Futura

Release 0.0.0

Mar 28, 2022

Contents:

CHAPTER 1

Futura

Futura is a software interface to generate arbitrary future background databases for LCA sensitivity analyses.

1.1 Introduction

The world of 2050 will be very different to the world of 2019.

For emerging technologies and long-lived products, decisions made today will have a direct effect on the environmental impact which occurs in the future. While it is impossible to accurately predict what the world will look like in the future, it is possible come up with sensible assumptions representing a range of possible future scenarios.

In order for the future impact of products to be estimated in a consistent and meaningful manner in LCA, the background system - most commonly the ecoinvent database - needs to be projected into the future alongside the foreground system modelled in a given study. This is not a trivial task. The latest version of the ecoinvent database contains over 18,000 interlinked processes. External factors, such as technology availability and electricity grid mixes, determined by future scenarios have the potential to affect every single one of these processes.

Recent developments in advanced open-source LCA software have opened up the opportunity to apply database wide changes to modelling assumptions. This technique can be extended to alter the entire database to reflect future scenarios in a systematic way.

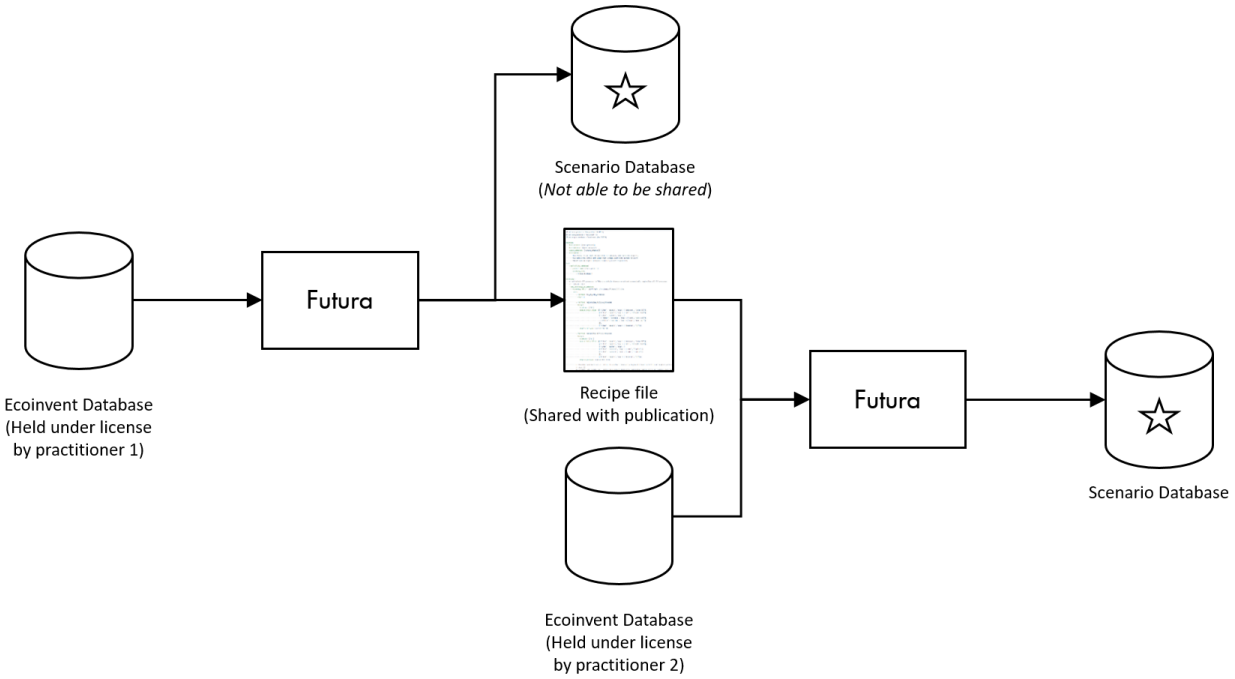
Futura is a new piece of open-source software which allows LCA practitioners to create and, importantly, **share** novel background databases representing arbitrary scenarios.

It allows users to import a base database and then start making targeted changes. These changes take three main forms:

- adding new technologies
- regionalising new or existing technologies, and
- altering market compositions.

