CoLibri Command Line Documentation
Version 8.3

Sascha Jung, Raphael Klein & Marcus Gastreich

August 19, 2024

Contents

1 Introduction **2**

2 Technical Prerequisites **3**

3 The ReactionSynthesizer **4**
3.1 General . . . 4
3.2 Program Options . . . 5
3.3 Example 1: Generating a single space for a reaction . . . 7
3.4 Example 2: Enumerating product molecules for a reaction . . . 8
3.5 Example 3: Transforming molecules for FastGrow databases or covalent docking . . . 8
3.6 Example 4: Removing protecting groups from product molecules . . . 10
3.7 Example 5: Debugging reaction definitions . . . 10
3.8 Example 6: Using reactions with functional group labels . . . 11

4 The FragspaceMerger **12**
4.1 General . . . 12
4.2 Program Options . . . 12
4.3 Example: Merging two single reaction spaces . . . 13

5 The SpaceLightDBCreator **14**
5.1 General . . . 14
5.2 Program Options . . . 14
5.3 Example: Creating a topological fragment space database . . . 16

6 Prepare Reaction Definitions for Space Generation **17**

7 General Options **21**

8 Further Reading, References **22**

1 Introduction

All links, references, table of contents lines etc. in this document are clickable.

Please note that this toolkit is a command line package.

CoLibri is a toolkit containing three separate command line (CLI) tools (React-ionSynthesizer, Fragspace-Merger and SpaceLightDBCreator) to *create* combinatorial chemical spaces (”fragment spaces” or ”topological fragment spaces”) of multi-billion size and beyond. As input, the tools take building blocks and a formalized reaction definition (as `.smirks` or `.rxn` file). This input is transformed into a highly efficient description of a chemical space (either based on link compatibility or topology graphs, see below) that can then be searched either with our CLI tools SpaceLight, SpaceMACS and FTrees or with our Chemical Space Navigation Platform infiniSee. The spaces are encoded based on formalized reactions containing reagents with Lego®-like linkers (Figure 1). This type of spaces is the only way to access an immeasurably large pool of compounds and find numerous synthesizable analogs. Therefore, CoLibri is a mandatory toolkit to **create your own corporate chemical spaces** that can be searched by fingerprint similarity (SpaceLight, infiniSee’s Analog Hunter), by pharmacophore-based similarity (FTrees, infiniSee’s Scaffold Hopper) or by maximum common substructure (SpaceMACS, infiniSee’s Motif Matcher).

CoLibri contains three independent tools that let you

- build combinatorial spaces for FTrees and SpaceMACS (ReactionSynthesizer and FragspaceMerger, see Section 3.3 and 4.3)
- build combinatorial spaces for SpaceLight (SpaceLightDBCreator, see Section 5.3)
- enumerate up to 1 million product molecules per reaction (ReactionSynthesizer, see Section 3.4)
- generate fragments for FastGrow databases (ReactionSynthesizer, see Section 3.5)
- prepare compounds for covalent docking (ReactionSynthesizer, see Section 3.5)

To enable chemical space navigation at unparalleled speed on standard hardware, we formalize reactions and encode them as pseudo-linking reactions. There is a principal difference between ”classical” fragment spaces (”FTrees spaces”) and topological fragment space databases (”SpaceLight spaces”). For the classical spaces, the compatibility of the fragments is encoded for every reaction, e.g., one single space is created per reaction (see Section 3.3). Multiple single spaces are finally merged into the final, ”parent” chemical space (see Section 4.3) containing the overall fragment compatibilities and all fragments (Figure 1, classical space). For topological spaces, one so-called ”topology graph” is generated for every reaction and stored in a database (see Section 5.3). Every node of this topology graph contains formalized, virtual building blocks, or as we sometimes call them, ”reaction-fate foreseeing” fragments (Figure 1, topological fragment space).

Despite being processed in different ways, the input to generate both classical spaces and topological spaces is the same (see Figure 1). All you need are building blocks or synthons and appropriate reaction definitions that combine the building blocks to virtual molecules in a chemically meaningful way (see Section 6). From this input, the CoLibri tools can create optimized combinatorial fragment spaces containing trillions of virtual molecules and more that can be searched in seconds to minutes on modest hardware.

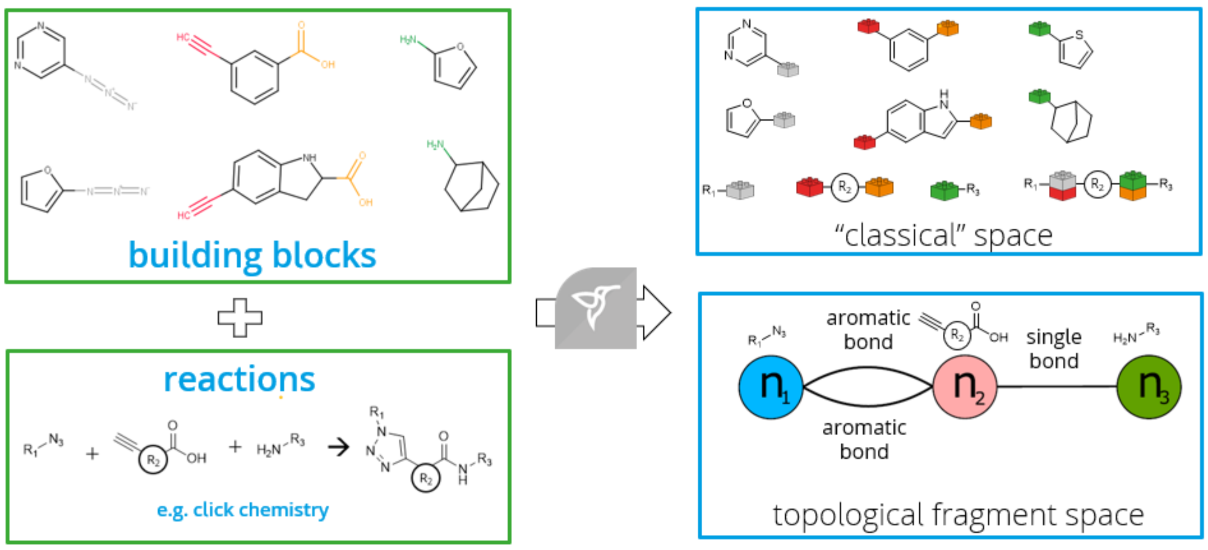


Figure 1: *Example reaction (click chemistry and amide coupling) and building blocks serving as input for the CoLibri toolkit.*

2 Technical Prerequisites

CoLibri contains three command line tools (ReactionSynthesizer, FragspaceMerger and SpacelightDB-Creator). The tools need the following to run:

- The **CoLibri package**
  (`https://www.biosolveit.de/download/?product=colibri`)
  Depending on your operating system, some libraries may have to be installed (get in touch with us: `mailto:support@biosolveit.com`)
- A **shell** (Linux/Unix) or a terminal (macOS), or a command line environment (Windows; e.g.: cmd.exe)
- A valid **license** (from `mailto:license@biosolveit.com`)

The license setup instructions will come with the license that we will send out — or has already been sent out to you. A "test license" that you can request online and that is sent to you instantaneously can simply be placed next to the respective executable (e.g. reaction_synthesizer.exe, reaction_synthesizer, or ReactionSynthesizer — depending on your operating system). For macOS please read on...

macOS Specialties On macOS, the executable will typically reside inside the *.app package, e.g.:

```
/Applications/ReactionSynthesizer.app/Contents/MacOS/ReactionSynthesizer
```

To place the short term test license there, you will have to go into the *.app package using a right mouse click on *.app in the Finder, and click on "Show package contents". In there, you will see the Contents/ subfolder, in there the MacOS subfolder, and in there, the respective executable. If you are about to use the **test license**, place it right there, next to the executable. A longer term license will be handled separately, we will tell you how when we send out that license to you.

When you call a tool for the first time, go to the Finder, and navigate to the Applications folder. Do a right(!) click on respective *.app, and — if applicable — confirm that you want to open the program. It will flash up once, and you are good to go at the terminal prompt from there on.

