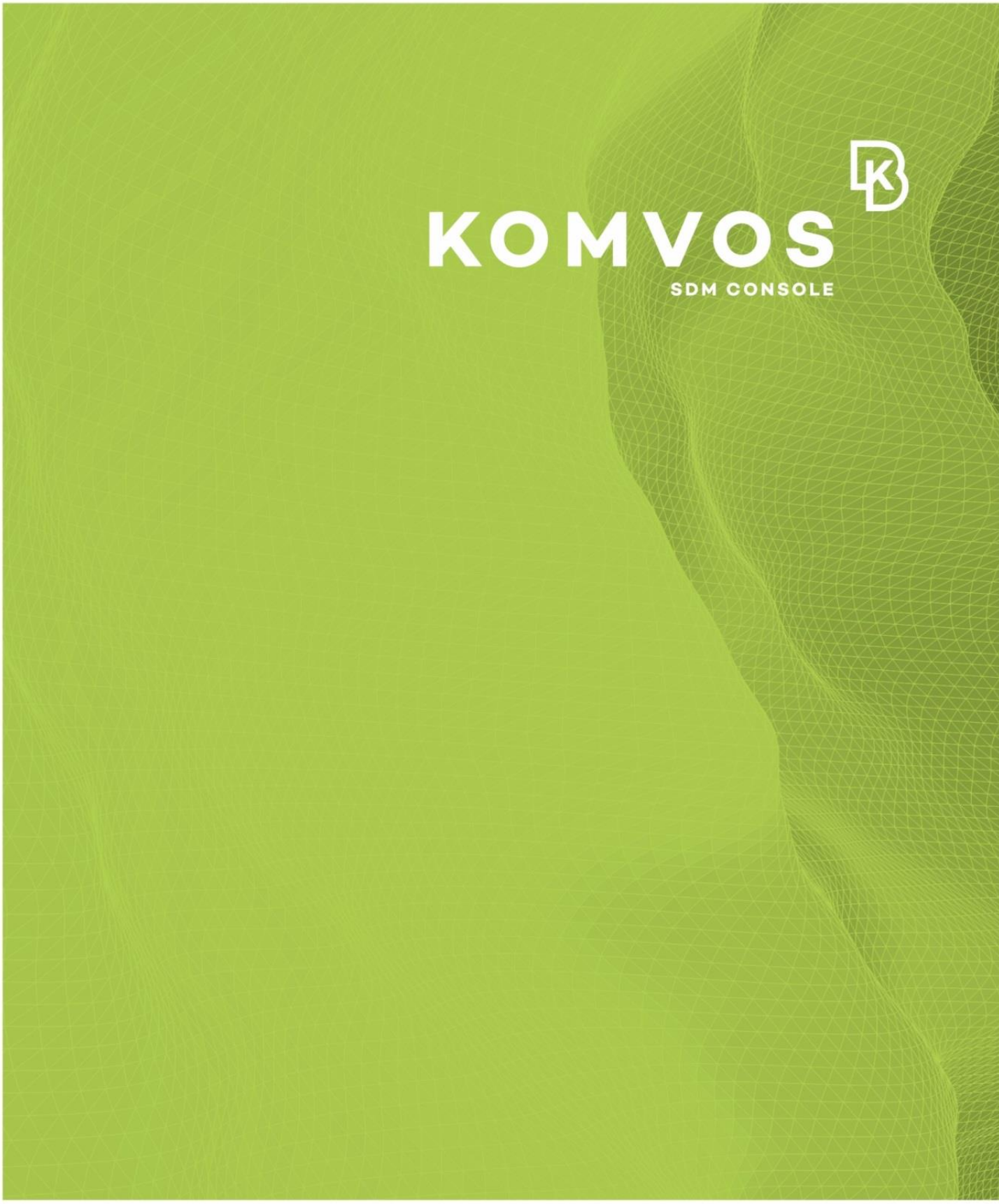
BETA
SIMULATION SOLUTIONS

physics on screen



KOMVOS version 24.1.x User Guide

Updated in December 2023

BETA CAE Systems International AG

D4 Business Village Luzern, Platz 4

CH-6039 Root D4, Switzerland

T +41 41 545 3650, F +41 41 545 3651

Table of Contents

1. Introduction

1.1. Purpose and function

KOMVOS is a standalone, desktop application that serves as a rich client of Simulation Process and Data Management systems. With KOMVOS, the users can search for data, review product structures, preview models in the embedded 3D viewer, send data to external applications, export data, review processed results, design and execute processes, get notifications on completed HPC jobs, and many more.

KOMVOS comes with a rich portfolio of functionality that fully covers the needs of different CAE user personas:

- **Model build engineers**: Such users can import data from PDM systems, run CAD conversion and meshing batch operations, version control their data, identify carry-over parts and commonalities between different variants, compare different part and subsystem versions, etc.
- **CAE modelling team leaders**: Such users can populate the CAE library with templates and other library data like materials and header files, review data pedigrees to understand data dependencies and perform impact or root cause analysis, monitor the progress of team tasks, etc. Furthermore, users can capture Standard Operating Procedures as Process Templates, including information on data flow, software applications and assignees.
- **Analysis engineer**: Such users can build Simulation Runs in ANSA or other pre-processors and store them in the SDM system, make use of standard processes for job submission, monitoring and post-processing, review Simulation Runs and their processed results in the embedded Viewer, compare on-the-fly the models of different simulation versions and their results in the embedded viewer or logbook, create correlation studies with Test Results, build Optimization Studies from DOEs or with the aid of Machine Learning, etc.
- **CAE Manager**: These users can extract reports for the simulation, visualize Summarized Reports in the embedded Viewer, get statistics for past process executions, etc.

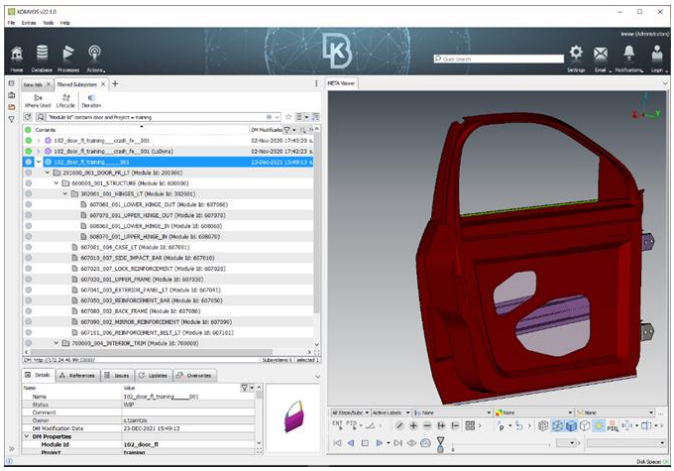
KOMVOS can be used as a front-end for both the primary SDM solutions offered by BETA CAE Systems: **File-based DM** and **SPDRM**, the server-based SDM solution.

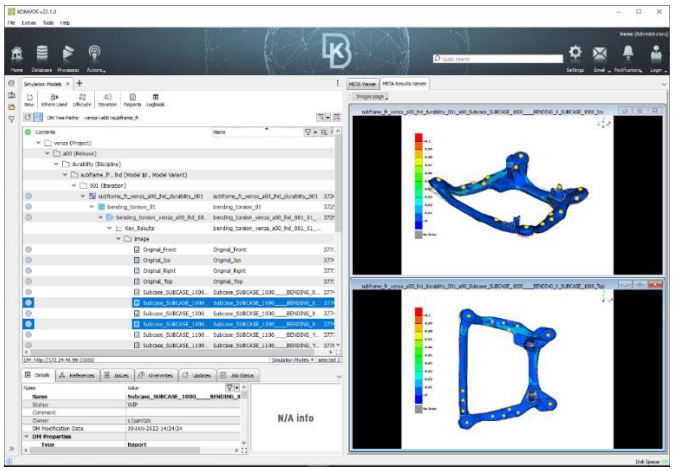
- **File-based DM** is an entry level solution, suitable for small projects and collocated teams of a size up to 10-15 users.
- **SPDRM** is the scalable, enterprise-level solution that enables user management, data security, collaboration of local and remote teams, high concurrency and management of Business Processes.

Furthermore, KOMVOS includes out-of-the-box interfaces that enable the use of third-party applications as data back-ends. Right now, two such options are available:

- **Interface to MSC SimManager**: For the teams that already make use of MSC's SimManager as an Enterprise SDM system, the built-in interface enables data browsing and direct check-in and check-out of data.
- **Interface to ASAM-ODS-compliant Test Result servers**: For direct access to test data like curves, images, videos, key-values etc. that are mostly needed for correlation between simulation and test results, the built-in interface enables browsing, direct check-in and checkout of data through KOMVOS, Dashboard and ANSA/META.

KOMVOS offers an intuitive interface that facilitates all data and process management activities. The graphical user interface uses the same design elements with the BETA Suite and follows same design principles, requiring no additional effort for a user familiar with ANSA or META to get started. Especially for data management, the Data Management workspace of KOMVOS is very similar to the DM Browser of ANSA and META.





KOMVOS is highly customizable and can be adapted to the needs of every engineering team. Customization of KOMVOS is not limited to how information is presented to the end-user but also covers the data processing needs of different teams with the support for custom actions that run Python scripts and are installed either as context menu actions or as generic actions on the ribbon. KOMVOS' administrator can configure the application centrally and then let the user of each team access the appropriate set-up based on his/her profile.

1.2. About this document

This document is intended to guide end-users of KOMVOS on how it to use efficiently for various CAE-related tasks.

The primary function of KOMVOS is the management of data, so data navigation and handling functionality in the Data Management workspace is extensively described in chapter 4. Note that core data management concepts are not covered in this document. For such information, please refer to the Data Management Reference Manual.

Process and Issue Management workspaces, both enabled only with SPDRM as a back-end, are described in chapters 5 and 6 respectively.

Custom actions accessible through the Actions menu of the ribbon and through context menus are described in chapter 7.

KOMVOS comes with an out-of-the-box customization layer that facilitates CAE Model Build from CAD/PDM info. With this functionality, a user can create Subsystems by reading in a product structure and then use KOMVOS as pre-processing orchestration console, in order to manage actions like CAD files translation, extraction of mid-surface and meshing with the aid of ANSA workers in the background. This functionality is described in chapter 8.

Inbuilt support for the handling of Optimization Studies and integrated Machine Learning functionality are also included and enable the prediction of the impact of design changes in the early stages of product development. Information on these topics is given in chapter 9.

Finally, information on the various configuration possibilities of KOMVOS is given in chapter 10.

2. Getting started

There are two key locations for a KOMVOS session:

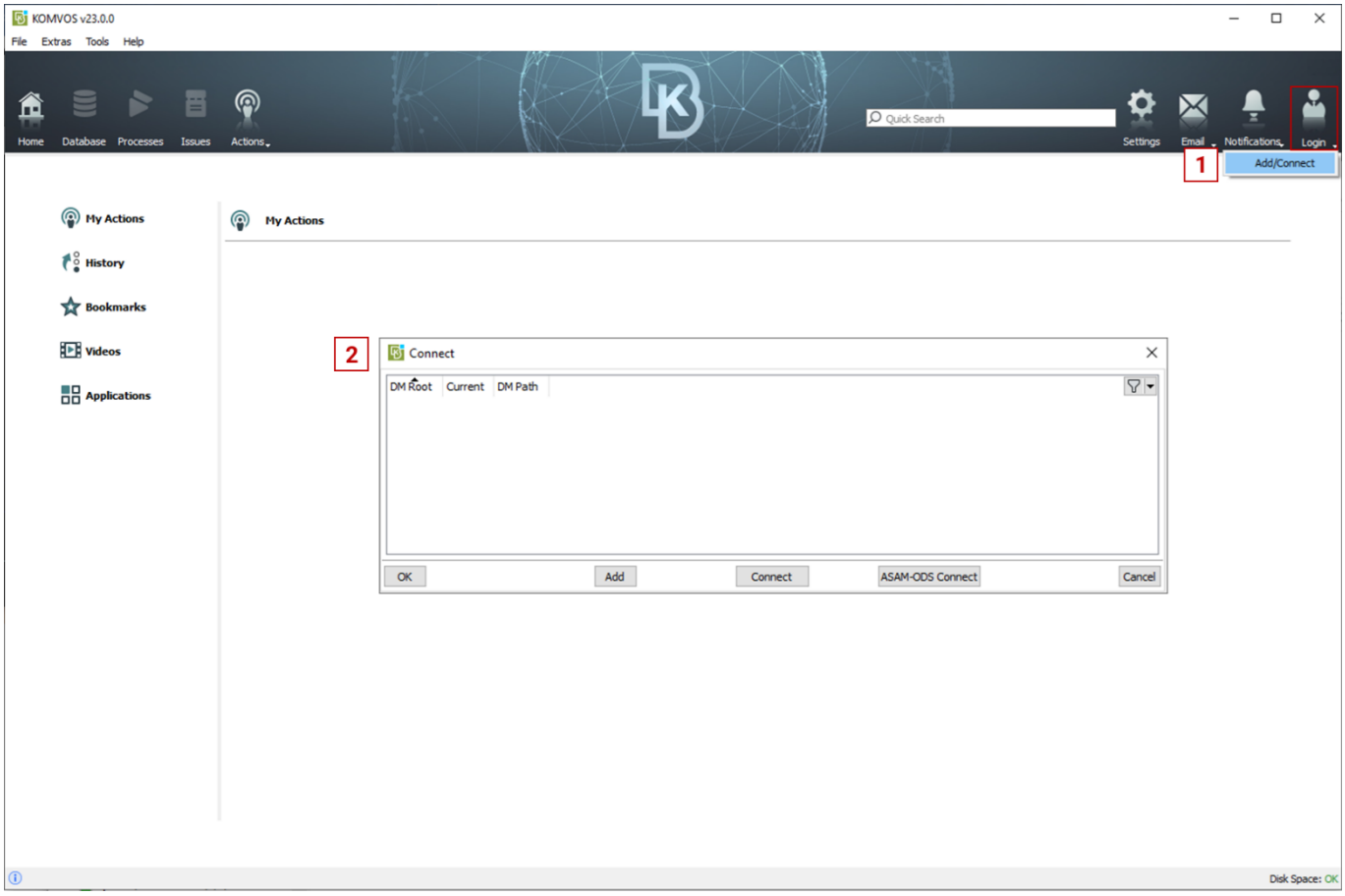
1. The location of the SDM back-end: This is the data (and process) provider and needs to be specified in order for KOMVOS to display data or process management information.

2. The SDM_CONSOLE_HOME: This is the name of the environmental variable that indicates the location where the central configuration of KOMVOS is done. By default, the SDM_CONSOLE_HOME is the **config** folder in the installation directory and contains the default configuration files that come with the installation. More information on the SDM_CONSOLE_HOME and on configuration possibilities is given in chapter 10.

This chapter provides information on what's required in order to get started with KOMVOS.

2.1. Connecting to an SDM back-end

The first thing to be configured when opening KOMVOS for the first time is the SDM back-end. This is done through the **Login** option on the right of the top ribbon.



In the Connect window that pops-up, the user can define the SDM back-end to be used. The following options are supported:

- **File-based DM**: A DM root directory can be selected by pressing the **Add** button. Through the File Manager that pops-up, an existing DM root directory or a new directory can be specified.
- **Server-based DM:** A server-based SDM back-end can be selected by pressing the **Connect** button. Through the window that pops-up, the server URL of an SPDRM server must be defined together with the user's credentials that will be submitted by KOMVOS to the server for authentication. Through this dialog it is also possible to establish a connection with an MSC SimManager back-end.
- **Test-results server**: A test-results server can be selected by pressing the **ASAM-ODS Connect** button. Through the window that pops-up, all information required by KOMVOS in order to establish a connection to the test-results server is specified.

The moment KOMVOS gets connected to an SDM back-end, several settings are read that considerably affect the content, functionality and overall layout of the application. For more information on how the back-end may affect KOMVOS, please refer to paragraph 10.1.

2.2. Configured environment upon launch

Central configuration of KOMVOS is done in its "home" directory, which is the directory indicated by the environment variable `SDM_CONSOLE_HOME`. By default, this environment variable is set in the KOMVOS start script and points to the **config** folder in the installation directory. If this environment variable has some value already before launching KOMVOS, that value prevails.

If the `SDM_CONSOLE_HOME` directory contains a folder named **SPDRM**, then this folder will be used as source for all configuration files when KOMVOS is connected to an SPDRM back-end. If this file does not exist, the source for all configuration files will be the `SDM_CONSOLE_HOME` even in case of SPDRM back-end.

Every time the user closes KOMVOS, all modifications related to UI adjustments are stored automatically in the user's home directory under:

```
.BETA/KOMVOS/version_XX.X.X/KOMVOS.xml
```

For more information on the configuration of KOMVOS, please refer to chapter 10 of this guide.

2.3. Default configuration

2.3.1. For file-based back-ends

The default configuration of KOMVOS for file-based back-ends is made available in the **config** directory in KOMVOS installation (default `$SDM_CONSOLE_HOME`).

It includes a `dm_views.xml` that customizes the views of the Database workspace and the embedded META Viewer. Note that the default dm_views.xml tunes the data views assuming the default data model.

Additionally, the default configuration includes a customization layer that facilitates CAE Model Build from CAD/PDM The customization is implemented with Python scripts that are also available within the **config** directory.

An in-depth description of the default configuration of KOMVOS for file-based back-ends is given in chapter 8 of this guide.

For more information on configuration topics, please refer to Chapter 10.

2.3.2. For SPDRM

The default configuration of KOMVOS for SPDRM back-ends is made available in the **config/SPDRM** directory in KOMVOS installation (default `$SDM_CONSOLE_HOME/SPDRM`).

It includes a `dm_views.xml` that customizes the views of the Database workspace assuming the default data model being used by SPDRM.

For more information on configuration topics, please refer to Chapter 10.

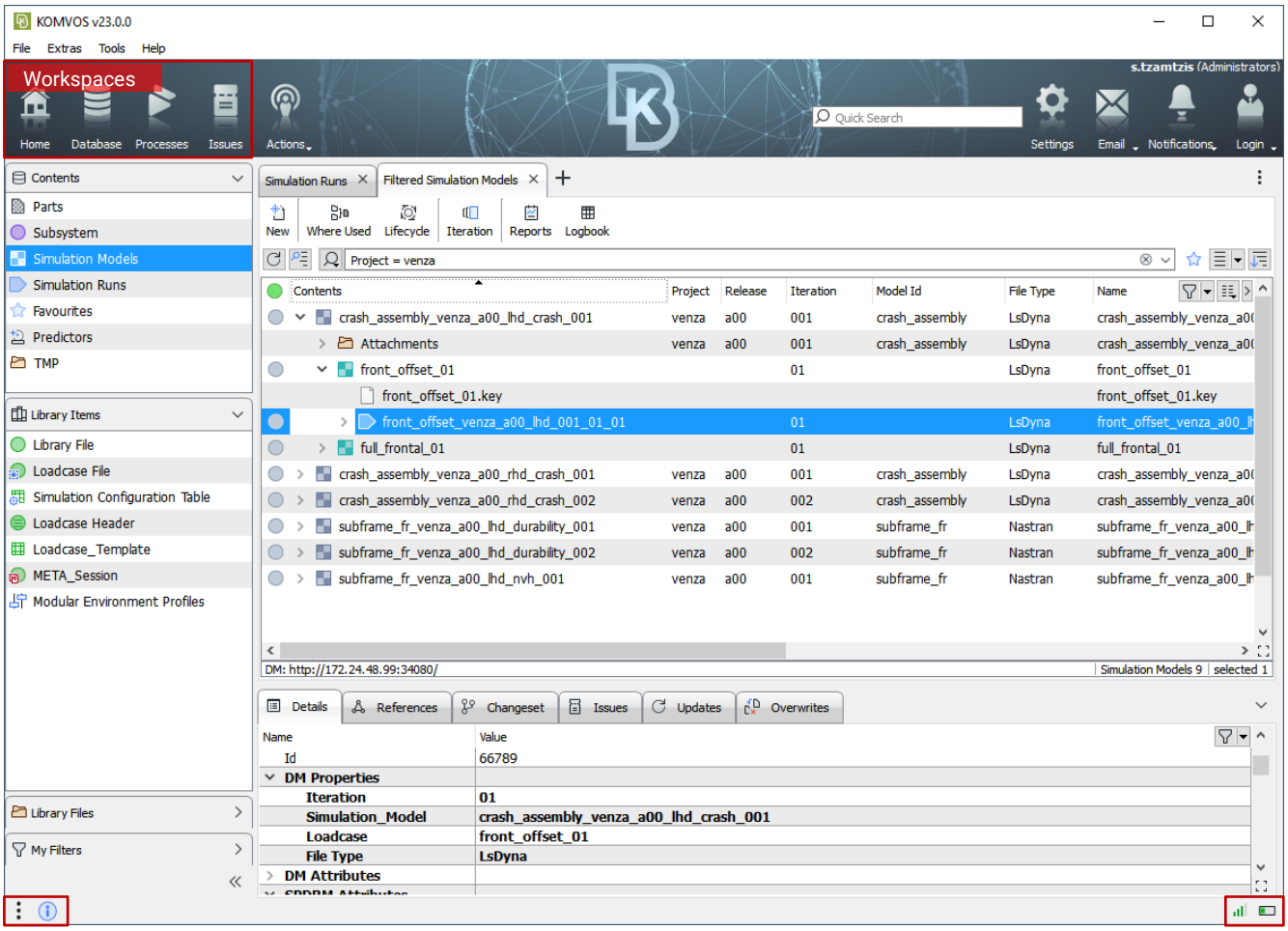
3. User Interface

The graphical user interface of KOMVOS facilitates all data and process management activities. It uses the same design elements with the BETA Suite and follows same design principles, making it quite intuitive and requiring no additional effort for a user familiar with ANSA or META to get started.

At the top of the user interface a toolbar offers access to several tools. Looking at the toolbar buttons, from left to right:

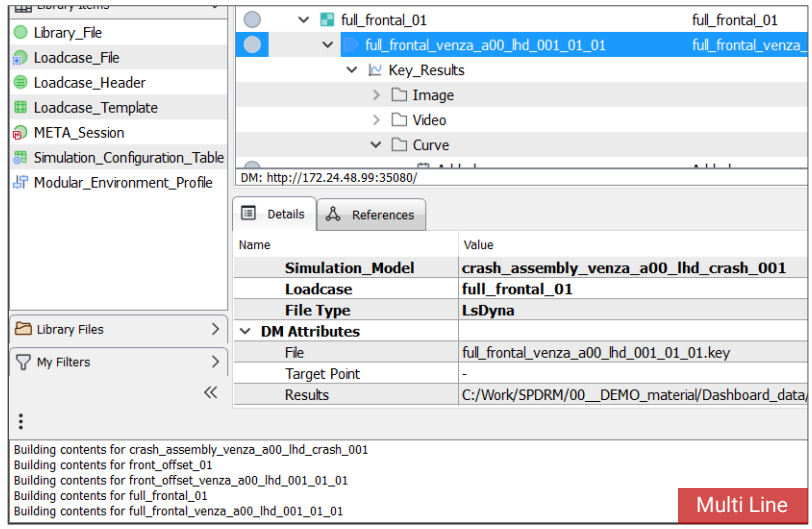
- The various **Workspaces** can be accessed.
- The available **Actions** can be invoked.
- A **Quick Search** in the database can be performed.
- The KOMVOS **Settings** can be accessed.
- **Email** messages can be sent.
- The details of any received **Notifications** can be viewed.
- The **Login** to an SDM backend can be achieved.

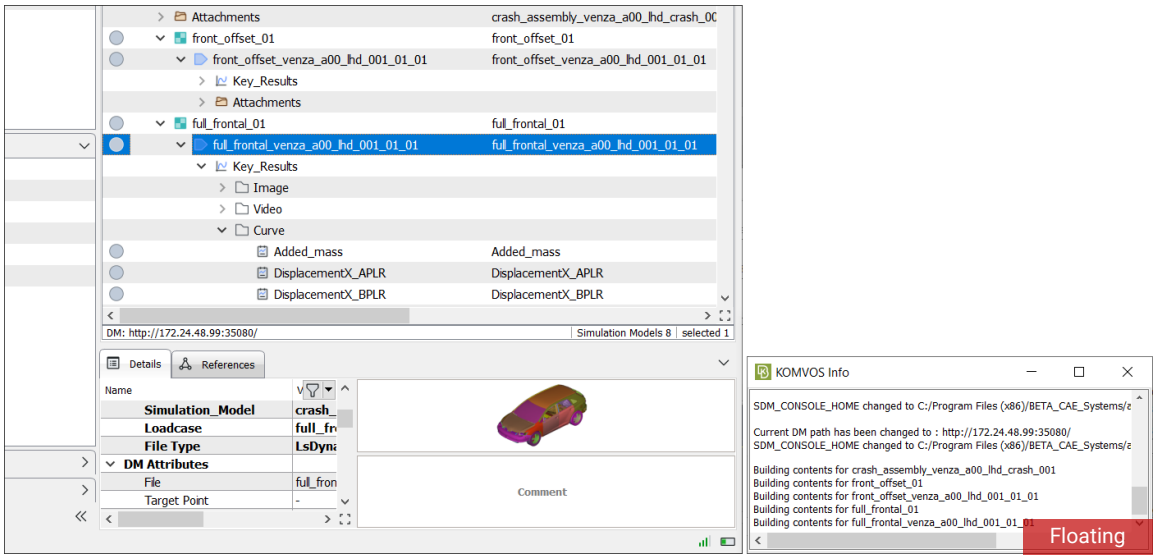
Below the toolbar, the main section of the user interface displays the contents of the active *Workspace*.



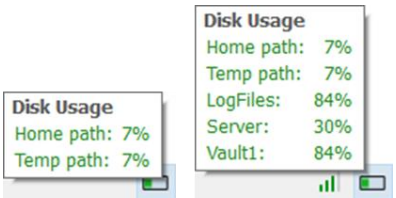
At the bottom left corner, in the KOMVOS *Info Feed*, information messages will appear, that can be accessed at a later time pressing the respective button ⓘ.

In case the user would like to keep information messages always visible, pressing the ⋮ button, alternative views are offered, with the capability to change to a multi-line embedded information window or a floating *KOMVOS Info* window.

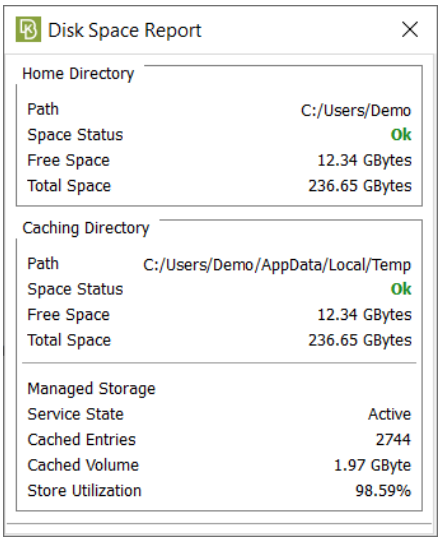




At the bottom right corner of the window, an indication of the connection quality (server based back-ends) and the disk space usage, is displayed. On mouse hover, a summary of the disk space usage is displayed in the form of a tooltip. Pressing the disk usage button, the *Disk Space Report* window appears with all related information.



File-based DM *SPDRM back-end*



If not already configured, the first thing to be done after opening KOMVOS is the *Login* to an SDM back-end, using the respective button in the toolbar. A detailed description is provided in chapter 2 of this guide.

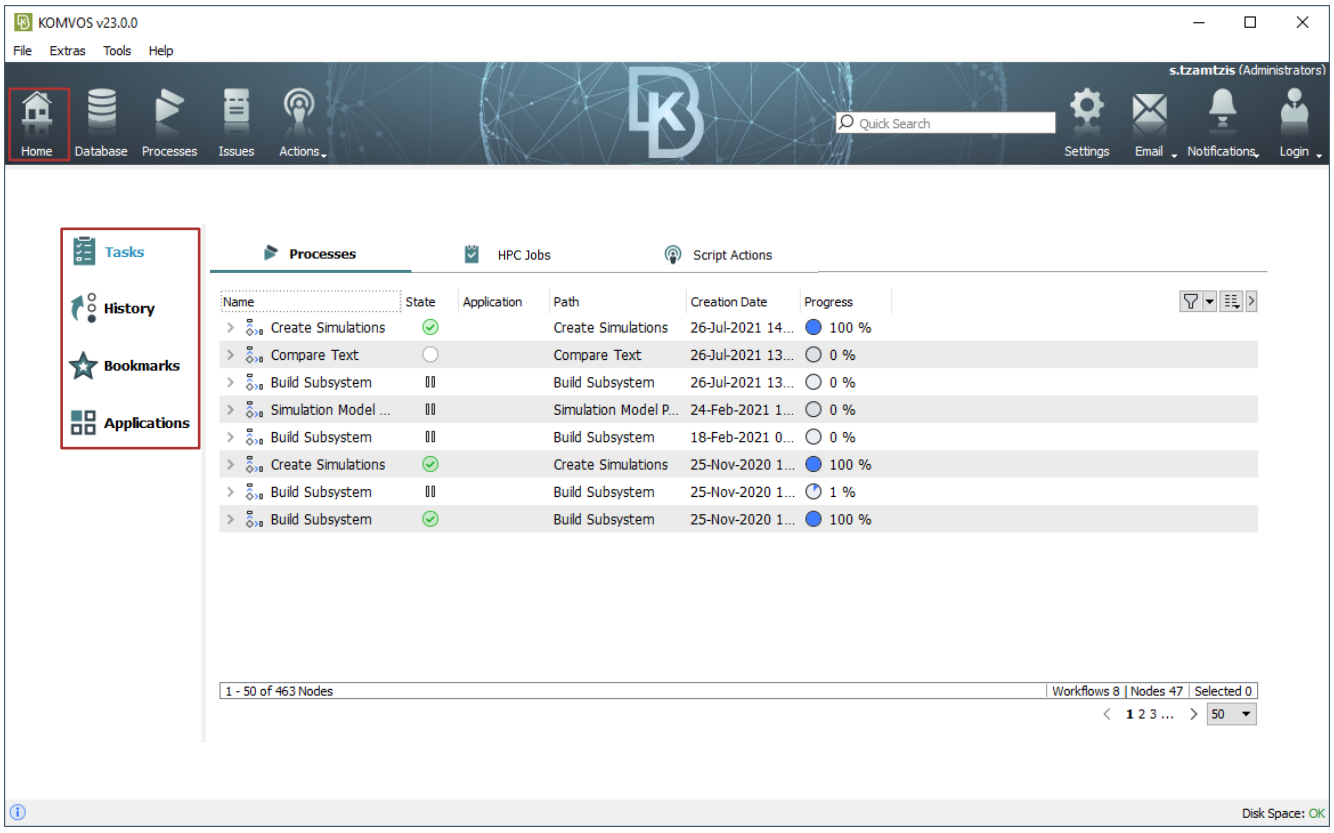
Once connected to an SDM back-end, access to the available *Workspaces* will be enabled. The following workspaces are offered in KOMVOS:

- The *Home* workspace offers access to recently performed actions, submitted jobs, running processes, bookmarks and an applications launchpad.
- The *Database* workspace offers access to data management functionality.
- The *Processes* workspace offers access to the SPDRM process management functionality. It is only enabled when connected to an SPDRM back-end of version 1.6.0 or later.
- The *Issues* workspace offers access to the SPDRM issue management functionality. It is only enabled when connected to an SPDRM back-end of version 1.8.0 or later.

A more detailed description of the user interface in each workspace is given in the respective paragraphs of this chapter.

3.1. Home

The **Home** workspace is the starting page of KOMVOS and offers a collection of information interesting to the logged in user. The information is organized in 4 categories that can be accessed through the shortcuts on the left.

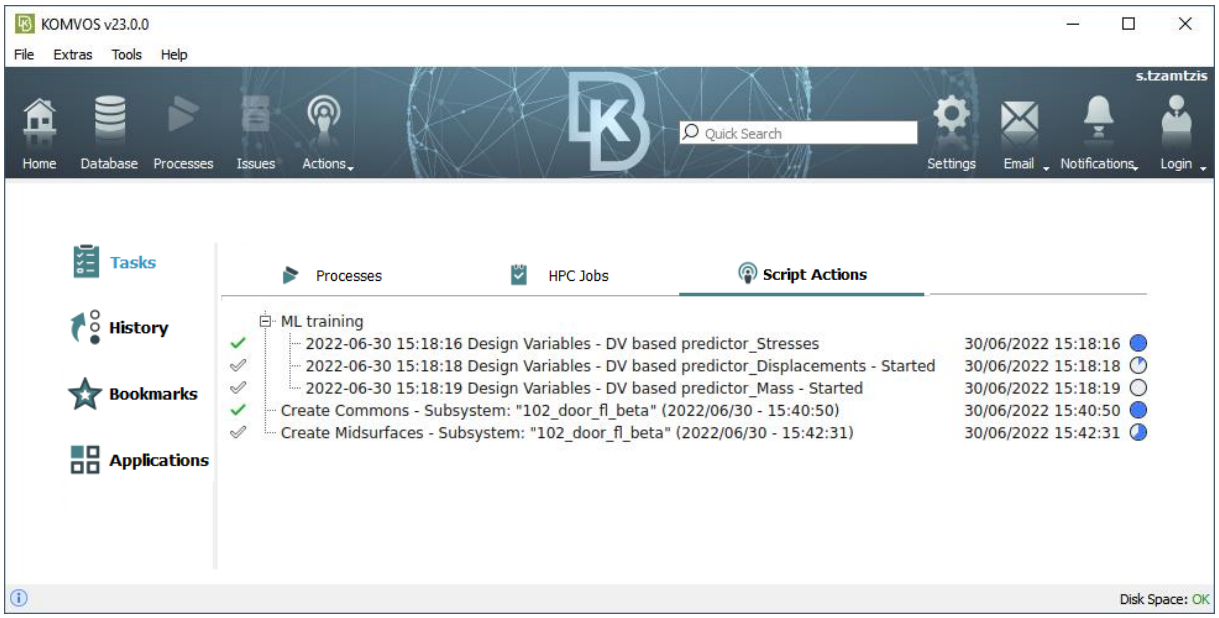


3.1.1. Tasks

All process-related pages are grouped under the **Tasks** into three sub-tabs.

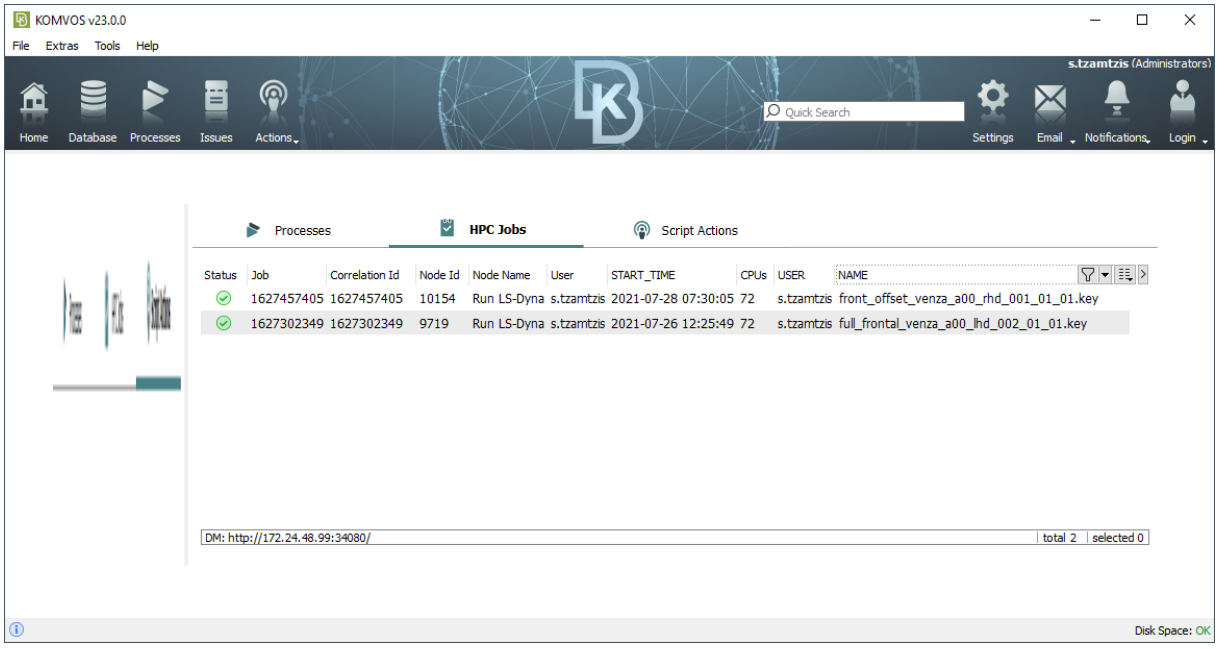
3.1.1.1. Script Actions

All KOMVOS' script actions that offer progress monitoring functionality and were initiated by the logged in user will be shown in this list.



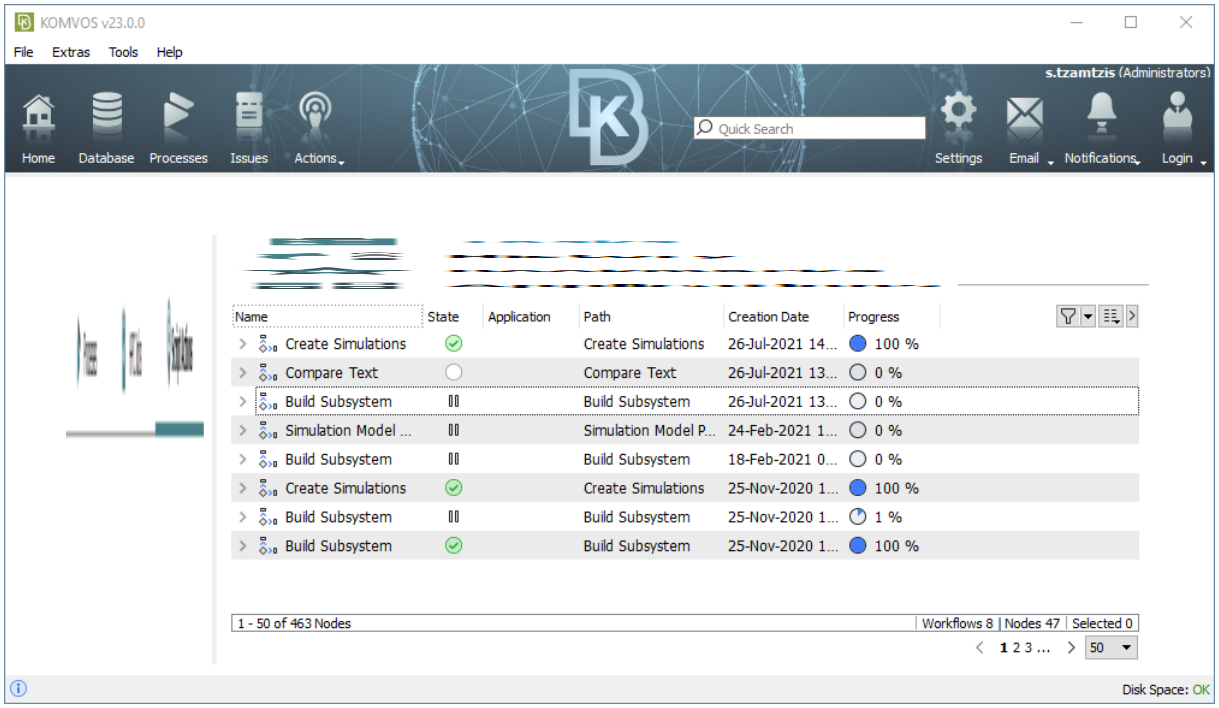
3.1.1.2. HPC Jobs

A list of all HPC jobs submitted by the logged in user can be accessed using *HPC Jobs* sub-tab. The user can access important submission information, such as the *Job* and *Correlation Id*, the submission *Status*, etc. This option is only available when connected to an SPDRM back-end of version 1.7.0 or later.



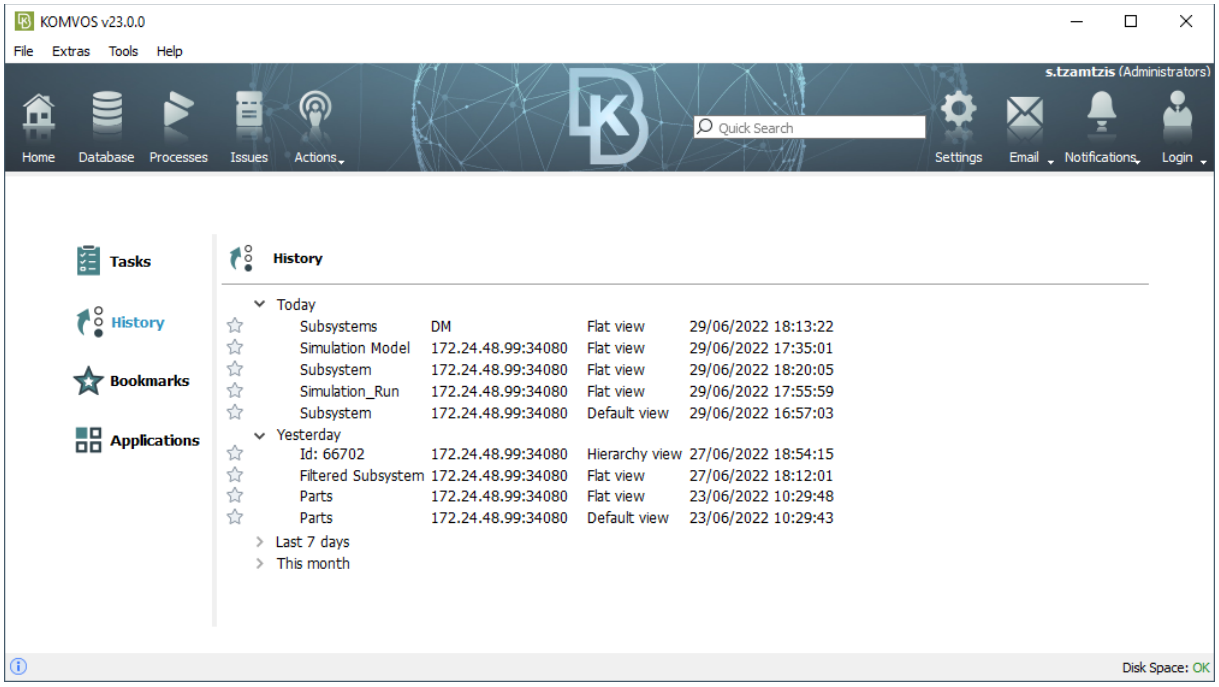
3.1.1.3. Processes

A list of all SPDRM process instances assigned to the logged in user can be accessed here. This option is only available when connected to an SPDRM back-end of version 1.6.0 or later. The user can access useful information, such as the workflow *Name*, the instance *Creation Date*, the execution *State* and *Progress*, etc. Using the context menu, each process instance can be opened in the *Processes* workspace.



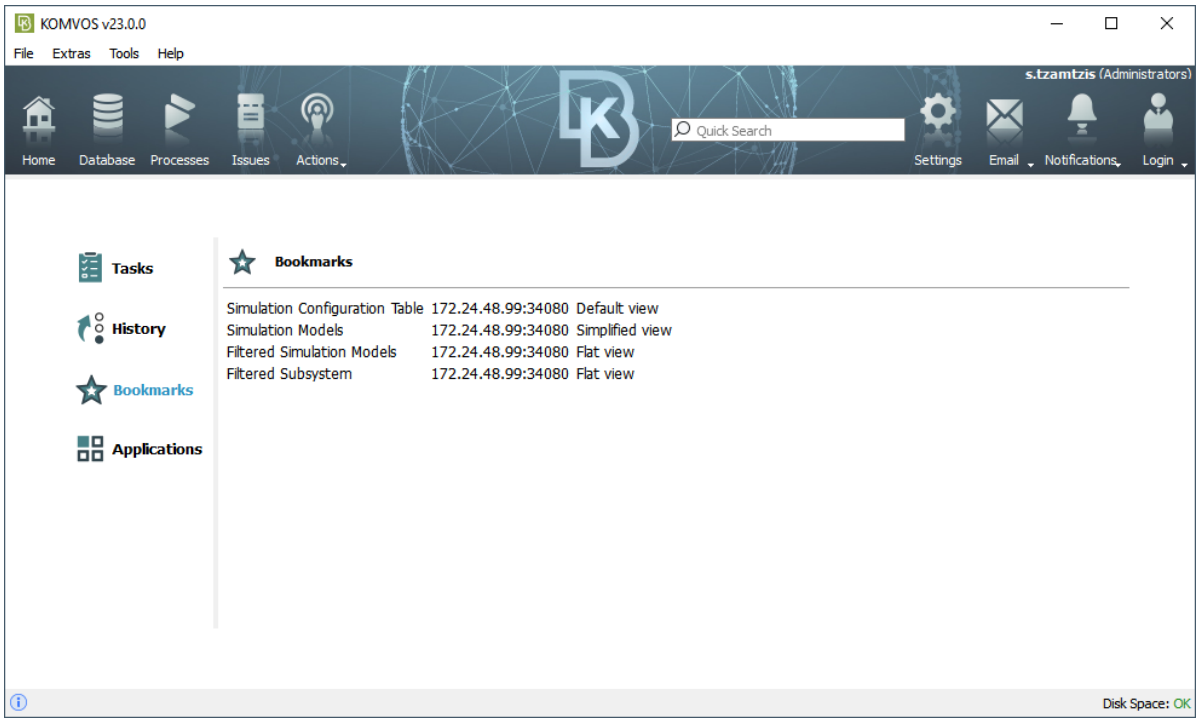
3.1.2. History

This offers access to recently viewed DM objects, in any SDM back-end that the user may have accessed through KOMVOS. Using the context menu, the selected entities can be re-opened in the *Database* workspace.



3.1.3. Bookmarks

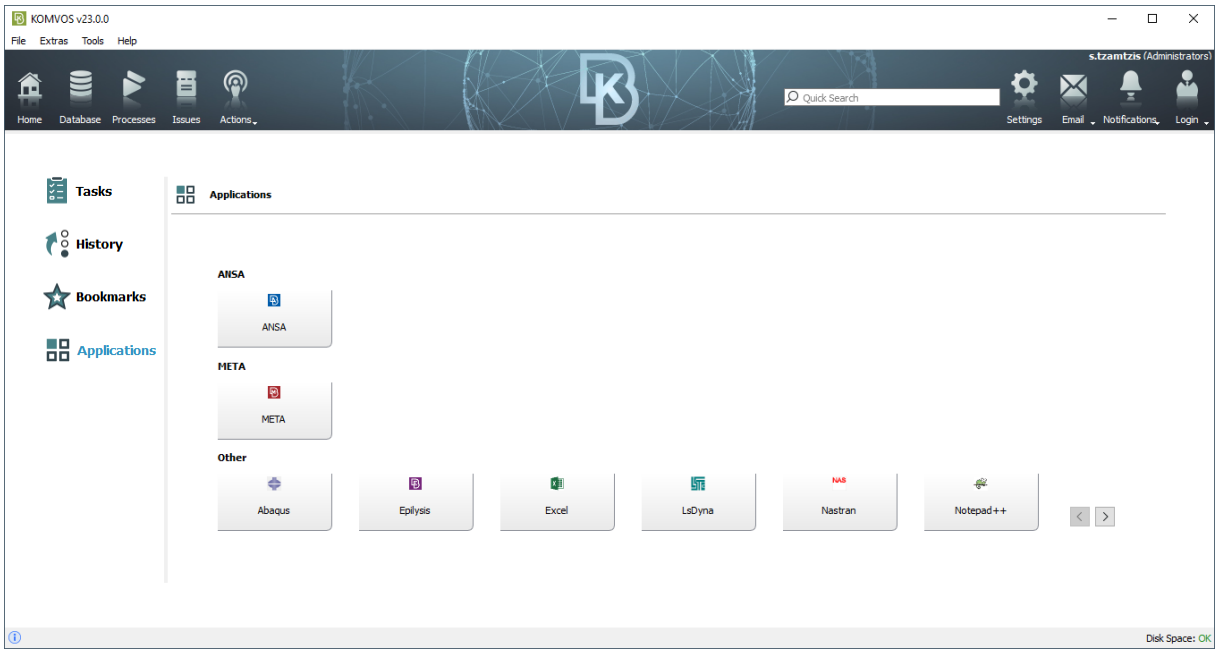
Using the **Bookmarks**, the user can easily access *Database* views added as favorites. Any of the listed bookmarks can be opened in a new tab in the *Database* workspace.



