Electronic Supplementary Information

General Data Management Workflow to Process Tabular Data in Automated and High-throughput Heterogeneous Catalysis Research

Erwin Lam[a,c,*], Tanguy Maury[a,c], Sebastian Preiss[a,c], Yuhui Hou[a], Hannes Frey[a], Caterina Barillari[b], Paco Laveille[a,*]

[a]ETH Zurich, Swiss Cat+ East, Vladimir-Prelog-Weg 1-5, 8093 Zurich, Switzerland

[b]ETH Zurich, ID SIS, Klingelbergstrasse 48, 4056 Basel, Switzerland

[c]These authors contributed equally

*Corresponding author. E-mail: elam@ethz.ch; plaveille@ethz.ch

Supplementary Note 1: OpenBIS "Dropbox Function"

Automatic data upload to openBIS can be performed with its integrated "Dropbox function". Upon implementation, the openBIS network drive "eln-lims-dropbox" of the corresponding openBIS platform can be mounted. Within this network drive, the folder name, following a naming convention, provides information about location, data type and data name. With this information, files are automatically uploaded to openBIS (Figure S1).

Therefore, a Python script can be implemented, tailored to the output of individual instruments to automate data upload, where upon creation of the experiment's final output file, all relevant data to the experiment is directly uploaded to openBIS. Currently at ETHZ SwissCAT+, two Python scripts have been implemented to perform data upload on instruments from Chemspeed (Swing XL) and Avantium (XR/XD fixed-bed reactors) (See Supplementary Note 2).

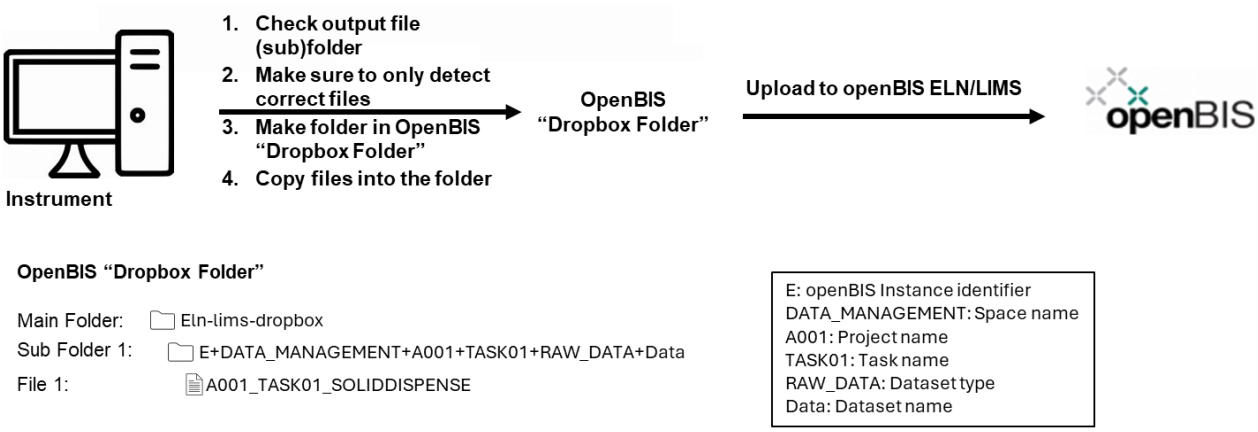

Figure S1. Workflow to automated data upload of raw output files to openBIS.

Supplementary Note 2: Automatic file upload of Avantium XR/XD fixed-bed reactors & Chemspeed Swing XL

A watcher script (See Avantium_Watcher.py) has been implemented to automatically upload Avantium fixed-bed testing related data. More precisely, the output Excel files containing set and process values of the instrument (gas flow rates, temperature, pressure, valve position etc.), and the GC concentration data is uploaded as “RAW_DATA” to openBIS. Since this data file is generally manually post-processed, the instruction to use the script is to save the final processed output file in a destination folder locally. The watcher script detects that a new file has been created and checks if the filename fulfills certain conditions. At ETHZ SwissCAT+, experiments are grouped into projects (labelled AXXX), experiments (labelled TASKXX) and runs (labelled RUNXX). Therefore, the filename should contain the following strings in the order: AXXX_TASKXX_RUNXX (X = integers). This information is required for the script to know where to store the data in openBIS which follows a similar project/experiment structure (See Supplementary Note 1). In addition, all the raw data from the GC analysis of an experiment will also be uploaded as “GC_DATA”. The folder containing all the GC related data that should have the same naming pattern (AXXX_TASKXXX_RUNXX), will be zipped and uploaded to openBIS.

The watcher script (Chemspeed_Watcher.py) has been implemented to upload automatically the logfile and output CSV files after a run. The script monitors the folder where the logfiles are created during a Chemspeed Swing XL experiment. Two types of logfiles are created from the log file folder either from a simulation run or an actual experimental run. Logfiles of simulation runs always have the tag “(simulated)” and those files are ignored by the Python script. The logfile of an actual experiment is continuously updated while the Chemspeed Executor software is running. The Python script therefore monitors the logfile for updates and uploads the logfile as “LOG_DATA” through the “openBIS Dropbox” function once the logfile stops updating. At the same time, the Python script monitors the folder with the CSV output file of the Chemspeed run that contains information about liquid dispense, solid dispense, heating temperature, shaking rate and reaction time that gets uploaded as “RAW_DATA”. Since the CSV output file gets updated once the experimental run is finished, the Python script uploads the file only when the modification time of the CSV output file is newer than the modification time of the logfile. Within ETHZ SwissCAT+, to ensure the right files are copied, further controls have been implemented following the project/experiment structure (See Supplementary Note 1) at ETHZ SwissCAT+. Logfiles should have the naming convention: AXXX_TASKXX_RUNXX. Output CSV files should have the naming convention: AXXX_TASKXXX_RUNXX_OUTPUT.csv and stored in specific folder structures (e.g. AXXX/TASKXXX/AXXX_TASKXXX_RUNXX_OUTPUT.csv).