3 The ReactionSynthesizer

The ReactionSynthesizer generates a ”space” for a single reaction (single space) and/or enumerates up to 1 million product molecules for that reaction (see Section 3.3 and 3.4). It can also be used to prepare compounds for FastGrow databases or covalent docking (see Section 3.5).

3.1 General

An overview of all command line options is available by calling `--help`:

```
./reaction_synthesizer --help

Program options:
-r [ --reaction ] arg                 Input reaction definition file. Supported file types are *.rxn and
                                      *.smirks.
-s [ --sma ] arg                      SMARTS functional group definition file. Needed if atom labels are
                                      used in RXN reactions.
-i [ --input ] arg                    Input molecule files to be searched for reagents. Supported file
                                      types are *.smi, *.smiles, *.mol, *.mol2 and *.sdf.
--input-reagent-1 arg                 Optional input molecule files explicitly used for reagent 1.
--input-reagent-2 arg                 Optional input molecule files explicitly used for reagent 2.
--input-reagent-3 arg                 Optional input molecule files explicitly used for reagent 3.
--input-reagent-4 arg                 Optional input molecule files explicitly used for reagent 4.
-p [ --protecting-groups ] arg        Input file with protecting groups SMARTS to be removed after
                                      clipping.
-o [ --output-path ] arg              Path for the output directory in which generated spaces, enumerated
                                      products and logs are written.
--information arg (=1)                Information level in the stats.log output file.
                                          1: Essential information
                                          2: Basic matching information
                                          3: Full matching information
--check-reaction [=arg(=1)]           Checks the reaction definition for potential failures. If this check
                                      is activated, the program stops right after the check, no further
                                      steps such as space generation or product enumeration is performed.
-e [ --enumerate ] arg                Products are enumerated if a molecule file name is provided.
                                      Supported file types are *.mol2, *.sdf, *.smi and *.smiles. The
                                      output file is stored under the output-path. The number of products
                                      is limited to 1000000.
-c [ --random-products ] arg (=0)     Enumerate as many random products as specified. Performs full
                                      enumeration (limited to 1000000) when set to 0.
                                      Note: Requires '--enumerate'
-g [ --generate-fragspace ] [=arg(=1)]
                                     Generate a fragment space for the reaction.
General options:
-h [ --help ]                         Print this help message
--license-info                        Print license info
--thread-count arg                    Maximum number of threads used for calculations. The default is to
                                      use all available cores.
--version                             Print version info
-v [ --verbosity ] arg (=2)           Set verbosity level
                                       0 [silent]
                                       1 [error]
                                       2 [warning]
                                       3 [workflow]
                                       4 [steps]
```

The abbreviated, one-letter options are preceded with one dash - whereas the longer, named options are preceded with two dashes: --. If an option needs an argument (arg), you can include or omit the equals sign. Adapt the command line usage to your operating system and shell.

3.2 Program Options

In this section the individual command line options are described in more detail. The ReactionSynthesizer generates a single space (”per-reaction” space) for one specific reaction. Multiple single spaces can then be merged into an overall ”parent” space containing all the fragments from various reactions with the FragspaceMerger (see Section 4 and example in Section 4.3). Therefore, the minimal input required to run the ReactionSynthesizer is a reaction definition file containing a formalized reaction (.smirks or .rxn format) and an input file containing appropriate building blocks that can be processed with that very reaction. A minimal call to generate a single space for a particular reaction then may look like the following:

```
./reaction_synthesizer -r <path/reaction.smirks> -i <path/building_blocks.sdf>
                       -o <path/to/output/directory> -g
```

–r [––reaction] arg Specify the input reaction file. The file must contain the formalized reaction either in `.smirks` or `.rxn` format. More information on how to properly formulate a reaction definition can be found in Section 6.

–s [––sma] arg Specify a SMARTS functional group definition file. The file is only needed if atom labels are used in `.rxn` files. A pre-compiled file (named `FunctionalGroupLabel.txt`) with common functional groups can be found in the example folder inside the ReactionSynthesizer directory. You may extend this file depending on your needs. See Section 3.8 for an example and Section 6 for more information.

–i [––input] arg Specify an input file containing building blocks that are suitable to be used with the reaction given via the `-r` option. Supported file types are `.smi`, `.smiles`, `.mol`, `.mol2` and `.sdf`. The file will be scanned for matching building blocks for all reagents specified in the reaction file. Only compatible building blocks will be read from the file, incompatible ones are skipped. Please note: You can specify multiple input files simultaneously by using the `-i` option multiple times in a row:

```
./reaction_synthesizer -i compounds_1.sdf -i building_blocks_2.smi
```

––input-reagent-1 arg Input molecule files explicitly used for reagent 1. The files should contain only building blocks which match the first reagent specified in the reaction definition given with `-r` option. Supported file types are `.smi`, `.smiles`, `.mol`, `.mol2` and `.sdf`.

––input-reagent-2 arg Input molecule files explicitly used for reagent 2. The files should contain only building blocks which match the second reagent specified in the reaction definition given with `-r` option. Supported file types are `.smi`, `.smiles`, `.mol`, `.mol2` and `.sdf`.

––input-reagent-3 arg Input molecule files explicitly used for reagent 3. The files should contain only building blocks which match the third reagent specified in the reaction definition given with `-r` option. Supported file types are `.smi`, `.smiles`, `.mol`, `.mol2` and `.sdf`.

––input-reagent-4 arg Input molecule files explicitly used for reagent 4. The files should contain only building blocks which match the fourth reagent specified in the reaction definition given with `-r` option. Supported file types are `.smi`, `.smiles`, `.mol`, `.mol2` and `.sdf`.

–p [––protecting-groups] arg With this option you can specify a file containing SMARTS definitions for protecting groups. The file should be a simple text file containing line-separated SMARTS definitions. If you provide such a file, the specified protecting groups present in the input building blocks will be clipped and removed from the generated fragments. Consequently, the protecting group is also not present in the product molecules. An example file (named `ProtectingGroups.txt`) with common protecting groups can be found in the example folder in the ReactionSynthesizer directory. You may extend this file depending on your needs. For an example usage, see Section 3.6.

–o [––output-path] arg Specify the path to an output directory to which the space files (fragment-space-file (`.fsf`), fragment-link-file (`.flf`) and fragment-files (`.smi`), if `-g` option is specified), enumerated product molecule file (if `-e` option is specified) and the log file (`stats.log`) are written. Please note: If the directory does not already exist, it will be created.

––information arg(=1) Information level for the `stats.log` output file. Takes a number (1-3) as argument:

1	Essential information
2	Basic matching information
3	Full matching information

––check-reaction If specified, the reaction given with the `-r` option is checked for potential errors. If this option is specified, the program terminates right after the check. No space generation or product enumeration will be executed even if the reaction is valid and the `-g` and `-e` options are specified. Found errors will be written to a file named `validation.log` in the output directory. Please note: The `validation.log` file is only generated if errors are found in the reaction definition file, otherwise no log file will be written.
Example call:
```
./reaction_synthesizer -r reaction.rxn --check-reaction -o out_dir
```

–e [––enumerate] arg With this option you can enumerate all product molecules (limited to 1 million compounds) that can be formed from the input building blocks with the specified reaction (`-r` option). Takes a file name as argument (supported file types are `.smi`, `.smiles`, `.mol`, `.mol2` and `.sdf`). Please note: You only need to give a name for the file here (with suffix), no path is required. The file will be written to the output directory specified with the `-o` option. See Section 3.4 for an example.

–c [––random-products] arg(=0) Randomly enumerate the specified number of products. Takes a number as argument. If set to 0, all possible product molecules will be enumerated (limited to 1 million molecules). Please note: Requires the `-e` option to be set. See Section 3.4 for an example.

–g [––generate-fragspace] Generates a fragment space for the provided reaction (`-r` option) and building blocks (`-i` or `--input-reagent` option). This will create a fragment-space-file (`.fsf`), a fragment-label-file (`.flf`) and the corresponding fragment-files (`.smi`) in the directory specified with the `-o` option. See Section 3.3 for an example. Please note: The number of generated fragment-files varies depending on the reaction definition (see `-r` option).