3.1.4. Applications

This offers access to a launchpad to quickly launch relevant applications.

- With SPDRM back-end: All applications registered in SPDRM are listed here. Registration of SPDRM applications is done by the SPDRM Administrator through the SPDRM Administration Console. For more information, please refer to the SPDRM Administrator's guide.
- With file-based DM: ANSA and META, as these are configured in the KOMVOS' Settings.

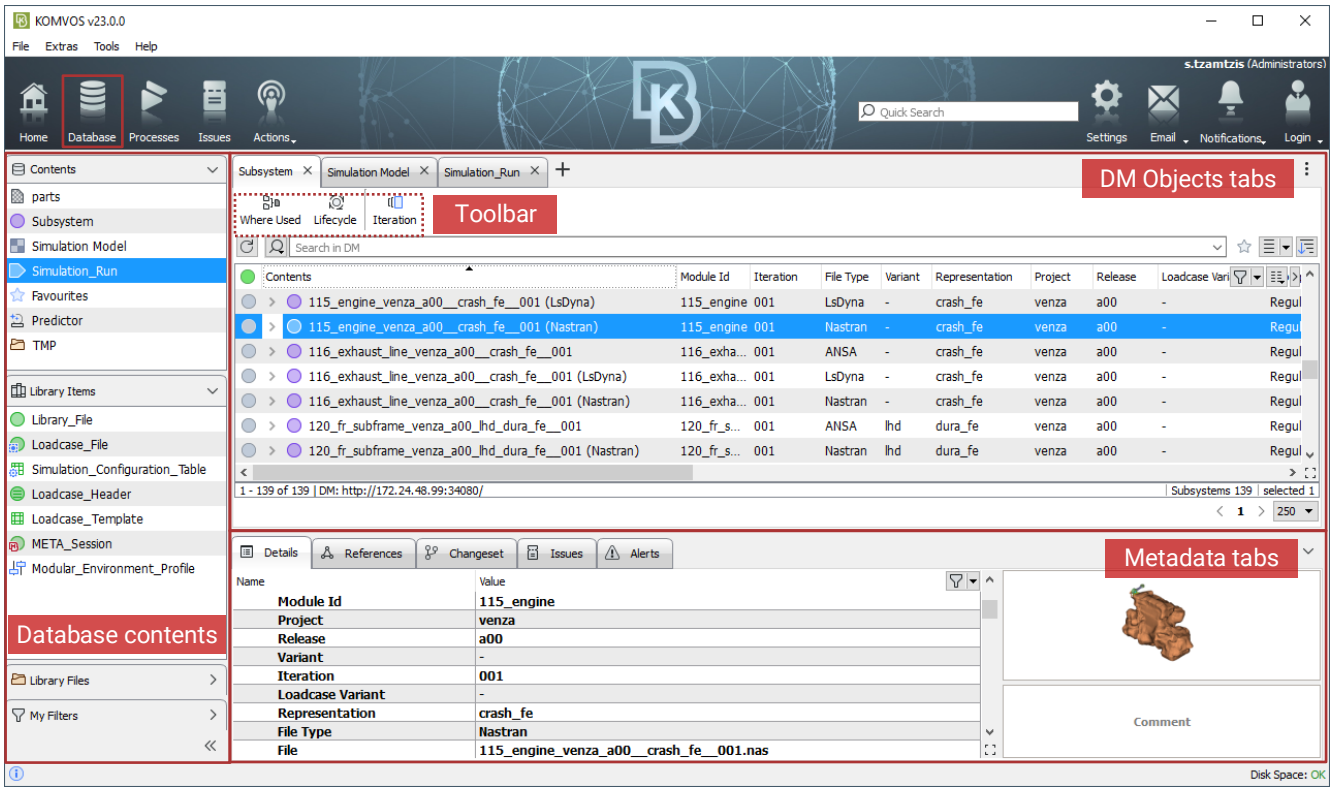


3.2. Database

The **Database** workspace offers an organized overview of the data stored in the SDM repository, coupled with functionality for effective data management. The window is split vertically in two main sections:

- On the left, there's the Database Contents Index pane that lists the data types contained in the database. The contents of each container are opened in a new tab in the section on the right. In addition, a container listing all user-defined filters for queries in the database is available.
- On the right, the workspace is split horizontally in two sections:
  - At the top, several tabs can be opened, each tab listing the DM objects of a different data type. A toolbar offering quick access to data management tools is available in each tab.
  - At the bottom, various metadata tabs are listed, providing information for the DM object selected in the upper section, in the DM object list. The availability of some metadata tabs varies, depending on whether KOMVOS is connected to a file-based or an SPDRM back-end.

For a detailed description of the *Data Management* functionality in KOMVOS, please refer to chapter 4 of this guide.



3.3. Process

When connected to an SPDRM back-end of version 1.6.0 or later, the **Process** workspace is activated offering access to process management functionality. The window is split vertically in three sections:

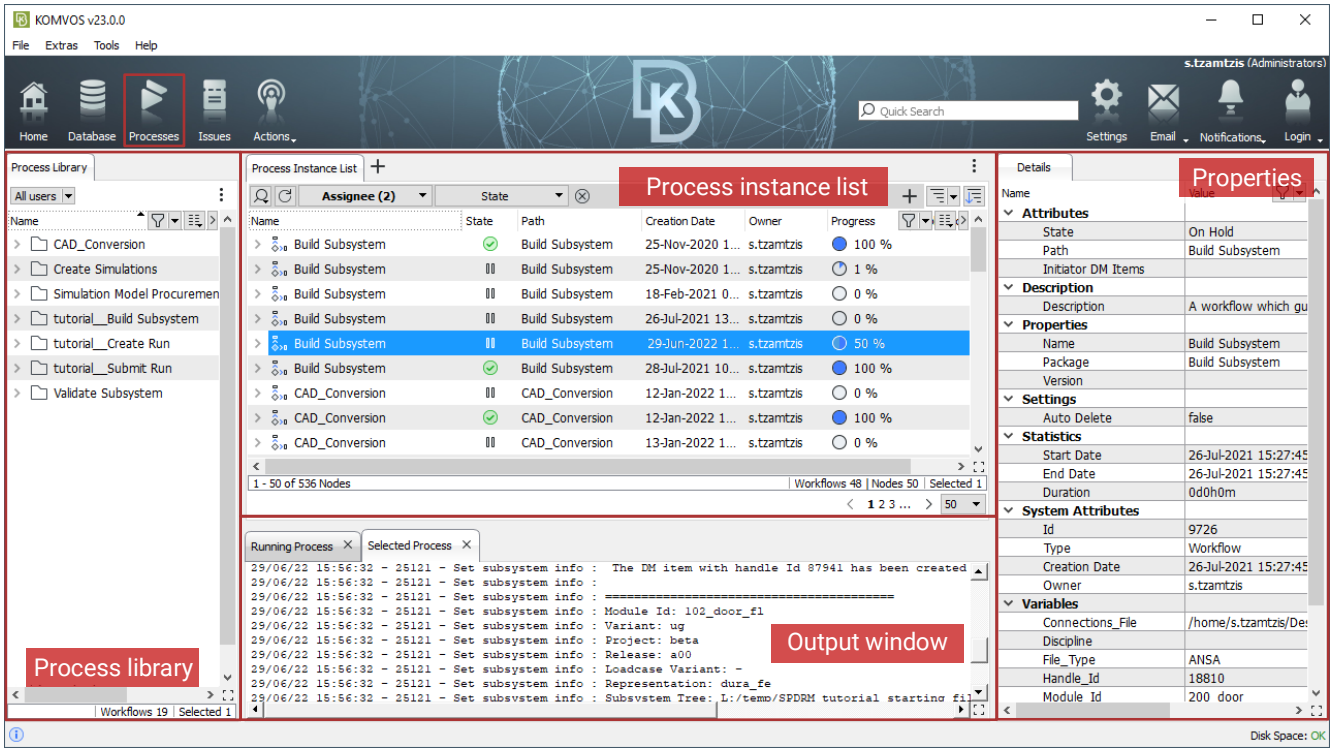
- On the left, the *Process Library* lists all the process templates that are stored in the SPDRM database.
- The middle section is split horizontally in two sections:
  - At the top, the *Process Instance List* and/or the *Process Diagram* tabs are available. The *Process Instance List* displays all the workflow instances that have been created in SPDRM and are visible to the logged in user, offering search and filtering capabilities for quick focus on the desired instances.

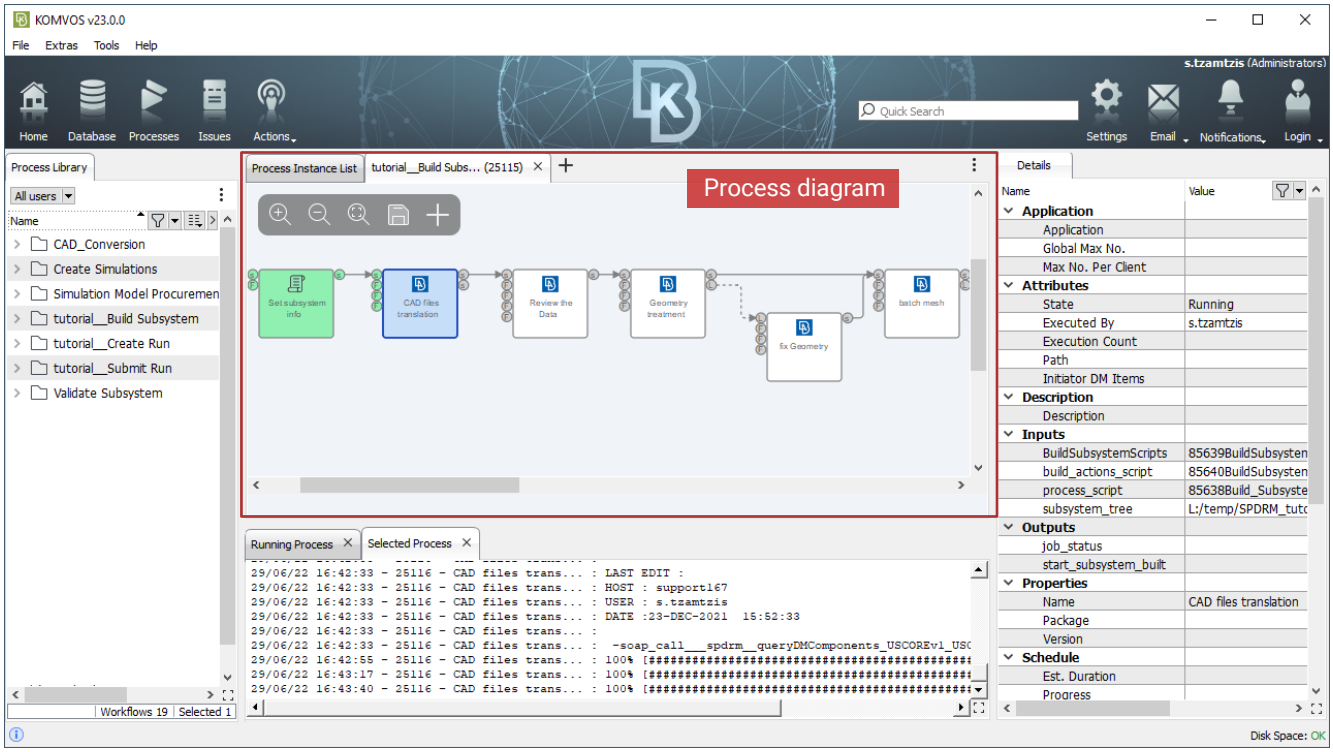
The *Process Diagram* tabs offer the visual representation of existing process definitions and instances, as well as process design functionality.

- At the bottom, the *Output* window displays the output of the *Running* or the *Selected Process*.

- On the right, the *Details* tab lists the properties of the entity (e.g. a process definition, an instance, a task, etc.) selected in the *Process* workspace.

For a detailed description of the *Process Management* functionality in KOMVOS, please refer to chapter 5 of this guide.

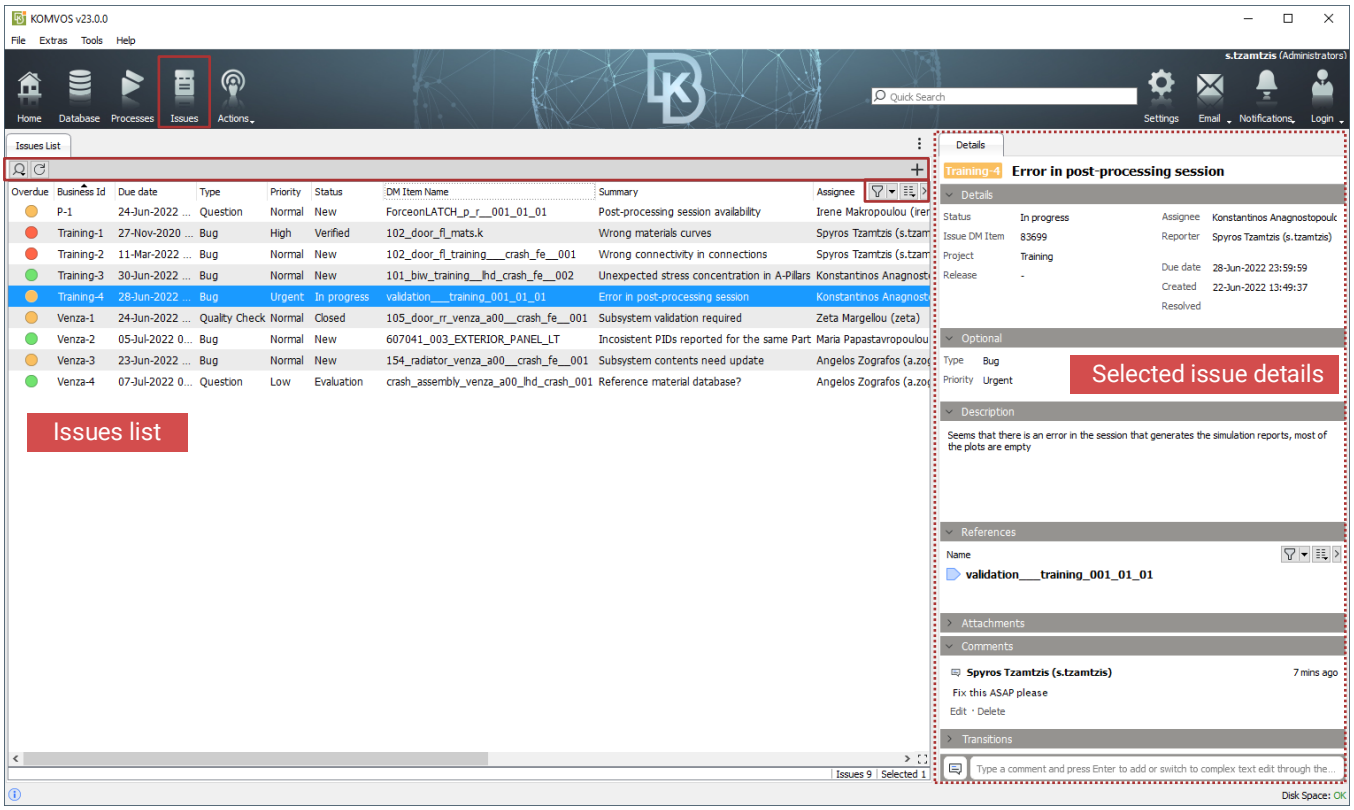




3.4. Issues

When connected to an SPDRM back-end of version 1.8.0 or later, the **Issues** workspace is activated. It is a workspace dedicated to the SPDRM *Issue Management*, a solution that provides CAE engineers and analysts with structured procedures for the processing and management of quality issues that concern model and simulation data.

The workspace facilitates the display and search/filtering of existing issues. Pressing on the **Issues** button in the toolbar, all issues reported in the SPDRM environment are listed in the workspace KOMVOS. The user can search for issues, filter the listed issues, control the visibility of attributes of the listed issues as columns and view the issues of interest in detail.



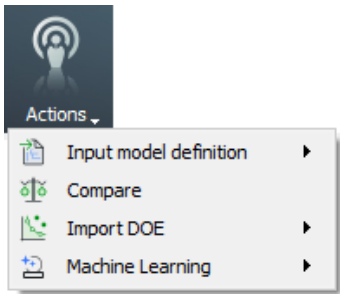
For a detailed description of the *Issue Management* functionality in KOMVOS, please refer to chapter 06 of this guide.

3.5. Actions

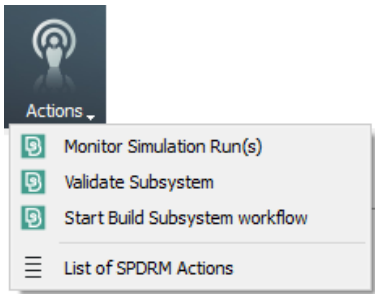
Using the **Actions** button in the toolbar, various general actions that do not apply to a specific DM object can be accessed. These actions are in essence scripts, whose execution can be triggered by selecting the respective action.

There are two categories of *Actions* in KOMVOS:

- Generic actions, available when connected to either a file-based SDM back-end or a server based SPDRM back-end. These are defined in the `dm_actions.xml` file that is stored in the configuration folder of KOMVOS.
- SPDRM actions, available only when connected to a server based SPDRM back-end. They can be distinguished from generic actions by the characteristic SPDRM icon. These are defined by an SPDRM administrator and are part of the system configuration.



Generic Actions



SPDRM Actions

For a detailed description of *Actions* in KOMVOS, please refer to chapter 07 of this guide.

4. Data Management

KOMVOS offers an overview of the data stored in the SDM repository, in an organized manner, providing the users with a series of functionalities for efficient data management. Through the **Database** workspace, it is possible to search for data, edit DM objects stored in the database, import new or export existing DM objects, explore the relationships between DM Objects, compare different versions and variants, create new iterations, review simulation run results, etc.

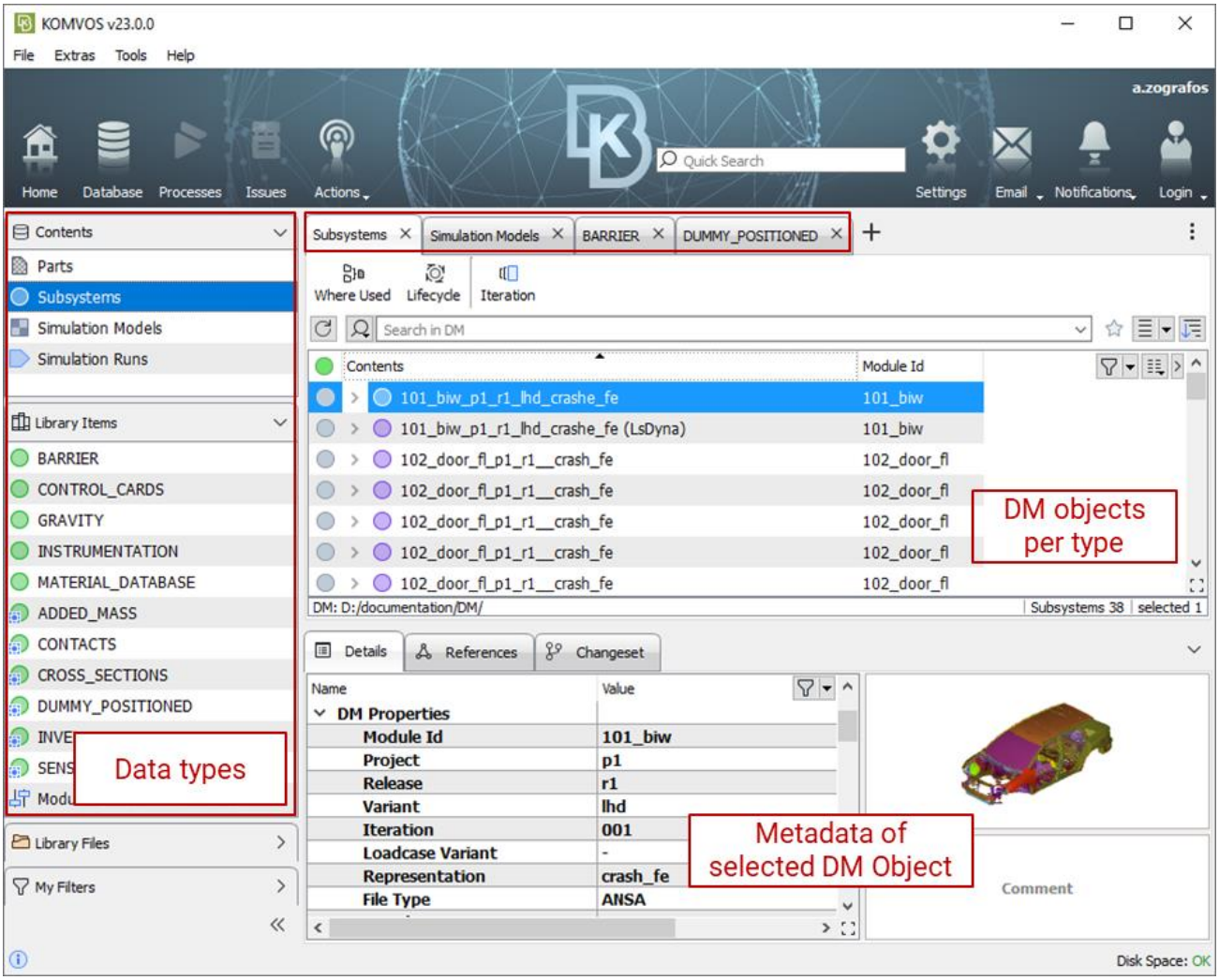
4.1. Data Browsing

Connecting to an SDM back-end, the contents of the database can be accessed by switching to the **Database** workspace. The user interface is split vertically in two main panels:

The left panel, which is collapsible, lists the datatypes of the DM Objects that are available in the SDM system. The DM contents are grouped in three major categories:

- **Contents**: Model and Simulation data types (e.g. Parts, Subsystems, Simulation Models, Simulation Runs)
- **Library items**: Library Item data types (e.g. Materials, Dummies)
- **Library files**: "Simple file in folder" library items (e.g. scripts, batch mesh sessions)

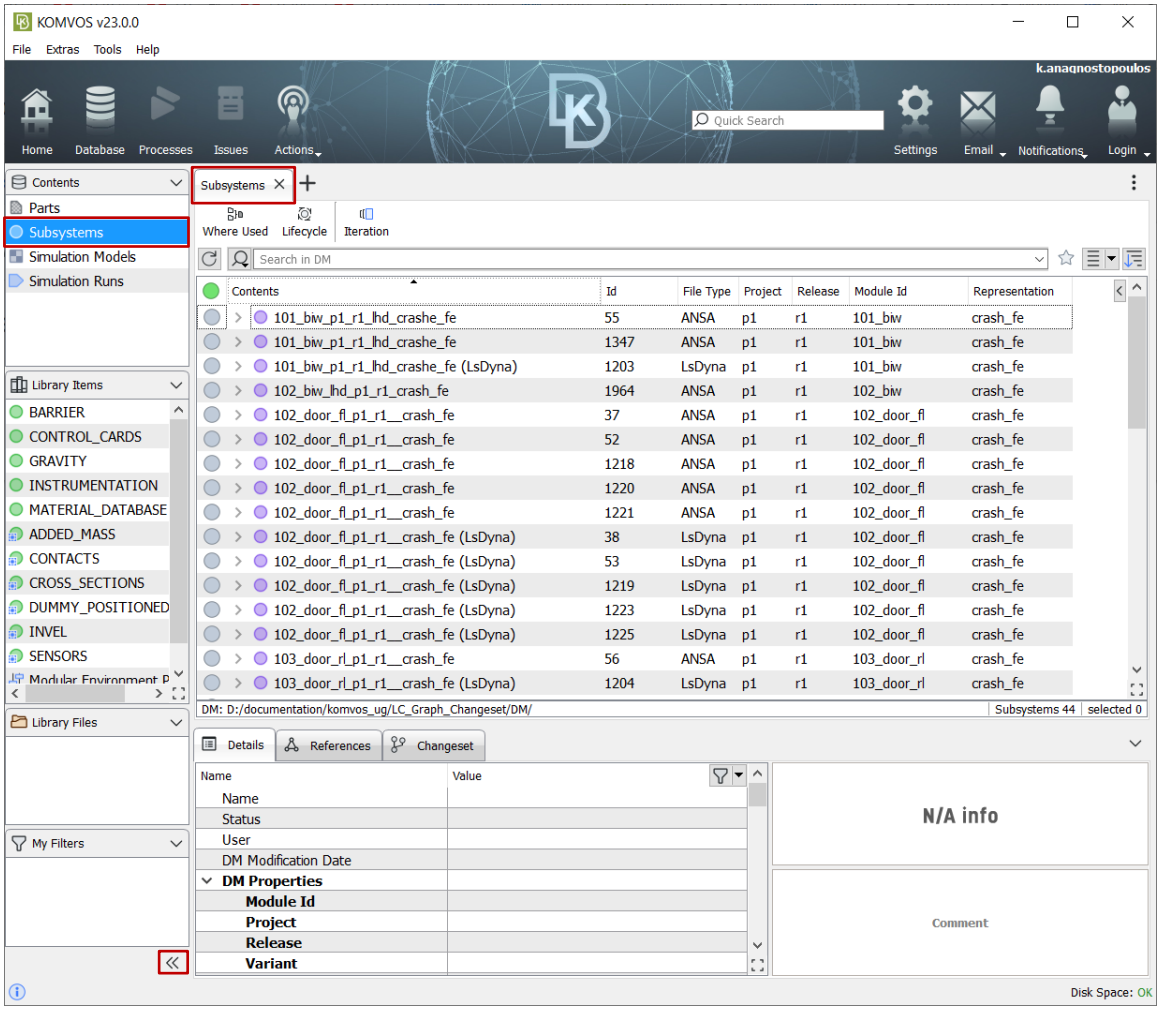
The right panel displays in different tabs the actual DM Objects per type.



4.1.1. Containers

The DM objects that are available in the SDM system are listed in the left panel of the workspace. This can be collapsed or expanded on demand, using the respective button, to arrange the user interface as desired.

To access the DM objects of each container, double-click on the respective listed data type and a new tab is opened in the right panel. Each tab displays the main list of DM Objects of a specific data type.



4.1.2. List

The DM object tabs in the main list of the *Database* browser offer functionality that enable users to browse, query and review the contents of the database.

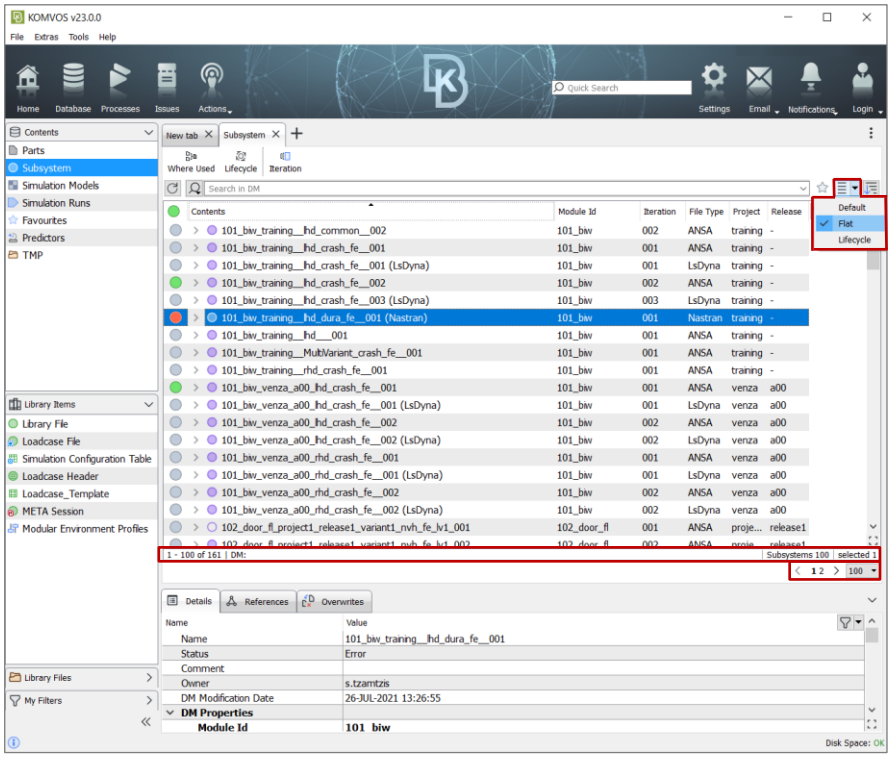
The structure and layout of the information displayed in the list is organized in different “views”. Each data type (e.g. Subsystems, Simulation Models, Simulation Runs, etc.) has its own views that are accessible through the **View Modes** button. It is also possible to either expand or collapse the containers or hierarchy of the selected DM Objects in the list. Any of the tab views can be bookmarked using the respective button, and are then accessed through the *Bookmarks* shortcut in the *Home* tab, as described in Chapter 3 of this guide.

More details on the *View Modes* can be found in paragraph 3.1.1 of the Data Management Reference Manual.

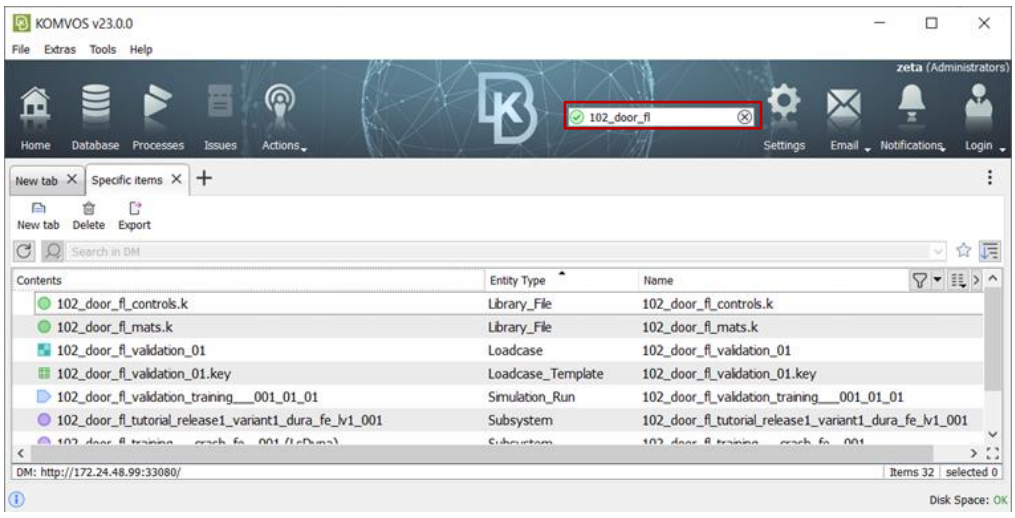
At the bottom of the list, additional information is displayed:

- The SDM repository
- The total number of DM Objects of the listed data type stored in the database
- The type and number of listed DM objects
- The number of entities selected in the list
- Indexing and navigation of the pages of available objects

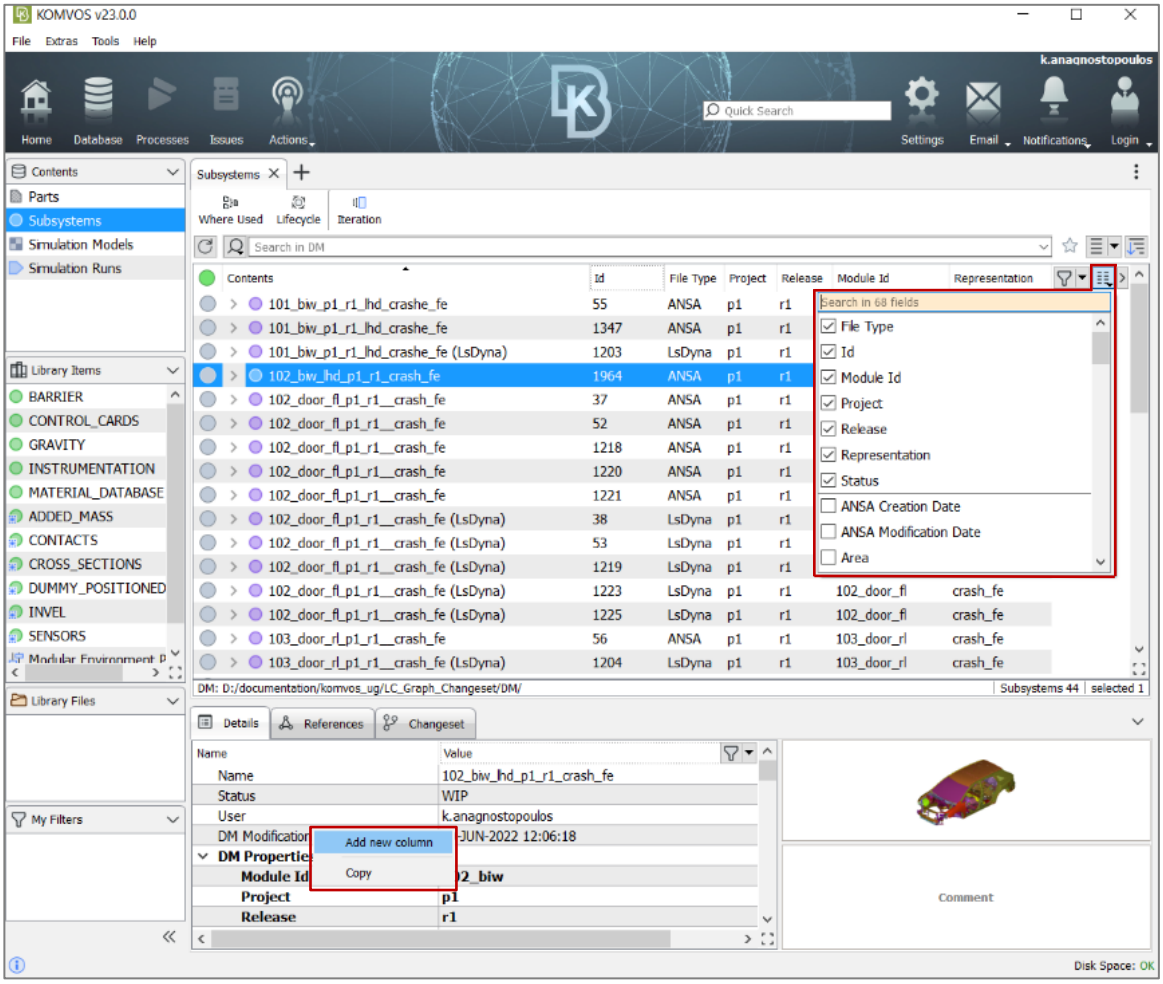
Apart from providing information for the total number of the listed and available objects, the user can control the number of listed objects per page, using the respective widget  . This information is stored in the user settings. Using the previous and next buttons, or selecting the displayed page numbers, navigation through the pages of the listed objects is possible.



The **Quick Search** field can be used to search in the database for objects that match the given info in their *Name, Module Id* or *Id*. For example, a keyword can be typed and by pressing Enter, the database is searched for existing objects that contain this keyword in their *Name* or *Module Id*. The results are listed in a new tab, as shown below.

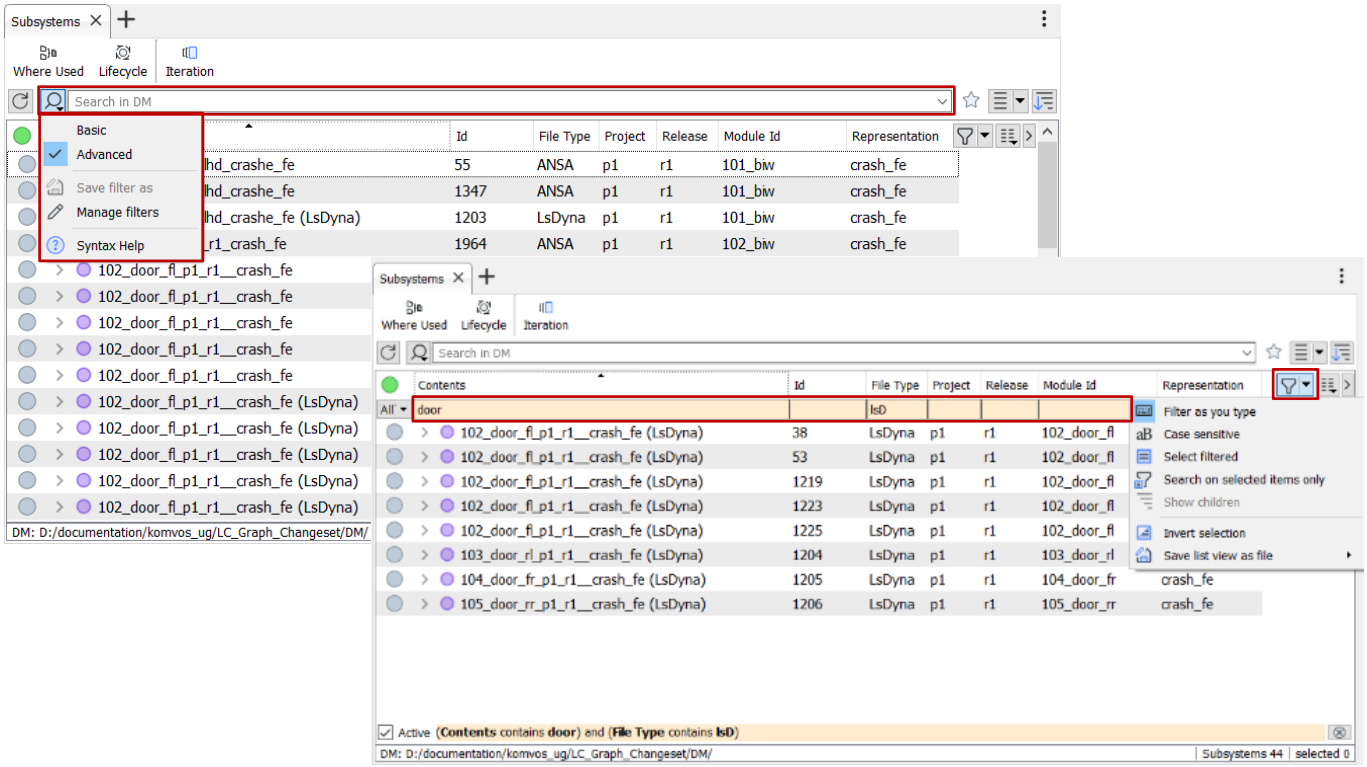


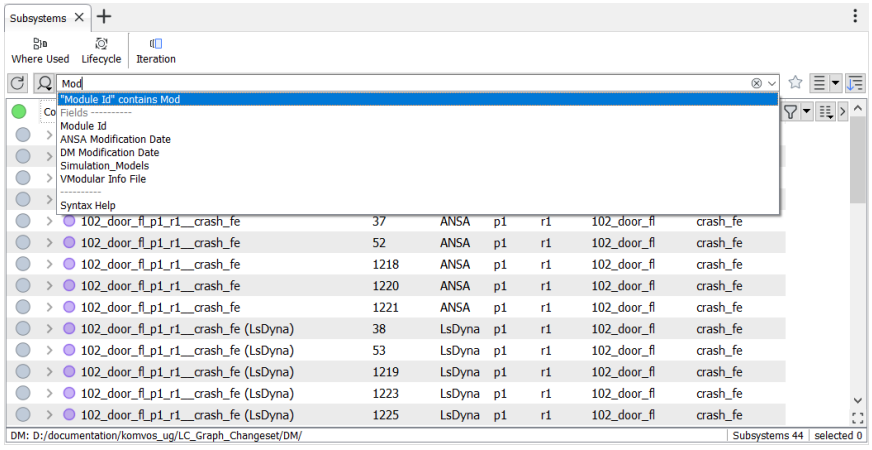
The list of the DM Objects can be considered as a dynamic configurable table. The displayed information can be enriched by adding columns of the available metadata. The user can control the visibility of the respective fields, by clicking on the **Show/Hide** columns button and selecting or deselecting the desired fields. The addition or removal of a column can also be controlled from the context menu of a field in the **Details** tab or from the context menu on the respective column header.



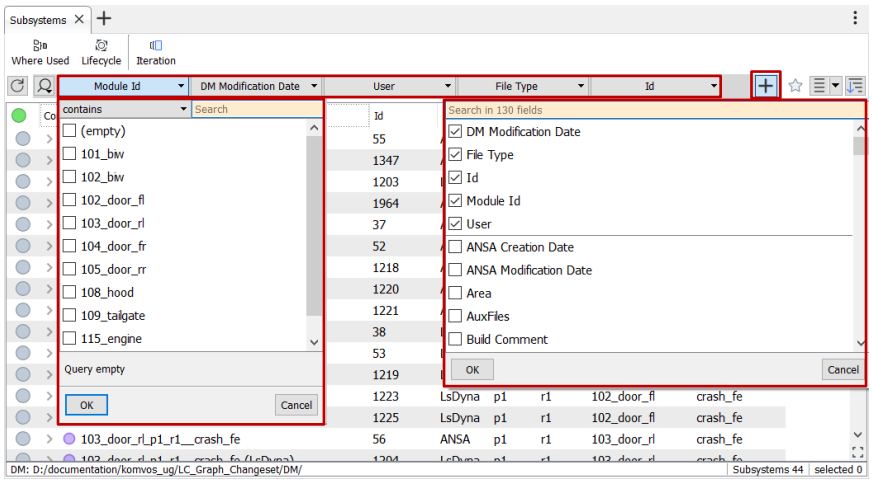
Searching in the database in KOMVOS is performed based on the metadata of the stored DM objects, using the BETA Query Language (BETA QL), which is common across all products of the BETA suite. BETA QL search supports two modes for the definition of queries in the **Search in DM** field: **Basic** and **Advanced**. Once a database query is performed and the matching results are listed, it is possible to restrict the number of the displayed DM Objects according to filtering values applied in the displayed columns. With the option **Show Filter** enabled, right below of each column's label, a filtering field is added. The settings for the filters can be set from the respective menu and there is also the capability to save the list that is resulted from the application of the various filters.

In chapter 3.5.2 of the Data Management Reference Manual performing queries using the BETA QL, as well as the use of filters are described in detail.

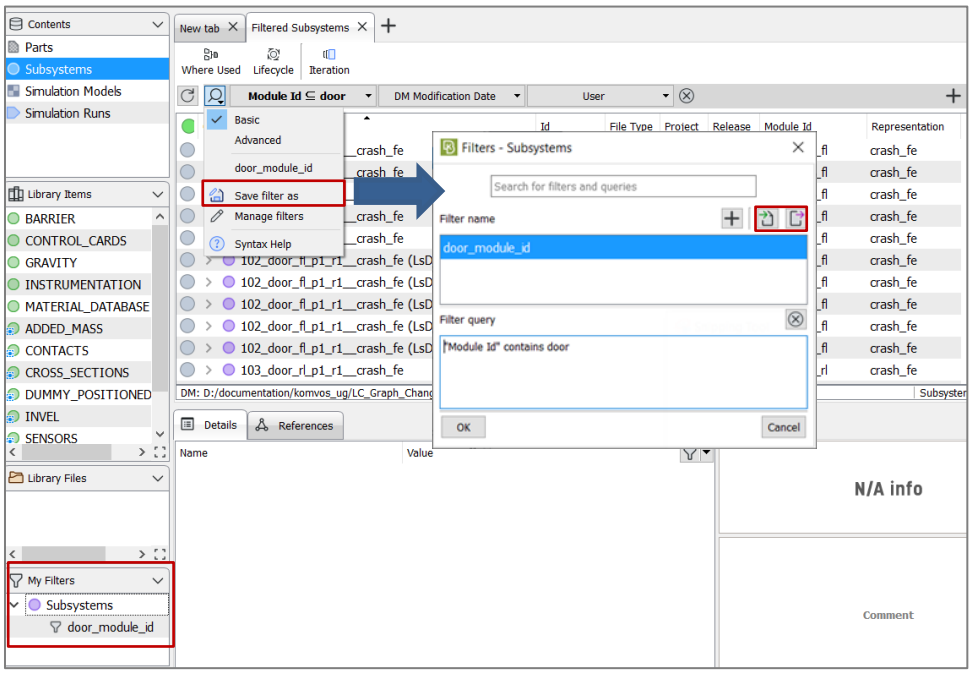




With the *Advanced* option selected, the user can dynamically define the filtering expressions. While typing, KOMVOS automatically suggests options that are matching the current expression. The user can be navigated through the suggestions using the keyboard arrow keys and press Enter to select one of them.



With the *Basic* filter option selected, each available field can be added as a different button. By pressing the respective button, a drop-down menu is opened, and the desired values can be checked and applied with the **OK** button. The combination of the added fields and the selected values for each one, result to the applied filtering expression.

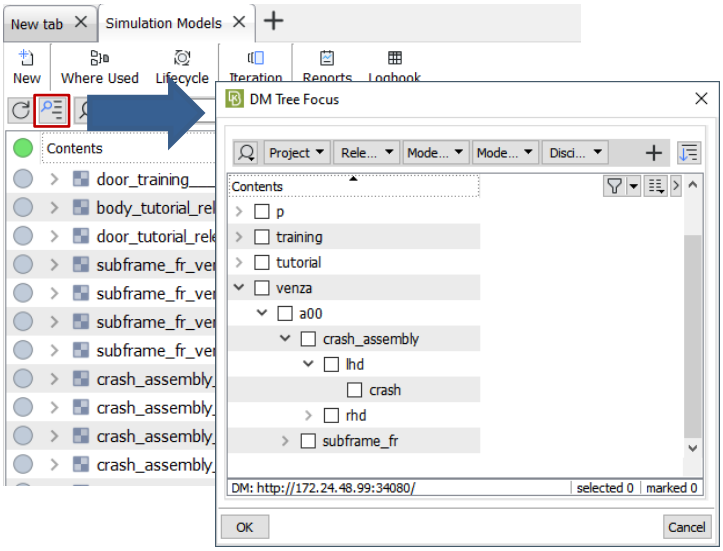


Defined filters can be saved and reused. Once defined, the **Save filter as** option is available in the drop-down menu. By selecting the option, the *Filters* window opens. The user can provide a name of the filter and in a separate area at the bottom, review the query expression. It is also possible to export the defined filters in xml format or input already existing ones.

Pressing the **OK** button, the defined filters are stored and will be listed in the *My Filters* area at the left panel of KOMVOS main window.

When connected to an SPDRM back-end , the **DM tree focus** is available in order to isolate and focus only on branches of the data structure that represent the current working projects. By pressing the respective button, the *DM Tree focus* window opens. The configuration of the DM tree focus is similar to the one described for the Basic filter.

A detailed description for the DM tree focus functionality can be found in paragraph 3.5.4 of the Data Management Reference Manual.



4.1.2.1. Tabs

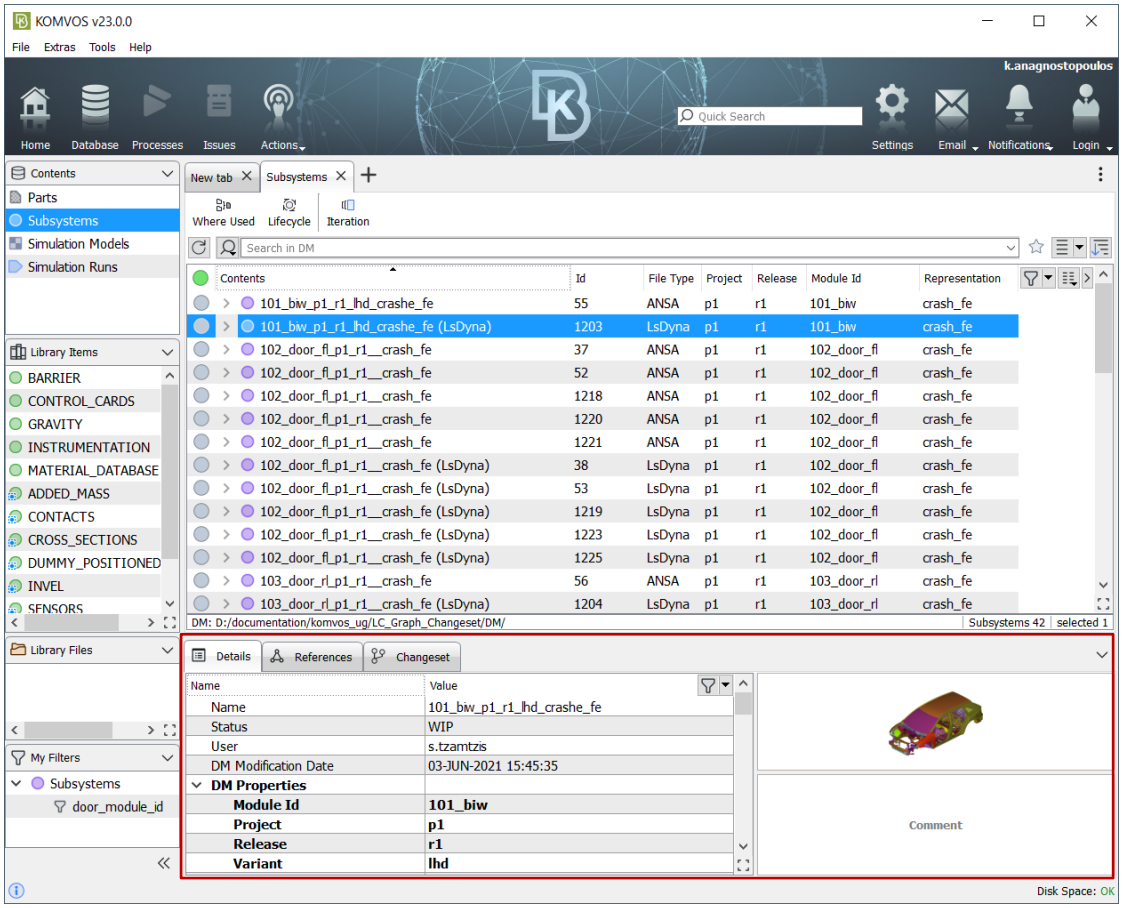
In each DM object tab in the *Database* workspace, the bottom section of the window displays the metadata for the selected DM Object. The information is organized in different sub-tabs, which vary depending on the SDM back-end.

