WasteAndMaterialFootprint Documentation

Release 0.1.2

Stewart Charles McDowall

Dec 28, 2023

CONTENTS:

CHAPTER

ONE

INTRODUCTION

The WasteAndMaterialFootprint tool is a python package that allows one to calculate the waste and material footprint of any product or service inside of life cycle assessment (LCA) databases. Currently is has been tested with ecoinvent 3.8, 3.9, 3.9.1 and 3.10.

CHAPTER

TWO

INSTALLATION

2.1 Dependencies

The program is written in Python and the required packages are listed in the *requirements.txt* file. These should be installed automatically when installing the program.

The main dependencies are:

- brightway2 components: bw2io, bw2data, bw2calc
- premise
- wurst

2.2 Installation Instructions

It is recommended to use a fresh virtual environment to install the program.

You can simply clone or download the repo and run it without installation:

```
python venv -m <name>
source <name>/bin/activate
```

This will not install any of the dependencies, so you will need to install them manually if you don't already have them.

```
pip install -r requirements.txt
```

Alternatively: the program can be installed using pip:

Either from PyPI:

```
pip install WasteAndMaterialFootprint
```

Or, probably best to install the latest version from GitHub:

```
pip install git+https://github.com/Stew-McD/WasteAndMaterialFootprint.git
```

Or for an editable install (good for development and testing):

```
git clone https://github.com/Stew-McD/WasteAndMaterialFootprint.git
cd WasteAndMaterialFootprint
pip install -e .
```

CHAPTER

THREE

USAGE

The program can be used directly from the command line, or imported as a Python module. This will run the program using the default settings. See the configuration section for more information on how to change the settings.

3.1 Command Line

You should clone the repo, navigate to the `WasteAndMaterialFootprint` folder, and then run the program using:

```
python src/WasteAndMaterialFootprint/main.py
```

Configuration files can be found in *src/WasteAndMaterialFootprint/config/*. These can be edited before running the main script.

3.2 Python Module

The program can be imported as a Python module if it has been installed with pip. The main function can then be run using:

```
import WasteAndMaterialFootprint as wmf
wmf.run()
```

As with the command line, the configuration files can be found in *src/WasteAndMaterialFootprint/config/*. These can be edited before running the main script. It is also possible to edit the configuration settings directly in the Python script, and accessed in interactive terminal sessions like iPython and Jupyter. For example:

```
>>> import WasteAndMaterialFootprint as wmf

>>> wmf.user_settings.use_premise
True
>>> wmf.user_settings.use_wmf
False
>>> wmf.user_settings.use_wmf = True
>>> wmf.user_settings.use_wmf
True
>>> wmf.user_settings.project_base
'SSP-cutoff'
>>> wmf.user_settings.project_base = "other project"
>>> wmf.user_settings.project_base
'other project'
```

The individual modules can also be imported and used separately. For example:

```
import WasteAndMaterialFootprint as wmf

# only use premise
wmf.FutureScenarios.MakeFutureScenarios()

# only do waste or material searches
database = 'my database'
project = 'my project'
wmf.ExplodeDatabase(project, database)
wmf.SearchWaste(project, database)
wmf.SearchMaterial(project, database)

# check databases

wmf.VerifyDatabase(project, database)

# or with the internal settings:

database = wmf.user_settings.database_name

# check original database
project_base = wmf.user_settings.project_base
wmf.VerifyDatabase(project_base, database)
# (this will return '0', because it was not edited)

# check final database
project = wmf.user_settings.project_wmf
wmf.VerifyDatabase(project, database)
# (this will return '1', if it was edited correctly)
```

CHAPTER

FOUR

CONFIGURATION MODULES

By default, the program will create a folder `config` in the current working directory containing the default configuration files:

4.1 General Settings: `user_settings.py`

This is the main configuration file, the one that you might want to edit to match your project structure and your needs. By default, the program will take a brightway2 project named `default` and copy that to a new project named `SSP-cutoff`, which is then copied to a new project named `WMFootprint-SSP-cutoff`.

Doing it this way isolates the components and allows you to keep your original brightway2 project as it was. If space is an issue, you can set all of the project names to be the same.

If you are happy with the default settings, you can just run the program and it will create the databases for you. If you want to change the settings, you can edit the `user_settings.py` file that you can find in the `config` directory of your working directory.

These are some extracts from `user_settings.py` with the most important settings (the ones you might want to change) and their default values:

```
# Choose whether to use premise to create future scenario databases
use_premise = True
# Choose whether to use WasteAndMaterialFootprint to edit the databases (you could also␣
↪turn this off and just use the package as an easy way to make a set of future scenario␣
↪databases)
use_wmf = True

# Choose the names of the projects to use
project_premise_base = "default"
project_premise = "SSP-cutoff"
project_base = project_premise
project_wmf = f"WMFootprint-{project_base}"

# Choose the name of the database to use (needed for premise only, the WMF tool will run␣
↪all databases except the biospheres)
database_name = "ecoinvent-3.9.1-cutoff"

# if you want to use a fresh project
delete_existing_premise_project = False
delete_existing_wmf_project = False
```

(continues on next page)

(continued from previous page)

```
# Choose the premise scenarios to generate (see FutureScenarios.py for more details)
# Not all combinations are available, the code in FutureScenarios.py will filter out the␣
↪scenarios that are not possible
# the default is to have an optimistic and a pessimistic scenario with SSP2 for 2030,␣
↪2065 and 2100

models = ["remind"]
ssps = ["SSP2"]
rcps = ["Base","PkBudg500"]
years = [2030,2065,2100,]
```

4.2 Waste Search Settings: `queries_waste.py`

This file sets up search parameters for different waste and material flow categories, crucial for the `SearchWaste.py` script. It leverages a `.pickle` file created by `ExplodeDatabase.py`.

4.2.1 Categories

Handles various categories like digestion, composting, incineration, recycling, landfill, etc.

4.2.2 Query Types

Two sets of queries are created:

1. `queries_kg` for waste flows in kilograms.
2. `queries_m3` for waste flows in cubic meters.

4.2.3 Adjusting Search Terms

- **Search Keywords**: Tweak the `AND`, `OR`, `NOT` lists to refine your search.

4.2.4 Category-Specific Changes

- **Adding Categories**: You can add new categories to the `names` list.
- **Modifying Queries**: Update the query parameters for each category based on your requirements.

4.2.5 Optimising Search Efficiency

You can choose to include or exclude whatever you want. For instance, "non-hazardous" is not included as it's derivable from other categories and slows down the process.

4.2.6 Validating Search Terms

Isolate the function of `SearchWaste.py` to validate your search terms. That means, turning off the other functions in `user_settings.py`, or running the module directly. You can achieve this by setting the following in `user_settings.py`:

```
use_premise = False
do_search = True
do_methods = False
do_edit = False
```

4.3 Material Search Settings: `queries_materials.py`

The `queries_materials` module creates demand methods in the WasteAndMaterialFootprint tool. It aligns with the EU CRM list 2023 and the ecoinvent database, incorporating additional strategic materials for comprehensive analysis. More can be easily added, as wished by the user.