Supplementary Note 3: Jupyter Notebook for the data processing example.

With the data files uploaded to openBIS and the configuration file, the data processing workflow can be performed on a Jupyter Notebook requiring few lines of inputs. The directory needs to be changed to where the data should be stored locally, and all the python libraries are loaded.

```
#Change directory to the source code directory
%cd "DIRECTORY_PATH" #Input directory path


#Load all the required library for openBIS data download, reading the data, merging #the data and processing the data
from src.data.openBIS_queryV3 import openbis_query
from src.data.Read_Data import read_data
from src.data.Merge import merge
from src.data.Process_FixedBed import process_fixedbed
```

The data management tasks require the location of the configuration file and one line for each individual step (*i.* download data from openBIS, *ii.* read the data onto a dictionary, *iii.* merge the data, *iv.* process the data). The final merged and processed data will be stored in the local folder and uploaded to openBIS (if save_to_openbis=TRUE).

```
#Path of the instruction/configuration file
config_file_name = "CONFIG_FILE_PATH "
#Function to run the data management workflow
obq = openbis_query(data_source="OPENBUS_URL/SPACE/PROJECT/",config_file_name= config_file_name)
obq = read_data(data_source="DATAFOLDER",config_file_name= config_file_name, tasks_to_process = ["TASKS"])
obm = merge(data_sets=obq.data,config_file_name= config_file_name, save_to_openbis=True)
obp = process_fixedbed(config_file_name= config_file_name,data_sets= obm.data_processed)
```

```
openbis_query(data_source,config_file_name)
         data_source: str, path of the data on openBIS
         config_file_name: str, path of the instruction/configuration file
read_data(data_source,config_file_name, tasks_to_process)
         data_source: str, path of the local data folder
         config_file_name: str, path of the instruction/configuration file
         tasks_to_process: list, list of tasks to process
merge(data_sets,config_file_name, save_to_openbis)
         data_sets: dict, dictionary of data
         config_file_name: str, path of the instruction/configuration file
         save_to_openbis: bool, if True: upload merged data to openBIS
process_fixedbed(data_sets, config_file_name)
         data_sets: dict, dictionary of data
         config_file_name: str, path of the instruction/configuration file
```

Supplementary Note 4: Multiple files

During the read data process (Read.py), multiple files that are stored in the dictionary with the same FILE_ID are first concatenated during the merging process (Mergy.py) before merging with other FILE_ID's data (Figure S2).

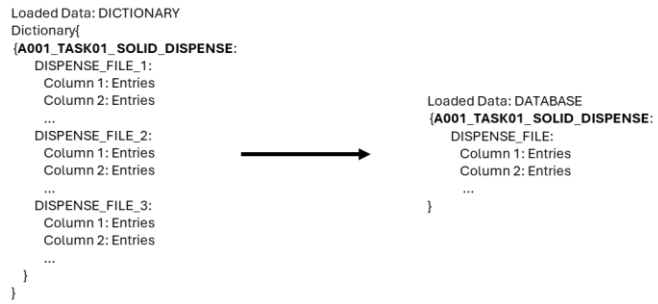

Figure S2. Concatenation of multiple files within a dictionary's FILE_ID prior to merging.

Supplementary Note 5: Identical file structure of different Tasks:

For different tasks with identical file and merging procedure, the configuration files with information about the files do not need to be repeated. For example, for the first TASKS (TASK01), the information can be collected with “&A001_STANDARD_1” and the instruction can be reused for subsequent tasks with “*A001_STANDARD_1”

```
TASKS:
  TASK01: &A001_STANDARD_1 #Collect file information into “A001_STANDARD_1”


    METAL_DISPENSE:
      NAME: METAL_DISPENSE
      FILE_ID: !!str CombVialPrep
      DATA_SHEET: !!int 0
      CSV_SEPARATOR: !!str ","
      TARGET_ID: !!str Combinational Vial Number

    SOLID_DISPENSE:
      NAME: SOLID_DISPENSE
      FILE_ID: !!str SolidDispense
      DATA_SHEET: !!int 0
      CSV_SEPARATOR: !!str ","
      SOURCE_ID: !!str Impregnation Vial Number

    IMPREGNATION:
      NAME: IMPREGNATION
      FILE_ID: !!str Impregnation
      DATA_SHEET: !!int 0
      CSV_SEPARATOR: !!str ","
      SOURCE_ID: !!str Combinational Vial Number
      TARGET_ID: !!str Impregnation Vial Number

  TASK02: *A001_STANDARD_1 #Use same file information as TASK01 “A001_STANDARD_1”
  TASK03: *A001_STANDARD_1
  TASK04: *A001_STANDARD_1
  TASK05: *A001_STANDARD_1
```

Supplementary Note 6: MERGING_ORDER section and sequence of combining files

The example below illustrates how the configuration file should be set up to merge four files within a set sequence from an experimental workflow. The four files have the “NAME” METAL_DISPENSE, SOLID_DISPENSE, IMPREGNATION and IMPREGNATION_TO_BARCODE. The MERGING_ORDER section indicates on how the map should be structured to perform the following merging sequence:

