message-ix-models

IIASA Energy, Climate, and Environment (ECE) Program

Jul 25, 2022

USER GUIDE

`message_ix_models` provides tools for research using the **MESSAGEix-GLOBIOM family of models** developed by the IIASA Energy, Climate, and Environment (ECE) Program and its collaborators. This 'family' includes single-country and other models derived from the main, global model; all built in the MESSAGEix framework and on the ix modeling platform (ixmp).

Among other tasks, the tools allow modelers to:

- retrieve input data from various upstream sources,
- process/transform upstream data into model input parameters,
- create, populate, modify, and parametrize scenarios,
- conduct model runs,
- set up model *variants* with additional details or features, and
- report quantities computed from model outputs.

CHAPTER

ONE

INSTALLATION

1.1 From PyPI

1. Run:

```
$ pip install message-ix-models
```

1.2 From source

1. Clone the code.
2. Run[1]:

```
$ pip install --no-use-pep517 --editable .
```

[1] See pypa/pip#7953.

CHAPTER

TWO

DATA, METADATA, AND CONFIGURATION

Many, varied kinds of data are used to prepare and modify MESSAGEix-GLOBIOM scenarios. Other data are produced by code as incidental or final output. These can be categorized in several ways. One is by the purpose they serve:

- **data**—actual numerical values—used or produced by code,
- **metadata**, information describing where data is, how to manipulate it, how it is structured, etc.;
- **configuration** that otherwise affects how code works.

Another is whether they are **input** or **output** data.

This page describes how to store and handle such files in both `message_ix_models` and `message_data`.

2.1 Choose locations for data

These are listed in order of preference.

2.1.1 (1) `message_ix_models/data/`

- Files in this directory are **public**, and are packaged, published, and installable from PyPI with `message_ix_models`; in standard Python terms, these are "package data".
- This is the preferred location for:
  - General-purpose metadata.
  - Basic configuration, e.g. for reporting, not specific to any model variant or project.
  - Data for publicized model variants and completed/published projects.
- Data here can be loaded with *`load_package_data()`* or other, more specialized code.
- Documentation files like `doc/pkg-data/*.rst` describe the contents of these files. For example: *Node code lists*.

2.1.2 (2) `data/` directory in the `message_data` repo

- Files in this directory are **private** and not installable from PyPI (because `message_data` is not installable).
- This is the preferred location for:
  - Data for model variants and projects under current development.
  - Specific data files that cannot be made public, e.g. due to licensing issues.
- Data here can be loaded with *`load_private_data()`* or other, more specialized code.

2.1.3 (3) Other, system-specific ("local") directories

- These are the preferred location for:
  1. Caches i.e. temporary data files used to speed up other code.
  2. Output e.g. data or figure files generated by reporting.
  3. Data files not distributed with `message_ix_models`, e.g. those with access conditions (registration, payment, etc.).
- These kinds of data **must not** be committed to `message_ix_models`. Caches and output **should not** be committed to `message_data`.
- Each user **may** configure a location for these data, appropriate to their system.

  This setting can be made in multiple ways. In order of ascending precedence:

  1. The default location is the *current working directory*, i.e. whichever directory the *Command-line interface* is invoked in, or in which Python code is run that imports and uses `message_ix_models`.
  2. The `ixmp` configuration file setting `message local data`.
  3. The `MESSAGE_LOCAL_DATA` environment variable.
  4. The `--local-data` CLI option and related options such as `--cache-path` or the `--output` option to the `report` command.

5. Code that directly modifies the `local_data` setting on *Context*.

   – This location **should** be outside the Git-controlled directories for `message_ix_models` or `message_data`. If not, use `.gitignore` files to hide these from Git.

- Use *Context.get_local_path()* and *local_data_path()* to construct paths under this directory.

2.2 General guidelines

Always consider: "Will this code work on another researcher's computer?"

Prefer text formats
…such as e.g. CSV and YAML. CSV files up to several thousand lines are compressed by Git automatically, and Git can handle diffs to these files easily.

***Do not* hard-code paths**
Data stored with (1) or (2) above can be retrieved with the utility funtions mentioned, instead of hard-coded paths.

For system-specific paths (3) only, get a *Context* object and use it to get an appropriate `Path` object pointing to a file

```
# Store a base path
project_path = context.get_local_path("myproject", "output")

# Use the Path object to generate a subpath
run_id = "foo"
output_file = project_path.joinpath("reporting", run_id, "all.xlsx")
```

Keep input and output data separate
Where possible, use (1) or (2) above for the former, and (3) for the latter.

Use a consistent scheme for data locations
For a submodule for a specific model variant or project named, e.g. `message_ix_models.model.[name]` or `message_ix_models.project.[name]`, keep input data in a well-organized directory under:

- `[base]/[name]/` —preferred, flatter,
- `[base]/model/[name]/`,
- `[base]/project/[name]/`,
- or similar,

where `[base]` is (1) or (2), above.

Keep *project-specific configuration files* in the same locations, or (less preferable) alongside Python code files:

```
# Located in `message_ix_models/data/`:
config = load_package_data("myproject", "config.yaml")

# Located in `data/` in the message_data repo:
config = load_private_data("myproject", "config.yaml")

# Located in the same directory as the code
config = yaml.safe_load(open(Path(__file__).with_name("config.yaml")))
```

Use a similar scheme for output data, except under (3).

Re-use configuration
: Configuration to run a set of scenarios or to prepare reported submissions **should** re-use or extend existing, general-purpose code. Do not duplicate code or configuration. Instead, adjust or selectively overwrite its behaviour via project-specific configuration read from a file.

2.3 Large/binary input data

These data, such as Microsoft Excel spreadsheets, **must not** be committed as ordinary Git objects. This is because the entire file is re-added to the Git history for even small modifications, making it very large (see issue #37).

Instead, use one of the following patterns, in order of preference. Whichever pattern is used, code for handling large input data must be in `message_ix_models`, even if the data itself is private, e.g. in `message_data` or another location.

2.3.1 Fetch from a remote source

Use a configuration file in `message_ix_models` to store metadata, i.e. the Internet location and other information needed to retrieve the data. Then, write code that retrieves the data and caches it locally:

```
import requests

# Load some configuration
config = yaml.safe_load(load_package_data("big-data-source", "config.yaml"))

# Local paths for the cached raw files and extracted file(s)
cache_path = context.get_cache_path("big-data-source")
downloaded = cache_path / "downloaded_file.zip"
extracted = cache_path / "extracted_file.csv"

with open(downloaded) as f:
    remote_data = requests.get(config["url"])
    # Handle the data, writing to `f`

# Extract the data from `downloaded` to `extracted`
```

This pattern is preferred because it can be replicated by anyone, and the reference data is public.

2.3.2 Use Git Large File Storage (LFS)

Git LFS is a Git extension that allows for storing large, binary files without bloating the commit history. Essentially, Git stores a one-line text file with a hash of the full file, and the full file is stored separately. The IIASA GitHub account has up to 300 GB of space for LFS objects.

To use this pattern, simply `git add ...` and `git commit` files in an appropriate location (above). New or unusual binary file extensions may require a `git lfs` command or modification to `.gitattributes` to ensure they are tracked by LFS and not by ordinary Git history. See the Git LFS documentation at the link above for more detail.

2.3.3 Retrieve data from existing databases

These include the same IIASA ENE ixmp databases that are used to store scenarios. Documentation **must** be provided that ensures this data is reproducible, i.e. any original source and code to create the database used by `message_data`.

2.3.4 Other patterns

Some other patterns exist, but should not be repeated in new code, and should be migrated to one of the above patterns.

- SQL queries against a Oracle/JDBC database. See International Energy Agency, below, and issue #53 for a description of how to replace/simplify this code.

2.4 Configuration

Context objects are used to carry configuration, environment information, and other data between parts of the code. Scripts and user code can also store values in a Context object.

```
# Get an existing instance of Context. There is always at
# least 1 instance available
c = Context.get_instance()

# Store a value using attribute syntax
c.foo = 42

# Store a value with spaces in the name using item syntax
c["PROJECT data source"] = "Source A"

# my_function() responds to 'foo' or 'PROJECT data source'
my_function(c)

# Store a sub-dictionary of values
c["PROJECT2"] = {"setting A": 123, "setting B": 456}

# Create a subcontext with all the settings of `c`
c2 = deepcopy(c)

# Modify one setting
c2.foo = 43

# Run code with this alternate setting
my_function(c2)
```

For the CLI, every command decorated with `@click.pass_obj` gets a first positional argument `context`, which is an instance of this class. The settings are populated based on the command-line parameters given to `mix-models` or (sub)commands.

2.4.1 Top-level settings

See model- and project-specific documentation for further context settings, e.g. *model.bare*.

Setting	Type	Description
cache_path	Path	Base path cache, e.g. as given by the `--cache-path` CLI option. Default *local_data*/cache/.
dry_run	bool	Whether an operation should be carried out, or only previewed.
local_data	Path	Base path for system-specific (3) data, e.g. as given by the `--local-data` CLI option.
platform_info	dict	Dictionary with keyword arguments for the `ixmp.Platform` constructor, from the `--platform` or `--url` CLI options.
scenario_info	dict	Dictionary with keys 'model' and 'scenario' as given by the `--model`/`--scenario` or `--url` CLI options.
url	dict	A scenario URL, e.g. as given by the `--url` CLI option.
units	pint.UnitRegistry	**Deprecated.** Use `from iam_units import registry`.

CHAPTER

THREE

COMMAND-LINE INTERFACE

This page describes how to use the **mix-models** command-line interface (CLI) to perform common tasks. **mix-models** is organized into **commands** and **subcommands**, sometimes in multiple levels. Our goal is that the *semantics* of all commands are similar, so that interacting with each command feels similar.

3.1 Controlling CLI behaviour

To support a variety of complex use-cases, the MESSAGEix stack takes configuration and inputs from several places:

3.1.1 `ixmp` configuration file: `config.json`

`ixmp` keeps track of named Platforms and their associated databases, and stores information in its `config.json` file. See `ixmp.config`. List existing platforms:

```
$ ixmp platform list
```

To add a specific database, you can use the ixmp CLI[1]:

```
$ ixmp platform add [PLATFORMNAME] jdbc oracle [COMPUTER]:[PORT]/[SERVICENAME]␣
↪[USERNAME] [PASSWORD]
```

You may also want to make this the *default* platform. Unless told otherwise, `message_ix_models` creates `Platform` objects without any arguments (`mp = ixmp.Platform()`); this loads the default platform. Set the default:

[1] `[COMPUTER]` is in this case either the hostname or the IP address.

```
$ ixmp platform add default [PLATFORMNAME]
```

`message_ix` stores only one configuration value in `config.json`: `'message model dir'`, the path to the GAMS model files. MESSAGEix-GLOBIOM uses the GAMS model files from the current `message_ix` `master` branch, so you should not set this, or unset it when using `message_ix_models`.

`message_ix_models` will use the `config.json` value `"message_local_data"` for local data, if it is set and not overridden.

3.1.2 Environment variables

Some code responds to environment variables. For example, ixmp responds to `IXMP_DATA`, which tells it where to find the file `config.json`.

`message_ix_models` responds to `MESSAGE_LOCAL_DATA`; see *the discussion of local data*.

3.1.3 CLI parameters (arguments and options)

Each command has zero or more arguments and options. **Arguments** are mandatory and follow the command name in a certain order. **Options**, as the name implies, are not required. If an option is omitted, a default value is used; the code and `--help` text make clear what the default behaviour is.

Arguments and options are **hierarchical**. Consider the following examples:

```
$ mix-data --opt0=foo cmd1 --opt1=bar arg1 cmd2 --opt2=baz arg2
$ mix-data --opt0=foo cmd1            arg1 cmd3 --opt3=baz arg3a arg3b
```

In these examples:

- `--opt0` is an option that (potentially) affects **any** command, including the subcommands `cmd2` or `cmd3`.
- `--opt1` and `arg1` are an option and mandatory argument to the command `cmd1`. They might not have any relevance to other `mix-data` commands.
- `cmd2` and `cmd3` are distinct subcommands of `cmd1`.
  - They *may* respond to `--opt1` and `arg1`, and to `--opt0`; at least, they *must* not contradict them.
  - They each may have their own options and arguments, which can be distinct.

Tip: Use `--help` for any (sub)command to read about its behaviour. If the help text does not make the behaviour clear, file an issue.

3.1.4 Configuration files and metadata

For some features of the code, the default behaviour is very elaborate and serves for most uses; but we also provide the option to override it. This default behaviour or optional behaviour is defined by reading an input file. These are stored in the *package data* directory.

For example, `mix-models report` loads reporting configuration from `message_ix_models/data/report/global.yaml`, a YAML file with hundreds of lines. Optionally, a different file can be used:

```
$ mix-models report --config other
```

…looks for a file `other.yaml` in the *local data* directory or current working directory. Or:

```
$ mix-models report --config /path/to/another/file.yaml
```

…can be used to point to a file in a different directory.

3.2 Important CLI options and commands

3.2.1 Top-level options and commands

`mix-models --help` describes these:

```
$ mix-models --help
Usage: mix-models [OPTIONS] COMMAND [ARGS]...

  Command-line interface for MESSAGEix-GLOBIOM model tools.

  Every tool and script in this repository is accessible through this CLI.
  Scripts are grouped into commands and sub-commands. For help on specific
  (sub)commands, use --help, e.g.:

          mix-models cd-links --help
          mix-models cd-links run --help

  The top-level options --platform, --model, and --scenario are used by
  commands that access specific message_ix scenarios; these can also be
  specified with --url.

  For more information, see
  https://docs.messageix.org/projects/models2/en/latest/cli.html

Options:
  --url ixmp://PLATFORM/MODEL/SCENARIO[#VERSION]
                                  Scenario URL.
  --platform PLATFORM             Configured platform name.
  --model MODEL                   Model name for some commands.
  --scenario SCENARIO             Scenario name for some commands.
  --version INTEGER               Scenario version.
  --local-data PATH               Base path for local data.
  -v, --verbose                   Print DEBUG-level log messages.
  --help                          Show this message and exit.

Commands:
  cd-links         CD-LINKS project.
  dl               Retrieve data from primary sources.
  engage           ENGAGE project.
  iiasapp          Import power plant capacity.
  material         Model with materials accounting.
  prep-submission  Prepare scenarios for submission to database.
  report           Postprocess results.
  res              MESSAGE-GLOBIOM reference energy system (RES).
```

(continues on next page)

(continued from previous page)

```
  techs            Export data from data/technology.yaml to CSV.
  transport        MESSAGEix-Transport variant.
```

To explain further:

--platform PLATFORM or --url
: By default, message_data connects to the default ixmp Platform. These options direct it to work with a different Platform.

--model MODEL --scenario SCENARIO or --url
: Many commands use an *existing* `Scenario` as a starting point, and begin by cloning that Scenario to a new (model name, scenario name). For any such command, these top-level options define the starting point/initial Scenario to clone/'baseline'.

  In contrast, see `--output-model`, below.

3.2.2 Common options

Since *message_ix_models.model* and *message_ix_models.project* codes often perform similar tasks, their CLI options and arguments are provided in *util.click* for easy re-use. These include:

ssp argument
: This takes one of the values 'SSP1', 'SSP2', or 'SSP3'.

  Commands that will not work for one or more of the SSPs should check the argument value given by the user and raise `NotImplementedError`.

--output-model NAME option
: This option is a counterpart to the top-level `--url`/`--model`/`--scenario` options. A command that starts from one Scenario, and builds one or more Scenarios from it will clone *to* a new (model name, scenario name); `--output-model` gives the model name.

  Current code generates a variety of fixed (non-configurable) scenario names; use `--help` for each command to see which.

To employ these in new code, refer to the example of existing code.

CHAPTER

FOUR

REPRODUCIBILITY

On this page:

Elsewhere:

- A high-level introduction, to how testing supports validity, reproducibility, interoperability, and reusability, in `message_ix_models` and related packages.
- *Test utilities and fixtures (testing)* (`message_ix_models.testing`), on a separate page.

4.1 Strategy

The code in `model.bare` generates a "bare" reference energy system. This is a Scenario that has the same *structure* (ixmp 'sets') as actual instances of the MESSAGEix-GLOBIOM global model, but contains no *data* (ixmp 'parameter' values). Code that operates on the global model can be tested on the bare RES; if it works on that scenario, this is one indication (necessary, but not always sufficient) that it should work on fully-populated scenarios. Such tests are faster and lighter than testing on fully-populated scenarios, and make it easier to isolate errors in the code that is being tested.

4.2 Test suite (`message_ix_models.tests`)

`message_ix_models.tests` contains a suite of tests written using Pytest.

The following is automatically generated documentation of all modules, test classes, functions, and fixtures in the test suite. Each test **should** have a docstring explaining what it checks.

`tests`	Test suite for `message_ix_models`.

4.2.1 message_ix_models.tests

Test suite for `message_ix_models`.

Modules

message_ix_models.tests.model	Tests of *message_ix_models.model* and submodules.
message_ix_models.tests.test_cli	Basic tests of the command line.
message_ix_models.tests.test_import	
message_ix_models.tests.test_testing	
message_ix_models.tests.test_util	Tests of *message_ix_models.util*.
message_ix_models.tests.test_workflow	
message_ix_models.tests.tools	
message_ix_models.tests.util	Tests of submodules of *message_ix_models.util*.

message_ix_models.tests.model

Tests of *message_ix_models.model* and submodules.

Modules

message_ix_models.tests.model.test_bare	
message_ix_models.tests.model.test_build	
message_ix_models.tests.model.test_cli	
message_ix_models.tests.model.test_disutility	Tests of *model.disutility*.
message_ix_models.tests.model.test_emissions	
message_ix_models.tests.model.test_structure	

message_ix_models.tests.model.test_bare

Module Attributes

SET_SIZE	Number of items in the respective YAML files.

message_ix_models.tests.model.test_bare.SET_SIZE

message_ix_models.tests.model.test_bare.**SET_SIZE = {'commodity': 13, 'level': 6, 'node': 15, 'technology': 377, 'year': 28}**

Number of items in the respective YAML files.

Functions

test_create_res(request, test_context, ...)	

message_ix_models.tests.model.test_bare.test_create_res

message_ix_models.tests.model.test_bare.**test_create_res**(*request*, *test_context*, *settings*, *expected*)

message_ix_models.tests.model.test_build

Functions

scenario(test_context)	
spec()	
test_apply_spec0(caplog, scenario, spec)	Require missing element raises ValueError.
test_apply_spec1(caplog, scenario, spec)	Add data using the data= argument.
test_apply_spec2(caplog, scenario, spec)	Remove an element, with fast=True.
test_apply_spec3(caplog, scenario, spec)	Actually remove data.

message_ix_models.tests.model.test_build.scenario

message_ix_models.tests.model.test_build.**scenario**(*test_context*)

message_ix_models.tests.model.test_build.spec

message_ix_models.tests.model.test_build.**spec**()

message_ix_models.tests.model.test_build.test_apply_spec0

message_ix_models.tests.model.test_build.**test_apply_spec0**(*caplog*, *scenario*, *spec*)

Require missing element raises ValueError.

message_ix_models.tests.model.test_build.test_apply_spec1

message_ix_models.tests.model.test_build.**test_apply_spec1**(*caplog*, *scenario*, *spec*)

Add data using the data= argument.

message_ix_models.tests.model.test_build.test_apply_spec2

message_ix_models.tests.model.test_build.**test_apply_spec2**(*caplog*, *scenario*, *spec*)

Remove an element, with fast=True.

message_ix_models.tests.model.test_build.test_apply_spec3

message_ix_models.tests.model.test_build.**test_apply_spec3**(*caplog*, *scenario*, *spec*)

Actually remove data.

message_ix_models.tests.model.test_cli

Functions

test_create_bare(mix_models_cli)	The `res create-bare` CLI command can be invoked.

message_ix_models.tests.model.test_cli.test_create_bare

message_ix_models.tests.model.test_cli.**test_create_bare**(*mix_models_cli*)

The `res create-bare` CLI command can be invoked.

message_ix_models.tests.model.test_disutility

Tests of *model.disutility*.

Functions

groups()	Fixture: list of 2 consumer groups.
minimal_test_data(scenario)	Generate data for *test_minimal()*.
scenario(request, test_context, techs)	Fixture: a `Scenario` with technologies given by *techs()*.
spec(groups, techs, template)	Fixture: a prepared spec for the minimal test case.
techs()	Fixture: list of 2 technologies for which groups can have disutility.
template()	Fixture: :class:.`Code` with annotations, for *disutility.get_spec()*.
test_add(scenario, groups, techs, template)	*disutility.add()* runs on the bare RES; the result solves.
test_data_conversion(scenario, spec)	*data_conversion()* runs.
test_data_source(scenario, spec)	*data_source()* runs.
test_get_data(scenario, spec)	*get_data()* runs.
test_get_spec(groups, techs, template)	*get_spec()* runs and produces expected output.
test_minimal(scenario, groups, techs, template)	Expected results are generated from a minimal test case.

message_ix_models.tests.model.test_disutility.groups

message_ix_models.tests.model.test_disutility.**groups**()

Fixture: list of 2 consumer groups.

message_ix_models.tests.model.test_disutility.minimal_test_data

message_ix_models.tests.model.test_disutility.**minimal_test_data**(*scenario*)

Generate data for *test_minimal()*.

message_ix_models.tests.model.test_disutility.scenario

message_ix_models.tests.model.test_disutility.**scenario**(*request*, *test_context*, *techs*)

Fixture: a `Scenario` with technologies given by *techs()*.

message_ix_models.tests.model.test_disutility.spec

message_ix_models.tests.model.test_disutility.**spec**(*groups*, *techs*, *template*)

Fixture: a prepared spec for the minimal test case.

message_ix_models.tests.model.test_disutility.techs

message_ix_models.tests.model.test_disutility.**techs**()
Fixture: list of 2 technologies for which groups can have disutility.

message_ix_models.tests.model.test_disutility.template

message_ix_models.tests.model.test_disutility.**template**()
Fixture: :class:.`Code` with annotations, for *disutility.get_spec()*.

message_ix_models.tests.model.test_disutility.test_add

message_ix_models.tests.model.test_disutility.**test_add**(*scenario*, *groups*, *techs*, *template*)
disutility.add() runs on the bare RES; the result solves.

message_ix_models.tests.model.test_disutility.test_data_conversion

message_ix_models.tests.model.test_disutility.**test_data_conversion**(*scenario*, *spec*)
data_conversion() runs.

message_ix_models.tests.model.test_disutility.test_data_source

message_ix_models.tests.model.test_disutility.**test_data_source**(*scenario*, *spec*)
data_source() runs.

message_ix_models.tests.model.test_disutility.test_get_data

message_ix_models.tests.model.test_disutility.**test_get_data**(*scenario*, *spec*)
get_data() runs.

message_ix_models.tests.model.test_disutility.test_get_spec

message_ix_models.tests.model.test_disutility.**test_get_spec**(*groups*, *techs*, *template*)
get_spec() runs and produces expected output.

message_ix_models.tests.model.test_disutility.test_minimal

message_ix_models.tests.model.test_disutility.**test_minimal**(*scenario*, *groups*, *techs*, *template*)
Expected results are generated from a minimal test case.

message_ix_models.tests.model.test_emissions

Functions

add_test_data(scenario)	
test_add_tax_emission(request, caplog, ...)	

message_ix_models.tests.model.test_emissions.add_test_data

message_ix_models.tests.model.test_emissions.**add_test_data**(*scenario*)

message_ix_models.tests.model.test_emissions.test_add_tax_emission

message_ix_models.tests.model.test_emissions.**test_add_tax_emission**(*request*, *caplog*, *test_context*)

message_ix_models.tests.model.test_structure

Functions

test_cli_techs(session_context, mix_models_cli)	Test the *techs* CLI command.
test_codelists(kind, exp)	`codelists()` returns the expected IDs.
test_process_units_anno()	

message_ix_models.tests.model.test_structure.test_cli_techs

message_ix_models.tests.model.test_structure.**test_cli_techs**(*session_context*, *mix_models_cli*)

Test the *techs* CLI command.

message_ix_models.tests.model.test_structure.test_codelists

message_ix_models.tests.model.test_structure.**test_codelists**(*kind*, *exp*)

`codelists()` returns the expected IDs.

message_ix_models.tests.model.test_structure.test_process_units_anno

message_ix_models.tests.model.test_structure.**test_process_units_anno**()

Classes

TestGetCodes()	Test `get_codes()` for different code lists.

message_ix_models.tests.model.test_structure.TestGetCodes

class message_ix_models.tests.model.test_structure.**TestGetCodes**

Bases: `object`

Test `get_codes()` for different code lists.

__init__()

Methods

__init__()	
test_commodities()	
test_get_codes(name)	The included code lists can be loaded.
test_hierarchy()	get_codes() returns objects with the expected hierarchical relationship.
test_levels()	
test_node_historic_country()	get_codes() handles ISO 3166 alpha-3 codes for historic countries.
test_nodes(codelist, to_check, length, member)	Tests of node codelists.
test_technologies()	
test_year(codelist, length)	Year code lists can be loaded and contain the correct number of codes.

test_get_codes(*name*)

The included code lists can be loaded.

test_hierarchy()

get_codes() returns objects with the expected hierarchical relationship.

test_node_historic_country()

get_codes() handles ISO 3166 alpha-3 codes for historic countries.

test_nodes(*codelist*, *to_check*, *length*, *member*)

Tests of node codelists.

test_year(*codelist*, *length*)

Year code lists can be loaded and contain the correct number of codes.

Seealso
`TestScenarioInfo.test_year_from_codes()`.

message_ix_models.tests.test_cli

Basic tests of the command line.

Functions

test_cli_debug(mix_models_cli)	The 'debug' CLI command can be invoked.
test_cli_export_test_data(monkeypatch, ...)	The **export-test-data** command can be invoked.
test_cli_help(mix_models_cli, subcommand)	--help works for every CLI command.

message_ix_models.tests.test_cli.test_cli_debug

message_ix_models.tests.test_cli.**test_cli_debug**(*mix_models_cli*)

The 'debug' CLI command can be invoked.

message_ix_models.tests.test_cli.test_cli_export_test_data

message_ix_models.tests.test_cli.**test_cli_export_test_data**(*monkeypatch*, *session_context*, *mix_models_cli*, *tmp_path*)

The **export-test-data** command can be invoked.

message_ix_models.tests.test_cli.test_cli_help

message_ix_models.tests.test_cli.**test_cli_help**(*mix_models_cli*, *subcommand*)

–help works for every CLI command.

message_ix_models.tests.test_import

Functions

test_import()	Test that the package can be imported.

message_ix_models.tests.test_import.test_import

message_ix_models.tests.test_import.**test_import**()

Test that the package can be imported.

message_ix_models.tests.test_testing

Functions

test_bare_res_no_request(test_context)	`bare_res()` works with *request* = `None`.
test_bare_res_solved(request, test_context)	`bare_res()` works with *solve* = `True`.
test_cli_runner(mix_models_cli)	

test_not_ci_skip()	Test not_ci(action="skip").
test_not_ci_xfail()	Test not_ci(action="skip").

message_ix_models.tests.test_testing.test_bare_res_no_request

message_ix_models.tests.test_testing.**test_bare_res_no_request**(*test_context*)

`bare_res()` works with *request* = `None`.

message_ix_models.tests.test_testing.test_bare_res_solved

message_ix_models.tests.test_testing.**test_bare_res_solved**(*request*, *test_context*)

`bare_res()` works with *solve* = `True`.

This test can be removed once this feature of the test function is used by another test.

message_ix_models.tests.test_testing.test_cli_runner

message_ix_models.tests.test_testing.**test_cli_runner**(*mix_models_cli*)

message_ix_models.tests.test_testing.test_not_ci_skip

message_ix_models.tests.test_testing.**test_not_ci_skip**()

Test not_ci(action="skip").

message_ix_models.tests.test_testing.test_not_ci_xfail

message_ix_models.tests.test_testing.**test_not_ci_xfail**()

Test not_ci(action="skip").

message_ix_models.tests.test_util

Tests of *message_ix_models.util*.