This function uses the string tests `startswith` in `SearchMaterial.py` to identify activities beginning with the specified material name. This allows one to be more specific with the search terms (the `,` can be critical sometimes).

4.3.1 Structure and Customisation

Tuple Structure

- **First Part (Activity Name)**: Specifies the exact activity in the database (e.g., `market for chromium`).
- **Second Part (Material Category)**: Aggregates related activities under a common category (e.g., `chromium`), enhancing data processing efficiency.

Customisation Options

- **Add or Remove Materials**: Adapt the tuple list by including new materials or removing irrelevant ones.
- **Refine Search Terms**: Update material categories for a better fit with your database, ensuring precision in naming, especially with the use of commas.

4.3.2 Usage Considerations

- **Material Quantity**: The current list comprises over 40 materials. Modify this count to suit your project's scope.
- **Database Alignment**: Check that the material names correspond with your specific database version, like ecoinvent v3.9.1.

Example Tuples

- ("market for chromium", "chromium")
- ("market for coal", "coal")
- ("market for cobalt", "cobalt")
- ("market for coke", "coke")
- ("market for copper", "copper")
- ("market for tap water", "water")
- ("market for water,", "water")

CHAPTER

FIVE

WASTEANDMATERIALFOOTPRINT MODULES

This section provides an overview of the various modules within the WasteAndMaterialFootprint (WMF) program, each with a brief description and a link to its detailed documentation.

5.1 FutureScenarios

The *FutureScenarios* module is an integral part of the WasteAndMaterialFootprint project, enabling the creation of future scenario databases. It leverages the [premise](https://github.com/polca-project/premise) Python package to generate future scenario databases based on integrated assessment models (IAMs) for brightway2 projects.

5.1.1 Configuration

Before using the *FutureScenarios* module, it's crucial to set up your desired scenarios in the *user_settings.py* file. This configuration determines the specific scenarios the module will process.

Configuring Projects and Premise Settings

- **Project Settings**: Define the base and target projects for the scenarios. For instance, *project_premise_base* as the source, and *project_premise* as the destination project.
- **Database Selection**: Specify the database to use, like *ecoinvent-3.9.1-cutoff*.
- **Premise Key**: A key is required for accessing premise functionality. It can be hardcoded or read from a file.
- **Multiprocessing and Batch Size**: Decide on using multiprocessing and the size of batches to process scenarios.
- **Deletion Settings**: Choose whether to delete existing projects before creating new ones.

Selecting Scenarios

Define the models, SSPs (Shared Socioeconomic Pathways), RCPs (Representative Concentration Pathways), and years you want to explore. The script will then attempt to create databases for each combination of these parameters, provided they are available.

5.1.2 Implementation

Module Overview

The *FutureScenarios* module consists of several key functions and a main script that orchestrates the entire process of creating future scenario databases.

Key Functions

- **make_possible_scenario_list**: Generates a list of feasible scenarios based on user preferences and available data.
- **check_existing**: Checks if a scenario already exists in the project to avoid redundancy.
- **FutureScenarios**: The core function that invokes the *premise* module to create new databases for each specified scenario.
- **MakeFutureScenarios**: The main function that calls *FutureScenarios* if the *use_premise* flag is set to True.

Process Flow

1. **Initialisation**: The script starts by importing necessary libraries and user settings. It sets up logging and changes the working directory if necessary.
2. **Scenario Filtering**: It filters out unavailable or existing scenarios.
3. **Project Preparation**: Depending on user settings, it either uses an existing project or creates a new one.
4. **Scenario Processing**: For each scenario, it calls *premise* to update or create a database reflecting that scenario.
5. **Database Writing**: After processing, the script writes the new databases to Brightway2.
6. **Cleanup and Conclusion**: The script concludes by adding GWP factors and returning to the original directory.

5.2 ExplodeDatabase

The ExplodeDatabase module is the first fundamental component of the WasteAndMaterialFootprint (WMF) tool, designed for processing Brightway2 life cycle assessment (LCA) databases. It provides a mechanism to expand a given LCA database into a detailed list of all exchanges, facilitating subsequent analysis with the search modules.

5.2.1 Primary Function

The module's core functionality is to 'explode' a Brightway2 database. This term refers to the process of decomposing the database into a single-level, comprehensive list of all exchanges. Each exchange in the database is extracted and documented, including detailed attributes such as name, amount, unit, product, production volume, type, and location.

5.2.2 Process Overview

1. Utilising the wurst library, the module opens the specified Brightway2 database.
2. It extracts information from each process in the database.
3. The extracted data is converted into a pandas DataFrame for ease of analysis and manipulation.
4. The module then expands ('explodes') this DataFrame to detail each exchange individually.
5. The resulting data, now a comprehensive list of exchanges, is saved in a .pickle binary file for efficient storage and retrieval.

5.2.3 Usage

The ExplodeDatabase function is invoked with a single argument, the name of the Brightway2 database to be processed. It performs the exploding process, logs the operation, and saves the resulting DataFrame as a pickle file. The function is designed for internal use within the WMF tool and does not return a value but rather saves the output for subsequent use.

5.3 SearchWaste

The SearchWaste module is a part of the WasteAndMaterialFootprint tool, dedicated to processing waste-related data. It loads data from a specified '< db name >_exploded.pickle' file, executes predefined search queries on this data, and generates CSV files containing the results along with corresponding log entries. The search queries are structured as dictionaries, specified in the *config/queries_waste.py* file, and include fields such as NAME, CODE, and search terms like keywords_AND, keywords_OR, and keywords_NOT.

5.3.1 Functionality

The module provides the `SearchWaste()` function, which is responsible for three main actions:

1. Loading data from the '< db name >_exploded.pickle' file.

```
pickle_path = os.path.join(dir_tmp, db_name + "_exploded.pickle")
df = pd.read_pickle(pickle_path)
```

2. Running the specified search queries on this data. These queries are designed to filter and identify relevant waste exchanges based on specific criteria.

   This functionality is implemented in the `search()` function, which is a subfunction of the `SearchWaste()` function. The `search()` function takes one argument, a search query from the list that is produced by the configuration module *queries_waste.py*.