3.3 Example 1: Generating a single space for a reaction

In the example folder within the ReactionSynthesizer directory you can find two example reaction definition files (`amidecoupling.smirks` and `suzuki.smirks`) and a file with suitable building blocks (`building_blocks.smi`). They cover two prominent examples of reactions used in Medicinal Chemistry: amide coupling and Suzuki reaction (see Figure 2). More detailed information on how to properly formalize reactions for space generation can be found on our website.[2] To get a first impression on how the ReactionSynthesizer works, simply generate a single space for the amide coupling reaction. Therefore, on the command line, navigate into the example folder and run the following:

```
../reaction_synthesizer -r amidecoupling.smirks -i building_blocks.smi
                         -o amide_single_space -g
```

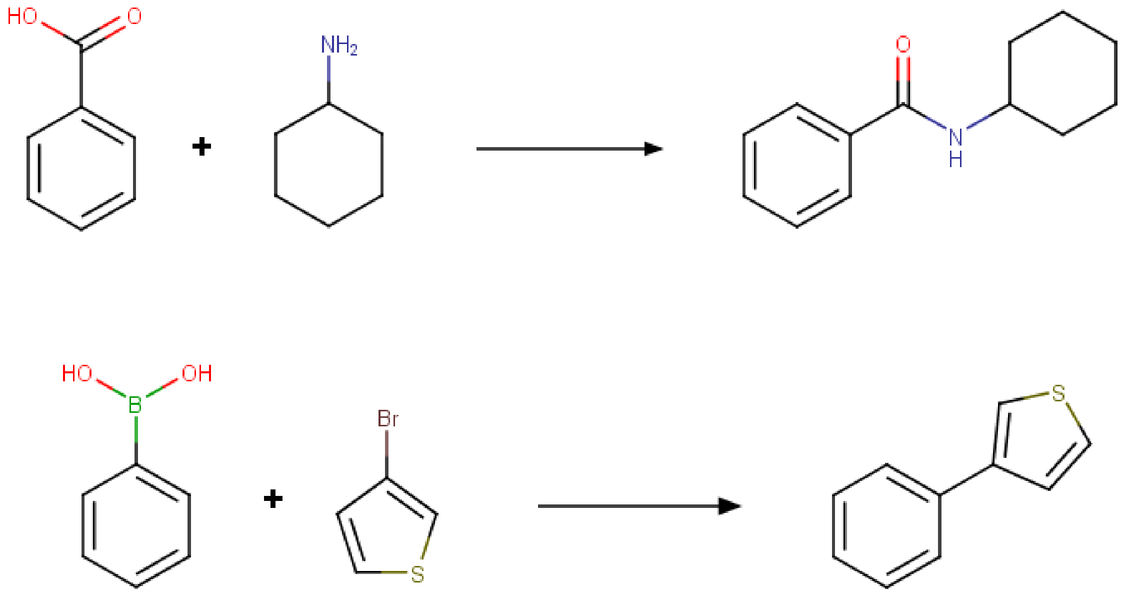


Figure 2: *Example of an amide reaction (top) and a Suzuki reaction (bottom) covered by the example reaction definition files and example building blocks.*

The above call will create a new directory named `amide_single_space` within the example folder to which several output files are written. You can open the output files with a text editor:

- `fragspace_fragments_0.smi`: fragment-file. Fragments generated from the input building blocks for the first reagent (acid). Contains all acid fragments with linkers.
- `fragspace_fragments_1.smi`: Fragments generated from the input building blocks for the second reagent (amine). Contains all amine fragments with linkers.
- `amidecoupling.flf`: fragment-label-file. Contains information to which reaction and to which reagent position the fragments belong.
- `amidecoupling.fsf`: fragment-space-file. Connects information on linker compatibilities with the corresponding fragment-label-file and fragment-files.
- `stats.log`: Contains statistics and details on the applied reaction and mapping.

Re-run for the Suzuki reaction, this time with additional enumeration of all product molecules:

```
../reaction_synthesizer -r suzuki.smirks -i building_blocks.smi
                        -o suzuki_single_space -g -e suzuki_products.sdf
```

The output folder `suzuki_single_space` will now also contain one file named `suzuki_products.sdf` with all products that could be created from the input building blocks.

3.4 Example 2: Enumerating product molecules for a reaction

The ReactionSynthesizer can also be used to enumerate product molecules from building blocks (see also Section 3.3). Simply use the amide coupling reaction from Example 1 again:

```
../reaction_synthesizer -r amidecoupling.smirks -i building_blocks.smi
                        -o amide_enum -e amide_products.sdf
```

This call will generate an output folder named `amide_enum` which contains two files: `amide_products.sdf` containing all the product molecules that could be constructed from suitable building blocks in the input file and the `stats.log` file with matching information.

It is also possible to randomly enumerate only a specific number of product molecules with the `-c` option:

```
../reaction_synthesizer -r amidecoupling.smirks -i building_blocks.smi
                        -o amide_enum -e amide_products_random.sdf -c 2
```

The output SD file `amide_products_random.sdf` now contains only 2 randomly enumerated product molecules.

3.5 Example 3: Transforming molecules for FastGrow databases or covalent docking

You can generate input molecules for FastGrow databases or covalent docking with the ReactionSynthesizer by formulating a transformation reaction, e.g. the product molecules must contain a linker (Figure 3). This linker then can serve as attachment point to a protein residue during covalent docking. The linker compound can also be added to a FastGrow database. Compounds that do not carry such a linker cannot be processed by FlexX/FastGrow/SeeSAR, they need to be pre-processed as described in this section prior to using them for covalent docking or growing.

On the command-line, navigate into the example folder within the ReactionSynthesizer directory. Open the file `vinylsulfone_transform.rxn` with a chemical drawing program (Figure 3). This reaction definition may serve as an example for the transformation of a compound with a vinylsulfone warhead into its protein bound state (Figure 3). The product contains a linker (R) which may serve as the attachment point for a nucleophilic residue during covalent docking. With this reaction definition, you can now prepare the compounds in `vinylsulfones.sdf` for covalent docking:

```
../reaction_synthesizer -r vinylsulfon_transform.rxn -i vinylsulfones.smi
                        -o vs_transformed -e vs_transformed.sdf
```

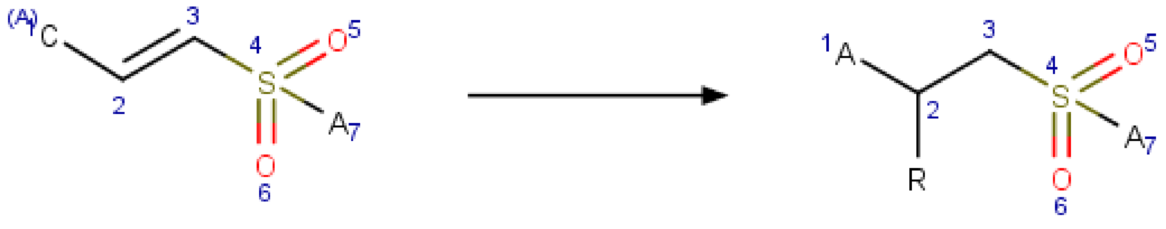


Figure 3: *Example of a reaction definition to transform compounds with a vinylsulfone warhead into their protein-bound state. The transformed molecule contains a linker (R).*

The output SD file `vs_transformed.sdf` contains the transformed compounds. You can directly use this file to perform covalent docking within SeeSAR or with FlexX (for more information visit `https://www.biosolveit.de/download/?product=flexx`).

As a second example, you can transform some building blocks with a carboxylic acid as functional group to fragments suitable to be incorporated into FastGrow databases:

```
../reaction_synthesizer -r acid_transform.rxn -i building_blocks.smi
                         -o fragments -e fastgrow_fragments.sdf
```

The file `fastgrow_fragments.sdf` contains the transformed acids with a linker (see Figure 4) and can serve as an input file for the FastGrowDBCreator (more information: `https://www.biosolveit.de/download/?product=fastgrow`).