In case of a file-based back-end, the following information tabs are available:

- The **Details** tab lists the primary, secondary and additional attributes of the selected item.
- The **References** tab lists the DM objects related to the selected items. The types of relationships are grouped in three categories (i.e. *Where Used, Lifecycle, Other Links*) which are displayed in separate sub-tabs.
- The **Changeset** tab lists all the recorded modifications of the selected Part/Subsystem that represent the difference from its predecessor.
- The **Charts** tab lists all Charts used when KOMVOS is used for DOE Studies.

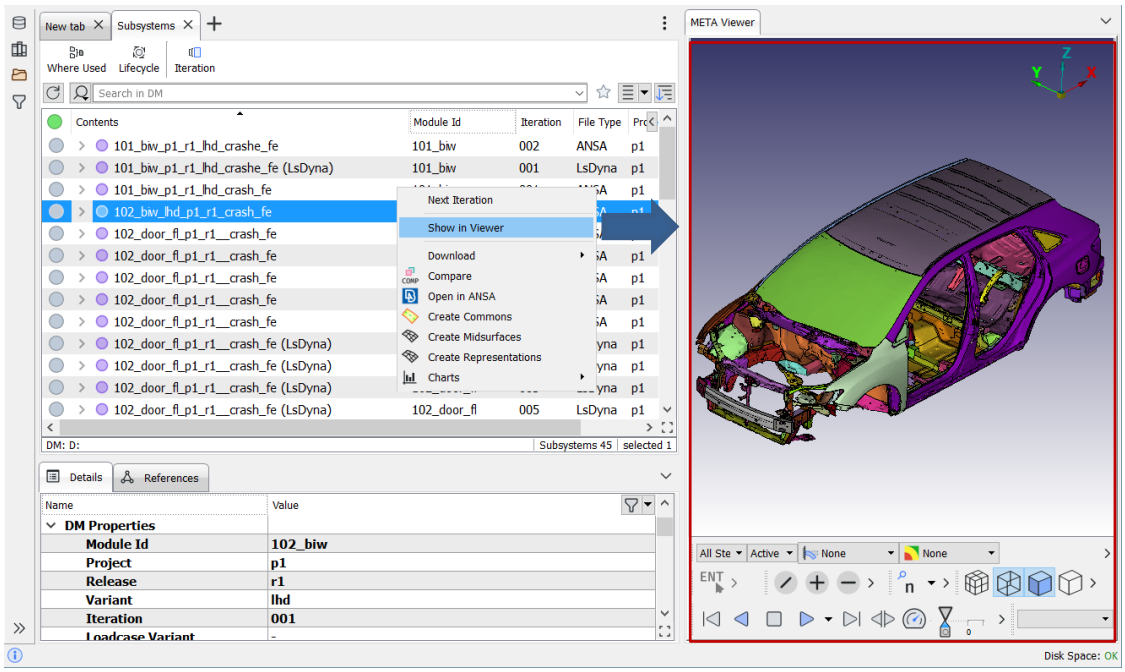
When KOMVOS is connected to an SPDRM back-end, in addition to the tabs described above for file-based DM, which are still available, the following tabs are offered:

- The **Issues** tab lists any SPDRM issues that are associated with the selected DM object. For more details, please refer to chapter 6 of this guide.
- The **Overwrites** tab lists a history of overwritten files. It is applicable only for plain files (e.g. plain "file in folder" Library items or files attached to DM objects)
- The **Updates** tab lists the newer versions stored in the database for the selected DM object
- The **Alerts** tab lists any alerts that have been generated in the system for the selected DM object. For more information, please refer to paragraph 4.2.2 of this guide.
- The **Job Status** tab lists all submitted jobs associated with the selected DM object. Available for DM Objects of Simulation Model type.

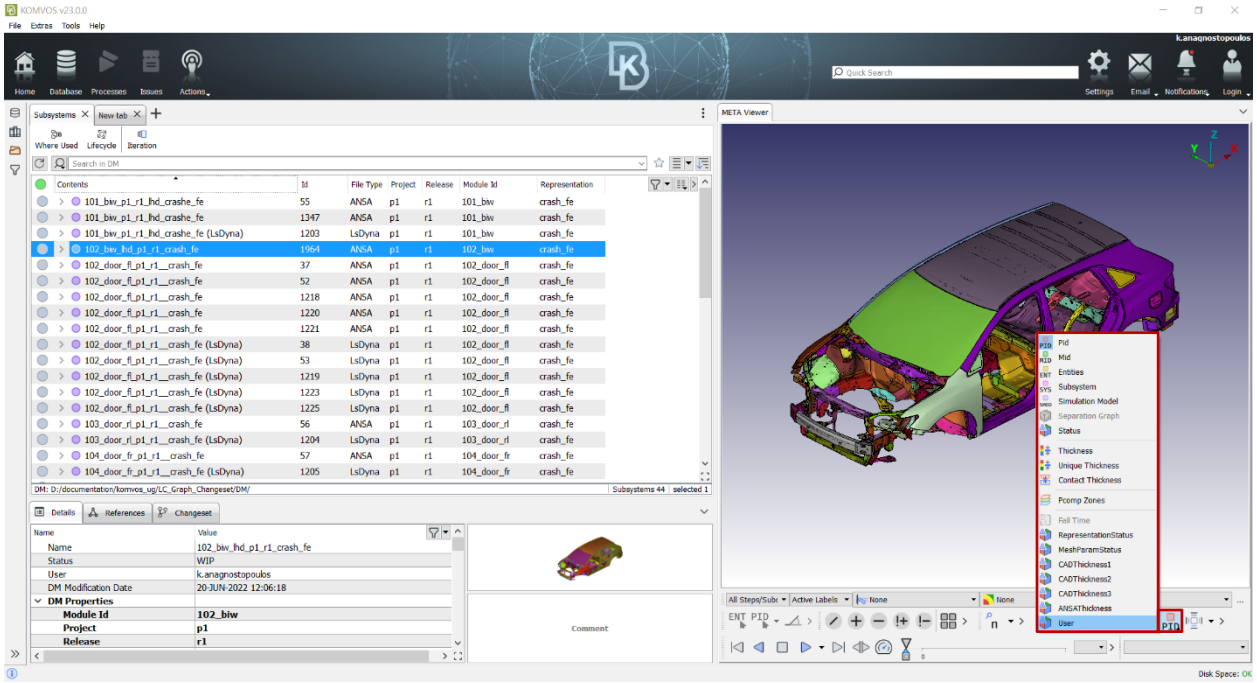


4.1.3. Viewer

The *Database* Workspace comes with an embedded META Viewer, allowing the visualization of models and post-processed results of all types. With the context menu option **Show in Viewer**, the embedded META Viewer is initialized in the right panel and the selected object is displayed. If the META Viewer is already initialized, the respective context menu options are **Show, Hide, Show only** and **Clear Viewer**.

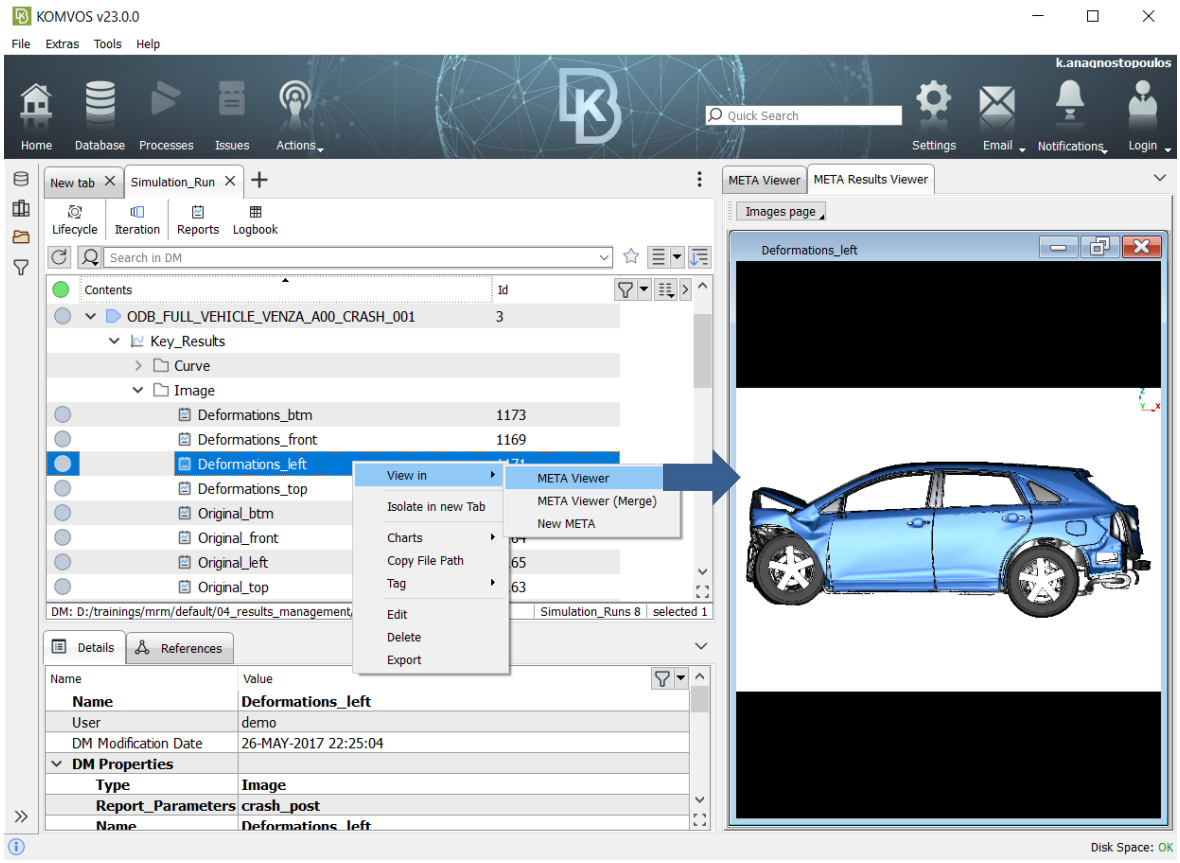


It is possible to view in the embedded META Viewer the Parts of a Subsystem colored according to the values of one or more of their attributes. These attributes can either be custom ones or already existing. A definition of a new Style in the **Styles** toolbar of the META Viewer is required. The `dm_views.xml` and `palette.xml` files are used to configure the colored values.



The procedure of configuring the colors of the parts is described in paragraph 5.2.1.8. "*Defining custom draw modes in the Viewer*" of the Data Management Reference Manual.

The META Viewer can also be used for the viewing of post-processed results. Selecting a Report DM object and triggering the context menu, the option **View in > META Viewer** can be used to display the selected Report in the embedded META Viewer.



For more information on the capabilities of viewing Simulation and Test Results, please refer to paragraphs 4.2 and 4.4 respectively, in the Data Management Reference Manual.

4.2. Data Browsing Tools

With model data and simulation results stored in the SDM database in an organized manner, KOMVOS offers functionality that enables engineers and analysts to perform various tasks for the effective management of their data:

- The fast and accurate comparison and matching of model variants can be achieved
- The evolution of CAE data and their relations and dependencies can easily be explored
- The commonalities between model variant can be identified

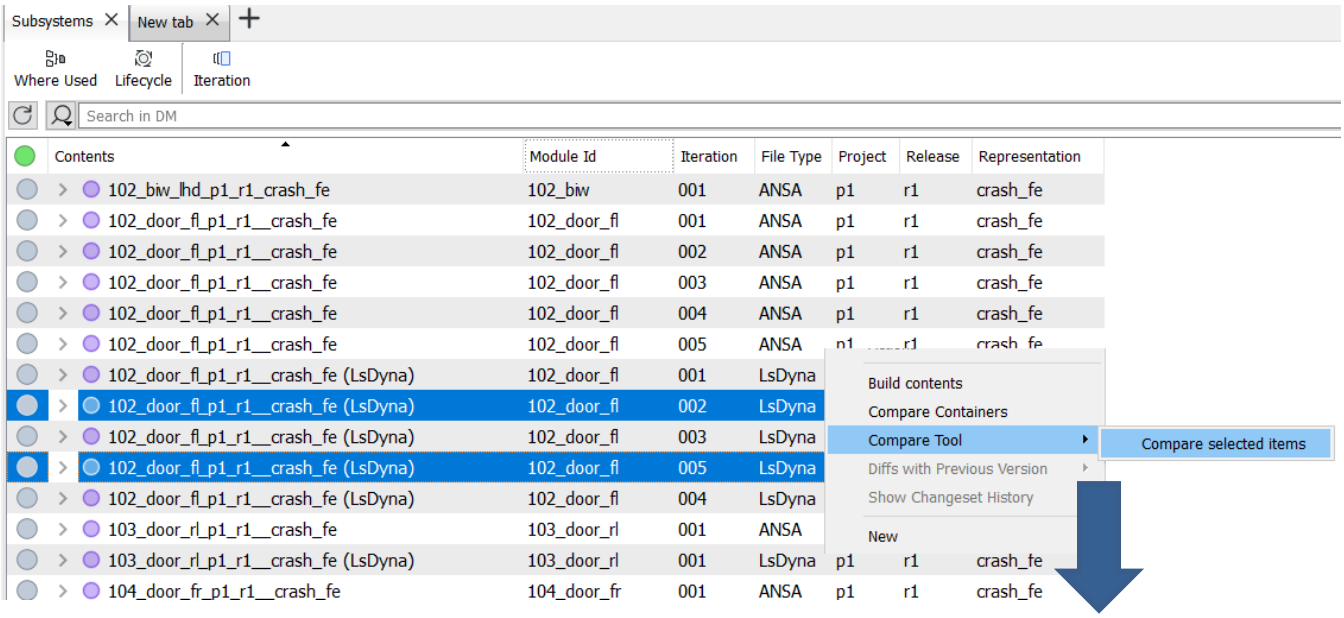
4.2.1. Comparing data

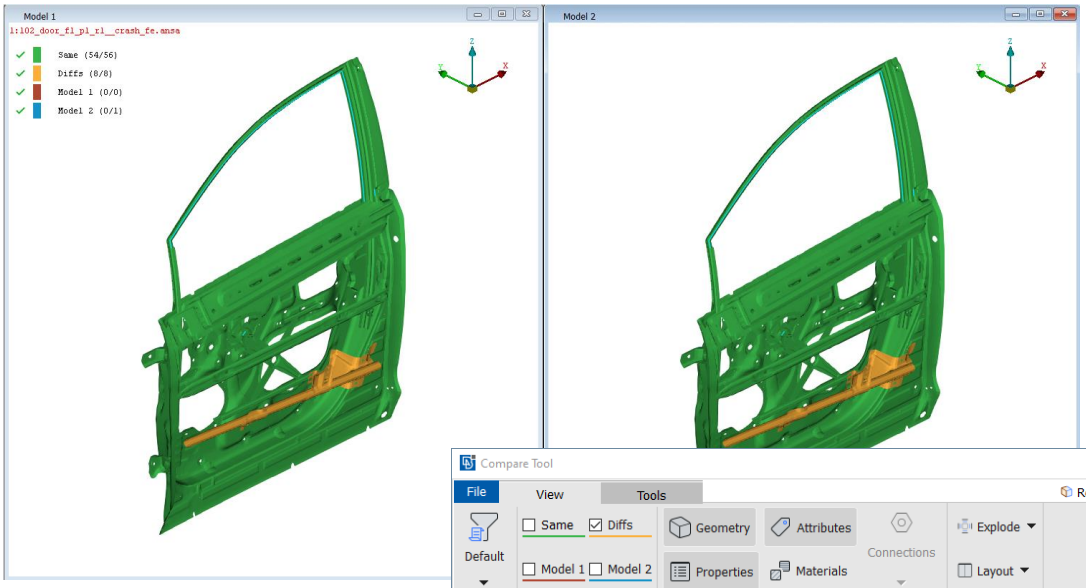
During the evolution of the design, several base modules are updated and stored in the data repository, which are then consumed in various model variants and simulations. It is important to be able to identify the modifications between the different base module versions and the model variants prepared by the various teams, in order to eventually understand their impact on simulation results.

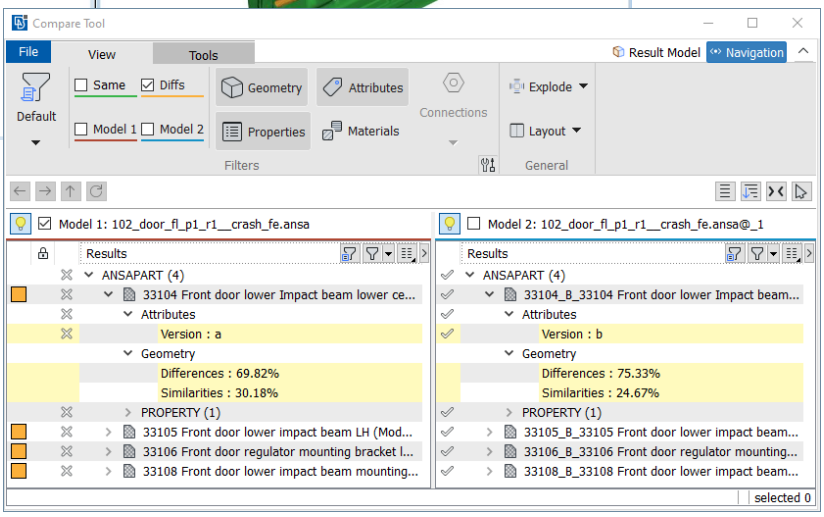
KOMVOS offers powerful functionality that enables users to compare both base modules and compound model containers, either in terms of their geometry and engineering attributes, or in terms of their metadata and contents.

4.2.1.1. Comparing Base Modules

ANSA Compare Tool can be triggered directly from within KOMVOS. In order to make comparisons between two Base Modules (e.g. two different versions of a Subsystem), the two can be selected in the respective tab of the *Database* workspace and through the context menu, the **Compare Tool > Compare selected items** action can be triggered. A detailed description of the Compare Tool is available in Chapter 19 of the ANSA User's Guide.

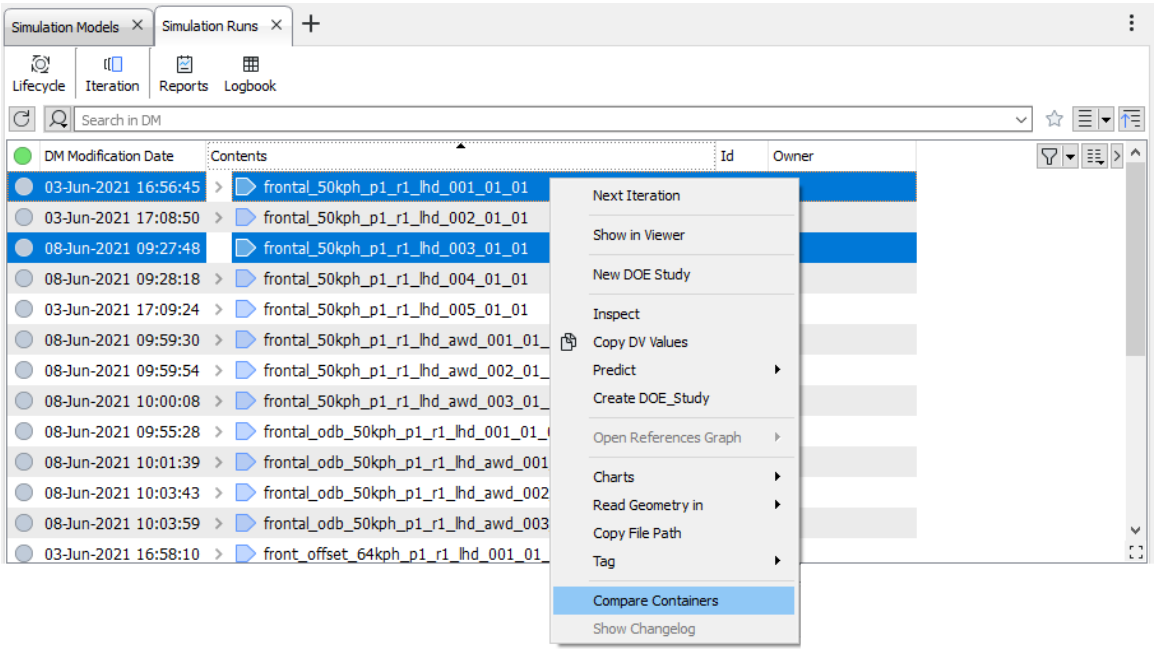




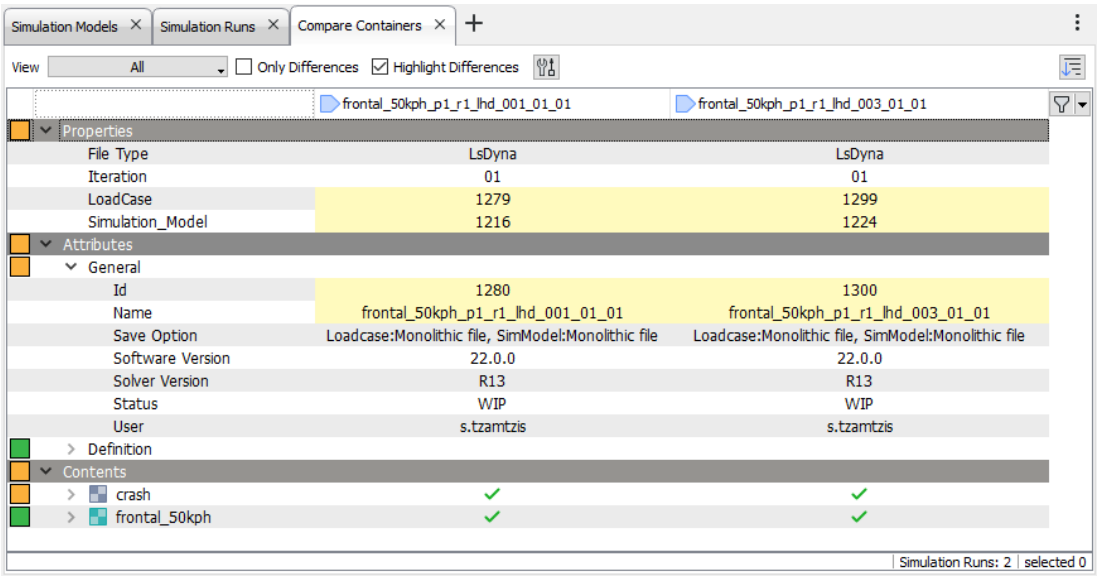


4.2.1.2. Comparing Compound Containers

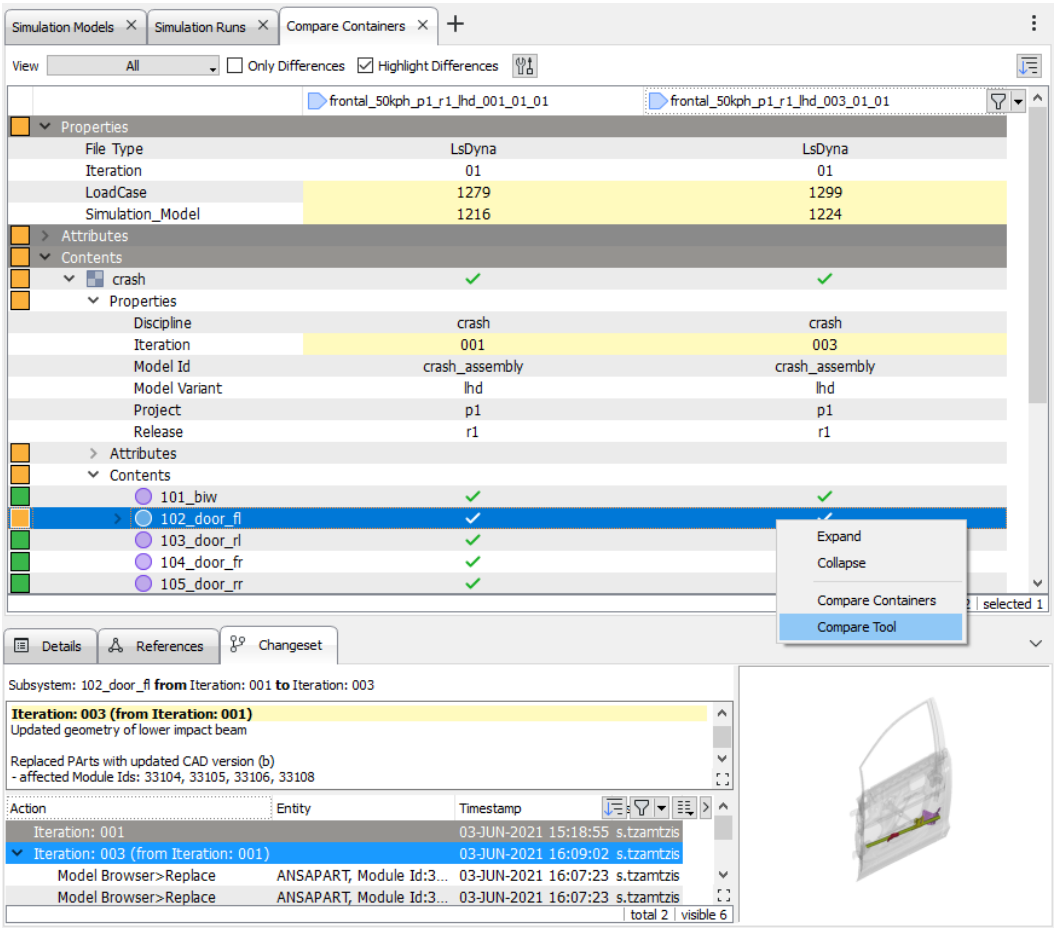
The comparison between two objects is not limited to Base Modules. It is also possible to select two Compound Containers (i.e. Subsystem Groups, Simulation Models, Loadcases and Simulation Runs) and use the context menu option **Compare containers** to trigger a comparison between them.



A new tab labeled "*Compare Containers*" is opened. Each container is added to a different column and their attributes and contents are matched per line. Based on the comparison results, a standard palette of colors is used to classify them.

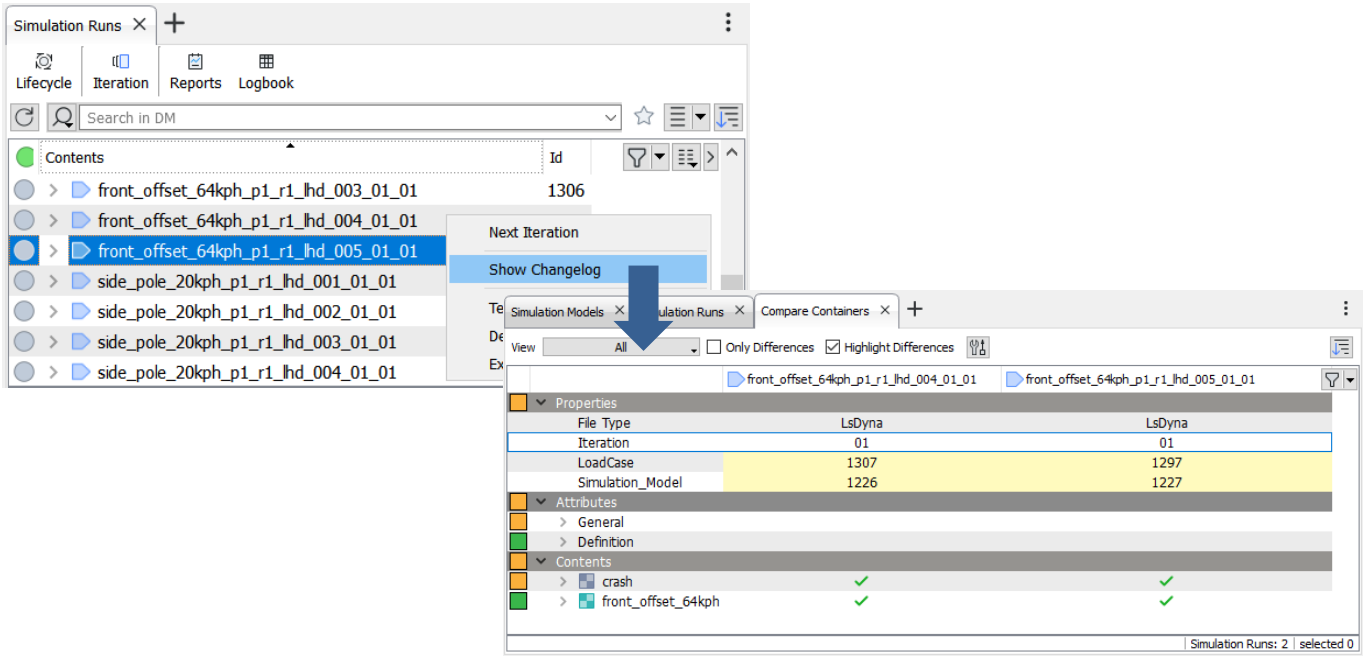


The comparison can proceed to a deeper level, into the contents of the selected containers, by expanding the respective listed entities. Once the comparison has reached the Subsystem level, it is possible to select any two listed items with differences and in the *Changeset* tab, review their complete *Changeset History*. Moreover, it is possible to perform a geometric comparison of the two subsystems in ANSA, invoking the ANSA Compare Tool.



The *Changeset* functionality is described meticulously in paragraph 3.9.2 of the Data Management Reference Manual, while a detailed description of the *Compare Containers* tool is offered in paragraph 3.10.2.

In terms of lifecycle, it is possible in KOMVOS to compare any version of a Compound Container with its parent. This action can be triggered from the context menu of the selected object with the **Show Changelog** option. A new Compare Containers tab is opened, and the comparison results are presented.

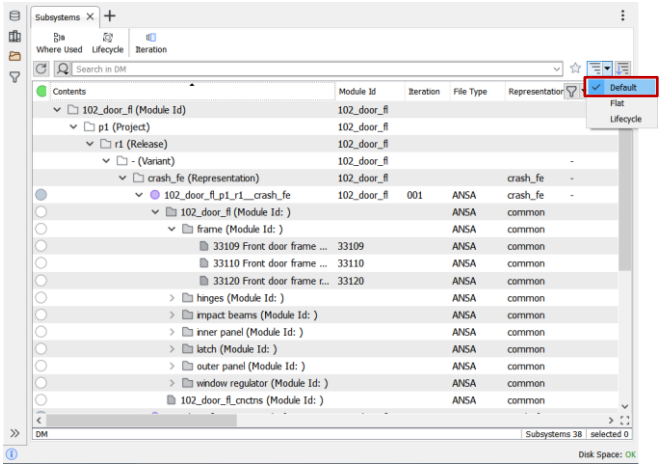


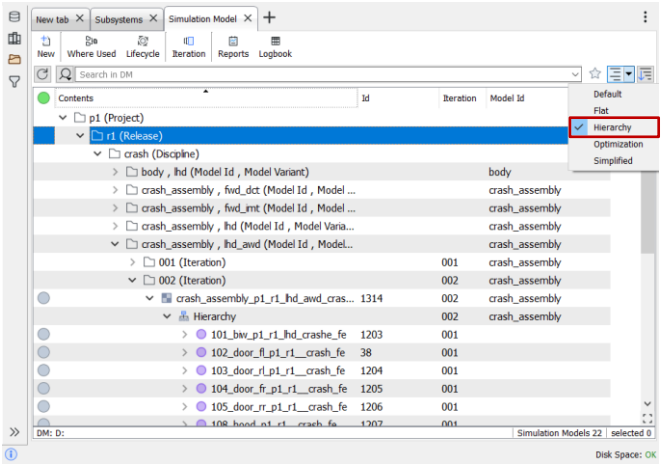
4.2.2. Exploring data relations

A key feature of KOMVOS is the easy, in-depth traceability of data relations. Reference links that are created between data objects as these are stored in the database, allowing the user to explore the relationships in terms of contents, hierarchy, usability and lifecycle. The contents and hierarchy of a data object are displayed in dedicated views in the database browser, while the references of a data object are listed in the **References** info tab and are also presented graphically in the **References Graphs**.

4.2.2.1. Contents and hierarchy

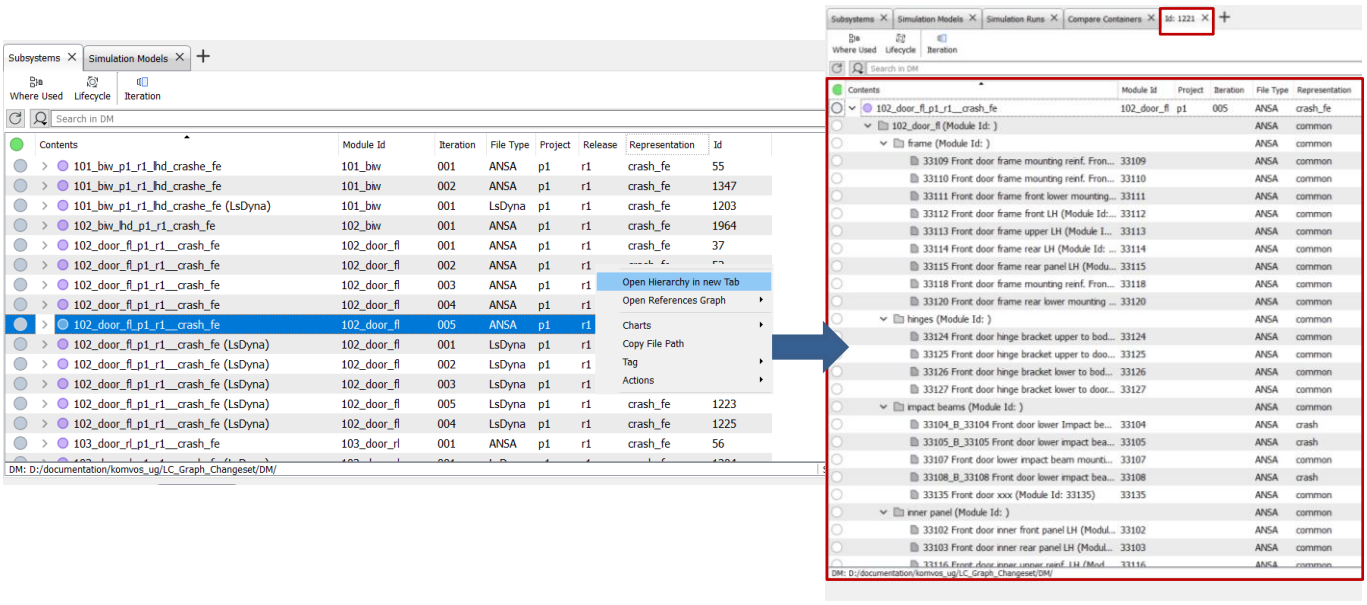
KOMVOS provides the capability for the user to examine the contents of a DM object. The Parts hierarchy in case of Subsystems and the contents of compound Model Containers (i.e. Simulation Models, Loadcases, Simulation Runs) can be displayed in the view modes labeled *Default* and *Hierarchy* respectively. When switching to these views for the specific object type, expanding the list view item of the respective DM object will list its contents in place.
Regardless of the DM Object type, the hierarchy can be viewed from a dedicated context menu option labeled **Open**

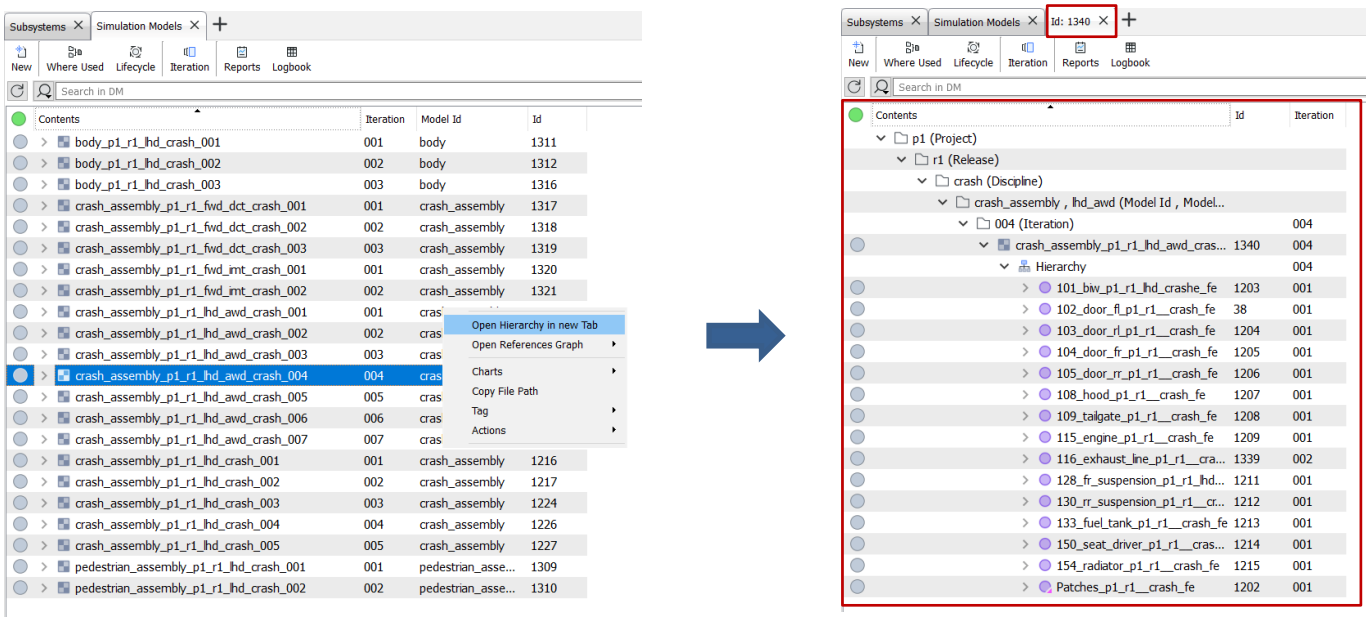




Hierarchy in new Tab. A new tab is opened in the database workspace and the contents of the object that are displayed vary depending on the object type:

- Subsystems show their Parts Structure
- Subsystem Groups show their Subsystem and Parts Structure
- Simulation Models show their contained Subsystems
- Simulation Runs show their contained Simulation Model and LoadCase

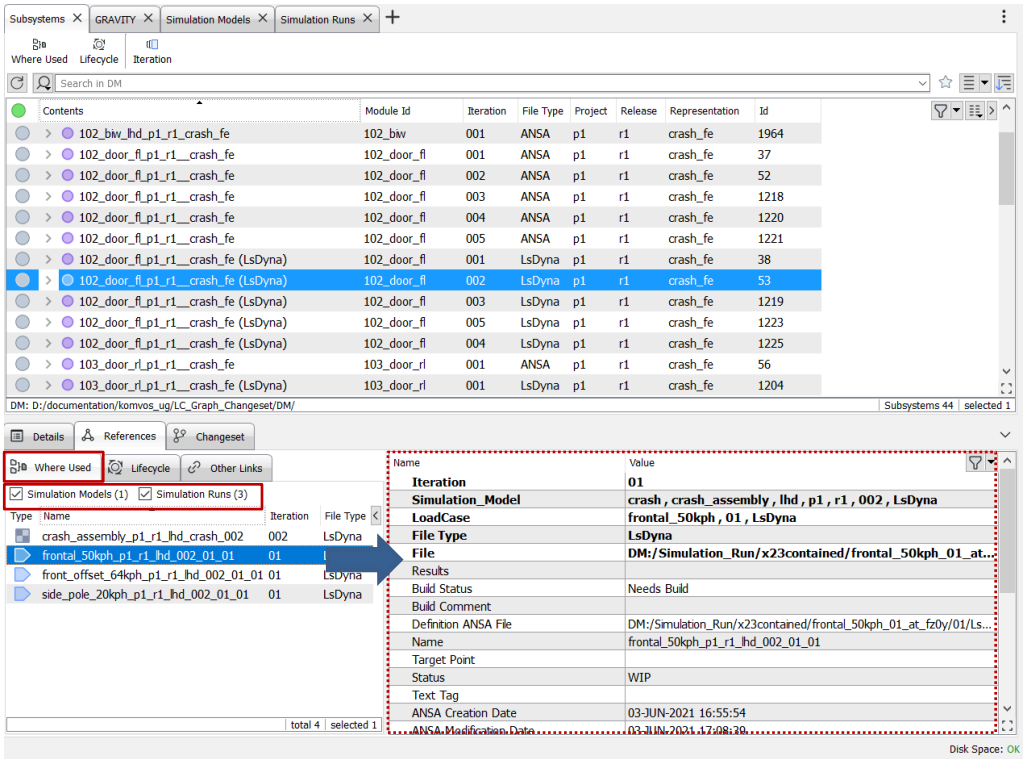




The views for the different DM object lists can be configured in order to display the data in the SDM client in the right context. For example, any of the available views can be adapted in order to enable the listing of the hierarchy/contents when the DM object container is expanded (e.g. the Subsystem *Flat* view). Similarly, it is possible to modify the existing *Default* and *Hierarchy* views, in order to change the grouping levels that are displayed (i.e. the attribute-based grouping). For more information on customizing the database browser views, refer to Chapter 5 of the Data Management Reference Manual.

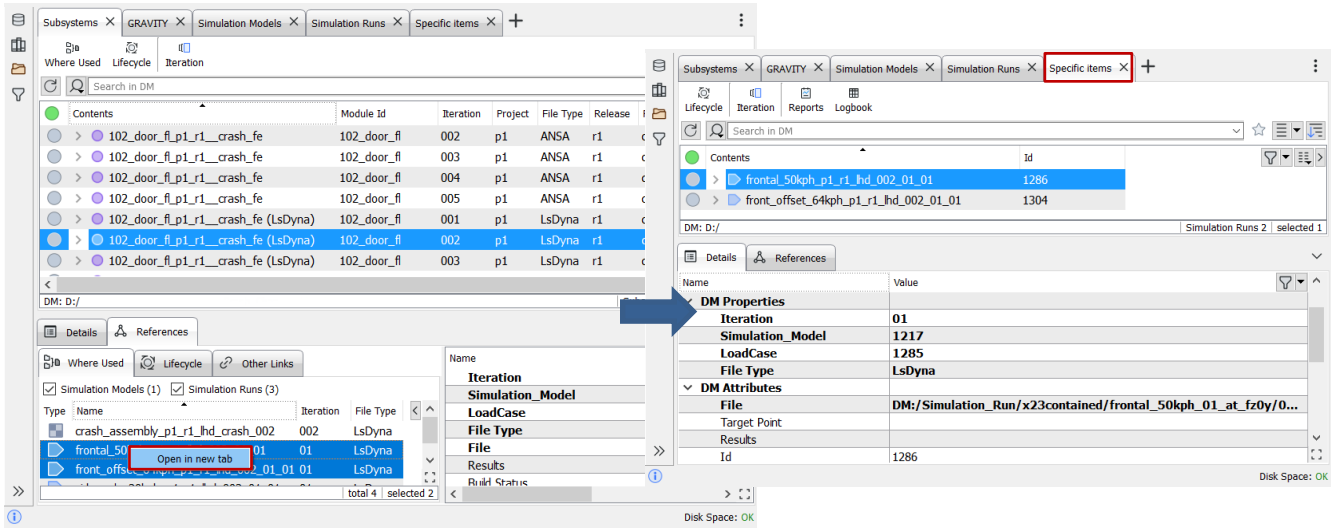
4.2.2.2. "Where-used" relations

All relations of data using other data, can be explored either in a list or graphically. In the *Where Used* tab, the DM objects that are using the selected object are listed (e.g. the Subsystems, Simulation Models and Simulation Runs using the selected Part). The user can control which entities are shown through the respective DM object type checkboxes. Next to each DM object type checkbox label, the total number of listed references is displayed.

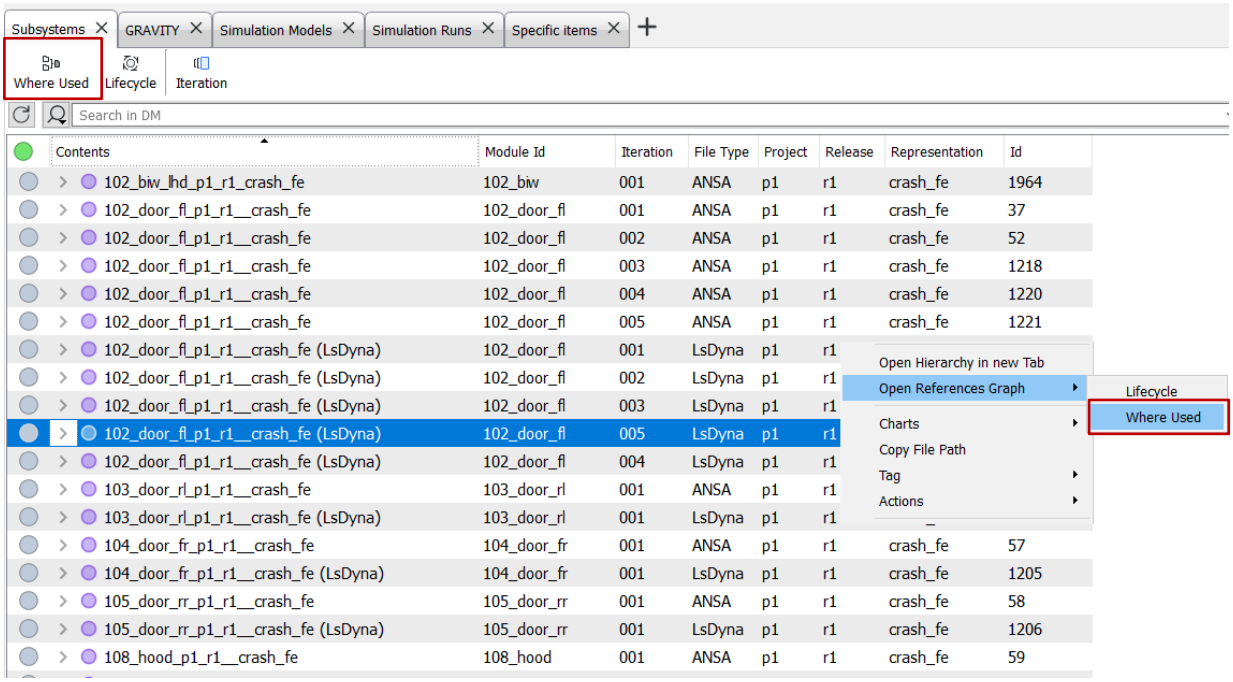


By selecting a listed reference, its' metadata are displayed in a separate section on the right.