   The search function is applied as follows, where *df* is the dataframe of the exploded database and *query* is a dictionary:

```
NAME_BASE = query["name"]
UNIT = query["unit"]
NAME = NAME_BASE + "-" + UNIT
CODE = NAME.replace(" ", "")
query.update({"code": NAME, "db_name": db_name})
AND = query["AND"]
OR = query["OR"]
```

(continues on next page)

(continued from previous page)

```
NOT = query["NOT"]
DBNAME = query["db_name"]

# Apply the search terms to the dataframe
df_results = df[
    (df["ex_name"].apply(lambda x: all(i in x for i in AND)))
    & (df["ex_unit"] == UNIT)
    & (df["ex_amount"] != 0)
    # & (df["ex_amount"] != -1)
].copy()

# Apply OR and NOT search filters
if OR:
    df_results = df_results[
        df_results["ex_name"].apply(lambda x: any(i in x for i in OR))
    ]
if NOT:
    df_results = df_results[
        df_results["ex_name"].apply(lambda x: not any(i in x for i in NOT))
    ]
```

3. Producing CSV files to store the results of these queries and creating log entries for each search operation. When customising the search configuration, it is important to check these files to see that the correct exchanges are being captured. The files are used by the subsequent modules to edit the exchanges and to produce LCIA methods.

5.3.2 Usage

```
wmf.SearchWaste(db_name, output_dir)
```

The `SearchWaste()` function is invoked with two arguments: the name of the Brightway2 database to be processed and the name of the directory to store the results. The search queries are specified in the *config/queries_waste.py* file. The function is designed for internal use within the WMF tool and does not return a value but rather saves the output for subsequent use. It could be used separately, if you would have a .pickle file with exploded database as well as the config files in the right locations.

5.4 SearchMaterial

The SearchMaterial module's primary function is to load data from '< db name >_exploded.pickle', execute search queries based on the list of materials, and store the results in a CSV file along with a corresponding log entry. The search queries are formatted as tuples, where the first entry is the name of the activity and the second is the material grouping, e.g. *("market for sodium borates", "borates")*. These queries are defined by the list in *queries_materials.py*, which can be easily modified by the user to change the scope or the groupings as desired.

5.4.1 Functionality

The module's core function, `SearchMaterial()`, takes a database name and a Brightway2 project as inputs. It searches for materials within the specified database, extracts relevant information such as ISIC and CPC classifications, and saves this information in CSV format. It also extracts and saves related material exchanges and saves them as separate files based on the material grouping. The files are used by the subsequent modules to edit the exchanges and to produce LCIA methods.

5.4.2 Search queries

The module *queries_material.py* contains the list of materials used to create demand methods in the WasteAndMaterialFootprint tool. The materials are chosen to match activities in the ecoinvent v3.9.1 database and future databases made with premise, and to align with the EU CRM list 2023. Additional strategic materials are included, along with some other materials of interest.

NOTE: The search function in *SearchMaterial.py* uses *str.startswith*, finding all activities that start with the material name.

```
# Filter activities based on the materials list
acts_all = pd.DataFrame([x.as_dict() for x in db])
materials_df = pd.DataFrame(queries_materials, columns=["name", "group"])

acts = acts_all[
    acts_all["name"].apply(
        lambda x: any(x.startswith(material) for material in materials_df.name)
    )
].reset_index(drop=True)
```

The comma in material names is crucial for filtering specific activities in the database. For example, with the comma, the query *market for gold,* will catch the activity *market for nickel* and *market for nickel, 90%* but not *market for nickel plating*.

```
queries_materials = [
    ("market for aluminium", "aluminium"),
    ("market for antimony", "antimony"),
    # ... [rest of the list] ...
    ("market for zirconium", "zirconium"),
]
```

5.5 MakeCustomDatabase

The MakeCustomDatabase module's function is to facilitate the creation and integration of custom databases into the Brightway2 project. This module is responsible for generating an xlsx representation of a Brightway2 database based on the output of the modules *SearchWaste* and *SearchMaterial* and importing it into as a database with *bwio* from the Brightway2 framework.

5.5.1 Main Functions

- **dbWriteExcel**: This function creates an xlsx file that represents a custom Brightway2 database, using data from predefined directories and database settings.
- **dbExcel2BW**: This function imports the custom database, created by dbWriteExcel, into Brightway2, making it available for the *ExchangeEditor* and *MethodEditor* modules.

5.5.2 Important Code Snippets

- **Function to Collect Filenames from Directories**:

```
def get_files_from_tree(dir_searchmaterial_results, dir_searchwaste_results):
    # Implementation of the function
```

- **Function to Create Excel File for Custom Database**:

```
def dbWriteExcel():
    #.... snippet to show structure of the activities in the database ...
    for NAME in names:
    count += 1
    CODE = NAME
    UNIT = determine_unit_from_name(NAME)

    if "Waste" in NAME:
        TYPE = "waste"
        CODE = (
            NAME.replace("WasteFootprint_", "")
            .capitalize()
            .replace("kilogram", "(kg)")
            .replace("cubicmeter", "(m3)")
            .replace("-", " ")
        )
    elif "Material" in NAME:
        TYPE = "natural resource"
        CODE = NAME.replace("MaterialFootprint_", "").capitalize()
    else:
        TYPE = "?"

    db_entry = {
        "Activity": NAME,
        "categories": "water, air, land",
        "code": CODE,
        "unit": UNIT,
        "type": TYPE,
```

(continues on next page)

(continued from previous page)

```
    }

# ... .....
```

- **Function to Determine Unit from Name**:

```
def determine_unit_from_name(name):
    # Implementation of the Function
```

- **Function to Import Database into Brightway2**:

```
def dbExcel2BW():
    # Implementation of the Function
```

5.6 MethodEditor

The MethodEditor module is a key component of the WasteAndMaterialFootprint tool, offering functionalities to manage methods related to waste and material footprints in a project. This module is instrumental in adding, deleting, and checking methods within the context of environmental impact assessments.

5.6.1 Function Summary

- **`AddMethods`**: This function adds new methods to a project based on entries in a custom biosphere database. It is useful for incorporating specific waste and material footprint methodologies into the project's analysis framework.
- **`DeleteMethods`**: This function removes specific methods from a project, particularly those associated with waste and material footprints, aiding in maintaining the relevance and accuracy of the project's methodological toolbox.
- **`CheckMethods`**: This function lists and checks the methods in a project, focusing on those linked to waste and material footprints, ensuring that the methods are correctly implemented and aligned with the project's objectives.