METAL_DISPENSE and SOLID_DISPENSE files are first merged using the column names indicated in the TARGET_ID and SOURCE_ID. The IMPREGNATION file is then merged with the combined file of METAL_DISPENSE and SOLID_DISPENSE. Lastly, the IMPREGNATION_TO_BARCODE file is merged into the combined file to have all four individual files merged into one file.

```
TASKS:
  TASK01:

    METAL_DISPENSE:
      NAME: METAL_DISPENSE
      FILE_ID: !!str CombVialPrep
      DATA_SHEET: !!int 0
      CSV_SEPARATOR: !!str ","
      TARGET_ID: !!str Combinational Vial Number

    SOLID_DISPENSE:
      NAME: SOLID_DISPENSE
      FILE_ID: !!str SolidDispense
      DATA_SHEET: !!int 0
      CSV_SEPARATOR: !!str ","
      SOURCE_ID: !!str Impregnation Vial Number

    IMPREGNATION:
      NAME: IMPREGNATION
      FILE_ID: !!str Impregnation
      DATA_SHEET: !!int 0
      CSV_SEPARATOR: !!str ","
      SOURCE_ID: !!str Combinational Vial Number
      TARGET_ID: !!str Impregnation Vial Number

    IMPREGNATION_TO_BARCODE:
      NAME: IMPREGNATION_TO_BARCODE
      FILE_ID: !!str ImpregnationBarcode
      DATA_SHEET: !!int 0
      CSV_SEPARATOR: !!str ","
      SOURCE_ID: !!str Impregnation Vial Number
```

```
MERGING_ORDER:
  TASK01: !!map {
    left: {
      left: {
          left:     [METAL_DISPENSE, FILE_ID],
          left_id:  [METAL_DISPENSE, TARGET_ID],
          right:    [SOLID_DISPENSE, FILE_ID],
          right_id: [SOLID_DISPENSE, SOURCE_ID]
        },
        left_id:  [SOLID_DISPENSE, SOURCE_ID],
        right:    [IMPREGNATION, FILE_ID],
        right_id: [IMPREGNATION, TARGET_ID]
      },
    left_id:  [IMPREGNATION, TARGET_ID],
    right:    [IMPREGNATION_TO_BARCODE, FILE_ID],
    right_id: [IMPREGNATION_TO_BARCODE, SOURCE_ID]
  }
```

Supplementary Note 7: Merging on multiple columns and filename

In the example below, the REACTOR DATA section will consist of 3 individual files from fixed-bed testing ran on 3 different Avantium fixed-bed unit (XR and XD). From the output file content, it would not be straightforward to know which files corresponds to which Avantium fixed-bed unit. To have an interpretable and logical process to merge such data files, further features were implemented into the Mergy.py function to take strings of the output filename to allow data merging.

For example, in the following case three files are generated:

A001_TASKXX_CATALYSIS_XR_RUN01

A001_TASKXX_CATALYSIS_XDB_RUN01

A001_TASKXX_CATALYSIS_XDC_RUN01

The relevant data are stored in the data sheet with the name “Run Data” (DATA_SHEET: !!str Run Data). The instruction file takes the fourth entry of the filename separated by “_” and adds this information as a new column entry. The fourth entry would be XR, XDB or XDC and corresponds to the name of the fixed-bed units. The name of the new column entry (UNIT) is provided in the configuration file.

```
Project: !!str A001

TASKS:
  TASK01: &A001_STANDARD_1
    REACTOR_LOADING:
      NAME: REACTOR_LOADING
      FILE_ID: !!str ReactorLoading
      DATA_SHEET: !!int 0
      CSV_SEPARATOR: !!str ","
      SOURCE_ID: !!str Catalysis Vial Number Barcode
      TARGET_ID: !!str Reactor
      TARGET_UNIT: !!str Unit
      RUN_NUMBER: !!str Run

    REACTOR_DATA:
      NAME: REACTOR_DATA
      FILE_ID: !!str CATALYSIS
      CSV_SEPARATOR: !!str ","
      DATA_SHEET: !!str Run Data
      SOURCE_ID: !!str Reactor # generated via special case
      COLUMN_TO_ADD:
        UNIT:
          ONLY_PARTS: !!int 0
          LENGTH_IGNORE: None
          NAME: !!str Unit
          IS_INT: !!bool False
          POSITION: 4
```

For the merging process, multiple column names are provided to merge data. In the example below, the merging occurs on the REACTOR_LOADING file from the TARGET_ID and TARGET_UNIT that corresponds to column names of Reactor and Unit. This is matched with the column entries of the REACTOR_DATA file column names provided by the SOURCE_ID (Reactor) and the newly generated column ([COLUMN_TO_ADD, UNIT, NAME]) that corresponds to the string in the filename (XR, XDB or XDC)

```
MERGING_ORDER:
  TASK01: &A001_MERGE_STANDARD_1 !!map {
        left: {
        left:     [REACTOR_LOADING, FILE_ID],
        left_id:  [REACTOR_LOADING, [TARGET_ID, TARGET_UNIT]],
        right:    [REACTOR_DATA, FILE_ID],
        right_id: [REACTOR_DATA, [SOURCE_ID, [COLUMN_TO_ADD, UNIT, NAME]]],
  }
```

Supplementary Note 8: Running the demonstration data management workflow.

A demonstration Jupyter notebook with the data already downloaded locally is included. It allows to demonstrate the data processing tasks of project A001 without the need of an openBIS platform.

- The Jupyter notebook first changes to the correct directory and loads the required libraries. (Note: the directory change is set to go two directories backwards with respect to the Jupyter notebook. Therefore, this command would continue to move two directories backwards if executed multiple times without restarting the kernel).

```
import os
os.chdir(os.path.join(os.getcwd(), "..", ".."))

from src.data.Read_Data import read_data
from src.data.Merge import merge
from src.data.Process_FixedBed import process_fixedbed
```

- The configuration file path ./src/config/config_a001.yml is assigned to the variable config_file_name.

```
config_file_name = "./src/config/config_A001.yml"
```

- The data are loaded into the dictionary by running the read_data function which requires:
  - Path of the data: data_source = "./data/A001"
  - Path of the configuration file: config_file_name = config_file_name
  - TASKS to process: tasks_to_process = ["TASK01", "TASK02", "TASK03", "TASK04", "TASK05", "TASK06"].
- With this information, the data are loaded into the dictionary:

```
obr = read_data(data_source="./data/A001",config_file_name= config_file_name,
                tasks_to_process =
["TASK01","TASK02","TASK03","TASK04","TASK05","TASK06" ])
```

- Each of the tasks are loaded in subdictionaries: TASK0X_dict (X = 1, 2, 3, 4, 5, 6)

```
for key in obr.data.keys():
    print(key)
OUTPUT:
TASK02_dict
TASK01_dict
TASK05_dict
TASK06_dict
TASK03_dict
TASK04_dict
```