Functions

test_as_codes()	Forward reference to a child is silently dropped.
test_as_codes_invalid(data)	as_codes() rejects invalid data.
test_broadcast(caplog)	
test_check_support(test_context)	*check_support()* raises an exception for missing/non-matching values.
test_convert_units(recwarn)	*convert_units()* and *series_of_pint_quantity()* work.
test_copy_column()	
test_ffill()	
test_iter_parameters(test_context)	Parameters indexed by set 'node' can be retrieved.
test_load_package_data(path)	Existing package data can be loaded.
test_load_package_data_invalid()	load_package_data() raises an exception for an unsupported file type.
test_load_package_data_twice(caplog)	Loading the same data twice logs a message.
test_load_private_data(*parts[, suffix])	
test_local_data_path(tmp_path_factory, ...)	
test_make_source_tech0()	
test_make_source_tech1(test_mp)	Test make_source_tech() with a Scenario object as input.
test_maybe_query()	*maybe_query()* works as intended.
test_package_data_path()	
test_private_data_path()	
test_strip_par_data(caplog, test_context)	Test the "dry run" feature of `strip_par_data()`.

message_ix_models.tests.test_util.test_as_codes

message_ix_models.tests.test_util.**test_as_codes**()

Forward reference to a child is silently dropped.

message_ix_models.tests.test_util.test_as_codes_invalid

message_ix_models.tests.test_util.**test_as_codes_invalid**(*data*)

as_codes() rejects invalid data.

message_ix_models.tests.test_util.test_broadcast

message_ix_models.tests.test_util.**test_broadcast**(*caplog*)

message_ix_models.tests.test_util.test_check_support

message_ix_models.tests.test_util.**test_check_support**(*test_context*)

check_support() raises an exception for missing/non-matching values.

message_ix_models.tests.test_util.test_convert_units

message_ix_models.tests.test_util.**test_convert_units**(*recwarn*)

convert_units() and *series_of_pint_quantity()* work.

message_ix_models.tests.test_util.test_copy_column

message_ix_models.tests.test_util.**test_copy_column**()

message_ix_models.tests.test_util.test_ffill

message_ix_models.tests.test_util.**test_ffill**()

message_ix_models.tests.test_util.test_iter_parameters

message_ix_models.tests.test_util.**test_iter_parameters**(*test_context*)

Parameters indexed by set ‘node’ can be retrieved.

message_ix_models.tests.test_util.test_load_package_data

message_ix_models.tests.test_util.**test_load_package_data**(*path*)

Existing package data can be loaded.

message_ix_models.tests.test_util.test_load_package_data_invalid

message_ix_models.tests.test_util.**test_load_package_data_invalid**()

load_package_data() raises an exception for an unsupported file type.

message_ix_models.tests.test_util.test_load_package_data_twice

message_ix_models.tests.test_util.**test_load_package_data_twice**(*caplog*)

Loading the same data twice logs a message.

message_ix_models.tests.test_util.test_load_private_data

message_ix_models.tests.test_util.**test_load_private_data**(**parts*, *suffix=None*)

message_ix_models.tests.test_util.test_local_data_path

message_ix_models.tests.test_util.**test_local_data_path**(*tmp_path_factory*, *session_context*)

message_ix_models.tests.test_util.test_make_source_tech0

message_ix_models.tests.test_util.**test_make_source_tech0**()

message_ix_models.tests.test_util.test_make_source_tech1

message_ix_models.tests.test_util.**test_make_source_tech1**(*test_mp*)

Test make_source_tech() with a Scenario object as input.

message_ix_models.tests.test_util.test_maybe_query

message_ix_models.tests.test_util.**test_maybe_query**()

maybe_query() works as intended.

message_ix_models.tests.test_util.test_package_data_path

message_ix_models.tests.test_util.**test_package_data_path**()

message_ix_models.tests.test_util.test_private_data_path

message_ix_models.tests.test_util.**test_private_data_path**()

message_ix_models.tests.test_util.test_strip_par_data

message_ix_models.tests.test_util.**test_strip_par_data**(*caplog*, *test_context*)

Test the "dry run" feature of `strip_par_data()`.

message_ix_models.tests.test_workflow

Functions

changes_a(s)	Change a scenario by modifying structure data, but not data.
changes_b(s)	Change a scenario by modifying parameter data, but not structure.
test_workflow(request, test_context)	

message_ix_models.tests.test_workflow.changes_a

message_ix_models.tests.test_workflow.**changes_a**(*s*)

Change a scenario by modifying structure data, but not data.

message_ix_models.tests.test_workflow.changes_b

message_ix_models.tests.test_workflow.**changes_b**(*s*)

Change a scenario by modifying parameter data, but not structure.

message_ix_models.tests.test_workflow.test_workflow

message_ix_models.tests.test_workflow.**test_workflow**(*request*, *test_context*)

message_ix_models.tests.tools

Modules

message_ix_models.tests.tools.test_advance	

message_ix_models.tests.tools.test_advance

Functions

test_fuzz_data()	Test that `_fuzz_data()` runs successfully.
test_get_advance_data(test_context)	Test *get_advance_data()*.

message_ix_models.tests.tools.test_advance.test_fuzz_data

message_ix_models.tests.tools.test_advance.**test_fuzz_data**()

Test that `_fuzz_data()` runs successfully.

NB this only produces a file in the `pytest` temporary directory. To update the test specimen in the package directory that is used by *test_get_advance_data()*, see the body of _fuzz_data.

message_ix_models.tests.tools.test_advance.test_get_advance_data

message_ix_models.tests.tools.test_advance.**test_get_advance_data**(*test_context*)

Test *get_advance_data()*.

message_ix_models.tests.util

Tests of submodules of *message_ix_models.util*.

Modules

message_ix_models.tests.util.test_cache	
message_ix_models.tests.util.test_click	Basic tests of the command line.
message_ix_models.tests.util.test_context	
message_ix_models.tests.util.test_logging	
message_ix_models.tests.util.test_node	Tests of *message_ix_models.util.node*.
message_ix_models.tests.util.test_scenarioinfo	
message_ix_models.tests.util.test_sdmx	

message_ix_models.tests.util.test_cache

Functions

test_cached(caplog, test_context, tmp_path)	*cached()* works as expected.

message_ix_models.tests.util.test_cache.test_cached

message_ix_models.tests.util.test_cache.**test_cached**(*caplog*, *test_context*, *tmp_path*)

cached() works as expected.

Todo: test behaviour when *SKIP_CACHE* is True

Classes

TestEncoder()	

message_ix_models.tests.util.test_cache.TestEncoder

class message_ix_models.tests.util.test_cache.**TestEncoder**

Bases: object

__init__()

Methods

__init__()	
test_sdmx()	message_ix_models configures Encoder for Code.

test_sdmx()

message_ix_models configures Encoder for Code.

message_ix_models.tests.util.test_click

Basic tests of the command line.

Functions

temporary_command(group, func)	Context manager that temporarily attaches command *func* to *group*.
test_default_path_cb(session_context, ...)	Test *default_path_cb()*.
test_store_context(mix_models_cli)	Test *store_context()*.

message_ix_models.tests.util.test_click.temporary_command

message_ix_models.tests.util.test_click.**temporary_command**(*group*, *func*)

Context manager that temporarily attaches command *func* to *group*.

message_ix_models.tests.util.test_click.test_default_path_cb

message_ix_models.tests.util.test_click.**test_default_path_cb**(*session_context*, *mix_models_cli*)

Test *default_path_cb()*.

message_ix_models.tests.util.test_click.test_store_context

message_ix_models.tests.util.test_click.**test_store_context**(*mix_models_cli*)

Test *store_context()*.

message_ix_models.tests.util.test_context

Classes

TestContext()	

message_ix_models.tests.util.test_context.TestContext

class message_ix_models.tests.util.test_context.**TestContext**

Bases: object

__init__()

Methods

__init__()	
test_clone_to_dest(caplog, test_context)	
test_deepcopy(session_context)	Paths are preserved through deepcopy().
test_default_value(test_context)	
test_get_cache_path(pytestconfig, test_context)	cache_path() returns the expected output.
test_get_instance(session_context)	
test_get_local_path(tmp_path_factory, ...)	
test_get_platform(session_context)	
test_get_scenario(test_context)	
test_handle_cli_args()	
test_only()	
test_repr()	
test_set_scenario(test_context)	
test_use_defaults(caplog)	

test_deepcopy(*session_context*)

Paths are preserved through deepcopy().

test_get_cache_path(*pytestconfig*, *test_context*)

cache_path() returns the expected output.

message_ix_models.tests.util.test_logging

Functions

test_mark_time(caplog)	
test_silence_log(caplog)	

message_ix_models.tests.util.test_logging.test_mark_time

message_ix_models.tests.util.test_logging.**test_mark_time**(*caplog*)

message_ix_models.tests.util.test_logging.test_silence_log

message_ix_models.tests.util.test_logging.**test_silence_log**(*caplog*)

message_ix_models.tests.util.test_node

Tests of *message_ix_models.util.node*.

Functions

input()	Fixture: test data for adapt_R11_R14().
test_adapt_df(input, func, N, expected, ...)	adapt_R11_R14() handles pandas.DataFrame.
test_adapt_qty(input, func, expected, node_loc)	adapt_R11_R14() handles genno.Quantity.
test_identify_nodes(caplog, test_context, ...)	
test_identify_nodes1(test_context)	
test_mapping_adapter()	Generic test of MappingAdapter.

message_ix_models.tests.util.test_node.input

message_ix_models.tests.util.test_node.**input**()

Fixture: test data for adapt_R11_R14().

message_ix_models.tests.util.test_node.test_adapt_df

message_ix_models.tests.util.test_node.**test_adapt_df**(*input*, *func*, *N*, *expected*, *target_nodes*)

adapt_R11_R14() handles pandas.DataFrame.

message_ix_models.tests.util.test_node.test_adapt_qty

message_ix_models.tests.util.test_node.**test_adapt_qty**(*input*, *func*, *expected*, *node_loc*)

adapt_R11_R14() handles genno.Quantity.

message_ix_models.tests.util.test_node.test_identify_nodes

message_ix_models.tests.util.test_node.**test_identify_nodes**(*caplog*, *test_context*, *regions*)

message_ix_models.tests.util.test_node.test_identify_nodes1

message_ix_models.tests.util.test_node.**test_identify_nodes1**(*test_context*)

message_ix_models.tests.util.test_node.test_mapping_adapter

message_ix_models.tests.util.test_node.**test_mapping_adapter**()

Generic test of MappingAdapter.

message_ix_models.tests.util.test_scenarioinfo

Classes

TestScenarioInfo()
TestSpec()

message_ix_models.tests.util.test_scenarioinfo.TestScenarioInfo

class message_ix_models.tests.util.test_scenarioinfo.**TestScenarioInfo**

Bases: object

__init__()

Methods

__init__()	
test_empty()	ScenarioInfo created from scratch.
test_from_scenario(test_context)	ScenarioInfo initialized from an existing Scenario.
test_repr()	
test_units(caplog)	Test both *io_units()* and *units_for()*.
test_update()	
test_year_from_codes(caplog, codelist, y0, ...)	

test_empty()

ScenarioInfo created from scratch.

test_from_scenario(*test_context*)

ScenarioInfo initialized from an existing Scenario.

test_units(*caplog*)

Test both *io_units()* and *units_for()*.

message_ix_models.tests.util.test_scenarioinfo.TestSpec

class message_ix_models.tests.util.test_scenarioinfo.**TestSpec**

Bases: object

__init__()

Methods

__init__()
test_getitem()
test_merge()
test_setitem()

message_ix_models.tests.util.test_sdmx

Functions

test_eval_anno(caplog)

message_ix_models.tests.util.test_sdmx.test_eval_anno

message_ix_models.tests.util.test_sdmx.**test_eval_anno**(*caplog*)

4.3 Running the test suite

Some notes for running the test suite.

cached() is used to cache the data resulting from slow operations, like parsing large input files. Data are stored in a location described by the *Context* setting `cache_path`. The test suite interacts with caches in two ways:

- `--local-cache`: Giving this option causes pytest to use whatever cache directory is configured for normal runs/usage of `message_ix_models` or **`mix-models`**.
- By default (without `--local-cache`), the test suite uses pytest's built-in cache fixture. This creates and uses a temporary directory, usually `.pytest_cache/d/cache/` within the repository root. This location is used *only* by tests, and not by normal runs/usage of the code.

In either case:

- The tests use existing cached data in these locations and skip over code that generates this data. If the generating code is changed, the cached data **must** be deleted in order to actually check that the code runs properly.
- Running the test suite with `--local-cache` causes the local cache to be populated, and this will affect subsequent runs.
- The continuous integration (below) services don't preserve caches, so code always runs.

4.4 Continuous testing

The test suite (*message_ix_models.tests*) is run using GitHub Actions for new commits on the `main` branch, or on any branch associated with a pull request.

Because it is closed-source and requires access to internal IIASA resources, including databases, continuous integration for `message_data` is handled by an internal server running TeamCity: https://ene-builds.iiasa.ac.at/project/message (requires authorization)

4.5 Prepare data for testing

Use the `export-test-data` CLI command:

```
mix-models --url="ixmp://ixmp-dev/ENGAGE_SSP2_v4.1.7/baseline" export-test-data
```

See also the documentation for `export_test_data()`. Use the **`--exclude`**, **`--nodes`**, and **`--techs`** options to control the content of the resulting file.

CHAPTER

FIVE

DISTRIBUTED COMPUTING

This page introduces considerations, tools, and features for using **distributed** or **high-throughput computing** with MESSAGEix-GLOBIOM.

5.1 Overview

Scenarios in the MESSAGEix-GLOBIOM global model family are characterized by:

- ca. 100 MB of data, depending on storage format (e.g. in a `JDBCBackend` local, HyperSQL database, or Scenario/model data in Excel files).
- `solve()` times of between 10 and 60 minutes, depending on hardware and configuration, plus similar amounts of time to run the legacy reporting in `message_data`.
- Memory usage of ~10 GB or more using `JDBCBackend`, currently the only supported backend.

These resource needs can be a bottleneck in applications, for example:

- where many/related scenarios must be solved.
- when iteration (repeatedly solving 1 or more scenarios) is a key approach in developing code that sets up scenarios.

To improve research productivity, researchers may choose to run scenarios or 'workflows' (a combination of solving scenarios and pre- and post-processing steps or codes) through **distributed computing**, i.e. not on their local machine. Hardware and software environments for distributed computing can vary widely, and can be categorized in multiple ways, such as:

1. More powerful single-CPU systems, accessed remotely.
2. Cloud services, e.g. Google Compute Engine; Amazon AWS; Github Actions; etc. providing access to one or more machines.
3. Dedicated cluster systems (sometimes labelled **high-throughput computing**, HTC, or **high-performance computing**, HPC, systems) for scientific computing, operated by a variety of parties.

5.2 Tooling

- `message_ix_models` and related packages (`ixmp`, `message_ix`, `message_data`) aim to provide *simple, general-purpose features* that allow working with a variety of distributed systems.
- Specific configuration necessarily depends on the specific system(s) in use and the researcher's application.
- These features should not duplicate features of existing tools such as Slurm, HTCondor, etc., but rather allow `message_ix` & co. to be used with/through those.
- The individual features, tools, utilities, etc. should each be simple, i.e. do one thing, and do it well.

Todo: Extend.

CHAPTER

SIX

MODELS AND VARIANTS (`MODEL`)

6.1 `model.structure`: Model structure information

message_ix_models.model.structure.**codelists**(*kind: str*) → List[str]

Return a valid IDs for code lists of *kind*.

Parameters

kind (*str*) – "node" or "year".

message_ix_models.model.structure.**process_units_anno**(*set_name: str*, *code: Code*, *quiet: bool = False*) → None

Process an annotation on *code* with id="units".

The annotation text is wrapped as `'registry.Unit("{text}")'`, such that it can be retrieved with *eval_anno()* or *ScenarioInfo.units_for()*. If *code* has direct children, the annotation is also copied to those codes.

Parameters

- **set_name** (*str*) – Used in logged messages when *quiet* is `False`.
- **quiet** (*bool, optional*) – If `False` (the default), log on level WARNING if:
  - the annotation is missing, or
  - its text is not parseable with the `pint` application registry, i.e. `iam_units.registry`.

  Otherwise, log on DEBUG.

message_ix_models.model.structure.**get_codes**(*name: str*) → List[Code]

Return codes for the dimension/set *name* in MESSAGE-GLOBIOM scenarios.

The information is read from `data/name.yaml`, e.g. `data/technology.yaml`.

When *name* includes "node", then child codes are automatically populated from the ISO 3166 database via `pycountry`. For instance:

```
myregion:
  name: Custom region
  child: [AUT, SCG]
```

…results in a region with child codes for Austria (a current country) and the formerly-existing country Serbia and Montenegro.

Parameters

name (*str*) – Any `.yaml` file in the folder `message_ix_models/data/`.

Returns
Every Code has `id`, `name`, `description`, and `annotations` attributes. Calling `str()` on a code returns its `id`.

Return type
list of Code

The available code lists are reproduced as part of this documentation. The returned code objects have annotations that vary by set. See:

- *Other code lists*
- *Node code lists*

Also available is `cd_links/unit.yaml`. This is a project-specific list of units appearing in CD-LINKS scenarios.

Example:

```
>>> from message_ix_models.model.structure import get_codes
>>> codes = get_codes("node/R14")

# Show the codes
>>> codes
[<Code ABW: Aruba>,
 <Code AFG: Afghanistan>,
 <Code AGO: Angola>,
 ...
 <Code ZWE: Zimbabwe>,
 <Code World: World>,
 ...
 <Code R11_PAS: Other Pacific Asia>,
 <Code R11_SAS: South Asia>,
 <Code R11_WEU: Western Europe>]

# Retrieve one code matching a certain ID
>>> world = codes[codes.index("World")]

# Get its children's IDs strings, e.g. for a "node" dimension
>>> [str(c) for c in world.child]
['R11_AFR',
 'R11_CPA',
 'R11_EEU',
 'R11_FSU',
 'R11_LAM',
 'R11_MEA',
 'R11_NAM',
 'R11_PAO',
 'R11_PAS',
 'R11_SAS',
 'R11_WEU']

# Navigate from one ISO 3166-3 country code to its parent
>>> AUT = codes[codes.index("AUT")]
>>> AUT.parent
<Code R11_WEU: Western Europe>
```

See also:

`adapt_R11_R14()`

CHAPTER

SEVEN

REPRODUCE THE RES (`MODEL.BARE`)

In contrast to `model.create`, this module creates the RES 'from scratch'. *`create_res()`* begins by creating a new, totally empty `Scenario` and adding data to it (instead of cloning and modifying an existing scenario).

Note: Currently, the Scenario returned by *`create_res()`*...

- is not complete, nor the official/preferred version of MESSAGEix-GLOBIOM, and as such **must not** be used for actual research,
- however, it **should** be used for creating unit tests of other code that is designed to operate on MESSAGEix-GLOBIOM scenarios; code that works against the bare RES should also work against MESSAGEix-GLOBIOM scenarios.

`bare.get_spec()` can also be used directly, to get a *description* of the RES based on certain settings/options, but without any need to connect to a database, load an existing Scenario, or call *`bare.create_res()`*. This can be useful in code that processes data into a form compatible with MESSAGEix-GLOBIOM.

7.1 Context settings

Setting	Type	Description
regions	str	The 'node' set (regional aggregation) to use; must be "R14" (default), "R11", "RCP" or "ISR".
years	str	The 'year' set (time periods) to use; must be "B" (default) or "A".
res_with_dummies	bool	If `True`, create and include dummy technologies. See `get_dummy_data()`. Default `False`

See documentation for further context settings in *Top-level settings*.

7.2 Code reference

message_ix_models.model.bare.**SETTINGS = {'regions': ['R14', 'ADVANCE', 'ISR', 'R11', 'R12', 'R32', 'RCP'], 'res_with_dummies': [False, True], 'years': ['B', 'A']}**

Settings and valid values; the default is listed first.

- `regions`: recognized lists of nodes; these match the files `data/node/*.yaml`.
- `years`: recognized lists of time periods ("years"); these match the files `data/year/*.yaml`

message_ix_models.model.bare.**create_res**(*context*, *quiet=True*)

Create a 'bare' MESSAGEix-GLOBIOM reference energy system (RES).

Parameters

- **context** (*.Context*) – `Context.scenario_info` determines the model name and scenario name of the created Scenario. If not provided, the defaults are:
  - Model name generated by *name()*.
  - Scenario name "baseline".
- **quiet** (*bool, optional*) – Passed to *quiet* argument of *build.apply_spec()*.

Returns
A scenario as described by `bare.get_spec()`, prepared using *apply_spec()*.

Return type
message_ix.Scenario

message_ix_models.model.bare.**name**(*context*)

Generate a candidate name for a model given *context*.

The name has a form like:

```
MESSAGEix-GLOBIOM R99 YA +D
```

where:

- "R99" is the node list/regional aggregation.
- "YA" indicates the year codelist from `data/year/A.yaml` is used.
- "+D" indicates dummy set elements are included in the structure.

bare.**get_spec**() → Mapping[str, *ScenarioInfo*]

Return the spec for the MESSAGE-GLOBIOM global model RES.

Return type
`dict` of *ScenarioInfo* objects

Since the RES is the base for all variants of MESSAGEix-GLOBIOM, the 'require' and 'remove' portions of the spec are empty.

For the 'add' section, *message_ix_models.model.structure.get_codes()* is used to retrieve data from the YAML files in `message_ix_models`.

Settings are retrieved from *context*, as above.

message_ix_models.model.data.**get_data**(*scenario*, *context*, *spec*, ***options*)

Data for the bare RES.

Todo: With ixmp#212 merged, some *model.bare* code could be moved to a new class and method like `MESSAGE_GLOBIOM.initialize()`.

CHAPTER

EIGHT

BUILDING MODELS (`MODEL.BUILD`)

apply_spec() can be used to be build (compose, assemble, construct, …) models given three pieces of information:

- A `message_ix.Scenario` to be used as a base. This scenario may be empty.
- A specification, or *Spec*, which is trio of *ScenarioInfo* objects; see below.
- An optional function that adds or produces *data* to add to the *scenario*.

The spec is applied as follows:

1. For each `ixmp` set that exists in *scenario*:
   a. **Required** elements from *Spec.require*, if any, are checked.

      If they are missing, `apply_spec()` raises `ValueError`. This indicates that *spec* is not compatible with the given *scenario*.

   b. Elements from *Spec.remove*, if any, are **removed**.

      Any parameter values which reference these set elements are also removed, using `strip_par_data()`.

   c. New set elements from *Spec.add*, if any, are **added**.
2. Elements in `spec.add.set["unit"]` are added to the Platform on which *scenario* is stored.
3. The *data* argument, a function, is called with *scenario* as the first argument, and a keyword argument *dry_run* from *apply_spec()*. *data* may either add to *scenario* directly (by calling `Scenario.add_par()` and similar methods); or it can return a `dict` that can be passed to `add_par_data()`.

The following modules use this workflow and can be examples for developing similar code:

- *model.bare*
- *model.disutility*
- `message_data.model.transport`

8.1 Code reference

`message_ix_models.model.build.`**`apply_spec`**(*scenario: Scenario*, *spec: Union[*Spec, *Mapping[str,* ScenarioInfo*]]*, *data: Optional[Callable] = None*, ***options*)

Apply *spec* to *scenario*.

Parameters

- **spec** (*.Spec*) – Specification of changes to make to *scenario*.

- **data** (*callable, optional*) – Function to add data to *scenario*. *data* can either manipulate the scenario directly, or return a `dict` compatible with `add_par_data()`.
- **dry_run** (*bool*) – Don't modify *scenario*; only show what would be done. Default `False`. Exceptions will still be raised if the elements from `spec['required']` are missing; this serves as a check that the scenario has the required features for applying the spec.
- **fast** (*bool*) – Do not remove existing parameter data; increases speed on large scenarios.
- **quiet** (*bool*) – Only show log messages at level `ERROR` and higher. If `False` (default), show log messages at level `DEBUG` and higher.
- **message** (*str*) – Commit message.

See also:

`add_par_data`, `strip_par_data`, `Code`, *ScenarioInfo*

`message_ix_models.model.build.`**`ellipsize`**`(`*elements: List*`)` → str

Generate a short string representation of *elements*.

If the list has more than 5 elements, only the first two and last two are shown, with "…" between.

CHAPTER

NINE

EMISSIONS DATA (MODEL.EMISSIONS)

model.emissions contains codes for working with emissions data, including policies on emissions.

In general, models created with `message_ix_models`:

- Use tonnes of carbon equivalent ("t C") as units for mass of emissions.
- Use "USD / t C" as units for price of emissions. Because (as of 2022-07-20) `iam_units` treats "USD" as an alias for "USD_2005", this is implicitly USD_2005 / t C.

`message_ix_models.model.emissions.`**`add_tax_emission`**(*scen: Scenario*, *price: float*, *conversion_factor: Optional[float] = None*, *drate_parameter='drate'*) → None

Add a global CO_2 price to *scen*.

A carbon price is implemented on node="World" by populating the MESSAGEix parameter `tax_emission`, starting from the first model year and covering the entire model horizon. The tax has an annual growth rate equal to the discount rate.

The other dimensions of `tax_emission` are filled with type_emission="TCE" and type_tec="all".

Parameters

- **scen** (*message_ix.Scenario*) –
- **price** (*float*) – Price in the first model year, in USD / tonne CO_2.
- **conversion_factor** (*float, optional*) – Factor for converting *price* into the model's internal emissions units, currently USD / tonne carbon. Optional: a default value is retrieved from `iam_units`.
- **drate_parameter** (*str; one of "drate" or "interestrate"*) – Name of the parameter to use for the growth rate of the carbon price.

CHAPTER

TEN

CONSUMER DISUTILITY

This module provides a generalized consumer disutility formulation, currently used by `message_data.model.transport`. The formulation rests on the concept of "consumer groups"; each consumer group may have a distinct disutility associated with using the outputs of each technology. A set of 'pseudo-'/'virtual'/non-physical "usage technologies" converts the outputs of the actual technologies into the commodities demanded by each group, while also requiring input of a costly "disutility" commodity.

10.1 Method & usage

Use this code by calling *`add()`*, which takes arguments that describe the concrete usage:

Consumer groups
: This is a list of `Code` objects describing the consumer groups. The list must be 1-dimensional, but can be composed (as in `message_data.model.transport`) from multiple dimensions.

Technologies
: This is a list of `Code` objects describing the technologies for which the consumers in the different groups experience disutility. Each object must be have 'input' and 'output' annotations (`Code.anno`); each of these is a `dict` with the keys 'commodity', 'input', and 'unit', describing the source or sink for the technology.

Template
: This is also a `Code` object, similar to those in `technologies`; see below.

The code creates a source technology for the "disutility" commodity. The code does *not* perform the following step(s) needed to completely parametrize the formulation:

- Set consumer group-specific `demand` parameter values for new commodities.
- Set the amounts of "disutility" commodities used as `input` to the new usage technologies.

These must be parametrized based on the particular application.

10.2 Detailed example

This example is similar to the one used in *`test_disutility.test_minimal()`*:

```
# Two consumer groups
groups = [Code(id="g0"), Code(id="g1")]

# Two technologies, for which groups may have different disutilities.
techs = [Code(id="t0"), Code(id="t1")]
```

(continues on next page)

(continued from previous page)

```
# Add generalized disutility formulation to some technologies
disutility.add(
    scenario,
    groups=groups,
    technologies=techs,

    template=Code(
        # Template for IDs of conversion technologies
        id="usage of {technology} by {group}",

        # Templates for inputs of conversion technologies
        input=dict(
            # Technology-specific output commodity
            commodity="output of {technology}",
            level="useful",
            unit="kg",
        ),

        # Templates for outputs of conversion technologies
        output=dict(
            # Consumer-group-specific demand commodity
            commodity="demand of group {group}",
            level="useful",
            unit="kg",
        ),
    ),
    **options,
)
```

add() uses *get_spec()* to generate a specification that adds the following:

- For the set `commodity`:
  - The single element "disutility".
  - One element per *technologies*, using the *template* "input" annotation, e.g. "output of t0" generated from `output of {technology}` and the id "t0". These **may** already be present in the *scenario*; if not, the spec causes them to be added.
  - One elements per *groups*, using the *template* "output" annotation, e.g. "demand of group g1" generated from `demand of group {group}` and the id "g1". These **may** already be present in the *scenario*; if not, the spec causes them to be added.
- For the set `technology`:
  - The single element "disutility source".
  - One element per each combination of disutility-affected technology (*technologies*) and consumer group (*groups*). For example, "usage of t0 by g1" generated from `usage of {technology} by {group}`, and the ids "t0" and "g1".

The spec is applied to the target scenario using *model.build.apply_spec()*. If the arguments produce a spec that is inconsistent with the target scenario, an exception will by raised at this point.

Next, *add()* uses *data_conversion()* and *data_source()* to generate:

- `output` and `var_cost` parameter data for "disutility source". This technology outputs the unitless commodity "disutility" at a cost of 1.0 per unit.

- `input` and `output` parameter data for the new usage technologies. For example, the new technology "usage of t0 by g1"…
  - …takes input from the *technology-specific* commodity "output of t0".
  - …takes input from the common commodity "disutility", in an amount specific to group "g1".
  - …outputs to a *group-specific* commodity "demand of group g1".