5.6.2 Important Code Snippets

- **Function to Add Methods to a Project**:

```
def AddMethods():
    # ... snippet to show structure of method keys added ...

    # Assign characterization factor based on type
    ch_factor = -1.0 if m_type == "waste" else 1.0

    # Assign method key and description based on type
    if m_type == "waste":
        name_combined = m_code.split(" ")[0] + " combined"
        method_key = (
            "WasteAndMaterialFootprint",
            "Waste: " + name_combined,
            m_code,
```

(continues on next page)

(continued from previous page)

```
        )
        description = "For estimating the waste footprint of an activity"
    else:
        method_key = ("WasteAndMaterialFootprint", "Demand: " + m_code, m_code)
        description = "For estimating the material demand footprint of an activity"

    # ... end snippet ...
```

- **Function to Delete Methods from a Project**:

```
def DeleteMethods():
    # Implementation of the function
```

- **Function to Check Methods in a Project**:

```
def CheckMethods():
    # ... snippet to show how methods are accessed ...

    methods_wasteandmaterial = [
        x for x in list(bd.methods) if "WasteAndMaterial Footprint" == x[0]
    ]

    for m in methods_wasteandmaterial:
        method = bd.Method(m)
        print(method.load())
        print(method.metadata)

    print(len(methods_wasteandmaterial))

    # ... end snippet ...
```

5.7 ExchangeEditor

The ExchangeEditor module is the final major part of the WasteAndMaterialFootprint tool, responsible for editing exchanges with wurst and Brightway2. It appends relevant exchanges from the *db_wmf* (a database containing waste and material exchange details) to activities identified by *WasteAndMaterialSearch()* in a specified project's database (*db_name*).

This module takes the longest time to run, as it iterates over each exchange that was found by the search modules, which can be >200,000 exchanges. For each database, this can take around 15-20 minutes to run.

5.7.1 Function Summary

- **`ExchangeEditor`**: This function modifies the specified project's database by appending exchanges from the *db_wmf* to activities identified by *WasteAndMaterialSearch()*. Each appended exchange replicates the same amount and unit as the original technosphere waste and material exchange.

5.7.2 Important Code Snippets

- **Function to Append Exchanges to Activities in a Database**:

```
def ExchangeEditor(project_wmf, db_name, db_wmf_name):
    # ... snippet to show addition of the custom exchange ...

    # Iterate over each category (NAME)
    for NAME, df in sorted(file_dict.items(), reverse=False):
        countNAME += 1
        progress_db = f"{countNAME:2}/{len(file_dict.items())}"
        count = 0

        # For each exchange in the current category's DataFrame
        for exc in tqdm(
            df.to_dict("records"),
            desc=f" - {progress_db} : {NAME} ",
            bar_format=bar_format,
            colour="magenta",
            smoothing=0.01,
        ):
            # Extract details of the exchange
            code, name, location, ex_name, amount, unit, ex_location, database = (
                exc["code"],
                exc["name"],
                exc["location"],
                exc["ex_name"],
                exc["ex_amount"],
                exc["ex_unit"],
                exc["ex_location"],
                db_name,
            )

            KEY = (database, code)
            WMF_KEY = (
                db_wmf_name,
                NAME.split("_")[1]
                .capitalize()
                .replace("_", " ")
                .replace("-", " ")
                .replace("kilogram", "(kg)")
                .replace("cubicmeter", "(m3)"),
            )
            # Retrieve the process and wasteandmaterial exchange from the databases
            try:
                process = bd.get_activity(KEY)
```

(continues on next page)

(continued from previous page)

```
            wasteandmaterial_ex = bd.get_activity(WMF_KEY)
            before = len(process.exchanges())

            process.new_exchange(
                input=wasteandmaterial_ex,
                amount=amount,
                unit=unit,
                type="biosphere",
            ).save()
            after = len(process.exchanges())

    # ... end of snippet ...
```

5.8 VerifyDatabase

The VerifyDatabase module contains functionality for verifying an edited WMF database within a given project in Brightway2. It performs this verification by calculating LCA scores for random activities within the specified database using selected methods.

5.8.1 Function Summary

- **`VerifyDatabase`**: This function verifies a database within a Brightway2 project by calculating LCA scores for random activities using Waste Footprint and Material Demand Footprint methods. Since it is not expected that every activity and method combination would result in a non-zero score, the function allows users to specify the number of activities and methods to be used in the verification process. The function also allows users to specify whether they want to check the Waste Footprint, Material Demand Footprint, or both.

```
def VerifyDatabase(project_name, database_name, check_material=True, check_
↪waste=True, log=True):
    # ... snippet to show verification process ... #

    print(f"\n** Verifying database {database_name} in project {project_name} **\n")

    # Loop until a non-zero score is obtained
    while lca_score == 0 and count < 5:
        try:
            count += 1
            # Get a random activity from the database
            act = bd.Database(database_name).random()

            # Initialize the list of methods
            methods = []

            # Find methods related to Waste Footprint
            if check_waste:
                methods_waste = [x for x in bd.methods if "Waste" in x[1]]
                methods += methods_waste

            # Find methods related to Material Demand Footprint
```

(continues on next page)

(continued from previous page)

```
            if check_material:
                methods_material = [x for x in bd.methods if "Demand" in x[1]]
                methods += methods_material

            if not check_waste and not check_material:
                method = bd.methods.random()
                methods.append(method)

            # Choose a random method
            method = choice(methods)

            # Perform LCA calculation
            lca = bc.LCA({act: 1}, method)
            lca.lci()
            lca.lcia()

            # Get the lca score
            lca_score = lca.score

            # Print the result
            log_statement = f"\tScore: {lca_score:2.2e} \n\tMethod: {method[2]} \n\
↪tActivity: {act['name']} \n\tDatabase: {database_name}\n"

        except Exception as e:
            # Print any errors that occur
            log_statement = (
                f"@@@@@@@@@  Error occurred with '{database_name}': {e}! @@@@@@@@@"
            )

    # ...  end snippet ... #
```

5.8.2 Usage

This function is called automatically after each database is processed, and again after all databases have been processed. The function can also be called manually by the user by invoking the following command:

```
wmf.VerifyDatabase(project_name, database_name, check_material=True, check_waste=True,␣
↪log=True)
```

5.9 FutureScenarios

Contains scenarios for future environmental impact assessments.

5.10 ExplodeDatabase

Responsible for expanding a Brightway2 database into detailed exchange lists.

5.11 SearchWaste

Provides functions for searching and categorizing waste-related data.

5.12 SearchMaterial

Enables detailed search and categorization of material-related data within a database.

5.13 MakeCustomDatabase

Facilitates the creation of custom databases for use in environmental impact assessments.

5.14 MethodEditor

Manages methods within the WMF program for waste and material footprint calculations.

5.15 ExchangeEditor

Handles the editing and appending of exchanges in Brightway2 databases.

5.16 VerifyDatabase

Performs verification of databases by calculating LCA scores for random activities.

CHAPTER

SIX

EXAMPLES

See the *examples* directory for an examples of how to use the WasteAndMaterialFootprint package. The folder *batteries* contains a small case study using the package to calculate the waste and material footprints of several battery technologies in ecoinvent 3.10.

CHAPTER

SEVEN

API REFERENCE

7.1 WMFootprint package

7.1.1 WasteAndMaterialFootprint.main module

main Module

Main module of the *WasteAndMaterialFootprint* tool.

This script serves as the entry point for the *WasteAndMaterialFootprint* tool. It orchestrates the overall process, including the setup and execution of various subprocesses like database explosion, material and waste searches, and the editing of exchanges.