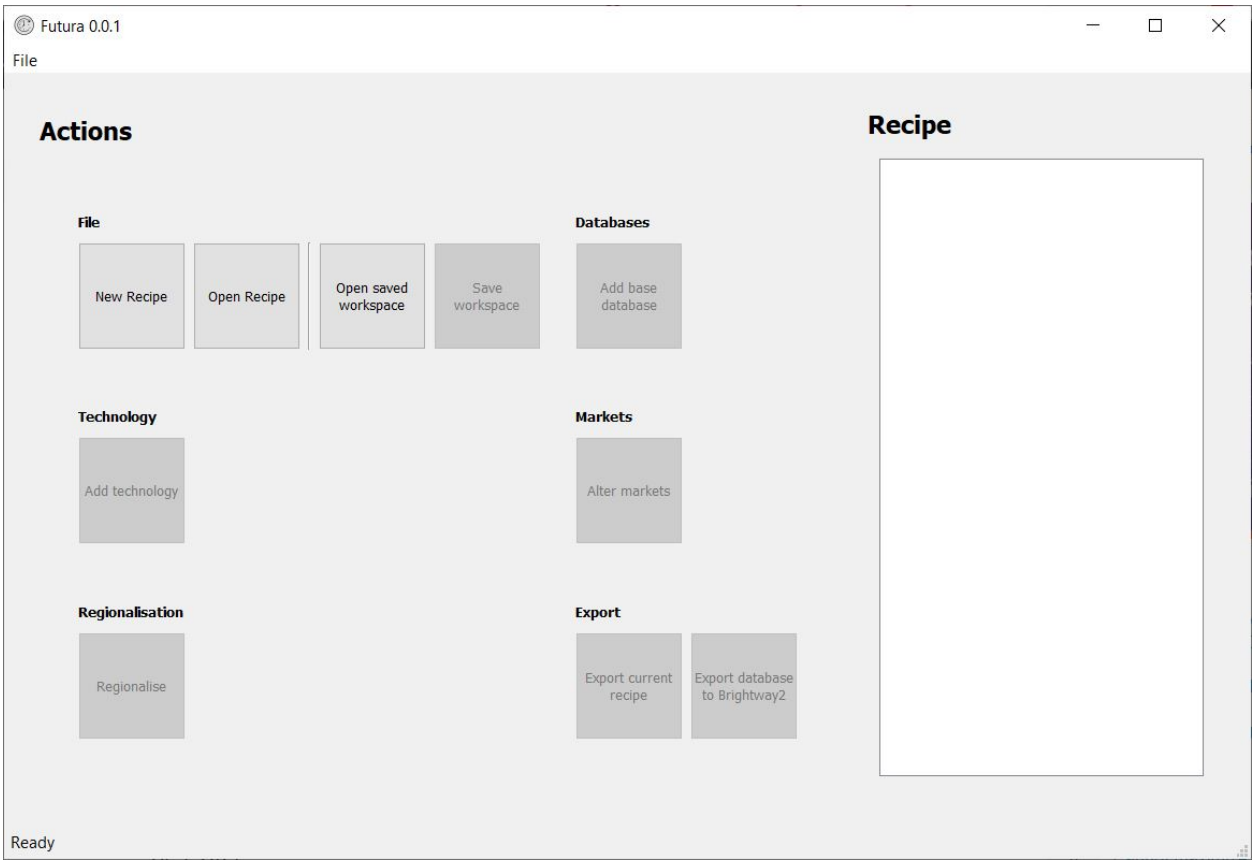
These actions allow the creation of scenarios ranging from the simple, such as altered electricity grid mixes, to the complex, where technologies such as carbon capture and storage or electrified transportation have become commonplace.

All changes made are automatically added to a *recipe* which can be exported as a human readable file (written in yaml). This recipe file contains no proprietary or licensed data and can be shared publicly, for example as supplementary material to a publication. This recipe can be imported by other users, and combined with their own version of the base database to exactly recreate the modified database that was originally created.



The additive and transparent nature of this system means that initially simple scenarios can be built upon by other practitioners to progress towards more comprehensive scenarios in a stepwise manner.

Futura includes a scripting interface, in order to integrate with existing scripting based softwares, but also includes a graphical user interface, allowing users to carry out all of the tasks required to create and share modified background databases without writing a single line of code.


Futura 0.0.1
File
Actions
Recipe
File
New Recipe
Open Recipe
Open saved workspace
Save workspace
Databases
Add base database
Technology
Add technology
Markets
Alter markets
Regionalisation
Regionalise
Export
Export current recipe
Export database to Brightway2
Ready

CHAPTER 2

Installation

2.1 Basic Installation

The easiest way to install Futura is via conda. This ensures the version of Python (3.6+) is correct, all of the dependencies are correct and there are no conflicts.

You can also do it in only 3 commands.