Note that the *technologies* towards which the groups have disutility are assumed to already be configured to `output` to the corresponding commodities. For example, the technology "t0" outputs to the commodity "output of t0"; the `output` values for this technology are **not** added/introduced by *add()*.

(Dis)utility is generally dimensionless. In `pint` and thus also `message_ix_models`, this should be represented by `""`. However, to work around iiasa/ixmp#425, *data_conversion()* and *data_source()* return data with `"-"` as units. See GH #45 for more information.

10.3 Code reference

See also *message_ix_models.tests.model.test_disutility*.

`message_ix_models.model.disutility.`**`add`**(*scenario: Scenario*, *groups: Sequence[Code]*, *technologies: Sequence[Code]*, *template: Code*, ***options*) → *Spec*

Add disutility formulation to *scenario*.

`message_ix_models.model.disutility.`**`data_conversion`**(*info*, *spec*) → Mapping[str, DataFrame]

Generate input and output data for disutility conversion technologies.

`message_ix_models.model.disutility.`**`data_source`**(*info*, *spec*) → Mapping[str, DataFrame]

Generate data for a technology that emits the "disutility" commodity.

`message_ix_models.model.disutility.`**`dp_for`**(*col_name: str*, *info:* ScenarioInfo) → Series

`pandas.DataFrame.assign()` helper for `duration_period`.

Returns a callable to be passed to `pandas.DataFrame.assign()`. The callable takes a data frame as the first argument, and returns a `pandas.Series` based on the `duration_period` parameter in *info*, aligned to *col_name* in the data frame.

Currently (2021-04-07) unused.

`message_ix_models.model.disutility.`**`get_data`**(*scenario*, *spec*, ***kwargs*) → Mapping[str, DataFrame]

Get data for disutility formulation.

Calls *data_conversion()* and *data_source()*.

Parameters
spec (*dict*) – The output of *get_spec()*.

`message_ix_models.model.disutility.`**`get_spec`**(*groups: Sequence[Code]*, *technologies: Sequence[Code]*, *template: Code*) → *Spec*

Get a spec for a disutility formulation.

Parameters

- **groups** (*list of Code*) – Identities of the consumer groups with distinct disutilities.
- **technologies** (*list of Code*) – The technologies to which the disutilities are applied.
- **template** (*.Code*) –

CHAPTER

ELEVEN

SPECIFIC RESEARCH PROJECTS (PROJECT)

CHAPTER

TWELVE

GENERAL PURPOSE MODELING TOOLS

"Tools" can include, *inter alia*:

- Codes for retrieving data from specific data sources and adapting it for use with `message_ix_models`.
- Codes for modifying scenarios; although tools for building models should go in *`message_ix_models.model`*.

On this page:

12.1 ADVANCE data

`get_advance_data`([query])	Return data from the ADVANCE Work Package 2 data snapshot at *`LOCATION`*.
`advance_data`(variable[, query])	Return a single ADVANCE data *variable* as a `genno.Quantity`.

`message_ix_models.tools.advance.LOCATION = ('advance', 'advance_compare_20171018-134445.csv.zip')`

Expected location of the ADVANCE WP2 data snapshot.

This is a location relative to a parent directory. The specific parent directory depends on whether `message_data` is available:

Without `message_data`:
The code finds the data within *(3) Other, system-specific ("local") directories* (see discussion there for how to configure this location). Users should:

1. Visit https://tntcat.iiasa.ac.at/ADVANCEWP2DB/dsd?Action=htmlpage&page=about and register for access to the data.
2. Log in.
3. Download the snapshot with the file name given in *`LOCATION`* to a subdirectory `advance/` within their local data directory.

With `message_data`:
The code finds the data within *(2) data/ directory in the message_data repo*. The snapshot is stored directly in the repository using Git LFS.

Handle data from the ADVANCE project.

message_ix_models.tools.advance.**DIMS = ['model', 'scenario', 'region', 'variable', 'unit', 'year']**

Standard dimensions for data produced as snapshots from the IIASA ENE Program "WorkDB".

Todo: Move to a common location for use with other snapshots in the same format.

message_ix_models.tools.advance.**NAME = 'advance_compare_20171018-134445.csv'**

Name of the data file within the archive.

message_ix_models.tools.advance.**_fuzz_data**(*size=100.0*, *include: List[Tuple[str, str]] = []*)

Select a subset of the data for use in testing.

Parameters

- **size** (*numeric*) – Number of rows to include.
- **include** (*sequence of 2-tuple (str, str)*) – (variable name, unit) to include. The data will be partly duplicated to ensure the given variable name(s) are included.

message_ix_models.tools.advance.**_read_workdb_snapshot**(*path: Path*, *name: str*) → Series

Read the data file.

The expected format is a ZIP archive at *path* containing a member at *name* in CSV format, with columns corresponding to *DIMS*, except for "year", which is stored as column headers ('wide' format). (This corresponds to an older version of the "IAMC format," without more recent additions intended to represent sub-annual time resolution using a separate column.)

Todo: Move to a general location for use with other files in the same format.

Data returned by this function is cached using *cached()*; see also *SKIP_CACHE*.

message_ix_models.tools.advance.**advance_data**(*variable: str*, *query: Optional[str] = None*) → Quantity

Return a single ADVANCE data *variable* as a genno.Quantity.

Parameters

query (*str, optional*) – Passed to *get_advance_data()*.

Returns

with the dimensions *DIMS* and name *variable*. If the units of the data for *variable* are consistent and parseable by pint, the returned Quantity has these units; otherwise units are discarded and the returned Quantity is dimensionless.

Return type

genno.Quantity

message_ix_models.tools.advance.**get_advance_data**(*query: Optional[str] = None*) → Series

Return data from the ADVANCE Work Package 2 data snapshot at *LOCATION*.

Parameters

query (*str, optional*) – Passed to pandas.DataFrame.query() to limit the returned values.

Returns

with a pandas.MultiIndex having the levels *DIMS*.

Return type

pandas.Series

Data returned by this function is cached using *cached()*; see also *SKIP_CACHE*.

CHAPTER

THIRTEEN

LOW-LEVEL UTILITIES (`util`)

Submodules:

click	Command-line utilities.
context	Context and settings for `message_ix_models` code.
importlib	Load model and project code from `message_data`.
_logging	Logging utilities.
scenarioinfo	`ScenarioInfo` class.

Commonly used:

adapt_R11_R12	Adapt data using mappings for 1 or more dimension(s).
adapt_R11_R14	Adapt data using mappings for 1 or more dimension(s).
as_codes(data)	Convert *data* to a `list` of `Code` objects.
broadcast(df[, labels])	Fill missing data in *df* by broadcasting.
cached(func)	Decorator to cache the return value of a function *func*.
check_support(context[, settings, desc])	Check whether a Context is compatible with certain *settings*.
convert_units(s, unit_info[, store])	Convert units of *s*, for use with `apply()`.
copy_column(column_name)	For use with `pandas.DataFrame.assign()`.
ffill(df, dim, values[, expr])	Forward-fill *df* on *dim* to cover *values*.
identify_nodes(scenario)	Return the ID of a node codelist given the contents of *scenario*.
load_package_data(*parts[, suffix])	Load a `message_ix_models` package data file and return its contents.
load_private_data(*parts)	Load a private data file from `message_data` and return its contents.
local_data_path(*parts)	Construct a path for local data.
make_io(src, dest, efficiency[, on])	Return input and output data frames for a 1-to-1 technology.
make_matched_dfs(base, **par_value)	Return data frames derived from *base* for multiple parameters.
make_source_tech(info, common, **values)	Return parameter data for a 'source' technology.
maybe_query(series, query)	Apply `pandas.Series.query()` if the *query* arg is not `None`.
merge_data(base, *others)	Merge dictionaries of DataFrames together into *base*.
package_data_path(*parts)	Construct a path to a file under `message_ix_models/data/`.
private_data_path(*parts)	Construct a path to a file under `data/` in `message_data`.
same_node(df)	Fill 'node_origin'/'node_dest' in *df* from 'node_loc'.
series_of_pint_quantity(*args, **kwargs)	Suppress a spurious warning.
Context(*args, **kwargs)	Context and settings for `message_ix_models` code.
ScenarioInfo([scenario])	Information about a `Scenario` object.
Spec(add, remove, require)	A specification for the structure of a model or variant.

message_ix_models.util.**as_codes**(*data: Union[List[str], Dict[str, Union[Dict, Code]]]*) → List[Code]

Convert *data* to a `list` of `Code` objects.

Various inputs are accepted:

- `list` of `str`.
- `dict`, in which keys are `Code.id` and values are further `dict` with keys matching other `Code` attributes.

message_ix_models.util.**broadcast**(*df: DataFrame*, *labels: Optional[DataFrame] = None*, ***kwargs*) → DataFrame

Fill missing data in *df* by broadcasting.

broadcast() is suitable for use with partly-filled data frames returned by `message_ix.util.make_df()`, with 1 column per dimension, plus a "value" column. It is also usable with `pandas.DataFrame.pipe()` for chained operations.

labels (if any) are handled first: one copy or duplicate of *df* is produced for each row (set of labels) in this argument. Then, *kwargs* are handled; *broadcast()* returns one copy for each element in the cartesian product of the dimension labels given by *kwargs*.

Parameters

- **labels** (*pandas.DataFrame*) – Each column (dimension) corresponds to one in *df*. Each row represents one matched set of labels for those dimensions.
- **kwargs** – Keys are dimensions. Values are labels along that dimension to fill.

Returns

The length is either 1 or an integer multiple of the length of *df*.

Return type

pandas.DataFrame

Raises

ValueError – if any of the columns in *labels* or *kwargs* are not present in *df*, or if those columns are present but not empty.

Examples

```
>>> from message_ix import make_df
>>> from message_ix_models.util import broadcast
# Create a base data frame with some empty columns
>>> base = make_df("input", technology="t", value=[1.1, 2.2])
# Broadcast (duplicate) the data across 2 dimensions
>>> df = base.pipe(broadcast, node_loc=["node A", "node B"], mode=["m0", "m1"])
# Show part of the result
>>> df.dropna(axis=1)
  mode node_loc technology  value
0   m0   node A          t    1.1
1   m0   node A          t    2.2
2   m0   node B          t    1.1
3   m0   node B          t    2.2
4   m1   node A          t    1.1
5   m1   node A          t    2.2
6   m1   node B          t    1.1
7   m1   node B          t    2.2
```

message_ix_models.util.**cached**(*func: Callable*) → Callable

Decorator to cache the return value of a function *func*.

On a first call, the data requested is returned and also cached under *Context.get_cache_path()*. On subsequent calls, if the cache exists, it is used instead of calling the (possibly slow) *func*.

When *SKIP_CACHE* is true, *func* is always called.

See also:

Caching in the genno documentation

message_ix_models.util.**check_support**(*context*, *settings={}*, *desc: str = ''*) → None

Check whether a Context is compatible with certain *settings*.

Raises

- **NotImplementedError** – if any *context* value for a key of *settings* is not among the values in *settings*.
- **KeyError** – if the key is not set on *context* at all.

See also:

Advertise and check compatibility

message_ix_models.util.**convert_units**(*s: Series*, *unit_info: Mapping[str, Tuple[float, str, Optional[str]]]*, *store='magnitude'*) → Series

Convert units of *s*, for use with `apply()`.

`s.name` is used to retrieve a tuple of (*factor*, *input_unit*, *output_unit*) from *unit_info*. The (`float`) values of *s* are converted to `pint.Quantity` with the *input_unit* and factor; then cast to *output_unit*, if provided.

Parameters

- **s** (*pandas.Series*) –
- **unit_info** (*dict (str -> tuple)*) – Mapping from quantity name (matched to `s.name`) to 3-tuples of (*factor*, *input_unit*, *output_unit*). *output_unit* may be `None`. For example, see `ikarus.UNITS`.
- **store** (*"magnitude" or "quantity"*) – If "magnitude", the values of the returned series are the magnitudes of the results, with no output units. If "quantity", the values are scalar `Quantity` objects.

Returns
Same shape, index, and values as *s*, with output units.

Return type
pandas.Series

message_ix_models.util.**eval_anno**(*obj: AnnotableArtefact*, *id: str*)

Retrieve the annotation *id* from *obj*, run `eval()` on its contents.

This can be used for unpacking Python values (e.g. `dict`) stored as an annotation on a `Code`.

Returns `None` if no attribute exists with the given *id*.

message_ix_models.util.**identify_nodes**(*scenario: Scenario*) → str

Return the ID of a node codelist given the contents of *scenario*.

Returns
The ID of the *Node code lists* containing the regions of *scenario*.

Return type
str

Raises
ValueError – if no codelist can be identified, or the nodes in the scenario do not match the children of the "World" node in the codelist.

message_ix_models.util.**load_package_data**(**parts: str*, *suffix: Optional[str] = '.yaml'*) → Any

Load a `message_ix_models` package data file and return its contents.

Data is re-used if already loaded.

Example

The single call:

```
>>> info = load_package_data("node", "R11")
```

1. loads the metadata file `data/node/R11.yaml`, parsing its contents,
2. stores those values at `PACKAGE_DATA["node R11"]` for use by other code, and
3. returns the loaded values.

Parameters

- **parts** (*iterable of str*) – Used to construct a path under `message_ix_models/data/`.
- **suffix** (*str, optional*) – File name suffix, including, the ".", e.g. `.yaml`.

Returns
Configuration values that were loaded.

Return type
dict

`message_ix_models.util.`**load_private_data**(**parts: str*) → Mapping

Load a private data file from `message_data` and return its contents.

Analogous to `load_package_data`, but for non-public data.

Parameters
parts (*iterable of str*) – Used to construct a path under `data/` in the `message_data` repository.

Returns
Configuration values that were loaded.

Return type
dict

Raises
RuntimeError – if `message_data` is not installed.

`message_ix_models.util.`**local_data_path**(**parts*) → Path

Construct a path for local data.

The setting `message local data` in the user's ixmp configuration file is used as a base path. If this is not configured, the current working directory is used.

Parameters
parts (*sequence of str or Path*) – Joined to the base path using `Path.joinpath()`.

See also:

Choose locations for data

`message_ix_models.util.`**make_io**(*src*, *dest*, *efficiency*, *on='input'*, ***kwargs*)

Return input and output data frames for a 1-to-1 technology.

Parameters

- **src** (*tuple (str, str, str)*) – Input commodity, level, unit.

- **dest** (*tuple (str, str, str)*) – Output commodity, level, unit.
- **efficiency** (*float*) – Conversion efficiency.
- **on** (*'input' or 'output'*) – If 'input', *efficiency* applies to the input, and the output, thus the activity level of the technology, is in dest[2] units. If 'output', the opposite.
- **kwargs** – Passed to `make_df()`.

Returns
Keys are 'input' and 'output'; values are data frames.

Return type
dict (str -> pd.DataFrame)

`message_ix_models.util.`**`maybe_query`**(*series: Series*, *query: Optional[str]*) → Series

Apply `pandas.Series.query()` if the *query* arg is not `None`.

`query()` is not chainable (pandas-dev/pandas#37941). Use this function with `pandas.Series.pipe()`, passing an argument that may be `None`, to have a chainable query operation that can be a no-op.

`message_ix_models.util.`**`package_data_path`**(**parts*) → Path

Construct a path to a file under `message_ix_models/data/`.

Use this function to access data packaged and installed with `message_ix_models`.

Parameters
parts (*sequence of str or Path*) – Joined to the base path using `Path.joinpath()`.

See also:

Choose locations for data

`message_ix_models.util.`**`private_data_path`**(**parts*) → Path

Construct a path to a file under `data/` in `message_data`.

Use this function to access non-public (e.g. embargoed or proprietary) data stored in the `message_data` repository.

Parameters
parts (*sequence of str or Path*) – Joined to the base path using `Path.joinpath()`.

See also:

Choose locations for data

`message_ix_models.util.`**`series_of_pint_quantity`**(**args*, ***kwargs*) → Series

Suppress a spurious warning.

Creating a `pandas.Series` with a list of `pint.Quantity` triggers a warning "The unit of the quantity is stripped when downcasting to ndarray," even though the entire object is being stored and the unit is **not** stripped. This function suppresses this warning.

`message_ix_models.util.cache.`**`SKIP_CACHE = False`**

Controls whether cached data is returned for functions decorated with *cached()*. Set to `True` to force reload.

13.1 util.click

Command-line utilities.

These are used for building CLIs using `click`.

message_ix_models.util.click.**PARAMS = {'dest': <Option dest>, 'dry_run': <Option dry_run>, 'force': <Option force>, 'nodes': <Option nodes>, 'output_model': <Option output_model>, 'platform_dest': <Option platform_dest>, 'policy_path': <Option policy_path>, 'quiet': <Option quiet>, 'regions': <Option regions>, 'rep_out_path': <Option rep_out_path>, 'rep_template': <Option rep_template>, 'run_reporting_only': <Option run_reporting_only>, 'ssp': <Argument ssp>, 'verbose': <Option verbose>, 'years': <Option years>}**

Common command-line parameters (arguments and options). See *common_params()*.

message_ix_models.util.click.**common_params**(*param_names: str*)

Decorate a click.command with common parameters *param_names*.

param_names must be a space-separated string of names appearing in *PARAMS*, e.g. `"ssp force output_model"`. The decorated function receives keyword arguments with these names:

```
@click.command()
@common_params("ssp force output_model")
def mycmd(ssp, force, output_model)
    # ...
```

message_ix_models.util.click.**default_path_cb**(**default_parts*)

Return a callback function for click.Option handling.

If no option value is given, the callback uses `Context.get_local_path()` and *default_parts* to provide a path that is relative to local data directory, e.g. the current working directory (see *Data, metadata, and configuration*).

message_ix_models.util.click.**store_context**(*context: Union[Context,* Context*]*, *param*, *value*)

Callback that simply stores a value on the `Context` object.

Use this for parameters that are not used directly in a @click.command() function, but need to be carried by the Context for later use.

13.2 util.context

Context and settings for `message_ix_models` code.

class message_ix_models.util.context.**Context**(**args*, ***kwargs*)

Context and settings for `message_ix_models` code.

Context is a subclass of `dict`, so common methods like `copy()` and `setdefault()` may be used to handle settings. To be forgiving, it also provides attribute access; `context.foo` is equivalent to `context["foo"]`.

Context provides additional methods to do common tasks that depend on configurable settings:

clone_to_dest([create])	Return a scenario based on the `--dest` command-line option.
`close_db`()	
delete()	Hide the current Context from future *get_instance()* calls.
get_cache_path(*parts)	Return a path to a local cache file.
get_local_path(*parts[, suffix])	Return a path under `local_data`.
get_platform([reload])	Return a `ixmp.Platform` from `platform_info`.
get_scenario()	Return a `message_ix.Scenario` from `scenario_info`.
handle_cli_args([url, platform, model_name, ...])	Handle command-line arguments.
only()	Return the only *Context* instance.
use_defaults(settings)	Update from *settings*.

clone_to_dest(*create=True*) → Scenario

Return a scenario based on the `--dest` command-line option.

Parameters

create (*bool, optional*) – If `True` (the default) and the base scenario does not exist, a bare RES scenario is created. Otherwise, an exception is raised.

Returns

To prevent the scenario from being garbage collected, keep a reference to its Platform:

Return type

Scenario

See also:

`create_res`

To use this method, either decorate a command with `common_params()`:

```
from message_data.tools.cli import common_params

@click.command()
@common_params("dest")
@click.pass_obj
def foo(context, dest):
    scenario, mp = context.clone_to_dest()
```

or, store the settings `dest_scenario` and `dest_platform` on *context*:

```
c = Context.get_instance()

c.dest_scenario = dict(model="foo model", scenario="foo scenario")
scenario_mp = context.clone_to_dest()
```

The resulting scenario has the indicated model- and scenario names.

If `--url` (or `--platform`, `--model`, `--scenario`, and optionally `--version`) are given, the identified scenario is used as a 'base' scenario, and is cloned. If `--url`/`--platform` and `--dest` refer to different Platform instances, then this is a two-platform clone.

If no base scenario can be loaded, *bare.create_res()* is called to generate a base scenario.

delete()

Hide the current Context from future *get_instance()* calls.

get_cache_path(**parts*) → Path

Return a path to a local cache file.

classmethod get_instance(*index=0*) → *Context*

Return a Context instance; by default, the first created.

Parameters
index (*int, optional*) – Index of the Context instance to return, e.g. `-1` for the most recently created.

get_local_path(**parts*, *suffix=None*)

Return a path under `local_data`.

get_platform(*reload=False*) → Platform

Return a `ixmp.Platform` from `platform_info`.

When used through the CLI, `platform_info` is a 'base' platform as indicated by the –url or –platform options.

If a Platform has previously been instantiated with *get_platform()*, the same object is returned unless *reload=True*.

get_scenario() → Scenario

Return a `message_ix.Scenario` from `scenario_info`.

When used through the CLI, `scenario_info` is a 'base' scenario for an operation, indicated by the `--url` or `--platform/--model/--scenario` options.

handle_cli_args(*url=None*, *platform=None*, *model_name=None*, *scenario_name=None*, *version=None*, *local_data=None*, *verbose=False*, *_store_as=('platform_info', 'scenario_info')*)

Handle command-line arguments.

May update the `data_path`, `platform_info`, `scenario_info`, and/or `url` settings.

classmethod only() → *Context*

Return the only *Context* instance.

Raises
IndexError – If there is more than one instance.

set_scenario(*scenario: Scenario*) → None

Update `scenario_info` to match an existing *scenario*.

use_defaults(*settings*)

Update from *settings*.

13.3 util.importlib

Load model and project code from message_data.

class message_ix_models.util.importlib.**MessageDataFinder**

Load model and project code from message_data.

13.4 util._logging

Logging utilities.

class message_ix_models.util._logging.**Formatter**(*colorama*)

Formatter for log records.

Parameters

colorama (*module*) – If provided, colorama is used to colour log messages printed to stdout.

format(*record*)

Format *record*.

Records are formatted like:

```
model.transport.data.add_par_data  220 rows in 'input'
...add_par_data:  further messages
```

…with the calling function name (e.g. 'add_par_data') coloured for legibility on first occurrence, then dimmed when repeated.

message_ix_models.util._logging.**make_formatter**()

Return a *Formatter* instance for the message_ix_models logger.

See also:

setup

message_ix_models.util._logging.**setup**(*level='NOTSET'*, *console=True*)

Initialize logging.

Parameters

- **level** (*str, optional*) – Log level for message_ix_models and message_data.
- **console** (*bool, optional*) – If True, print all messages to console using a *Formatter*.

message_ix_models.util._logging.**silence_log**()

Context manager to temporarily silence log output.

Examples

```
>>> with silence_log():
>>>     log.warning("This message is not recorded.")
```

13.5 util.node

Utilities for nodes.

message_ix_models.util.node.**NODE_DIMS = ['n', 'node', 'node_loc', 'node_origin', 'node_dest', 'node_rel', 'node_share']**

Names of dimensions indexed by 'node'.

Todo: to be robust to changes in `message_ix`, read these names from that package.

message_ix_models.util.node.**R11_R12 = (('R11_AFR', 'R12_AFR'), ('R11_CPA', 'R12_CHN'), ('R11_EEU', 'R12_EEU'), ('R11_FSU', 'R12_FSU'), ('R11_LAM', 'R12_LAM'), ('R11_MEA', 'R12_MEA'), ('R11_NAM', 'R12_NAM'), ('R11_PAO', 'R12_PAO'), ('R11_PAS', 'R12_PAS'), ('R11_CPA', 'R12_RCPA'), ('R11_SAS', 'R12_SAS'), ('R11_WEU', 'R12_WEU'))**

Mapping from R11 to R12 node IDs.

message_ix_models.util.node.**R11_R14 = (('R11_AFR', 'R14_AFR'), ('R11_FSU', 'R14_CAS'), ('R11_CPA', 'R14_CPA'), ('R11_EEU', 'R14_EEU'), ('R11_LAM', 'R14_LAM'), ('R11_MEA', 'R14_MEA'), ('R11_NAM', 'R14_NAM'), ('R11_PAO', 'R14_PAO'), ('R11_PAS', 'R14_PAS'), ('R11_FSU', 'R14_RUS'), ('R11_SAS', 'R14_SAS'), ('R11_FSU', 'R14_SCS'), ('R11_FSU', 'R14_UBM'), ('R11_WEU', 'R14_WEU'))**

Mapping from R11 to R14 node IDs.

message_ix_models.util.node.**adapt_R11_R12 = <message_ix_models.util.common.MappingAdapter object>**

Adapt data from the R11 to the R14 node list.

The data is adapted using the mappings in *R11_R12* for each of the dimensions in *NODE_DIMS*.

message_ix_models.util.node.**adapt_R11_R14 = <message_ix_models.util.common.MappingAdapter object>**

Adapt data from the R11 to the R14 node list.

The data is adapted using the mappings in *R11_R14* for each of the dimensions in *NODE_DIMS*.

message_ix_models.util.node.**identify_nodes**(*scenario: Scenario*) → str

Return the ID of a node codelist given the contents of *scenario*.

Returns
The ID of the *Node code lists* containing the regions of *scenario*.

Return type
str

Raises
ValueError – if no codelist can be identified, or the nodes in the scenario do not match the children of the "World" node in the codelist.

13.6 util.scenarioinfo

ScenarioInfo class.

class message_ix_models.util.scenarioinfo.**ScenarioInfo**(*scenario=None*)

Information about a Scenario object.

Code that prepares data for a target Scenario can accept a ScenarioInfo instance. This avoids the need to load a Scenario, which can be slow under some conditions.

ScenarioInfo objects can also be used (e.g. by *apply_spec()*) to describe the contents of a Scenario *before* it is created.

ScenarioInfo objects have the following convenience attributes:

set	Elements of ixmp/message_ix sets.
io_units(technology, commodity[, level])	Return units for the MESSAGE input or output parameter.
is_message_macro	True if a MESSAGE-MACRO scenario.
N	Elements of the set 'node'.
units_for(set_name, id)	Return the units associated with code *id* in MESSAGE set *set_name*.
Y	Elements of the set 'year' that are >= the first model year.
y0	First model year, if set, else Y[0].
yv_ya	pandas.DataFrame with valid year_vtg, year_act pairs.

Parameters

scenario (*message_ix.Scenario*) – If given, *set* is initialized from this existing scenario.

See also:

Spec

property N

Elements of the set 'node'.

property Y

Elements of the set 'year' that are >= the first model year.

io_units(*technology: str*, *commodity: str*, *level: Optional[str] = None*) → Unit

Return units for the MESSAGE input or output parameter.

These are implicitly determined as the ratio of:

- The units for the origin (for input) or destination *commodity*, per *units_for()*.
- The units of activity for the *technology*.

Parameters

level (*str*) – Placeholder for future functionality, i.e. to use different units per (commodity, level). Currently ignored. If given, a debug message is logged.

is_message_macro: bool = False

True if a MESSAGE-MACRO scenario.

par: `Dict[str, DataFrame] = {}`

Elements of `ixmp`/`message_ix` parameters.

set: `Dict[str, List] = {}`

Elements of `ixmp`/`message_ix` sets.

units_for(*set_name: str*, *id: str*) → Unit

Return the units associated with code *id* in MESSAGE set *set_name*.

`ixmp` (or the sole `JDBCBackend`, as of v3.5.0) does not handle unit information for variables and equations (unlike parameter values), such as MESSAGE decision variables `ACT`, `CAP`, etc. In `message_ix_models` and `message_data`, the following conventions are (generally) followed:

- The units of `ACT` and others are consistent for each `technology`.
- The units of `COMMODITY_BALANCE`, `STOCK`, `commodity_stock`, etc. are consistent for each `commodity`.

Thus, codes for elements of these sets (e.g. *Commodities (commodity.yaml)*) can be used to carry the standard units for the corresponding quantities. *units_for()* retrieves these units, for use in model-building and reporting.

Todo: Expand this discussion and transfer to the `message_ix` docs.

See also:

io_units

update(*other:* ScenarioInfo)

Update with the set elements of *other*.

y0: `int = -1`

First model year, if set, else `Y[0]`.

year_from_codes(*codes: List[Code]*)

Update using a list of *codes*.

The following are updated:

- *set* `year`
- *set* `cat_year`, with the first model year.
- *par* `duration_period`

Any existing values are discarded.

After this, the attributes *y0* and *Y* give the first model year and model years, respectively.

Examples

Get a particular code list, create a ScenarioInfo instance, and update using the codes:

```
>>> years = get_codes("year/A")
>>> info = ScenarioInfo()
>>> info.year_from_codes(years)
```

Use populated values:

```
>>> info.y0
2020
>>> info.Y[:3]
[2020, 2030, 2040]
>>> info.Y[-3:]
[2090, 2100, 2110]
```

property yv_ya

pandas.DataFrame with valid `year_vtg`, `year_act` pairs.

class message_ix_models.util.scenarioinfo.**Spec**(*add: ~message_ix_models.util.scenarioinfo.ScenarioInfo = <factory>*, *remove: ~message_ix_models.util.scenarioinfo.ScenarioInfo = <factory>*, *require: ~message_ix_models.util.scenarioinfo.ScenarioInfo = <factory>*)

A specification for the structure of a model or variant.