The script supports both single and multiple project/database modes, as well as the option to use multiprocessing. It also facilitates the use of the premise module to generate future scenario databases.

Customisation:

- Project and database names, and other settings can be edited in *config/user_settings.py*.
- Waste search query terms can be customised in *config/queries_waste.py*.
- The list of materials can be modified in *config/queries_materials.py*.

Usage:

To use the default settings, run the script with *python main.py*. Arguments can be provided to change project/database names or to delete the project before running.

`WasteAndMaterialFootprint.main.`**`EditExchanges`**`(`*`args`*`)`

Edit exchanges in the database.

This function adds waste and material flows to the activities and verifies the database.

Parameters
`args` – Dictionary containing database and project settings.

Returns
None

WasteAndMaterialFootprint.main.**ExplodeAndSearch**(*args*)

Exploding the database into separate exchanges, searching for waste and material flows, and processing these results.

This includes:

- ExplodeDatabase.py
- SearchWaste.py
- SearchMaterial.py

Parameters
args – Dictionary containing database and project settings.

Returns
None

WasteAndMaterialFootprint.main.**run**()

Main function serving as the wrapper for the WasteAndMaterialFootprint tool.

This function coordinates the various components of the tool, including:
creating future scenario databases, setting up and processing each database for waste and material footprinting, and combining results into a custom database. adding LCIA methods to the project for each of the waste/material flows.

The function supports various modes of operation based on the settings in *config/user_settings.py*. Specifications for material and waste searches can be customised in *queries_materials*.

7.1.2 Submodules

7.1.3 WasteAndMaterialFootprint.ExchangeEditor module

ExchangeEditor Module

This module is responsible for editing exchanges with wurst and Brightway2. It appends relevant exchanges from the *db_wmf* (database containing waste and material exchange details) to activities identified by *WasteAndMaterialSearch()* in the specified project's database (*db_name*). Each appended exchange replicates the same amount and unit as the original technosphere waste and material exchange.

WasteAndMaterialFootprint.ExchangeEditor.**ExchangeEditor**(*project_wmf*, *db_name*, *db_wmf_name*)

Append relevant exchanges from *db_wmf* to each activity in *db_name* identified by *WasteAndMaterialSearch()*.

This function modifies the specified project's database by appending exchanges from the *db_wmf* to activities identified by *WasteAndMaterialSearch()*. The appended exchanges mirror the quantity and unit of the original technosphere waste and material exchange.

Parameters

- **project_wmf** (*str*) – Name of the Brightway2 project to be modified.
- **db_name** (*str*) – Name of the database within the project where activities and exchanges are stored.
- **db_wmf_name** (*str*) – Name of the database containing waste and material exchange details.

Returns
None. Modifies the given Brightway2 project by appending exchanges and logs statistics about the added exchanges.

Return type
None

Raises
Exception – If any specified process or exchange is not found in the database.

7.1.4 WasteAndMaterialFootprint.ExplodeDatabase module

ExplodeDatabase Module

This module is responsible for exploding a Brightway2 database into a single-level list of all exchanges. It utilizes the wurst package to unpack the database, explode it to a list of all exchanges, and save this data in a DataFrame as a .pickle binary file.

WasteAndMaterialFootprint.ExplodeDatabase.**ExplodeDatabase**(*db_name*)

Explode a Brightway2 database into a single-level list of all exchanges using wurst.

Parameters
db_name (*str*) – Name of the Brightway2 database to be exploded.

Returns
None The function saves the output to a file and logs the operation, but does not return any value.

Return type
None

7.1.5 WasteAndMaterialFootprint.FutureScenarios module

FutureScenarios Module

This module is responsible for creating future databases with premise.

WasteAndMaterialFootprint.FutureScenarios.**FutureScenarios**(*scenario_list*)

Create future databases with premise.

This function processes scenarios and creates new databases based on the premise module. It configures and uses user-defined settings and parameters for database creation and scenario processing.

Returns
None The function does not return any value but performs operations to create and configure databases in Brightway2 based on specified scenarios.

Raises
Exception – If an error occurs during the processing of scenarios or database creation.

WasteAndMaterialFootprint.FutureScenarios.**MakeFutureScenarios**()

Main function to run the FutureScenarios module. Only activated if *use_premise* is set to True in *user_settings.py*.

Calls the *FutureScenarios* function to create new databases based on the list of scenarios and settings specified in *user_settings.py*.

WasteAndMaterialFootprint.FutureScenarios.**check_existing**(*desired_scenarios*)

Check the project to see if the desired scenarios already exist, and if so, remove them from the list of scenarios to be created. Quite useful when running many scenarios, as it can take a long time to create them all, sometimes crashes, etc.

args: desired_scenarios (list): list of dictionaries with scenario details

returns: new_scenarios (list): list of dictionaries with scenario details that do not already exist in the project

WasteAndMaterialFootprint.FutureScenarios.**make_possible_scenario_list**(*filenames*, *desired_scenarios*, *years*)

Make a list of dictionaries with scenario details based on the available scenarios and the desired scenarios.

args: filenames (list): list of filenames of available scenarios desired_scenarios (list): list of dictionaries with scenario details years (list): list of years to be used

returns: scenarios (list): list of dictionaries with scenario details that are available and desired

7.1.6 WasteAndMaterialFootprint.MakeCustomDatabase module

MakeCustomDatabase Module

This module contains functions for creating an xlsx representation of a Brightway2 database and importing it into Brightway2.

Main functions: - dbWriteExcel: Creates an xlsx file representing a custom Brightway2 database. - dbExcel2BW: Imports the custom database (created by dbWriteExcel) into Brightway2.

WasteAndMaterialFootprint.MakeCustomDatabase.**dbExcel2BW**()

Import the custom database (created by dbWriteExcel) into Brightway2.

This function imports a custom Brightway2 database from an Excel file into the Brightway2 software, making it available for further environmental impact analysis.

Returns
None

WasteAndMaterialFootprint.MakeCustomDatabase.**dbWriteExcel**()

Create an xlsx file representing a custom Brightway2 database.

This function generates an Excel file which represents a custom database for Brightway2, using predefined directory and database settings.

Returns
Path to the generated xlsx file.

WasteAndMaterialFootprint.MakeCustomDatabase.**determine_unit_from_name**(*name*)

Determine the unit based on the name.

Parameters
name – The name from which to infer the unit.

Returns
The inferred unit as a string.

WasteAndMaterialFootprint.MakeCustomDatabase.**get_files_from_tree**(*dir_searchmaterial_results*, *dir_searchwaste_results*)

Collects filenames from the SearchMaterial and SearchWasteResults directories.

Parameters

- **dir_searchmaterial_results** – Directory path for SearchMaterial results.
- **dir_searchwaste_results** – Directory path for SearchWasteResults.

Returns
Sorted list of filenames.