If you don't have conda installed yet, first install Miniconda (available here)

1. On the command line/console, create a new environment called futura:

```
conda create -n futura python=3
```

2. Activate the futura environment using one of these:

```
# Mac/Linux
source activate futura
# Windows
activate futura
```

3. Install futura:

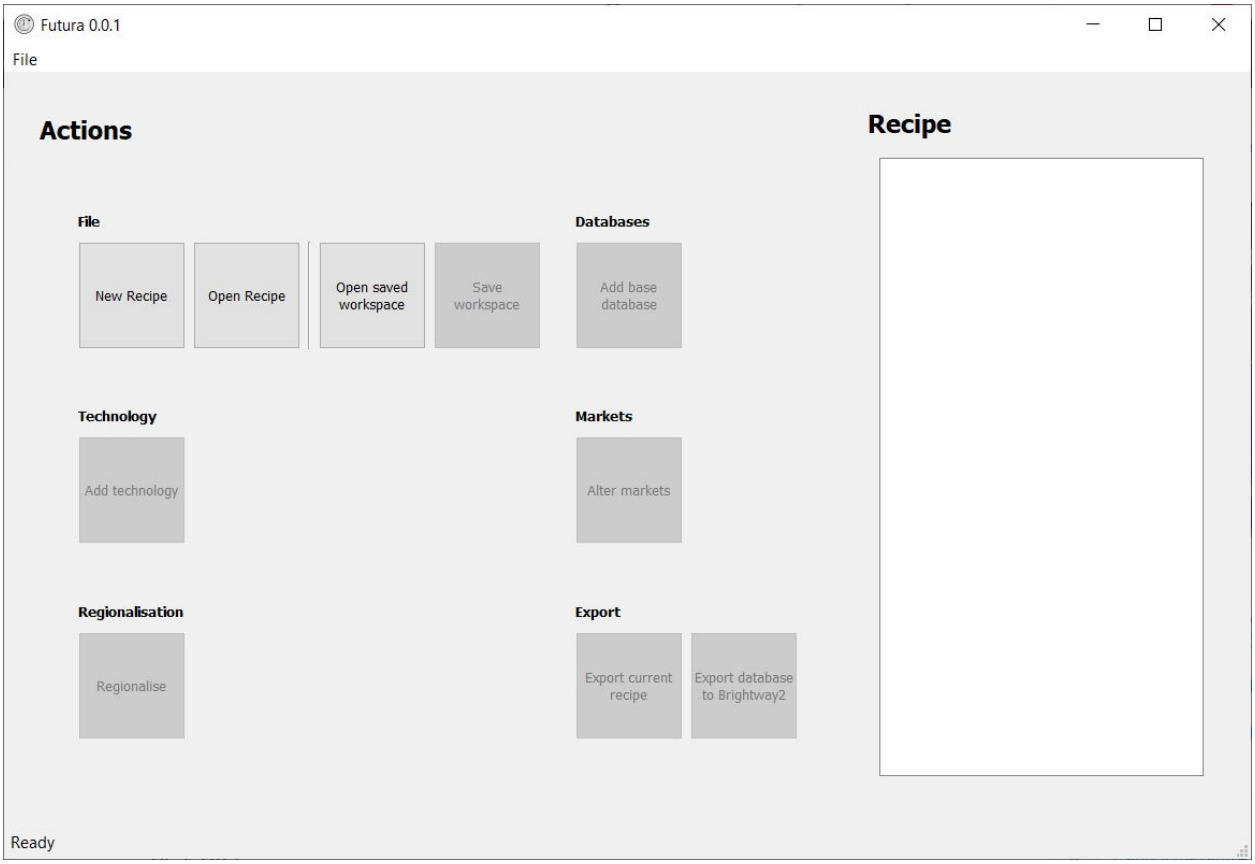
```
conda install -y -q -c conda-forge -c cmutel -c haasad -c konstantinstadler -c␣
↪pjamesjoyce futura
```

2.2 Using Futura

The easiest way to use Futura is via its user interface. Once you have installed futura you can launch the user interface by activating your futura environment and typing `futura_ui` like this:

```
$ activate futura
(futura) $ futura_ui
```

This will launch the user interface (written with Pyside2) which looks a bit like this:



CHAPTER 3

Technical Reference

3.1 futura_ui

3.1.1 futura_ui package

Subpackages

futura_ui.app package

Subpackages

futura_ui.app.models package

Submodules

futura_ui.app.models.geo module

futura_ui.app.models.pandasmodel module

futura_ui.app.models.recipemodel module

futura_ui.app.models.treemodel module

Module contents

futura_ui.app.ui package

Subpackages

futura_ui.app.ui.dialogs package

Submodules

futura_ui.app.ui.dialogs.add_base_database module

futura_ui.app.ui.dialogs.brightway_dialog module

futura_ui.app.ui.dialogs.ecoinvent_dialog module

futura_ui.app.ui.dialogs.edit_production module

futura_ui.app.ui.dialogs.new_recipe module

futura_ui.app.ui.dialogs.progress module

futura_ui.app.ui.dialogs.run_dialog module

futura_ui.app.ui.dialogs.transfer_production module

Module contents

futura_ui.app.ui.main package

Submodules

futura_ui.app.ui.main.main module

Module contents

futura_ui.app.ui.panels package

Submodules

futura_ui.app.ui.panels.frame module

futura_ui.app.ui.panels.left module

futura_ui.app.ui.panels.right module

Module contents

futura_ui.app.ui.recipes package

futura_ui.app.application module

futura_ui.app.controller module

futura_ui.app.signals module

futura_ui.app.threads module

futura_ui.app.utils module

futura_ui.app.wrappers module

Module contents

futura_ui.bin package

Submodules

futura_ui.bin.run_futura module

futura_ui.bin.run_futura_ui module

Module contents

Module contents

3.2 futura

3.2.1 futura package

Submodules

futura.constants module

futura.default_filters module

futura.ecoinvent module

futura.ecoinvent.**check_database**(*project_name*, *database*)

futura.loader module

class futura.loader.**FuturaLoader**(*recipe_filepath=None*, *autocreate=True*)

Bases: object

The FuturaLoader class sits at the centre of Futura and (via the FuturaDatabase, FuturaSaver and FuturaExecutor classes) allows you to load, run and save recipes for creating new databases. It also stores the current database and can be saved to disk itself and reloaded for quickly picking up where you left off.

Parameters

- **recipe_filepath** (*str, optional*) – Optionally pass a filepath to open an existing recipe. If left blank a blank recipe is created
- **autocreate** (*bool, optional*) – Automatically run the recipe when it is loaded. Default is *True*

Variables

- ***recipe*** (*dict*) – Dictionary representation of the current recipe. Can be set by *load_recipe()*
- ***database*** (*FuturaDatabase*) – A *FuturaDatabase* representing the current working database
- **executor** (*FuturaRecipeExecutor*) – A FuturaRecipeExecutor object which is invoked to run the current recipe
- **recipe_filepath** (*str*) – Path of the loaded recipe. Set if/when a recipe is loaded
- **load_path** (*str*) – Path of the loaded .fl file. Set if/when an .fl file is loaded

load(*load_path*)

load_recipe(*filename*)

Docstring: This is load_recipe - I need to write a docstring

Returns Parsed recipe as a dict

Return type dict

run()

save(*save_path=None*)

write_database(*project=None*, *database=None*, *overwrite=True*)

class futura.loader.**FuturaSaver**(*loader*)

Bases: object

TODO: Write doctring

futura.markets module

class futura.markets.**FuturaMarket**(*market*, *database*)

Bases: object

add_alternative_exchanges(*include_transport=False*)

add_pv(*process_name*, *new_pv*)

get_pv(*process_name*)

percentages

plot

production_volumes

relink()

rewrite_pvs()

set_pv(*process_name*, *new_pv*)

sorted_percentages

subtract_pv(*process_name*, *new_pv*)

total_production

transfer_pv(*from_name*, *to_name*, *factor=None*, *amount=None*)

futura.markets.**add_exchange_to_activity**(*base_activity*, *activity_to_link_to*)

futura.markets.**alter_production_volumes**(*processes*, *production_volumes*, *mismatch_ok=False*)

futura.markets.**find_possible_additional_market_exchanges**(*process*, *database*, *include_transport=False*)

futura.markets.**fix_exchange_production_volumes**(*process*, *database*)

futura.markets.**get_input_processes_to_market**(*process*, *database*)

futura.markets.**get_processes_from_exchanges**(*process*, *database*, *reference_product*)

futura.markets.**update_technosphere_exchanges_from_pvs**(*process*, *database*)

futura.proxy module

class futura.proxy.**WurstDatabase**(**args*, ***kwargs*)
Bases: list

class futura.proxy.**WurstFilter**(*func*, *x=None*, *signature=None*)
Bases: object

class futura.proxy.**WurstFilterSet**(**args*, ***kwargs*)
Bases: list

class futura.proxy.**WurstProcess**(**args*, ***kwargs*)
Bases: dict

futura.recipe module

class futura.recipe.**FuturaRecipeExecutor**(*loader*)
Bases: object

TODO: Write doctring

actions

database

db

ecoinvent_fallback()