- All the data are located in obr.data and individual loaded files can be displayed by indicating the path of the files:

```
obr.data["TASK01_dict"]["METAL_DISPENSE"]["A001_TASK01_CombVialPrep.csv"]
```

- TASK01_dict being the subdictionary, METAL_DISPENSE the FILE_ID from the configuration file and A001_TASK01_CombVialPrep.csv the filename.
- Using the data from obr (obr.data) and the configuration file, data are merged.
    - Data set dictionary: data_sets = obr.data
    - Path of the configuration file: config_file_name = config_file_name
    - Upload to openBIS: save_to_openbis = FALSE (no openBIS upload in this example)

```
obm = merge(data_sets= obr.data,config_file_name= config_file_name,
save_to_openbis=False)
```

- Each of the merged files are stored in subdictionaries: TASK0X (X = 1, 2, 3, 4, 5, 6)

```
for key in obm.data.keys():
    print(key)
OUTPUT:
TASK03
TASK02
TASK04
TASK05
TASK06
TASK01
```

- The merged dataframes for each TASK can be accessed by:

```
obm.data["TASK01"]
```

- Using the data from obm (obm.data) and the configuration file, data are processed.
    - Data set dictionary: data_sets= obm.data
    - Path of the configuration file: config_file_name = config_file_name

```
obp = process_fixedbed(config_file_name= config_file_name,data_sets=
obm.data_processed)
```

- Each of the processed files are stored in subdictionaries: TASK0X (X = 1, 2, 3, 4, 5, 6)

```
for key in obp.full_data.keys():
    print(key)
OUTPUT:
TASK03
TASK02
TASK04
TASK05
TASK06
TASK01
```

- The dataframes in the subdictionaries are concatenated into one full dataframe and can be accessed by:

```
obp.full_dataframe
```

- In the presented case study, four data points per condition were collected which are averaged. The dataframe of the mean values can be accessed by:

```
obp.full_dataframe_mean
```

- Exported CSV files can be found in the ./A001 folder consisting of:
    - TASK0X/TASK0X_Data.csv: Merged data for TASK0X
    - TASK0X/TASK0X_rename.csv: Merged and renamed data for TASK0X (See Supplementary Note 10)
    - TASK0X/TASK0X_processed.csv: Processed data for TASK0X
    - A001_Full_Data.csv: Data from all TASKS collected in one file
    - A001_Full_Data_Mean.csv: Data from all TASKS with the datapoints of four measurements averaged

Supplementary Note 9: Instruction for catalyst performance calculation.

A custom script is implemented to allow calculating reaction metrics for Avantium fixed-bed reactors. The script is tailored to process carbon conversion reactions (e.g. CO_2 hydrogenation, propane dehydrogenation). In the REACTOR_UNIT section, information about the different units is provided. The first subsection indicates the name of the unit (e.g. FLOWRENCE_T2101B). Required information are the identifier of the unit (e.g. XDB) and its setup under SETUP. The SETUP section contains information about the inlet values such as temperature (TEMPERATURE), pressure (PRESSURE) and gas flows (GAS_FLOW). For each section the NAME is given which corresponds to the description of the section, the unit, and the column name of the set point (SET_POINT) and process value (PROCESS_VALUE). Within the GAS_FLOW section, each subsection corresponds to individual mass flow controllers (e.g. FIC_110), including the name of the gas (NAME), unit (UNIT), gas composition, set point and process value. Having the gas composition allows to obtain the overall flowrate of a specific gas e.g. N_2 is present in FIC_130 and FIC_140. These provided information are unique for individual fixed-bed reactors and should generally remain the same within the same unit.

```
REACTOR_UNIT:
#-------------------------------------------------------------------------------#
  FLOWRENCE_T2101B:
    NAME: !!str XDB
    FILE_ID: !!str T2101B
    SETUP: &XD_SETUP
      INLET:
        TEMPERATURE:
          NAME: !!str Reactor Temperature
          UNITS: !!str "[C]"
          SET_POINT: !!str Reactor_Temperature_SP
          PROCESS_VALUE: !!str Reactor_Temperature_PV

        PRESSURE:
          NAME: !!str Reactor Pressure
          UNITS: !!str "[Barg]"
          SET_POINT: ReactorPressure_SP
          PROCESS_VALUE: !!str Reactor_Inlet_Pressure

        GAS_FLOW:
          FIC_110:
            NAME: !!str H2
            UNITS: !!str "[mL Min-1]"
            COMPOSITION: !!map {"H2": 1}
            SET_POINT: !!str FIC_110_SP
            PROCESS_VALUE: !!str FIC_110_PV
            CONTROL_VALVE: !!str KCV_115_SP

          FIC_130:
            NAME: !!str N2
```

```
            UNITS: !!str "[mL Min-1]"
            COMPOSITION: !!map {"N2": 1}
            SET_POINT: !!str FIC_130_SP
            PROCESS_VALUE: !!str FIC_130_PV

          FIC_140:
            NAME: !!str 40_CO2_in_N2
            UNITS: !!str "[mL Min-1]"
            COMPOSITION: !!map {"CO2": 0.4, "N2": 0.6}
            SET_POINT: !!str FIC_140_SP
            PROCESS_VALUE: !!str FIC_140_PV
```

The configuration file contains a “Reaction” section where reactants (REACTANTS) and products (PRODUCTS) are provided. The script will use the NAME provided in the configuration file and match it with the gas chromatogram concentration columns (e.g. FID_F_**CH4**_conc) in the output file and use this column entry to perform calculation of reactant conversion, product selectivity and formation rate. The number of carbon atoms are provided in the COMPOSITION section in order to normalize the gas concentration by carbon number for conversion and selectivity calculation.

```
#-------------------------------------------------------------------------------#
#                                  Reaction                                     #
#-------------------------------------------------------------------------------#

REACTION:
  REACTANTS:
    REACTANT_A:
      NAME: !!str "CO2"
      COMPOSITION: !!map {"C": 1, "H": 0}

  PRODUCTS:
    PRODUCT_A:
      NAME: !!str "MeOH"
      COMPOSITION: !!map {"C": 1, "H": 4}

    PRODUCT_B:
      NAME: !!str "CO"
      COMPOSITION: !!map {"C": 1, "O": 1}

    PRODUCT_C:
      NAME: !!str "CH4"
      COMPOSITION: !!map {"C": 1, "H": 4}
```

Supplementary Note 10: Renaming of columns on multiple Avantium fixed-bed reactors.