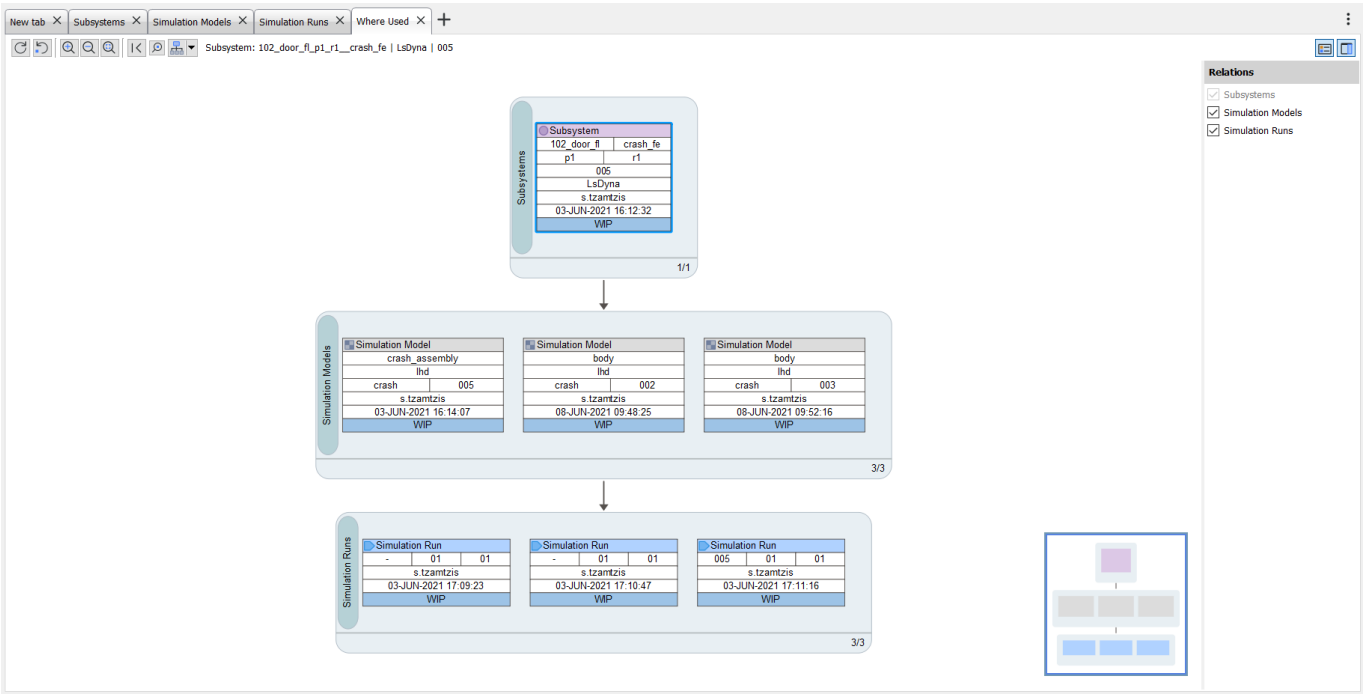
It is possible from the Where Used tab, to select an arbitrary number of the listed entities and to isolate them in a new separate tab. The **Open in a new tab** context menu option is available, under the condition that the selected items in the Where Used tab are of the same type. When this option is triggered, a new separate tab labeled **Specific items** is opened. The previously selected items in the Where Used tab are listed, allowing the user to make any modifications or actions on these items.



A graphical representation of the “Where-Used” relations is possible through the **Where Used** graph. The graph can be invoked either through the context menu option **Open References Graph > Where Used**, or by pressing the **Where Used** button on the toolbar, for the selected item.



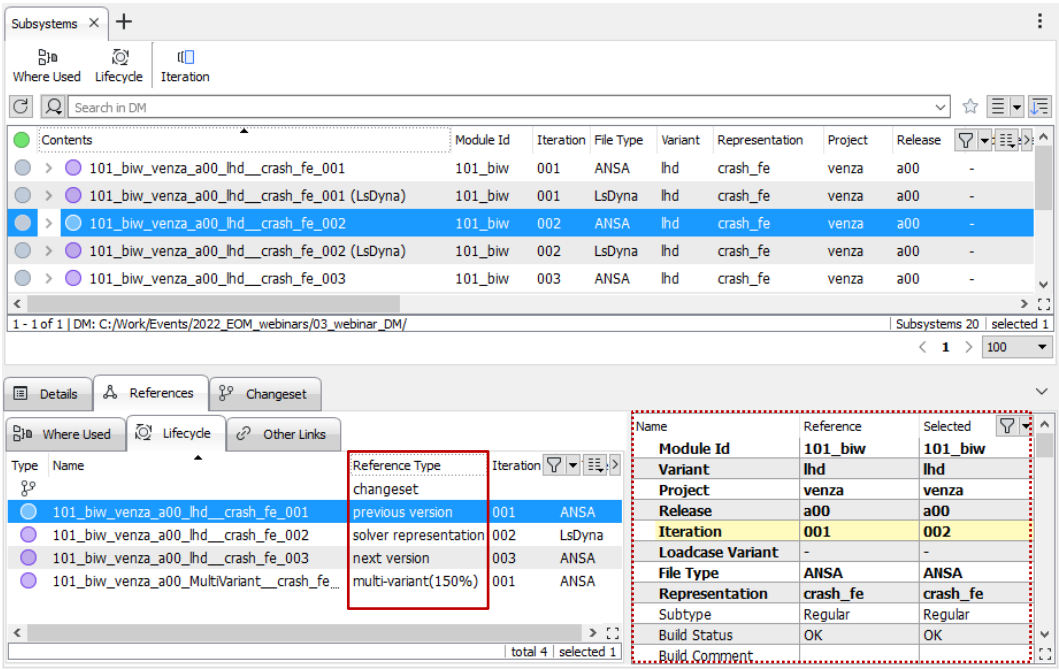
A new tab with the *Where Used* graph is opened in the *Database* workspace. All the relations between the selected item and the containers that use it, are displayed in a tree view graph. The detailed description for the *Where Used* graph can be found in paragraph 3.8.2.1 of the Data Management Reference Manual.



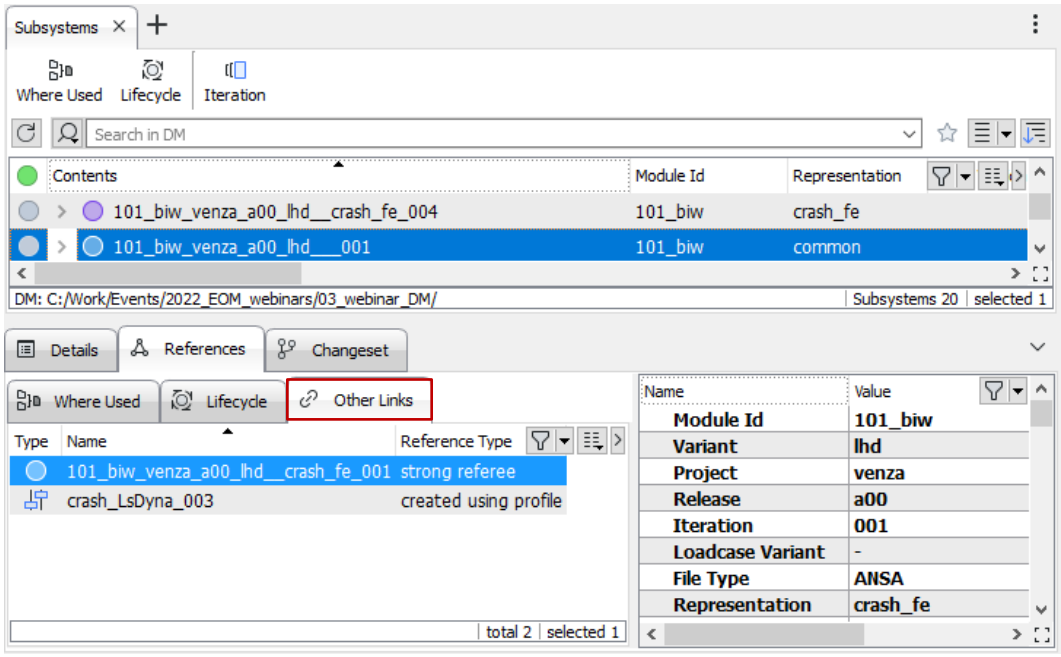
4.2.2.3. Lifecycle relations

Information about the evolution of an object in the repository is stored along with the object. The **Lifecycle** tab lists any data objects that are related with the item selected in the main tab of the database browser with links captured during the course of the object's lifecycle. These may include previous or next versions, alternative representations or variants, links with changeset objects, etc.

Selecting a DM object in the database browser, the user can access its existing lifecycle relations in the **Lifecycle** tab. The *Reference Type* column displays the type of lifecycle link between the selected DM object and its references. Selecting any of the listed references, its metadata are listed in the section on the right and any differences between the DM object selected in the main list and its reference will be highlighted.

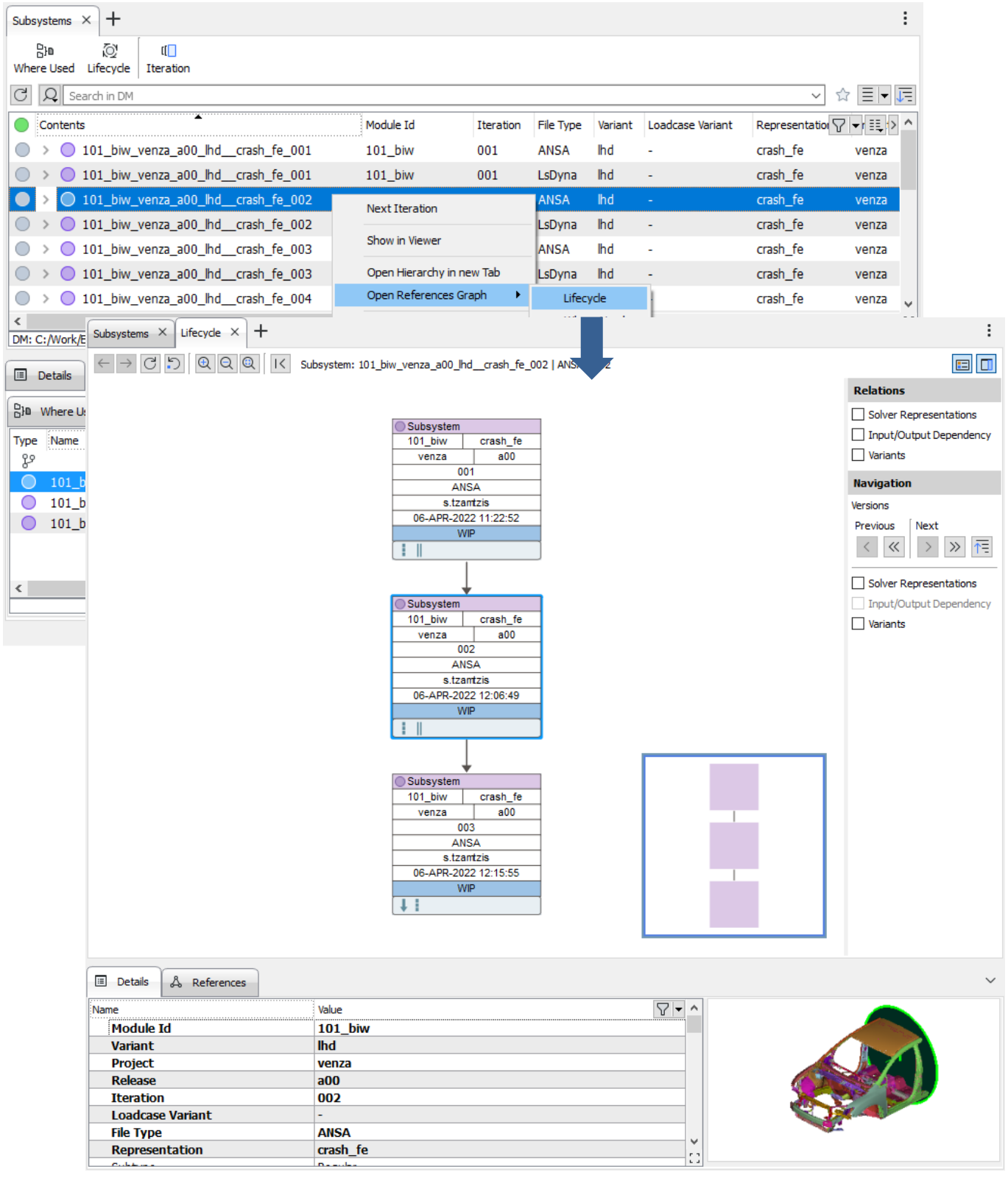


Some additional lifecycle relations supported by the SDM back-end are listed in the **Other Links** tab.



A detailed description of the lifecycle relations is given in paragraph 3.8.2.2 of the Data Management Reference Manual.

A graphical representation of lifecycle references is also offered in KOMVOS. This can be accessed by selecting any model container in the database browser and using the **Lifecycle** button in the toolbar or the **Open References Graph > Lifecycle** context menu option.


Subsystems
Where Used
Lifecycle
Iteration
Search in DM
Contents
Module Id
Iteration
File Type
Variant
Loadcase Variant
Representation
101_biw_venza_a00_lhd__crash_fe_001
101_biw_venza_a00_lhd__crash_fe_002
101_biw_venza_a00_lhd__crash_fe_003
101_biw_venza_a00_lhd__crash_fe_004
Next Iteration
Show in Viewer
Open Hierarchy in new Tab
Open References Graph
Lifecycle
Subsystems
Lifecycle
Details
Where Used
Type
Name
Subsystem: 101_biw_venza_a00_lhd__crash_fe_002 | ANSA
Subsystem
101_biw
crash_fe
venza
a00
001
002
003
ANSA
s.tzamtzis
06-APR-2022 11:22:52
06-APR-2022 12:06:49
06-APR-2022 12:15:55
WIP
Relations
Solver Representations
Input/Output Dependency
Variants
Navigation
Versions
Previous
Next
Details
References
Name
Value
Module Id
101_biw
Variant
lhd
Project
venza
Release
a00
Iteration
002
Loadcase Variant
-
File Type
ANSA
Representation
crash_fe

4.2.2.4. Data alerts

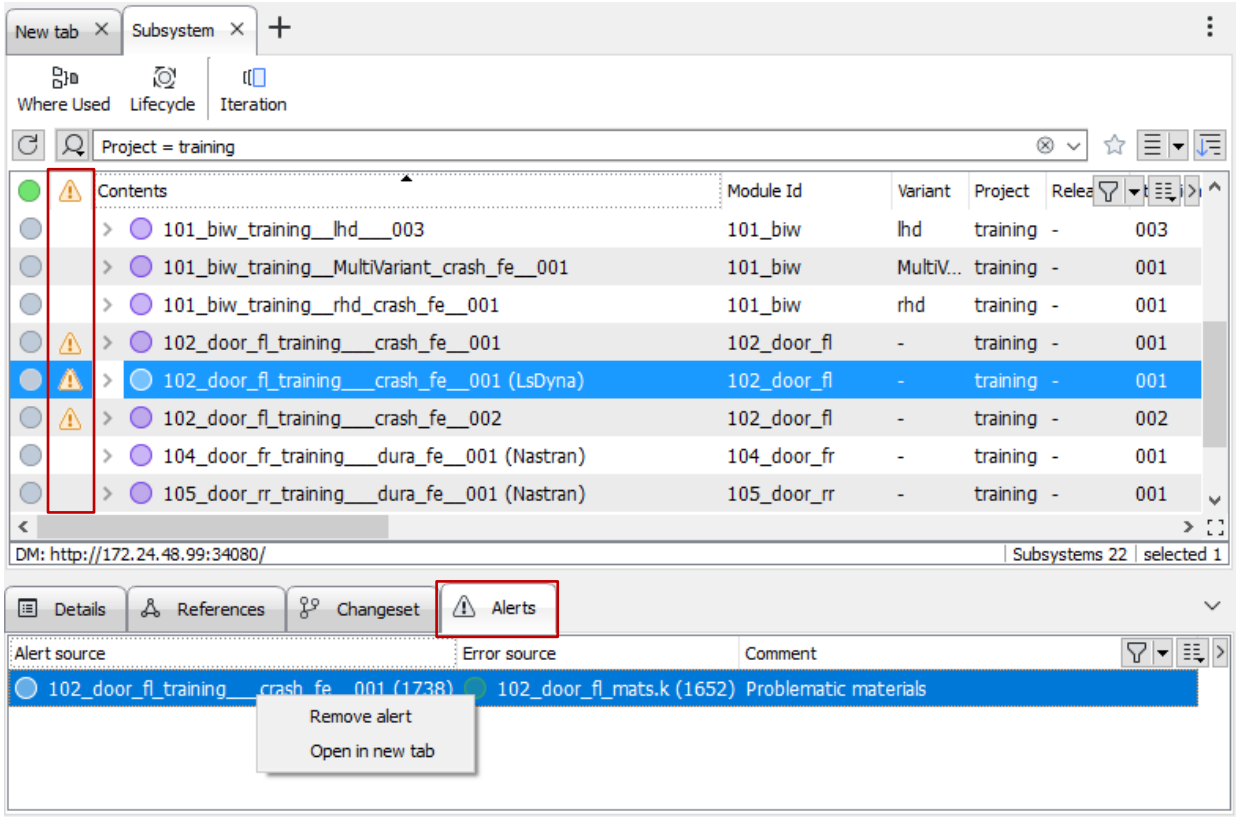
When connected to an SPDRM back-end of v1.8.0 or later, KOMVOS facilitates the management of data alerts. The alert mechanism offered by SPDRM is a solution, which, in case of errors, can raise alerts for the affected objects and propagate them to all their dependencies, offering the means to complete error impact analysis.

The generation of alerts is supported through two different mechanisms:

- By switching the value of a DM object attribute to a pre-defined value (e.g. switching the Status of a Subsystem to *Error*). This will generate an alert for all the descendants of the object whose attribute was modified. A detailed description of how to configure this is given in the SPDRM administration guide.
- By creating an *Issue* for a particular DM object. The *Issue Management* functionality enables the reporting and tracking of issues related to model and simulation data and is described in detail in chapter 6 of this guide. It offers the option to generate an alert for a specific DM object, for which an issue is created.

In KOMVOS, the user can access information about existing alerts in two ways:

- By adding the **Has Alerts** attribute as a column in the database browser lists. This will display a characteristic icon ⚠ for any DM object for which alerts have been raised.
- By switching to the **Alerts** bottom tab, which will display detailed information for the existing alerts of the selected DM object.

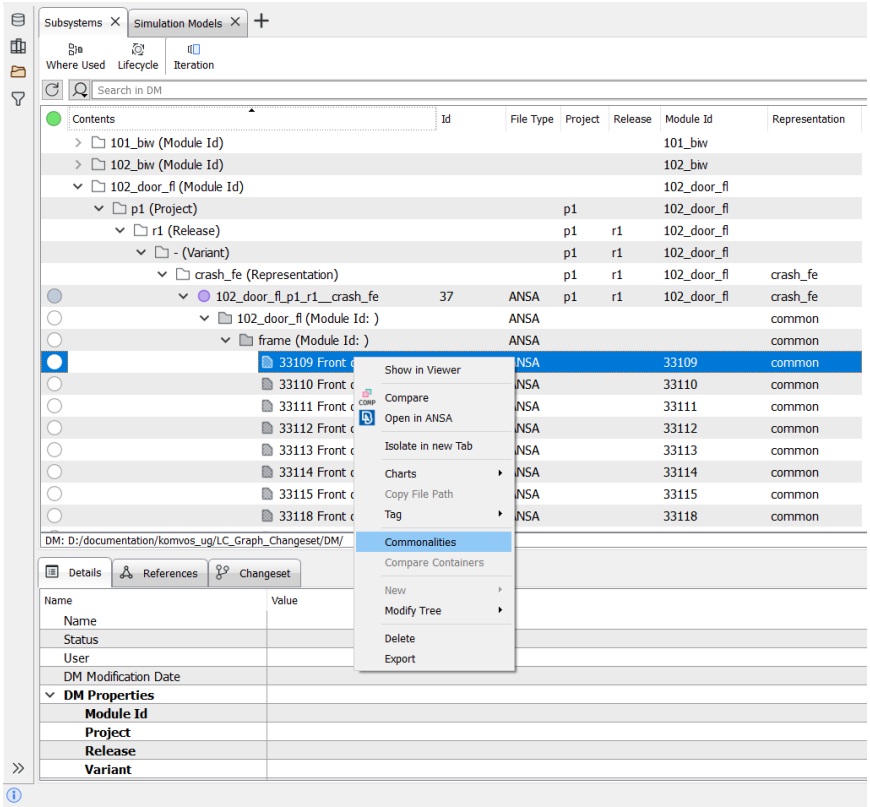


The following information is available in the **Alerts** tab:

- The *Alert source* column displays the DM object that transmitted the alert to the DM object selected in the main database browser list.
- The *Error source* column displays the DM object that raised the alert in the system.
- The *Comment* column displays the comment of the *Error source* DM object.

Using the context menu on the listed alerts, it is possible to select the option **Remove alert**, in order to clear an alert for the selected DM object, and the option **Open in new tab,** in order to open either the *Alert source* or the *Error source* item in a new tab in the database browser and review its metadata in detail.

4.2.3. Identifying common parts among variants

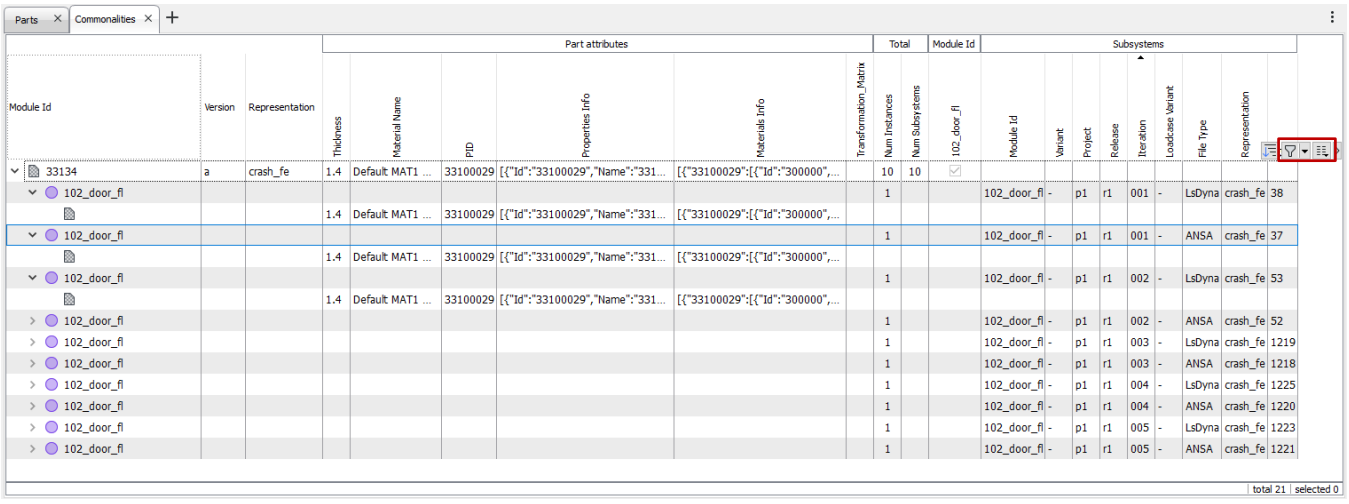


When handling complex structures, there's often the need to identify the different versions and variants of a model that make use of a particular part. Although this is already possible through the "Where Used" references, KOMVOS offers an additional, more elaborate view of this info through the **Commonalities** function.

This functionality, given one or more parts, identifies all the different versions and variants of Subsystems that use it and presents them in a concise table view.

The **Commonalities** function can be executed from the context menu of the selected Parts.

Once the function is executed, a new tab opens in the *Database* Workspace, listing all the different Subsystems where the selected Part is used in. The table in the **Commonalities** tab displays some key attributes of the Part, as well as the primary attributes of the Subsystems that contain the Part. The user can control the attributes that are displayed as columns in the table and filter the listed items .



4.3. Results Review

KOMVOS offers a complete solution for the assessment of both simulation and tests results. The user can view, edit and add results in the respective repository. The functionalities and the tools available in the Database workspace are described in the following chapters for each results category.

4.3.1. Simulation Results

The main tools and functionalities for the review and management of simulation results are listed below:

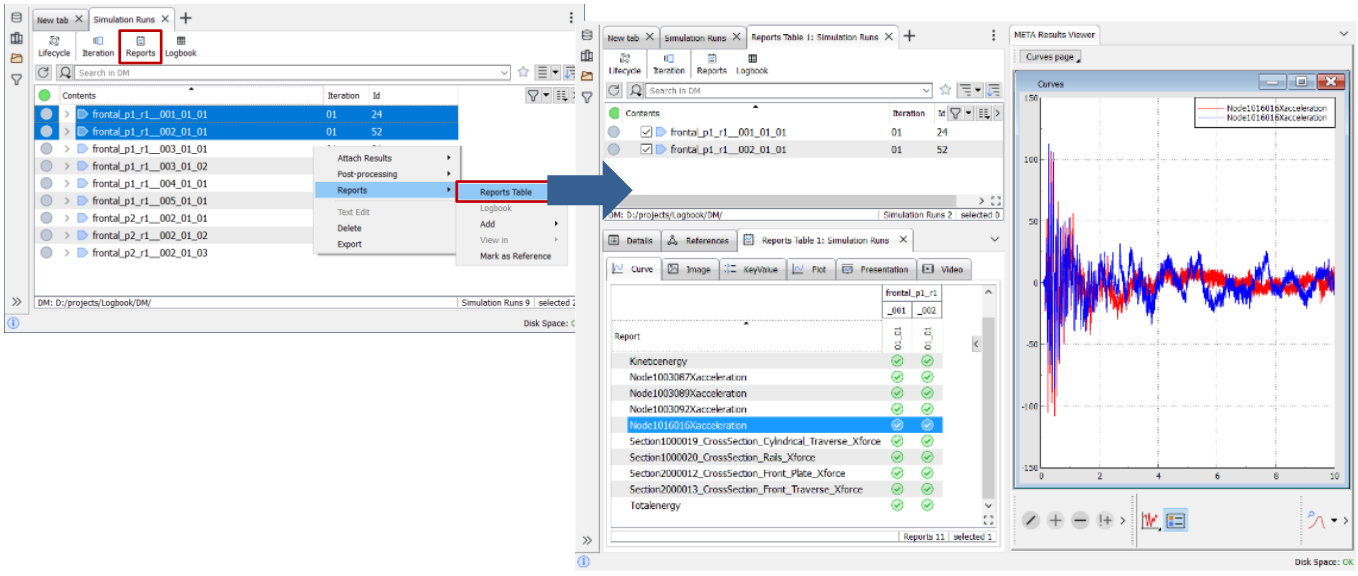
- Reports Table
- Logbook
- Add existing Reports in SDM system
- “In place” edit of Reports
- Recording of session files

Reports Table

The main tool for viewing the reports of a Simulation Run is the **Reports Table**. This tool can be accessed through the Simulation Models, Loadcases and Simulation Runs context menu option **Reports > Reports Table** or through the respective button at the top of the tab. Each tab shows a list of all available Reports of this type.

Using the Reports Table, the analyst can view all types of Reports graphically in the embedded META viewer. Furthermore, it is possible to compare results of different Simulation Run versions. The results are automatically overlayed and can be viewed both as plain values and graphically. Thus, the analyst can compare results in a quick and efficient manner. A Report can be viewed graphically with double click, as it is opened in the embedded META Viewer in the right section of the main window.

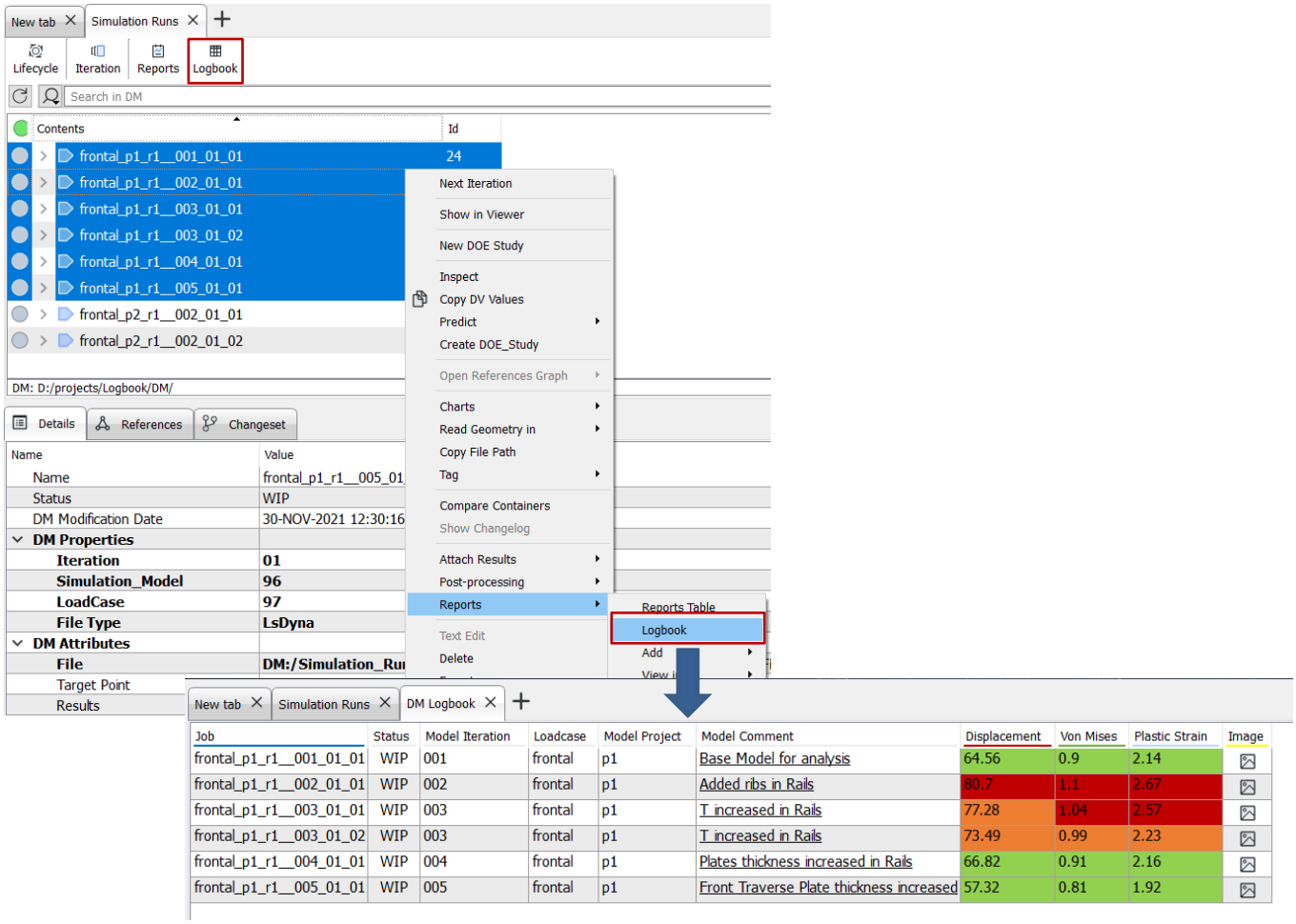
The detailed description for the **Reports Table** and the view of Results in the embedded META Viewer can be found in the 4.2 chapter of the Data Management Reference Manual.



Logbook

To assess a simulation, key values of the results are usually collected and compared against threshold values. This task can be quite time consuming and is usually carried out with some script or automation in a spreadsheet editor. To simplify this task, KOMVOS offers the **Results Logbook** that is available in the **Database** workspace.

The **Logbook** uses an Excel file as a template for the presentation of the key values. To use the **Logbook** for some DM objects, a template for the respective object type (Simulation Model, LoadCase, Simulation Run) must be defined first. Having set the template for the desired object types, the user can either select the option **Reports > Logbook** from the context menu or press the respective button at the toolbar, in order to trigger the **Logbook** action. The **Logbook** is displayed in a new tab.



The complete set-up and configuration of the Logbook is described in detail in paragraph 4.3.3 of the Data Management Reference Manual.

Add existing Reports and raw Results in a SDM system

It is possible to add both existing Reports and raw results under a Simulation Run. This functionality offers the capability to group all related results, both raw and processed, under a single object in an organized manner.

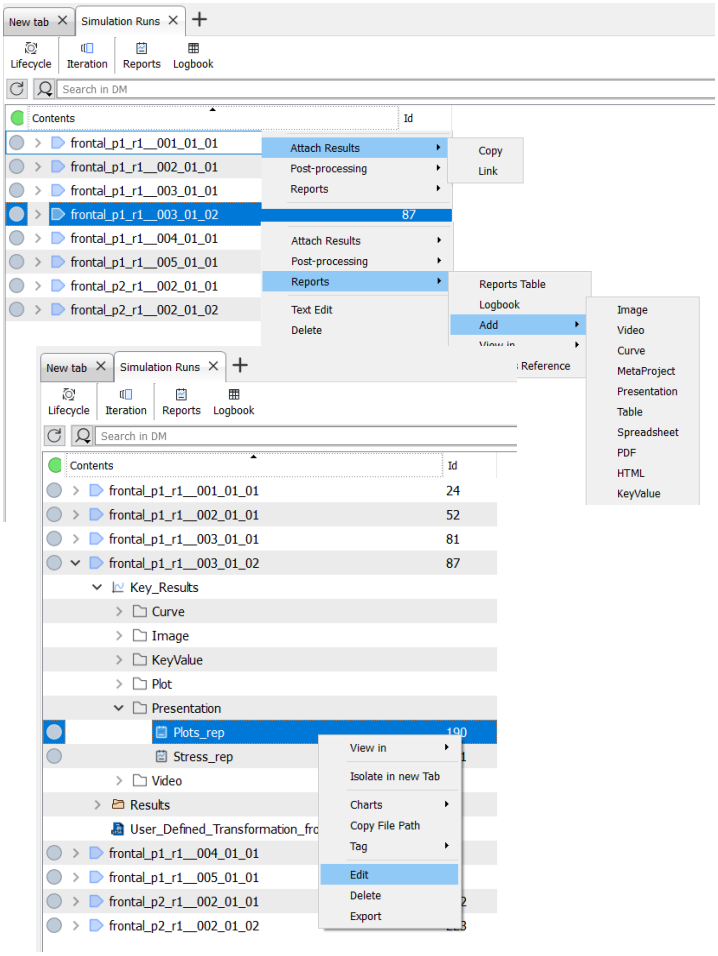
The raw results of the solver would be copied or linked in the Simulation Run through the **Attach results>Copy** and **Attach results>Link** functions of the context menu (due to the big size of raw result files it is recommended to use the "Link" option).

In case of existing key-results like images, curves, etc. produced by the post-processor and stored as files in the file system, they can be associated to the Simulation Run manually through the Simulation Run context menu option **Reports>Add file**.

Handling of the results is also possible through the KOMVOS Python API.

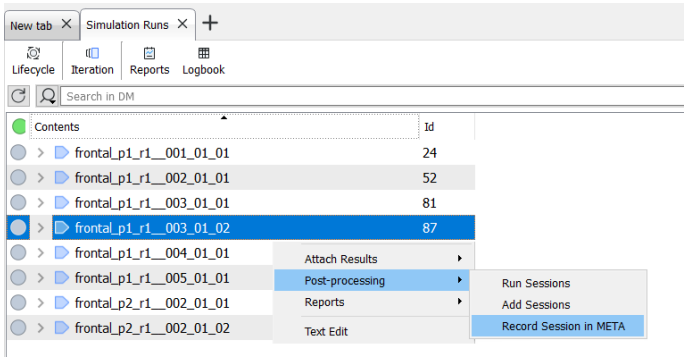
Through the **Edit** option in context menu of a Report attached under a Simulation Run, the user can edit the Report "on the spot".

The detailed description for the addition of existing Reports is available in paragraph 4.5 of the Data Management Reference Manual.



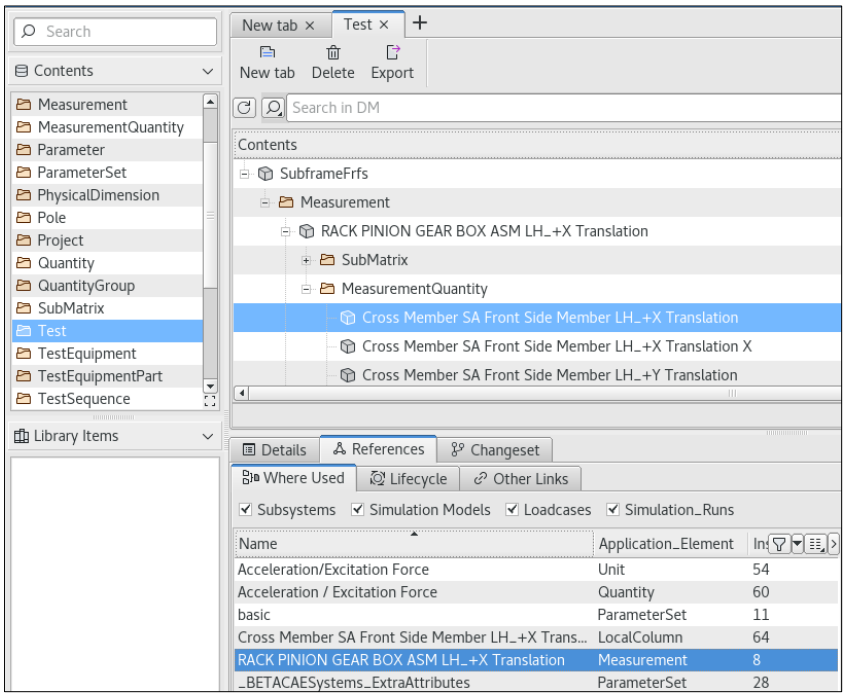
Recording of session files

The recording of a session file may be initiated through the **Post-processing > Record session in META** option of the context menu of a Simulation Run. The process is described in paragraph 4.6 of the Data Management Reference Manual.



4.3.2. Test Results

KOMVOS can connect to Test Results servers according to the ASAM-ODS standard in order for the simulation and test engineers to review Test Results like Channels, Photos, Movies, 3D Measurements, etc. The main tool for the management of the ASAM ODS databases in KOMVOS is the **Database** Workspace.

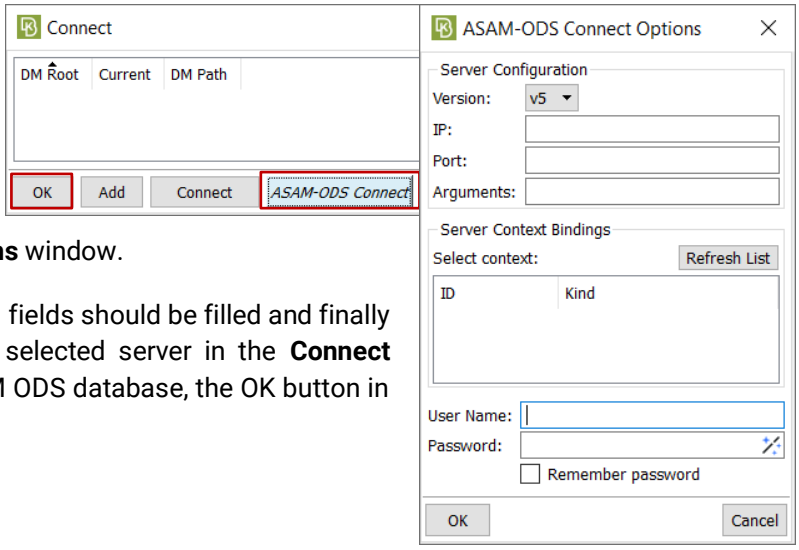


The key capabilities offered, include:

- Viewing the collected items in various simple or hierarchical views.
- Applying queries to fetch instances of application elements.
- Viewing attributes / relations of the instances of application elements.
- Visualize curves, images, videos or other data in META Viewer or in the current META window.

A detailed description for the viewing of Test Results by connecting to a Test Results server can be found in sub-chapter 4.4 of the Data Management Reference Manual.

ASAM ODS v5 and v6 servers are supported and can be connected to KOMVOS. To connect to an ASAM ODS server, either the **Login > Add/Connect** option can be selected or the **Extras > Connect** located in the top ribbon of the main window. In the **Connect** window that opens, the **ASAM-ODS Connect** button, when pressed triggers the **ASAM-ODS Connect Options** window.

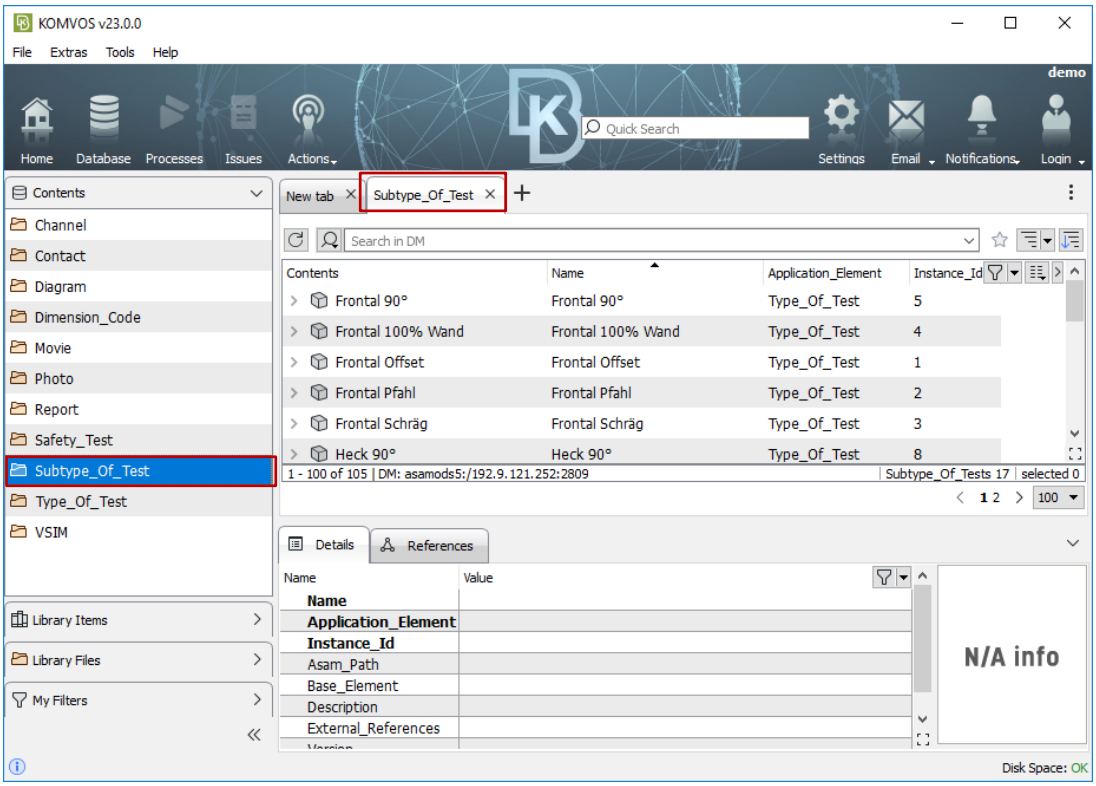


In the **ASAM-ODS Connect Options**, the required fields should be filled and finally the OK button should be pressed to add the selected server in the **Connect** Window. To finalize the connection to the ASAM ODS database, the OK button in the Connect window must be pressed.

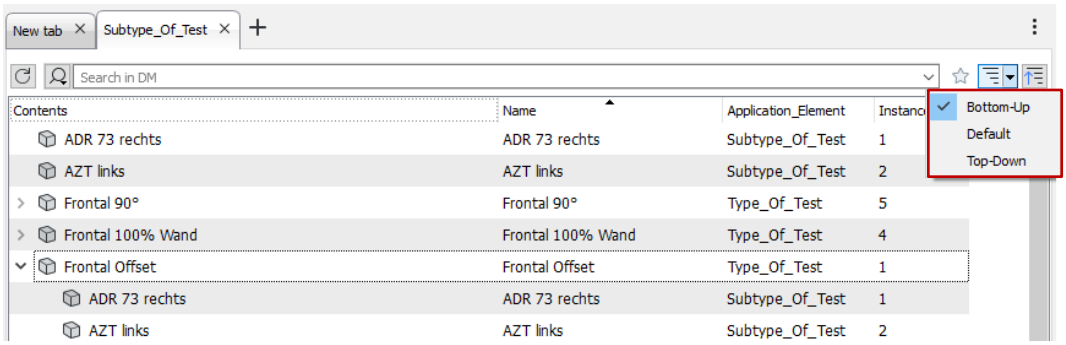
Viewing application instances

Once connected to the specified server, within the Database workspace of KOMVOS, the basic application elements (e.g. *Channel*, *Contact*, *Safety_Test*, *Subtype_Of_Test* etc.) are listed in the *Contents Index* pane on the left.

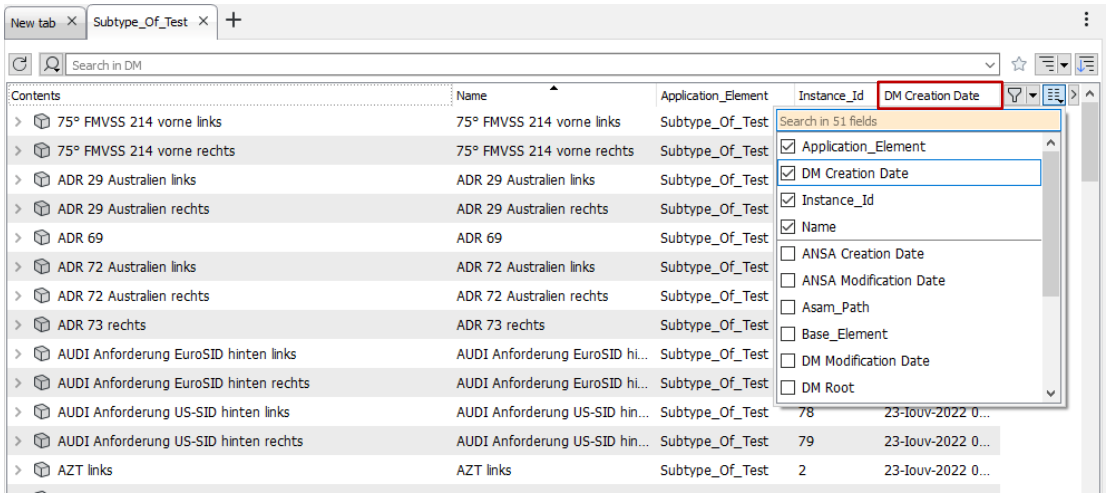
With double click on a specific application element, a new tab is created. The tab contains all the application instances of this Application Element.



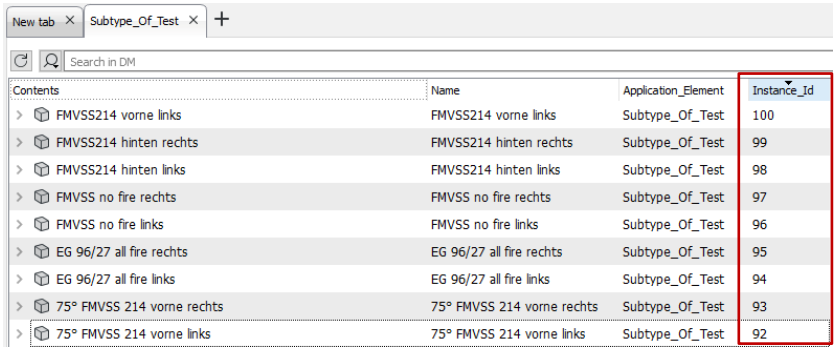
A tree mode view can be also defined through the *Change View* button. There is the option to select *Bottom-Up* tree view, where all the high hierarchy related application elements (parent elements) are depicted as well. For more information on the View, refer to chapter 4 of the Data Management Reference Manual.



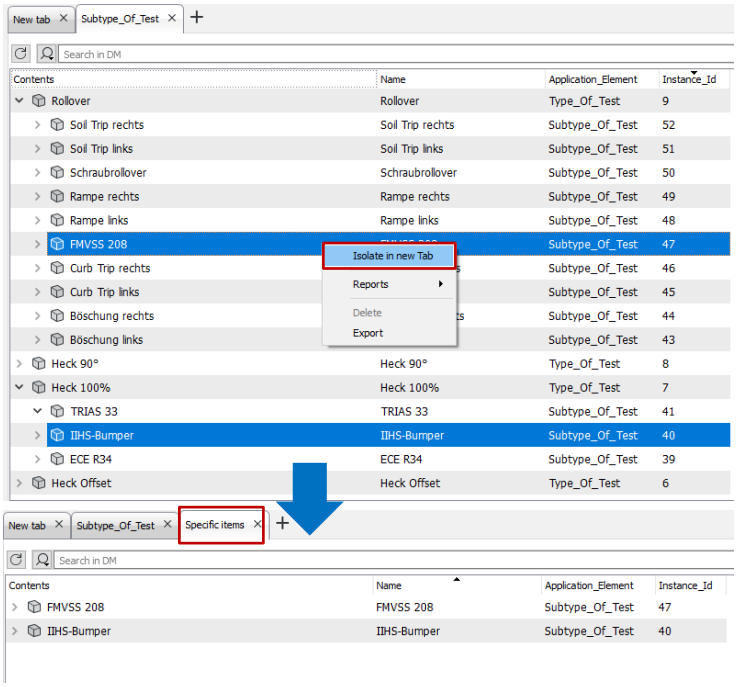
Information about the name, application element type and instance id can be found in the respective columns *Name*, *Application_Element* and *Instance_Id* which are shown by default. By pressing *the Show/Hide Columns* button, existing columns can be hidden, or extra columns can be added. Enable/Disable the respective checkboxes to add/remove columns respectively.