execute_recipe()

execute_recipe_action(*recipe_action*, ***kwargs*)

recipe

recipe_generator()

set_market(*market_filter*)

set_process(*process_filter*)

futura.recipe.**add_default_CCS_processes**(*database, *, technology_file='/home/docs/checkouts/readthedocs.org/user_packages/futura-0.0.4-py3.7.egg/futura/assets/lci-Carma-CCS-base-GLO2.xlsx', funcs=[<function fix_ch_only_processes>, functools.partial(<function regionalise_based_on_filters>, location_filter=WurstFilterSet: [WurstFilter: exclude(WurstFilter: equals('database', 'Carma CCS')), WurstFilter: equals('unit', 'kilowatt hour'), WurstFilter: contains('name', 'hard coal'), WurstFilter: doesnt_contain_any('name', '['coal mine', 'co-generation']')], base_activity_filter=WurstFilterSet: [WurstFilter: equals('database', 'Carma CCS'), WurstFilter: either(WurstFilter: contains('name', 'Hard coal'), WurstFilter: contains('name', 'hard coal')), WurstFilter: equals('location', 'GLO')], progress_message='hard coal CCS activities'), functools.partial(<function regionalise_based_on_filters>, location_filter=WurstFilterSet: [WurstFilter: exclude(WurstFilter: equals('database', 'Carma CCS')), WurstFilter: equals('unit', 'kilowatt hour'), WurstFilter: contains('name', 'lignite'), WurstFilter: doesnt_contain_any('name', '['co-generation']')], base_activity_filter=WurstFilterSet: [WurstFilter: equals('database', 'Carma CCS'), WurstFilter: either(WurstFilter: contains('name', 'Lignite'), WurstFilter: contains('name', 'lignite')), WurstFilter: equals('location', 'GLO')], progress_message='lignite CCS activities'), functools.partial(<function regionalise_based_on_filters>, location_filter=WurstFilterSet: [WurstFilter: exclude(WurstFilter: equals('database', 'Carma CCS')), WurstFilter: equals('unit', 'kilowatt hour'), WurstFilter: contains('name', 'natural gas'), WurstFilter: doesnt_contain_any('name', '['co-generation', 'import', 'aluminium industry', 'burned', '10MW']')], base_activity_filter=WurstFilterSet: [WurstFilter: equals('database', 'Carma CCS'), WurstFilter: either(WurstFilter: contains('name', 'natural gas'), WurstFilter: contains('name', 'Natural gas'), WurstFilter: contains('name', 'ATR-H2')), WurstFilter: equals('location', 'GLO')], progress_message='natural gas CCS activities'), functools.partial(<function regionalise_based_on_filters>, location_filter=WurstFilterSet: [WurstFilter: exclude(WurstFilter: equals('database', 'Carma CCS')), WurstFilter: equals('unit', 'kilowatt hour'), WurstFilter: contains('name', 'wood'), WurstFilter: doesnt_contain_any('name', '['treatment', 'ethanol', 'pellets', 'label-certified', '2000 kW']'), WurstFilter: contains('name', 'state-of-the-art')], base_activity_filter=WurstFilterSet: [WurstFilter: equals('database', 'Carma CCS'), WurstFilter: either(WurstFilter: contains('name',*

futura.recipe.**add_hard_coal_ccs**(*database, *, location_filter=WurstFilterSet: [WurstFilter: exclude(WurstFilter: equals('database', 'Carma CCS')), WurstFilter: equals('unit', 'kilowatt hour'), WurstFilter: contains('name', 'hard coal'), WurstFilter: doesnt_contain_any('name', '['coal mine', 'co-generation']')], base_activity_filter=WurstFilterSet: [WurstFilter: equals('database', 'Carma CCS'), WurstFilter: either(WurstFilter: contains('name', 'Hard coal'), WurstFilter: contains('name', 'hard coal')), WurstFilter: equals('location', 'GLO')], progress_message='hard coal CCS activities'*)

futura.recipe.**add_lignite_ccs**(*database, *, location_filter=WurstFilterSet: [WurstFilter: exclude(WurstFilter: equals('database', 'Carma CCS')), WurstFilter: equals('unit', 'kilowatt hour'), WurstFilter: contains('name', 'lignite'), WurstFilter: doesnt_contain_any('name', '['co-generation']')], base_activity_filter=WurstFilterSet: [WurstFilter: equals('database', 'Carma CCS'), WurstFilter: either(WurstFilter: contains('name', 'Lignite'), WurstFilter: contains('name', 'lignite')), WurstFilter: equals('location', 'GLO')], progress_message='lignite CCS activities'*)