A Spec collects 3 *ScenarioInfo* instances at the attributes *add*, *remove*, and *require*. This is the type that is accepted by *apply_spec()*; *Building models (model.build)* describes how a Spec is used to modify a `Scenario`. A Spec may also be used to express information about the target structure of data to be prepared; like *ScenarioInfo*, this can happen before the target `Scenario` exists.

Spec also provides:

- Dictionary-style access, e.g. `s["add"]` is equivalent to `s.add`.
- Error checking; setting keys other than add/remove/require results in an error.
- *merge()*, a helper method.

add: *ScenarioInfo*

Structure to be added to a base scenario.

static merge(*a:* Spec, *b:* Spec) → *Spec*

Merge Specs *a* and *b* together.

Returns a new Spec where each member is a union of the respective members of *a* and *b*.

remove: *ScenarioInfo*

Structure to be removed from a base scenario.

require: *ScenarioInfo*

Structure that must be present in a base scenario.

CHAPTER

FOURTEEN

TEST UTILITIES AND FIXTURES (`testing`)

Fixtures:

mix_models_cli(request, session_context, tmp_env)	A *CliRunner* object that invokes the **mix-models** CLI.
session_context(pytestconfig, tmp_env)	A *Context* connected to a temporary, in-memory database.
test_context(request, session_context)	A copy of *session_context()* scoped to one test function.
user_context(request)	Context which can access user's configuration, e.g.

Marks:

NIE	Shorthand for marking a parametrized test case that is expected to fail because it is not implemented.
not_ci([reason, action])	Mark a test as xfail or skipif if on CI infrastructure.

Others:

EXPORT_OMIT	Items with names that match (partially or fully) these names are omitted by *export_test_data()*.
CliRunner([charset, env, echo_stdin, mix_stderr])	Subclass of `click.testing.CliRunner` with extra features.
bare_res(request, context[, solved])	Return or create a Scenario containing the bare RES, for use in testing.
export_test_data(context)	Export a subset of data from a scenario, for use in tests.
pytest_addoption(parser)	Add two command-line options to pytest:

class message_ix_models.testing.**CliRunner**(*charset: str = 'utf-8'*, *env: Optional[Mapping[str, Optional[str]]] = None*, *echo_stdin: bool = False*, *mix_stderr: bool = True*)

Subclass of `click.testing.CliRunner` with extra features.

assert_exit_0(**args*, ***kwargs*)

Assert a result has exit_code 0, or print its traceback.

If any *args* or *kwargs* are given, *invoke()* is first called. Otherwise, the result from the last call of *invoke()* is used.

Raises

AssertionError – if the result exit code is not 0. The exception contains the traceback from within the CLI.

Return type
click.testing.Result

invoke(**args*, ***kwargs*)

Invoke the **mix-models** CLI.

message_ix_models.testing.**EXPORT_OMIT = ['aeei', 'cost_MESSAGE', 'demand_MESSAGE', 'demand', 'depr', 'esub', 'gdp_calibrate', 'grow', 'historical_gdp', 'kgdp', 'kpvs', 'lakl', 'land', 'lotol', 'mapping_macro_sector', 'MERtoPPP', 'prfconst', 'price_MESSAGE', 'ref_', 'sector']**

Items with names that match (partially or fully) these names are omitted by *export_test_data()*.

message_ix_models.testing.**NIE = MarkDecorator(mark=Mark(name='xfail', args=(), kwargs={'raises': <class 'NotImplementedError'>}))**

Shorthand for marking a parametrized test case that is expected to fail because it is not implemented.

message_ix_models.testing.**bare_res**(*request*, *context:* Context, *solved: bool = False*) → Scenario

Return or create a Scenario containing the bare RES, for use in testing.

The Scenario has a model name like "MESSAGEix-GLOBIOM [regions] [start]:[duration]:[end]", e.g. "MESSAGEix-GLOBIOM R14 2020:10:2110" (see *bare.name()*) and the scenario name "baseline".

This function should:

- only be called from within test code, i.e. in `message_data.tests`.
- be called once for each test function, so that each test receives a fresh copy of the RES scenario.

Parameters

- **request** (*.Request or None*) – The pytest `request` fixture. If provided the pytest test node name is used for the scenario name of the returned Scenario.
- **context** (*.Context*) – Passed to *testing.bare_res()*.
- **solved** (*bool, optional*) – Return a solved Scenario.

Returns
The scenario is a fresh clone, so can be modified freely without disturbing other tests.

Return type
.Scenario

message_ix_models.testing.**export_test_data**(*context:* Context)

Export a subset of data from a scenario, for use in tests.

The context settings `export_nodes` (default: "R11_AFR" and "R11_CPA") and `export_techs` (default: "coal_ppl") are used to filter the data exported. In addition, any item (set, parameter, variable, or equation) with a name matching *EXPORT_OMIT or* the context setting `export_exclude` is discarded.

The output is stored at `data/tests/`*model name*_*scenario name*_*techs*`.xlsx` in `message_data`.

See also:

Prepare data for testing

message_ix_models.testing.**mix_models_cli**(*request*, *session_context*, *tmp_env*)

A *CliRunner* object that invokes the **mix-models** CLI.

message_ix_models.testing.**not_ci**(*reason=None*, *action='skip'*)

Mark a test as xfail or skipif if on CI infrastructure.

Checks the GITHUB_ACTIONS environment variable; returns a pytest mark.

message_ix_models.testing.**pytest_addoption**(*parser*)

Add two command-line options to pytest:

--local-cache
: Use existing, local cache files in tests. This option can speed up tests that *use* the results of slow data loading/parsing. However, if cached values are not up to date with the current code, unexpected failure may occur.

--jvmargs
: Additional arguments to give for the Java Virtual Machine used by ixmp's JDBCBackend. Used by *session_context()*.

message_ix_models.testing.**session_context**(*pytestconfig*, *tmp_env*)

A *Context* connected to a temporary, in-memory database.

This Context is suitable for modifying and running test code that does not affect the user/developer's filesystem and configured ixmp databases.

Uses the tmp_env() fixture from ixmp. This fixture also sets:

- Context.cache_path, depending on whether the **--local-cache** CLI option was given:
  - If not given: pytest's standard cache directory.
  - If given: the /cache/ directory under the user's "message local data" directory.
- the "message local data" config key to a temporary directory /data/ under the pytest tmp_path directory.

message_ix_models.testing.**test_context**(*request*, *session_context*)

A copy of *session_context()* scoped to one test function.

message_ix_models.testing.**user_context**(*request*)

Context which can access user's configuration, e.g. platform names.

CHAPTER

FIFTEEN

MULTI-SCENARIO WORKFLOWS (`WORKFLOW`)

15.1 Concept & design

Research with MESSAGEix models often involves multiple scenarios that are related to one another or derived from one another by certain modifications. Together, the solutions/reported information from these scenarios provide the output data used in research products, e.g. a plot comparing total emissions in a policy scenario to a reference scenario.

`model.build` provides tools to build models or scenarios based on (possibly empty) base scenarios; and `tools` provides tools for manipulating scenarios or model input data (parameters). The `Workflow` API provided in this module allows researchers to use these pieces and atomic, reusable functions to define arbitrarily complex workflows involving many, related scenarios; and then to solve, report, or otherwise operate on those scenarios.

The generic pattern for workflows is:

- Each scenario has zero or 1 (or more?) base/precursor/antecedent scenarios. These must exist before the target scenario can be created.
- A workflow 'step' includes:
  - The precursor scenario is cloned to the target scenario name.
  - Modifications are applied. These can be modifications to structure or to data. For example:
    - Setting up a model variant, e.g. adding the MESSAGEix-Materials structure to a base MESSAGEix-GLOBIOM model.
    - Changing policy variables via constraint parameters.
    - Any other possible modification.
  - The target scenario is optionally solved.
  - The target scenario is optionally reported.
- A workflow can consist of any number of scenarios and steps.
- The same precursor scenario can be used as the basis for multiple target scenarios.

- A workflow is 'run' starting with the earliest precursor scenario, ending with 1-to-many target scenarios.

The implementation is based on the observation that these form a graph (DAG) of nodes (scenarios) and edges (steps), in the same way that `message_ix.reporting` calculations do; and so the `dask` DAG features (via `genno`) can be used to organize the workflow.

15.2 Usage

15.2.1 General

Define a workflow using ordinary Python functions, each handling the modifications/manipulations in a single, atomic workflow step. These functions they **must**:

- Accept 1 argument: either the precursor scenario, or `None`.
- Return either:
  - a `Scenario` object: required if the argument is `None`.
  - `None`. In this case, the modifications are reflected in the `Scenario` given as an argument.

The functions **may** call any other code, and can be from one to many lines; they **should** be reusable, i.e. respond in simple and obvious ways to a few clearly-defined arguments.

```
def base_scenario(arg=None) -> Scenario:
    """Generate a base scenario."""
    return testing.bare_res(request, test_context, solved=False)

def changes_a(s: Scenario) -> None:
    """Change a scenario by modifying structure data, but not data."""
    with s.transact():
        s.add_set("technology", "test_tech")

    # Here, invoke other code to further modify `s`

def changes_b(s: Scenario, value=100.0) -> None:
    """Change a scenario by modifying parameter data, but not structure.

    This function takes an extra argument, `values`, so functools.partial()
    can be used to supply different values when it is used in different
    workflow steps. See below.
    """
    with s.transact():
        s.add_par(
            "technical_lifetime",
            make_df(
                "technical_lifetime",
                node_loc=s.set("node")[0],
                year_vtg=s.set("year")[0],
                technology="test_tech",
                value=100.0,
                unit="y",
            ),
        )
```

(continues on next page)

(continued from previous page)

```
    # Here, invoke other code to further modify `s`
```

With the steps defined, the workflow is composed using a *Workflow* instance. Call *Workflow.add()* to define each target model with its precursor and the function that will create the former from the latter:

```
from message_ix_models import Context, Workflow

# Create the workflow
ctx = Context.get_instance()
wf = Workflow(ctx)

# "Model/base" is created from nothing by calling base_scenario()
wf.add("Model/base", None, base_scenario)

# "Model/A" is created from "Model/base" by calling changes_a()
wf.add("Model/A", "Model/base", changes_a)

# "Model/B1" is created from "Model/A" by calling changes_b() with the
# default value
wf.add("Model/B1", "Model/A", changes_b)

# "Model/B2" is similar, but uses a different value
wf.add("Model/B2", "Model/A", partial(changes_b, value=200.0))
```

Finally, the workflow is triggered using *Workflow.run()*, giving either one scenario identifier or a list of identifiers. The indicated scenarios are created and solved; if this requires any precursor scenarios, those are first created and solved, etc. as required. Other, unrelated scenarios/steps are not created.

```
s1, s2 = wf.run(["Model/B1", "Model/B2"])
```

15.2.2 Common use cases

Todo: Expand with discussion of workflow patterns common in research projects using MESSAGEix, e.g.:

- Run the same scenario with multiple emissions budgets.

15.3 API reference

Tools for modeling workflows.

class message_ix_models.workflow.**Workflow**(*context:* Context, *solve=True*)

Workflow containing multiple `Scenarios`.

Parameters

- **context** (*.Context*) – Context object with settings common to the entire workflow.
- **solve** (*bool, optional*) – Passed to every *WorkflowStep* created using *add()*.

add(*name: str*, *base: Optional[str]*, *callback: Callable*)

Add a step to the workflow.

Parameters

- **name** (*str*) – "`model name/scenario name`" for the `Scenario` produced by the step.
- **base** (*str or None*) – Base scenario, if any.
- **callback** (*Callable*) – Function to be executed to modify the base into the target Scenario.

run(*scenarios: Union[str, List[str]]*)

Run the workflow to generate one or more scenarios.

Parameters

scenarios (*str or list of str*) – Identifier(s) of scenario(s) to generate.

class message_ix_models.workflow.**WorkflowStep**(*name: str*, *callback: Callable*, *solve=True*)

Single step in a multi-scenario workflow.

Parameters

- **name** (*str*) – "`model name/scenario name`" for the `Scenario` produced by the step.
- **callback** (*Callable*) – Function to be executed to modify the base into the target Scenario.
- **solve** (*bool, optional*) – If `True`, the created scenario is solved when it is created.
- **report** (*bool, optional*) – If `True`, the created scenario is reported after it is created and maybe (according to *solve*) solved.

CHAPTER

SIXTEEN

NODE CODE LISTS

The codes in these lists denote **regions** and **countries**.

When loaded using *get_codes()*, the `Code.child` attribute is a list of child codes. See the function documentation for how to retrieve these.

See also:

`adapt_R11_R12()`, `adapt_R11_R14()`, *identify_nodes()*.

16.1 Models with global scope

16.1.1 32-region aggregation (R32)

```
World:
  name: World
  description: |-
    Region code list for the SSP 32-region aggregation.

    Source: https://tntcat.iiasa.ac.at/SspDb/dsd?Action=htmlpage&page=about

R32ANUZ:
  parent: World
  name: Australia & New Zealand
  description: This region includes Australia and New Zealand.
```

(continues on next page)

(continued from previous page)

```
  child: [AUS, NZL]

R32BRA:
  parent: World
  name: Brazil
  child: [BRA]

R32CAN:
  parent: World
  name: Canada
  child: [CAN]

R32CAS:
  parent: World
  name: Central Asia
  description: This region includes the countries of Central Asia.
  child: [ARM, AZE, GEO, KAZ, KGZ, TJK, TKM, UZB]

R32CHN:
  parent: World
  name: China excl. Taiwan
  description: China (Mainland, Hongkong, Macao; excl. Taiwan).
  child: [CHN, HKG, MAC]

R32EEU:
  parent: World
  name: Eastern Europe
  description: |-
    Eastern Europe (excl. former Soviet Union and EU member states).

    The source page describes “the former Yugoslav Republic of Macedonia”; this entity␣
↪was renamed in 2018 to “North Macedonia”.
  child: [ALB, BIH, HRV, MKD, MNE, SRB]

R32EEU-FSU:
  parent: World
  name: Former Soviet Union in Eastern Europe
  description: Eastern Europe, former Soviet Union (excl. Russia and EU members).
  child: [BLR, MDA, UKR]

R32EFTA:
  parent: World
  name: European Free Trade Association
  description: |-
    This region includes Iceland, Norway, Switzerland.

    The source omits Liechtenstein, but it is included as a child.
  child: [ISL, LIE, NOR, CHE]

R32EU12-H:
  parent: World
  name: EU member states new in 2004, high income
```

(continues on next page)

(continued from previous page)

```
  description: New EU member states that joined as of 2004 - high income.
  child: [CYP, CZE, EST, HUN, MLT, POL, SVK, SVN]

R32EU12-M:
  parent: World
  name: EU member states new in 2004, middle income
  description: New EU member states that joined as of 2004 - medium income.
  child: [BGR, LTU, LVA, ROU]

R32EU15:
  parent: World
  name: EU member states pre-2004
  description: This region includes European Union member states that joined prior to␣
↪2004.
  child: [AUT, BEL, DEU, DNK, ESP, FIN, FRA, GBR, GRC, IRL, ITA, LUX, NLD, PRT, SWE]

R32IDN:
  parent: World
  name: Indonesia
  child: [IDN]

R32IND:
  parent: World
  name: India
  child: [IND]

R32JPN:
  parent: World
  name: Japan
  child: [JPN]

R32KOR:
  parent: World
  name: Republic of Korea
  child: [KOR]

R32LAM-L:
  parent: World
  name: Latin America, low income
  description: This region includes the countries of Latin America (excl. Brazil,␣
↪Mexico) - low income.
  child: [BLZ, GTM, HND, HTI, NIC]

R32LAM-M:
  parent: World
  name: Latin America, middle & high income
  description: |-
    This region includes the countries of Latin America (excl. Brazil, Mexico) - medium␣
↪and high income.

    The source includes “Netherlands Antilles” which has a provisional ISO 3166-2 alpha-
↪3 code (ANT), but is not a country. It was dissolved in 2010 into BES, CUW and SXM,␣
```

(continues on next page)

(continued from previous page)

```
↪also included.
  child: [ABW, AIA, ANT, BES, BHS, BMU, BOL, BRB, CHL, COL, CRI, CUB, CUW, DMA, DOM, ECU,
↪ GLP, GRD, GUF, GUY, JAM, KNA, LCA, PAN, PER, PRY, MTQ, SLV, SUR, SXM, TTO, URY, VCT,␣
↪VEN]

R32MEA-H:
  parent: World
  name: Middle East & Asia, high income
  description: This region includes the countries of Middle East Asia - high income.
  child: [ARE, BHR, ISR, KWT, OMN, QAT, SAU]

R32MEA-M:
  parent: World
  name: Middle East & Asia, low & middle income
  description: This region includes the countries of Middle East Asia - low and medium␣
↪income.
  child: [IRN, IRQ, JOR, LBN, PSE, SYR, YEM]

R32MEX:
  parent: World
  name: Mexico
  child: [MEX]

R32NAF:
  parent: World
  name: North Africa
  description: This region includes the countries of North Africa.
  child: [DZA, EGY, ESH, LBY, MAR, TUN]

R32OAS-CPA:
  parent: World
  name: Other Asia
  description: This region includes the countries of Other Asia - former Centrally␣
↪Planned Asia.
  child: [KHM, LAO, MNG, VNM]

R32OAS-L:
  parent: World
  name: Other Asia, low income
  description: This region includes the countries of Other Asia - low income.
  child: [BGD, FJI, FSM, MMR, NPL, PHL, PNG, PRK, SLB, TLS, TON, VUT, WSM]

R32OAS-M:
  parent: World
  name: Other Asia, middle & high income
  description: This region includes the countries of Other Asia - medium and high income.
  child: [BRN, BTN, GUM, LKA, MDV, MYS, NCL, PYF, SGP, THA]

R32PAK:
  parent: World
  name: Pakistan & Afghanistan
  description: This region includes Pakistan and Afghanistan.
```

(continues on next page)

(continued from previous page)

```
  child: [AFG, PAK]

R32RUS:
  parent: World
  name: Russian Federation
  child: [RUS]

R32SAF:
  parent: World
  name: South Africa
  child: [ZAF]

R32SSA-L:
  parent: World
  name: Sub-Saharan Africa, low income
  description: This region includes the countries of Subsahara Africa (excl. South␣
↪Africa) - low income.
  child: [BDI, BEN, BFA, CAF, CIV, CMR, COD, COG, COM, CPV, DJI, ERI, ETH, GHA, GIN, GMB,
↪ GNB, KEN, LBR, LSO, MDG, MLI, MOZ, MRT, MWI, NER, NGA, RWA, SDN, SEN, SLE, SOM, SSD,␣
↪STP, SWZ, TCD, TGO, TZA, UGA, ZMB, ZWE]

R32SSA-M:
  parent: World
  name: Sub-Saharan Africa, middle & high income
  description: This region includes the countries of Subsahara Africa (excl. South␣
↪Africa) - medium and high income.
  child: [AGO, BWA, GAB, GNQ, MUS, MYT, NAM, REU, SYC]

R32TUR:
  parent: World
  name: Turkey
  child: [TUR]

R32TWN:
  parent: World
  name: Taiwan
  child: [TWN]

R32USA:
  parent: World
  name: United States of America
  description: United States of America.
  child: [PRI, USA, VIR]
```

16.1.2 14-region aggregation (R14)

```
# Region code list
#
# - See message_data.tools.regions.
# - The ISO 3166-1 alpha-3 codes are not defined in this file, but loaded from
#   a copy of the ISO database, e.g. in pycountry.
# - Among others, there are no assignments for:
#   - ATA Antarctica
#   - IOT British Indian Ocean Territory
#   - SGS South Georgia

World:
  name: World
  description: R14 regions

R14_AFR:
  parent: World
  name: Sub-Saharan Africa
  child: [AGO, BDI, BEN, BFA, BWA, CAF, CIV, CMR, COD, COG, COM, CPV, DJI, ERI, ETH, GAB,
↪ GHA, GIN, GMB, GNB, GNQ, KEN, LBR, LSO, MDG, MLI, MOZ, MRT, MUS, MWI, MYT, NAM, NER,␣
↪NGA, REU, RWA, SDN, SEN, SHN, SLE, SOM, STP, SWZ, SYC, TCD, TGO, TZA, UGA, ZAF, ZMB,␣
↪ZWE]

R14_CAS:
  parent: World
  name: Central Asia
  child: [KAZ, KGZ, TJK, TKM, UZB]

R14_CPA:
  parent: World
  name: Centrally Planned Asia
  child: [CHN, KHM, LAO, MAC, MNG, PRK, TWN, VNM]

R14_EEU:
  parent: World
  name: Central and Eastern Europe
  description: >-
    Serbia and Montenegro (SCG) and Yugoslavia (YUG) still included in this list,
    even though their ISO 3166-1 codes were deleted in 2006 and 2003, respectively.
  child: [ALB, BGR, BIH, CZE, EST, HRV, HUN, LTU, LVA, MKD, MNE, POL, ROU, SCG, SRB, SVK,
↪ SVN, YUG]

R14_LAM:
  parent: World
  name: Latin America and The Caribbean
  description: >-
    The source includes “Netherlands Antilles” which has a provisional ISO 3166-2 alpha-
↪3 code (ANT),
    but is not a country. It was dissolved in 2010 into BES, CUW and SXM, also included.
  child: [ABW, AIA, ANT, ARG, ATG, BES, BHS, BLZ, BMU, BOL, BRA, BRB, CHL, COL, CRI, CUB,
↪ CUW, CYM, DMA, DOM, ECU, FLK, GLP, GRD, GTM, GUF, GUY, HND, HTI, JAM, LCA, MEX, MSR,␣
↪MTQ, NIC, PAN, PER, PRI, PRY, SLV, SUR, SXM, TCA, TTO, URY, VCT, VEN, VGB]
```

(continues on next page)

(continued from previous page)

```
R14_MEA:
  parent: World
  name: Middle East and North Africa
  child: [ARE, BHR, DZA, EGY, ESH, IRN, IRQ, ISR, JOR, KWT, LBN, LBY, MAR, OMN, PSE, QAT,
↪ SAU, SDN, SSD, SYR, TUN, YEM]

R14_NAM:
  parent: World
  name: North America
  child: [CAN, GUM, SPM, USA]

R14_PAO:
  parent: World
  name: Pacific OECD
  child: [AUS, JPN, NZL]

R14_PAS:
  parent: World
  name: Other Pacific Asia
  child: [ASM, BRN, CCK, COK, CXR, FJI, FSM, IDN, KIR, KOR, MHL, MMR, MNP, MYS, NCL, NFK,
↪ NIU, NRU, PCN, PHL, PLW, PNG, PYF, SGP, SLB, THA, TKL, TLS, TON, TUV, VUT, WLF, WSM]

R14_RUS:
  parent: World
  name: Russia
  child: [RUS]

R14_SAS:
  parent: World
  name: South Asia
  child: [AFG, BGD, BTN, IND, LKA, MDV, NPL, PAK]

R14_SCS:
  parent: World
  name: Caspian States
  child: [ARM, AZE, GEO]

R14_UBM:
  parent: World
  name: Ukraine, Belarus, and Moldova
  child: [BLR, MDA, UKR]

R14_WEU:
  parent: World
  name: Western Europe
  child: [AND, AUT, BEL, CHE, CYP, DEU, DNK, ESP, FIN, FRA, FRO, GBR, GIB, GRC, GRL, IMN,
↪ IRL, ISL, ITA, LIE, LUX, MCO, MLT, NLD, NOR, PRT, SJM, SMR, SWE, TUR, VAT]
```

16.1.3 11-region aggregation (R11)

```
# Region code list
#
# - See message_data.tools.regions.
# - The ISO 3166-1 alpha-3 codes are not defined in this file, but loaded from
#   a copy of the ISO database, e.g. in pycountry.
# - Among others, there are no assignments for:
#   - ATA Antarctica
#   - IOT British Indian Ocean Territory
#   - SGS South Georgia

World:
  name: World
  description: R11 regions

R11_AFR:
  parent: World
  name: Sub-Saharan Africa
  child: [AGO, BDI, BEN, BFA, BWA, CAF, CIV, CMR, COD, COG, COM, CPV, DJI, ERI, ETH, GAB,
↪ GHA, GIN, GMB, GNB, GNQ, KEN, LBR, LSO, MDG, MLI, MOZ, MRT, MUS, MWI, MYT, NAM, NER,␣
↪NGA, REU, RWA, SEN, SHN, SLE, SOM, STP, SWZ, SYC, TCD, TGO, TZA, UGA, ZAF, ZMB, ZWE]

R11_CPA:
  parent: World
  name: Centrally Planned Asia
  child: [CHN, HKG, KHM, LAO, MNG, PRK, VNM]

R11_EEU:
  parent: World
  name: Central and Eastern Europe
  description: >-
    Serbia and Montenegro (SCG) and Yugoslavia (YUG) still included in this list,
    even though their ISO 3166-1 codes were deleted in 2006 and 2003, respectively.
  child: [ALB, BGR, BIH, CZE, EST, HRV, HUN, LTU, LVA, MKD, MNE, POL, ROU, SCG, SRB, SVK,
↪ SVN, YUG]

R11_FSU:
  parent: World
  name: Former Soviet Union
  child: [ARM, AZE, BLR, GEO, KAZ, KGZ, MDA, RUS, TJK, TKM, UKR, UZB]

R11_LAM:
  parent: World
  name: Latin America and The Caribbean
  description: >-
    The source includes “Netherlands Antilles” which has a provisional ISO 3166-2 alpha-
↪3 code (ANT),
    but is not a country. It was dissolved in 2010 into BES, CUW and SXM, also included.
  child: [ABW, AIA, ANT, ARG, ATG, BES, BHS, BLZ, BMU, BOL, BRA, BRB, CHL, COL, CRI, CUB,
↪ CUW, CYM, DMA, DOM, ECU, FLK, GLP, GRD, GTM, GUF, GUY, HND, HTI, JAM, KNA, LCA, MEX,␣
↪MSR, MTQ, NIC, PAN, PER, PRY, SLV, SUR, SXM, TCA, TTO, URY, VCT, VEN, VGB]
```

(continues on next page)

(continued from previous page)

```
R11_MEA:
  parent: World
  name: Middle East and North Africa
  child: [ARE, BHR, DZA, EGY, ESH, IRN, IRQ, ISR, JOR, KWT, LBN, LBY, MAR, OMN, PSE, QAT,
↪ SAU, SDN, SSD, SYR, TUN, YEM]

R11_NAM:
  parent: World
  name: North America
  child: [CAN, GUM, PRI, SPM, USA, VIR]

R11_PAO:
  parent: World
  name: Pacific OECD
  child: [AUS, JPN, NZL]

R11_PAS:
  parent: World
  name: Other Pacific Asia
  description: >-
    Trust Territory of the Pacific Islands (PCI) still included in this list,
    but it was dissolved into MHL, FSM, MNP and PLW in 1986.
  child: [ASM, BRN, CCK, COK, CXR, FJI, FSM, IDN, KIR, KOR, MAC, MHL, MMR, MNP, MYS, NCL,
↪ NFK, NIU, NRU, PCI, PCN, PHL, PLW, PNG, PYF, SGP, SLB, THA, TKL, TLS, TON, TUV, TWN,␣
↪VUT, WLF, WSM]

R11_SAS:
  parent: World
  name: South Asia
  child: [AFG, BGD, BTN, IND, LKA, MDV, NPL, PAK]

R11_WEU:
  parent: World
  name: Western Europe
  child: [AND, AUT, BEL, CHE, CYP, DEU, DNK, ESP, FIN, FRA, FRO, GBR, GIB, GRC, GRL, IMN,
↪ IRL, ISL, ITA, LIE, LUX, MCO, MLT, NLD, NOR, PRT, SJM, SMR, SWE, TUR, VAT]
```

16.1.4 12-region aggregation (R12)

```
# Region code list
#
# - See message_data.tools.regions.
# - The ISO 3166-1 alpha-3 codes are not defined in this file, but loaded from
#   a copy of the ISO database, e.g. in pycountry.
# - Among others, there are no assignments for:
#   - ATA Antarctica
#   - IOT British Indian Ocean Territory
#   - SGS South Georgia

World:
```

(continues on next page)