Since the Avantium fixed-bed reactors may have identical column names with different values, a renaming process with the instructions located in the configuration file was implemented. For example, the mass flow controller assigned to FIC_110 may correspond to H_2 in one unit and to N_2 in another unit. The instructions to rename the files are provided in the configuration file.

In the REACTOR_UNIT section, each individual fixed-bed reactor is described. In the example below, an Avantium Flowrence XD unit is presented and illustrates the renaming of temperature, pressure and flowrates. For the TEMPERATURE section, the column name of the set and process value of the temperature are given and how it should be renamed. The script will take the SET_POINT (e.g. Reactor_Temperature_SP) and PROCESS_VALUE (e.g. Reactor_Temperature_PV) column names and rename them with the NAME + SET_POINT_LABEL + UNIT (e.g. “Reactor Temperature_SP_[C]” for the set point). In case of the flowrates, the set point value FIC_110_SP will be converted to H2_SP_[mL Min-1].

```
REACTOR_UNIT:
#-------------------------------------------------------------------------------#
  FLOWRENCE_T2101B:
    NAME: !!str XDB
    FILE_ID: !!str T2101B
    SETUP: &XD_SETUP
      INLET:
        TEMPERATURE:
          NAME: !!str Reactor Temperature
          UNITS: !!str "[C]"
          SET_POINT: !!str Reactor_Temperature_SP
          PROCESS_VALUE: !!str Reactor_Temperature_PV

        PRESSURE:
          NAME: !!str Reactor Pressure
          UNITS: !!str "[Barg]"
          SET_POINT: ReactorPressure_SP
          PROCESS_VALUE: !!str Reactor_Inlet_Pressure

        GAS_FLOW:
          FIC_110:
            NAME: !!str H2
            UNITS: !!str "[mL Min-1]"
            COMPOSITION: !!map {"H2": 1}
            SET_POINT: !!str FIC_110_SP
            PROCESS_VALUE: !!str FIC_110_PV
            CONTROL_VALVE: !!str KCV_115_SP

DATA_PROCESSING:
  PROCESS_VALUE_LABEL: !!str "_PV"
  SET_POINT_LABEL: !!str "_SP"
```

Supplementary Note 11: Tableau to visualize large data sets in interactive dashboards.

Data visualization allows to have a quick and powerful assessment of the data quality (Figure S3-S5). In the context of large data sets, their efficient visualization is key to obtain an assessment of their quality and message. However, plotting large amounts of datapoints in one plot may make it difficult to understand the data. To overcome this challenge, interactive visualization tools/dashboards allow to navigate easily through data and perform filtration and highlighting steps. To achieve this goal, ETHZ SwissCAT+ uses Tableau to create dashboards for data visualization. Data such as conversion vs. selectivity plots can be created and the plot can be dynamically modified by filtering by temperature or tasks (Figure S3). The catalyst composition can be visualized in stacked bar plots and filtered by the metal loading of individual metals (Figure S4). To obtain information about the performance of a whole batch/task, the average performance such as selectivity can be visualized (Figure S5).

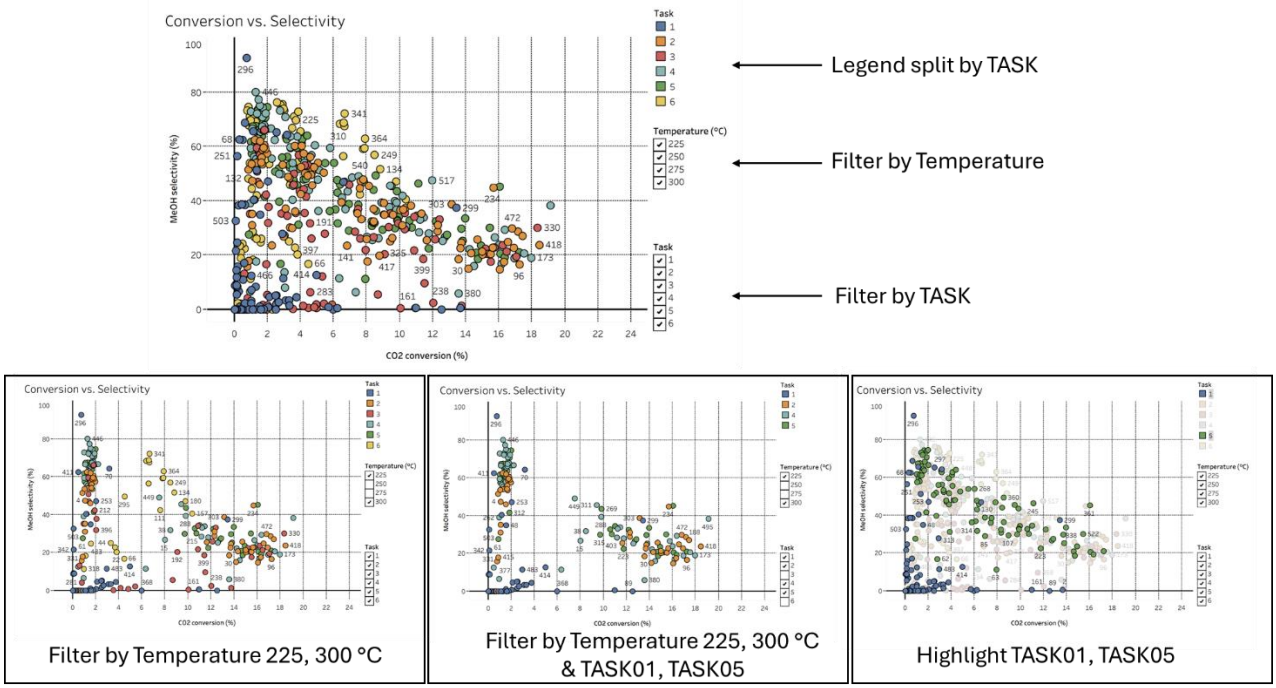

Figure S3. Example of interactive visualization dashboard to create conversion vs. selectivity scatter plot and filter plot by temperature and tasks.