futura.recipe.**add_natural_gas_ccs**(*database, *, location_filter=WurstFilterSet: [WurstFilter: exclude(WurstFilter: equals('database', 'Carma CCS')), WurstFilter: equals('unit', 'kilowatt hour'), WurstFilter: contains('name', 'natural gas'), WurstFilter: doesnt_contain_any('name', '['co-generation', 'import', 'aluminium industry', 'burned', '10MW']')], base_activity_filter=WurstFilterSet: [WurstFilter: equals('database', 'Carma CCS'), WurstFilter: either(WurstFilter: contains('name', 'natural gas'), WurstFilter: contains('name', 'Natural gas'), WurstFilter: contains('name', 'ATR-H2')), WurstFilter: equals('location', 'GLO')], progress_message='natural gas CCS activities'*)

futura.recipe.**add_wood_ccs**(*database, *, location_filter=WurstFilterSet: [WurstFilter: exclude(WurstFilter: equals('database', 'Carma CCS')), WurstFilter: equals('unit', 'kilowatt hour'), WurstFilter: contains('name', 'wood'), WurstFilter: doesnt_contain_any('name', '['treatment', 'ethanol', 'pellets', 'label-certified', '2000 kW']'), WurstFilter: contains('name', 'state-of-the-art')], base_activity_filter=WurstFilterSet: [WurstFilter: equals('database', 'Carma CCS'), WurstFilter: either(WurstFilter: contains('name', 'wood'), WurstFilter: contains('name', 'Wood')), WurstFilter: equals('location', 'GLO')], progress_message='wood CCS activities'*)

futura.regionalisation module

futura.regionalisation.**create_regional_activities**(*base_activity*, *new_regions*, *db*, *production_volumes=None*, *remove_production_from_original=True*, *relink_now=True*, *keep_invalid=True*)

futura.regionalisation.**create_regional_activities_from_filter**(*base_activity_filter*, *new_regions*, *db*, *production_volumes=None*, *remove_production_from_original=True*, *relink_now=True*)

futura.storage module

class futura.storage.**FuturaStorage**

Bases: object

config

databases

ecoinvent_versions

write_config(*config*)

write_default_config()

futura.technology module

futura.technology.**add_default_CCS_processes**(*database*, **, technology_file='/home/docs/checkouts/readthedocs.org/user_builds/futura/envs/latest/lib/python3.7/site-packages/futura-0.0.4-py3.7.egg/futura/assets/lci-Carma-CCS-base-GLO2.xlsx', funcs=[<function fix_ch_only_processes>, functools.partial(<function regionalise_based_on_filters>, location_filter=WurstFilterSet: [WurstFilter: exclude(WurstFilter: equals('database', 'Carma CCS')), WurstFilter: equals('unit', 'kilowatt hour'), WurstFilter: contains('name', 'hard coal'), WurstFilter: doesnt_contain_any('name', ['coal mine', 'co-generation'])], base_activity_filter=WurstFilterSet: [WurstFilter: equals('database', 'Carma CCS'), WurstFilter: either(WurstFilter: contains('name', 'Hard coal'), WurstFilter: contains('name', 'hard coal')), WurstFilter: equals('location', 'GLO')], progress_message='hard coal CCS activities'), functools.partial(<function regionalise_based_on_filters>, location_filter=WurstFilterSet: [WurstFilter: exclude(WurstFilter: equals('database', 'Carma CCS')), WurstFilter: equals('unit', 'kilowatt hour'), WurstFilter: contains('name', 'lignite'), WurstFilter: doesnt_contain_any('name', ['co-generation'])], base_activity_filter=WurstFilterSet: [WurstFilter: equals('database', 'Carma CCS'), WurstFilter: either(WurstFilter: contains('name', 'Lignite'), WurstFilter: contains('name', 'lignite')), WurstFilter: equals('location', 'GLO')], progress_message='lignite CCS activities'), functools.partial(<function regionalise_based_on_filters>, location_filter=WurstFilterSet: [WurstFilter: exclude(WurstFilter: equals('database', 'Carma CCS')), WurstFilter: equals('unit', 'kilowatt hour'), WurstFilter: contains('name', 'natural gas'), WurstFilter: doesnt_contain_any('name', ['co-generation', 'import', 'aluminium industry', 'burned', '10MW'])], base_activity_filter=WurstFilterSet: [WurstFilter: equals('database', 'Carma CCS'), WurstFilter: either(WurstFilter: contains('name', 'natural gas'), WurstFilter: contains('name', 'Natural gas'), WurstFilter: contains('name', 'ATR-H2')), WurstFilter: equals('location', 'GLO')], progress_message='natural gas CCS activities'), functools.partial(<function regionalise_based_on_filters>, loca-*