(continued from previous page)

```
  name: World
  description: R12 regions

R12_AFR:
  parent: World
  name: Sub-Saharan Africa
  child: [AGO, BDI, BEN, BFA, BWA, CAF, CIV, CMR, COD, COG, COM, CPV, DJI, ERI, ETH, GAB,
↪ GHA, GIN, GMB, GNB, GNQ, KEN, LBR, LSO, MDG, MLI, MOZ, MRT, MUS, MWI, MYT, NAM, NER,␣
↪NGA, REU, RWA, SEN, SHN, SLE, SOM, STP, SWZ, SYC, TCD, TGO, TZA, UGA, ZAF, ZMB, ZWE]

R12_RCPA:
  parent: World
  name: Rest Centrally Planned Asia
  child: [KHM, LAO, MNG, PRK, VNM]

R12_CHN:
  parent: World
  name: China
  child: [CHN, HKG]

R12_EEU:
  parent: World
  name: Central and Eastern Europe
  description: >-
    Serbia and Montenegro (SCG) and Yugoslavia (YUG) still included in this list,
    even though their ISO 3166-1 codes were deleted in 2006 and 2003, respectively.
  child: [ALB, BGR, BIH, CZE, EST, HRV, HUN, LTU, LVA, MKD, MNE, POL, ROU, SCG, SRB, SVK,
↪ SVN, YUG]

R12_FSU:
  parent: World
  name: Former Soviet Union
  child: [ARM, AZE, BLR, GEO, KAZ, KGZ, MDA, RUS, TJK, TKM, UKR, UZB]

R12_LAM:
  parent: World
  name: Latin America and The Caribbean
  description: >-
    The source includes “Netherlands Antilles” which has a provisional ISO 3166-2 alpha-
↪3 code (ANT),
    but is not a country. It was dissolved in 2010 into BES, CUW and SXM, also included.
  child: [ABW, AIA, ANT, ARG, ATG, BES, BHS, BLZ, BMU, BOL, BRA, BRB, CHL, COL, CRI, CUB,
↪ CUW, CYM, DMA, DOM, ECU, FLK, GLP, GRD, GTM, GUF, GUY, HND, HTI, JAM, KNA, LCA, MEX,␣
↪MSR, MTQ, NIC, PAN, PER, PRY, SLV, SUR, SXM, TCA, TTO, URY, VCT, VEN, VGB]

R12_MEA:
  parent: World
  name: Middle East and North Africa
  child: [ARE, BHR, DZA, EGY, ESH, IRN, IRQ, ISR, JOR, KWT, LBN, LBY, MAR, OMN, PSE, QAT,
↪ SAU, SDN, SSD, SYR, TUN, YEM]

R12_NAM:
```

(continues on next page)

(continued from previous page)

```
  parent: World
  name: North America
  child: [CAN, GUM, PRI, SPM, USA, VIR]

R12_PAO:
  parent: World
  name: Pacific OECD
  child: [AUS, JPN, NZL]

R12_PAS:
  parent: World
  name: Other Pacific Asia
  description: >-
    Trust Territory of the Pacific Islands (PCI) still included in this list,
    but it was dissolved into MHL, FSM, MNP and PLW in 1986.
  child: [ASM, BRN, CCK, COK, CXR, FJI, FSM, IDN, KIR, KOR, MAC, MHL, MMR, MNP, MYS, NCL,
↪ NFK, NIU, NRU, PCI, PCN, PHL, PLW, PNG, PYF, SGP, SLB, THA, TKL, TLS, TON, TUV, TWN,␣
↪VUT, WLF, WSM]

R12_SAS:
  parent: World
  name: South Asia
  child: [AFG, BGD, BTN, IND, LKA, MDV, NPL, PAK]

R12_WEU:
  parent: World
  name: Western Europe
  child: [AND, AUT, BEL, CHE, CYP, DEU, DNK, ESP, FIN, FRA, FRO, GBR, GIB, GRC, GRL, IMN,
↪ IRL, ISL, ITA, LIE, LUX, MCO, MLT, NLD, NOR, PRT, SJM, SMR, SWE, TUR, VAT]
```

16.1.5 5-region aggregation (RCP)

```
# Codes for the "node" dimension of the Representative Concentration Pathways
#
# - See message_data.tools.regions.
# - Since ixmp does not support the "." character in IDs, the names "R5.2ASIA"
#   are transformed to "R5_ASIA" etc. The original code is left in a
#   description.
# - The ISO 3166-1 alpha-3 codes are not defined in this file, but loaded from
#   a copy of the ISO database, e.g. in pycountry.
# - Among others, there are no assignments for:
#   - ATA Antarctica
#   - IOT British Indian Ocean Territory
#   - SGS South Georgia

World:
  name: World
  description: RCP regions

R5_ASIA:
```

(continues on next page)

(continued from previous page)

```
  parent: World
  description: |-
    Officially "R5.2ASIA".

    Trust Territory of the Pacific Islands (PCI) still included in this list, but it was␣
↪dissolved into MHL, FSM, MNP and PLW in 1986.
  child: [AFG, ASM, BGD, BRN, BTN, CCK, CHN, COK, CXR, FJI, FSM, GUM, HKG, IDN, IND, KHM,
↪ KIR, KOR, LAO, LKA, MAC, MDV, MHL, MMR, MNG, MNP, MYS, MYT, NCL, NFK, NIU, NPL, NRU,␣
↪PAK, PCI, PCN, PHL, PLW, PNG, PRK, PYF, SGP, SLB, SYC, THA, TKL, TLS, TON, TUV, TWN,␣
↪VNM, VUT, WSM]

R5_LAM:
  parent: World
  description: |-
    Officially "R5.2LAM".

    The source includes “Netherlands Antilles” which has a provisional ISO 3166-2 alpha-
↪3 code (ANT), but is not a country. It was dissolved in 2010 into BES, CUW and SXM,␣
↪also included.
  child: [ABW, AIA, ANT, ARG, ATG, BES, BHS, BLZ, BMU, BOL, BRA, BRB, CHL, COL, CRI, CUB,
↪ CUW, CYM, DMA, DOM, ECU, GLP, GRD, GTM, GUF, GUY, HND, HTI, JAM, KNA, LCA, MEX, MSR,␣
↪MTQ, NIC, PAN, PER, PRY, SLV, SUR, SXM, TTO, URY, VCT, VEN]

R5_MAF:
  parent: World
  description: Officially "R5.2MAF".
  child: [AGO, ARE, BDI, BEN, BFA, BHR, BWA, CAF, CIV, CMR, COD, COG, COM, CPV, DJI, DZA,
↪ EGY, ERI, ESH, ETH, GAB, GHA, GIN, GMB, GNB, GNQ, IRN, IRQ, ISR, JOR, KEN, KWT, LBN,␣
↪LBR, LBY, LSO, MAR, MDG, MLI, MOZ, MRT, MUS, MWI, NAM, NER, NGA, OMN, PSE, QAT, REU,␣
↪RWA, SAU, SDN, SEN, SLE, SOM, SSD, STP, SWZ, SYR, TCD, TGO, TUN, TZA, UGA, YEM, ZAF,␣
↪ZMB, ZWE]

R5_OECD:
  parent: World
  description: |-
    Officially "R5.2OECD".

    Serbia and Montenegro (SCG) and Yugoslavia (YUG) still included in this list, even␣
↪though their ISO 3166-1 codes were deleted in 2006 and 2003, respectively.
  child: [ALB, AND, AUS, AUT, BEL, BGR, BIH, CAN, CHE, CYP, CZE, DEU, DNK, ESP, EST, FIN,
↪ FLK, FRA, FRO, GBR, GIB, GRC, GRL, HRV, HUN, IMN, IRL, ISL, ITA, JPN, LIE, LTU, LUX,␣
↪LVA, MCO, MKD, MLT, MNE, NLD, NOR, NZL, POL, PRI, PRT, ROU, SCG, SHN, SJM, SMR, SPM,␣
↪SRB, SVK, SVN, SWE, TCA, TUR, USA, VAT, VGB, VIR, WLF, YUG]

R5_REF:
  parent: World
  description: Officially "R5.2REF".
  child: [ARM, AZE, BLR, GEO, KAZ, KGZ, MDA, RUS, TJK, TKM, UKR, UZB]
```

16.2 Others

These include models scoped to a single country or region, or a subset of all countries or regions, as well as code lists used in specific data sets from which `message_ix_models` handles data.

16.2.1 ADVANCE project (`ADVANCE`)

```yaml
# Node code list for the ADVANCE project
#
# Source: https://db1.ene.iiasa.ac.at/ADVANCEDB/dsd?Action=htmlpage&page=10
# Transcribed 2022-07-22 by P.N. Kishimoto
#

World:
  # Countries represented individually
  child: [BRA, CHN, IND, JPN, RUS, USA]

EU:
  description: >-
    European Union (28 member states from the accession of HRV in 2013 to the
    withdrawal of GBR in 2020).
  child: [AUT, BEL, BGR, CYP, CZE, DEU, DNK, ESP, EST, FIN, FRA, GBR, GRC, HRV, HUN, IRL,
↪ ITA, LTU, LUX, LVA, MLT, NLD, POL, PRT, ROU, SVK, SVN, SWE]

OECD90+EU:
  description: Includes the OECD 1990 countries as well as EU members and candidates.
  child: [ALB, AUS, AUT, BEL, BGR, BIH, CAN, CHE, CYP, CZE, DEU, DNK, ESP, EST, FIN, FJI,
↪ FRA, GBR, GRC, GUM, HRV, HUN, IRL, ISL, ITA, JPN, LTU, LUX, LVA, MKD, MLT, MNE, NCL,␣
↪NLD, NOR, NZL, POL, PRT, PYF, ROU, SLB, SRB, SVK, SVN, SWE, TUR, USA, VUT, WSM]

REF:
  description: Countries from the Reforming Ecomonies of the Former Soviet Union.
  child: [ARM, AZE, BLR, GEO, KAZ, KGZ, MDA, RUS, TJK, TKM, UKR, UZB]

ASIA:
  description: The region includes most Asian countries with the exception of the Middle␣
↪East, Japan and Former Soviet Union states.
  child: [AFG, BGD, BRN, BTN, CHN, HKG, IDN, IND, KHM, KOR, LAO, LKA, MAC, MDV, MMR, MNG,
↪ MYS, NPL, PAK, PHL, PNG, PRK, SGP, THA, TMP, TWN, VNM]

MAF:
  description: This region includes the countries of the Middle East and Africa.
  child: [DZA, AGO, ARE, BDA, BEN, BFA, BHR, BWA, CAF, CIV, CMR, COD, COG, COM, CPV, DJI,
↪ EGY, ERI, ESH, ETH, GAB, GHA, GIN, GMB, GNB, GNQ, IRN, IRQ, ISR, JOR, KEN, KWT, LBN,␣
↪LBR, LBY, LSO, MAR, MDG, MLI, MOZ, MRT, MUS, MWI, NAM, NER, NGA, OMN, QAT, REU, RQA,␣
↪SAU, SDN, SEN, SLE, SOM, SWZ, SYR, TCD, TGO, TUN, TZA, UGA, YEM, ZAF, ZMB, ZWE]

LAM:
  description: This region includes the countries of Latin America and the Caribbean.
  child: [ARG, ANT, BHS, BLZ, BOL, BRA, BRB, CHL, COL, CRA, CUB, DOM, EDU, GLP, GTM, GUY,
↪ HND, HTI, JAM, MEX, MTQ, NIC, PAN, PER, PRI, PRY, SLV, SUR, TTO, URY, VEN]
```

16.2.2 Israel (ISR)

```yaml
# Codes for the "node" dimension of the MESSAGE-IL model

World:
  name: World
  description: MESSAGE-IL regions

Israel:
  name: Israel
  parent: World
  child: [ISR]
```

CHAPTER

SEVENTEEN

YEARS OR TIME PERIODS (`YEAR/*.YAML`)

- See also:
  - The discussion of Years, periods, and time slices in the `message_ix` documentation, which explains the standard sense of time periods used across the MESSAGEix framework and specific models based on it.
  - *`ScenarioInfo.year_from_codes()`*
- These are not the only possible meanings of these codes; others may be used in data from other sources.

  For instance, the ID `2020` could be used to represent the period from 2017-07-01 to 2022-06-30. These lists alone cannot resolve these differences; they exist only to provide clarity about the sense used in `message_ix_models`.
- When working with data from other sources, `message_ix_models` code **must**:
  - Explicitly note (e.g. in comments or docstrings) any differing time discretization used in the other data.
  - Perform appropriate conversion *or* record a decision to use the data directly, without conversion.
- It is **optional** for code to fill Scenario parameters for the full set of historical years.

  For instance, when working with list `B`, code for a model variant or project could only populate parameter values for the historical periods `2010` and `2015`, but not `2005` and earlier. This **should** be described and documented on at the scope (function or module) where such subsets are selected from the full codelist.

17.1 List `A`

```
# Time periods used in CD-LINKS, inter alia

1960:
  description: Period from 1951-01-01 to 1960-12-31.
  # Durations of subsequent periods are implied by the prior period
  duration_period: 10

1970:
  description: Period from 1961-01-01 to 1970-12-31.
1980:
  description: Period from 1971-01-01 to 1980-12-31.
1990:
  description: Period from 1981-01-01 to 1990-12-31.
2000:
  description: Period from 1991-01-01 to 2000-12-31.
2010:
  description: Period from 2001-01-01 to 2010-12-31.
```

(continues on next page)

(continued from previous page)

```
2020:
  description: Period from 2011-01-01 to 2020-12-31.
  # Periods before this are historical
  firstmodelyear: true

2030:
  description: Period from 2021-01-01 to 2030-12-31.
2040:
  description: Period from 2031-01-01 to 2040-12-31.
2050:
  description: Period from 2041-01-01 to 2050-12-31.
2060:
  description: Period from 2051-01-01 to 2060-12-31.
2070:
  description: Period from 2061-01-01 to 2070-12-31.
2080:
  description: Period from 2071-01-01 to 2080-12-31.
2090:
  description: Period from 2081-01-01 to 2090-12-31.
2100:
  description: Period from 2091-01-01 to 2100-12-31.
2110:
  description: Period from 2101-01-01 to 2110-12-31.
```

17.2 List B

```
# Time periods used in ENGAGE, inter alia

1950:
  description: Period from 1946-01-01 to 1950-12-31.
  # Durations of subsequent periods are implied by the prior period
  duration_period: 5
1955:
  description: Period from 1951-01-01 to 1955-12-31.
1960:
  description: Period from 1956-01-01 to 1960-12-31.
1965:
  description: Period from 1961-01-01 to 1965-12-31.
1970:
  description: Period from 1966-01-01 to 1970-12-31.
1975:
  description: Period from 1971-01-01 to 1975-12-31.
1980:
  description: Period from 1976-01-01 to 1980-12-31.
1985:
  description: Period from 1981-01-01 to 1985-12-31.
1990:
  description: Period from 1986-01-01 to 1990-12-31.
1995:
```

(continues on next page)

(continued from previous page)

```
  description: Period from 1991-01-01 to 1995-12-31.
2000:
  description: Period from 1996-01-01 to 2000-12-31.
2005:
  description: Period from 2001-01-01 to 2005-12-31.
2010:
  description: Period from 2006-01-01 to 2010-12-31.
2015:
  description: Period from 2011-01-01 to 2015-12-31.

2020:
  description: Period from 2016-01-01 to 2020-12-31.
  # Periods before this are historical
  firstmodelyear: true

2025:
  description: Period from 2021-01-01 to 2025-12-31.
2030:
  description: Period from 2026-01-01 to 2030-12-31.
2035:
  description: Period from 2031-01-01 to 2035-12-31.
2040:
  description: Period from 2036-01-01 to 2040-12-31.
2045:
  description: Period from 2041-01-01 to 2045-12-31.
2050:
  description: Period from 2046-01-01 to 2050-12-31.
2055:
  description: Period from 2046-01-01 to 2055-12-31.
2060:
  description: Period from 2056-01-01 to 2060-12-31.
2070:
  description: Period from 2061-01-01 to 2070-12-31.
2080:
  description: Period from 2071-01-01 to 2080-12-31.
2090:
  description: Period from 2081-01-01 to 2090-12-31.
2100:
  description: Period from 2091-01-01 to 2100-12-31.
2110:
  description: Period from 2101-01-01 to 2110-12-31.
```

CHAPTER

EIGHTEEN

OTHER CODE LISTS

18.1 Commodities (`commodity.yaml`)

Each of these codes has the following annotations:

level
: Level where this commodity typically (not exclusively) occurs.

unit
: Units typically associated with this commodity.

```yaml
coal:
  name: Coal
  units: GWa

crudeoil:
  name: Crude oil
  description: >-
    For secondary energy, use 'fueloil', 'lightoil', etc.
  level: primary
  units: GWa

d_heat:
  name: (?) District heat
  description: >-
    FIXME provide an unambiguous description of what this commodity represents.

electr:
  name: Electricity
  units: GWa

ethanol:
  name: Ethanol
  units: GWa
```

(continues on next page)

(continued from previous page)

```
freshwater_supply:
  name: (?) Fresh water
  description: >-
    FIXME provide an unambiguous description of what this commodity represents.
  units: foo bar

fueloil:
  name: Fuel oil
  description: Heavy fuel oil.
  level: secondary
  units: GWa

gas:
  name: Natural Gas
  units: GWa

hydrogen:
  name: Gaseous hydrogen
  units: GWa

lh2:
  name: Liquid hydrogen
  units: GWa

lightoil:
  name: Light oil
  description: Includes gasoline, diesel oil.
  # level: secondary
  units: GWa

methanol:
  name: Methanol
  units: GWa

transport:
  name: Transportation
  description: >-
    For MESSAGEix-Transport, this commodity is not used; it is replaced by a
    disaggregated set of transport service demands (representing e.g. light-
    duty vehicles, civil aviation, freight transport, etc.)
  level: useful
  units: GWa

# The following codes also appear in a recent (2020-02-28) SSP2 scenario, but
# are not currently used by model.bare.create_res.
#
# Aff_CO2_G4M
# Agri_CH4
# Agri_N2O
# Agri_N2O_calc
# Agricultural Demand
```

(continues on next page)

(continued from previous page)

```
# Agricultural Demand|Bioenergy
# Agricultural Demand|Bioenergy|1st generation
# Agricultural Demand|Bioenergy|2nd generation
# Agricultural Demand|Feed
# Agricultural Demand|Feed|Crops
# Agricultural Demand|Food
# Agricultural Demand|Food|Crops
# Agricultural Demand|Food|Livestock
# Agricultural Demand|Non-Food
# Agricultural Demand|Non-Food|Crops
# Agricultural Demand|Non-Food|Livestock
# Agricultural Production
# Agricultural Production|Energy Crops
# Agricultural Production|Livestock
# Agricultural Production|Non-Energy Crops
# Agricultural Production|Non-Energy Crops|Cereals
# BCA_LandUseChangeEM
# BCA_SavanBurnEM
# Biodiesel_G1
# bioenergy
# Bioethanol_G1
# biomass
# CalAnim
# CalCrop
# CalTot
# CH4_LandUseChangeEM
# CH4_SavanBurnEM
# CO_LandUseChangeEM
# CO_SavanBurnEM
# CO2_oil
# CO2_rem
# cooling__bio_hpl
# cooling__bio_istig
# cooling__bio_istig_ccs
# cooling__bio_ppl
# cooling__coal_adv
# cooling__coal_adv_ccs
# cooling__coal_ppl
# cooling__coal_ppl_u
# cooling__foil_hpl
# cooling__foil_ppl
# cooling__gas_cc
# cooling__gas_cc_ccs
# cooling__gas_hpl
# cooling__gas_ppl
# cooling__geo_hpl
# cooling__geo_ppl
# cooling__igcc
# cooling__igcc_ccs
# cooling__loil_cc
# cooling__loil_ppl
# cooling__nuc_hc
```

(continues on next page)

(continued from previous page)

```
# cooling__nuc_lc
# cooling__solar_th_ppl
# CrpLnd
# crude_1
# crude_2
# crude_3
# crude_4
# crude_5
# crude_6
# crude_7
# crude_8
# Def_CO2_G4M
# Def_CO2_GLO
# dumagr
# dumfert
# Emissions|CH4|Land Use
# Emissions|CH4|Land Use|Agricultural Waste Burning
# Emissions|CH4|Land Use|Agriculture
# Emissions|CH4|Land Use|Agriculture|AWM
# Emissions|CH4|Land Use|Agriculture|Enteric Fermentation
# Emissions|CH4|Land Use|Agriculture|Rice
# Emissions|CH4|Land Use|Savannah Burning
# Emissions|CO2|Land Use
# Emissions|CO2|Land Use|Negative
# Emissions|CO2|Land Use|Positive
# Emissions|N2O|Land Use
# Emissions|N2O|Land Use|Agricultural Waste Burning
# Emissions|N2O|Land Use|Agriculture
# Emissions|N2O|Land Use|Agriculture|AWM
# Emissions|N2O|Land Use|Agriculture|Cropland Soils
# Emissions|N2O|Land Use|Agriculture|Pasture
# Emissions|N2O|Land Use|Savannah Burning
# EnergyRoundwood
# exports
# Fertilizer Use|Nitrogen
# Fertilizer Use|Phosphorus
# Fmg_CO2_G4M
# Food Demand
# Food Demand|Crops
# Food Demand|Livestock
# Food Energy Demand
# Food Energy Demand|Livestock
# ForestBiomass
# ForestBiomass_G4M
# ForestHarvestDF_G4M
# ForestHarvestFM_G4M
# ForestHarvestTot_G4M
# ForestHarvestTot_GLO
# Forestry Demand|Roundwood
# Forestry Demand|Roundwood|Industrial Roundwood
# Forestry Demand|Roundwood|Wood Fuel
# Forestry Production|Forest Residues
```

(continues on next page)

(continued from previous page)

```
# Forestry Production|Roundwood
# Forestry Production|Roundwood|Industrial Roundwood
# Forestry Production|Roundwood|Wood Fuel
# freshwater_instream
# FuelWood
# FuelWood_G4M
# gas
# gas_1
# gas_2
# gas_3
# gas_4
# gas_5
# gas_6
# gas_7
# gas_8
# gas_afr
# gas_cpa
# gas_eeu
# gas_nam
# gas_pao
# gas_pas
# gas_sas
# gas_weu
# GrsLnd
# i_feed
# i_spec
# i_therm
# IrriWithdrawal
# Land Cover
# Land Cover|Cropland
# Land Cover|Cropland|Cereals
# Land Cover|Cropland|Energy Crops
# Land Cover|Cropland|Irrigated
# Land Cover|Forest
# Land Cover|Forest|Afforestation and Reforestation
# Land Cover|Forest|Forestry
# Land Cover|Forest|Managed
# Land Cover|Forest|Natural Forest
# Land Cover|Other Natural Land
# Land Cover|Pasture
# lh2
# lignite
# LiquidTotal
# LNG
# LoggingResidues
# LU_CO2
# LU_GHG
# LucGrs_CO2
# LucOth_CO2
# NewFor_G4M
# NH3_LandUseChangeEM
# NH3_ManureEM
```

(continues on next page)

(continued from previous page)

```
# NH3_RiceEM
# NH3_SavanBurnEM
# NH3_SoilEM
# non-comm
# NOx_LandUseChangeEM
# NOx_SavanBurnEM
# NOx_SoilEM
# nucfuel
# OCA_LandUseChangeEM
# OCA_SavanBurnEM
# oil_st
# Olc_CO2_GLO
# Olc_CO2_GLO_neg
# Olc_CO2_GLO_pos
# OldFor_G4M
# OtherLnd
# OthSolidNonComm
# PlantationBiomass
# PlantationHarvest_GLO
# PltArt
# PltFor
# PltFor_gr
# Price_BIO
# Price_CO2
# Price|Agriculture|Non-Energy Crops and Livestock|Index
# Price|Agriculture|Non-Energy Crops|Index
# Price|Primary Energy|Biomass
# pu
# puq
# puq2
# rc_spec
# rc_therm
# saline_supply
# SawmillResidues
# SawmillResidues_G4M
# shipping
# SO2_LandUseChangeEM
# SO2_SavanBurnEM
# SolidExogenous
# SolidExogenous_G4M
# SolidTotal
# SolidTotal_G4M
# TCE
# TimberIndust
# TimberIndust_G4M
# Tot_CO2_G4M
# total_cost
# TotalLnd
# TotFor_G4M
# u5
# u5q
# u5t
```

(continues on next page)

(continued from previous page)

```
# upstream_landuse
# uq
# uranium
# VOC_LandUseChangeEM
# VOC_SavanBurnEM
# water_constraint
# Water|Withdrawal|Irrigation
# Yield|Cereal
# Yield|Oilcrops
# Yield|Sugarcrops
```

18.2 Levels (`level.yaml`)

This code list has no annotations and no hierarchy.

```
primary:
  name: Primary Energy
  description: >-
    A form found in nature that has not been subjected to any human engineered
    conversion process.

secondary:
  name: Secondary Energy
  description: Forms which have been transformed from primary energy.

final:
  name: Final Energy
  description: >-
    Represents end-use demands of the end-use sectors (e.g. industry, transport,
    residential, commercial and agriculture).

import:
  name: Imports

useful:
  name: Useful Energy
  description: >-
    Represents energy-service demands (or activity levels) of the end-use
    sectors in non-energy units.

water_supply:
  name: Water Supply
  description: >-
    FIXME provide an unambiguous description of what this level represents.

# The following codes also appear in a recent (2020-02-28) SSP2 scenario, but
# are not currently used by model.bare.create_res.
#
# cooling
# export
```

(continues on next page)

(continued from previous page)

```
# land_use_reporting
# land_use
# piped-gas
# resource
# stocks
# water_supply_constraint
```

18.3 Technologies (`technology.yaml`)

Warning: This list is *only for reference*; particular MESSAGE-GLOBIOM scenarios may not contain all these technologies, or may contain other technologies not listed.

Each of these codes has the following annotations:

`sector`
: A categorization of the technology.

`input`
: (`commodity`, `level`) for input to the technology.

`output`
: (`commodity`, `level`) for output from the technology.

`vintaged`
: `True` if the technology is subject to vintaging.