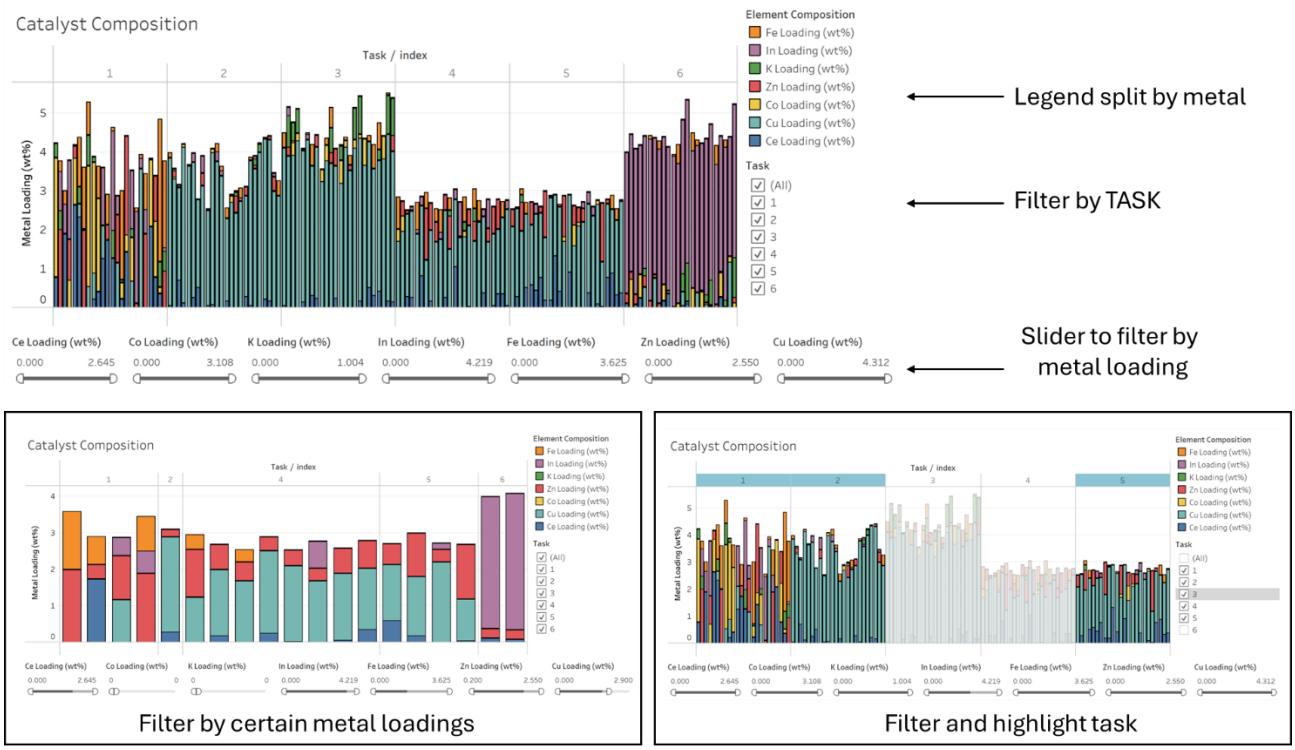

Figure S4. Example of interactive visualization dashboard to create metal composition bar plots and filter plots by tasks and metal loading.

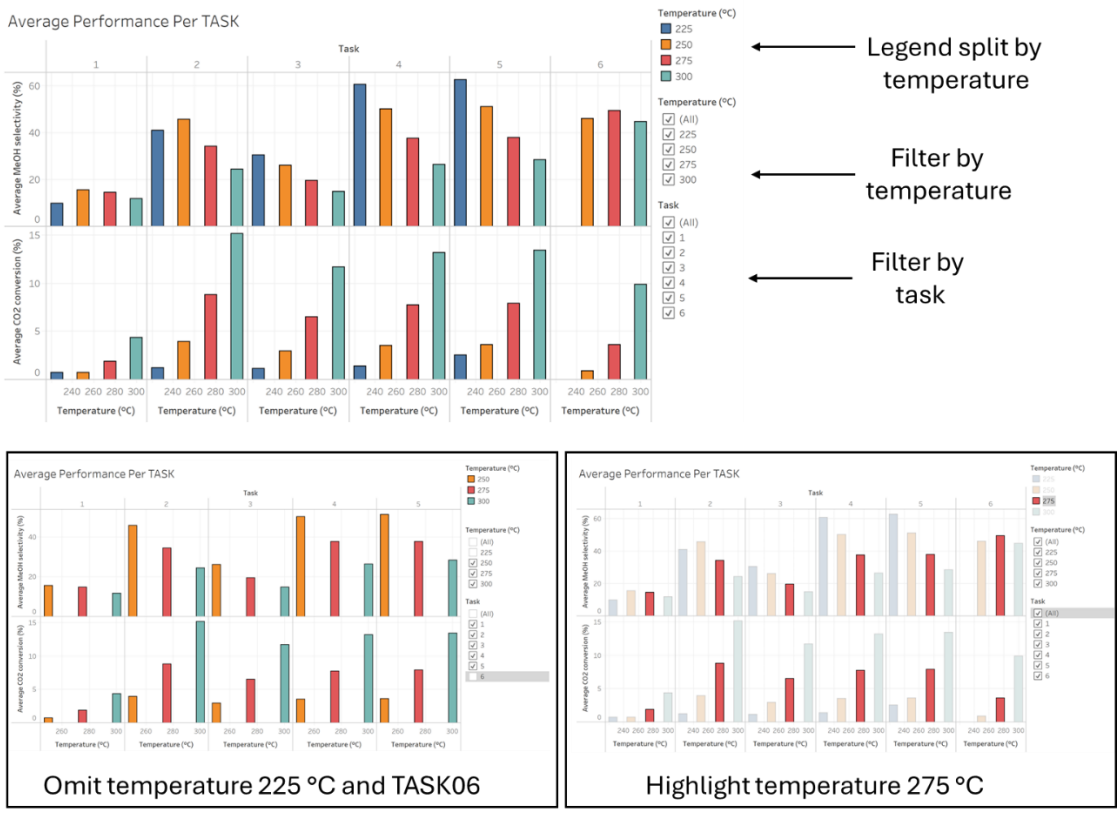

Figure S5. Example of interactive visualization dashboard to create average MeOH selectivity bar plots and filter by temperature and task.