futura.technology.**add_hard_coal_ccs**(*database, *, location_filter=WurstFilterSet: [WurstFilter: exclude(WurstFilter: equals('database', 'Carma CCS')), WurstFilter: equals('unit', 'kilowatt hour'), WurstFilter: contains('name', 'hard coal'), WurstFilter: doesnt_contain_any('name', '['coal mine', 'co-generation']')], base_activity_filter=WurstFilterSet: [WurstFilter: equals('database', 'Carma CCS'), WurstFilter: either(WurstFilter: contains('name', 'Hard coal'), WurstFilter: contains('name', 'hard coal')), WurstFilter: equals('location', 'GLO')], progress_message='hard coal CCS activities'*)

futura.technology.**add_lignite_ccs**(*database, *, location_filter=WurstFilterSet: [WurstFilter: exclude(WurstFilter: equals('database', 'Carma CCS')), WurstFilter: equals('unit', 'kilowatt hour'), WurstFilter: contains('name', 'lignite'), WurstFilter: doesnt_contain_any('name', '['co-generation']')], base_activity_filter=WurstFilterSet: [WurstFilter: equals('database', 'Carma CCS'), WurstFilter: either(WurstFilter: contains('name', 'Lignite'), WurstFilter: contains('name', 'lignite')), WurstFilter: equals('location', 'GLO')], progress_message='lignite CCS activities'*)

futura.technology.**add_natural_gas_ccs**(*database, *, location_filter=WurstFilterSet: [WurstFilter: exclude(WurstFilter: equals('database', 'Carma CCS')), WurstFilter: equals('unit', 'kilowatt hour'), WurstFilter: contains('name', 'natural gas'), WurstFilter: doesnt_contain_any('name', '['co-generation', 'import', 'aluminium industry', 'burned', '10MW']')], base_activity_filter=WurstFilterSet: [WurstFilter: equals('database', 'Carma CCS'), WurstFilter: either(WurstFilter: contains('name', 'natural gas'), WurstFilter: contains('name', 'Natural gas'), WurstFilter: contains('name', 'ATR-H2')), WurstFilter: equals('location', 'GLO')], progress_message='natural gas CCS activities'*)

futura.technology.**add_technology_to_database**(*database*, *technology_file*, *funcs=None*)

futura.technology.**add_wood_ccs**(*database, *, location_filter=WurstFilterSet: [WurstFilter: exclude(WurstFilter: equals('database', 'Carma CCS')), WurstFilter: equals('unit', 'kilowatt hour'), WurstFilter: contains('name', 'wood'), WurstFilter: doesnt_contain_any('name', '['treatment', 'ethanol', 'pellets', 'label-certified', '2000 kW']'), WurstFilter: contains('name', 'state-of-the-art')], base_activity_filter=WurstFilterSet: [WurstFilter: equals('database', 'Carma CCS'), WurstFilter: either(WurstFilter: contains('name', 'wood'), WurstFilter: contains('name', 'Wood')), WurstFilter: equals('location', 'GLO')], progress_message='wood CCS activities'*)

futura.technology.**fix_ch_only_processes**(*database*)

futura.technology.**regionalise_based_on_filters**(*database*, *location_filter*, *base_activity_filter*, *progress_message=None*)

futura.technology.**regionalise_multiple_processes**(*database*, *locations*, *base_activity_filter*, *progress_message=None*)

futura.utils module

futura.utils.**convert_parameters_to_wurst_style**(*parameter_list*)

futura.utils.**create_filter_from_description**(*description*, *database_filter=None*)

futura.utils.**find_location_given_lookup_dict**(*db*, *lookup_dict*)
Utility function for the utility function above :param db: database to fix :param lookup_dict: dictionary of locations :return: list of locations

futura.utils.**fix_products_and_locations_external**(*external_data*, *existing_data*)

futura.utils.**fix_unset_technosphere_and_production_exchange_locations**(*db*, *matching_fields=('name', 'unit')*)
Utility function from wurst publication supplementary materials to fix unset technosphere and production exchanges. Database is fixed in place, function returns nothing

Parameters

- **db** – database to fix
- **matching_fields** – fields on which to search for exchanges