Application elements can be sorted with respect to the column information, by pressing the corresponding column name.



Selected elements can be also isolated in a new tab. This option is available in the context menu of the selected application elements. The selected items are then displayed in a new tab alone.



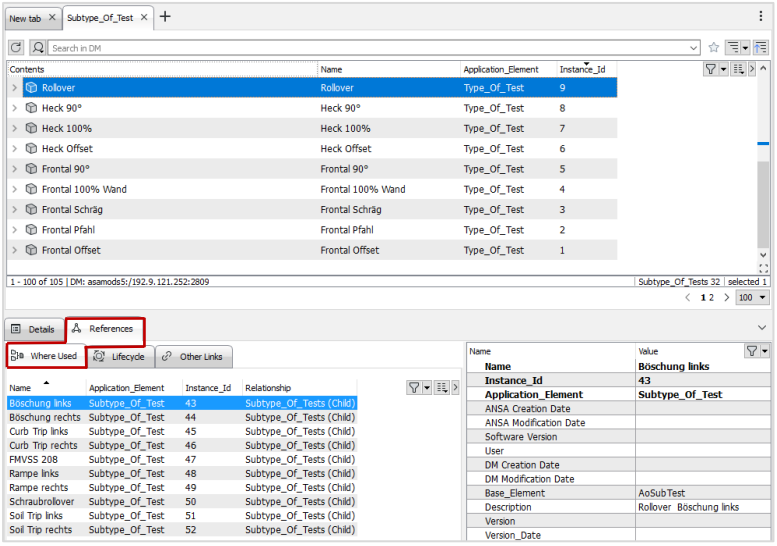
Filtering to fetch application instances

By selecting an application instance, its attributes can be viewed in the Details tab of the bottom section. Selecting a listed attribute in the Details tab and using the option *Add new column* of the context menu, the attribute will be added as a new column in the main list, displaying the value of each application instance. Existing columns can be removed by selecting the *Remove column* context menu option on the column header.

In the *References > Where Used* tab, all the relations of the selected application instance (both high and low hierarchy connected application instances) are listed. By selecting application instances of the *Where Used* tab, the relative attributes are displayed in the bottom right section of window.

More information on the filtering can be found in the chapter 3.5.2 of the Data Management Reference Manual.

Viewing attributes/relations of application instances



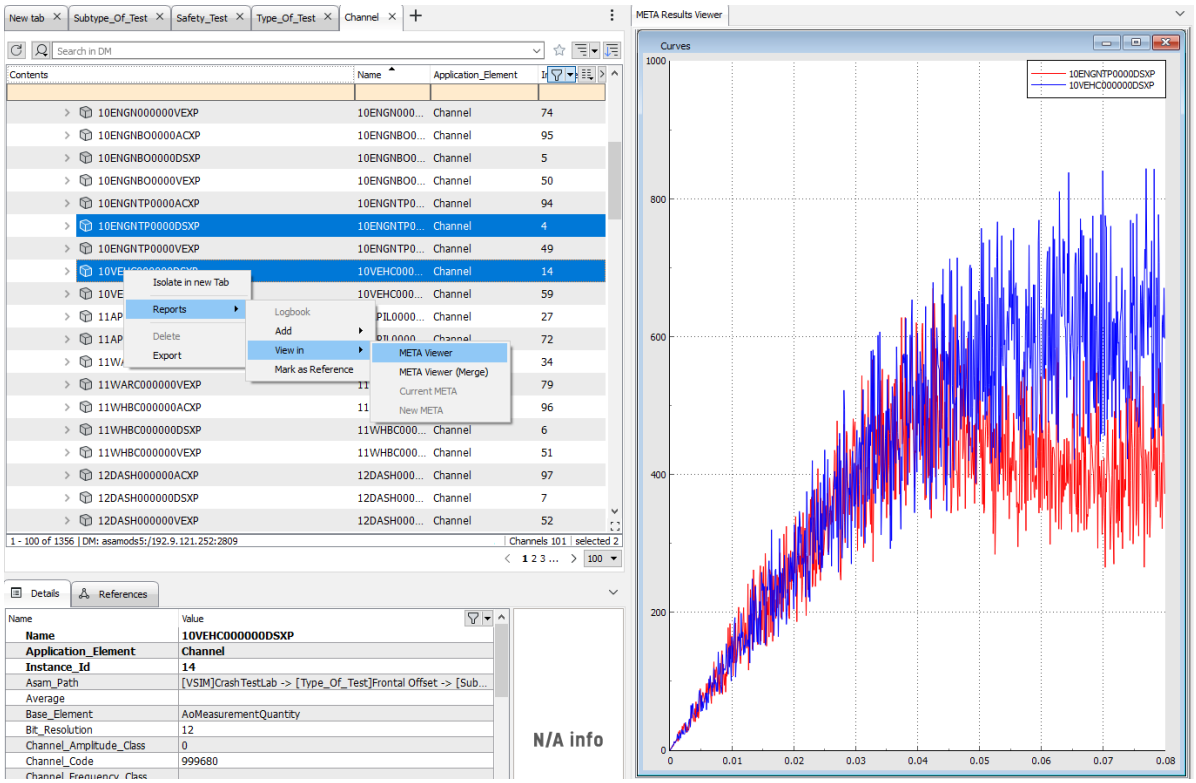
By selecting an application element, its' attributes can be viewed in the Details tab of the bottom section. Through right mouse button **Add new column**, a new column can be added, that contains information about the selected attribute. Existing columns can be removed by selecting the *Remove column* option respectively.

In the **References > Where Used** tab, all the relations of the selected application element (both high and low hierarchy connected application elements) are depicted. By selecting application elements of the **Where Used** tab, the relative attributes are displayed in the right section of window.

Visualizing data in the embedded META Viewer

Data from servers according to the ASAM-ODS standard can be visualized through the context menu of the selected application instances. From the context menu option **Reports > View in > META Viewer** results are visualized in the embedded META Viewer, in the right section of the main window.

Multi-selection is also available for the same type of results. As an example, two or more selected curves can be sent to META Viewer. The **META Viewer (Merge)** option gives the possibility to append new curve data to the existing plot.



4.4. Import data

KOMVOS enables the easy addition of data in the SDM system. This is achieved in two ways:

- By importing existing files
- By creating new DM Object definitions

In both cases, it is the user's responsibility to provide the identity of the imported data to the system.

4.4.1. Import existing files

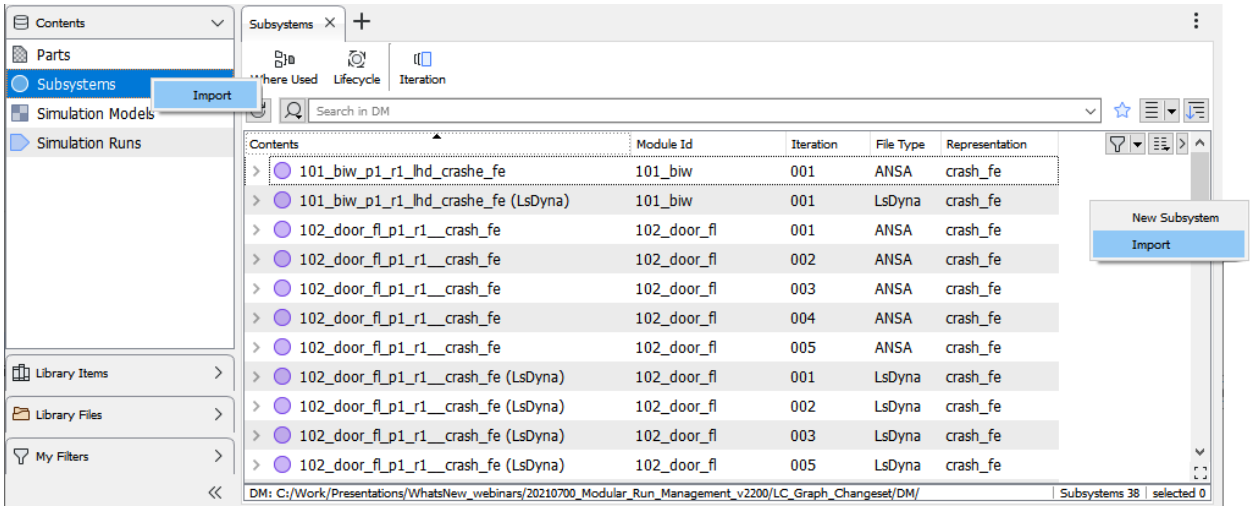
During the import of existing files, once the representation file, the identity and any other metadata are provided, the steps of the import process are:

1. Conflict detection and resolution
2. Indexing
3. Addition of the DM Header (where applicable)
4. Upload to the SDM system

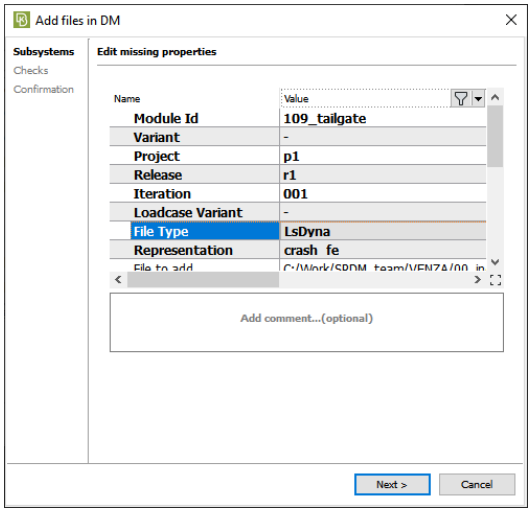
Detailed information about the process of importing data can be found in Chapter 3 of the Data Management Reference Manual.

Subsystem import

The import of existing files as Subsystems in KOMVOS is initiated using the **Import** option on the *Subsystems* data type container in the *Contents* list of the *Database* workspace. The same action is accessible through the context menu in the *Subsystems* list.

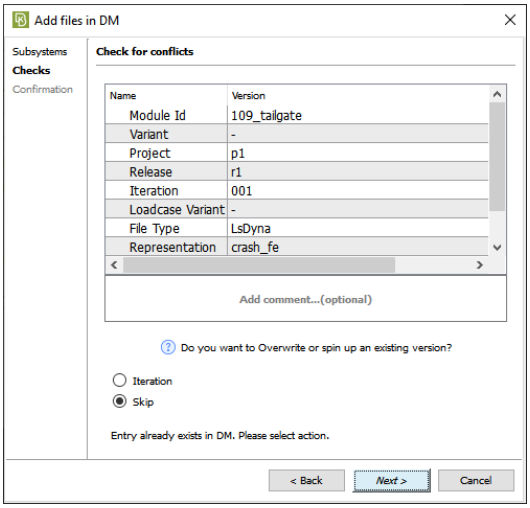


A file manager window opens for the user to select a file for import. Upon the file selection, the **Add Files in DM** wizard opens, guiding the user through the import process.



First, the properties of the Subsystem must be defined. All attributes that are required for the unique identification of the Subsystem, according to the data model, need to be defined in order to proceed with the import. In case of any missing or invalid properties, the wizard cannot proceed to the next page.

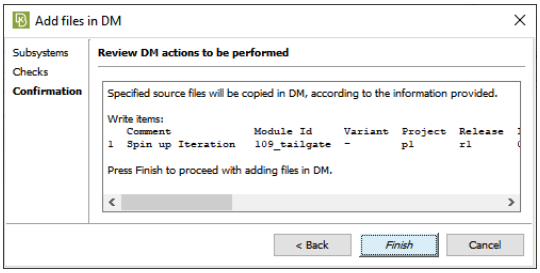
Once all properties are well defined, pressing the **Next** button advances the wizard to the next page.



In the next page of the wizard, KOMVOS performs a check for potential conflicts within the SDM system. Essentially, a conflict is detected if there is a Subsystem with the same primary attributes already in the SDM system. Any existing conflict is reported in the wizard and the user is prompted to take some action:

- Save the Subsystem as a new iteration
- Overwrite the existing Subsystem
- Skip saving and halt the process

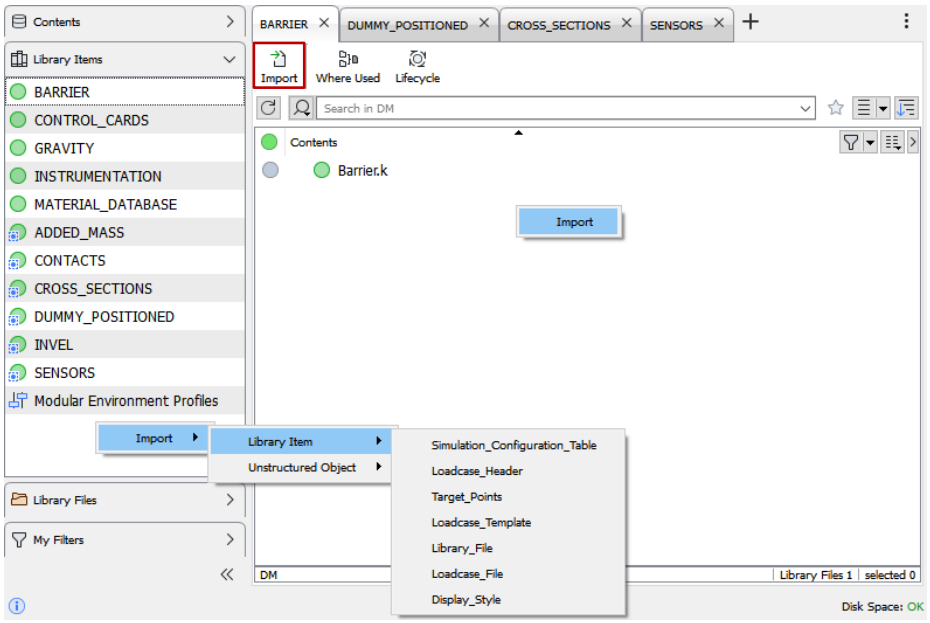
More information on conflict resolution is given in paragraph 3.2.1 of the Data Management Reference Manual.



In the final page of the wizard, information about the Subsystem to be imported to the SDM system is displayed.

Pressing the **Finish** button, the Subsystem import is finalized.

Library Items import



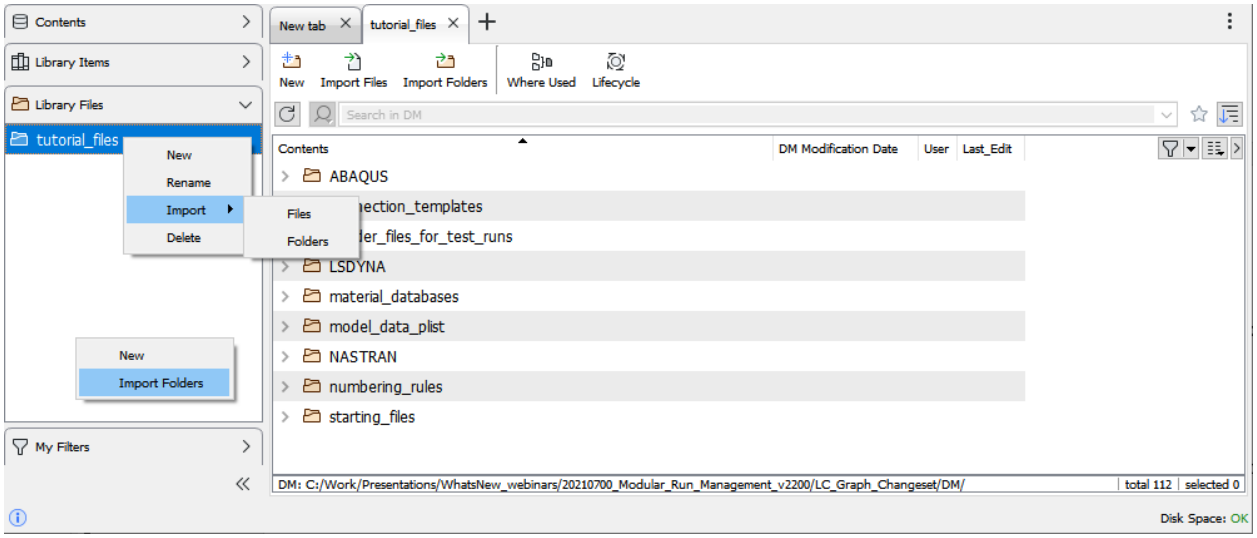
The import of existing files as Library Items in KOMVOS is initiated using the **Import** option in the context menu of the *Library Items* section of the *Database* workspace. Depending on the data model, the desired Library Item type needs to be selected.

Alternatively, the import function can be triggered using the **Import** toolbar button or the respective context menu option in any list of Library Items.

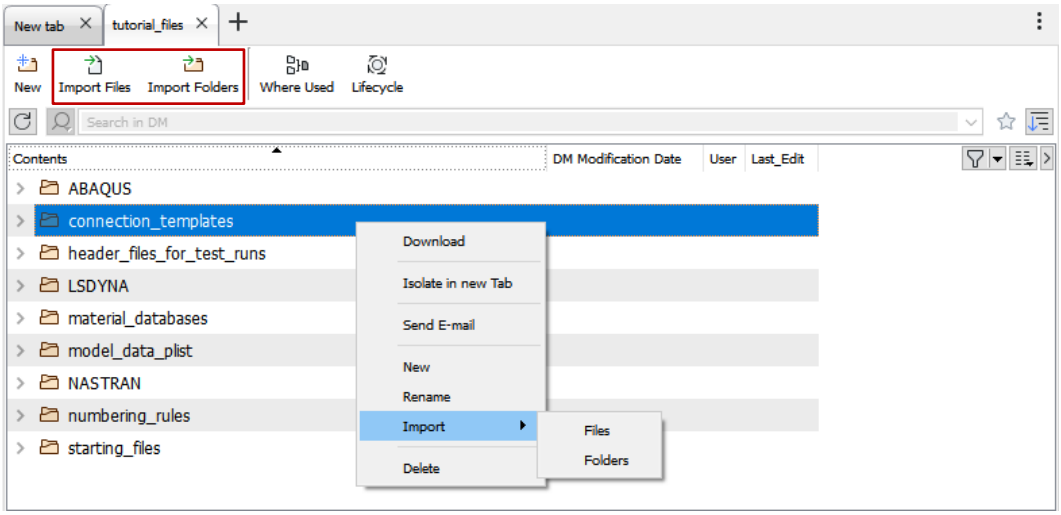
A file manager window opens for the user to select a file for import. Upon the file selection, the **Add Files in DM** wizard opens, guiding the user through the import process. The wizard pages are described in detail in the Subsystems import section above.

Library Files import

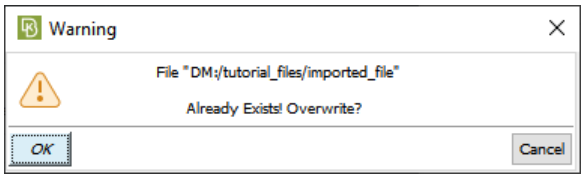
The import of existing files as Library Files in KOMVOS is initiated using the **Import Folder** option in the context menu of the *Library Files* section of the *Database* workspace. It is also possible to select one of the existing Library Files folders and use the **Import** context menu option to import *Files* or *Folders* directly in it.



Alternatively, the import function can be triggered using the **Import Files** or **Import Folders** toolbar button or the respective context menu option in an already opened Library Files folder tab.



If the **Import Folders** option is used and a folder with the same name already exists in the SDM system, the folder contents are merged. If the **Import Files** option is used and a Library File with the same name already exists, the user is asked whether the existing file should be overwritten, or if the import should be cancelled.

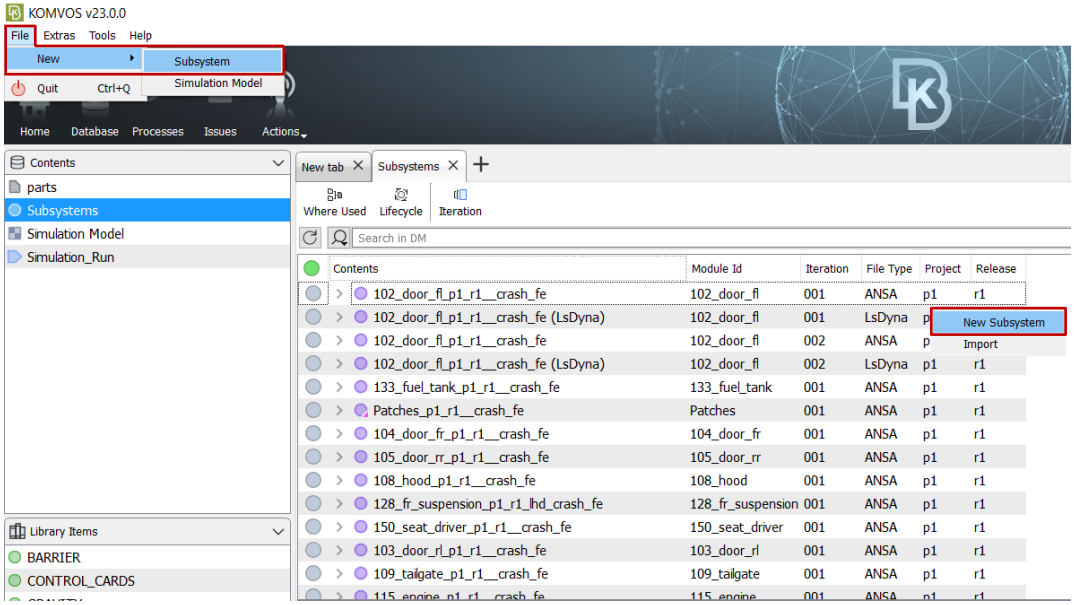


4.4.2. Creation of new DM objects

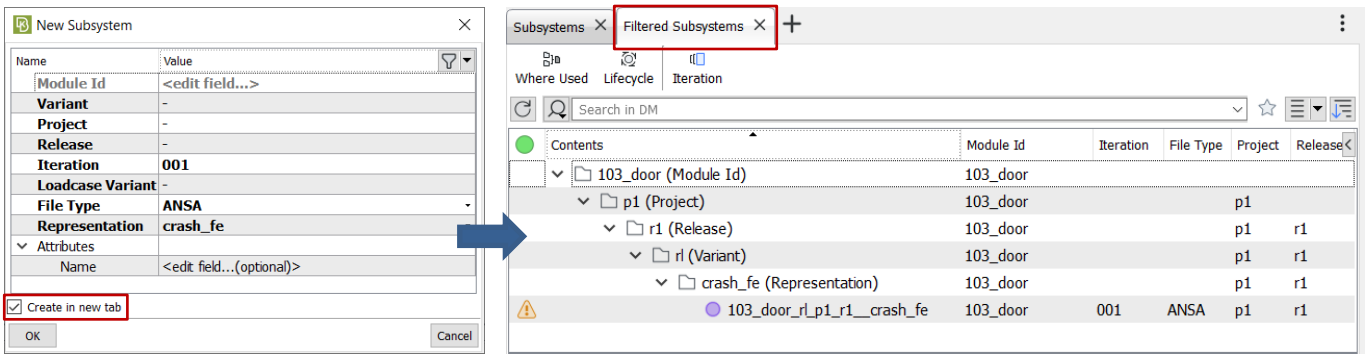
There are cases where a DM object needs to be created as an empty placeholder, in order to get its attachments at a later time. Through Import this is not possible. So, for certain DM object types, KOMVOS enables the creation of new, empty DM objects. For file-based back-ends, this functionality is supported for Subsystems and Simulation Models, while for an SPDRM back-end, for Simulation Models, Loadcases and Simulation Runs.

4.4.2.1. In file-based SDM back-ends

When connected to a file-based DM, the **File>New>Subsystem** or **File>New>Simulation Model** option can be selected in the main menu of KOMVOS, in order to initiate the creation of a new Subsystem or Simulation Model definition respectively. Alternatively, the context menu option **New>Subsystem** in the *Subsystems* tab or **New>Simulation Model** in the *Simulation Models* tab can be used.

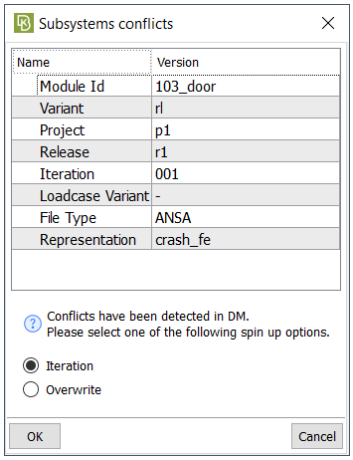


In the *New* window that opens, the user needs to define the properties of the new DM object. If the checkbox **Create in new tab** is enabled, once the **OK** button is pressed, the newly created DM object is opened in a new tab.



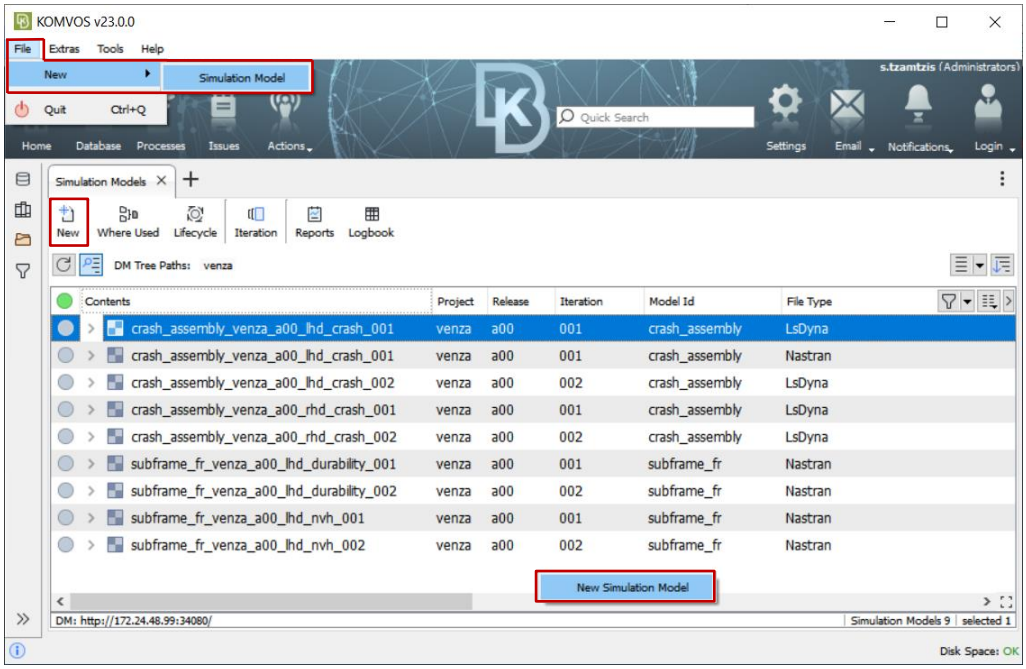
For file-based back-ends, the newly created DM object definition must be saved by explicitly selecting the option **Save** from the context menu.

During the saving procedure, KOMVOS queries the SDM for objects with the same properties. In case there is an object with the same properties, the *Conflicts* window will appear and the user is prompted to select a conflict resolution option.



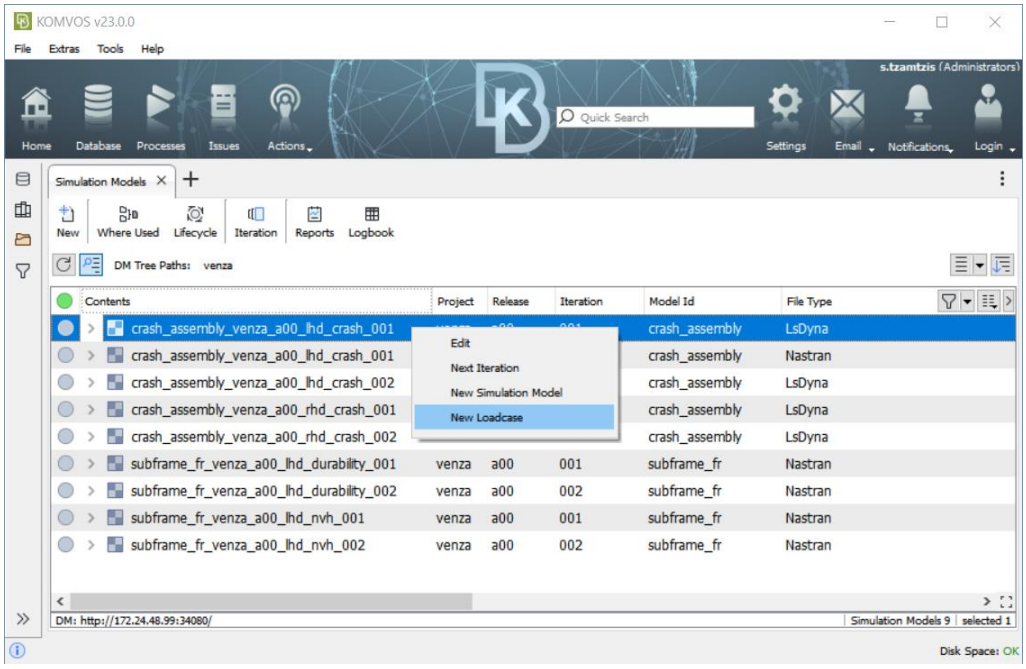
4.4.2.2. In SPDRM back-ends

When connected to an SPDRM back-end, the **File>New>Simulation Model** option can be selected in the main menu of KOMVOS, in order to initiate the creation of a new Simulation Model definition. Alternatively, the context menu option **New>Simulation Model** in the *Simulation Models* tab and the respective toolbar button can be used.

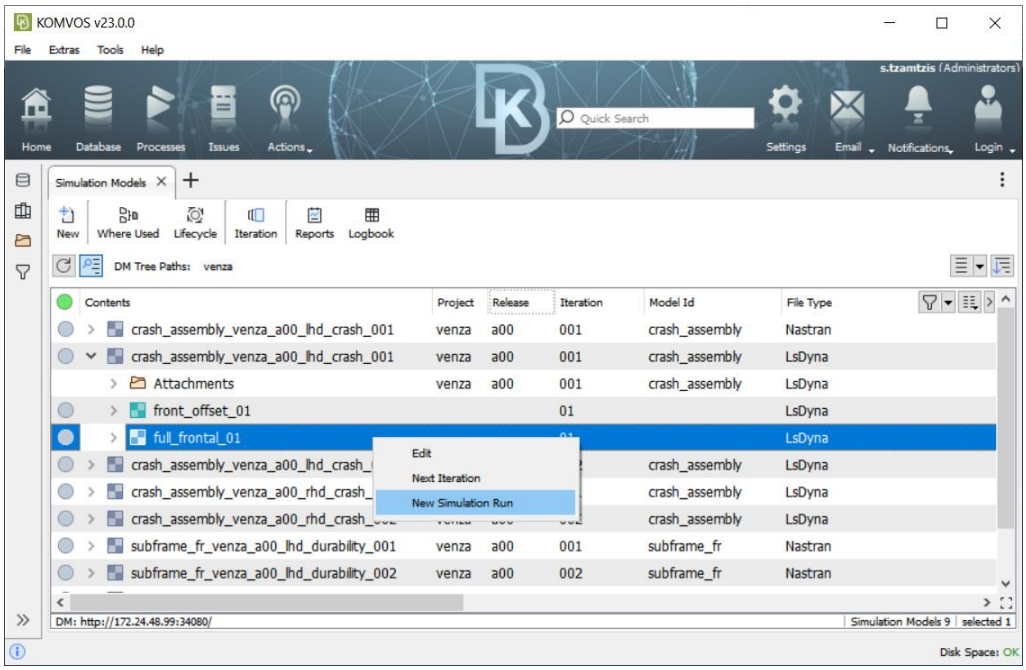


Similar to what is described above for file-based SDM back-ends, the *New* window opens, where the user needs to define the properties of the new Simulation Model definition. If the checkbox **Create in new tab** is enabled, once the **OK** button is pressed, the newly created DM object is opened in a new tab. In addition, after pressing **OK**, the newly created Simulation Model definition is automatically saved in the SDM. In case of existing conflicts, a new version of the Simulation Model will be created, with no history lifecycle link to the existing version.

With an SPDRM back-end, KOMVOS also offers the option to create new Loadcase definitions for existing Simulation Models. This is achieved by selecting the context menu option **New Loadcase** on a Simulation Model.



After providing the Loadcase attributes in the window that appears and pressing **OK** to finalize the Loadcase definition creation, the new Loadcase will appear under the selected Simulation Model. In addition, it is also possible to create a new Simulation Run, by selecting the context menu option **New Simulation Run** on an existing Loadcase. The user will need to provide the Simulation Run attributes and press **OK** to finalize the Simulation Run definition creation.



4.5. Edit data

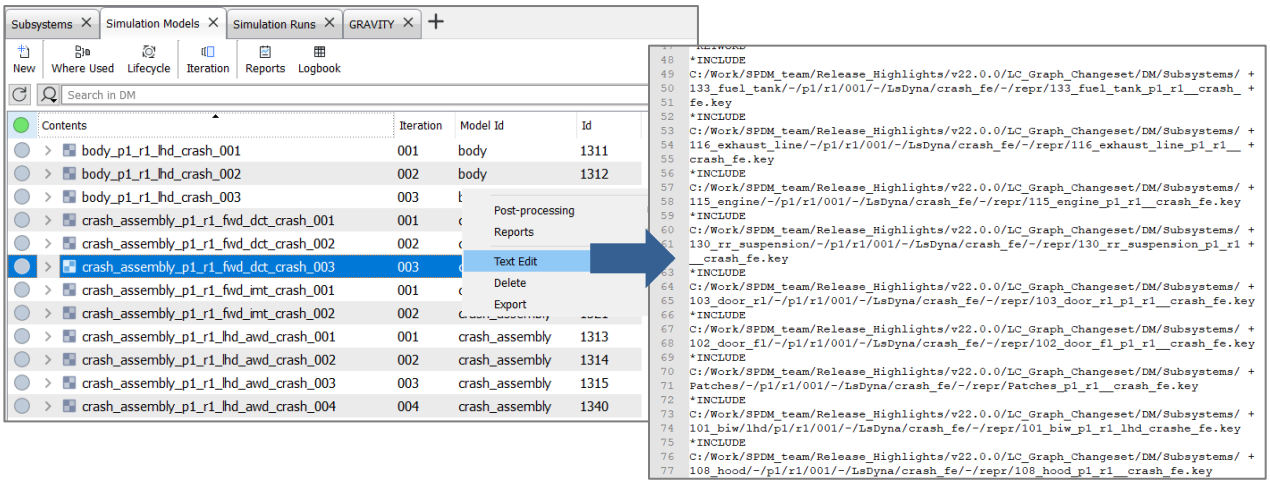
The *Database* workspace in KOMVOS is a dynamic user interface, that allows the user not only to view data, their respective relations and attached results, but also to edit the file of DM objects or their attributes.

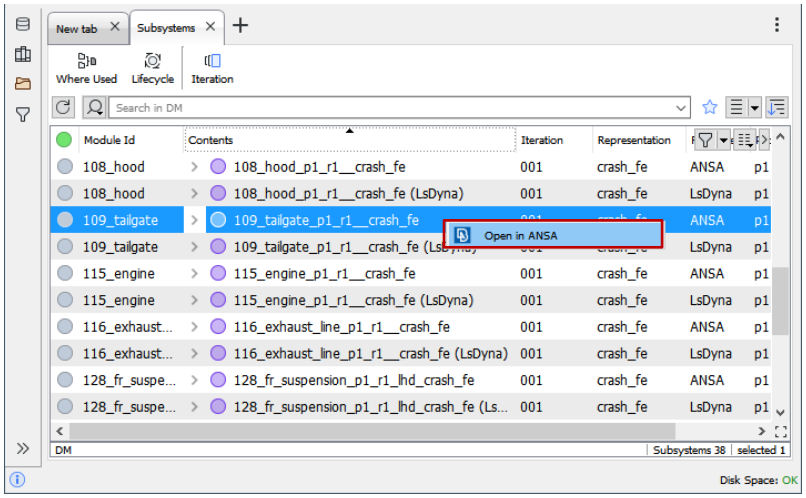
4.5.1. Edit the file of DM objects

Different options are offered for the editing of data, depending on whether KOMVOS is connected to a file-based or an SPDRM back-end.

File-based SDM back-ends

When connected to a file-based SDM back-end, it is possible to directly edit the file of any DM objects whose representation file is a solver keyword file, using the **Text Edit** context menu option. Once selected, the keyword file is opened with the system's default text editor.





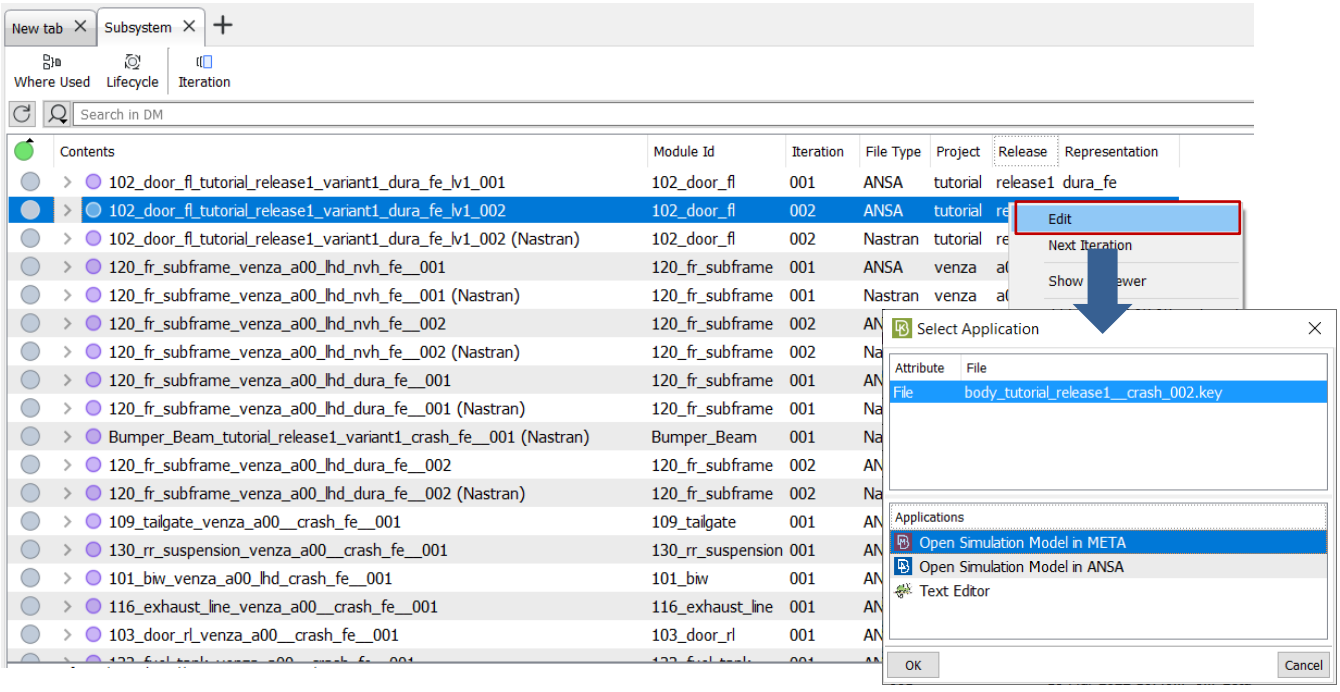
For DM objects whose representation file is an ANSA database, the default KOMVOS configuration for file-based back-ends offers the DM object action **Open in ANSA** in the context menu. This option allows the user to open the model in an ANSA session for viewing purposes. Modifications of the representation file are not possible.

For more information on the DM object actions, please refer to chapter 7 of this guide.

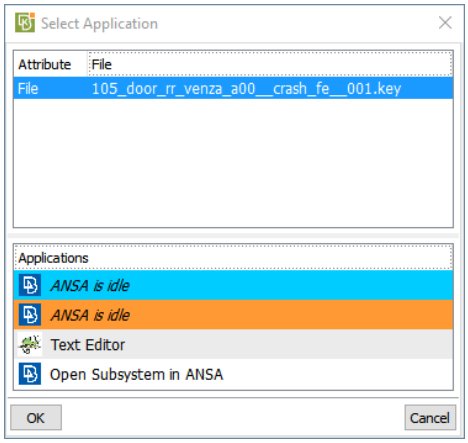
SPDRM back-ends

When connected to an SPDRM back-end, editing of data is performed through the context menu option **Edit**, which will launch a predefined application in order to edit the selected DM object.

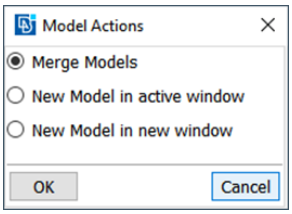
Depending on the SPDRM configuration (i.e. definition of *Mimetypes*), one or more applications may be defined as editors for the various files of each DM object. In that case, the *Select Application* window will be launched, allowing the user to select both the file to be edited, as well as the desired application. The available applications per data type and file extension can be defined by the SPDRM Administrator. For more information refer to SPDRM Administrator's guide paragraph 6.2.



If the opened file is modified and the applied changes are saved, after quitting the application, KOMVOS automatically detects that the initial file was modified and the **Add file in DM** wizard opens, in order to decide the versioning policy of the edited DM item.



KOMVOS enables the reusability of ANSA/META sessions through a built-in mechanism that keeps track of ANSA/META applications that were launched for the editing of DM Objects. When editing a DM Item and an open ANSA/META session already exists, the auto-reusability mechanism takes effect and the *Select Application* window shows any existing ANSA/META sessions that can be selected as editors.



When more than one items are opened in the same ANSA session, it is not possible to make modifications to any of the items.

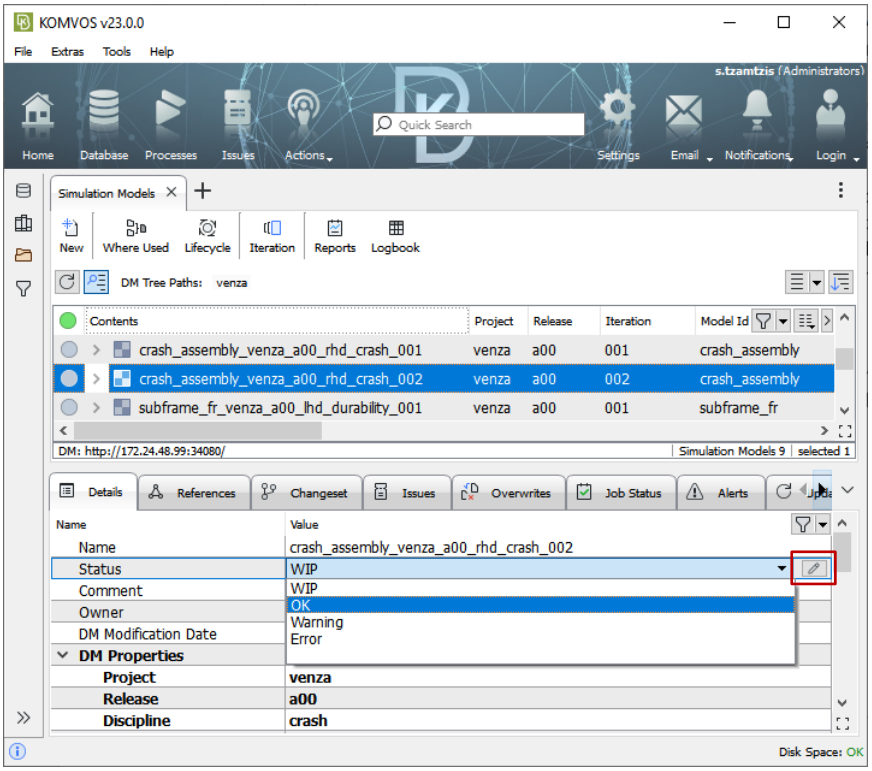
More information can be found in paragraph 3.4.1.1 of the Data Management Reference Manual.

With KOMVOS connected to an SPDRM back-end, it is also possible to view multiple items in the editor. Opening multiple items is only available for applications which have been configured accordingly (i.e. the “Allow Multi Edit” flag is active in their mimetype definition). For more information, please refer to paragraph 6.2 of SPDRM Administrator’s guide.

This functionality is supported out of the box, for editors that support opening multiple files through command line options. In case of ANSA and META, that do not support opening multiple files through command line, viewing multiple items in the same session can be achieved by re-using an existing session, as described in paragraph 3.4.1.1 of the Data Management Reference Manual.

4.5.2. Edit the attributes of DM objects

KOMVOS enables the in-place editing of DM object attributes, directly in the SDM client, regardless of the back-end.



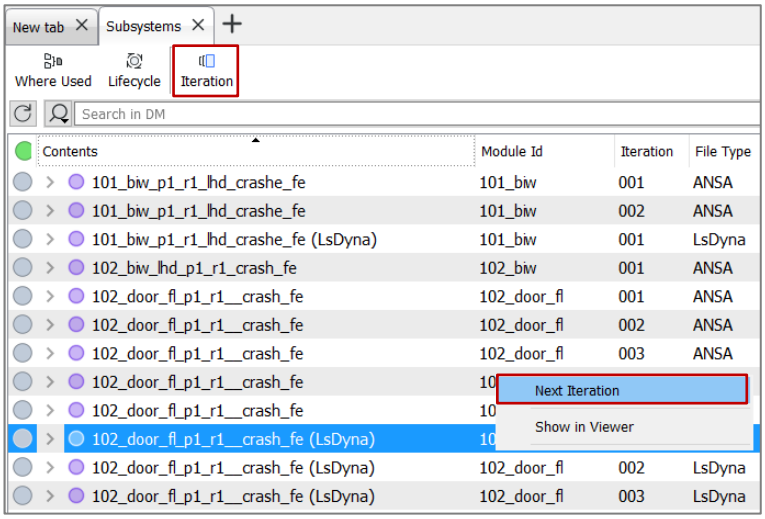
In all cases, selecting a DM object in the *Database* browser and viewing its *Details* tab, a characteristic icon will appear in the value field of any attributes whose editing is allowed., when the mouse is hovered over the attribute.

In principle, modification is allowed for the values of all secondary and additional attributes, except for the indexing metadata and some MRM-related attributes added by ANSA.

Pressing the left mouse button anywhere inside the attribute value field or directly on the edit icon , its value can be directly modified.

