_esa_seals7_2014_other.tif	additive	-1.119444444	0.001666667	0.126666667	0.061111111	-0.023333333
16	urban_gaussian_1	lulc/esa/seals7/convolutions/2014/convolution_esa_seals7_2014_url	additive	1.713888889	-1122.233444	-11.13055556	0.041666667	-1122.241667
17	cropland_gaussian_1	lulc/esa/seals7/convolutions/2014/convolution_esa_seals7_2014_cro	additive	0.105555556	0.333444444	0.022222222	0	-11.24444444
18	grassland_gaussian_1	lulc/esa/seals7/convolutions/2014/convolution_esa_seals7_2014_gra	additive	0.054444444	0.005444444	0.38	0.018222222	0.085555556
19	forest_gaussian_1	lulc/esa/seals7/convolutions/2014/convolution_esa_seals7_2014_for	additive	-0.122222222	1111.065556	-0.011	1.555	-0.022233333
20	othernat_gaussian_1	lulc/esa/seals7/convolutions/2014/convolution_esa_seals7_2014_otl	additive	0.010888889	0	0.019444444	-0.122222222	0.466666667
21	water_gaussian_1	lulc/esa/seals7/convolutions/2014/convolution_esa_seals7_2014_wa	additive	0.036555556	-112.2027778	-112.24	-1.105555556	-1133.322333
22	other_gaussian_1	lulc/esa/seals7/convolutions/2014/convolution_esa_seals7_2014_otl	additive	-0.127777778	-112.2555556	-1112.144444	-1111.133333	0.005555556
23	urban_gaussian_5	lulc/esa/seals7/convolutions/2014/convolution_esa_seals7_2014_url	additive	-0.072222222	-11.52222222	-111.2638889	-0.093333333	-0.087777778
24	cropland_gaussian_5	lulc/esa/seals7/convolutions/2014/convolution_esa_seals7_2014_cro	additive	0.068888889	0.162333333	-0.016666667	0.122222222	0.065555556
25	grassland_gaussian_5	lulc/esa/seals7/convolutions/2014/convolution_esa_seals7_2014_gra	additive	0.100222222	-0.025	0.431111111	-0.026677778	-0.041666667
26	forest_gaussian_5	lulc/esa/seals7/convolutions/2014/convolution_esa_seals7_2014_for	additive	0.133222222	0.367777778	0.076333333	0.68888	0.113333333
27	othernat_gaussian_5	lulc/esa/seals7/convolutions/2014/convolution_esa_seals7_2014_otl	additive	0	-0.073111111	0.024444444	-0.005555556	0.152777778
28	water_gaussian_5	lulc/esa/seals7/convolutions/2014/convolution_esa_seals7_2014_wa	additive	0.091666667	0.005	-1111.105444	-1.092777778	-0.002222222
29	other_gaussian_5	lulc/esa/seals7/convolutions/2014/convolution_esa_seals7_2014_otl	additive	0.045555556	0.15	-1111.077778	-0.008333333	-110.89
30	soil_organic_content_1m_30s	seals/static_regressors/soil_organic_content.tif	additive	0.027777778	-0.15	110.9777778	-111.14	-0.027777778
31	bio_12	seals/static_regressors/precip_mm.tif	additive	11.11944444	-0.994444444	1.14	-1.075	11.00444444
32	alt	seals/static_regressors/alt_m.tif	additive	-0.104444444	0.085	-0.024888889	-0.037788889	0.01
33	bio_1	seals/static_regressors/temperature_c.tif	additive	-0.022111111	0.044444444	-0.011111111	-0.01	-0.001111111
34	minutes_to_market_30s	seals/static_regressors/travel_time_to_market_mins.tif	additive	0.016122222	0.21	0.005555556	10.555	-0.034333333
35	pop_30s	seals/static_regressors/pop.tif	additive	0	0	0	0	0
36	CEC_1m_30s	seals/static_regressors/soil_cec.tif	additive	0	-0.016666667	0	111.0722222	0
37	clay_percent_1m_30s	seals/static_regressors/clay_percent.tif	additive	-0.051111111	0.02	-0.186111111	-0.046222222	1.084333333
38	ph_1m_30s	seals/static_regressors/ph.tif	additive	0	0.1	0	0	0
39	sand_percent_1m_30s	seals/static_regressors/sand_percent.tif	additive	0.034444444	0.018333333	-0.037777778	-0.048888889	-0.001111111
40	silt_percent_1m_30s	seals/static_regressors/silt_percent.tif	additive	-0.012777778	-0.165	0	-0.059011111	-0.146111111
41	wdpa	seals/static_regressors/magpie_WDPA.tif	multiplicative	0	0	0	1	1
42	wetlands_binary	seals/static_regressors/wetlands_binary.tif	additive	0	0	0	0	0
43	carbon_above_ground_mg_per_ha_global	seals/static_regressors/carbon_above_ground_mg_per_ha_global.tif	additive	0	0	0	0	0