Returns nothing

futura.utils.**remove_nones**(*db*)

futura.wrappers module

class futura.wrappers.**FuturaDatabase**
Bases: object

TODO: Write doctring

Variables ***db*** (*WurstDatabase*) – database

extract_BW2Package(*packagefilepath*)

extract_bw2_database(*project_name*, *database_name*)

extract_excel_data(*excelfilepath*)

get_ecoinvent(*db_name=None*, *download_path=None*, *store_download=True*, ***kwargs*)
Download and import ecoinvent to FuturaDatabase. Sets db directly

Optional kwargs:

Parameters

- **db_name** (*str, optional*) – name to give imported database. Default is downloaded filename
- **download_path** (*str, optional*) – path to download .7z file to default is download to temporary directory (.7z file is deleted after import)
- **store_download** (*bool, optional*) – store the .7z file for later reuse, default is True, only takes effect if no download_path is provided

- **username** (*str, optional*) – ecoinvent username
- **password** (*str, optional*) – ecoinvent password
- **version** (*str, optional*) – ecoinvent version, eg '3.5'
- **system_model** (*str, optional*) – ecoinvent system model, one of {'cutoff', 'apos', 'consequential'}

Returns None

Return type None

load(*load_path*)

save(*save_directory=None*, *save_filename=None*)

write_database(*project*, *name*, *overwrite=False*)

futura.wurst_monkeypatch module

futura.wurst_monkeypatch.**allocate_inputs**(*exc*, *lst*)
Allocate the input exchanges in `lst` to `exc`, using production volumes where possible, and equal splitting otherwise.

Always uses equal splitting if `RoW` is present.

futura.wurst_monkeypatch.**copy_to_new_location**(*ds*, *location*)
Copy dataset and substitute new `location`.

Doesn't change exchange locations, except for production exchanges.

Returns the new dataset.

futura.wurst_monkeypatch.**default_global_location**(*database*)
Set missing locations to `` `GLO` `` for datasets in `database`.

Changes location if `location` is missing or `None`. Will add key `location` if missing.

futura.wurst_monkeypatch.**get_possibles**(*exchange*, *data*)
Filter a list of datasets `data`, returning those with the same name, reference product, and unit as in `exchange`.

Returns a generator.

futura.wurst_monkeypatch.**relink_technosphere_exchanges**(*ds*, *data*, *exclusive=True*, *drop_invalid=False*, *keep_invalid=False*, *biggest_first=False*, *contained=True*, *exclude=None*)
Find new technosphere providers based on the location of the dataset.

Designed to be used when the dataset's location changes, or when new datasets are added.

Uses the name, reference product, and unit of the exchange to filter possible inputs. These must match exactly. Searches in the list of datasets `data`.

Will only search for providers contained within the location of `ds`, unless `contained` is set to `False`, all providers whose location intersects the location of `ds` will be used.

A `RoW` provider will be added if there is a single topological face in the location of `ds` which isn't covered by the location of any providing activity.

If no providers can be found, *relink_technosphere_exchanes* will try to add a *RoW* or *GLO* providers, in that order, if available. If there are still no valid providers, a `InvalidLink` exception is raised, unless `drop_invalid` is `True`, in which case the exchange will be deleted.

Allocation between providers is done using `allocate_inputs`; results seem strange if `contained=False`, as production volumes for large regions would be used as allocation factors.

Input arguments:

- `ds`: The dataset whose technosphere exchanges will be modified.
- `data`: The list of datasets to search for technosphere product providers.
- `exclusive`: Bool, default is `True`. Don't allow overlapping locations in input providers.
- `drop_invalid`: Bool, default is `False`. Delete exchanges for which no valid provider is available.
- `keep_invalid`: Bool, default is `False`. Keep potentially invalid exchanges from original datasets where not valid alternative provider available.
- `biggest_first`: Bool, default is `False`. Determines search order when selecting provider locations. Only relevant is `exclusive` is `True`.
- `contained`: Bool, default is `True`. If ture, only use providers whose location is completely within the `ds` location; otherwise use all intersecting locations.
- `exclude`: List, optional list of locations to exclude possible exchanges from.

Modifies the dataset in place; returns the modified dataset.

Module contents

`futura.`**`log`**(*MESSAGE*, **args*, ***kwargs*)

`futura.`**`return_WurstProcess`**(*wrapper=None*, *enabled=None*, *adapter=None*, *proxy=<class 'FunctionWrapper'>*)

`futura.`**`warn`**(*MESSAGE*, **args*, ***kwargs*)

CHAPTER 4

Indices and tables

- genindex
- modindex
- search

Python Module Index

f

Index

G

L

P

R

S

T

U

W