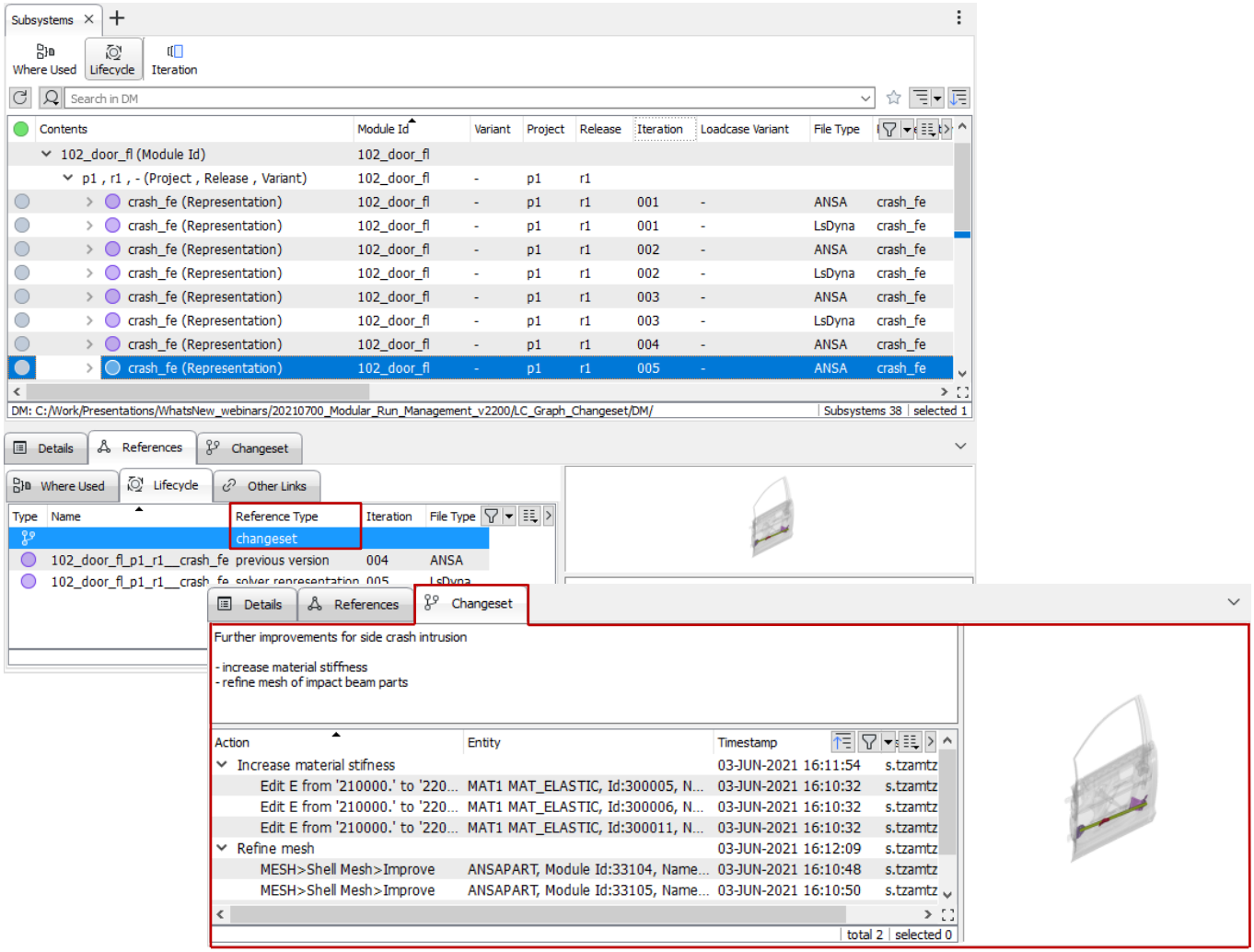
4.6. Creating New Iterations

KOMVOS offers the capability to initiate the creation of new iterations of existing DM objects directly through the SDM client. One or more DM objects of the same type tab can be selected and through the context menu the user can select the **Next Iteration** option (or the **Iteration** button in the toolbar). This action invokes ANSA, automatically connected to the SDM system, and ANSA loads the definition files of the selected objects.



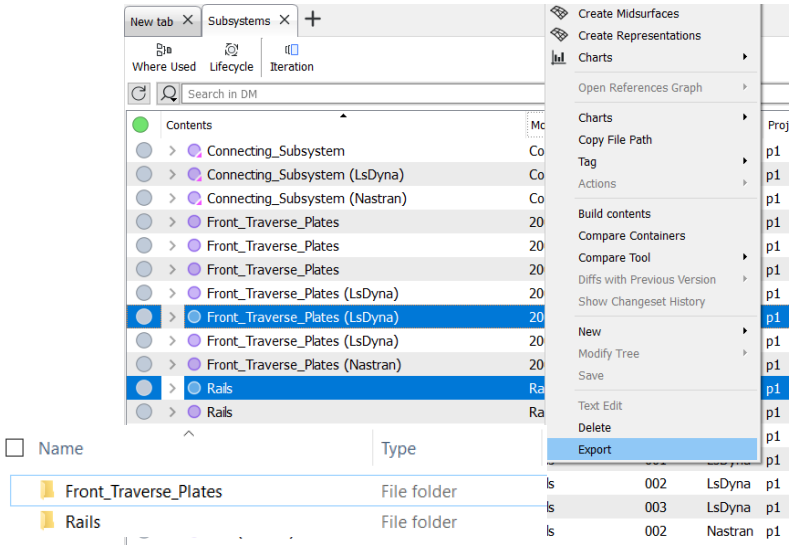
More information on the Next Iteration function can be found in paragraph 3.9.1 of the Data Management Reference Manual.

Once the new versions are created, they can be saved in the SDM system. For Parts and Subsystems, the applied modifications that led to the creation of the new versions are recorded and reported as a changeset, which represents the delta between two successive versions. In KOMVOS, changesets are shown as metadata of the Part or Subsystem DM Objects they relate to, either in the **Lifecycle** tab or in the dedicated **Changeset** tab.



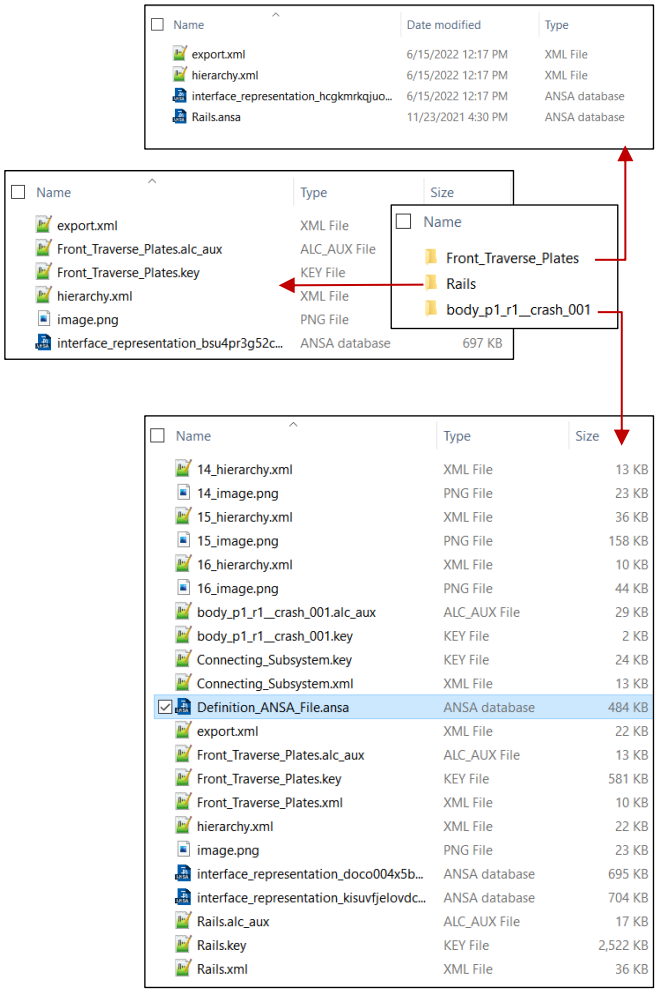
4.7. Export data

Through KOMVOS, selected data can be exported on demand. The **Export** function is needed to share data with users that do not have access to the SDM system or to transfer data to a different file system. In order to export data, the desired objects can be selected in the respective tab and through the context menu, the user can select the **Export** option. The **Export** functionality is available for both single and multiple selections.



Once the **Export** option is selected, the file-manager window pops up, prompting the user to choose the directory, in which the data will be exported. For each selected object, a sub-folder named after the exported object, is created in the selected export directory.

Upon export, all the files attached to the selected DM objects are copied by the data manager to the export folder. Additionally, an xml file is written for each DM object, that contains the object's metadata (attributes and hierarchy).



The contents of each exported folder differ, depending on the object type:

- Base Modules:
  - all files attached to the selected DM Objects
  - xml file containing the DM object's metadata (attributes and hierarchy)
- Compound Containers (i.e. DM objects that contain other DM objects):
  - Depending on the "Save Option" either a single file is exported containing the information of all contents or all contents are exported as separate files with the main file including references to those

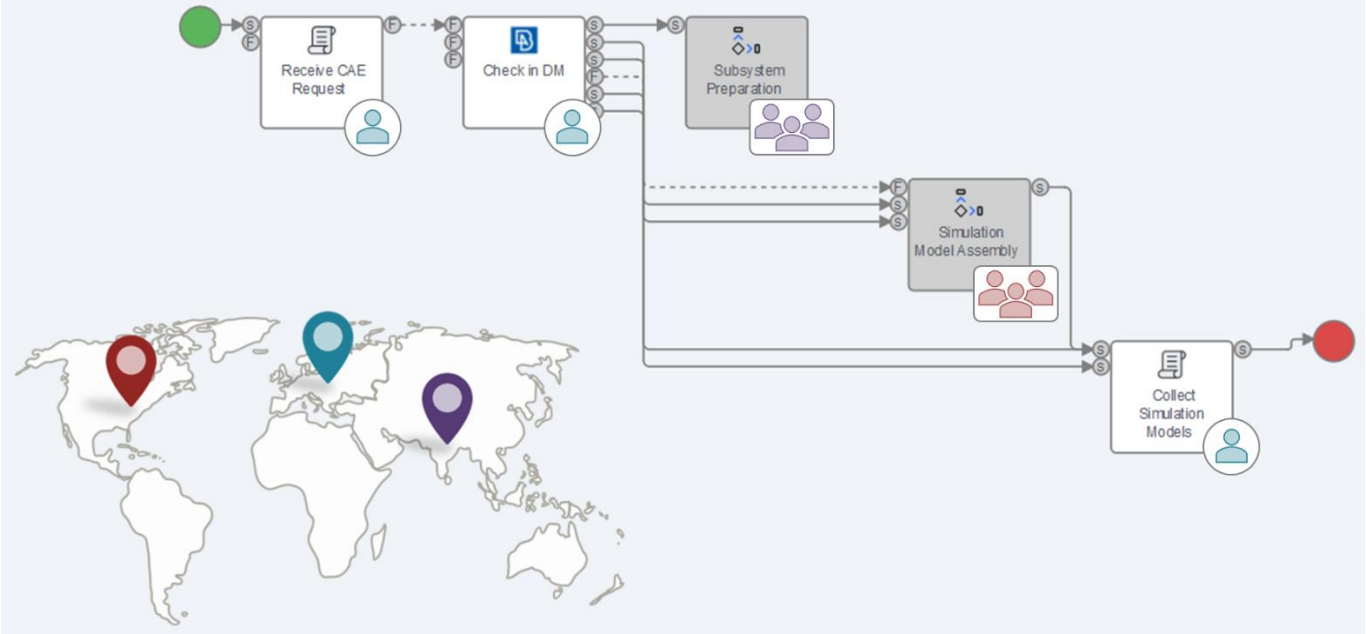
More information on the export procedure, the exported files and the handling of absolute paths in case of solver representation files, can be found in paragraph 3.3 of the Data Management Reference Manual.

5. Process Management

KOMVOS offers a dedicated workspace for the management of Business Processes. Within the Process Management workspace, users can design and execute processes and monitor their progress.

A Process is a series of tasks, in pre-defined order, that are pre-configured to execute certain batch or interactive actions. Within a process, each task may be assigned to a specific team or user, who will receive a notification once the task is ready to be executed. The tasks can be either Python scripts or Application Nodes, with the latter allowing the launch of any software application with proper inputs and certain expected outputs. Finally, tasks are pre-configured to be executed either on the executing user's workstation or on remote resources.

Designed processes are stored as Templates in the Process Library. A process template can then be executed either interactively, within the Process Workspace, or completely in batch, through script actions.



Process Management in KOMVOS is powered by SPDRM and therefore, it is only available when KOMVOS is connected to an SPDRM back-end (version 1.6.0 and later).

Process Execution is supported from KOMVOS version 22.0.0.

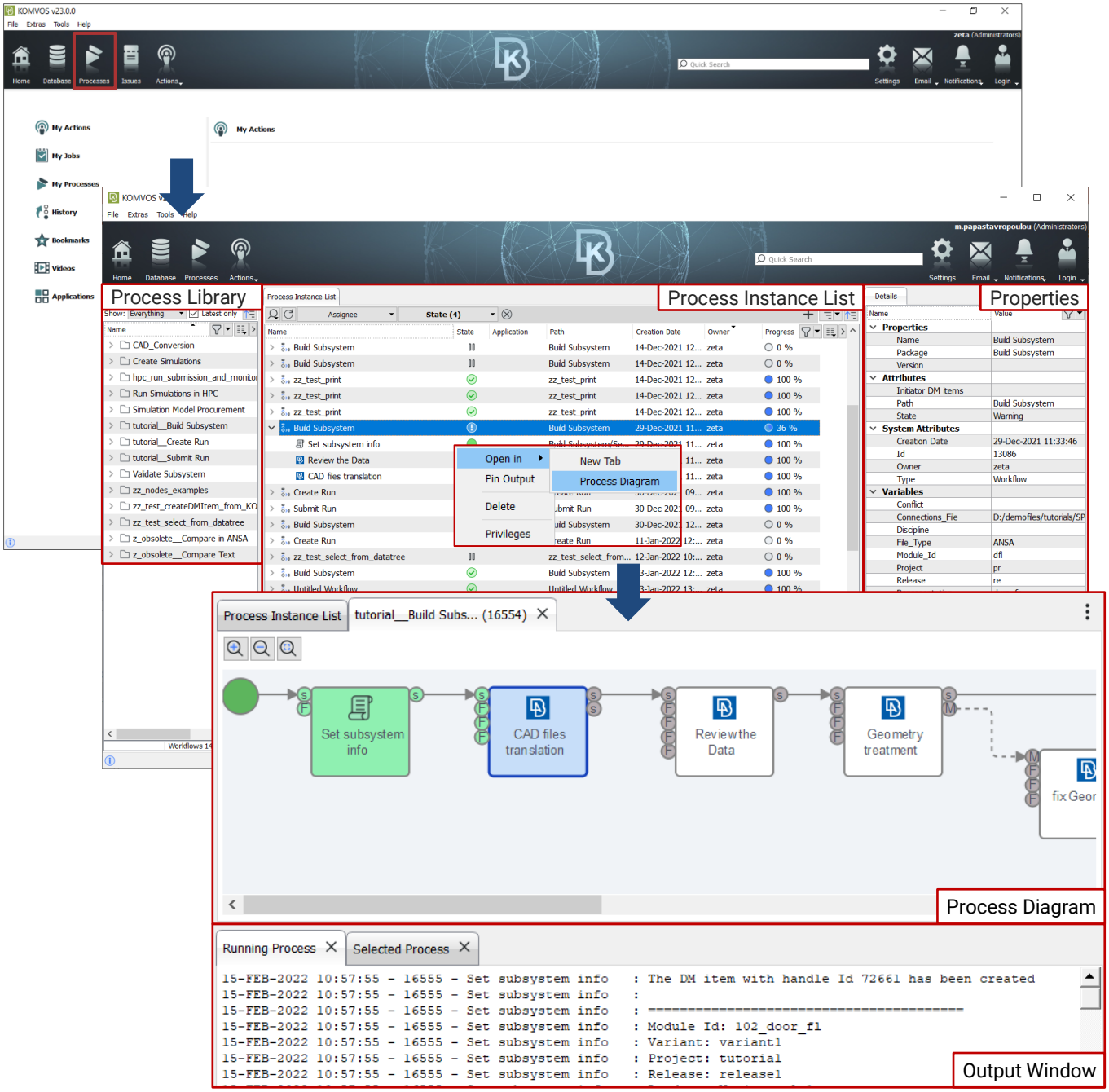
Process Design is supported from KOMVOS version 23.0.0 and is only available when KOMVOS is connected to an SPDRM back-end of version 1.8.0 and later.

KOMVOS version 23.0.0 supports all features of SPDRM processes during process execution. However, when it comes to process design, it supports the most frequently used features but not all the functionality supported by SPDRM. Therefore, there are certain process features that cannot be created in KOMVOS and still require the use of the SPDRM Admin Console (e.g. sub-process nodes, task scheduling, etc.). The handling of such process features is described in the Process Execution paragraph, as they can be encountered in processes created in the SPDRM Admin Console, but is missing from the Process Design paragraph.

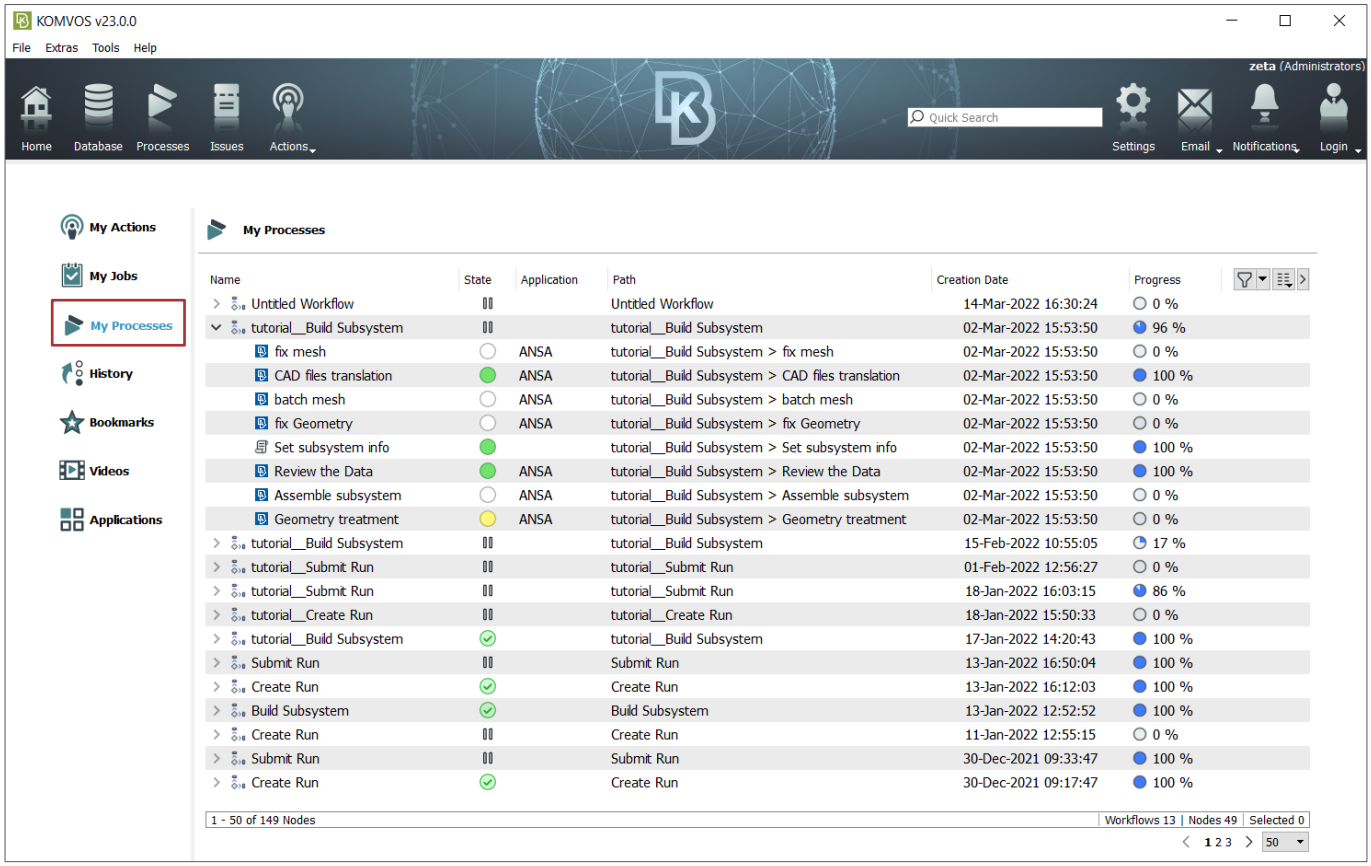
5.1. Introduction to the Process Workspace

Users can access the Process Workspace through the **Processes** button of the main toolbar. Within this workspace the *Process Library* with all the process templates is docked on the left side, the *Process Instance List* with running or previously executed processes is located in the middle, and the *Details* list, displaying all the information about the selected process item is docked on the right. The bottom area is where the *Output Window* resides, which shows the process output messages.

The middle area, where the *Process Instance List* is shown by default, is also used to display processes in the *Process Diagram*.



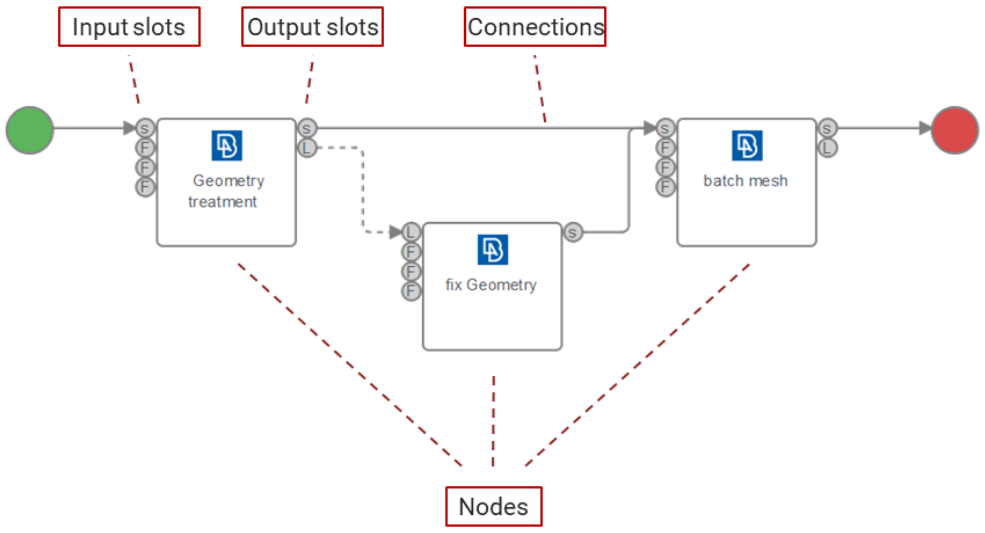
A pre-filtered Process Instance List, containing only the tasks assigned to the logged-in user, can be directly accessed through the **My Processes** panel in the **Home** Workspace.



5.1.1. Process components

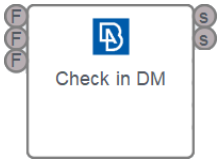
A process is a group of nodes (representing individual tasks), that are connected with each other forming a workflow. Nodes can contain other nodes and form a sub-process.

The image below highlights the main elements of a process.

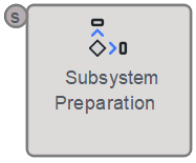


There are several types of nodes, each serving a different purpose.

Script Nodes execute Python scripts (SPDRM Script API).

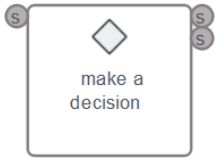


Application Nodes execute a pre-registered application that can be ANSA, META, EPILYSIS or any 3rd party application.



Sub-Process Nodes group a set of actions under a single node. Their contained nodes exchange data with the parent process through output and input streams.

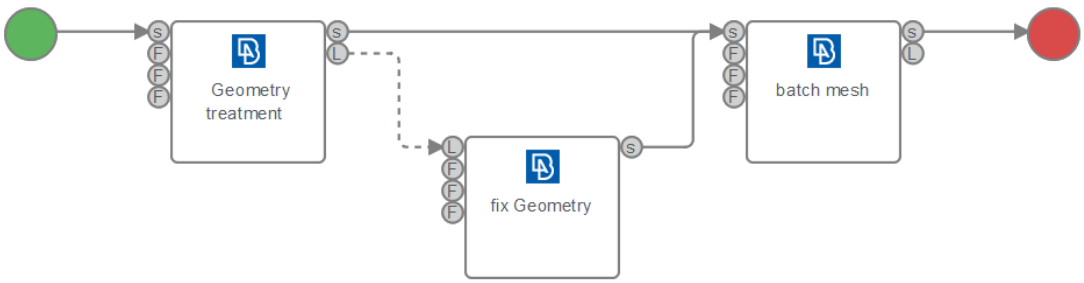
MxN Nodes are special type of sub-process nodes. They're given as input a list of *M* files and they execute the same sub-process *M* times (with a different input file each time), optionally delegated to N different resources. Sub-processes may be executed in parallel, in batches, or sequentially.



Decision Nodes are used to drive the process choosing among two possible paths, based on a condition statement. The node fills one of its output slots based on whether the statement is evaluated to True or False.

Observer Nodes are used for job submission and monitoring on HPC systems. There are two types of observer nodes, the **Script** and the **Application Observer Nodes**. Both these types submit a job, either through Script or by calling an Application, register their interest for a particular job id and then get in an idle mode, until the respective job shows some progress.

Nodes are connected with each other through their input and output slots, forming a workflow. Every node can have one or more input and output slots, and nodes can be connected with one or more connection lines. The input/output slots carry information of certain types. The supported slot types are **String**, **Float**, **Integer**, **Boolean**, **Date**, **Single file**, and **List of files**. Connection lines can only be created between compatible slot types. The connection lines between nodes are represented with dashed lines when they carry **Files**, and with continuous lines in all other cases.



An input slot can have a default value or can get its value from a connection with another slot. In all cases, however, in order for a node to become ready for execution, all its input slots must be filled (i.e. have a value) at run time (unless they are marked as *optional* during process design).

During the execution of a process the nodes and slots go through certain states. A default color map is in place, that allows the user to quickly and easily understand the state of nodes and input and output slots. As the state of slots and nodes changes during the execution, their color also changes, reflecting the transition from one state to another. In the table below, all possible states and colors of nodes are presented.

Basic node states

Not ready: Input Slots do not have a value. Node cannot be executed.

Ready: Input Slots are filled. Node is ready to be executed.

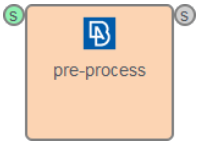
Running: Node is currently being executed.

Finished: Node has finished its execution.

Observing: Node is in observing state and receives information about the progress of the submitted jobs from the server. “Submitted Jobs” option of the context menu provides this information.

Additional node states

Scheduled: Input Slot data are available. Node is ready to be executed but it is scheduled to start at a particular time in the future.



Paused: The node has been paused by the user (only for application nodes).

Pre-finished: Node is completed but requires confirmation by the user, appears only when the “Auto Complete” option is deactivated (Node’s Properties).

Polling: Input slot of the node is polling for its input data.

Input and output slots also follow a certain color coding to represent their states during the execution of the process. These possible states are presented below:

Input Slots:

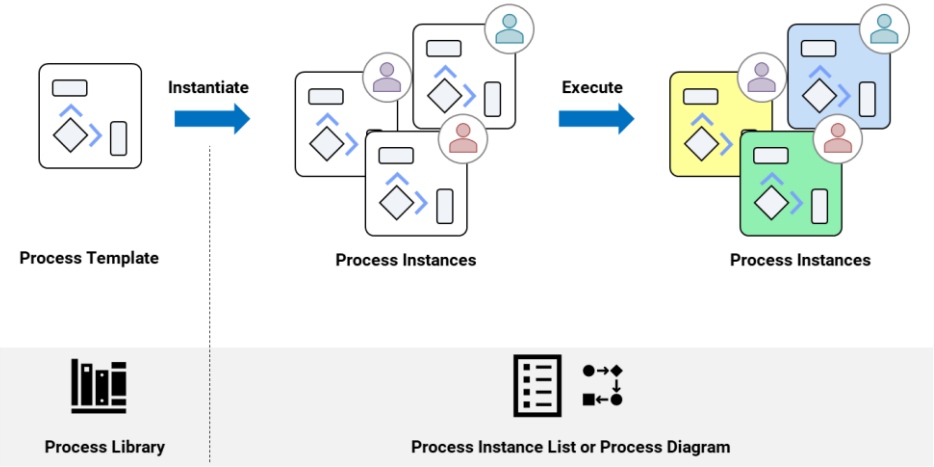
- Does not have a value (Obligatory slot) - Node cannot start
- Does not have a value (Optional slot) - Node can start
- Has a value - Node can start
- Polling for a particular file - Node cannot start

Output Slots:

- Does not have a value yet
- Has a value but its data have not been transferred to the connected input slot yet
- Has a value and its data have been released to the connected input slot – if any

5.2. Process Execution

The lifecycle of a process is shown in the image below:

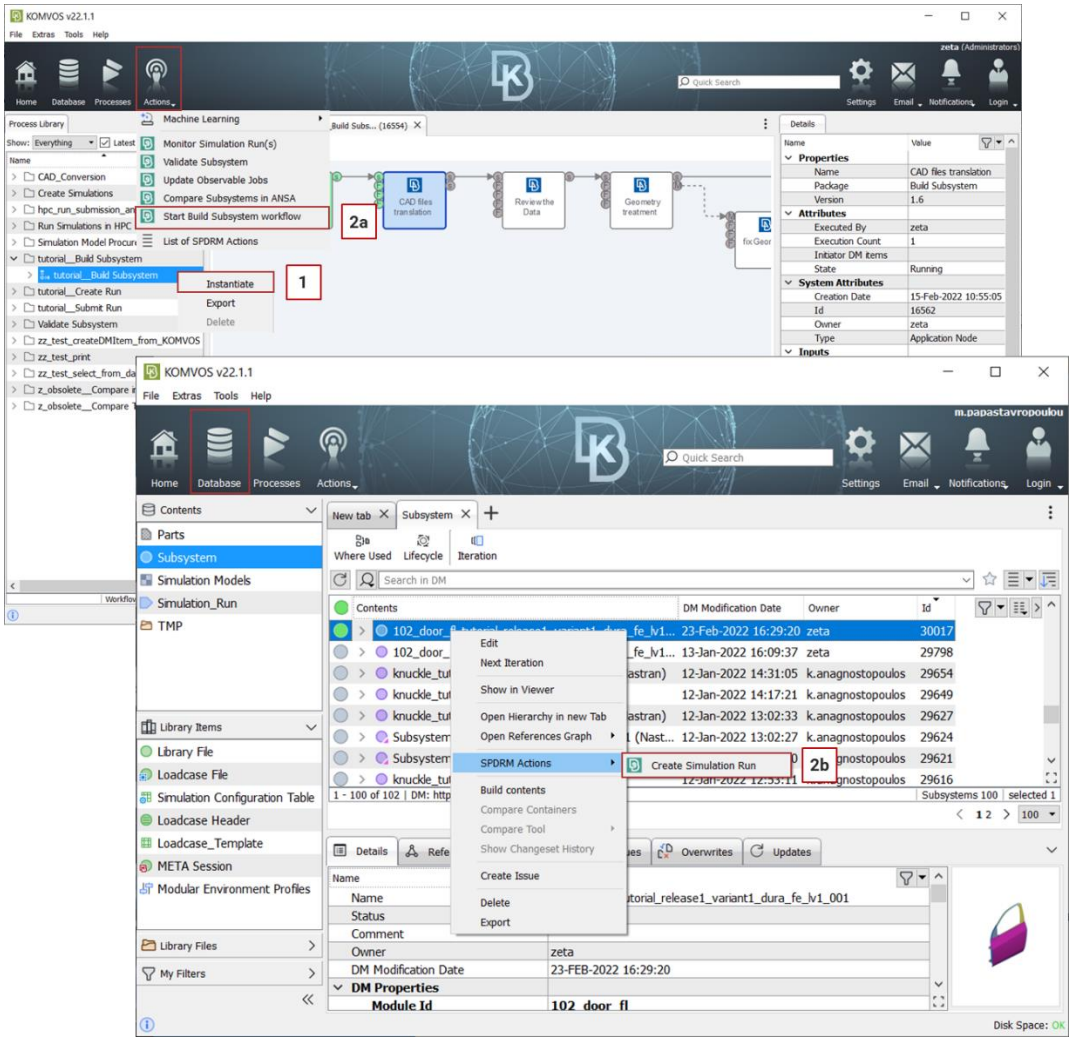


- A process always starts as a **template**, from the *Process Library*. Process Templates are also referred to as Process Definitions.
- From the Process Library, a template is *instantiated* manually or through a script, creating a **process instance**. A process instance always has an executing user. Process instances can be seen in a list view in the *Process Instance List* or in a diagram view in the *Process Diagram*.
- Process nodes are *executed*. Execution can be triggered through the *Process Instance List* or through the *Process Diagram View*.
- Usually Process Instantiation is followed by automatic process execution.

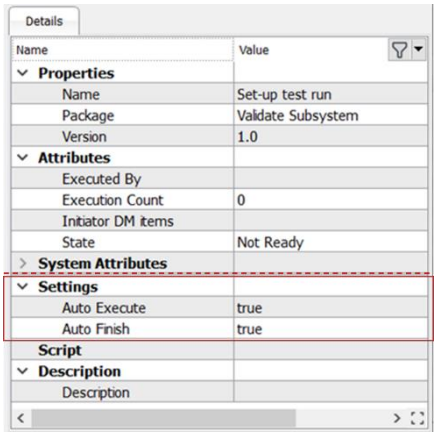
5.2.1. Starting a process

A process in KOMVOS can be started:

1. Through the **Process Library**, with the *Instantiate* option of the context menu
2. Through an SPDRM Script Action:
    a. Generic or
    b. DM object type-specific

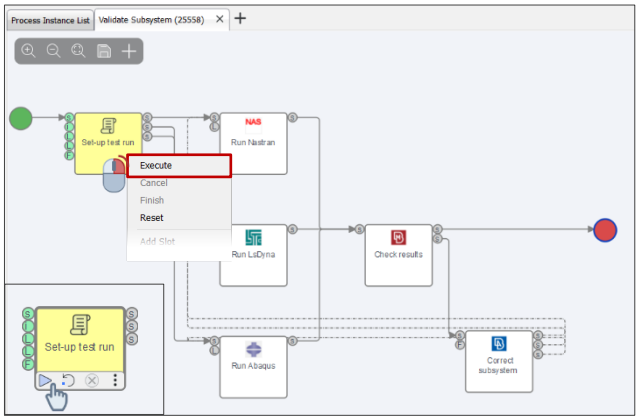


Both these options create an instance of the process template which is visible in the Process Instance List and also in the Process Diagram.

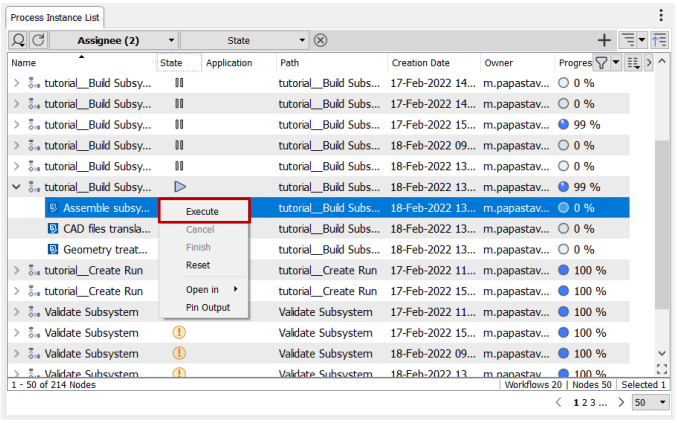


The execution of an instantiated process can be either automatic or manual, depending on the **Auto Execute** and **Auto Finish** settings of each process node, as defined in the node *Properties* window.

If a node is not set to be automatically executed when a process is instantiated, the user can manually **Execute** it by selecting the respective option from the node context menu, provided that the node is ready to be executed.



Node Execution in Process Diagram

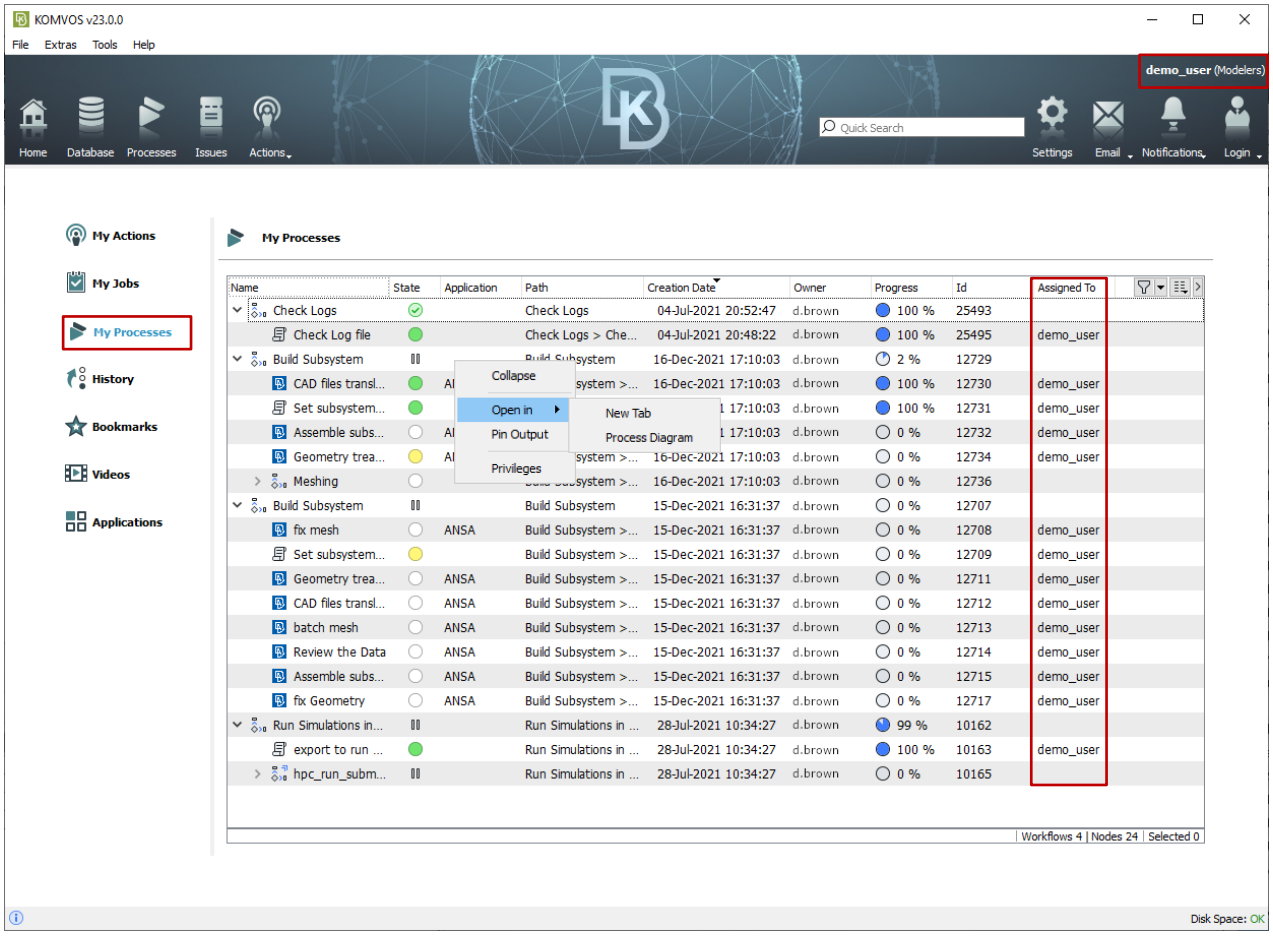


Node Executuion in the Process Instance List

5.2.2. Finding processes of interest

5.2.2.1. “My Processes” tab

The easiest way for users to identify their assignments is directly through the **My Processes** tab of the **Home** workspace. This tab offers a list of processes pre-filtered according to the current logged-in user, so that only processes instantiated by (i.e. owned by) or assigned to the current user are displayed.

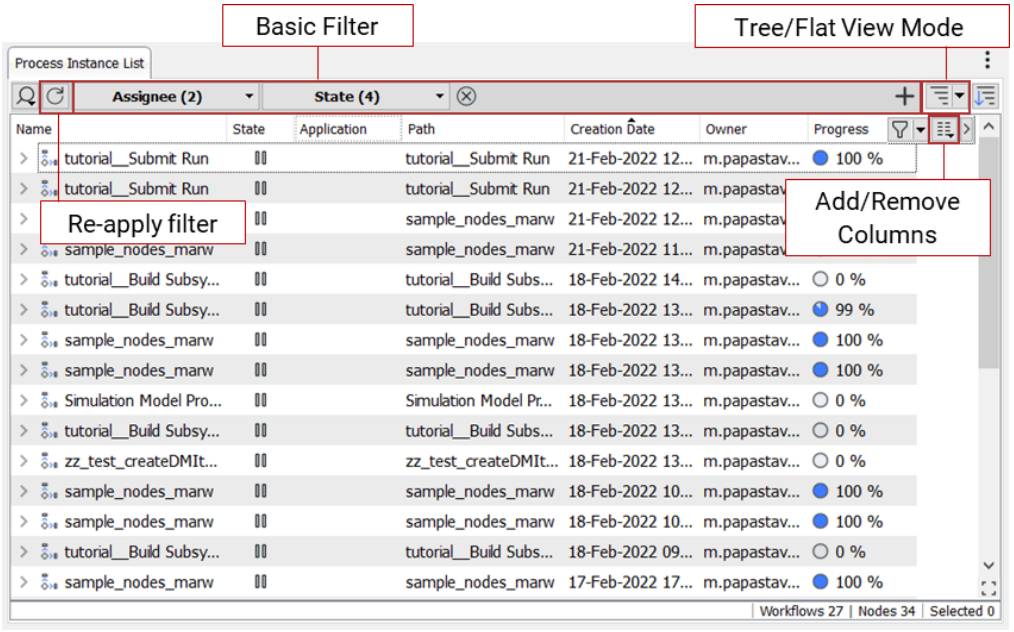


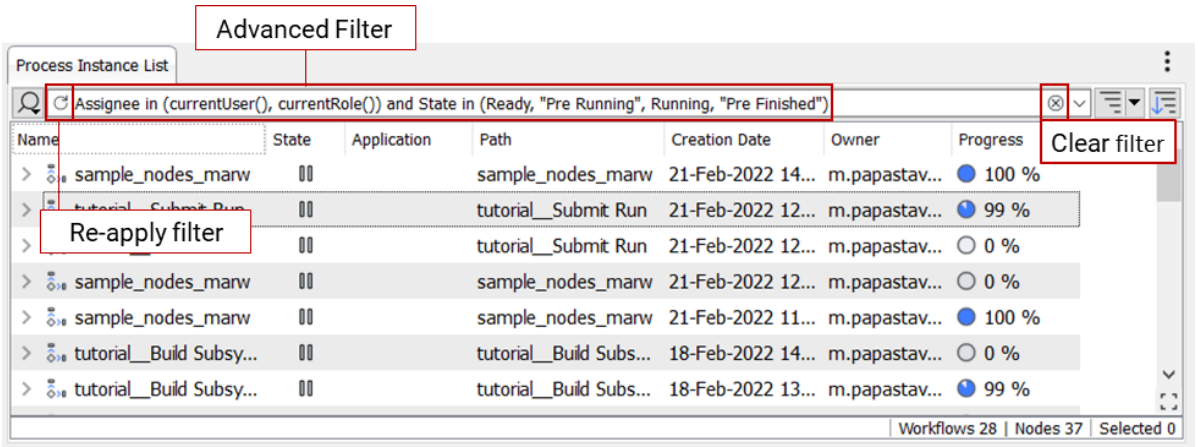
Through this list, processes can be opened in the Process Diagram or can be isolated in new Process Instance List tabs.

5.2.2.2. Process Instance List

All *live* process instances visible to the logged-in user are listed in the *Process Instance List* of the **Processes** workspace. By default, the *Process Instance List* opens with an applied filter that isolates the node instances assigned to the **Current user & role**, being in one of the following **States**: *Ready, Pre-running, Running, Pre-finished*. This is the default filter that will be active whenever the *Process Instance List* is opened for the first time in a KOMVOS session. In addition, the instances are listed by default from newest to oldest.

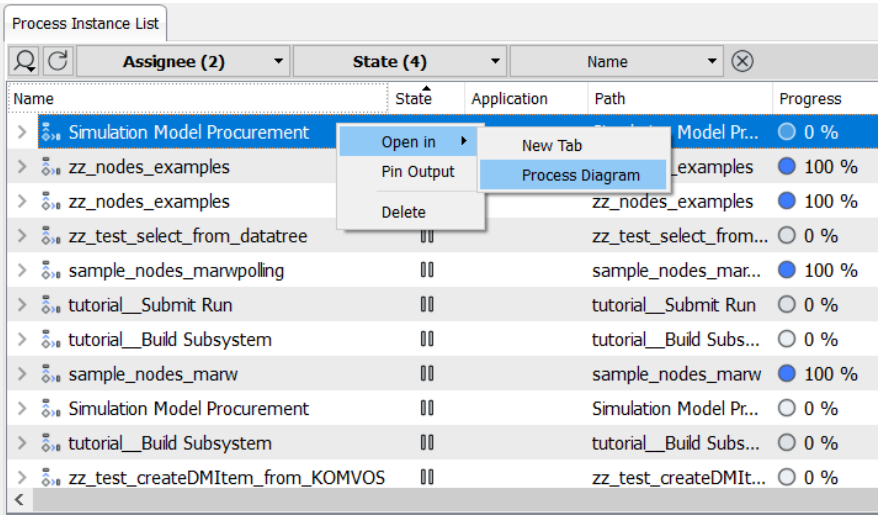
Furthermore, in the default layout, process instances are presented in a tree view, representing the process structure. Some key list manipulation functionality is shown below:





The filtering is facilitated by the same mechanism that is used in the Database workspace with the Dual Filter. (*Dual* as filters can be defined using two modes, the **Basic** and the **Advanced**). In the heart of the filtering mechanism lies the powerful BETA query language (BETA QL), that enables the easy definition of advanced queries. More information on the operation of the Dual filter and the BETA QL can be found in chapter 3.5.2.1 of the Data Management Reference Manual.

Once the process or node of interest is identified, it can be executed directly or it can be opened in the Process Diagram.



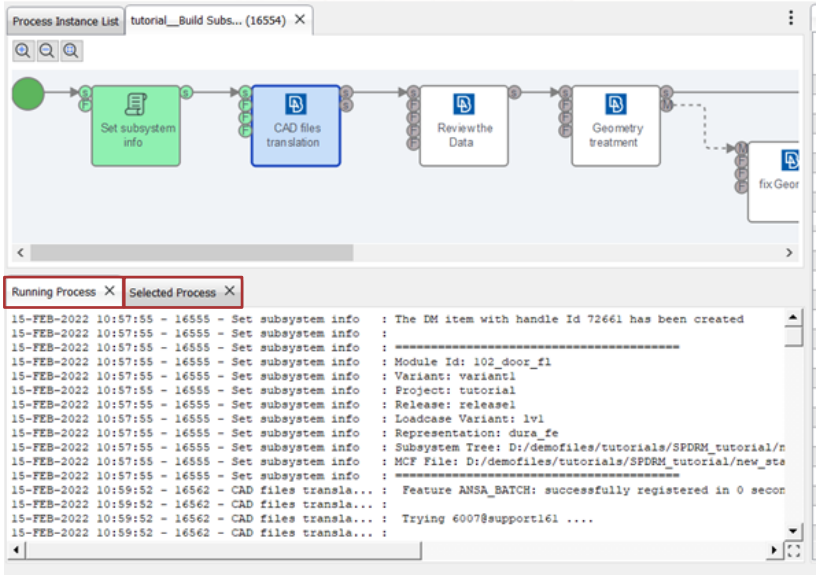
5.2.3. Runtime topics

This paragraph describes certain topics related to process runtime. Specifically:

- **Process Output Messages**: Information on where and how the executing user can access the output of a node
- **Node Execution Actions**: Information on the actions available on each node type, depending on the node state
- **Notifications**: Introduction of KOMVOS' Notification Center that accommodates all process-related notifications
- **Node Execution Directory**: Process related file management concepts

5.2.3.1. Process output messages

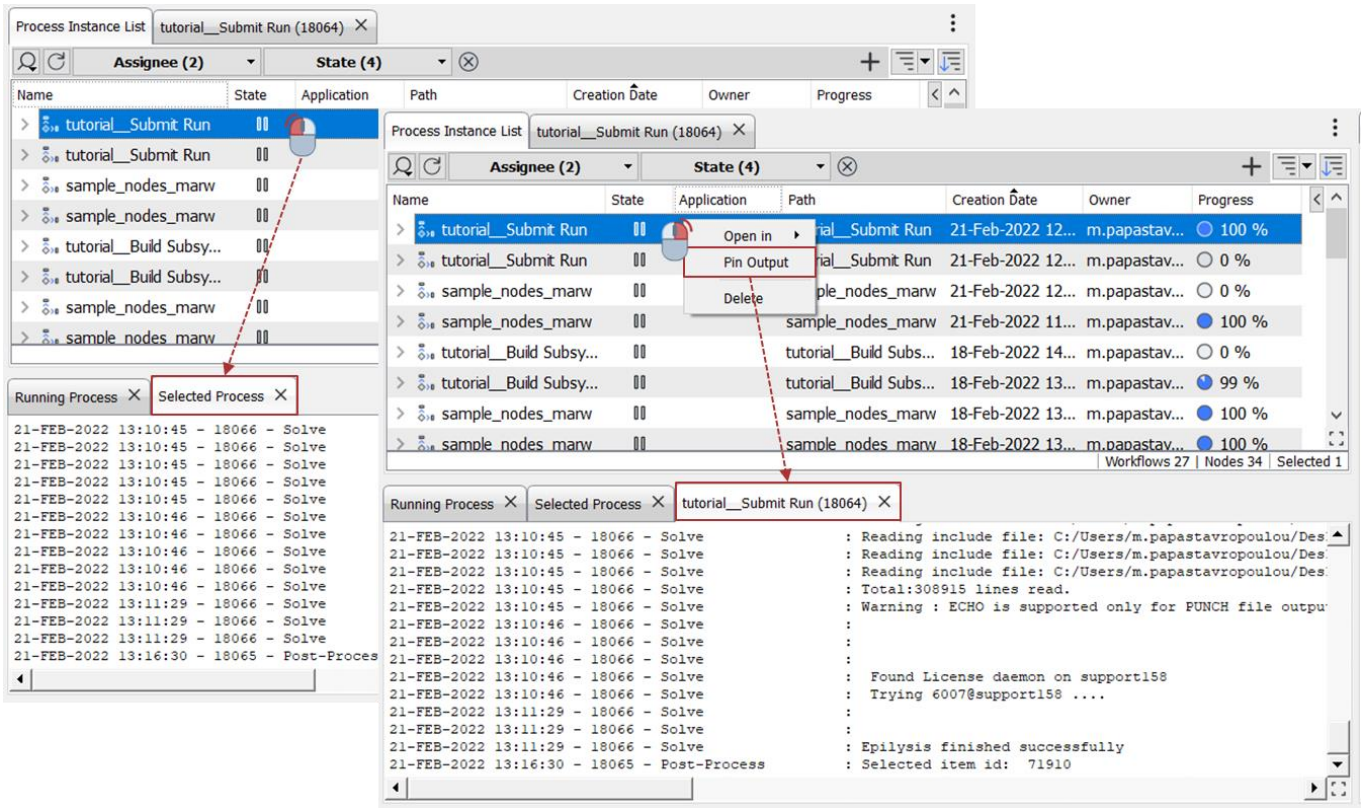
The output of all applications and scripts that run through the process is displayed in the *Output* window that appears by default as a tab at the bottom of the KOMVOS Processes workspace. The Output is made available in two different tabs:



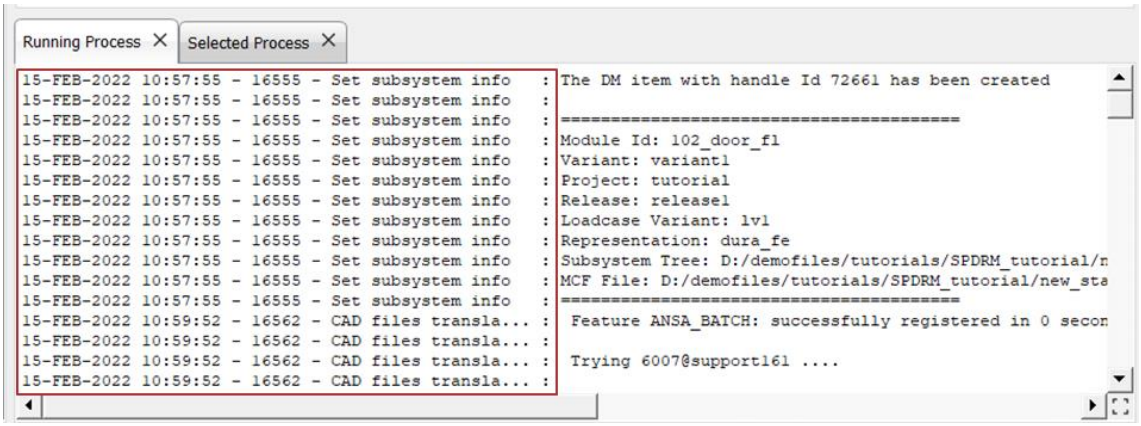
Selected Process: This tab is in sync with the selected node or sub-process, both in the *Process Diagram* and in the *Process Instance List*. It shows the logs that have been produced by the selected node or sub-process. Selecting a whole process in the Process Diagram or in the Process Instance List, all node outputs are collected and displayed in the Selected Process tab, in order of execution.