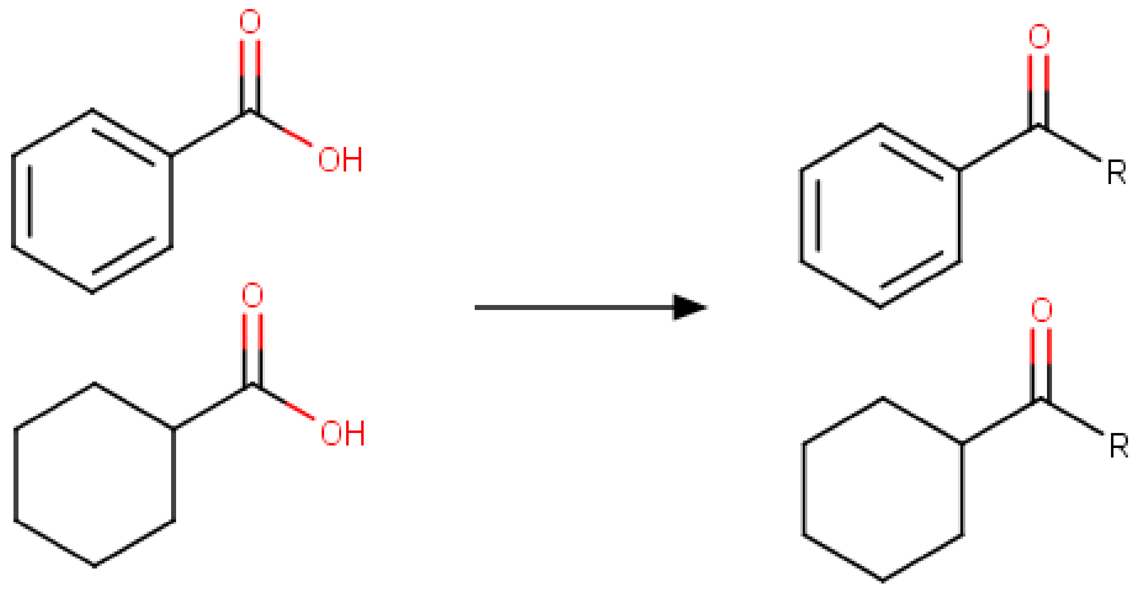


Figure 4: *Example of acids transformed to fragments with a linker. These fragments can be directly incorporated into FastGrow databases.*

3.6 Example 4: Removing protecting groups from product molecules

You can remove protecting groups from product molecules. As an example can serve the compound **1** in Figure 5 that has one Boc-protected amine and one free primary amine group. This compound can react with its free amine group with benzoic acid (**2** in Figure 5) to form an amide. Again, on the command line, navigate into the example folder and execute the following two calls:

```
../reaction_synthesizer -r amidecoupling.smirks -i boc_amine_and_acid.smi
                        -o boc_uncleaved -e boc_uncleaved.sdf
```

```
../reaction_synthesizer -r amidecoupling.smirks -i boc_amine_and_acid.smi
                        -o boc_cleaved -e boc_cleaved.sdf
                        -p ProtectingGroups.txt
```

The above calls will generate the new folders `boc_cleaved` and `boc_uncleaved` containing the files `boc_cleaved.sdf` and `boc_uncleaved.sdf`, respectively. In Figure 5 the resulting enumerated product molecules are shown for both cases. When using the `-p` argument with the file `ProtectingGroups.txt`, the Boc group is removed and is not present in the final product **4**.

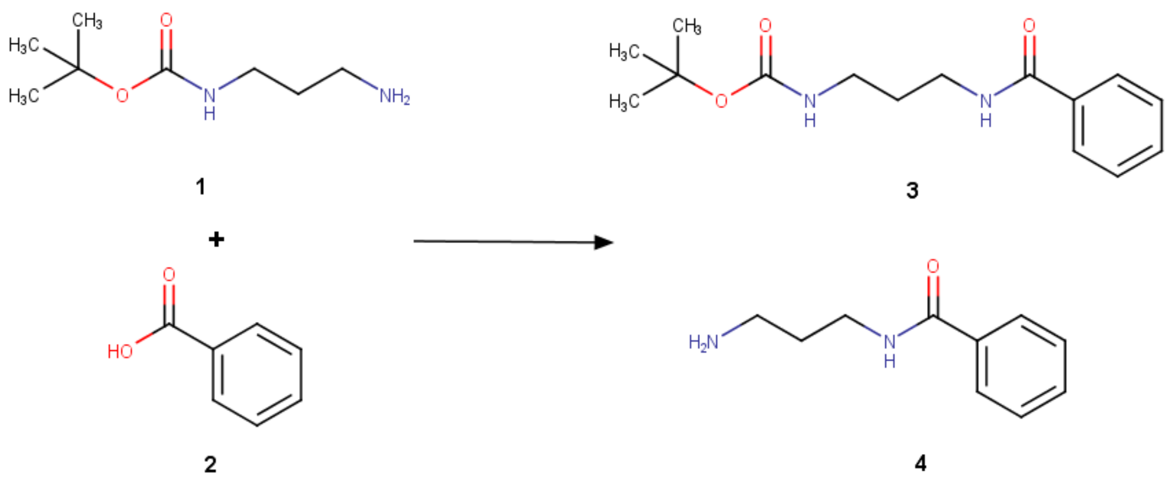


Figure 5: *Example product molecule obtained from an amide coupling reaction.* ***3****: without protecting group removal,* ***4****: with protecting group removal.*

3.7 Example 5: Debugging reaction definitions

The ReactionSynthesizer can be used to debug reaction definitions. Formalized reaction definitions quickly get complex, so it is often a good idea to check for potential errors or ambiguities before generating a space. The example folder in the ReactionSynthesizer directory contains an example reaction definition file named `amidecoupling_error.smirks` with a formal error. On the command-line, navigate into the example folder and execute the following:

```
../reaction_synthesizer -r amidecoupling_error.smirks -o . --check-reaction
```

The above call will create a file named `validation.log` in the example folder with information why the reaction definition is not valid (Missing mapping Label in product: 3). More on information on how to properly formalize reactions can be found in Section 6.

3.8 Example 6: Using reactions with functional group labels

A detailed description on how to define reactions with functional group labels can be found in Section 6. If you use reaction definitions that contain labels you have to specify the `FunctionalGroupLabel.txt` file with the `-s` option. The example folder in the ReactionSynthesizer directory contains an example reaction: `esterification_labels.rxn`. This is the same reaction definition that is generated as an example step-by-step in Section 6 (see Figure 11). On the command line, navigate into the example folder and execute the following:

```
../reaction_synthesizer -r esterification_labels.rxn -i building_blocks.smi
                        -o ester_space -e esters.sdf -g
                        -s FunctionalGroupLabel.txt
```

This call creates the folder `ester_space` that contains the space files and the enumerated products (`esters.sdf`).

4 The FragspaceMerger

The FragspaceMerger is used to merge multiple single reaction spaces generated with the ReactionSynthesizer (see Section 3 and the example in Section 3.3). The output is a master space containing multiple reaction definitions along with all corresponding fragments. The output comprises three files: the merged fragment_label file (`.flf`), the merged fragment_space file (`.fsf`) and the fragment file (`.smi`). Merging multiple single reaction space files results in unification of all involved fragments into one file and enhances the encoding of the fragment compatibilities. This leads to faster runtimes in FTrees and SpaceMACS searches. Additionally, if you zip the three merged files (e.g using 7zip) you have one chemical space file that is ready to be used with infiniSee, FTrees and SpaceMACS. You can also re-name the `.zip` file to `.space`.

4.1 General

An overview of all command line options is available by calling fragspace_merger with `--help`:

```
./fragspace_merger --help

Program options:
-i [ --input-fsf-paths ] arg  List of paths to the FSF files to merge.
                              Note: The .flf and fragment files specified in each FSF have to be
                              in the appropriate relative paths.
                              Note: Can't be used together with '--input-base-dir'.
-d [ --input-base-dir ] arg   Base directory to search for fragment spaces.
                              Each sub-directory (within the base) must contain exactly one
                              '.fsf' and one '.flf' file plus all (at least one) corresponding fragment files.
                              Note: Can't be used together with '--input-fsf-paths'.
-o [ --output-dir ] arg       Path for the output directory.
-f [ --out-file-name ] arg    Output base file name (without suffix) for the merged flf, fsf
                              and molecule files.

General options:
-h [ --help ]                 Print this help message
--license-info                Print license info
--version                     Print version info
-v [ --verbosity ] arg (=2)   Set verbosity level
                                   0 [silent]
                                   1 [error]
                                   2 [warning]
                                   3 [workflow]
                                   4 [steps]
```

The abbreviated, one-letter options are preceded with one dash `-` whereas the longer, named options are preceded with two dashes: `--`. If an option needs an argument (arg), you can include or omit the equals sign. Adapt the command line usage to your operating system and shell.