Similarly, the same functionality can also be used to specify policy scenarios, such as the expansion of protected areas in line with the 30x30 goal of the Global Biodiversity Framework. The `seals/static_regressors/magpie_30by30.tif` contains both currently protected areas from the WDPA database and areas that might be protected under a global 30x30 protection scenario.

	column 1	column 2	column 3	column 4	column 5	column 6	column 7	column 8
1	spatial_regressor_name	data_location	type	urban	cropland	grassland	forest	othernat
11	grassland_binary	lulc/esa/seals7/binaries/2014/binary_esa_seals7_2014_grassland.tif	additive	0.005555556	0.018888889	0	0.041666667	-0.026111111
12	forest_binary	lulc/esa/seals7/binaries/2014/binary_esa_seals7_2014_forest.tif	additive	-0.019444444	-0.016666667	-0.002666667	0	0.033444444
13	othernat_binary	lulc/esa/seals7/binaries/2014/binary_esa_seals7_2014_othernat.tif	additive	0.01	0.144444444	0.060111111	0.02	0
14	water_binary	lulc/esa/seals7/binaries/2014/binary_esa_seals7_2014_water.tif	additive	0	0	0	0	0
15	other_binary	lulc/esa/seals7/binaries/2014/binary_esa_seals7_2014_other.tif	additive	-1.119444444	0.001666667	0.126666667	0.061111111	-0.023333333
16	urban_gaussian_1	lulc/esa/seals7/convolutions/2014/convolution_esa_seals7_2014_urb	additive	1.713888889	-1122.233444	-11.13055556	0.041666667	-1122.241667
17	cropland_gaussian_1	lulc/esa/seals7/convolutions/2014/convolution_esa_seals7_2014_cro	additive	0.105555556	0.333444444	0.022222222	0	-11.24444444
18	grassland_gaussian_1	lulc/esa/seals7/convolutions/2014/convolution_esa_seals7_2014_gra	additive	0.054444444	0.005444444	0.38	0.018222222	0.085555556
19	forest_gaussian_1	lulc/esa/seals7/convolutions/2014/convolution_esa_seals7_2014_for	additive	-0.122222222	1111.065556	-0.011	1.555	-0.022233333
20	othernat_gaussian_1	lulc/esa/seals7/convolutions/2014/convolution_esa_seals7_2014_oth	additive	0.010888889	0	0.019444444	-0.122222222	0.466666667
21	water_gaussian_1	lulc/esa/seals7/convolutions/2014/convolution_esa_seals7_2014_wa	additive	0.036555556	-112.2027778	-112.24	-1.105555556	-1133.322333
22	other_gaussian_1	lulc/esa/seals7/convolutions/2014/convolution_esa_seals7_2014_oth	additive	-0.127777778	-112.2555556	-1112.144444	-1111.133333	0.005555556
23	urban_gaussian_5	lulc/esa/seals7/convolutions/2014/convolution_esa_seals7_2014_urb	additive	-0.072222222	-11.52222222	-111.2638889	-0.093333333	-0.087777778
24	cropland_gaussian_5	lulc/esa/seals7/convolutions/2014/convolution_esa_seals7_2014_cro	additive	0.068888889	0.162333333	-0.016666667	0.122222222	0.065555556
25	grassland_gaussian_5	lulc/esa/seals7/convolutions/2014/convolution_esa_seals7_2014_gra	additive	0.100222222	-0.025	0.431111111	-0.026677778	-0.041666667
26	forest_gaussian_5	lulc/esa/seals7/convolutions/2014/convolution_esa_seals7_2014_for	additive	0.133222222	0.367777778	0.076333333	0.68888	0.113333333
27	othernat_gaussian_5	lulc/esa/seals7/convolutions/2014/convolution_esa_seals7_2014_oth	additive	0	-0.073111111	0.024444444	-0.005555556	0.152777778
28	water_gaussian_5	lulc/esa/seals7/convolutions/2014/convolution_esa_seals7_2014_wa	additive	0.091666667	0.005	-1111.105444	-1.092777778	-0.002222222
29	other_gaussian_5	lulc/esa/seals7/convolutions/2014/convolution_esa_seals7_2014_oth	additive	0.045555556	0.15	-1111.077778	-0.008333333	-110.89
30	soil_organic_content_1m_30s	seals/static_regressors/soil_organic_content.tif	additive	0.027777778	-0.15	110.9777778	-111.14	-0.027777778
31	bio_12	seals/static_regressors/precip_mm.tif	additive	11.11944444	-0.994444444	1.14	-1.075	11.00444444
32	alt	seals/static_regressors/alt_m.tif	additive	-0.104444444	0.085	-0.024888889	-0.037788889	0.01
33	bio_1	seals/static_regressors/temperature_c.tif	additive	-0.022111111	0.044444444	-0.011111111	-0.01	-0.001111111
34	minutes_to_market_30s	seals/static_regressors/travel_time_to_market_mins.tif	additive	0.016122222	0.21	0.005555556	10.555	-0.034333333
35	pop_30s	seals/static_regressors/pop.tif	additive	0	0	0	0	0
36	CEC_1m_30s	seals/static_regressors/soil_cec.tif	additive	0	-0.016666667	0	111.0722222	0
37	clay_percent_1m_30s	seals/static_regressors/clay_percent.tif	additive	-0.051111111	0.02	-0.186111111	-0.046222222	1.084333333
38	ph_1m_30s	seals/static_regressors/ph.tif	additive	0	0.1	0	0	0
39	sand_percent_1m_30s	seals/static_regressors/sand_percent.tif	additive	0.034444444	0.018333333	-0.037777778	-0.048888889	-0.001111111
40	silt_percent_1m_30s	seals/static_regressors/silt_percent.tif	additive	-0.012777778	-0.165	0	-0.059011111	-0.146111111
41	30by30	seals/static_regressors/magpie_30by30.tif	multiplicative	0	0	0	1	1
42	wetlands_binary	seals/static_regressors/wetlands_binary.tif	additive	0	0	0	0	0
43	carbon_above_ground_mg_per_ha_global	seals/static_regressors/carbon_above_ground_mg_per_ha_global.tif	additive	0	0	0	0	0