`type`
: Same as `output[1]`.

```
# This file describes a possible set of base technologies to be used in the
# global model. It will be usable by both model creation and reporting code.
#
# Each entry includes the following fields:
# - name, description: required.
# - sector: a label to group multiple technologies to a notional sector.
#   Required.
# - output (required) and input (optional): either
#   - a list of [commodity, level] giving the output generated or input used by
#     the technology; or,
#   - a list of 2 or more such lists, if the technology has multiple inputs or
#     outputs.
# - vintaged: True if the technology's properties vary by year_vintage.
#   Optional; False if omitted.
# - type: In the Excel file used to create this YAML file (see
#   https://github.com/iiasa/message_data/issues/74), 'type' appears to be
#   always the same as 'output'/level; *unless* the 'output'/commodity is a
#   dummy commodity, in which case it is 'dummy'.
#
#   TODO if this is the case, remove 'type' from this file, and generate 'type'
#        in tools.technologies.get_info.
```

(continues on next page)

(continued from previous page)

```yaml
CF4_TCE:
  name: CF4_TCE
  description: Tetrafluoromethane (CF4) Total Carbon Emissions
  type: primary
  sector: dummy
  output: [dummy, primary]

CH4_TCE:
  name: CH4_TCE
  description: Methane total carbon equivalent emissions
  type: dummy
  sector: dummy
  output: [dummy, primary]

CH4g_TCE:
  name: CH4g_TCE
  description: CH4 emissions from animals directly in Total Carbon Equivalent emissions
  type: dummy
  sector: dummy
  output: [dummy agriculture, primary]

CH4n_TCE:
  name: CH4n_TCE
  description: CH4 emissions from anaerobic waste decomposition in Total Carbon␣
↪Equivalent emissions
  type: dummy
  sector: dummy
  output: [dummy, primary]

CH4o_TCE:
  name: CH4o_TCE
  description: Dummy technology converting CH4 emissions from industrial and domestic␣
↪wastewater, non energy biomass burning and other CH4 emissions, to total carbon␣
↪equivalent emissions (TCE)
  type: dummy
  sector: dummy
  output: [dummy, primary]

CO2_TCE:
  name: CO2_TCE
  description: CO2 total carbon equivalent emissions
  type: dummy
  sector: dummy
  output: [dummy, primary]

dom_total:
  name: dom_total
  description: Used in balance equation for domestic energy supply and in the equation␣
↪for constraining net imports
  type: secondary
  sector: dummy
  output: [exports, secondary]
```

(continues on next page)

(continued from previous page)

```
dummy_producer:
  name: dummy_producer
  description: Technology added to produce dummy energy to avoid infeasibility if needed␣
↪(e.g. when CO2_TCE needs to go negative)
  type: primary
  sector: dummy
  output: [dummy, primary]

exp_total:
  name: exp_total
  description: Used in balance equation for exported energy supply and in the equation␣
↪for constraining net imports
  type: secondary
  sector: dummy
  output: [exports, secondary]

HFC_TCE:
  name: HFC_TCE
  description: HFC total carbon equivalent emissions
  type: primary
  sector: dummy
  output: [dummy, primary]

HFCo_TCE:
  name: HFCo_TCE
  description: Dummy technology converting HFC equiv emissions from solvents, fire␣
↪extinguishers, aerosols MDI, aerosols non-MDI to total carbon equivalent emissions␣
↪(TCE)
  type: primary
  sector: dummy
  output: [dummy, primary]

imp_total:
  name: imp_total
  description: Used in balance equation for imported energy supply
  type: exports
  sector: dummy
  output: [exports, secondary]

N2O_TCE:
  name: N2O_TCE
  description: N2O total carbon equivalent emissions
  type: dummy
  sector: dummy
  output: [dummy]

N2OG_TCE:
  name: N2OG_TCE
  description: N2O soil emissions total carbon equivalent emissions
  type: dummy
  sector: dummy
```

(continues on next page)

(continued from previous page)

```
  output: [dummy]

N2On_TCE:
  name: N2On_TCE
  description: N2O adipic acid total carbon equivalent emissions
  type: dummy
  sector: dummy
  output: [dummy]

N2Oo_TCE:
  name: N2Oo_TCE
  description: Dummy technology converting N2O emissions from Manure Management, Human␣
↪Sewage, Other Agricultural sources, Other Non Ag Sources to total carbon equivalent␣
↪emissions (TCE)
  type: dummy
  sector: dummy
  output: [dummy]

nica_con:
  name: nica_con
  type: dummy
  sector: dummy
  output: [dummy]

nitric_catalytic1:
  name: nitric_catalytic1
  description: Mitigation technology (catalytic converter) category 2 for N2O emissions
  type: dummy
  sector: dummy
  output: [dummy, primary]

nitric_catalytic2:
  name: nitric_catalytic2
  description: Mitigation technology (catalytic converter) category 1 for N2O emissions
  type: dummy
  sector: dummy
  output: [dummy, primary]

nitric_catalytic3:
  name: nitric_catalytic3
  description: Mitigation technology (catalytic converter) category 3 for N2O emissions
  type: dummy
  sector: dummy
  output: [dummy, primary]

nitric_catalytic4:
  name: nitric_catalytic4
  description: Mitigation technology (catalytic converter) category 4 for N2O emissions
  type: dummy
  sector: dummy
  output: [dummy, primary]
```

(continues on next page)

(continued from previous page)

```yaml
nitric_catalytic5:
  name: nitric_catalytic5
  description: Mitigation technology (catalytic converter) category 5 for N2O emissions
  type: dummy
  sector: dummy
  output: [dummy, primary]

nitric_catalytic6:
  name: nitric_catalytic6
  description: Mitigation technology (catalytic converter) category 6 for N2O emissions
  type: dummy
  sector: dummy
  output: [dummy, primary]

nitric_catalytic7:
  name: nitric_catalytic7
  description: Mitigation technology (catalytic converter) category 7 for N2O emissions
  type: dummy
  sector: dummy
  output: [dummy, primary]

SF6_TCE:
  name: SF6_TCE
  description: SF6 total carbon equivalent emissions
  type: primary
  sector: dummy
  output: [dummy, primary]

useful_feedstock:
  name: useful_feedstock
  description: Share constraint for industry feedstocks
  type: dummy
  sector: dummy
  output: [dummy_useful, primary]

useful_industry_sp:
  name: useful_industry_sp
  description: Share constraint for Industry Specific
  type: dummy
  sector: dummy
  output: [dummy_useful, primary]

useful_industry_th:
  name: useful_industry_th
  description: Share constraint for Industry Thermal
  type: dummy
  sector: dummy
  output: [dummy_useful, primary]

useful_res/comm_sp:
  name: useful_res/comm_sp
  description: Share constraint for Residential and Commercial Specific
```

(continues on next page)

(continued from previous page)

```
  type: dummy
  sector: dummy
  output: [dummy_useful, primary]

useful_res/comm_th:
  name: useful_res/comm_th
  description: Share constraint for Residential and Commercial Thermal
  type: dummy
  sector: dummy
  output: [dummy_useful, primary]

useful_transport:
  name: useful_transport
  description: Share constraint for Transport
  type: dummy
  sector: dummy
  output: [dummy_useful, primary]

bio_istig:
  name: bio_istig
  description: Advanced biomass power plant- gasified biomass is burned in gas turbine␣
↪plant - modes with and without net carbon release
  type: secondary
  vintaged: TRUE
  sector: electricity
  input: [biomass, primary]
  output: [electr, secondary]

bio_istig_ccs:
  name: bio_istig_ccs
  description: Advanced biomass power plant with carbon capture and storage- gasified␣
↪biomass is burned in gas turbine plant - modes with and without net carbon release
  type: secondary
  vintaged: TRUE
  sector: electricity
  input: [biomass, primary]
  output: [electr, secondary]

bio_ppl:
  name: bio_ppl
  description: Bio powerplant
  type: secondary
  vintaged: TRUE
  sector: electricity
  input:
    - [biomass, primary]
    - [cooling__bio_ppl, cooling]
    - [freshwater_supply, water_supply]
  output: [electr, secondary]

coal_adv:
  name: coal_adv
```

(continues on next page)

(continued from previous page)

```
  description: Advanced coal power plant
  type: secondary
  vintaged: TRUE
  sector: electricity
  input: [coal, secondary]
  output: [electr, secondary]

coal_adv_ccs:
  name: coal_adv_ccs
  description: Advanced coal power plant with carbon capture and storage
  type: secondary
  vintaged: TRUE
  sector: electricity
  input: [coal, secondary]
  output: [electr, secondary]

coal_ppl:
  name: coal_ppl
  description: Coal power-plant
  type: secondary
  vintaged: TRUE
  sector: electricity
  input: [coal, secondary]
  output: [electr, secondary]

coal_ppl_u:
  name: coal_ppl_u
  description: Coal power plant without abatement measures
  type: secondary
  vintaged: TRUE
  sector: electricity
  input: [coal, secondary]
  output: [electr, secondary]

elec_exp:
  name: elec_exp
  description: Net export of electricity
  type: exports
  sector: electricity
  input: [electr, secondary]
  output: [electr, exports]

elec_imp:
  name: elec_imp
  description: Net import of electricity
  type: secondary
  sector: electricity
  input: [electr, imports]
  output: [electr, secondary]

elec_t/d:
  name: elec_t/d
```

(continues on next page)

(continued from previous page)

```
  description: Grid technology cost converted to 2005$
  type: final
  vintaged: TRUE
  sector: electricity
  input: [electr, secondary]
  output: [electr, final]

foil_ppl:
  name: foil_ppl
  description: New standard oil power plant, Rankine cycle
  type: secondary
  vintaged: TRUE
  sector: electricity
  input: [fueloil, secondary]
  output: [electr, secondary]

gas_cc:
  name: gas_cc
  description: Gas combined cycle power-plant
  type: secondary
  vintaged: TRUE
  sector: electricity
  input: [gas, secondary]
  output: [electr, secondary]

gas_cc_ccs:
  name: gas_cc_ccs
  description: Gas combined cycle power-plant with carbon capture and storage
  type: secondary
  vintaged: TRUE
  sector: electricity
  input: [gas, secondary]
  output: [electr, secondary]

gas_ct:
  name: gas_ct
  description: Gas combustion-turbine power plant
  type: secondary
  vintaged: TRUE
  sector: electricity
  input: [gas, secondary]
  output: [electr, secondary]

gas_htfc:
  name: gas_htfc
  description: High temperature fuel cell powered with natural gas
  type: secondary
  vintaged: TRUE
  sector: electricity
  input: [gas, secondary]
  output: [electr, secondary]
```

(continues on next page)

(continued from previous page)

```
gas_ppl:
  name: gas_ppl
  description: Gas power plant, Rankine cycle
  type: secondary
  vintaged: TRUE
  sector: electricity
  input: [gas, secondary]
  output: [electr, secondary]

geo_ppl:
  name: geo_ppl
  description: Geothermal power plant
  type: secondary
  vintaged: TRUE
  sector: electricity
  output: [electr, secondary]

glb_elec_exp:
  name: glb_elec_exp
  description: Global net export of electricity
  type: imports
  sector: electricity
  output: [electr, imports]

glb_elec_imp:
  name: glb_elec_imp
  description: Global net import of electricity
  type: exports
  sector: electricity
  input: [electr, exports]
  output: [exports]

hydro_hc:
  name: hydro_hc
  description: High cost hydro power plant
  type: secondary
  vintaged: TRUE
  sector: electricity
  output: [electr, secondary]

hydro_lc:
  name: hydro_lc
  description: Low cost hydro power plant
  type: secondary
  vintaged: TRUE
  sector: electricity
  output: [electr, secondary]

igcc:
  name: igcc
  description: Integrated gasification combined cycle (IGCC) power plant
  type: secondary
```

(continues on next page)

(continued from previous page)

```
  vintaged: TRUE
  sector: electricity
  input: [coal, secondary]
  output: [electr, secondary]

igcc_ccs:
  name: igcc_ccs
  description: Integrated gasification combined cycle (IGCC) power plant with carbon␣
↪capture and storage
  type: secondary
  vintaged: TRUE
  sector: electricity
  input: [coal, secondary]
  output: [electr, secondary]

igcc_co2scr:
  name: igcc_co2scr
  description: New coal scrubber for igcc plants
  type: exports
  vintaged: TRUE
  sector: electricity
  output: [exports, secondary]

loil_cc:
  name: loil_cc
  description: Light oil combined cycle
  type: secondary
  vintaged: TRUE
  sector: electricity
  input: [lightoil, secondary]
  output: [electr, secondary]

loil_ppl:
  name: loil_ppl
  description: Existing light oil power-plant
  type: secondary
  vintaged: TRUE
  sector: electricity
  input: [lightoil, secondary]
  output: [electr, secondary]

nuc_hc:
  name: nuc_hc
  description: Nuclear power plant (~GEN III+), high cost
  type: secondary
  vintaged: TRUE
  sector: electricity
  input: [uranium, stocks]
  output: [electr, secondary]

nuc_lc:
  name: nuc_lc
```

(continues on next page)

(continued from previous page)

```
  description: Nuclear power plant (~GEN II), low cost
  type: secondary
  vintaged: TRUE
  sector: electricity
  input: [uranium, stocks]
  output: [electr, secondary]

solar_curtailment1:
  name: solar_curtailment1
  description: Solar PV curtailment steps
  type: dummy
  sector: electricity
  input: [electr, secondary]
  output: [dummy renewable, secondary]

solar_curtailment2:
  name: solar_curtailment2
  description: Solar PV curtailment steps
  type: dummy
  sector: electricity
  input: [electr, secondary]
  output: [dummy renewable, secondary]

solar_curtailment3:
  name: solar_curtailment3
  description: Solar PV curtailment steps
  type: dummy
  sector: electricity
  input: [electr, secondary]
  output: [dummy renewable, secondary]

solar_cv1:
  name: solar_cv1
  description: Quadratic systems integration costs added to solar PV
  type: dummy
  sector: electricity
  output: [dummy renewable, secondary]

solar_cv2:
  name: solar_cv2
  description: Quadratic systems integration costs added to solar PV
  type: dummy
  sector: electricity
  output: [dummy renewable, secondary]

solar_cv3:
  name: solar_cv3
  description: Quadratic systems integration costs added to solar PV
  type: dummy
  sector: electricity
  output: [dummy renewable, secondary]
```

(continues on next page)

(continued from previous page)

```yaml
solar_cv4:
  name: solar_cv4
  description: Quadratic systems integration costs added to solar PV
  type: dummy
  sector: electricity
  output: [dummy renewable, secondary]

solar_pv_ppl:
  name: solar_pv_ppl
  description: Solar photovoltaic power plant (no storage)
  type: dummy
  vintaged: TRUE
  sector: electricity
  output: [dummy renewable, secondary]

solar_res1:
  name: solar_res1
  description: Maximum solar electricity potential 1
  type: secondary
  sector: electricity
  output: [electr, secondary]

solar_res2:
  name: solar_res2
  description: Maximum solar electricity potential 2
  type: secondary
  sector: electricity
  output: [electr, secondary]

solar_res3:
  name: solar_res3
  description: Maximum solar electricity potential 3
  type: secondary
  sector: electricity
  output: [electr, secondary]

solar_res4:
  name: solar_res4
  description: Maximum solar electricity potential 4
  type: secondary
  sector: electricity
  output: [electr, secondary]

solar_res5:
  name: solar_res5
  description: Maximum solar electricity potential 5
  type: secondary
  sector: electricity
  output: [electr, secondary]

solar_res6:
  name: solar_res6
```

(continues on next page)

(continued from previous page)

```
  description: Maximum solar electricity potential 6
  type: secondary
  sector: electricity
  output: [electr, secondary]

solar_res7:
  name: solar_res7
  description: Maximum solar electricity potential 7
  type: secondary
  sector: electricity
  output: [electr, secondary]

solar_th_ppl:
  name: solar_th_ppl
  description: Solar thermal power plant with storage
  type: secondary
  vintaged: TRUE
  sector: electricity
  output: [electr, secondary]

stor_ppl:
  name: stor_ppl
  description: Generic electric storage
  type: secondary
  vintaged: TRUE
  sector: electricity
  input: [electr, secondary]
  output: [exports, secondary]

wind_curtailment1:
  name: wind_curtailment1
  description: Wind curtailment steps
  type: dummy
  sector: electricity
  input: [secondary, electricity]
  output: [dummy renewable, secondary]

wind_curtailment2:
  name: wind_curtailment2
  description: Wind curtailment steps
  type: dummy
  sector: electricity
  input: [secondary, electricity]
  output: [dummy renewable, secondary]

wind_curtailment3:
  name: wind_curtailment3
  description: Wind curtailment steps
  type: dummy
  sector: electricity
  input: [secondary, electricity]
  output: [dummy renewable, secondary]
```

(continues on next page)

(continued from previous page)

```
wind_cv1:
  name: wind_cv1
  description: Wind flexibility requirement and firm capacity contribution, quadratic␣
↪systems integration costs added to wind
  type: dummy
  sector: electricity
  output: [dummy renewable, secondary]

wind_cv2:
  name: wind_cv2
  description: Wind flexibility requirement and firm capacity contribution, quadratic␣
↪systems integration costs added to wind
  type: dummy
  sector: electricity
  output: [dummy renewable, secondary]

wind_cv3:
  name: wind_cv3
  description: Wind flexibility requirement and firm capacity contribution, quadratic␣
↪systems integration costs added to wind
  type: dummy
  sector: electricity
  output: [dummy renewable, secondary]

wind_cv4:
  name: wind_cv4
  description: Wind flexibility requirement and firm capacity contribution, quadratic␣
↪systems integration costs added to wind
  type: dummy
  sector: electricity
  output: [dummy renewable, secondary]

wind_ppl:
  name: wind_ppl
  description: Wind power plant onshore (provides capacity only)
  type: dummy
  vintaged: TRUE
  sector: electricity
  output: [dummy renewable, secondary]

wind_res1:
  name: wind_res1
  description: Wind onshore potential and generation 1
  type: secondary
  sector: electricity
  output: [electr, secondary]

wind_res2:
  name: wind_res2
  description: Wind onshore potential and generation 2
  type: secondary
```

(continues on next page)

(continued from previous page)

```
  sector: electricity
  output: [electr, secondary]

wind_res3:
  name: wind_res3
  description: Wind onshore potential and generation 3
  type: secondary
  sector: electricity
  output: [electr, secondary]

wind_res4:
  name: wind_res4
  description: Wind onshore potential and generation 4
  type: secondary
  sector: electricity
  output: [electr, secondary]

wind_ppf:
  name: wind_ppf
  description: Wind power plant offshore (provides capacity only)
  type: secondary
  vintaged: TRUE
  sector: electricity
  output: [dummy renewable, secondary]

wind_ref1:
  name: wind_ref1
  description: Wind offshore potential and generation 1
  type: secondary
  sector: electricity
  output: [electr, secondary]

wind_ref2:
  name: wind_ref2
  description: Wind offshore potential and generation 2
  type: secondary
  sector: electricity
  output: [electr, secondary]

wind_ref3:
  name: wind_ref3
  description: Wind offshore potential and generation 3
  type: secondary
  sector: electricity
  output: [electr, secondary]

wind_ref4:
  name: wind_ref4
  description: Wind offshore potential and generation 4
  type: secondary
  sector: electricity
  output: [electr, secondary]
```

(continues on next page)

(continued from previous page)

```
wind_ref5:
  name: wind_ref5
  description: Wind offshore potential and generation 5
  type: secondary
  sector: electricity
  output: [electr, secondary]

csp_sm3_res:
  name: csp_sm3_res
  description: Concentrating solar power (CSP) with solar multiple of 3 potential and␣
↪generation 1
  type: secondary
  sector: electricity
  output: [electr, secondary]

csp_sm3_res1:
  name: csp_sm3_res1
  description: Concentrating solar power (CSP) with solar multiple of 3 potential and␣
↪generation 2
  type: secondary
  sector: electricity
  output: [electr, secondary]

csp_sm3_res2:
  name: csp_sm3_res2
  description: Concentrating solar power (CSP) with solar multiple of 3 potential and␣
↪generation 3
  type: secondary
  sector: electricity
  output: [electr, secondary]

csp_sm3_res3:
  name: csp_sm3_res3
  description: Concentrating solar power (CSP) with solar multiple of 3 potential and␣
↪generation 4
  type: secondary
  sector: electricity
  output: [electr, secondary]

csp_sm3_res4:
  name: csp_sm3_res4
  description: Concentrating solar power (CSP) with solar multiple of 3 potential and␣
↪generation 5
  type: secondary
  sector: electricity
  output: [electr, secondary]

csp_sm3_res5:
  name: csp_sm3_res5
  description: Concentrating solar power (CSP) with solar multiple of 3 potential and␣
↪generation 6
```

(continues on next page)

(continued from previous page)

```
  type: secondary
  sector: electricity
  output: [electr, secondary]

csp_sm3_res6:
  name: csp_sm3_res6
  description: Concentrating solar power (CSP) with solar multiple of 3 potential and␣
↪generation 7
  type: secondary
  sector: electricity
  output: [electr, secondary]

csp_sm3_res7:
  name: csp_sm3_res7
  description: Concentrating solar power (CSP) with solar multiple of 3 potential and␣
↪generation 8
  type: secondary
  sector: electricity
  output: [electr, secondary]

csp_sm3_ppl:
  name: csp_sm3_ppl
  description: Concentrating solar power (CSP) with solar multiple of 3 (provides␣
↪capacity only)
  type: secondary
  vintaged: TRUE
  sector: electricity
  output: [dummy renewable, secondary]

csp_sm1_res:
  name: csp_sm1_res
  description: Concentrating solar power (CSP) with solar multiple of 1 potential and␣
↪generation 1
  type: secondary
  sector: electricity
  output: [electr, secondary]

csp_sm1_res1:
  name: csp_sm1_res1
  description: Concentrating solar power (CSP) with solar multiple of 1 potential and␣
↪generation 2
  type: secondary
  sector: electricity
  output: [electr, secondary]

csp_sm1_res2:
  name: csp_sm1_res2
  description: Concentrating solar power (CSP) with solar multiple of 1 potential and␣
↪generation 3
  type: secondary
  sector: electricity
  output: [electr, secondary]
```

(continues on next page)

(continued from previous page)

```
csp_sm1_res3:
  name: csp_sm1_res3
  description: Concentrating solar power (CSP) with solar multiple of 1 potential and␣
↪generation 4
  type: secondary
  sector: electricity
  output: [electr, secondary]

csp_sm1_res4:
  name: csp_sm1_res4
  description: Concentrating solar power (CSP) with solar multiple of 1 potential and␣
↪generation 5
  type: secondary
  sector: electricity
  output: [electr, secondary]

csp_sm1_res5:
  name: csp_sm1_res5
  description: Concentrating solar power (CSP) with solar multiple of 1 potential and␣
↪generation 6
  type: secondary
  sector: electricity
  output: [electr, secondary]

csp_sm1_res6:
  name: csp_sm1_res6
  description: Concentrating solar power (CSP) with solar multiple of 1 potential and␣
↪generation 7
  type: secondary
  sector: electricity
  output: [electr, secondary]

csp_sm1_res7:
  name: csp_sm1_res7
  description: Concentrating solar power (CSP) with solar multiple of 1 potential and␣
↪generation 8
  type: secondary
  sector: electricity
  output: [electr, secondary]

csp_sm1_ppl:
  name: csp_sm1_ppl
  description: Concentrating solar power (CSP) with solar multiple of 1 (provides␣
↪capacity only)
  type: secondary
  vintaged: TRUE
  sector: electricity
  output: [dummy renewable, secondary]

bio_extr_1:
  name: bio_extr_1
```

(continues on next page)

(continued from previous page)

```
  description: Biomass Extraction
  type: primary
  sector: extraction
  output: [biomass, primary]

bio_extr_2:
  name: bio_extr_2
  description: Biomass Extraction
  type: primary
  sector: extraction
  output: [biomass, primary]

bio_extr_3:
  name: bio_extr_3
  description: Biomass Extraction
  type: primary
  sector: extraction
  output: [biomass, primary]

bio_extr_4:
  name: bio_extr_4
  description: Biomass Extraction
  type: primary
  sector: extraction
  output: [biomass, primary]

bio_extr_5:
  name: bio_extr_5
  description: Biomass Extraction
  type: primary
  sector: extraction
  output: [biomass, primary]

bio_extr_6:
  name: bio_extr_6
  description: Biomass Extraction
  type: primary
  sector: extraction
  output: [biomass, primary]

coal_extr:
  name: coal_extr
  description: Hard coal extraction, world average grade A
  type: primary
  sector: extraction
  input: [coal, resource]
  output: [coal, primary]

coal_extr_ch4:
  name: coal_extr_ch4
  description: Describes efforts in CH4-reduction from coal mining
  type: primary
```

(continues on next page)

(continued from previous page)

```
  vintaged: TRUE
  sector: extraction
  input: [coal, resource]
  output: [coal, primary]

flaring_CO2:
  name: flaring_CO2
  description: Co2 emissions from gas flaring
  type: exports
  sector: extraction
  output: [exports, exports]

gas_extr_1:
  name: gas_extr_1
  description: Natural gas extraction, Cat I = Master et al.14.WPC "Identified Reserves"
  type: primary
  sector: extraction
  input: [resource]
  output: [primary]

gas_extr_2:
  name: gas_extr_2
  description: Natural gas extraction, Cat II = Master et al.14.WPC "Mode" undiscovered␣
↪natural gas
  type: primary
  sector: extraction
  input: [resource]
  output: [primary]

gas_extr_3:
  name: gas_extr_3
  description: Natural gas extraction, Cat III = Masters  et al.14.WPC Difference␣
↪between "Mode and 5%"  undiscovered natural gas
  type: primary
  sector: extraction
  input: [resource]
  output: [primary]

gas_extr_4:
  name: gas_extr_4
  description: Natural gas extraction, Cat IV = Estimated enhanced Recovery (30% of␣
↪Resources I+II+III)  plus 15% of historical production
  type: primary
  sector: extraction
  input: [resource]
  output: [primary]

gas_extr_5:
  name: gas_extr_5
  description: Natural gas extraction, Cat V = Non-conventional reserves (20% of Coal␣
↪bed; 15% of fractured Shale; 15% of Tight formation)
  type: primary
```

(continues on next page)

(continued from previous page)

```
  sector: extraction
  input: [resource]
  output: [primary]

gas_extr_6:
  name: gas_extr_6
  description: Natural gas extraction, Cat VI -VII= Non-conventional resources. Rest of␣
↪Coal bed (80%), fractured Shale (85%) and Tight formation (85%) were aggregated and␣
↪then distributed to VI (40%) nd VII (60%)
  type: primary
  sector: extraction
  input: [resource]
  output: [primary]

gas_extr_mpen:
  name: gas_extr_mpen
  description: Common Market penetration for all gas extraction technologies
  type: secondary
  sector: extraction
  output: [exports, secondary]

lignite_extr:
  name: lignite_extr
  description: Lignite extraction, world average grade A
  type: primary
  sector: extraction
  input: [lignite, resource]
  output: [coal, primary]

oil_extr_1:
  name: oil_extr_1
  description: Crude oil extraction, Cat I = Masters 14 WPC conv. oil reserves
  type: primary
  sector: extraction
  input: [crude 1 resource, primary]
  output: [crude oil, primary]

oil_extr_1_ch4:
  name: oil_extr_1_ch4
  description: Describes efforts in CH4-reduction from oil extraction of Cat I
  type: primary
  sector: extraction
  input: [crude 1 resource, primary]
  output: [crude oil, primary]

oil_extr_2:
  name: oil_extr_2
  description: Crude oil extraction, Cat II = Masters mode undiscovered conv. oil (incl.␣
↪NGL)
  type: primary
  sector: extraction
  input: [crude 2 resource, primary]
```

(continues on next page)

(continued from previous page)

```
  output: [crude oil, primary]

oil_extr_2_ch4:
  name: oil_extr_2_ch4
  description: Describes efforts in CH4-reduction from oil extraction of Cat II
  type: primary
  sector: extraction
  input: [crude 2 resource, primary]
  output: [crude oil, primary]

oil_extr_3:
  name: oil_extr_3
  description: Crude oil extraction, Cat III = Masters 5% - Masters 50%
  type: primary
  sector: extraction
  input: [crude 3 resource, primary]
  output: [crude oil, primary]

oil_extr_3_ch4:
  name: oil_extr_3_ch4
  description: Describes efforts in CH4-reduction from oil extraction of Cat III
  type: primary
  sector: extraction
  input: [crude 3 resource, primary]
  output: [crude oil, primary]

oil_extr_4:
  name: oil_extr_4
  description: Crude oil extraction, Cat IV = Recoverable "reserves" non-conventional oil
  type: primary
  sector: extraction
  input: [crude 4 resource, primary]
  output: [crude oil, primary]

oil_extr_4_ch4:
  name: oil_extr_4_ch4
  description: Describes efforts in CH4-reduction from oil extraction of Cat IV
  type: primary
  sector: extraction
  input: [crude 4 resource, primary]
  output: [crude oil, primary]

oil_extr_5:
  name: oil_extr_5
  description: Crude oil extraction, Cat V = Recoverable reserves of nonconventional oil␣
↪= Shale, tarsands/bitumen and heavy oils
  type: primary
  sector: extraction
  input: [crude 5 resource, primary]
  output: [crude oil, primary]

oil_extr_6:
```

(continues on next page)

(continued from previous page)

```
  name: oil_extr_6
  description: Crude oil extraction, Cat VI = 20% of estimated occurrences (-reserves)␣
↪of shale, heavy oils, tarsands/bitumen
  type: primary
  sector: extraction
  input: [crude 6 resource, primary]
  output: [crude oil, primary]

oil_extr_mpen:
  name: oil_extr_mpen
  description: Common Market penetration for all oil extraction technologies
  type: secondary
  sector: extraction
  output: [exports, secondary]

Feeds_1:
  name: Feeds_1
  description: Conservation cost curve step for industry feedstock demand
  type: useful
  sector: feedstock
  output: [i_feed, useful]

Feeds_2:
  name: Feeds_2
  description: Conservation cost curve step for industry feedstock demand
  type: useful
  sector: feedstock
  output: [i_feed, useful]

Feeds_3:
  name: Feeds_3
  description: Conservation cost curve step for industry feedstock demand
  type: useful
  sector: feedstock
  output: [i_feed, useful]

Feeds_4:
  name: Feeds_4
  description: Conservation cost curve step for industry feedstock demand
  type: useful
  sector: feedstock
  output: [i_feed, useful]

Feeds_5:
  name: Feeds_5
  description: Conservation cost curve step for industry feedstock demand
  type: useful
  sector: feedstock
  output: [i_feed, useful]

Feeds_con:
  name: Feeds_con
```

(continues on next page)

(continued from previous page)

```
  description: Joint diffusion constraint for feedstock conservation cost curve steps
  type: primary
  sector: feedstock
  output: [dummy agriculture, primary]

coal_fs:
  name: coal_fs
  description: Coal as industry feedstock
  type: useful
  sector: feedstock
  input: [coal, final]
  output: [i_feed, useful]

ethanol_fs:
  name: ethanol_fs
  description: Ethanol as industry feedstock
  type: useful
  sector: feedstock
  input: [ethanol, final]
  output: [i_feed, useful]

foil_fs:
  name: foil_fs
  description: Fuel oil as industry feedstock
  type: useful
  sector: feedstock
  input: [fueloil, final]
  output: [i_feed, useful]

gas_fs:
  name: gas_fs
  description: Gas as industry feedstock
  type: useful
  sector: feedstock
  input: [gas, final]
  output: [i_feed, useful]

loil_fs:
  name: loil_fs
  description: Lightoil as industry feedstock
  type: useful
  sector: feedstock
  input: [lightoil, final]
  output: [i_feed, useful]

methanol_fs:
  name: methanol_fs
  description: Methanol as industry feedstock
  type: useful
  sector: feedstock
  input: [methanol, final]
  output: [i_feed, useful]
```