4.2 Program Options

–i [––input-fsf-paths] arg Specify the path to a `.fsf` file (a single space generated with the ReactionSynthesizer, see Section 3 and the example in Section 3.3). Please note: The corresponding `.flf` file and fragspace-fragment files (`.smi`) must be located in the same directory. You can specify multiple `.fsf` files to be merged at once by using the option multiple times in a row. This option cannot be used together with the `-d` option. See Section 4.3 for an example.

–d [––input-base-dir] arg Specify a base directory whose subdirectories should contain every single reaction space (see Section 3) in a separate folder. Please note: The single space folders must contain the `.fsf` and `.flf` file as well as all associated fragspace-fragment (`.smi`) files. Please note: This is an alternative way to merge multiple single spaces. This option cannot be used together with the `-i` option. See Section 4.3 for an example.

–o [––output-dir] arg Specify the path to an output directory to which the merged `.fsf`, `.flf` and fragment file (`.smi`) is written. Please note: If the directory does not already exist, it is created.

–f [––out-file-name] arg | Specify an output base file name (without suffix) to be used for the merged `.flf`, `.fsf` and fragment (`.smi`) file.

4.3 Example: Merging two single reaction spaces

You can find the two single spaces which are generated in Example 1 (see Section 3.3) in the example folder underneath the FragspaceMerger folder. You can now merge these two single spaces into one "parent" (master) space: On the command line, navigate into the example folder within the Fragspace-Merger directory and execute the following call:

```
../fragspace_merger -i amide_single_space/amidecoupling.fsf
                    -i suzuki_single_space/suzuki.fsf
                    -o merged_space -f parent_space
```

Alternatively, you can also use the `-d` option to execute the same task:

```
../fragspace_merger -d . -o merged_space -f parent_space
```

The above calls will create the output folder `merged_space` which contains the three files `parent_space.fsf`, `parent_space.flf` and `parent_space.smi`. These files can now be used for Feature Tree searches (FTrees CLI tool or Scaffold Hopper in infiniSee). You can also zip the three files and use this file with FTrees, SpaceMACS and infiniSee (optionally, re-name the file extension from `.zip` to `.space`).
Please note that SpaceLight needs a different space file, as will be discussed in Section 5.

5 The SpaceLightDBCreator

The SpaceLightDBCreator generates topological fragment spaces for the use with SpaceLight (`https://www.biosolveit.de/download/?product=spacelight`). Similar to the ReactionSynthesizer (see Section 3), the SpaceLightDBCreator takes building blocks and reaction definitions (`.smirks` or `.rxn`) as input. The output is a topological fragment space database (`.tfsdb`) file.

5.1 General

An overview of all command line options is available by calling spacelight_db_creator with `--help`:

```
Program options:
-r [ --reaction ] arg             Input reaction definition file. Supported file types are
                                  *.rxn and *.smirks.
-s [ --sma ] arg                  SMARTS functional group definition file. Needed if atom
                                  labels are used in RXN reactions.
-i [ --input ] arg                Input molecule files to be searched for reagents. Supported
                                  file types are *.smi, *.smiles, *.mol, *.mol2 and *.sdf.
--input-reagent-1 arg             Optional input molecule files explicitly used for reagent 1.
--input-reagent-2 arg             Optional input molecule files explicitly used for reagent 2.
--input-reagent-3 arg             Optional input molecule files explicitly used for reagent 3.
--input-reagent-4 arg             Optional input molecule files explicitly used for reagent 4.
-p [ --protecting-groups ] arg    Input file with protecting groups SMARTS to be removed after
                                  clipping.
-o [ --output-file ] arg          Topological fragment space file to be written.
-f [ --insert-full-molecules ] [=arg(=1)]
                                  Insert molecules supplied by --input directly into the created
                                  databse.
--include-macrocycles [=arg(=1)]  Single rings with more than 9 ring atoms can be part of fragments and
                                  molecules and can be formed in cyclization reactions.
--protect-space [=arg(=1)]        Protect the IP value within the output file.

General options:
-h [ --help ]                     Print this help message
--license-info                    Print license info
--thread-count arg                Maximum number of threads used for calculations. The default
                                  is to use all available cores.
--version                         Print version info
-v [ --verbosity ] arg (=2)       Set verbosity level
                                       0 [silent]
                                       1 [error]
                                       2 [warning]
                                       3 [workflow]
                                       4 [steps]
```

The abbreviated, one-letter options are preceded with one dash – whereas the longer, named options are preceded with two dashes: `--`. If an option needs an argument (arg), you can include or omit the equals sign. Adapt the command line usage to your operating system and shell.

5.2 Program Options

–r [––reaction] arg Specify the input reaction file. The file must contain the formalized reaction either in `.smirks` or `.rxn` format. More information on how to properly formulate a reaction definition can be found in Section 6.

–s [––sma] arg Specify a SMARTS functional group definition file. The file is only needed if atom labels are used in `.rxn` files. A pre-compiled file (named `FunctionalGroupLabel.txt`) with common functional groups can be found in the example folder inside the ReactionSynthesizer directory. You may extend this file depending on your needs. See Section 6 and 5.3 for more information.

–i [––input] arg | Specify an input file containing building blocks suitable to be used with the reaction given via the `-r` option. Supported file types are `.smi`, `.smiles`, `.mol`, `.mol2` and `.sdf`. The file will be scanned for suitable building blocks for all reagents specified in the reaction file. Only suitable building blocks will be read from the file, non-suitable ones are skipped. Please note: You can specify multiple input files at once by using the `-i` option multiple times in a row, e.g.

```
./reaction_synthesizer -i compounds_1.sdf -i building_blocks_2.smi
```

––input-reagent-1 arg Input molecule files explicitly used for reagent 1. If provided, this file should contain only building blocks which are suitable for the first reagent specified in the reaction given with `-r` option. Supported file types are `.smi`, `.smiles`, `.mol`, `.mol2` and `.sdf`.

––input-reagent-2 arg Input molecule files explicitly used for reagent 2. If provided, this file should contain only building blocks which are suitable for the second reagent specified in the reaction given with `-r` option. Supported file types are `.smi`, `.smiles`, `.mol`, `.mol2` and `.sdf`.

––input-reagent-3 arg Input molecule files explicitly used for reagent 3. If provided, this file should contain only building blocks which are suitable for the third reagent specified in the reaction given with `-r` option. Supported file types are `.smi`, `.smiles`, `.mol`, `.mol2` and `.sdf`.

––input-reagent-4 arg Input molecule files explicitly used for reagent 4. If provided, this file should contain only building blocks which are suitable for the fourth reagent specified in the reaction given with `-r` option. Supported file types are `.smi`, `.smiles`, `.mol`, `.mol2` and `.sdf`.

–p [––protecting-groups] arg Specify a file containing SMARTS definitions for protecting groups. The file should be a simple text file containing line-separated SMARTS definitions. If you provide such a file, the specified protecting groups present in the input building blocks will be clipped and removed from the generated fragments. An example file (named `ProtectingGroups.txt`) with common protecting groups can be found in the example folder within the SpaceLightDBCreator directory. See Section 5.3 for more information.

–o [––output-file] arg Specify the path and name of the output file. The output file is a topological fragment space database (`.tfsdb`) file. The suffix (`.tfsdb`) is required. Please note: You can write topology graphs from different reactions step-wise into a single database. See the example in Section 5.3.

–f [––insert-full-molecules] Insert the compounds in the input file (specified with the `-i` option) as "full molecules" directly into the database (no fragment and topology graph generation). This is especially useful if you want to add molecules to the database which cannot be constructed from the reactions of the space.