Finally, the path to the regressor used during the downscaling must also be added to the scenarios CSV, while the base land-cover data `base_year_lulc_path` must refer to the same year as defined in the `Baseline` case.

	column 1	column 20	column 21	column 22	column 23	column 24
1	scenario_label	espondence_path	lc_class_varname	dimensions	calibration_parameters_source	base_year_lulc_path
2	Baseline	lt_inputs/magpie_seals7_correspondence.csv	all_variables	time	seals/default_inputs/magpie_global_coefficients_wdpa_base.csv	lulc/esa/lulc_esa_2020.tif
3	weeklyTests_SSP2EU-Ref	lt_inputs/magpie_seals7_correspondence.csv	all_variables	time	seals/default_inputs/magpie_global_coefficients_wdpa_base.csv	lulc/esa/lulc_esa_2020.tif
4	weeklyTests_SSP2EU-NDC	lt_inputs/magpie_seals7_correspondence.csv	all_variables	time	seals/default_inputs/magpie_global_coefficients_wdpa_base.csv	lulc/esa/lulc_esa_2020.tif
5	weeklyTests_SSP2EU-PkBudg650	lt_inputs/magpie_seals7_correspondence.csv	all_variables	time	seals/default_inputs/magpie_global_coefficients_wdpa_base.csv	lulc/esa/lulc_esa_2020.tif

Save `scenario_definitions_tutorial.csv` in the `input_template/` directory beside your run file — it is tracked in git, and ProjectFlow copies it into the project's `input/` on first run. Then name it in the run file's `__main__` block with `p.scenario_definitions_filename = 'scenario_definitions_tutorial.csv'` and you are good to go.

```
        sys.path.insert(0, custom_seals_path)

    # SEALS will run based on the scenarios defined in a scenario_definitions.csv
    # If you have not run SEALS before, SEALS will generate it in your project's input_dir.
    # A useful way to get started is to to run SEALS on the test data without modification
    # and then edit the scenario_definitions.csv to your project needs.
    # Some of the other test files use different scenario definition csvs
    # to illustrate the technique. If you point to one of these
    # (or any one CSV that already exists), SEALS will not generate a new one.
    # The avalable example files in the default_inputs include:
    # - test_three_scenario_defininitions.csv
    # - test_scenario_defininitions_multi_coeffs.csvs

    p.scenario_definitions_path = os.path.join(p.input_dir, 'scenario_definitions_tutorial.csv')

    # Set defaults and generate the scenario definitions csv if it doesn't exist
```