(continues on next page)

(continued from previous page)

```yaml
coal_gas:
  name: coal_gas
  description: Hard coal gasification
  type: secondary
  vintaged: TRUE
  sector: gas
  input:
    - [coal, secondary]
    - [freshwater_supply, water_supply]
  output: [gas, secondary]

g_ppl_co2scr:
  name: g_ppl_co2scr
  description: CO2 scrubber for natural gas power plant
  type: secondary
  vintaged: TRUE
  sector: gas
  output: [exports, secondary]

gas_bal:
  name: gas_bal
  description: Link technology to stabilize gas production
  type: secondary
  sector: gas
  input: [gas, primary]
  output: [gas, secondary]

gas_bio:
  name: gas_bio
  description: Synthesis gas production from biomass
  type: secondary
  vintaged: TRUE
  sector: gas
  input:
    - [biomass, primary]
    - [electr, secondary]
    - [freshwater_supply, water_supply]
  output: [gas, secondary]

gas_imp:
  name: gas_imp
  description: Piped Gas imports
  type: secondary
  sector: gas
  input: [gas, exports]
  output: [gas, secondary]

gas_rc:
  name: gas_rc
  description: Gas heating in residential/commercial sector
  type: final
```

(continues on next page)

(continued from previous page)

```
  sector: gas
  input: [gas, secondary]
  output: [gas, final]

gas_t/d:
  name: gas_t/d
  description: Transmission/Distribution of gas
  type: final
  vintaged: TRUE
  sector: gas
  input: [gas, secondary]
  output: [gas, final]

gas_t/d_ch4:
  name: gas_t/d_ch4
  description: Transmission/Distribution of gas with CH4 mitigation
  type: final
  vintaged: TRUE
  sector: gas
  input: [gas, secondary]
  output: [gas, final]

gfc_co2scr:
  name: gfc_co2scr
  description: New co2 scrubber for gas fuel cells
  type: secondary
  vintaged: TRUE
  sector: gas
  output: [exports, secondary]

glb_gas_exp:
  name: glb_gas_exp
  description: Global net export of gas
  type: exports
  sector: gas
  output: [gas, exports]

glb_LNG_exp:
  name: glb_LNG_exp
  description: Global net export of liquified natural gas
  type: imports
  sector: gas
  output: [LNG, imports]

h2_mix:
  name: h2_mix
  description: Hydrogen injection into the natural gas system
  type: secondary
  sector: gas
  input: [hydrogen, secondary]
  output: [gas, secondary]
```

(continues on next page)

(continued from previous page)

```
LNG_bal:
  name: LNG_bal
  description: Link technology to stabilize liquified natural gas production
  type: secondary
  sector: gas
  input: [LNG, primary]
  output: [LNG, secondary]

LNG_imp:
  name: LNG_imp
  description: LNG Imports
  type: imports
  sector: gas
  input: [LNG, imports]
  output: [LNG, secondary]

LNG_regas:
  name: LNG_regas
  description: LNG regasification (just link; losses are in trade)
  type: secondary
  vintaged: TRUE
  sector: gas
  input: [LNG, secondary]
  output: [gas, secondary]

bio_hpl:
  name: bio_hpl
  description: Biomass heating plant
  type: secondary
  vintaged: TRUE
  sector: heat
  input:
    - [biomass, primary]
    - [cooling__bio_hpl, cooling]
    - [freshwater_supply, water_supply]
  output: [d_heat, secondary]

coal_hpl:
  name: coal_hpl
  description: Coal heating plant
  type: secondary
  vintaged: TRUE
  sector: heat
  input: [coal, secondary]
  output: [d_heat, secondary]

foil_hpl:
  name: foil_hpl
  description: Fuel oil heating plant
  type: secondary
  vintaged: TRUE
  sector: heat
```

(continues on next page)

(continued from previous page)

```
  input: [fueloil, secondary]
  output: [d_heat, secondary]

gas_hpl:
  name: gas_hpl
  description: Natural gas heating plant
  type: secondary
  vintaged: TRUE
  sector: heat
  input: [gas, secondary]
  output: [d_heat, secondary]

geo_hpl:
  name: geo_hpl
  description: Geothermal heat plant
  type: secondary
  vintaged: TRUE
  sector: heat
  output: [d_heat, secondary]

heat_t/d:
  name: heat_t/d
  description: Transmission/Distribution of district heat
  type: final
  vintaged: TRUE
  sector: heat
  input: [d_heat, secondary]
  output: [d_heat, final]

po_turbine:
  name: po_turbine
  description: Pass out turbine
  type: secondary
  vintaged: TRUE
  sector: heat
  input: [electr, secondary]
  output: [d_heat, secondary]

glb_lh2_imp:
  name: glb_lh2_imp
  description: Global net import of liquid hydrogen
  type: exports
  sector: hydrogen
  input: [liquid hydrogen, exports]
  output: [exports]

h2_bio:
  name: h2_bio
  description: Hydrogen production from biomass with C (via gasification)
  type: secondary
  vintaged: TRUE
  sector: hydrogen
```

(continues on next page)

(continued from previous page)

```
  input:
    - [biomass, primary]
    - [freshwater_supply, water_supply]
  output:
    - [hydrogen, secondary]
    - [electr, secondary]

h2_bio_ccs:
  name: h2_bio_ccs
  description: Hydrogen production from biomass with C (via gasification) with carbon␣
↪capture and storage
  type: secondary
  vintaged: TRUE
  sector: hydrogen
  input:
    - [biomass, primary]
    - [freshwater_supply, water_supply]
  output:
    - [hydrogen, secondary]
    - [electr, secondary]

h2_co2_scrub:
  name: h2_co2_scrub
  description: CO2 scrubber for h2 production from coal and gas
  type: exports
  vintaged: TRUE
  sector: hydrogen
  input: [electr, secondary]
  output: [exports, secondary]

h2_coal:
  name: h2_coal
  description: Hydrogen production via coal gasification
  type: secondary
  vintaged: TRUE
  sector: hydrogen
  input:
    - [coal, secondary]
    - [freshwater_supply, water_supply]
  output:
    - [hydrogen, secondary]
    - [electr, secondary]

h2_coal_ccs:
  name: h2_coal_ccs
  description: Hydrogen production via coal gasification with carbon capture and storage
  type: secondary
  vintaged: TRUE
  sector: hydrogen
  input:
    - [coal, secondary]
    - [freshwater_supply, water_supply]
```

(continues on next page)

(continued from previous page)

```
  output:
    - [hydrogen, secondary]
    - [electr, secondary]

h2_elec:
  name: h2_elec
  description: Hydrogen production via electrolysis
  type: secondary
  vintaged: TRUE
  sector: hydrogen
  input: [electr, secondary]
  output: [hydrogen, secondary]

h2_liq:
  name: h2_liq
  description: Hydrogen liquefaction
  type: primary
  vintaged: TRUE
  sector: hydrogen
  input: [hydrogen, primary]
  output: [liquid hydrogen, primary]

h2_smr:
  name: h2_smr
  description: Hydrogen production via steam-methane reforming of natural gas
  type: secondary
  vintaged: TRUE
  sector: hydrogen
  input:
    - [gas, secondary]
    - [freshwater_supply, water_supply]
  output:
    - [hydrogen, secondary]
    - [electr, secondary]

h2_smr_ccs:
  name: h2_smr_ccs
  description: Hydrogen production via steam-methane reforming of natural gas with␣
↪carbon capture and storage
  type: secondary
  vintaged: TRUE
  sector: hydrogen
  input:
    - [gas, secondary]
    - [freshwater_supply, water_supply]
  output:
    - [hydrogen, secondary]
    - [electr, secondary]

h2_t/d:
  name: h2_t/d
  description: Transmission/Distribution of gaseous hydrogen (just linking technology)
```

(continues on next page)

(continued from previous page)

```
  type: final
  vintaged: TRUE
  sector: hydrogen
  input: [hydrogen, secondary]
  output: [hydrogen, final]

h2b_co2_scrub:
  name: h2b_co2_scrub
  description: CO2 scrubber for h2 production from biomass
  type: secondary
  vintaged: TRUE
  sector: hydrogen
  input: [electr, secondary]
  output: [exports, secondary]

lh2_bal:
  name: lh2_bal
  description: Link technology to stabilize liquid hydrogen production
  type: secondary
  sector: hydrogen
  input: [liquid hydrogen, primary]
  output: [liquid hydrogen, secondary]

lh2_exp:
  name: lh2_exp
  description: Exports of liquid hydrogen
  type: exports
  sector: hydrogen
  input: [liquid hydrogen, primary]
  output: [liquid hydrogen, exports]

lh2_imp:
  name: lh2_imp
  description: Imports of liquid hydrogen
  type: secondary
  sector: hydrogen
  input: [liquid hydrogen, exports]
  output: [liquid hydrogen, secondary]

lh2_regas:
  name: lh2_regas
  description: Regasification of liquid hydrogen
  type: secondary
  vintaged: TRUE
  sector: hydrogen
  input: [liquid hydrogen, secondary]
  output: [hydrogen, secondary]

lh2_t/d:
  name: lh2_t/d
  description: Transmission/Distribution of liquid hydrogen
  type: final
```

(continues on next page)

(continued from previous page)

```
  vintaged: TRUE
  sector: hydrogen
  input: [liquid hydrogen, secondary]
  output: [liquid hydrogen, final]

back_bio_ind:
  name: back_bio_ind
  description: Backstop for diagnosing model infeasibility
  type: final
  sector: industry
  output: [biomass, useful]

back_fs:
  name: back_fs
  description: Backstop for diagnosing model infeasibility
  type: useful
  sector: industry
  output: [i_feed, useful]

back_I:
  name: back_I
  description: Backstop for diagnosing model infeasibility
  type: useful
  sector: industry
  output: [i_spec, useful]

cement_CO2:
  name: cement_CO2
  description: Co2 emissions from cement production
  type: secondary
  sector: industry
  output: [exports, secondary]

cement_co2scr:
  name: cement_co2scr
  description: Cement CO2 scrubber (CCS)
  type: secondary
  vintaged: TRUE
  sector: industry
  output: [exports, secondary]

coal_i:
  name: coal_i
  description: Coal in industry thermal
  type: useful
  vintaged: TRUE
  sector: industry
  input: [coal, final]
  output: [i_therm, useful]

elec_i:
  name: elec_i
```

(continues on next page)

(continued from previous page)

```
  description: Electricity in industry thermal
  type: useful
  vintaged: TRUE
  sector: industry
  input: [electr, final]
  output: [i_therm, useful]

eth_i:
  name: eth_i
  description: Ethanol (without C) replacement for use as liquid fuel in industry thermal
  type: useful
  vintaged: TRUE
  sector: industry
  input: [ethanol, final]
  output: [i_therm, useful]

foil_i:
  name: foil_i
  description: Fuel oil for thermal uses in industry thermal
  type: useful
  vintaged: TRUE
  sector: industry
  input: [fueloil, final]
  output: [i_therm, useful]

gas_i:
  name: gas_i
  description: Gas for thermal uses in industry thermal
  type: useful
  vintaged: TRUE
  sector: industry
  input: [gas, final]
  output: [i_therm, useful]

h2_i:
  name: h2_i
  description: Gaseous hydrogen in industry thermal
  type: useful
  vintaged: TRUE
  sector: industry
  input: [hydrogen, final]
  output: [i_therm, useful]

heat_i:
  name: heat_i
  description: District heating for thermal uses in industry thermal
  type: useful
  vintaged: TRUE
  sector: industry
  input: [d_heat, final]
  output: [i_therm, useful]
```

(continues on next page)

(continued from previous page)

```yaml
hp_el_i:
  name: hp_el_i
  description: Electric heat pump in industry thermal
  type: useful
  vintaged: TRUE
  sector: industry
  input: [electr, final]
  output: [i_therm, useful]

hp_gas_i:
  name: hp_gas_i
  description: Natural gas heat pump in industry thermal
  type: useful
  vintaged: TRUE
  sector: industry
  input: [gas, final]
  output: [i_therm, useful]

loil_i:
  name: loil_i
  description: Lightoil for thermal uses in industry thermal
  type: useful
  vintaged: TRUE
  sector: industry
  input: [lightoil, final]
  output: [i_therm, useful]

meth_i:
  name: meth_i
  description: Methanol (with C) replacement for use as liquid fuel in industry thermal
  type: useful
  vintaged: TRUE
  sector: industry
  input: [methanol, final]
  output: [i_therm, useful]

solar_i:
  name: solar_i
  description: Solar thermal in industry thermal sector
  type: useful
  vintaged: TRUE
  sector: industry
  output: [i_therm, useful]

Ispec_1:
  name: Ispec_1
  description: Conservation cost curve step for industry specific demand
  type: useful
  sector: industry
  output: [i_spec, useful]

Ispec_2:
```

(continues on next page)

(continued from previous page)

```
  name: Ispec_2
  description: Conservation cost curve step for industry specific demand
  type: useful
  sector: industry
  output: [i_spec, useful]

Ispec_3:
  name: Ispec_3
  description: Conservation cost curve step for industry specific demand
  type: useful
  sector: industry
  output: [i_spec, useful]

Ispec_4:
  name: Ispec_4
  description: Conservation cost curve step for industry specific demand
  type: useful
  sector: industry
  output: [i_spec, useful]

Ispec_5:
  name: Ispec_5
  description: Conservation cost curve step for industry specific demand
  type: useful
  sector: industry
  output: [i_spec, useful]

Ispec_con:
  name: Ispec_con
  description: Joint diffusion constraint for industry specific conservation cost curve␣
↪steps
  type: primary
  sector: industry
  output: [dummy agriculture, primary]

Itherm_1:
  name: Itherm_1
  description: Conservation cost curve step for industry thermal demand
  type: useful
  sector: industry
  output: [i_therm, useful]

Itherm_2:
  name: Itherm_2
  description: Conservation cost curve step for industry thermal demand
  type: useful
  sector: industry
  output: [i_therm, useful]

Itherm_3:
  name: Itherm_3
  description: Conservation cost curve step for industry thermal demand
```

(continues on next page)

(continued from previous page)

```
  type: useful
  sector: industry
  output: [i_therm, useful]

Itherm_4:
  name: Itherm_4
  description: Conservation cost curve step for industry thermal demand
  type: useful
  sector: industry
  output: [i_therm, useful]

Itherm_5:
  name: Itherm_5
  description: Conservation cost curve step for industry thermal demand
  type: useful
  sector: industry
  output: [i_therm, useful]

Itherm_con:
  name: Itherm_con
  description: Joint diffusion constraint for industry thermal conservation cost curve␣
↪steps
  type: dummy
  sector: industry
  output: [dummy agriculture, primary]

solar_pv_I:
  name: solar_pv_I
  description: On-site solar photovoltaic power plant (no storage) in industry specific
  type: useful
  vintaged: TRUE
  sector: industry
  output: [i_spec, useful]

h2_fc_I:
  name: h2_fc_I
  description: Hydrogen fuel cell cogeneration system for industry specific
  type: useful
  vintaged: TRUE
  sector: industry
  input: [hydrogen, final]
  output: [i_spec, useful]

sp_coal_I:
  name: sp_coal_I
  description: Specific use of coal in industry
  type: useful
  sector: industry
  input: [coal, final]
  output: [i_spec, useful]

sp_el_I:
```

(continues on next page)

(continued from previous page)

```
  name: sp_el_I
  description: Specific use of electricity in industry
  type: useful
  sector: industry
  input: [electr, final]
  output: [i_spec, useful]

sp_eth_I:
  name: sp_eth_I
  description: Ethanol (without net C) replacement for specific use of light oil in␣
↪industry
  type: useful
  sector: industry
  input: [ethanol, final]
  output: [i_spec, useful]

sp_liq_I:
  name: sp_liq_I
  description: Specific use of light oil in industry
  type: useful
  sector: industry
  input: [lightoil, final]
  output: [i_spec, useful]

sp_meth_I:
  name: sp_meth_I
  description: Methanol (with C) replacement for  specific use of light oil in industry
  type: useful
  sector: industry
  input: [methanol, final]
  output: [i_spec, useful]

bio_extr_mpen:
  name: bio_extr_mpen
  description: Slack primary biomass created with new implementation of non commercial␣
↪biomass
  type: secondary
  sector: land
  output: [exports, secondary]

forest_CO2:
  name: forest_CO2
  description: Co2 emissions from forests
  type: secondary
  sector: land
  output: [exports, secondary]

sinks_1:
  name: sinks_1
  description: Potential for sinks (50. US$)
  type: secondary
  sector: land
```

(continues on next page)

(continued from previous page)

```
  output: [exports, secondary]

sinks_2:
  name: sinks_2
  description: Potential for sinks (100. US$)
  type: secondary
  sector: land
  output: [exports, secondary]

sinks_3:
  name: sinks_3
  description: Potential for sinks (200. US$)
  type: secondary
  sector: land
  output: [exports, secondary]

sinks_4:
  name: sinks_4
  description: Potential for sinks (300. US$)
  type: secondary
  sector: land
  output: [exports, secondary]

eth_bal:
  name: eth_bal
  description: Link technology to stabilize ethanol production
  type: secondary
  sector: liquids
  input: [ethanol, primary]
  output: [ethanol, secondary]

eth_bio:
  name: eth_bio
  description: Ethanol synthesis via biomass gasification
  type: primary
  vintaged: TRUE
  sector: liquids
  input:
    - [biomass, primary]
    - [freshwater_supply, water_supply]
  output:
    - [ethanol, primary]
    - [electr, secondary]

eth_bio_ccs:
  name: eth_bio_ccs
  description: Ethanol synthesis via biomass gasification with carbon capture and storage
  type: primary
  vintaged: TRUE
  sector: liquids
  input:
    - [biomass, primary]
```

(continues on next page)

(continued from previous page)

```
    - [freshwater_supply, water_supply]
  output:
    - [ethanol, primary]
    - [electr, secondary]

eth_exp:
  name: eth_exp
  description: Exports of ethanol (no accounting of CO2 transferred)
  type: exports
  sector: liquids
  input: [ethanol, primary]
  output: [ethanol, exports]

eth_imp:
  name: eth_imp
  description: Imports of ethanol (no accounting of CO2 transferred)
  type: secondary
  sector: liquids
  input: [ethanol, exports]
  output: [ethanol, secondary]

eth_t/d:
  name: eth_t/d
  description: Transmission/Distribution of methanol without net C (just linking␣
↪technology)
  type: final
  sector: liquids
  input: [ethanol, secondary]
  output: [ethanol, final]

foil_exp:
  name: foil_exp
  description: Net exports of crude oil at 95% of oil price
  type: exports
  sector: liquids
  input: [fueloil, secondary]
  output: [fueloil, exports]

foil_imp:
  name: foil_imp
  description: Net imports of residual oil at 95% of oil price
  type: secondary
  sector: liquids
  input: [fueloil, imports]
  output: [fueloil, secondary]

foil_t/d:
  name: foil_t/d
  description: Transmission/Distribution of fueloil  (just linking technology)
  type: final
  sector: liquids
  input: [fueloil, secondary]
```

(continues on next page)

(continued from previous page)

```
  output: [fueloil, final]

glb_eth_imp:
  name: glb_eth_imp
  description: Global net import of ethanol
  type: exports
  sector: liquids
  input: [ethanol, exports]
  output: [exports]

glb_foil_exp:
  name: glb_foil_exp
  description: Global net export of fuel oil
  type: imports
  sector: liquids
  output: [fueloil, imports]

glb_foil_imp:
  name: glb_foil_imp
  description: Global net import of fuel oil
  type: exports
  sector: liquids
  input: [fueloil, exports]
  output: [exports]

glb_loil_exp:
  name: glb_loil_exp
  description: Global net export of light oil
  type: imports
  sector: liquids
  output: [loil, imports]

glb_loil_imp:
  name: glb_loil_imp
  description: Global net import of light oil
  type: exports
  sector: liquids
  input: [loil, exports]
  output: [exports]

glb_meth_imp:
  name: glb_meth_imp
  description: Global net import of methanol
  type: exports
  sector: liquids
  input: [methanol, exports]
  output: [exports]

glb_oil_exp:
  name: glb_oil_exp
  description: Global net export of crude oil
  type: imports
```

(continues on next page)

(continued from previous page)

```
  sector: liquids
  output: [oil, imports]

glb_oil_imp:
  name: glb_oil_imp
  description: Global net import of crude oil
  type: exports
  sector: liquids
  input: [oil, exports]
  output: [exports]

liq_bio:
  name: liq_bio
  description: Second Generation Ethanol Production based on Biomass to FTL
  type: primary
  vintaged: TRUE
  sector: liquids
  input:
    - [biomass, primary]
    - [freshwater_supply, water_supply]
  output:
    - [ethanol, primary]
    - [electr, secondary]

liq_bio_ccs:
  name: liq_bio_ccs
  description: Second Generation Ethanol Production with carbon capture and storage␣
↪based on Biomass to FTL
  type: primary
  vintaged: TRUE
  sector: liquids
  input:
    - [biomass, primary]
    - [freshwater_supply, water_supply]
  output:
    - [ethanol, primary]
    - [electr, secondary]

loil_exp:
  name: loil_exp
  description: Net exports of crude oil at 15% above oil price
  type: exports
  sector: liquids
  input: [lightoil, secondary]
  output: [lightoil, exports]

loil_imp:
  name: loil_imp
  description: Net imports of light oil at 15% above crude price
  type: secondary
  sector: liquids
  input: [lightoil, imports]
```

(continues on next page)

(continued from previous page)

```
  output: [lightoil, secondary]

loil_std:
  name: loil_std
  description: Standard light oil power-plant
  type: secondary
  vintaged: TRUE
  sector: liquids
  output: [lightoil, secondary]

loil_t/d:
  name: loil_t/d
  description: Transmission/Distribution of light oil (just linking technology)
  type: final
  sector: liquids
  input: [lightoil, secondary]
  output: [lightoil, final]

meth_coal:
  name: meth_coal
  description: Methanol synthesis via coal gasification
  type: primary
  vintaged: TRUE
  sector: liquids
  input:
    - [coal, secondary]
    - [freshwater_supply, water_supply]
  output:
    - [methanol, primary]
    - [electr, secondary]

meth_coal_ccs:
  name: meth_coal_ccs
  description: Methanol synthesis via coal gasification with carbon capture and storage
  type: primary
  vintaged: TRUE
  sector: liquids
  input:
    - [coal, secondary]
    - [freshwater_supply, water_supply]
  output:
    - [methanol, primary]
    - [electr, secondary]

meth_exp:
  name: meth_exp
  description: Exports of methanol (no accounting of CO2 transferred)
  type: exports
  sector: liquids
  input: [methanol, primary]
  output: [methanol, exports]
```

(continues on next page)

(continued from previous page)

```yaml
meth_imp:
  name: meth_imp
  description: Imports of methanol (no accounting of CO2 transferred)
  type: secondary
  sector: liquids
  input: [methanol, exports]
  output: [methanol, secondary]

meth_ng:
  name: meth_ng
  description: Methanol synthesis via natural gas
  type: primary
  vintaged: TRUE
  sector: liquids
  input:
    - [gas, secondary]
    - [freshwater_supply, water_supply]
  output: [methanol, primary]

meth_ng_ccs:
  name: meth_ng_ccs
  description: Methanol synthesis via natural gas with carbon capture and storage
  type: primary
  vintaged: TRUE
  sector: liquids
  input:
    - [gas, secondary]
    - [freshwater_supply, water_supply]
  output: [methanol, primary]

meth_t/d:
  name: meth_t/d
  description: Transmission/Distribution of methanol with C
  type: final
  sector: liquids
  input: [methanol, secondary]
  output: [methanol, final]

oil_bal:
  name: oil_bal
  description: Link technology to stabilize crude oil production
  type: secondary
  sector: liquids
  input: [crude oil, primary]
  output: [crude oil, secondary]

oil_exp:
  name: oil_exp
  description: Net exports of crude oil at 100% of oil price
  type: exports
  sector: liquids
  input: [crude oil, primary]
```

(continues on next page)

(continued from previous page)

```
  output: [oil, exports]

oil_imp:
  name: oil_imp
  description: Net imports of oil
  type: secondary
  sector: liquids
  input: [oil, imports]
  output: [crude oil, secondary]

plutonium_prod:
  name: plutonium_prod
  description: Plutonium production
  type: stocks
  sector: liquids
  input: [plutonium, stocks]
  output: [plutonium, stocks]

ref_hil:
  name: ref_hil
  description: New deeply upgraded refineries
  type: secondary
  vintaged: TRUE
  sector: liquids
  input: [crude oil, secondary]
  output: [fueloil, secondary]

ref_lol:
  name: ref_lol
  description: Existing refineries (low yield)
  type: secondary
  vintaged: TRUE
  sector: liquids
  input: [crude oil, secondary]
  output: [fueloil, secondary]

SO2_scrub_ref:
  name: SO2_scrub_ref
  description: SO2 scrubber for refineries
  type: secondary
  vintaged: TRUE
  sector: liquids
  output: [exports, secondary]

syn_liq:
  name: syn_liq
  description: Coal liquefaction and light oil synthesis
  type: secondary
  vintaged: TRUE
  sector: liquids
  input:
    - [coal, secondary]
```

(continues on next page)

(continued from previous page)

```
    - [freshwater_supply, water_supply]
  output:
    - [lightoil, secondary]
    - [electr, secondary]

syn_liq_ccs:
  name: syn_liq_ccs
  description: Coal liquefaction and light oil synthesis with carbon capture and storage
  type: secondary
  vintaged: TRUE
  sector: liquids
  input:
    - [coal, secondary]
    - [freshwater_supply, water_supply]
  output:
    - [lightoil, secondary]
    - [electr, secondary]

adipic_thermal:
  name: adipic_thermal
  description: Thermal destruction tech for adipic acid sector
  type: dummy
  sector: non-co2
  output: [dummy, primary]

ammonia_secloop:
  name: ammonia_secloop
  description: Ammonia Secondary Loop Systems
  type: dummy
  sector: non-co2
  output: [dummy, primary]

enre_con:
  name: enre_con
  description: Joint diffusion constraint for enteric fermentation mitigation␣
↪technologies
  type: dummy
  sector: non-co2
  output: [dummy agriculture, primary]

ent_red1:
  name: ent_red1
  description: Mitigation for CH4 emissions from animals directly
  type: dummy
  sector: non-co2
  output: [dummy agriculture, primary]

ent_red2:
  name: ent_red2
  description: Mitigation for CH4 emissions from animals directly
  type: dummy
  sector: non-co2
```

(continues on next page)

(continued from previous page)

```
  output: [dummy agriculture, primary]

ent_red3:
  name: ent_red3
  description: Mitigation for CH4 emissions from animals directly
  type: dummy
  sector: non-co2
  output: [dummy agriculture, primary]

landfill_compost1:
  name: landfill_compost1
  description: Landfill mitigation technology (composting) technology 1 for CH4␣
↪mitigation
  type: dummy
  vintaged: TRUE
  sector: non-co2
  output: [dummy, primary]

landfill_compost2:
  name: landfill_compost2
  description: Landfill mitigation technology (composting) technology 2 for CH4␣
↪mitigation
  type: dummy
  vintaged: TRUE
  sector: non-co2
  output: [dummy, primary]

landfill_direct1:
  name: landfill_direct1
  description: Landfill mitigation technology (direct) 1 for CH4 mitigation
  type: dummy
  vintaged: TRUE
  sector: non-co2
  output: [dummy, primary]

landfill_direct2:
  name: landfill_direct2
  description: Landfill mitigation technology (direct) 1 for CH4 mitigation
  type: dummy
  vintaged: TRUE
  sector: non-co2
  output: [dummy, primary]

landfill_ele:
  name: landfill_ele
  description: Landfill mitigation technology (ele) 1 for CH4 mitigation
  type: dummy
  vintaged: TRUE
  sector: non-co2
  output: [dummy, primary]

landfill_flaring:
```

(continues on next page)