Running Process: This tab shows all the logs that were produced from any process since the start of KOMVOS session.

The option **Pin Output** can be used to create a new pinned tab, that will display the output of the selected process as *static* content, i.e. not in sync with the selected item.



Every line of output is prepended with a standard prefix consisting of *the execution time*, the *node id* and the *node name*. This is particularly useful in the Running Process tab, where different processes are running simultaneously, as successive output messages may come from different sources.



Note !

The logs of a Node are persistently stored in the executing user's home folder, following a hard-coded rotation policy. Due to this functionality, logs are not erased every time the user logs off. Selecting a previously executed node will still display its previous logs in the "Selected Process" tab.

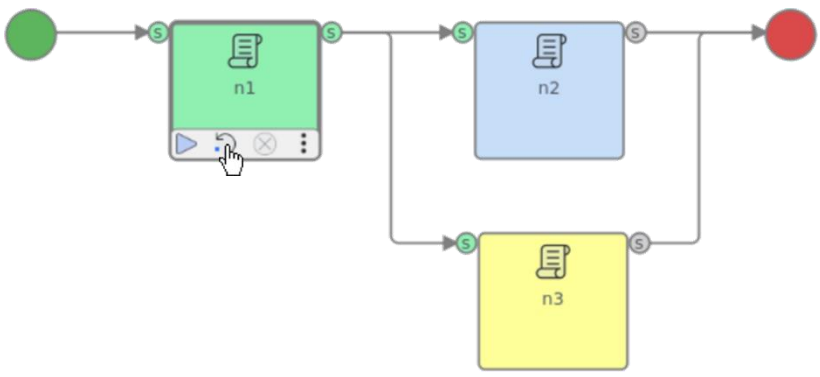
5.2.3.2. Node execution actions

The table below summarizes the available execution options per node type and state:

Action Name	Node Type	Node State	Description
Execute	All	Ready	This action executes a process node.
Finish	All	Pre-finished	In case the node was not marked as “auto-complete”, after its successful execution it will be set to Pre-finished state. While in this state, the node has not “released” its output slots, i.e. the values of its output slots have not been pushed to any input slots they connect to. When **Finish** is pressed, node turns into Finished state and output slots get released.
Cancel	All	Running Observing	In case the node execution takes too long or is expected to yield wrong results, the executing user can Cancel the execution. For a script node, this will halt script execution, for an application node it will send a termination signal to the application and finally for an Observing node, it will execute its cancellation script.
Reset	All	All	No matter the current state of a node, Reset will try to set it to a “ready” state, i.e. validate its input slots. In case the node is in a Running state, Reset will first Cancel the node and then Reset.
Pause	Application Node	Running	In cases of Application Nodes, Pause can be used to signify that when the application is closed, the node should not start executing its post-run script but, instead, should wait for manual Resume
Resume	Application Node	Paused	In cases of Application Nodes that were Paused, Resume will put the node back into Running mode.

5.2.3.3. Node cancellation requests

A cancellation request for a running node can be raised due to the actions *Reset/Execute* applied on a former *Finished* node. The response of KOMVOS to this action will differ, depending on the executors involved.

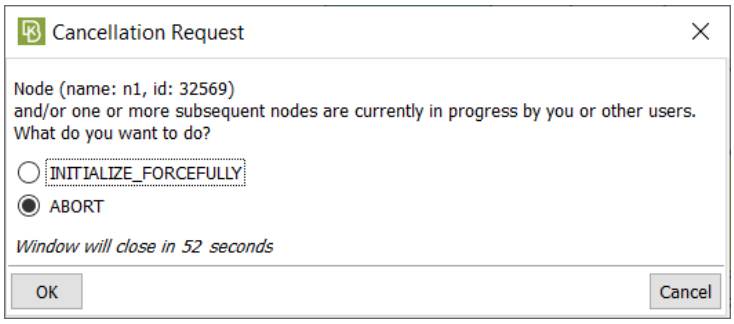


Let’s assume the example process shown on the left. Node *n1* has been executed and it is now finished. Node *n2* is at a running state. *User_A* who is the executor of node *n1*, applies Execute or Reset action on node *n1*.

There are two different responses expected, depending on the executor of the running node *n2*.

Case 1 – Executor of n2 is the same user, User_A

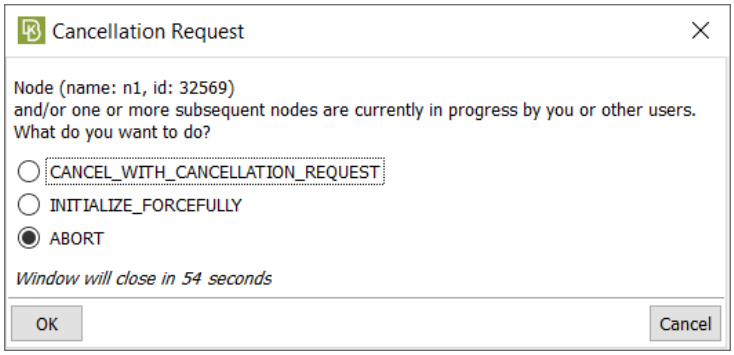
A warning message appears with two options:



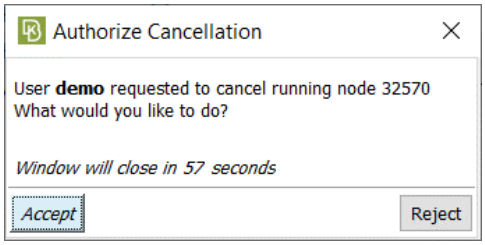
- *INITIALIZE_FORCEFULLY*, will first *Cancel* the running node *n2* and then apply the *Execute/Reset* action on node *n1*. (A notification is received about *n2* node that was cancelled).
- *ABORT*, will cancel the request.

Case 2 – Executor of n2 is another user, User_B

A warning message appears to *User_A* with three options this time:



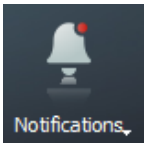
- *ABORT*, will cancel the request.
- *INITIALIZE_FORCEFULLY*, will apply the *Execute/Reset* action on node *n1* independently of node *n2* (node *n2* will continue its execution).
- *CANCEL_WITH_CANCELLATION_REQUEST*, will send a cancel request to *User_B* and wait for user's response. *User_B* is informed through a relevant message and can accept or reject the cancellation request.



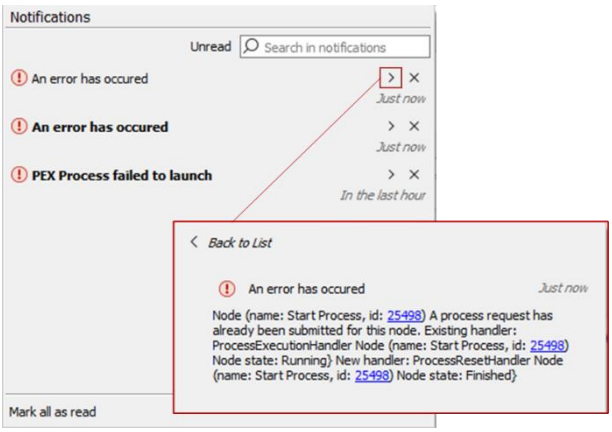
In case *User_B* selects the **Reject** option (or takes no action and the window is closed), node *n2* will continue running, and *n1* will remain as-is. A relative notification will pop up at *User_A* informing that the node could not be cancelled.

5.2.3.4. Notifications in KOMVOS

As several processes may be executed in parallel and asynchronously, important notifications that relate to the process execution are hard to deliver to the user in a timely manner through the Process Output window. For this reason, KOMVOS offers a Notifications Panel that provides an overview of alerts from processes and other asynchronous tasks.



Notifications in KOMVOS come in 3 severity levels: **Information**, which are marked with blue color, **Errors**, that are marked with red and **Warnings**, that are marked with orange. Information and Warning notifications do not require an immediate action. Error notifications usually do. The Notifications button in the toolbar shows a colored dot, according to the most critical unread notification.



Within the Notifications Panel the user can view collected notifications, acknowledge and delete them.

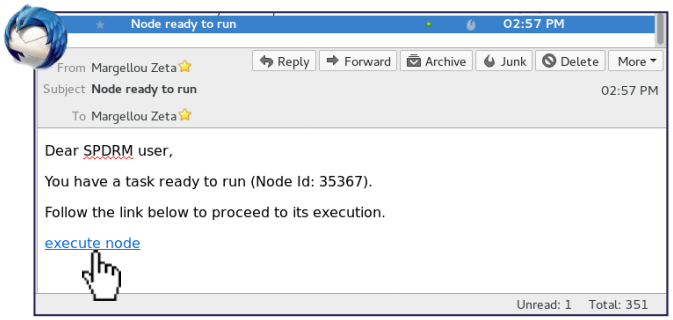
The description of a notification may contain hyperlinks that can redirect the user to the entity concerned (e.g. to the respective process node in the process instance list).

Right now, the Process engine sends notifications:

- When the logged-in user is assigned a new task
- In case Errors occurred during the process execution

5.2.3.5. Notifications through e-mail

A node may be configured such that an e-mail is sent to its assignee the moment it gets in *Ready* state.



The body text by the e-mail is configured by the process designer. It may contain a hyperlink that can be used to open KOMVOS and identify the respective node in the Process Instance List.

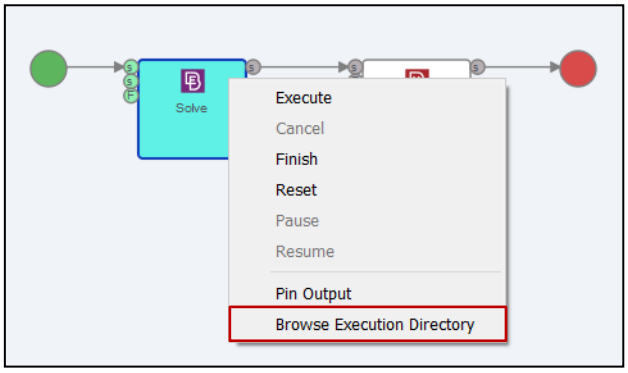
5.2.3.6. The Node Execution Directory

During the execution of a node, SPDRM creates a unique, temporary directory, to facilitate any data exchanges related to the node. This directory is called Node Execution Directory (or NodeExec).

- The files related to input slots of type **Single file** or **List of files** are made available within this directory.
- This directory is used as the CWD of launched Python scripts or external applications.
- In order for some files produced by the Node Script or by the External Application to be transmitted to the next node via an output slot-input slot connection, they must initially reside in this directory.
- A new Node Execution Directory is created for every execution of the node, i.e. even after resetting.

The Node Execution Directory is automatically created when the execution of a node starts and is automatically deleted when the node is finished and all of its output slots have been released.

Node Lifecycle	Not ready → Ready → Running → Paused → Finished (set info)			
State of the Node Execution Directory	Does not exist	Is created	Is retained	Is deleted

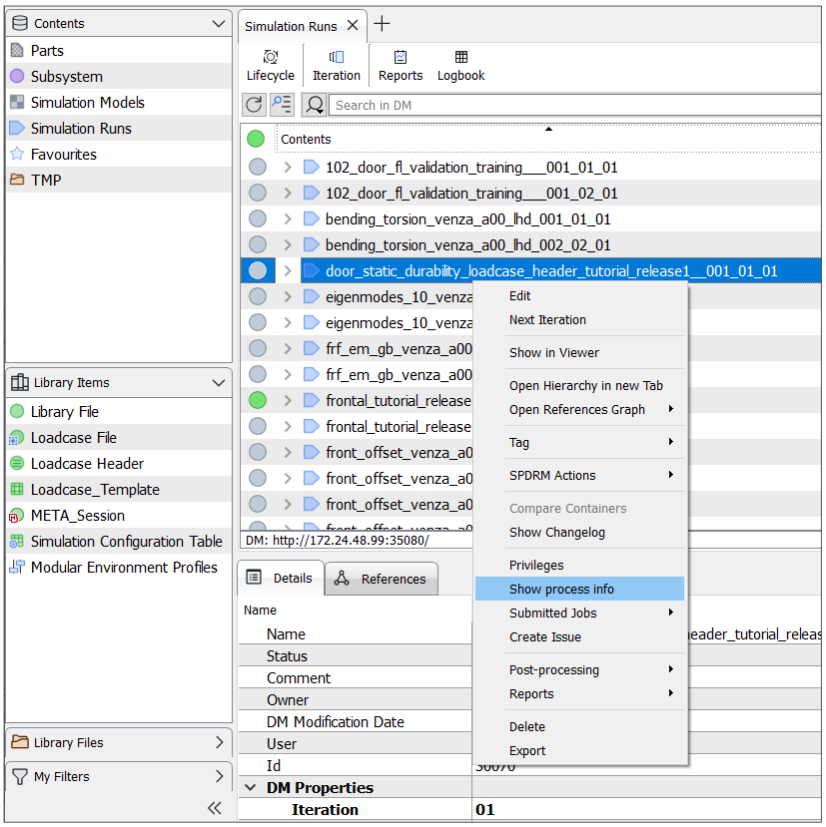


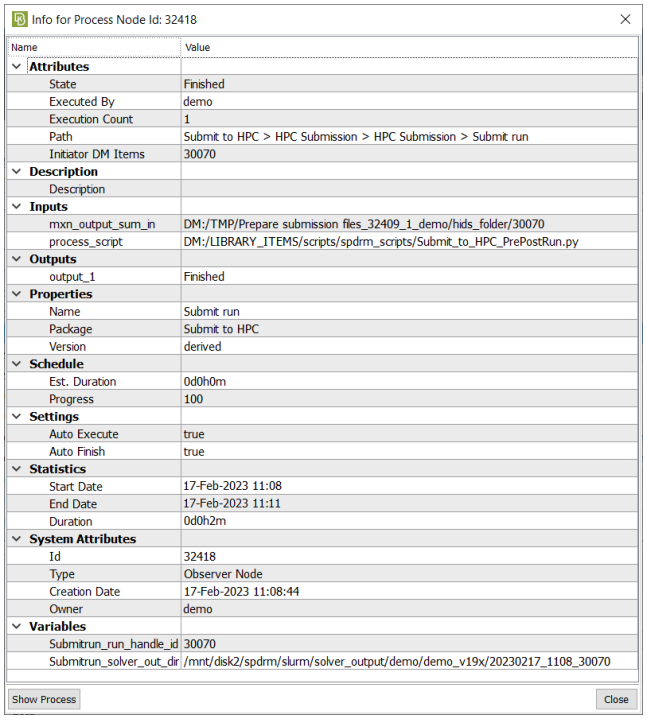
While a node is in Running, Paused or Pre-Finished state, its Node Execution Directory is accessible through the context menu option **Browse Execution Directory**.

5.2.3.7. Process information on data

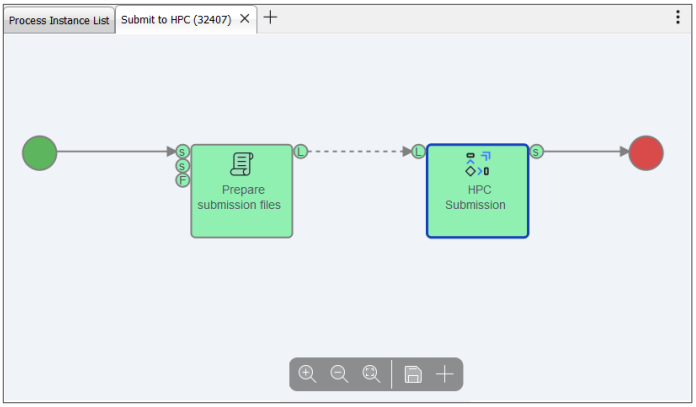
The data produced by a process, carry information about it and retain a relationship with the process instance until this is deleted.

Activating the option **Show process info** of the context menu of a DM Object, the *Info of Process* window appears.





This window provides information for all the attributes of the node that produced the selected DM Object during its execution. Furthermore, pressing the button **Show Process**, the user is directed to the *Processes* workspace, where the particular process opens in a new *Process Diagram*.

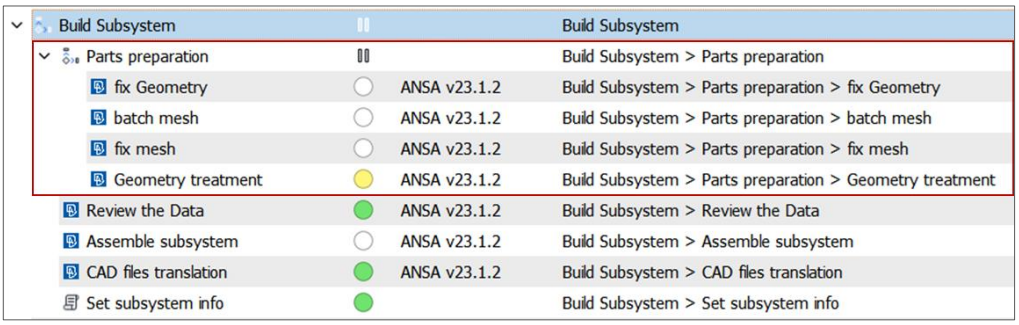


5.2.4. Execution of special node types

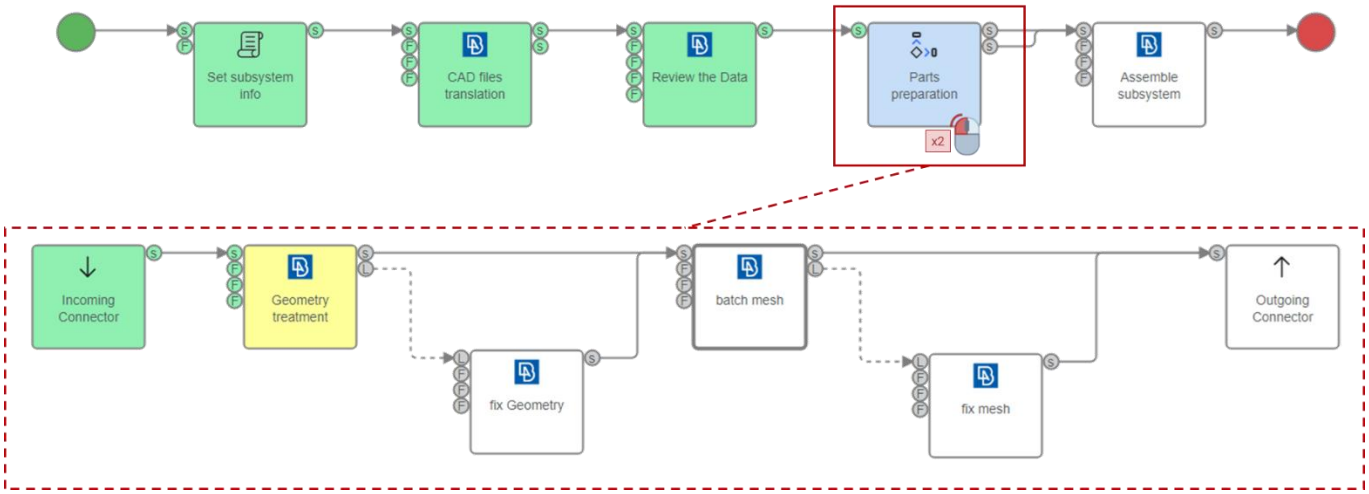
5.2.4.1. Sub-processes

Sub-process nodes can group a set of actions under a single node. Their contained nodes exchange data with the parent process through output and input streams.

A sub-process node can be viewed in the *Process Instance List* as a child item of the root process and it can be further expanded to reveal its contents.

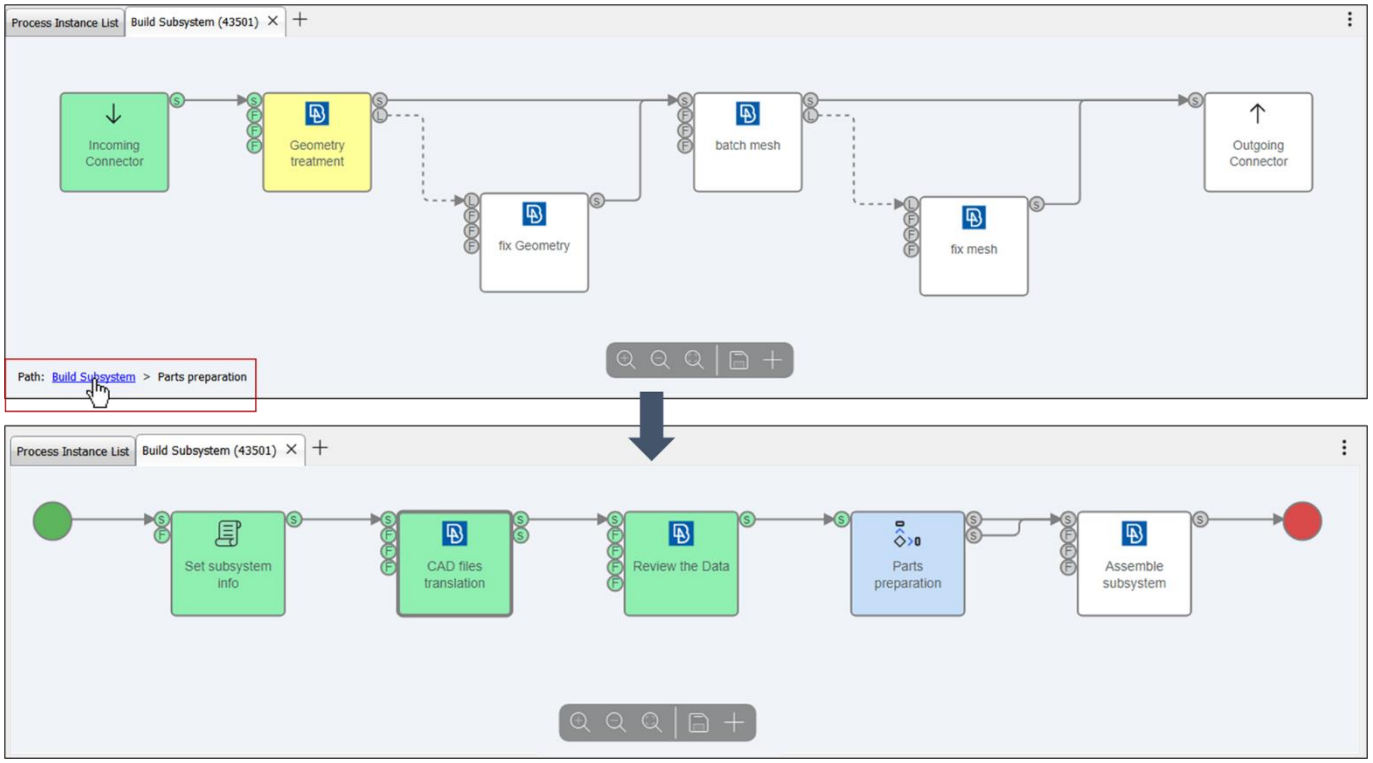


In the Process Diagram, the sub-process can be recognized from its characteristic node icon. From there, the user can be navigated in its contents with double click on the sub-process node.



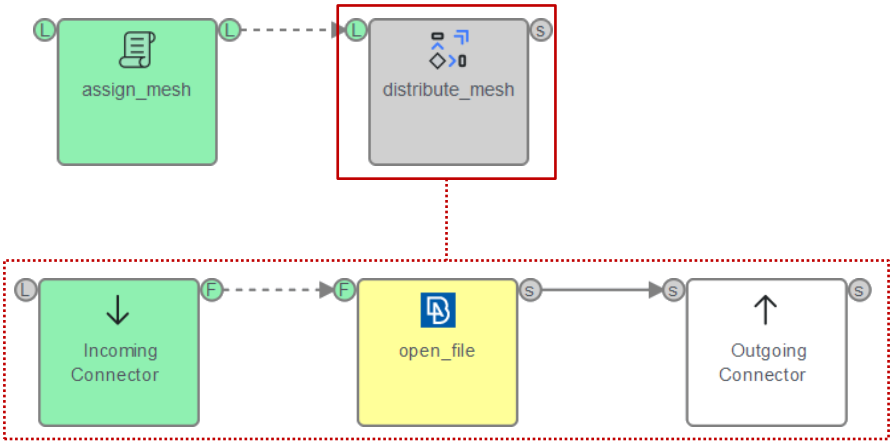
A sub-process will always display among its contents, the input and output streams, as special virtual nodes with names *Incoming* and *Outgoing Connector* respectively. No actions can be applied on these nodes.

Using the breadcrumb trail at the low left corner of the diagram, the user can navigate up at the root level.



5.2.4.2. MxN nodes

MxN nodes are special type of sub-process nodes. They get as input a list of M files and they execute the same sub-process M times (with a different input file each time), optionally delegated to N different resources. The **Input slot** that carries the data set to be distributed is always of type **List of files**.



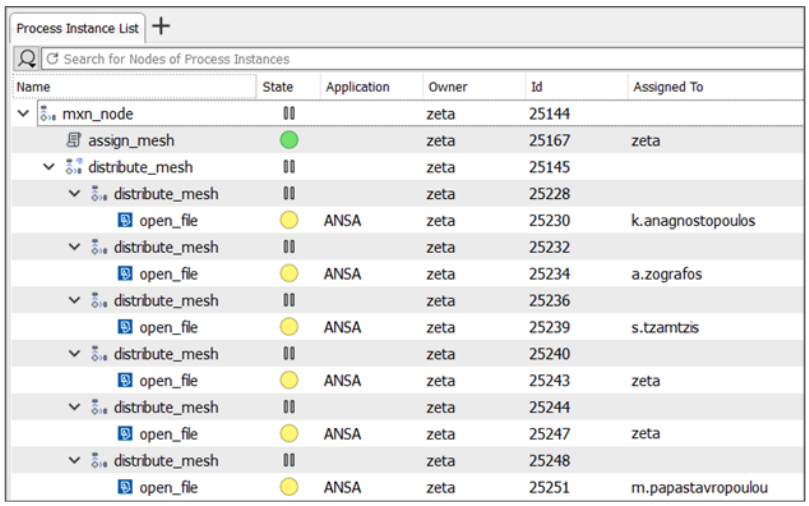
When the MxN node is executed, its containing process is instantiated as many times as the number of files carried by this Input slot. Each of these instances will be provided with one of these files as an input.

The MxN node can handle multiple **List of files** input slots. In such case the input data are arranged as multiple data sets. Thus, it is possible to add a data set of *'k'* files (e.g., *'k'* ANSA files need to be meshed), and another data set with *'j'* other files (e.g, *'j'* ANSA files with one meshing configuration each, to be used for the meshing).

In such case, the data sets can be either added to form a data set which will be the union of the two (*'k+j'* files to be distributed), or they can be combined to form a big data set of *'k*j'* unique combinations of files to be distributed. When executed, the MxN node will instantiate its subprocess accordingly, creating *'k+j'* or *'k*j'* number of instances, respectively.

This whole set-up takes place through the MxN node's settings, during process design.

When the MxN node is executed, it generates several process instances according to the number of inputs and their combination.


Process Instance List

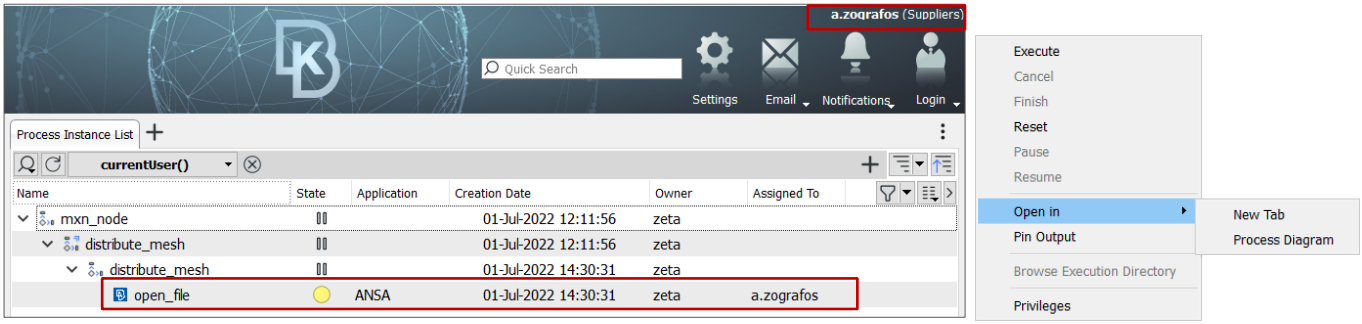
Search for Nodes of Process Instances

Name	State	Application	Owner	Id	Assigned To
mxn_node			zeta	25144	
assign_mesh			zeta	25167	zeta
distribute_mesh			zeta	25145	
distribute_mesh			zeta	25228	
open_file		ANSA	zeta	25230	k.anagnostopoulos
distribute_mesh			zeta	25232	
open_file		ANSA	zeta	25234	a.zografos
distribute_mesh			zeta	25236	
open_file		ANSA	zeta	25239	s.tzamtzis
distribute_mesh			zeta	25240	
open_file		ANSA	zeta	25243	zeta
distribute_mesh			zeta	25244	
open_file		ANSA	zeta	25247	zeta
distribute_mesh			zeta	25248	
open_file		ANSA	zeta	25251	m.papastavropoulou

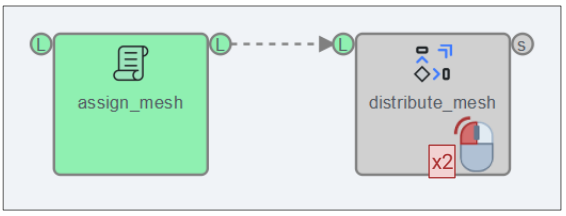
The produced instances can be executed either in parallel or sequentially. When executed in parallel, a max batch size can be defined, in order to prevent saturation of resources.

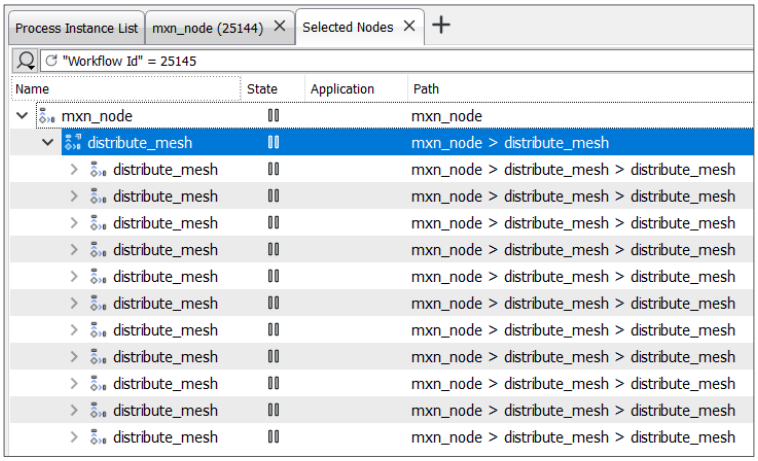
Additionally, they can be generated assigned to a specific user/role (as defined during the design of the MxN node), or they can be distributed to the users of a role.

Each of these users, when connected to the system, will be able to execute their assigned tasks, directly through the *Process Instance List* or by opening them in the *Process Diagram*.



In order for a single user to handle all MxN instances, double click on the MxN node in the *Process Diagram* will open all its instances in a new *Process Instance List* tab.



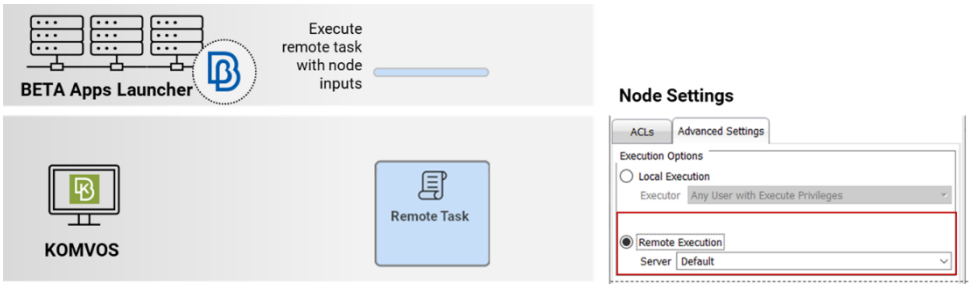


5.2.5. Execute tasks on remote resources

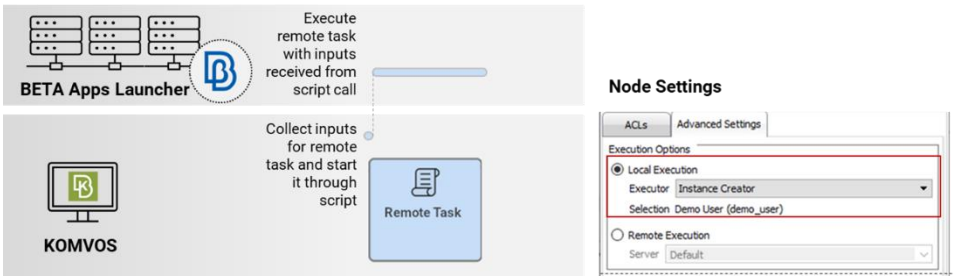
Process nodes can be executed on remote resources with the aid of the BETA Apps Launcher. This is particularly useful for the execution of “heavy” and resource-consuming tasks (e.g. ML Predictors training, CAD Translation, Meshing), or task that naturally need to be executed remotely (e.g. HPC submission on a remote server in another site).

Use of remote resources can be achieved in any of the following ways, according to the settings defined in the **Privileges** window opened through the context menu:

- **For all node types**: If the node is marked by the process designer for execution on remote resources, when it gets in Running mode it will be executed remotely.

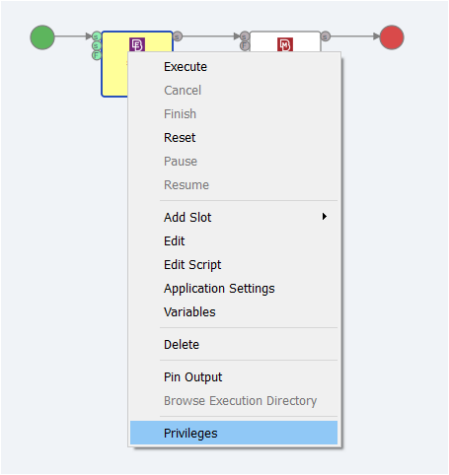


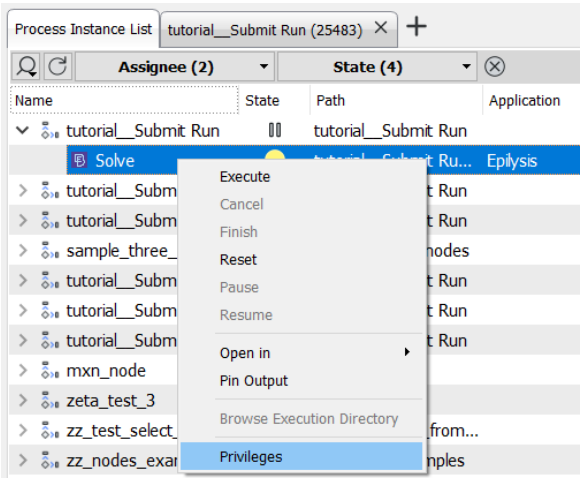
- **For script nodes**: Even if the node is not marked for execution on remote resources, its script may request the execution of tasks on the BETA Apps Launcher



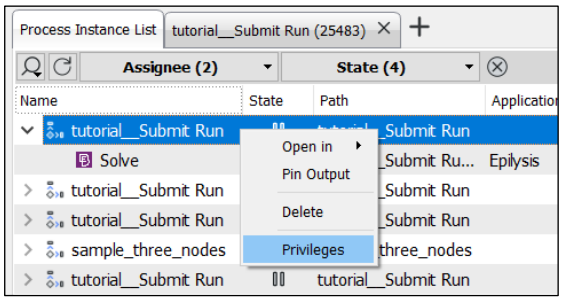
5.2.6. Privileges

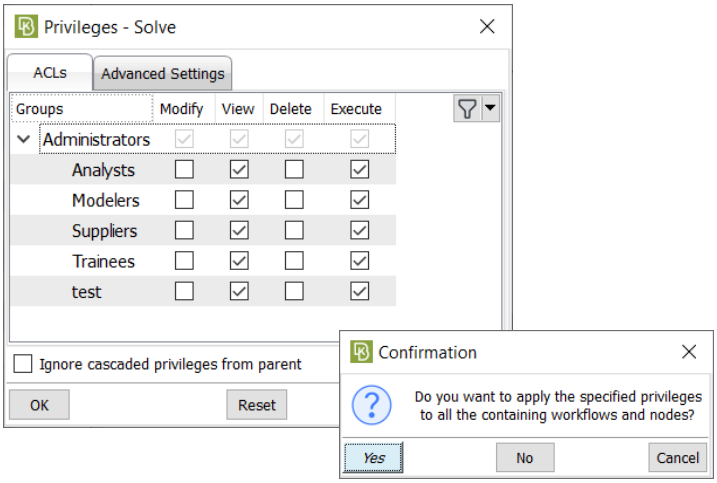
In Process Management, privileges on nodes are set with the aid of Access Control Lists (ACL) which essentially control which user roles will be able to View, Modify, Delete or Execute particular nodes. To access the *Privileges* window, select the **Privileges** option from the context menu of a node, eithr through the *Process Diagram* or through the *Process Instace List*.

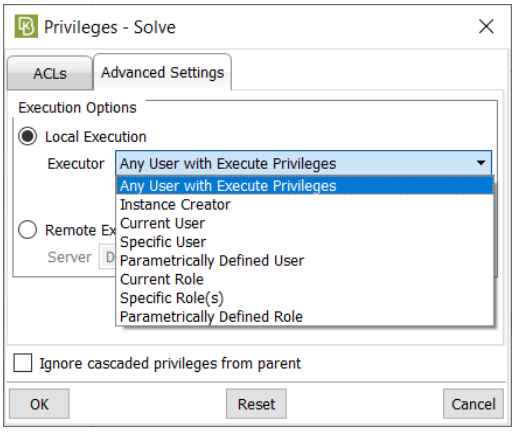




In case the user selects to set the privileges of top-level process or sub-process, a confirmation window will pop-up asking if the selected privileges should be cascaded to all child nodes and workflows.





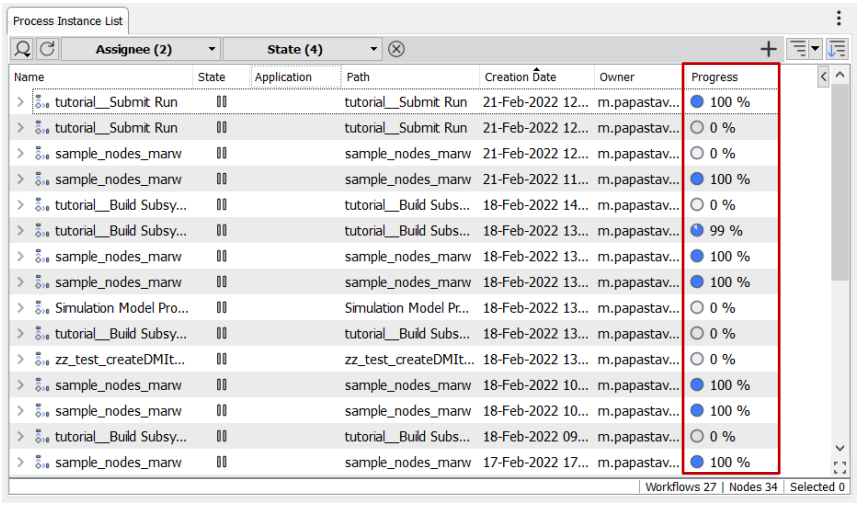


In the *Advanced Settings* tab the user can choose between a **Local Exexution** or a **Remote Execution** utilizing the BETA Apps Launcher (see paragraph 0). For the Local execution there are several options for the definition of the executing user of the node/process.

The **Ignore cascaded privileges from parent** option protects the privileges of the node from another set of privileges that might be pushed to the node while cascading privileges from a parent process.

5.3. Progress Monitoring

KOMVOS offers progress monitoring functionality from the *My Processes* tab in the Home workspace and from the *Process Instance List*. By enabling the **Progress** column, the user can have an overview of the overall progress of the process, as well as the progress of individual nodes.



KOMVOS will automatically calculate the progress of processes and nodes based on the following criteria:

- If the *Estimated Duration* of a node was defined in the Properties card, KOMVOS will calculate the node progress as a fraction of the execution time over the estimated duration.
- If a node has no *Estimated Duration* defined, KOMVOS will automatically set the progress to 99% when the status of the node changes to *Running* and will display progress equal to 100% when the node is *Finished*.
- The progress of a process containing nodes with defined *Estimated Duration* is calculated by KOMVOS using built-in algorithm based on the progress of its nodes and sub-processes.

5.4. Process Design

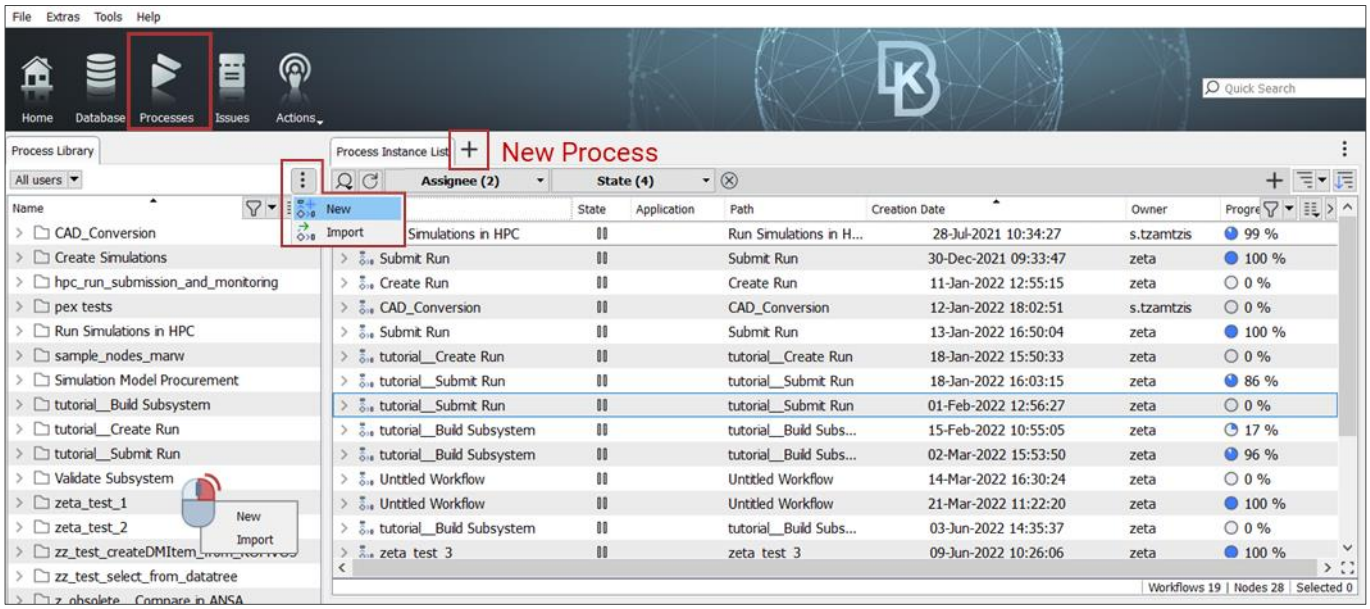
Process Design is supported when KOMVOS is connected to SPDRM back-ends of version 1.8.0 and later. In cases where features described in this paragraph are not available in the user's environment, please check the version of the server through **Help>About KOMVOS**.

The design of a process is implemented in a new *Process Diagram* tab. All designed workflows can be saved in the *Process Library* as **Definitions** (also called **Templates**) or in the *Process Instance List* as **Instances**.

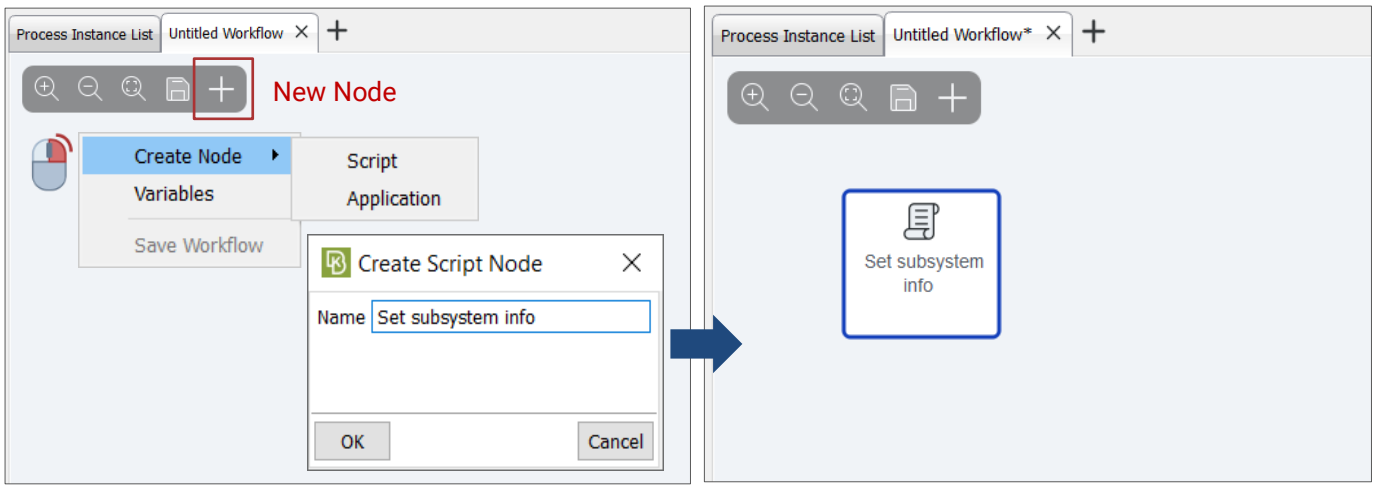
Processes that have been saved in the Process Library as Definitions can be exported in the form of *.json files or *.ser binary files for sharing out of the environment purposes.