––include-macrocycles Allows the incorporation and formation of macrocyclic compounds in the database, e.g. compounds containing single rings with more than 9 ring atoms. Macrocycles are allowed to be part of a building block (given by `-i` or `--input-reagent` option) and can be formed as a product of a reaction (given by `-r` option), i.e. if this option is set, products containing single rings with more than 9 ring atoms can be formed via cyclization reactions.

--protect-space Protects the IP value in the output `.tfsdb` database file. Encrypts the structure and name of molecules in the database. Please note: Adding further topologies to a protected database (see the example in Section 5.3) always requires the `--protect-space` option be set.

5.3 Example: Creating a topological fragment space database

On the command-line, navigate into the example folder within the SpaceLightDBCreator directory. Here you can find two example reaction definition files (named `amidecoupling.smirks` and `suzuki.smirks`) and a file with suitable building blocks (`building_blocks.smi`). They cover two prominent examples of reactions used in Medicinal Chemistry: amide coupling and Suzuki reaction (see Figure 2). More information on how to properly formalize reactions for space generation can be found on our website.[2][3] First, you can use the amide reaction to create a new database:

```
../spacelight_db_creator -r amidecoupling.smirks -i building_blocks.smi
                         -o topo_space.tfsdb
```

This call will create a new database file named `topo_space.tfsdb` within the example folder.
Next, simply add the topology graph for the Suzuki reaction to the existing database:

```
 ../spacelight_db_creator -r suzuki.smirks -i building_blocks.smi
                          -o topo_space.tfsdb
```

You can repeat this process for multiple reactions and their associated building blocks. We recommend to automate this process using shell or python scripts. For paying customers, we have pre-compiled workflow scripts to facilitate space generation for multiple reactions. Please get in touch with us, and we can provide you with the scripts and assistance.

Similar to the example in Section 3.6, you can also cleave protecting groups from products:

```
../spacelight_db_creator -r amidecoupling.smirks -i boc_amine_and_acid.smi
                         -o boc_cleaved.tfsdb -p ProtectingGroups.txt
```

Similar to the example in Section 3.8, you can also use functional group labels in reaction definitions:

```
../spacelight_db_creator -r esterification_labels.rxn -i building_blocks.smi
                         -o ester_space.tfsdb -s FunctionalGroupLabel.txt
```

More information on defining reaction definitions with functional group labels can be found in Section 6.

Topological fragment space database files can be directly searched with SpaceLight (for more information on SpaceLight visit `https://www.biosolveit.de/download/?product=spacelight`). You can also put the `.tfsdb` file in a zip container (e.g. together with the merged "classical" fragment space files, see the example in Section 4.3) and use this single zip file comfortably with all modes of infiniSee and all command line search tools.

6 Prepare Reaction Definitions for Space Generation

The ReactionSynthesizer and SpaceLightDBCreator need precise reaction definitions as input to generate spaces with meaningful chemistry. Reaction definitions in RXN or SMIRKS format are accepted. The generation of Reaction-SMARTS/SMIRKS may be assisted by tools like the SMARTSEditor and SMARTS PLUS webserver (`https://smarts.plus/`) that are able to visualize SMARTS expressions.[1]

However, the most intuitive way to generate formalized reaction definitions is to sketch a reaction with a chemical drawing program and export it as `.rxn` file. In the following, MarvinSketch is used as an example, but other sketchers (ChemDraw etc.) can be used in a comparable way. The first step is to simply draw a reaction. It is often sufficient to draw only the functional groups that react with each other, as illustrated for the esterification A in Figure 6. The sketched molecules are interpreted as substructures, so esterification A will be more generic than esterification B. If you draw aromatic rings, make sure that the bond type for all ring bonds is set to aromatic.

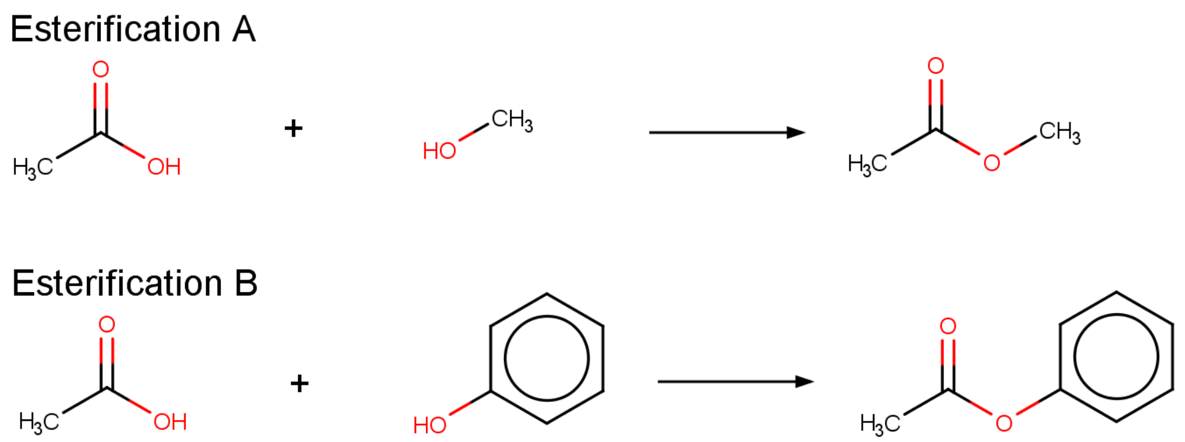


Figure 6: *Examples of drawing an esterfication reaction. Variant A is more generic than variant B.*

Let's continue with the more generic esterification A. As a second step, you must assign mapping numbers for all atoms that are present on both the product and the educt side. Therefore, right-click on an atom on the educt side, select "Map" from the context menu and assign a number. Now, right-click on the related atom in the product molecule and assign the same mapping number to it. It is important that all related atoms on educt and product side have a common unique mapping number (see Figure 7)! In this example, the OH-atom in the educt acid must not get a mapping number because it is not present in the product molecule. It will be removed during the reaction (leaving group).

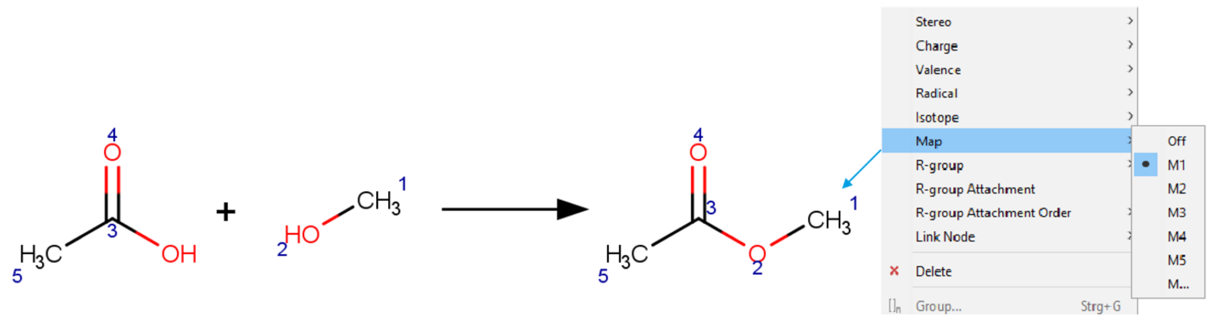


Figure 7: *Adding mapping numbers for every related atom pair on educt and product side.*