7.1.7 WasteAndMaterialFootprint.MethodEditor module

MethodEditor Module

This module provides functions for adding, deleting, and checking methods related to waste and material footprints in a project.

Function Summary:

- *AddMethods*: Adds new methods to a project based on a custom biosphere database.
- *DeleteMethods*: Removes specific methods from a project, particularly those related to waste and material footprints.
- *CheckMethods*: Lists and checks the methods in a project, focusing on those associated with waste and material footprints.

WasteAndMaterialFootprint.MethodEditor.**AddMethods**()

Add methods to the specified project based on entries in the custom biosphere database.

Parameters

- **project_wmf** – Name of the project.
- **db_wmf_name** – Name of the database.

WasteAndMaterialFootprint.MethodEditor.**CheckMethods**()

Check methods associated with the "WasteAndMaterial Footprint" in the specified project.

Parameters
project_wmf – Name of the project.

WasteAndMaterialFootprint.MethodEditor.**DeleteMethods**()

Delete methods associated with the "WasteAndMaterial Footprint" in the specified project.

Parameters
project_wmf – Name of the project.

7.1.8 WasteAndMaterialFootprint.SearchMaterial module

SearchMaterial Module

This script loads data from '<db name>_exploded.pickle', runs search queries, and produces a CSV to store the results and a log entry. The search queries are formatted as dictionaries with fields NAME, CODE, and search terms keywords_AND, keywords_OR, and keywords_NOT. These queries are defined in *config/queries_waste.py*.

WasteAndMaterialFootprint.SearchMaterial.**SearchMaterial**(*db_name*, *project_wmf*)

Search for materials in a specified database and extract related information.

This function takes a database name as input, sets the project to the respective database, and looks for activities involving a predefined list of materials. It extracts relevant details of these activities, such as ISIC and CPC classifications, and saves the details to a CSV file. It also extracts related material exchanges and saves them to another CSV file.

Parameters

- **db_name** – The name of the database to search in.

- **project_wmf** – The Brightway2 project to set as current for the search.

Returns
None

Raises
Exception – If there is any error in reading the materials list from the file.

7.1.9 WasteAndMaterialFootprint.SearchWaste module

SearchWaste Module

This script loads data from '<db name>_exploded.pickle', runs search queries, and produces CSV files to store the results and a log entry. The search queries are formatted as dictionaries with fields NAME, CODE, and search terms keywords_AND, keywords_OR, and keywords_NOT. These queries are defined in *config/queries_waste.py*.

Functionality

Provides a function, *SearchWaste()*, that loads data from '<db name>_exploded.pickle', runs search queries, and produces result CSVs and log entries.

WasteAndMaterialFootprint.SearchWaste.**SearchWaste**(*db_name*, *dir_searchwaste_results=PosixPath('/home/stew/code/gh/WasteAndM*

Load data from '<db name>_exploded.pickle', run search queries, and produce result CSVs and log entries.

This function processes waste-related data from a given database and runs predefined queries to identify relevant waste exchanges. The results are saved in CSV files and log entries are created for each search operation.

Parameters
db_name (*str*) – The database name to be used in the search operation.

Note: The queries are defined in *config/queries_waste.py*.

7.1.10 WasteAndMaterialFootprint.VerifyDatabase module

VerifyDatabase Module

This module contains a function to verify a (WasteAndMaterialFootprint) database within a given project in Brightway2. It performs a verification by calculating LCA scores for random activities within the specified database using selected methods.

WasteAndMaterialFootprint.VerifyDatabase.**VerifyDatabase**(*project_name*, *database_name*, *check_material=True*, *check_waste=True*, *log=True*)

Verifies a database within a given project in Brightway2 by calculating LCA scores for random activities using selected methods.

This function assesses the integrity and validity of a specified database within a Brightway2 project. It performs LCA calculations on random activities using Waste Footprint and Material Demand Footprint methods, and logs the results.

Parameters

- **project_name** (*str*) – The name of the Brightway2 project.
- **database_name** (*str*) – The name of the database to be verified.

- **check_material** (*bool*) – If True, checks for Material Demand Footprint methods.
- **check_waste** (*bool*) – If True, checks for Waste Footprint methods.
- **log** (*bool*) – If True, logs the results.

Returns

Exit code (0 for success, 1 for failure).

CHAPTER

EIGHT

CONFIGURATION

8.1 Configuration

8.1.1 config.user_settings module

user_settings Module

Configure Project and Database Settings for WasteAndMaterialFootprint tool.

This script is used to configure the project and database settings, as well as set up the essential paths for the data, config, and result directories.

The script allows for two modes of operation:

1. Single Mode (*single* is True): Set the project and database names to a single specified value.
2. Multiple projects/databases mode (*single* is False): Facilitates batch processing of multiple projects and databases.

Additionally, the script allows for the use of multiprocessing (*use_multiprocessing* is True).

Premise can also be used to make future databases (*use_premise* is True).

`config.user_settings.`**`generate_args_list`**`(`*single_database=None*`)`

Generate a list of argument dictionaries for processing multiple projects and databases.

This function is used when the tool is set to operate in multiple projects/databases mode. It generates a list of argument dictionaries, each containing project and database settings.

Returns

A list of dictionaries with project and database settings for batch processing.

```
"""
user_settings Module
====================

Configure Project and Database Settings for WasteAndMaterialFootprint tool.

This script is used to configure the project and database settings, as well as set up␣
↪the essential paths for the data, config, and result directories.

The script allows for two modes of operation:
    1. Single Mode (`single` is True): Set the project and database names to a single␣
↪specified value.
```

(continues on next page)

(continued from previous page)

```
    2. Multiple projects/databases mode (`single` is False): Facilitates batch processing␣
↪of multiple projects and databases.

Additionally, the script allows for the use of multiprocessing (`use_multiprocessing` is␣
↪True).

Premise can also be used to make future databases (`use_premise` is True).


"""

import os
import shutil
from itertools import product
from pathlib import Path

# ============================================================
# ------------------------------------------------------------
## SETTINGS FOR BRIGHTWAY2

# you can set the brightway2 directory here, otherwise it will use the default one
# custom_bw2_dir = os.path.expanduser("~") + '/brightway2data'
custom_bw2_dir = None
if custom_bw2_dir:
    os.environ["BRIGHTWAY2_DIR"] = custom_bw2_dir

import bw2data as bd

# ------------------------------------------------------------
## SETTINGS FOR PROJECTS
# set project name and other things here

project_premise_base = "default"
project_premise = "SSP-cutoff"
project_base = project_premise
# if you want to use the same project for the wmf tool, change this to project_base,␣
↪otherwise, it will create a new project
project_wmf = f"WMFootprint-{project_base}"

# ------------------------------------------------------------
## SETTINGS FOR THE WASTEANDMATERIAL FOOTPRINT TOOL
use_wmf = False
db_wmf_name = "WasteAndMaterialFootprint"  # name of the database that will be created␣
↪(pseudobiosphere)

# if you only want to run one database, set single to True and choose the database name␣
↪here
single = False
single_database = "ecoinvent_cutoff_3.9_remind_SSP5-Base_2050"

# if you want to use a fresh project
delete_wmf_project = False
```