Additional Information

Description of the configuration file inputs for openBIS data download, data reading and data merging.

```
#------------------------------------------------------------------------------#
#                                    Comments                                  #
#------------------------------------------------------------------------------#
#"""
# Args:
#   COMMENTS (str): Comments on the data
#"""
COMMENTS: !!str No comments
```

```
#------------------------------------------------------------------------------#
#                               OpenBIS Config                                 #
#------------------------------------------------------------------------------#
#"""
# Args:
#   URL (str): Url to the openBIS
#   DEFAULT_USER (str): openBIS username (prompt if null)
#   SAVE_TOKEN (bool): Save login token
#   PATH_STRUCTURE (seq): openBIS path structure
#   MAX_LOGIN_ATTEMPS (int): Number of login attempts before exit
#   DOWNLOAD_WORKERS (int): Number of parallel data downloads
#   DOWNLOAD_FILE_SOFT_LIMIT (int): Limit of file numbers before prompt to continue
#   DOWNLOAD_SIZE_SOFT_LIMIT (float): Limit of file size before prompt to continue
#"""
OPENBIS:
  URL_BASE: !!str openBIS_URL
  DEFAULT_USER: !!null
  SAVE_TOKEN: !!bool True
  PATH_STRUCTURE: !!seq ["space", "project", "experiment", "object", "dataset"]
  MAX_LOGIN_ATTEMPTS: !!int 3
  DOWNLOAD_WORKERS: !!int 10
  DOWNLOAD_FILE_SOFT_LIMIT: !!int 15
  DOWNLOAD_SIZE_SOFT_LIMIT: !!float 10e+6
```

```
#------------------------------------------------------------------------------#
#                               Folder Structure                               #
#------------------------------------------------------------------------------#
#"""
# Args:
#   Project (str): Name of the main folder
#   TASKS (str): Dictionary for subfolders and their information
#      SUBFOLDER (str): Name of subfolders e.g. TASK01, TASK02
#         FILES (str): Name of files to be expected e.g. REACTOR_LOADING
#            NAME (str): Name of the file within the instruction
#            FILE_ID (str): String in the file title for identification
#            DATA_SHEET (int or str): Title of excel datasheet if required
```

```
#            CSV_SEPARATOR (int or str): separator of the columns
#            TARGET_ID (str): Column title for merging
#            SOURCE_ID (str): Column title for merging
#"""
Project: !!str A001

TASKS:
  TASK01:
    REACTOR_LOADING:
      NAME: REACTOR_LOADING
      FILE_ID: !!str ReactorLoading
      DATA_SHEET: !!int 0
      CSV_SEPARATOR: !!str ","
      TARGET_ID: !!str Reactor
    REACTOR_DATA:
      NAME: REACTOR_DATA
      FILE_ID: !!str CATALYSIS
      CSV_SEPARATOR: !!str ","
      DATA_SHEET: !!str Run Data
      SOURCE_ID: !!str Reactor
```

```
#-------------------------------------------------------------------------------#
#                               Merging Information                             #
#-------------------------------------------------------------------------------#
#"""
# Args:
#   Project (str): Name of the main folder
#   MERGING_ORDER (str): Dictionary for subfolders and their information
#      SUBFOLDER (str): Name of subfolders e.g. TASK01, TASK02. Corresponding map
#                       with instructions on which files and column names to be
#                       merged (file 1 corresponds to left and file 2 corresponds
#                       to right)
#         left (str): Name of file title (e.g. REACTOR_LOADING) and file id
#                     (FILE_ID) to use as identifier
#         left_id (str): Name of file title (e.g. REACTOR_LOADING) and column name
#                        (TARGED_ID) to be expected
#         right (str): Name of file title (e.g. REACTOR_DATA) and file id
#                     (FILE_ID) to use as identifier
#         right_id (str): Name of file title (e.g. REACTOR_DATA) and column name
#                        (SOURCE_ID) to be expected
#"""

MERGING_ORDER:
  TASK01: !!map {
       left:     [REACTOR_LOADING, FILE_ID],
       left_id:  [REACTOR_LOADING, TARGET_ID],
       right:    [REACTOR_DATA, FILE_ID],
       right_id: [REACTOR_DATA, SOURCE_ID]
  }
```

```yaml
#------------------------------------------------------------------------------#
#                  Avantium Reactor Configuration for Renaming                 #
#------------------------------------------------------------------------------#
#"""
# Args:
#   REACTOR_UNIT (str): Dictionary for reactor units and their information
#      SUBFOLDER (str): Name of the unit (e.g. FLOWRENCE_T2101B)
#         NAME (str): Name of the file within the instruction
#         FILE_ID (str): String in the file title for identification
#         SETUP: Information about the reactor setup
#            INLET: Information about the inlet parameters
#               TEMPERATURE: Information about the temperature columns
#                  NAME (str): Column name to be renamed
#                  UNITS (str): Unit string for the renaming
#                  SET_POINT (str): Column name for temperature setpoint
#                  PROCESS_VALUE (str): Column name for temperature process value
#               PRESSURE: Information about the pressure columns
#                  NAME (str): Column name to be renamed
#                  UNITS (str): Unit string for the renaming
#                  SET_POINT (str): Column name for pressure setpoint
#                  PROCESS_VALUE (str): Column name for pressure process value
#               GAS_FLOW: Information about the MFC columns
#                  MFC_NAME (str): Name of the MFC (e.g. FIC_110)
#                     NAME (str): Column name to be renamed
#                     UNITS (str): Unit string for the renaming
#                     COMPOSITION (map): Gas composition
#                     SET_POINT (str): Column name for gas flow setpoint
#                     PROCESS_VALUE (str): Column name for gas flow process value
#"""
#------------------------------------------------------------------------------#
REACTOR_UNIT:
  FLOWRENCE_T2101B:
    NAME: !!str XDB
    FILE_ID: !!str T2101B
    SETUP: &XD_SETUP
      INLET:
        TEMPERATURE:
          NAME: !!str Reactor Temperature
          UNITS: !!str "[C]"
          SET_POINT: !!str Reactor_Temperature_SP
          PROCESS_VALUE: !!str Reactor_Temperature_PV

        PRESSURE:
          NAME: !!str Reactor Pressure
          UNITS: !!str "[Barg]"
          SET_POINT: ReactorPressure_SP
          PROCESS_VALUE: !!str Reactor_Inlet_Pressure

        GAS_FLOW:
          FIC_110:
```

```
            NAME: !!str H2
            UNITS: !!str "[mL Min-1]"
            COMPOSITION: !!map {"H2": 1}
            SET_POINT: !!str FIC_110_SP
            PROCESS_VALUE: !!str FIC_110_PV
```

```
#------------------------------------------------------------------------------#
#                               Data Processing
#------------------------------------------------------------------------------#
#"""
# Args:
#   PROCESS_VALUE (str): Name of the main folder
#   COLUMNS_TO_COPY (seq): Dictionary for subfolders and their information
#   PROCESS_VALUE_LABEL (str): Name of subfolders e.g. TASK01, TASK02
#   SET_POINT_LABEL (str): Name of files to be expected e.g. REACTOR_LOADING
#            NAME (str): Name of the file within the instruction
#            FILE_ID (str): String in the file title for identification
#            DATA_SHEET (int or str): Title of excel datasheet if required
#            CSV_SEPARATOR (int or str): separator of the columns
#            TARGET_ID (str): Column title for merging
#            SOURCE_ID (str): Column title for merging
#"""
#------------------------------------------------------------------------------#
DATA_PROCESSING:
  PROCESS_VALUE: "Target Weight (mg)"
  COLUMNS_TO_COPY: !!seq ["RunName",  "Unit", "Reactor", "Block", "Reactor
Temperature_SP_[C]", "Reactor Pressure_PV_[Barg]"]
  GROUP_TO_MEAN: !!seq ["Unit","Reactor","TASK", "Reactor Temperature_SP_[C]"]
  # Labeling of data
  PROCESS_VALUE_LABEL: !!str "_PV"
  SET_POINT_LABEL: !!str "_SP"
```