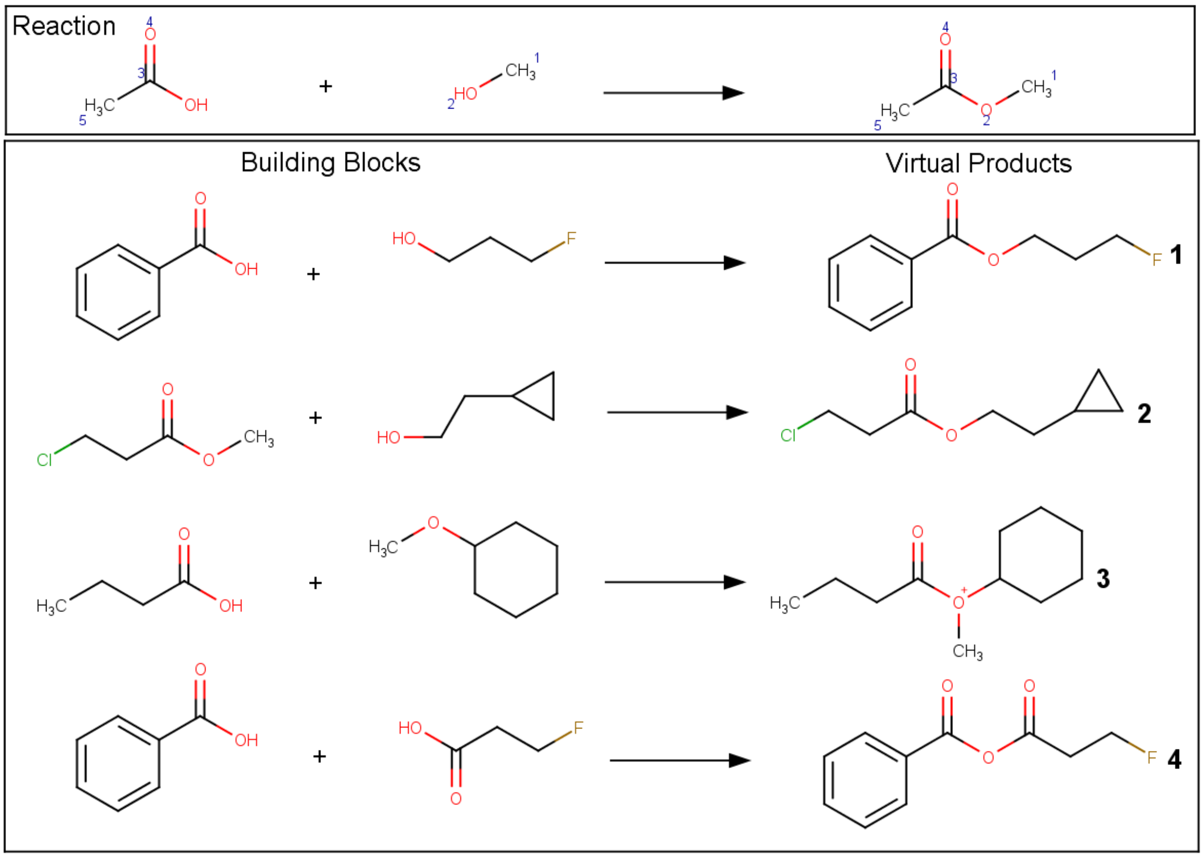


Figure 8: *Example of products and artifacts generated with an imprecise reaction definition.*

At this stage, the reaction is in principle properly defined to be used with ReactionSynthesizer and SpaceLightDBCreator. However, the reaction definition is not as precise as it should be to avoid unwanted products and artifacts. This should be illustrated with the building blocks in Figure 8.

Product **1** is a desired product of the esterification reaction. Product **3** is chemically invalid and products **2** and **4** were not intended to be formed with this reaction definition. It is therefore crucial to define the chemical environment of the functional group atoms as precise as possible. This can be achieved by using recursive SMARTS definitions. It is important that the SMARTS definitions only match exactly one atom. This can be best illustrated for a SMARTS that precisely defines the oxygen atom of the educt alcohol: `[OD1;$(O-[#6;!$(C=[O,N,S])])]`. This SMARTS matches an oxygen atom that has only one heavy atom as neighbor (`OD1`) and this neighbor atom must be a carbon atom that is not allowed to have a double bond to oxygen, sulfur or nitrogen (`$(O-[#6;!$(C=[O,N,S])])`).

You can assign the SMARTS to the oxygen atom of the educt alcohol as shown and described in Figure 9. This eliminates products **3** and **4**. To also eliminate product **2**, the carboxyl group of the educt acid has to be defined properly. One possibility is to precisely define the environment of the central carbon (carbonyl carbon) of the carboxyl group: `[C$(C(=[OD1])([OD1])[#6,#1])]`. This SMARTS pattern only matches the carbonyl carbon of carboxylic acids, but not of esters.

Now we have generated a very precise reaction definition for an esterification between a carboxylic acid and an alcohol. As a final step, just export the reaction as `.rxn` file (make sure to select MDL Rxnfile, do not use MDL Extended Rxnfile) and specify this file with the `-r` option of ReactionSynthesizer and SpaceLightDBCreator (see examples in Sections 3 and 5).

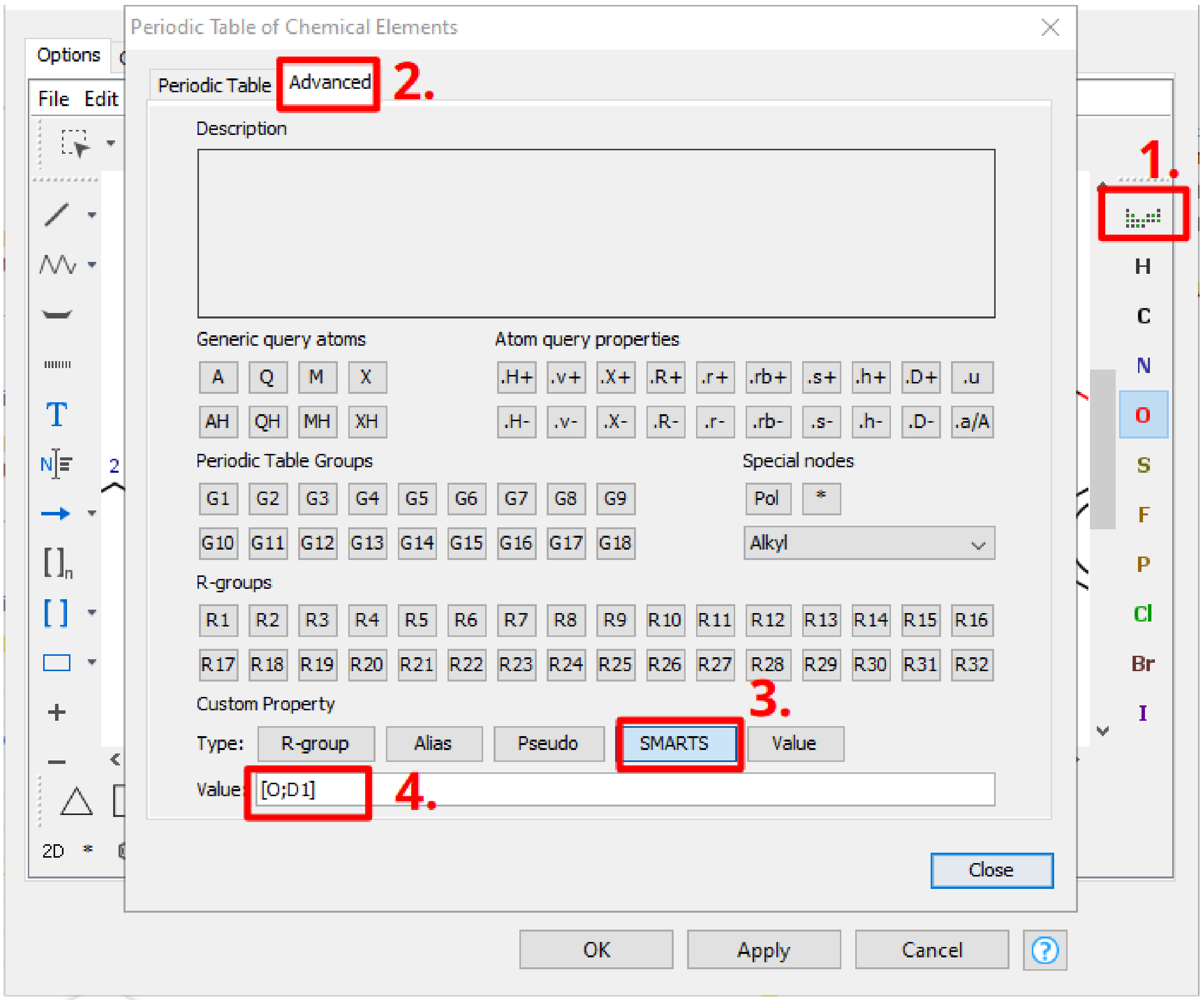
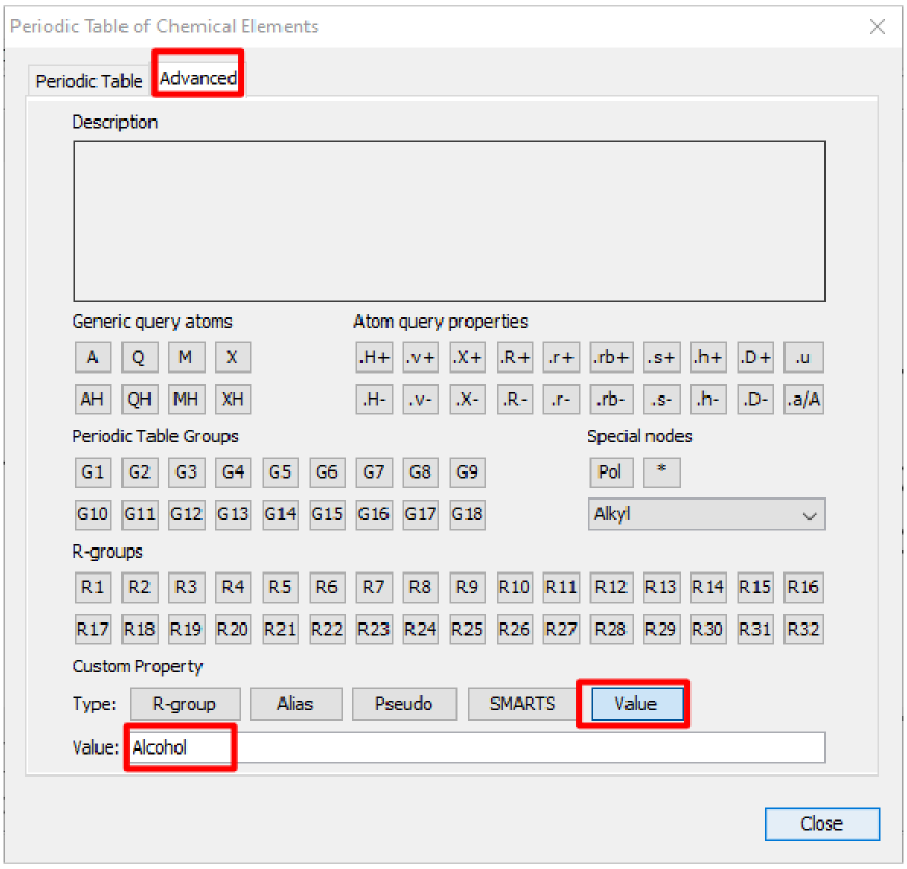