(continues on next page)

(continued from previous page)

```
use_multiprocessing = False
verbose = False

# set these to True if you want to run the different parts of the tool separately
do_search = True
do_methods = True
do_edit = True

# ------------------------------------------------------------
## PREMISE SETTINGS -  to construct future LCA databases with premise

use_premise = True

# this will be the database that premise will use to construct the future databases
database_name = "ecoinvent-3.9.1-cutoff"

# if you want to use a fresh project
delete_existing_premise_project = False

# if you want to use multiprocessing (some people have reported problems with this)
use_mp = True

# if you want to give premise multiple databases at once, increase this number,␣
↪otherwise, leave it at 1. From my experience, it is better to give it one database at␣
↪a time, otherwise memory issues can occur.
batch_size = 1

# This seems not to have much effect, because most of the print statemenents are in␣
↪`wurst`, not in `premise`
premise_quiet = True

if use_premise and project_base != project_premise:
    project_wmf = f"WMFootprint-{project_premise}"

# Get the premise key (at the moment, it is stored in the code to make it easier for␣
↪people, but it would be better to have it in a file)
premise_key = "tUePmX_S5B8ieZkkM7WUU2CnO8SmShwmAeWK9x2rTFo="
if premise_key is None:
    key_path = Path(__file__).parents[1] / ".secrets" / "premise_key.txt"
    with open(key_path, "r") as f:
        premise_key = f.read()


# ************************************************************
##  CHOOSE THE SCENARIOS YOU WANT TO USE WITH PREMISE

# Details of the scenarios can be found here:
# carbonbrief.org/explainer-how-shared-socioeconomic-pathways-explore-future-climate-
↪change/
# https://premise.readthedocs.io/en/latest/
# https://premisedash-6f5a0259c487.herokuapp.com/ (there is a nice dashboard here to␣
↪explore the scenarios)
```

(continues on next page)

(continued from previous page)

```
# Comment out the scenarios you don't want to use, otherwise all potential scenarios will␣
↪be attempted
# The full list of available scenarios is in the `filenames` variable (at the moment␣
↪there are 15, but there could be more in future updates)
# Not all combinations are available, the code in FutureScenarios.py will filter out the␣
↪scenarios that are not possible

models = [
    # "image",
    "remind",
]

ssps = [
    # "SSP1",
    "SSP2",
    # "SSP5",
]

# default is to have an optimistic and a pessimistic scenario
rcps = [
    "Base",
    # "RCP19",
    # "RCP26",
    # "NPi",
    # "NDC",
    "PkBudg500",
    # "PkBudg1150",
]

# If the years you put here are inside the range of the scenario, it will interpolate␣
↪the data, otherwise, probably it fails. Most of the scenarios are between 2020 and␣
↪2100, I think. 5 year intervals until 2050, then 10 year intervals until 2100.

years = [
    # 2020,
    # 2025,
    2030,
    # 2035,
    # 2040,
    # 2045,
    # 2050,
    # 2060,
    2065,
    # 2070,
    # 2080,
    # 2090,
    2100,
]

# this part makes all the possible combinations of the scenarios you want to use, the␣
```

(continues on next page)

(continued from previous page)

```
↪will be filtered out later if they are not possible

desired_scenarios = []
for model, ssp, rcp in product(models, ssps, rcps):
    desired_scenarios.append({"model": model, "pathway": ssp + "-" + rcp})


# Set the arguments list of projects and databases to be processed with the WMF tool
def generate_args_list(single_database=None):
    """
    Generate a list of argument dictionaries for processing multiple projects and␣
↪databases.

    This function is used when the tool is set to operate in multiple projects/databases␣
↪mode.
    It generates a list of argument dictionaries, each containing project and database␣
↪settings.

    :returns: A list of dictionaries with project and database settings for batch␣
↪processing.
    """
    bd.projects.set_current(project_base)
    if single:
        databases = [single_database]

    else:
        exclude = [
            "biosphere",
            db_wmf_name,  # this is the database that will be created by the wmf tool
        ]
        databases = sorted(
            [x for x in bd.databases if not any(sub in x for sub in exclude)]
        )

    args_list = []
    for database in databases:
        args = {
            "project_base": project_base,
            "project_wmf": project_wmf,
            "db_name": database,
            "db_wmf_name": db_wmf_name,
        }
        args_list.append(args)

    return args_list


# ----------------------------------------------------------------
## GENERAL DIRECTORY PATHS
# Set the paths (to the data, config, logs, and the results)
# By default, the program will create new directories in the working directory
```

(continues on next page)

(continued from previous page)

```
# Get the directory of the main script
cwd = Path.cwd()
# Set the paths
dir_config = cwd / "config"

list_materials = dir_config / "list_materials.txt"

dir_data = cwd / "data"

dir_tmp = dir_data / "tmp"
dir_logs = dir_data / "logs"

dir_searchwaste_results = dir_data / "SearchWasteResults"
dir_searchmaterial_results = dir_data / "SearchMaterialResults"
dir_databases_wasteandmaterial = dir_data / "DatabasesWasteAndMaterial"

dir_wmf = [
    dir_tmp,
    dir_logs,
    dir_searchwaste_results,
    dir_searchmaterial_results,
    dir_databases_wasteandmaterial,
]

# this will delete old results and logs
if delete_wmf_project:
    for dir in dir_wmf:
        if os.path.exists(dir):
            shutil.rmtree(dir)


# FIN #
# =============================================================
```

8.1.2 config.queries_waste module

queries_waste Module

This module defines the search parameters for each waste and material flow category. It is used in conjunction with *SearchWaste.py* and requires a .pickle file generated by *ExplodeDatabase.py*.

The queries are set up for different waste flow categories like digestion, composting, incineration, recycling, and landfill, among others. Each query is a dictionary containing search terms for the respective category.

The module creates two sets of queries: 1. *queries_kg* for waste flows measured in kilograms. 2. *queries_m3* for waste flows measured in cubic meters.