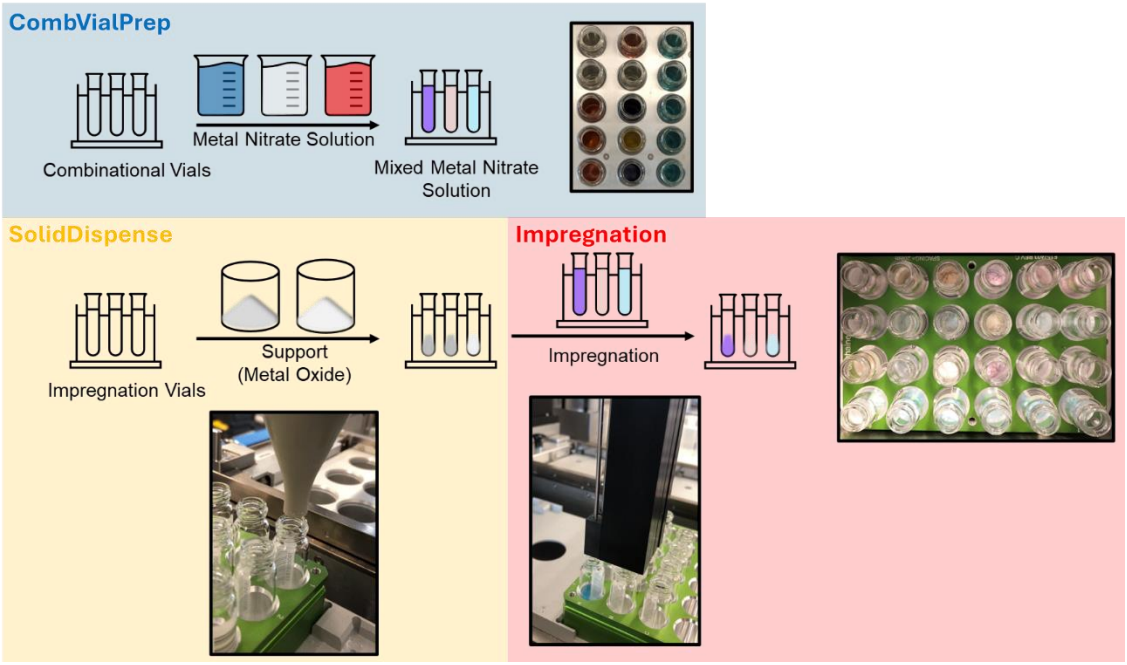

Figure S6. Experimental workflow and location of the output data of mixed metal nitrate solution preparation (CombVialPrep), metal oxide solid dispense (SolidDispense) and incipient wetness impregnation (Impregnation).

Figure S7. Experimental workflow and location of the output data of synthesis vial barcode scanning (ImpregnationBarcode), catalyst dispense, barcode scanning of synthesis and destination vial (CatalystDispenseBarcode), loading of reactors for testing (ReactorLoading), results of fixed-bed testing (CATALYSIS_XR/XDB/XDC_RUN01) and market cost of used metal (COST).

End of Supporting Information