Figure 9: *Adding SMARTS definitions to an atom to define its chemical environment with MarvinSketch. 1. Click on the "periodic table" | 2. Got to the "Advanced" tab. | 3. Choose "SMARTS" as custom property type. | 4. Enter the SMARTS definition. | 5. (not shown) Close the periodic table window and assign the SMARTS by clicking on the desired atom (in case of our example the oxygen atom of the educt alcohol).*

Alternatively to use SMARTS to properly define functional groups in reaction definitions you can also use "labels", which are simply pre-defined SMARTS-patterns that make the drawing more easily human-readable. You can find pre-defined SMARTS patterns and their associated human-readable labels in the `FunctionalGroupLabel.txt` file in the example folder inside the ReactionSynthesizer or SpaceLightDB-Creator repository folder. Instead of directly assigning a SMARTS to an atom of the functional group simply assign the corresponding label specified in the file. This can be done with Marvin Sketch in comparable manner (see Figure 10 and compare to Figure 9). The atom of the functional group to which the label must be added is described in the comment column of the file. For our example, you have to assign the label "Alcohol" to the oxygen of the alcohol group and the label "CarboxylicAcid" to the central carbon (carbonyl carbon) of the carboxyl group (see Figure 11).


Periodic Table of Chemical Elements
Periodic Table
Advanced
Description
Generic query atoms
A Q M X
AH QH MH XH
Atom query properties
.H+ .v+ .X+ .R+ .r+ .rb+ .s+ .h+ .D+ .u
.H- .v- .X- .R- .r- .rb- .s- .h- .D- .a/A
Periodic Table Groups
G1 G2 G3 G4 G5 G6 G7 G8 G9
G10 G11 G12 G13 G14 G15 G16 G17 G18
Special nodes
Pol *
Alkyl
R-groups
R1 R2 R3 R4 R5 R6 R7 R8 R9 R10 R11 R12 R13 R14 R15 R16
R17 R18 R19 R20 R21 R22 R23 R24 R25 R26 R27 R28 R29 R30 R31 R32
Custom Property
R-group Alias Pseudo SMARTS
Close

Type: Value

Value: Alcohol

Figure 10: *Adding labels to an atom with MarvinSketch. 1. Click on the "periodic table" | 2. Got to the "Advanced" tab. | 3. Choose "Value" as custom property type. | 4. Enter the label. | 5. (not shown) Close the periodic table window and assign the label by clicking on the desired atom (in case of our example the oxygen atom of the educt alcohol).*

If you use reaction definitions with labels always make sure to specify the `FunctionalGroupLabel.txt` file via the `-s / --sma` option (see Section 3.8 and 5.3 for examples). You may even extend the label file depending on your needs.

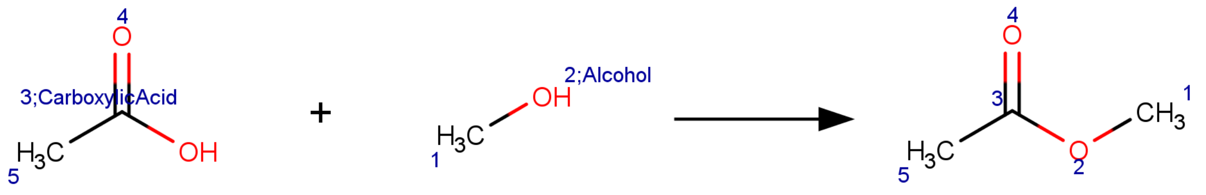


Figure 11: *Esterification reaction with functional group labels.*

7 General Options

This section describes the general options which are identical for all CoLibri tools.

–h [––help] Displays the command line help with short descriptions for every argument option.

––license-info Shows detailed information about the license setup you currently use. If you have any problems with your license, send an email to `mailto:support@biosolveit.com` and include this information.

––thread-count arg Specifies the maximum number of threads used by the tool. By default, all available logical cores of your computer are used. You may want to reduce the number of threads if you want to run other computations on your computer at the same time, or if you share the compute resource.

––version Displays information on the version of the respective tool on the command line. In quoting the tool, please mention this version number.

–v [––verbosity] arg Sets the verbosity level, e.g., the level of console output, with an integer argument. The default value is 2. The following options are available:

- 0 Silent. No messages will be displayed in the console during the run. Errors will be ignored whenever possible.
- 1 Error. Only error messages will be displayed.
- 2 Warning. The default setting, warnings and error messages will be displayed.
- 3 Workflow. In addition to errors and warnings, information on the different steps are displayed on the command line.
- 4 Steps. In addition to the 'Workflow' option, the progress of each step is displayed in detail.

8 Further Reading, References

Additional information about CoLibri is available at `https://www.biosolveit.de/products/#CoLibri`.

Complementary tools, e.g. the command line search tools FTrees, SpaceMACS and SpaceLight as well as the Chemical Space Navigation Platform infiniSee, can be obtained from the BioSolveIT website (`https://biosolveit.com`).

References

[1] `https://www.biosolveit.de/academic-drug-discovery/`.

[2] `https://www.biosolveit.de/wp-content/uploads/2021/06/KNIME_implementation-of-reactions.pdf`.

[3] `https://www.biosolveit.de/wp-content/uploads/2022/02/BeginnersGuide_SMARTSeditor.pdf`.

We wish you great success and much joy with CoLibri!