These queries are combined into the *queries_waste* list, which is then used by *SearchWaste.py* for filtering and extracting relevant data from the database.

```
"""
queries_waste Module
====================

This module defines the search parameters for each waste and material flow category. It␣
↪is used in conjunction with `SearchWaste.py` and requires a .pickle file generated by␣
↪`ExplodeDatabase.py`.

The queries are set up for different waste flow categories like digestion, composting,␣
↪incineration, recycling, and landfill, among others. Each query is a dictionary␣
↪containing search terms for the respective category.

The module creates two sets of queries:
1. `queries_kg` for waste flows measured in kilograms.
2. `queries_m3` for waste flows measured in cubic meters.

These queries are combined into the `queries_waste` list, which is then used by␣
↪`SearchWaste.py` for filtering and extracting relevant data from the database.

"""

names = [
    "digestion",
    "composting",
    "open burning",
    "incineration",
    "recycling",
    "landfill",
    "hazardous",
    # "non-hazardous", # not really needed, as it is just "total" - "hazardous", and it␣
↪makes the process slower (20k+ more exchanges)
    "carbon dioxide",  # in prospective databases, carbon capture and storage is included
    "total",
]

# QUERY FORMAT
# setup the dictionary of search terms for
# each waste and material flow category

queries_kg = []
for name in names:
    query = {
        "db_name": "",  # db_name
        "db_custom": "",  # db_wmf_name
        "name": "",
        "code": "",
        "unit": "kilogram",
        "AND": ["waste"],
        # if you replace "None" below, it must be with a with a list of strings, like␣
↪the other keywords have
        "OR": None,
        "NOT": None,
    }
```

(continues on next page)

(continued from previous page)

```
    # define here what the search parameters mean for
    # each waste and material flow category
    # if you want to customize the search parameters, you will
    # likely need some trial and error to make sure you get what you want

    query.update({"name": "WasteFootprint_" + name})
    if "landfill" in name:
        query.update({"OR": ["landfill", "dumped", "deposit"]})
    if "hazardous" == name:
        query.update({"OR": ["hazardous", "radioactive"]})
        query.update({"NOT": ["non-hazardous", "non-radioactive"]})
    if "non-hazardous" == name:
        query.update({"NOT": ["hazardous", "radioactive"]})
    if "incineration" in name:
        query["AND"] += ["incineration"]
    if "open burning" in name:
        query["AND"] += ["burning"]
    if "recycling" in name:
        query["AND"] += ["recycling"]
    if "composting" in name:
        query["AND"] += ["composting"]
    if "digestion" in name:
        query["AND"] += ["digestion"]
    if "radioactive" in name:
        query["AND"] += ["radioactive"]
    if "carbon dioxide" in name:
        query.update(
            {
                "OR": [
                    "carbon dioxide storage",
                    "carbon dioxide capture",
                    "carbon dioxide, captured",
                ]
            }
        )
        query.update({"NOT": ["methane"]})
        query.update({"AND": ""})

    # add the query to the list of queries
    queries_kg.append(query)

# add same queries defined above, now for liquid waste and material
queries_m3 = []
for q in queries_kg:
    q = q.copy()
    q.update({"unit": "cubic meter"})
    queries_m3.append(q)

queries_waste = queries_kg + queries_m3
```

8.1.3 config.queries_materials module

queries_materials Module

This module contains the list of materials that will be used to create demand methods in the WasteAndMaterialFootprint tool. The materials are specifically chosen to match with the activities in the ecoinvent v3.9.1 database (and the future databases made with premise) and to align with the EU CRM list 2023. Additional strategic materials are also included, as well as other materials of interest.

Note:

- The search function in *SearchMaterial.py* uses 'startswith', so it will catch all activities that start with the material name.
- The comma in material names is important for filtering specific activities in the database.
- If an intermediate product is included, the inputs and outputs might cancel out, resulting in a zero result.

queries_materials

List of tuples where each tuple contains the activity name (as appears in the database) and its corresponding material category. This list is used for creating demand methods in the WasteAndMaterialFootprint tool.

```
"""
queries_materials Module
========================

This module contains the list of materials that will be used to create demand methods in␣
↪the WasteAndMaterialFootprint tool. The materials are specifically chosen to match␣
↪with the activities in the ecoinvent v3.9.1 database (and the future databases made␣
↪with premise) and to align with the EU CRM list 2023. Additional strategic materials␣
↪are also included, as well as other materials of interest.

Note:
-----
- The search function in `SearchMaterial.py` uses 'startswith', so it will catch all␣
↪activities that start with the material name.
- The comma in material names is important for filtering specific activities in the␣
↪database.

queries_materials
-----------------
List of tuples where each tuple contains the activity name (as appears in the database)␣
↪and its corresponding material category. This list is used for creating demand methods␣
↪in the WasteAndMaterialFootprint tool.
"""

queries_materials = [
    ("market for aluminium", "aluminium"),
    ("market for antimony", "antimony"),
    ("market for bauxite", "bauxite"),
    ("market for beryllium", "beryllium"),
```

(continues on next page)

(continued from previous page)

```
    ("market for bismuth", "bismuth"),
    ("market for cadmium", "cadmium"),
    ("market for calcium borates", "borates"),
    ("market for cement", "cement"),
    ("market for cerium", "cerium"),
    ("market for chromium", "chromium"),
    ("market for coal", "coal"),
    ("market for cobalt", "cobalt"),
    ("market for coke", "coke"),
    ("market for copper", "copper"),
    ("market for dysprosium", "dysprosium"),
    ("market for erbium", "erbium"),
    ("market for europium", "europium"),
    ("market for electricity,", "electricity"),
    ("market for ferroniobium,", "niobium"),
    ("market for fluorspar,", "fluorspar"),
    ("market for gadolinium", "gadolinium"),
    ("market for gallium", "gallium"),
    ("market for gold", "gold"),
    ("market for graphite", "graphite"),
    ("market for hafnium", "hafnium"),
    ("market for helium", "helium"),
    ("market for holmium", "holmium"),
    ("market for hydrogen,", "hydrogen"),
    ("market for indium", "indium"),
    ("market for latex", "latex"),
    ("market for lithium", "lithium"),
    ("market for magnesium", "magnesium"),
    ("market for natural gas,", "natural gas"),
    ("market for nickel", "nickel"),
    ("market for palladium", "palladium"),
    ("market for petroleum", "petroleum"),
    ("market for phosphate", "phosphate rock"),
    ("market for platinum", "platinum"),
    ("market for rare earth", "rare earth"),
    ("market for rhodium", "rhodium"),
    ("market for sand", "sand"),
    ("market for selenium", "selenium"),
    ("market for scandium", "scandium"),
    ("market for silicon", "silicon"),
    ("market for silver", "silver"),
    ("market for sodium borates", "borates"),
    ("market for strontium", "strontium"),
    ("market for tantalum", "tantalum"),
    ("market for tellurium", "tellurium"),
    ("market for tin", "tin"),
    ("market for titanium", "titanium"),
    ("market for uranium", "uranium"),
    ("market for tungsten", "tungsten"),
    ("market for vanadium", "vanadium"),
    ("market for vegetable oil,", "vegetable oil"),
    ("market for tap water", "water"),
```

(continues on next page)

(continued from previous page)

```
    ("market for water,", "water"),
    ("market for zinc", "zinc"),
    ("market for zirconium", "zirconium"),
]
```

CHAPTER

NINE

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

C

W

INDEX

V

W