(continued from previous page)

```
  name: landfill_flaring
  description: Landfill mitigation technology (flaring) 1 for CH4 mitigation
  type: dummy
  vintaged: TRUE
  sector: non-co2
  output: [dummy, primary]

landfill_heatprdn:
  name: landfill_heatprdn
  description: Landfill mitigation technology (heat production) 1 for CH4 mitigation
  type: dummy
  vintaged: TRUE
  sector: non-co2
  output: [dummy, primary]

landfill_oxdn:
  name: landfill_oxdn
  description: Landfill mitigation technology (oxidation) 1 for CH4 mitigation
  type: dummy
  vintaged: TRUE
  sector: non-co2
  output: [dummy, primary]

leak_repair:
  name: leak_repair
  description: Leak-repairs, HFC-134a from refrigeration & AC(SR)
  type: dummy
  sector: non-co2
  output: [dummy, primary]

leak_repairsf6:
  name: leak_repairsf6
  description: Recycling gas carts for SF6 recovery during assembly of gas insulated␣
↪equipment
  type: dummy
  sector: non-co2
  output: [dummy, primary]

lfil_con:
  name: lfil_con
  description: Joint diffusion constraint for landfill mitigation technologies
  type: dummy
  sector: non-co2
  output: [dummy agriculture, primary]

manu_con:
  name: manu_con
  description: Joint diffusion constraint for manure management+B249 mitigation␣
↪technologies
  type: primary
  sector: non-co2
  output: [dummy agriculture, primary]
```

(continues on next page)

(continued from previous page)

```yaml
meth_bal:
  name: meth_bal
  description: Link technology to stabilize methanol production
  type: primary
  sector: non-co2
  input: [methanol, primary]
  output: [methanol, secondary]

mvac_co2:
  name: mvac_co2
  description: Transcritical vapor cycle CO2 systems for mobile vehicle air conditioners
  type: dummy
  sector: non-co2
  output: [dummy]

recycling_gas1:
  name: recycling_gas1
  description: Recycling gas carts for SF6 recovery during maintenance of gas insulated␣
↪equipment(SR)
  type: dummy
  sector: non-co2
  output: [dummy, primary]

refrigerant_recover:
  name: refrigerant_recover
  description: Recovery of refrigerant, HFC-134a from refrigeration and AC (SR)
  type: primary
  sector: non-co2
  output: [dummy, primary]

repl_hc:
  name: repl_hc
  description: Replacement with HC for foams
  type: primary
  sector: non-co2
  output: [dummy, primary]

replacement_so2:
  name: replacement_so2
  description: Replacing SF6 by SO2(SR)
  type: primary
  sector: non-co2
  output: [dummy, primary]

rice_red1:
  name: rice_red1
  description: Mitigation for CH4 emissions from rice
  type: primary
  sector: non-co2
  output: [dummy agriculture, primary]
```

(continues on next page)

(continued from previous page)

```
rice_red2:
  name: rice_red2
  description: Mitigation for CH4 emissions from rice
  type: primary
  sector: non-co2
  output: [dummy agriculture, primary]

rice_red3:
  name: rice_red3
  description: Mitigation for CH4 emissions from rice
  type: primary
  sector: non-co2
  output: [dummy agriculture, primary]

rire_con:
  name: rire_con
  description: Joint diffusion constraint for rice cultivation mitigation technologies
  type: primary
  sector: non-co2
  output: [dummy  agriculture, primary]

soil_red1:
  name: soil_red1
  description: Mitigation for N2Oemissions from soil
  type: primary
  sector: non-co2
  output: [dummy agriculture, primary]

soil_red2:
  name: soil_red2
  description: Mitigation for N2Oemissions from soil
  type: primary
  sector: non-co2
  output: [dummy agriculture, primary]

soil_red3:
  name: soil_red3
  description: Mitigation for N2Oemissions from soil
  type: primary
  sector: non-co2
  output: [dummy agriculture, primary]

vertical_stud:
  name: vertical_stud
  description: Soderberg process for CF4 from aluminum (SR)
  type: dummy
  sector: non-co2
  output: [dummy, primary]

u5-reproc:
  name: u5-reproc
  description: Uranium reprocessing
```

(continues on next page)

(continued from previous page)

```
  type: stocks
  vintaged: TRUE
  sector: nuclear
  input: [uranium, stocks]
  output: [uranium, stocks]

Uran_extr:
  name: Uran_extr
  description: Uranium extraction, milling for FBR blanket
  type: stocks
  sector: nuclear
  input: [uranium, resource]
  output: [uranium, stocks]

uran2u5:
  name: uran2u5
  description: Uranium extraction, milling, fluorination and enrichment per t U5
  type: stocks
  sector: nuclear
  input: [uranium, resource]
  output: [uranium, stocks]

bco2_tr_dis:
  name: bco2_tr_dis
  description: Technology for co2 transportation and disposal from biomass technologies
  type: secondary
  sector: other conversion
  output: [exports, secondary]

co2_tr_dis:
  name: co2_tr_dis
  description: Technology for co2 transportation and disposal
  type: secondary
  sector: other conversion
  output: [exports, secondary]

back_RC:
  name: back_RC
  description: Backstop for diagnosing model infeasibility
  type: useful
  sector: residential/commercial
  output: [rc_spec, useful]

solar_rc:
  name: solar_rc
  description: Solar thermal in residential/commercial sector
  type: useful
  sector: residential/commercial
  output: [rc_therm, useful]

biomass_rc:
  name: biomass_rc
```

(continues on next page)

(continued from previous page)

```
  description: Biomass with C for heating in residential/commercial sector
  type: useful
  sector: residential/commercial
  input: [biomass, final]
  output: [rc_therm, useful]

coal_rc:
  name: coal_rc
  description: Coal heating in residential/commercial sector
  type: useful
  sector: residential/commercial
  input: [coal, final]
  output: [rc_therm, useful]

elec_rc:
  name: elec_rc
  description: Electricity heating in residential/commercial sector
  type: useful
  sector: residential/commercial
  input: [electr, final]
  output: [rc_therm, useful]

eth_rc:
  name: eth_rc
  description: Ethanol (without C) replacement for use as liquid fuel in residential/
↪commercial
  type: useful
  sector: residential/commercial
  input: [ethanol, final]
  output: [rc_therm, useful]

foil_rc:
  name: foil_rc
  description: Fuel oil heating in residential/commercial sector
  type: useful
  sector: residential/commercial
  input: [fueloil, final]
  output: [rc_therm, useful]

h2_rc:
  name: h2_rc
  description: Hydrogen (gaseous) catalytic heating in
  type: secondary
  sector: residential/commercial
  input: [hydrogen, final]
  output: [rc_therm, useful]

heat_rc:
  name: heat_rc
  description: District heating in residential/commercial sector
  type: useful
  sector: residential/commercial
```

(continues on next page)

(continued from previous page)

```
  input: [d_heat, final]
  output: [rc_therm, useful]

hp_el_rc:
  name: hp_el_rc
  description: Electric heat pump in residential/commercial sector
  type: useful
  sector: residential/commercial
  input: [electr, final]
  output: [rc_therm, useful]

hp_gas_rc:
  name: hp_gas_rc
  description: Natural gas heat pump in residential/commercial sector
  type: useful
  sector: residential/commercial
  input: [gas, final]
  output: [rc_therm, useful]

loil_rc:
  name: loil_rc
  description: Lightoil heating in residential/commercial sector
  type: useful
  sector: residential/commercial
  input: [lightoil, final]
  output: [rc_therm, useful]

meth_rc:
  name: meth_rc
  description: Methanol (with C) replacement for use as liquid fuel in residential/
↪commercial
  type: useful
  sector: residential/commercial
  input: [methanol, final]
  output: [rc_therm, useful]

RCspec_1:
  name: RCspec_1
  description: Conservation cost curve step for residential/commercial specific demand
  type: useful
  sector: residential/commercial
  output: [useful]

RCspec_2:
  name: RCspec_2
  description: Conservation cost curve step for residential/commercial specific demand
  type: useful
  sector: residential/commercial
  output: [useful]

RCspec_3:
  name: RCspec_3
```

(continues on next page)

(continued from previous page)

```
  description: Conservation cost curve step for residential/commercial specific demand
  type: useful
  sector: residential/commercial
  output: [useful]

RCspec_4:
  name: RCspec_4
  description: Conservation cost curve step for residential/commercial specific demand
  type: useful
  sector: residential/commercial
  output: [useful]

RCspec_5:
  name: RCspec_5
  description: Conservation cost curve step for residential/commercial specific demand
  type: useful
  sector: residential/commercial
  output: [useful]

RCspec_con:
  name: RCspec_con
  description: Joint diffusion constraint for residential/commercial specific␣
↪conservation cost curve steps
  type: dummy
  sector: residential/commercial
  output: [dummy agriculture, primary]

RCtherm_1:
  name: RCtherm_1
  description: Conservation cost curve step for residential/commercial thermal demand
  type: useful
  sector: residential/commercial
  output: [rc_therm, useful]

RCtherm_2:
  name: RCtherm_2
  description: Conservation cost curve step for residential/commercial thermal demand
  type: useful
  sector: residential/commercial
  output: [rc_therm, useful]

RCtherm_3:
  name: RCtherm_3
  description: Conservation cost curve step for residential/commercial thermal demand
  type: useful
  sector: residential/commercial
  output: [rc_therm, useful]

RCtherm_4:
  name: RCtherm_4
  description: Conservation cost curve step for residential/commercial thermal demand
  type: useful
```

(continues on next page)

(continued from previous page)

```
  sector: residential/commercial
  output: [rc_therm, useful]

RCtherm_5:
  name: RCtherm_5
  description: Conservation cost curve step for residential/commercial thermal demand
  type: useful
  sector: residential/commercial
  output: [rc_therm, useful]

RCtherm_con:
  name: RCtherm_con
  description: Joint diffusion constraint for residential/commercial thermal␣
↪conservation cost curve steps
  type: dummy
  sector: residential/commercial
  output: [dummy agriculture, primary]

solar_pv_RC:
  name: solar_pv_RC
  description: Specific use of on-site solar photovoltaic power plant (no storage) in␣
↪residential/commercial
  type: useful
  vintaged: TRUE
  sector: residential/commercial
  output: [rc_spec, useful]

sp_el_RC:
  name: sp_el_RC
  description: Specific use of electricity in residential/commercial
  type: useful
  sector: residential/commercial
  input: [electr, final]
  output: [rc_spec, useful]

h2_fc_RC:
  name: h2_fc_RC
  description: Specific use of hydrogen fuel cell cogeneration system in res/comm
  type: useful
  vintaged: TRUE
  sector: residential/commercial
  input: [hydrogen, final]
  output: [rc_spec, useful]

bio_extr_chp:
  name: bio_extr_chp
  description: Supply of biomass without net C emissions;  defined here as agricultural␣
↪wastes, and other crops with cycle of 1 year or less
  type: primary
  sector: solids
  output: [biomass, primary]
```

(continues on next page)

(continued from previous page)

```
bio_ppl_co2scr:
  name: bio_ppl_co2scr
  description: New coal scrubber for power plants
  type: secondary
  vintaged: TRUE
  sector: solids
  output: [exports, secondary]

biomass_i:
  name: biomass_i
  description: Biomass with C in industry thermal
  type: useful
  vintaged: TRUE
  sector: solids
  input: [biomass, final]
  output: [i_therm, useful]

biomass_nc:
  name: biomass_nc
  description: Non-commercial biomass with C
  type: useful
  vintaged: TRUE
  sector: solids
  input: [biomass, primary]
  output: [non-comm, useful]

biomass_t/d:
  name: biomass_t/d
  description: Transmission/Distribution of biomass with C
  type: useful
  sector: solids
  input: [biomass, primary]
  output: [biomass, final]

c_ppl_co2scr:
  name: c_ppl_co2scr
  description: New coal scrubber for coal power plants
  type: secondary
  vintaged: TRUE
  sector: solids
  output: [exports, secondary]

cfc_co2scr:
  name: cfc_co2scr
  description: Co2 scrubber for coal fuel cells (CCS)
  type: secondary
  vintaged: TRUE
  sector: solids
  output: [exports, secondary]

coal_bal:
  name: coal_bal
```

(continues on next page)

(continued from previous page)

```
  description: Link technology to stabilize coal production
  type: secondary
  sector: solids
  input: [coal, primary]
  output: [coal, secondary]

coal_exp:
  name: coal_exp
  description: Net exports of coal at fixed price of about US$ 42/tce
  type: exports
  sector: solids
  input: [coal, primary]
  output: [coal, exports]

coal_imp:
  name: coal_imp
  description: Net imports of coal
  type: secondary
  sector: solids
  input: [coal, imports]
  output: [coal, secondary]

coal_t/d:
  name: coal_t/d
  description: Transmission/Distribution of coal
  type: final
  sector: solids
  input: [coal, secondary]
  output: [coal, final]

coal_t/d-in-06%:
  name: coal_t/d-in-06%
  description: Transmission/Distribution of coal industry like coal_t/d but imported␣
↪coal  with 0.6% S content
  type: final
  sector: solids
  input: [coal, secondary]
  output: [coal, final]

coal_t/d-in-SO2:
  name: coal_t/d-in-SO2
  description: Transmission/Distribution of coal industry like coal_t/d but processed␣
↪coal
  type: final
  sector: solids
  input: [coal, secondary]
  output: [coal, final]

coal_t/d-rc-06%:
  name: coal_t/d-rc-06%
  description: Transmission/Distribution of coal residential/commercial like coal_t/d␣
↪but imported coal with 0.6% S content
```

(continues on next page)

(continued from previous page)

```
  type: final
  sector: solids
  input: [coal, secondary]
  output: [coal, final]

coal_t/d-rc-SO2:
  name: coal_t/d-rc-SO2
  description: Transmission/Distribution of coal like coal_t/d but processed coal
  type: final
  sector: solids
  input: [coal, secondary]
  output: [coal, final]

glb_coal_exp:
  name: glb_coal_exp
  description: Global net export of coal
  type: imports
  sector: solids
  output: [coal, imports]

glb_coal_imp:
  name: glb_coal_imp
  description: Global net import of coal
  type: exports
  sector: solids
  input: [electr, exports]
  output: [exports]

back_trp:
  name: back_trp
  description: Backstop for diagnosing model infeasibility
  type: useful
  sector: transport
  output: [transport, useful]

coal_trp:
  name: coal_trp
  description: Coal-based transport
  type: useful
  sector: transport
  input: [coal, final]
  output: [transport, useful]

elec_trp:
  name: elec_trp
  description: Electricity-based transport
  type: useful
  sector: transport
  input: [electr, final]
  output: [transport, useful]

eth_fc_trp:
```

(continues on next page)

(continued from previous page)

```
  name: eth_fc_trp
  description: Ethanol (without net C) fuel cell-based transport
  type: useful
  sector: transport
  input: [ethanol, final]
  output: [transport, useful]

eth_ic_trp:
  name: eth_ic_trp
  description: Ethanol (without net C) ic-engine-based transport
  type: useful
  sector: transport
  input: [ethanol, final]
  output: [transport, useful]

foil_trp:
  name: foil_trp
  description: Fueloil-based transport
  type: useful
  sector: transport
  input: [fueloil, final]
  output: [transport, useful]

gas_trp:
  name: gas_trp
  description: Gas-based transport
  type: useful
  sector: transport
  input: [gas, final]
  output: [transport, useful]

h2_fc_trp:
  name: h2_fc_trp
  description: Hydrogen fuel cell-based transport (plus off-hours electricity generation)
  type: useful
  vintaged: TRUE
  sector: transport
  input: [liquid hydrogen, final]
  output: [transport, useful]

loil_trp:
  name: loil_trp
  description: Lightoil-based transport
  type: useful
  sector: transport
  input: [lightoil, final]
  output: [transport, useful]

meth_fc_trp:
  name: meth_fc_trp
  description: Methanol (with C) fuel cell-based transport
  type: useful
```

(continues on next page)

(continued from previous page)

```
  sector: transport
  input: [methanol, final]
  output: [transport, useful]

meth_ic_trp:
  name: meth_ic_trp
  description: Methanol (with C) ic-engine-based transport
  type: useful
  sector: transport
  input: [methanol, final]
  output: [transport, useful]

Trans_1:
  name: Trans_1
  description: Conservation cost curve step for transport demand
  type: useful
  sector: transport
  output: [transport, useful]

Trans_2:
  name: Trans_2
  description: Conservation cost curve step for transport demand
  type: useful
  sector: transport
  output: [transport, useful]

Trans_3:
  name: Trans_3
  description: Conservation cost curve step for transport demand
  type: useful
  sector: transport
  output: [transport, useful]

Trans_4:
  name: Trans_4
  description: Conservation cost curve step for transport demand
  type: useful
  sector: transport
  output: [transport, useful]

Trans_5:
  name: Trans_5
  description: Conservation cost curve step for transport demand
  type: useful
  sector: transport
  output: [transport, useful]

Trans_con:
  name: Trans_con
  description: Joint diffusion constraint for transport conservation cost curve steps
  type: primary
  sector: transport
```

(continues on next page)

(continued from previous page)

```
  output: [dummy agriculture, primary]

foil_bunker:
  name: foil_bunker
  description: Fuel oil demand for international shipping bunkers (global trade)
  type: useful
  sector: shipping
  input: [fueloil, final]
  output: [shipping, useful]

loil_bunker:
  name: loil_bunker
  description: Light oil demand for international shipping bunkers (global trade)
  type: useful
  sector: shipping
  input: [lightoil, final]
  output: [shipping, useful]

meth_bunker:
  name: meth_bunker
  description: Methanol demand for international shipping bunkers (global trade)
  type: useful
  sector: shipping
  input: [methanol, final]
  output: [shipping, useful]

eth_bunker:
  name: eth_bunker
  description: Ethanol demand for international shipping bunkers (global trade)
  type: useful
  sector: shipping
  input: [ethanol, final]
  output: [shipping, useful]

LNG_bunker:
  name: LNG_bunker
  description: Liquified natural gas demand for international shipping bunkers (global␣
↪trade)
  type: useful
  sector: shipping
  input: [LNG, final]
  output: [shipping, useful]

LH2_bunker:
  name: LH2_bunker
  description: Liquified hydrogen  demand for international shipping bunkers (global␣
↪trade)
  type: useful
  sector: shipping
  input: [lh2, final]
  output: [shipping, useful]
```

CHAPTER

NINETEEN

DEVELOPMENT PRACTICES

This page describes development practices for `message_ix_models` and `message_data` intended to help reproducibility, interoperability, and reusability.

In the following, the bold-face words **required**, **optional**, etc. have specific meanings as described in IETF RFC 2119.

On other pages:

- Contributing to development in the MESSAGEix docs. *All* of these apply to contributions to `message_ix_models` and `message_data`, including the Code style.
- *Data, metadata, and configuration*, for how to add and handle these.

On this page:

19.1 Advertise and check compatibility

There are multiple choices of the base structure for a model in the MESSAGEix-GLOBIOM family, e.g. different *Node code lists* and *Years or time periods (year/*.yaml)*.

Code that will only work with certain structures…

- **must** be documented, and include in its documentation any such limitation, e.g. "`example_func()` only produces data for R11 and year list B."
- **should** use *`check_support()`* in individual pieces of code to pre-emptively check and raise an exception. This prevents inadvertent use of the code where its data will be invalid:

```
def myfunc(context, *args):
    """A function that only works on R11 and years 'B'."""

    check_support(
        context,
        dict(regions=["R11"], years=["B"]),
        "Example data produced"
    )

    # ... function code to execute if the check passes
```

Code **may** also check a *Context* instance and automatically adapt data from certain structures to others, e.g. by interpolating data for certain periods or areas. To help with validation, code that does this **should** log on the `logging.INFO` level to advertise these steps.

CHAPTER

TWENTY

WHAT'S NEW

20.1 2022.7.25

- Add `get_advance_data()`, and related tools for data from the ADVANCE project, including the *node codelist* for the data (PR #76).
- Add unit annotations to *Commodities (commodity.yaml)* (PR #76).
- New utility methods *`ScenarioInfo.io_units()`* to derive units for `input` and `output` parameters from *`units_for()`* commodity stocks and technology activities (PR #76).
- Transfer *`add_tax_emission()`* from `message_data`, improve, and add tests (PR #76).
- Unit annotations on commodity and technology codes are copied to child codes using *`process_units_anno()`* (PR #76).
- `make_matched_dfs()` accepts `pint.Quantity` to set both magnitude and units in generated data (PR #76).
- `strip_par_data()` also removes the set element for which data is being stripped (PR #76).
- The common CLI options **`--verbose`** and **`--dry-run`** are stored on *`Context`* automatically (PR #76).
- New utility method *`Context.set_scenario()`* (PR #76).
- `iam_units.registry` is the default unit registry even when `message_data` is not installed (PR #76).
- Expand *`broadcast()`* to allow `DataFrame` with multiple dimensions as input (PR #74).

20.2 2022.5.6

- Bump minimum required version of `message_ix` to v3.4.0 from v3.2.0 (PR #71).
- Add a documentation page on *Distributed computing* (PR #59).
- Add *`testing.not_ci()`* for marking tests not to be run on continuous integration services; improve *`session_context()`* (PR #62).
- *`apply_spec()`* also adds elements of the "node" set using `ixmp.Platform.add_region()` (PR #62).
- Add new logo the documentation (PR #68).
- Add *`Workflow`*; see *Multi-scenario workflows (workflow)* (PR #60).

20.3 2022.3.30

- Add `adapt_R11_R12()`, a function for adapting data from the *11-region aggregation (R11)* to the *12-region aggregation (R12)* node lists (PR #56).
- Work around iiasa/ixmp#425 in *`disutility.data_conversion()`* (*docs*, PR #55).

20.4 2022.3.3

- Change the node name in R12.yaml from R12_CPA to R12_RCPA (PR #49).
- Register "message local data" ixmp configuration file setting and use to set the `Context.local_path` when provided. See *(3) Other, system-specific ("local") directories* (PR #47)

20.5 2022.1.26

- New *`Spec`* class for easier handling of specifications of model (or model variant) structure (PR #39)
- New utility function *`util.local_data_path()`* (PR #39).
- `repr()` of *`Context`* no longer prints a (potentially very long) list of all keys and settings (PR #39).
- *`as_codes()`* accepts a `dict` with `Code` values (PR #39).

20.6 Earlier releases

20.6.1 2021.11.24

- Add **`--years`** and **`--nodes`** to *`common_params()`* (PR #35).
- New utility function *`structure.codelists()`* (PR #35).

20.6.2 2021.7.27

- Improve caching using mod:*genno* v1.8.0 (PR #29).

20.6.3 2021.7.22

- Migrate utilities `cached()`, *`check_support()`*, *`convert_units()`*, *`maybe_query()`*, *`series_of_pint_quantity()`* (PR #27)
- Add *`testing.NIE`*.
- Add the `--jvmargs` option to **`pytest`** (see *`pytest_addoption()`*).
- Remove `Context.get_config_file()`, `get_path()`, `load_config()`, and `units()`, all deprecated since 2021-02-28.

20.6.4 2021.7.6

- Add `identify_nodes()`, a function for identifying a *Node code lists* based on a `Scenario` (PR #24).
- Add `adapt_R11_R14()`, a function for adapting data from the *11-region aggregation (R11)* to the *14-region aggregation (R14)* node lists (PR #24).
- Add *`export_test_data()`* and **`mix-models export-test-data`** command (PR #16). See *Prepare data for testing*.
- Allow use of pytest's persistent cache across test sessions (PR #23). See *Reproducibility*.
- Add the *12-region aggregation (R12)* node code list (PR #14).

20.6.5 2021.4.7

- Add *`model.disutility`*, code for setting up structure and data for generalized consumer disutility (PR #13)

20.6.6 2021.3.24

- Add *Years or time periods (year/*.yaml)*, YAML data files, *`ScenarioInfo.year_from_codes()`* and associated tests (GH #11, PR #12)

20.6.7 2021.3.22

- Migrate *`model.bare`*, *`model.build`*, `model.cli`, and associated documentation (PR #9)
- Migrate utilities: *`ScenarioInfo`*, `add_par_data()`, *`eval_anno()`*, `iter_parameters()`, and `strip_par_data()`.

20.6.8 2021.3.3

- Migrate *`util.click`*, `util.logging`; expand documentation (PR #8:).
- *`Context.clone_to_dest()`* method replaces `clone_to_dest()` function.
- Build PDF documentation on ReadTheDocs.
- Allow CLI commands from both `message_ix_models` and `message_data` via **`mix-models`**.
- Migrate **`mix-models techs`** CLI command.

20.6.9 2021.2.28

- Migrate *`Context`* class and *`testing`* module from `message_data` (PR #5:).
- Add *`load_private_data()`*, *`package_data_path()`*, *`private_data_path()`*.
- Document: *Data, metadata, and configuration* and *Command-line interface*.
- Update *node codelists* to ensure they contain both current and former ISO 3166 codes for countries that have changed status (PR #6:). For instance, ANT dissolved into BES, CUW, and SXM in 2010; all four are included in R11_LAM so this list can be used to handle data from either before or after 2010.

20.6.10 2021.2.26

- Add `get_codes()` and related code lists (PR #2:).
- Add `MessageDataFinder` and document *Migrating from message_data* (PR #3:).

20.6.11 2021.2.23

Initial release.

CHAPTER

TWENTYONE

MIGRATING FROM `message_data`

`message_ix_models` coexists with the private repository/package currently named `message_data`. The latter is the location for code related to new research that has not yet been completed and published, data that must remain closed-source permanently, etc.

Over time:

- All other code will be migrated from `message_data` to `message_ix_models`.
- Code and data for individual projects will be moved from `message_data` to `message_ix_models` at a suitable point during the process of publication. (This point may vary from project to project.)
- `message_data` may be renamed.

This page gives some practices and tips for using the two packages together.

Always import via `message_ix_models`

The package installs *`MessageDataFinder`* into Python's import system (importlib), which changes its default behaviour as follows: if

1. A module `message_ix_models.model.`*`model_name`* or `message_ix_models.project.`*`project_name`* is imported, and
2. This module does not actually exist in `message_ix_models`,
3. Then the code will instead file the respective modules `message_data.model.`*`model_name`* or `message_data.project.`*`project_name`*.

Even when using code that currently or temporarily lives in `message_data`, access it like this:

```
# Code in message_data/model/mymodelvariant.py
from message_ix_models.model import mymodelvariant

mymodelvariant.build(...)
```

This code is *future-proof*: it will not need adjustment if/when "mymodelvariant" is eventually moved from `message_data` to `message_ix_models`.

Use the `mix-models` command-line interface (CLI)

All CLI commands and subcommands defined in `message_data` are also made available through the `message_ix_models` CLI, the executable **`mix-models`**.

Use this program in documentation examples and in scripts. In a similar manner to the point above, these documents and scripts will remain correct if/when code is moved.

Don't import from `message_data` in `message_ix_models`

The open-source code should not depend on any private code. If this appears necessary, the code in `message_data` can probably be moved to `message_ix_models`.

Use *message_ix_models.tools* and *util* in message_data

The former have stricter quality standards and are more transparent, which is better for reproducibility.

At some points, similar code may appear in both packages as it is being migrated. In such cases, always import and use the code in `message_ix_models`, making any adjustments that are necessary.

CHAPTER

TWENTYTWO

RELEASING

22.1 Version numbers

`message_ix_models` uses date-based version numbers like `Y.M.D`. Thus version `2021.2.23` is released on 23 February 2021. This is to establish a more direct correspondence between outputs of the code and the version(s) used to produce it.

22.2 Procedure

Before releasing, check:

- https://github.com/iiasa/message-ix-models/actions?query=branch:main to ensure that the push and scheduled builds are passing.
- https://readthedocs.com/projects/iiasa-energy-program-message-ix-models/builds/ to ensure that the docs build is passing.

Address any failures before releasing.

1. Edit `doc/whatsnew.rst`. Comment the heading "Next release", then insert another heading below it, at the same level, with the version number and date. Make a commit with a message like "Mark vX.Y.Z in doc/whatsnew".
2. Tag the release candidate version, i.e. with a `rcN` suffix, and push:

   ```
   $ git tag v1.2.3rc1
   $ git push --tags origin main
   ```

3. Check:
   - at https://github.com/iiasa/message-ix-models/actions?query=workflow:publish that the workflow completes: the package builds successfully and is published to TestPyPI.
   - at https://test.pypi.org/project/message-ix-models/ that:
     - The package can be downloaded, installed and run.
     - The README is rendered correctly.

   Address any warnings or errors that appear. If needed, make a new commit and go back to step (2), incrementing the rc number.
4. (optional) Tag the release itself and push:

```
$ git tag v1.2.3
$ git push --tags origin main
```

This step (but *not* step (2)) can also be performed directly on GitHub; see (5), next.

5. Visit https://github.com/iiasa/message-ix-models/releases and mark the new release: either using the pushed tag from (4), or by creating the tag and release simultaneously.
6. Check at https://github.com/iiasa/message-ix-models/actions?query=workflow:publish and https://pypi.org/project/message-ix-models/ that the distributions are published.

CHAPTER

TWENTYTHREE

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

m

INDEX

Symbols

A

B

C

D

E

F

G

H

I

L

M

N

O

P

R

S

T

U

W

Y