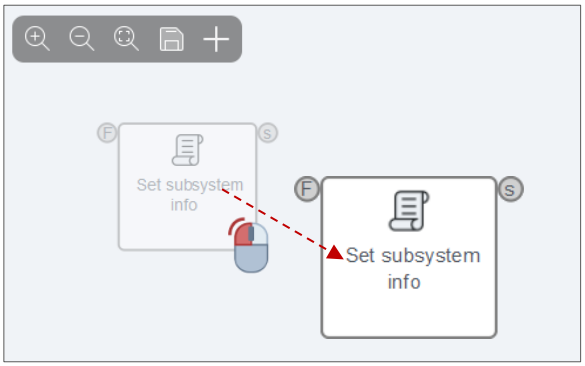
New process

To start the design of a new process, the user should first access the **Processes** workspace. Then, a new process design can start either by pressing the **Add** button, next to the *Process Instance List*, or by selecting the **New** option accessible from the button  or the context menu of the *Process Library*.



The design of the new process starts in a new empty *Process Diagram* tab. A new node can be added either with the **Create Node** option of the context menu on the *Process Diagram*, or by pressing the respective button of the toolbar. First the name of the node must be defined and when the **OK** button is pressed, the node is created.

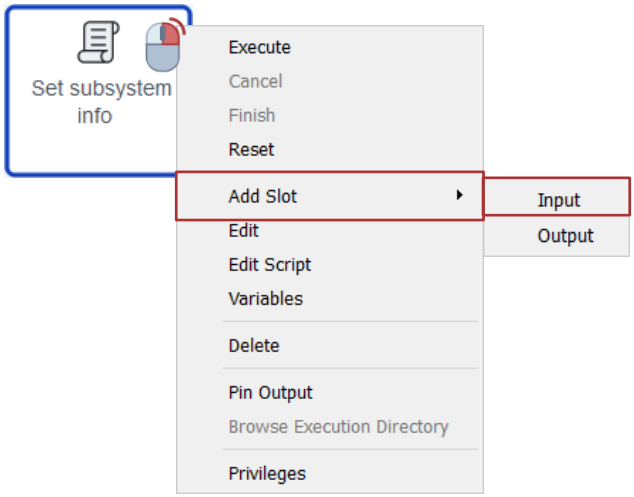




The node is automatically created at a default position of the diagram, but the user can move it with left mouse drag in the desired position. Additionally, the zoom level can be changed using the mouse roller or the respective toolbuttons.

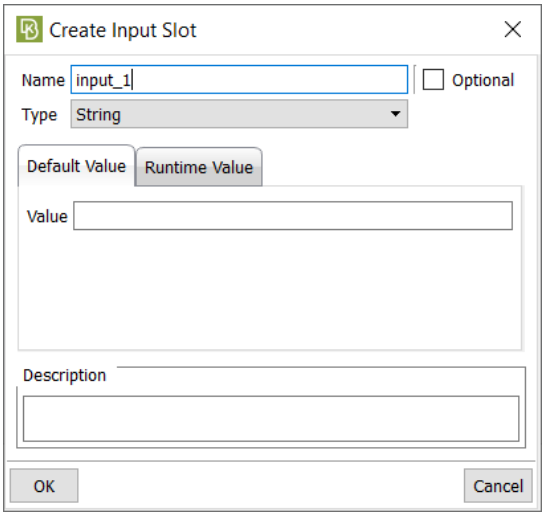
5.4.1. Input Slots

5.4.1.1. General



New input slots can be created for a node through the option **Add Slot** of its context menu.

When this option is activated, the *Create Input Slot* window pops up, for the definition of the slot.



The **Name** and the **Type** of the slot are the minimum information the user must provide.

Right below, there are two tabs that show the **Default Value** and the **Runtime Value** of the Input Slot.

During process design, a default **Value** can be specified for the slot. In this case, the input slot will have this initial value at the time of execution. If the input slot is connected to an output slot of a previous node, the default value will be overridden by the value transmitted through the connection.

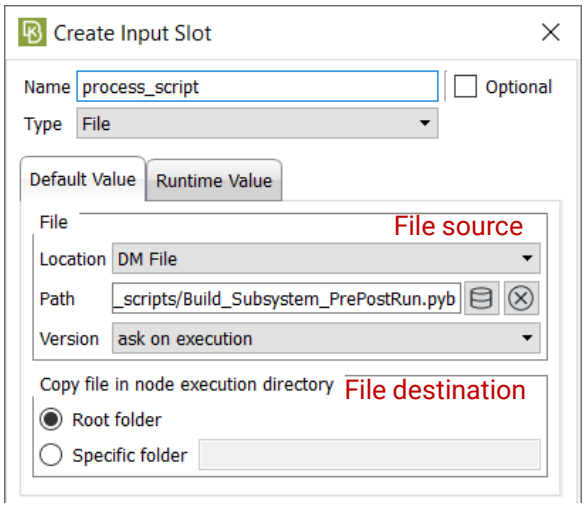
The Value field can also be filled with a variable, which will get a value at the time of execution according to other parameters of the process (more information on this topic can be found in paragraph 5.4.7).

In cases where an input slot is planned to be connected to the output slot of a previous node, its Value can remain empty, as the output slot will transmit its own value at the time of execution. Such input slots can be created automatically based on the output slot of the previous node. For more information, please refer to paragraph 5.4.3.

The **Description** text area at the bottom can accommodate some documentation for the input slot.

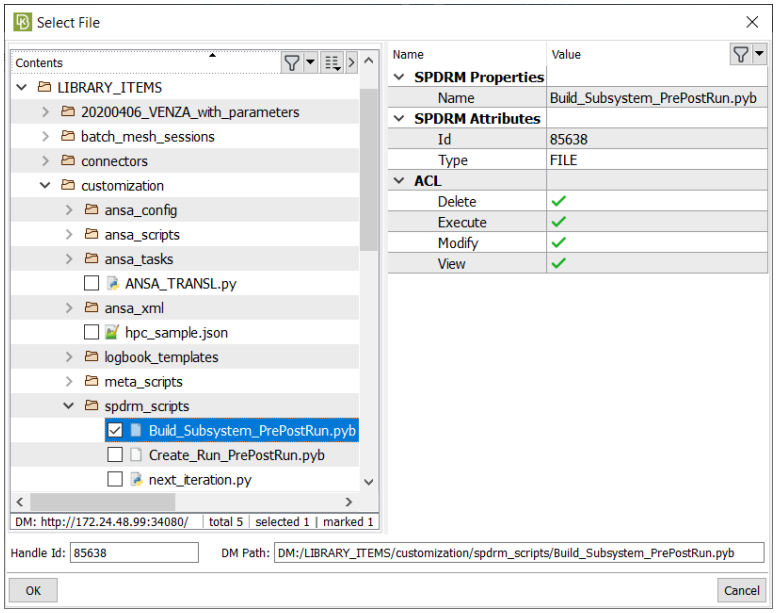
Depending on the selected **Type**, the UI of the *Create Input Slot* window changes to support the different information needed. Specifically, for the input slot types **File** and **List of Files**, there are additional options that the designer should define.

5.4.1.2. Type: File

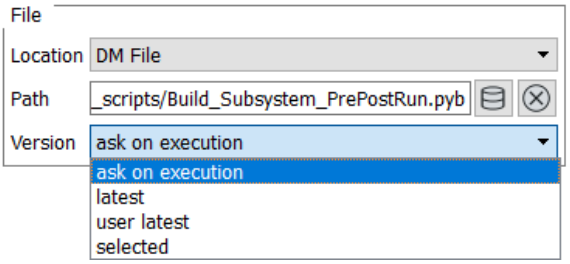


A file can be provided to the input slot, at the time of execution, either from the DM or the Local file system. The designer can define this option in the **Location** drop-down menu.

The selected file will be made available in the Node Execution Directory, at the time of execution. With the option **Root folder** active, the file will be available directly in the Node Execution Directory. The option **Specific folder**, will create a sub-directory with the given name inside the Node Execution Directory. The file will be available inside this sub-directory.



In case of a **DM File**, the user can press the **Browse** button to access the Data Manger, navigate to the folders, and select the required file.



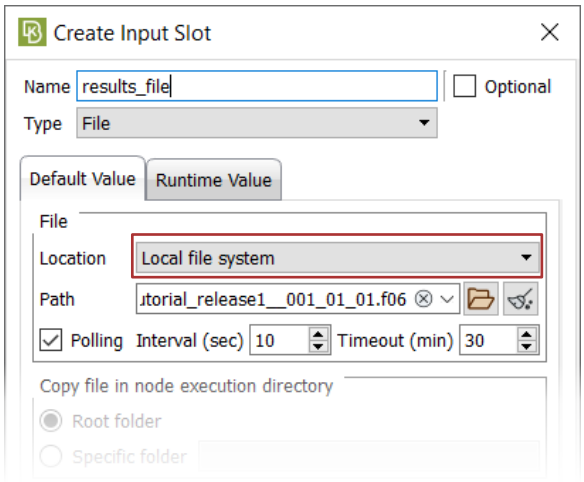
During the execution there are cases where multiple versions of the selected file exist in DM. In such case, the specific version to be used can be decided based on the **Version** option. The alternatives are:

ask on execution: at the time of execution, a window with the detected versions will pop-up for the executing user to select the desired one.

latest: the latest version of the file will be used,

user latest: the latest version of the ones created by the executing user will be used,

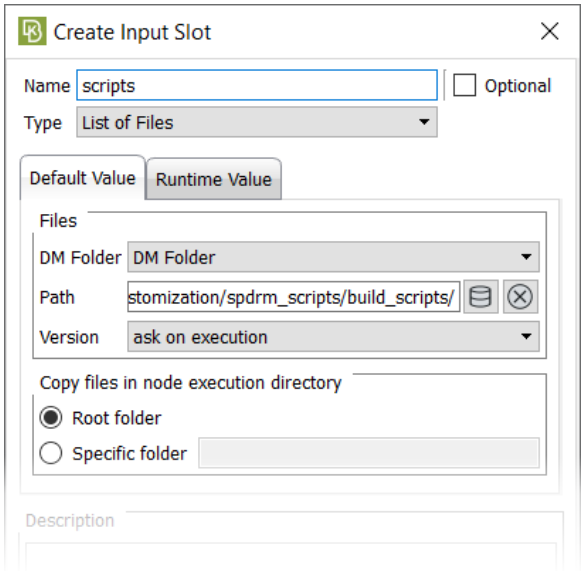
selected: the version explicitly selected during the design of the workflow will be used



When the location of the file is the **Local file system**, the designer can press the **Browse** button to navigate with a File Browser and select the file from the local or a network drive.

The option to poll for particular input data with a pre-defined interval is possible, by activating the **Polling** option. During the execution, the input slot will start polling for the file as soon as all other input slots are filled. When the file becomes available, the slot will be filled, and the execution of the node will be possible.

5.4.1.3. Type: List of Files



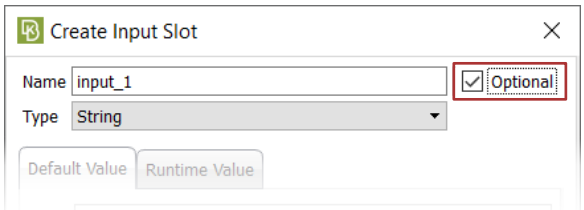
Similarly to the set-up of the input slot for a single file, the designer can select as input multiple files. This is achieved with the input slot Type **List of Files**, and is implemented with the use of folders. Thus, a folder with files can be provided to the Node Execution Directory, at the time of execution of the node.

The folder can be either a **DM Folder** or a folder from the **Local file system**.

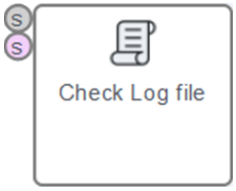
Similarly to the case of the single file described previously, in the **Version** option, the designer can define the desired handling in case multiple versions of the files of that folder are found at the time of execution.

The files will be made available in the Node Execution Directory, either at the top level or in a sub-folder of a specified name.

When the set-up of the input slot is finished, and the **OK** button is pressed, the slot is displayed on the left side of the node. More slots can be added following the same procedure. An existing slot can be edited with right click > Edit, or simply with double click. In both cases the *Edit Input Slot* window opens.

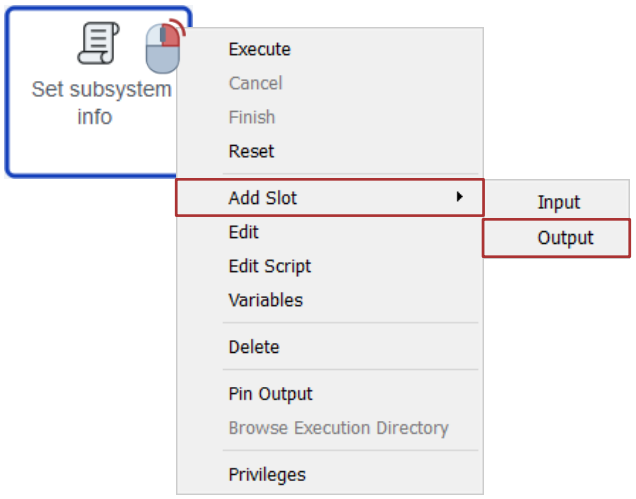


Important note: By default, in order for a node to get ready for execution, all its input slots need to be filled with data. In case the designer would like an input slot to be optional, the option shown on the left can be activated.
An optional input slot is drawn in pink color and if it has no value, it won't prevent the node from getting ready for execution.

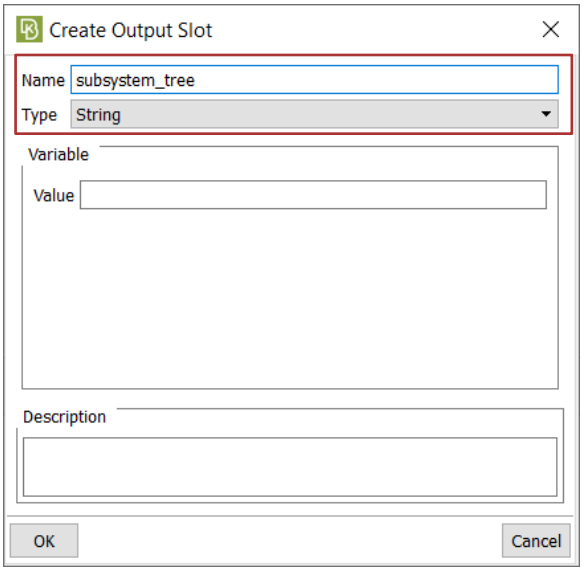


5.4.2. Output Slots

5.4.2.1. General



New output slots can be created for a node through the option **Add Slot** of its context menu. When the option is activated, the *Create Output Slot* window pops up, for the definition of the slot.

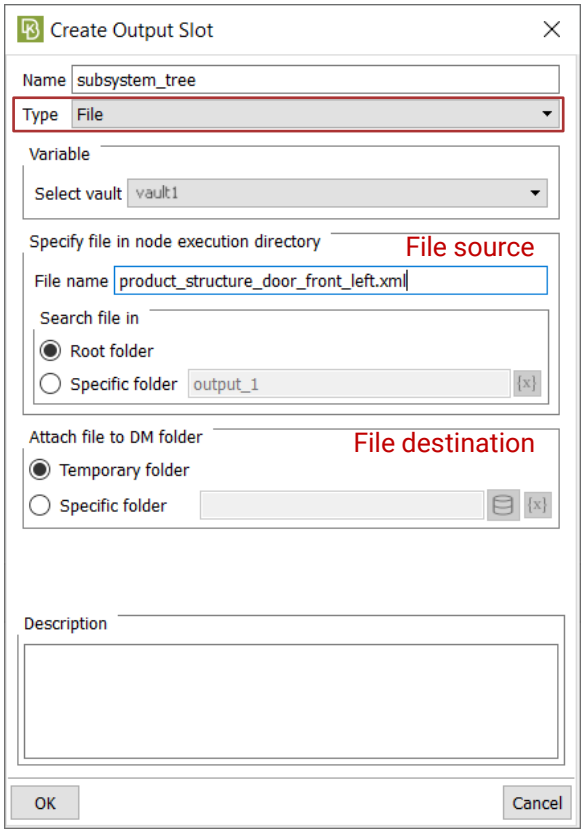


The **Name** and the **Type** of the slot are the minimum information the user must provide.

The **Value** field can be filled with a default value, at the time of designing the process. In this case, the output slot will get this value at the time of execution. This value will be released as soon as the node finishes its execution. Alternatively, the field can be filled with a variable which will get a value at the time of execution according to other parameters of the process (more information on this topic can be found in paragraph 5.4.7).

Depending on the activated **Type**, the UI of the *Create Output Slot* window changes to support the different information needed. Specifically, for the input types **File** and **List of Files**, there are additional options that the designer should define.

5.4.2.2. Type: File



The output slot of Type **File** is associated with a designated file which should be found, at the time the node finishes its execution, inside the Node Execution Directory. The designer must define a name for this file in the field **File name**. Additionally, one should define where exactly the file will be found inside the Node Execution Directory. This is defined in the **Search file in** group of options:

Root folder: the file should be found at the Node Execution Directory directly

Specific folder: the file should be found in a sub-folder of the Node Execution Directory, with a specific name.

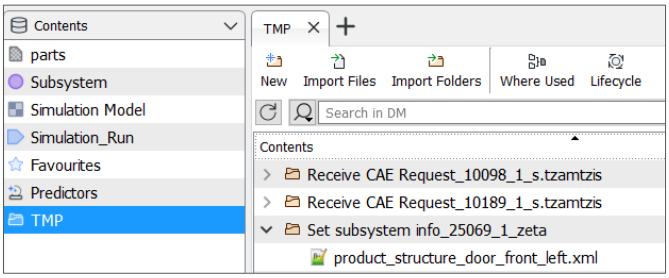
The file found by the output slot, is uploaded to DM the moment the slot is released. The destination is defined by the designer in the **Attach file to DM folder** options group.

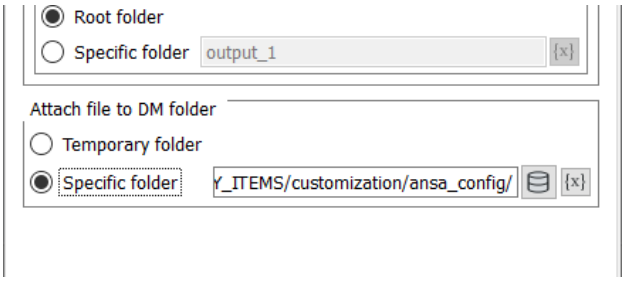
The alternative options are:

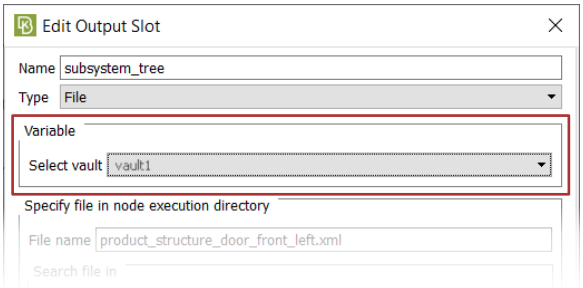
Temporary folder: the file will be uploaded in the *TMP* container, in a folder with name:

`<node_name>_<node_id>_<execution_count>_<executing_user>`

Specific folder: the file will be uploaded in a specific folder, either under the path selected with the **Browse** button  or with the aid of a process variable, by pressing this button  (please refer to 5.4.7).

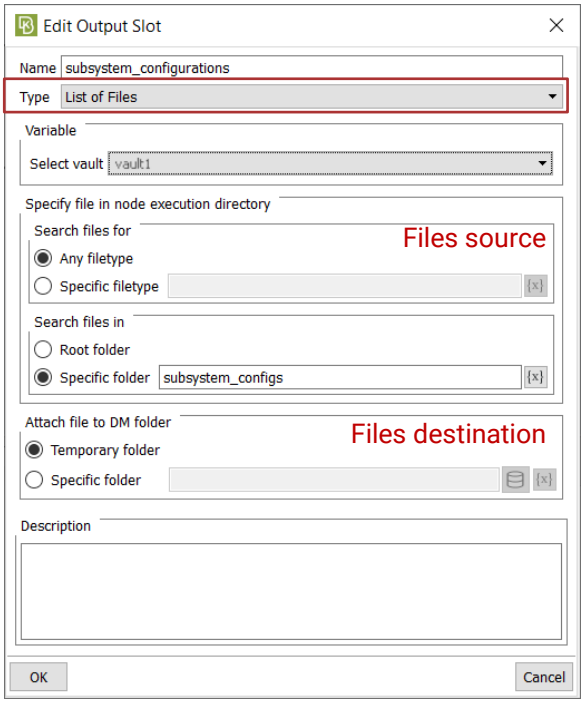






In case more than one vaults are available in the SPDRM configuration, the destination Vault name must also be defined.

5.4.2.3. Type: List of Files



Similarly to the set-up of the output slot for a single file, the designer can define output slots that consist of multiple files. This is achieved with the option **List of Files**. At the time the node finishes its execution, the files will be searched by the output slot in the following locations:

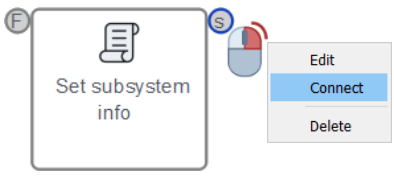
Root folder: in the Node Execution Directory, or

Specific folder: in a sub-folder with a predefined name.

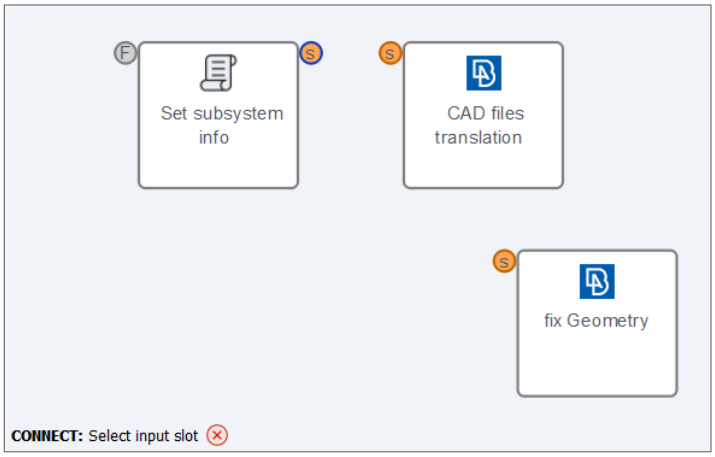
Optionally, the designer can restrict the filetypes to be identified by activating the option **Specific filetype** (e.g., type *.txt).

The files found by the output slot, will be uploaded to DM at the time the slot is released. The destination is defined by the designer in the **Attach file to DM folder** options group, as described above for the single file output slot.

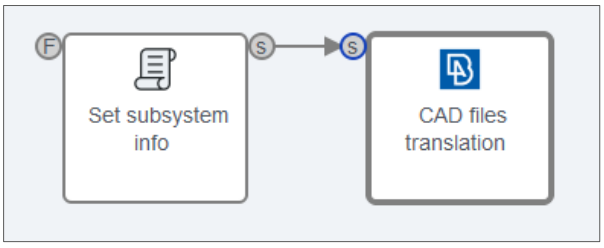
5.4.3. Connecting the slots

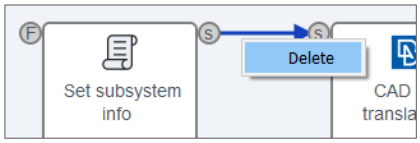


An output slot can be connected to an input slot of the same type. To create the connection, the designer can access the context menu of an output slot and select **Connect**. At that point, the diagram gets in *CONNECT mode* and waits for the selection of an input slot. Only the input slots of the same type can be selected, and these are highlighted in orange.

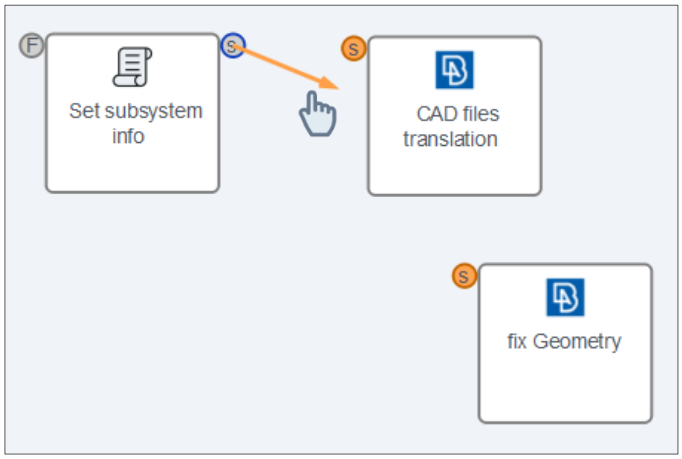


The user can select one of the valid input slots to create the connection. Once a valid slot is selected the connection line is designed in the diagram.



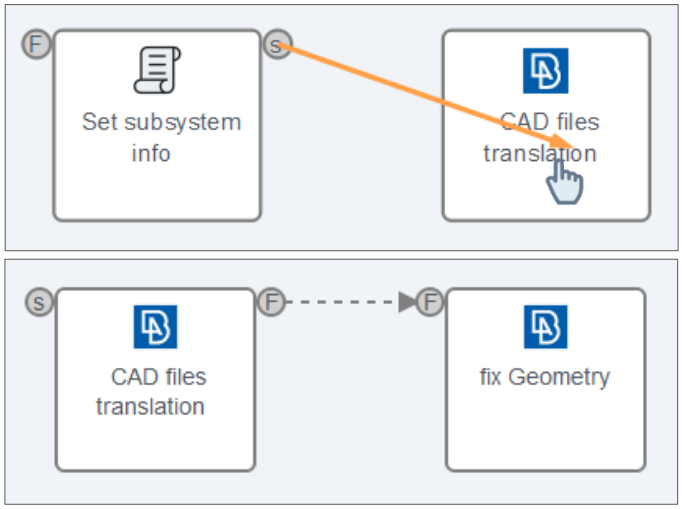


A connection line can be deleted with right mouse button click and the option **Delete** of its context menu.



There's also an interactive way for creating connections, with left mouse drag from the output slot of one node to a valid input slot of another node.

While this action is performed, the connection line to be created is previewed on the diagram in orange color and the eligible slots are highlighted. Once the user approaches the desired input slot and releases the mouse button, the connection is created.



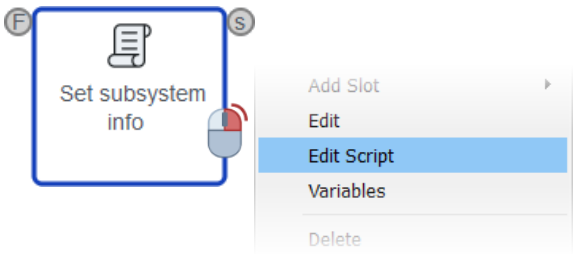
Finally, there is the option to connect two nodes by automatically creating the desired input slot on the target node. Performing the same action as described above, in case the user releases the mouse button on top of a node (instead of an input slot), KOMVOS will automatically create a new input slot of the same type and connect the nodes.

The connection line between slots of type **File** or **List of Files**, is displayed with a dashed line.

5.4.4. Configuring different types of nodes

5.4.4.1. Script Nodes

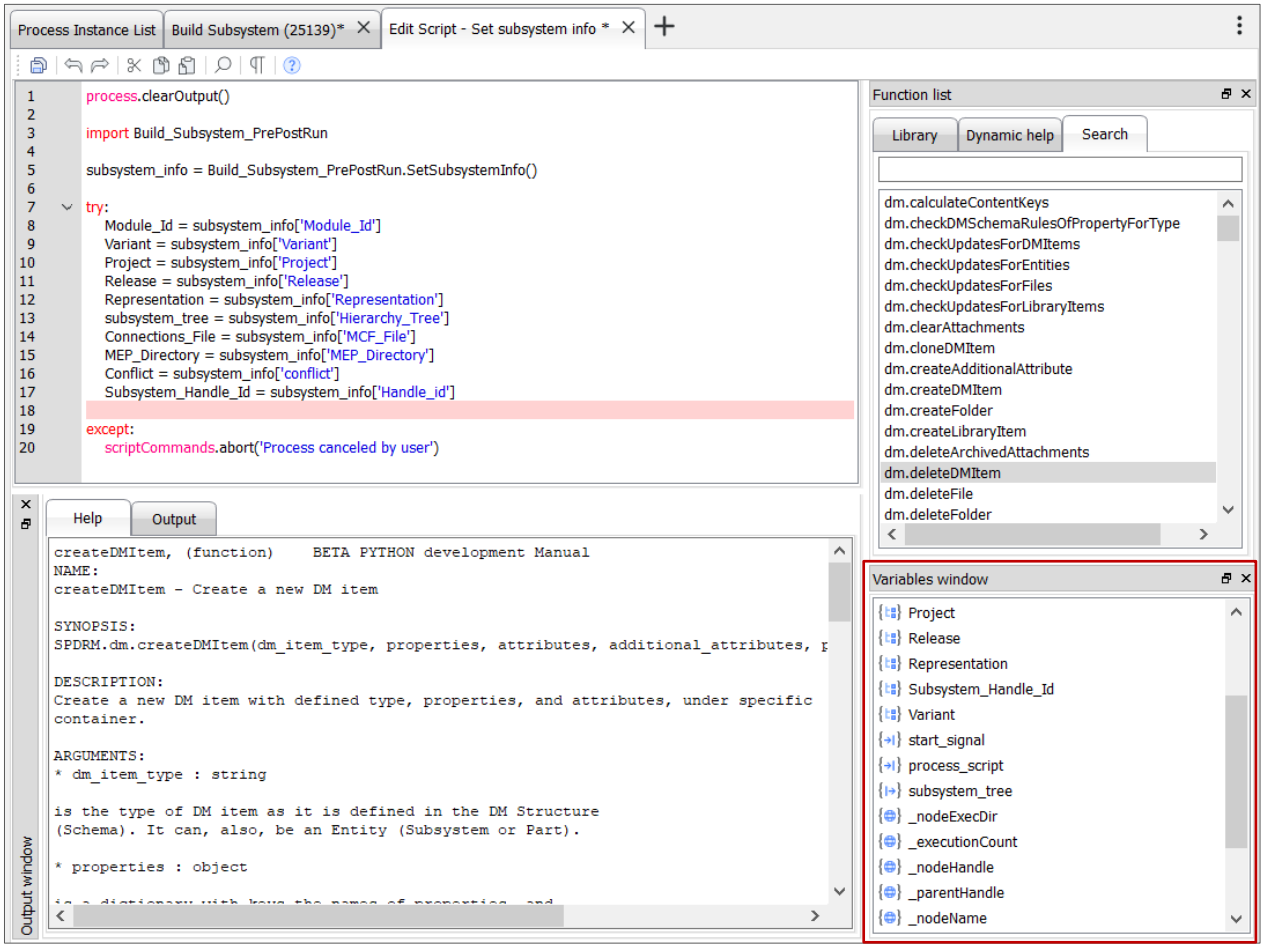
Script and Application nodes can execute scripts written with the SPDRM Python API to perform custom actions. Such scripts can be written inline, in the *Script Editor* of the node, or can be imported from file through input slots.



To access the script of a node, activate the **Edit Script** option of its context menu.

A script editor very similar to that of the BETA Suite opens in a new tab with name *Edit Script - <selected_node_name>*

All available script functions are listed in the *Function list* on the right panel. Additionally, a list of variables is available in the *Variables window* right below the Function list.

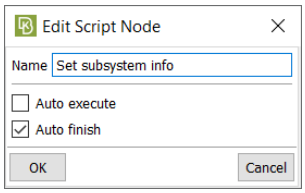


In this list all the node's variables are listed. These include:

- The node's variables added by the designer
- The node's variables inherited from the parent process
- The build-in variables of the node
- The input slots variables
- The output slots variables

Within the script of a node, all these variables belong to the same namespace and can therefore be referenced within the script directly by their name.

An existing Script node can be edited through the option **Edit** of its context menu. Here the **Name** can be changed. Additionally the designer can manage the following options:



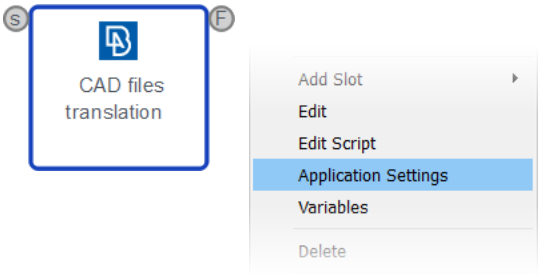
Auto execute: the node will be executed automatically as soon as its input data are available.

Auto finish: the node will be finished automatically when its script finishes its execution.

5.4.4.2. Application Nodes

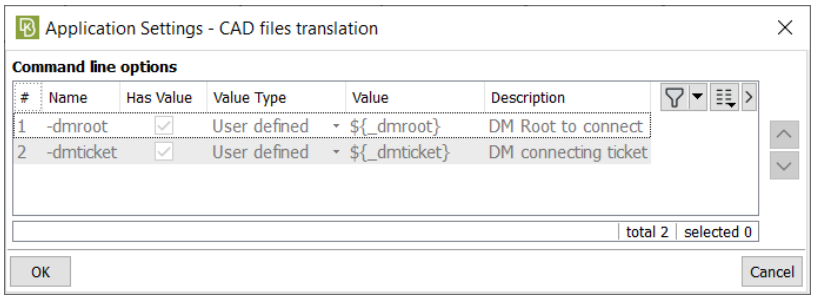
Application settings

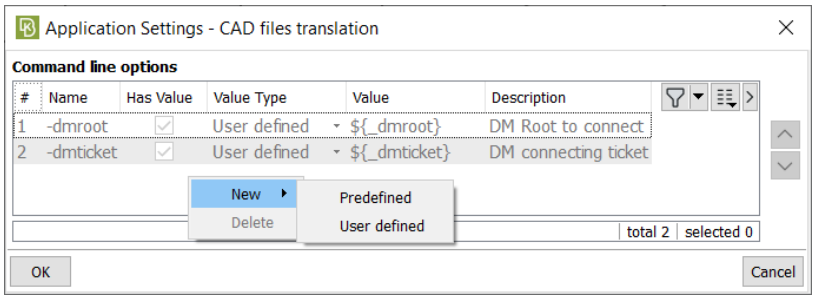
When application nodes are executed, they launch an external application among those registered in SPDRM by the system administrator. During the registration of applications, all required and optional command line options are defined. Thus, when the process designer creates an application node, it is possible to configure the command line options that will be passed to the application according to the central configuration. For more information on the central registration of external applications, please refer to the SPDRM Administrators Guide.



To access the application settings of the node, activate the option **Application Settings** of its context menu.

The *Application Settings* window opens. Any greyed out options that may appear are those marked by the system administrator as mandatory and cannot be deleted.

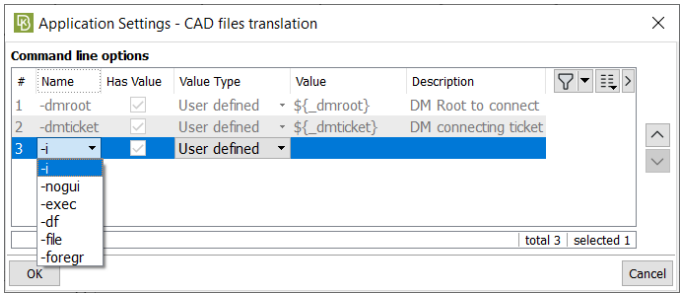




New options can be added through the context menu of the window and select the option **New**.

There are two types of options that can be added:

- **Predefined**: The options already configured centrally, in the SPDRM registered application setup
- **User defined**: Any additional option, that may not be among the predefined.

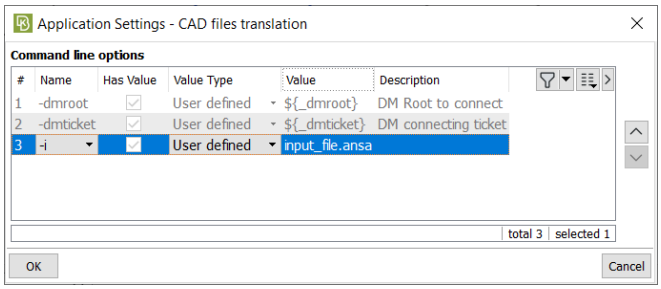


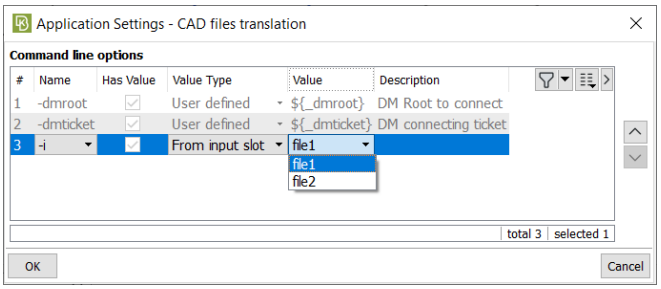
When a new **Predefined** option is added, its *Name* can be selected from the drop-down menu. The selected option might or might not require a value according to the setup of the registered application. This characteristic will be displayed in the *Has Value* column.

If a value is required, this can be configured as:

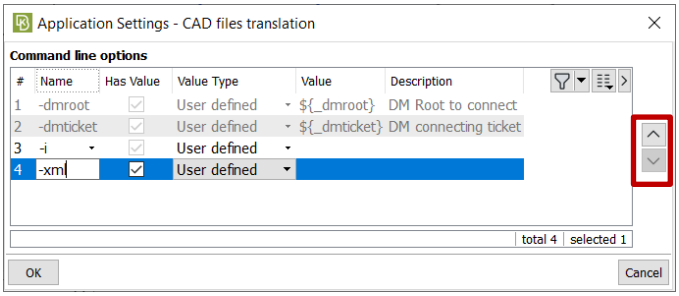
User defined: The designer should type its *Value* in the respective column, as a fixed value or with the use of a process variable (`${ }` syntax, see paragraph 5.4.7 for details).

From input slot: The *Value* will be filled upon execution with the runtime value of an input slot. The available input slots are listed in the drop-down menu.





Optionally a *Description* can be added for each action in the respective column.

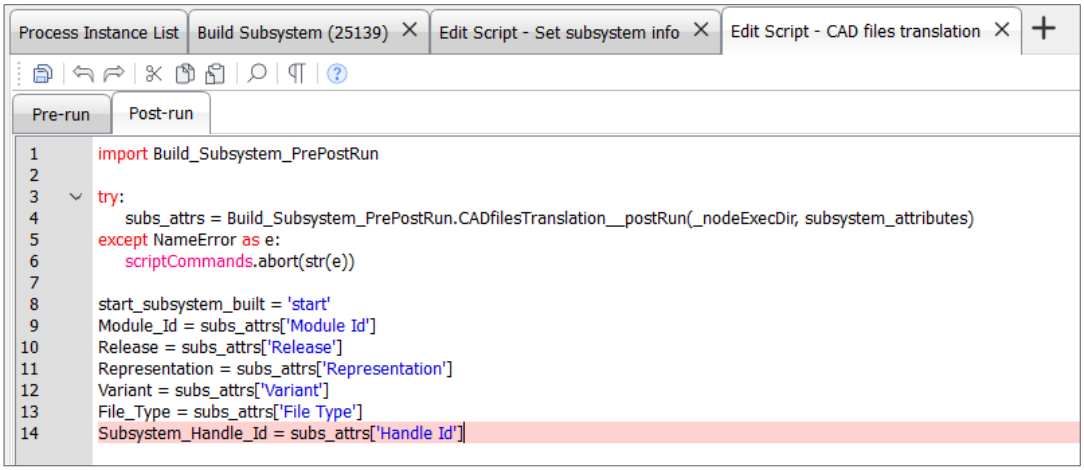


When the designer needs to add a command line option that is not already configured in the registered application setup by the SPDRM administrator, the **User defined** option can be used. In this case its name can be typed in the *Name* column manually.

Note also that the order of the options can be changed with the arrow buttons.

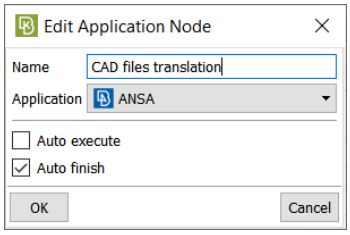
Pre- and post-run scripts

Application Noes can execute a script before launching the application (e.g. to prepare some inputs), and another script right after (e.g. to post-process the outputs). The designer can define these scripts in the *Script Editor* that opens through the context menu option **Edit Script** of the Application Nodes. In this case, the Script Editor includes two separate tabs, the *Pre-run* and the *Post-run*.



Note that in order for a modification to the Script Editor to take effect, the Save button must be pressed on the toolbar.

An existing Application node can be edited through the option **Edit** of its context menu. Here the **Name** and the executing **Application** can be changed. Additionally the designer can manage the following options:

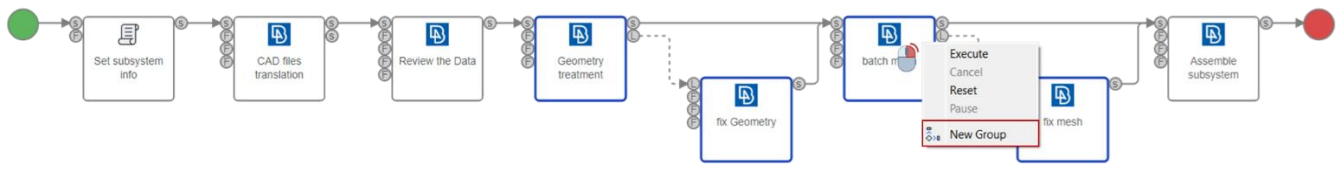


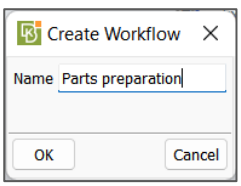
Auto execute: the node will be executed automatically as soon as its input data are available.

Auto finish: the node will be finished automatically when the application is terminated.

5.4.4.3. Sub-process Nodes

To create a sub-process, the user can select the nodes to be included one by one, holding the Ctrl key, and then activate the option **New Group** from the context menu.

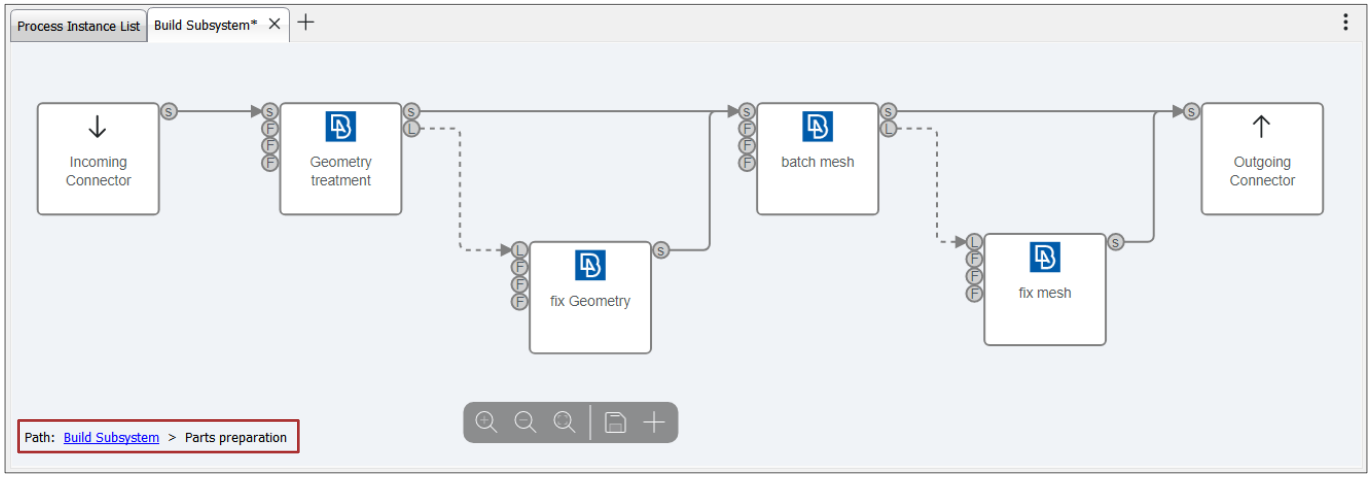




The *Create Workflow* window appears, for the user to name the new sub-process node. Upon confirmation the sub-process node is created and placed on the diagram, containing the selected nodes.

The nodes might need to be re-arranged.

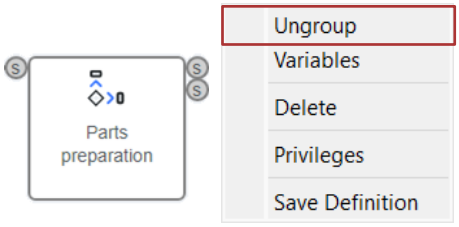
With double click on the sub-process node, the user can navigate now in its contents. The nodes have been automatically connected with the virtual nodes *Incoming/Outgoing Connector* which represent the input and output streams.



When inside a sub-process node, the **Path** appears in the breadcrumb trail of the lower left corner of the *Process Diagram*. This can be used for navigation to the root level.

Inside an existing sub-process the user can create nodes as described in the previous sections.

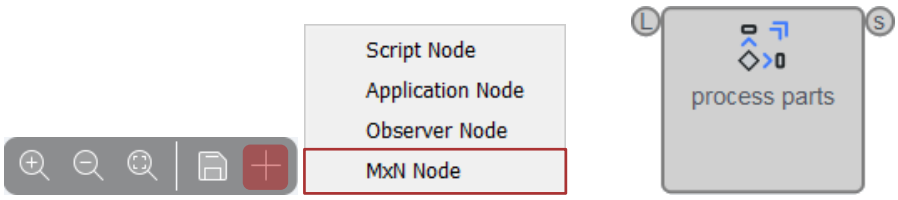
A sub-process node can be de-composed to its content nodes, by selecting it and activating the option **Ungroup.**



5.4.4.4. MxN Nodes

MxN nodes are special type of sub-process nodes. They get as input a list of *M* files and they execute the same sub-process *M* times (with a different input file each time), optionally delegated to *N* different resources. The Input slot that carries the data set to be distributed is always of type *List of files*.

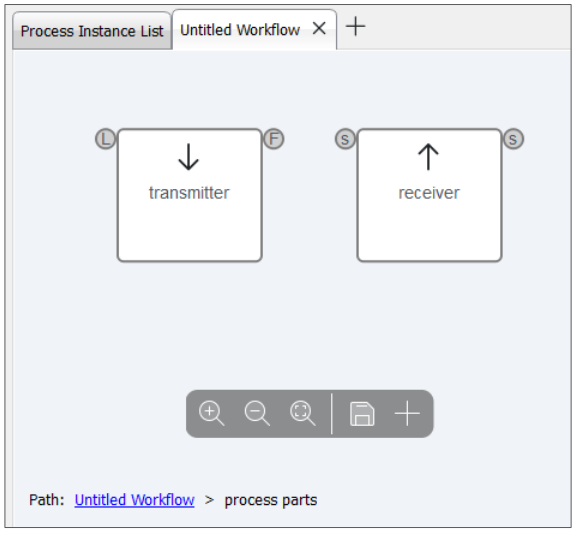
A new MxN node can be created either with the **Create Node** option of the context menu on the *Process Diagram*, or by pressing the respective button of the toolbar.



As soon as a name is given, the new node is placed in the *Process Diagram*. The designer should then:

- Design the sub-process that will be instantiated *M* times, for each input file.
- Define the *List of files* input of the MxN node.
- Configure other settings of the MxN node that affect its execution.

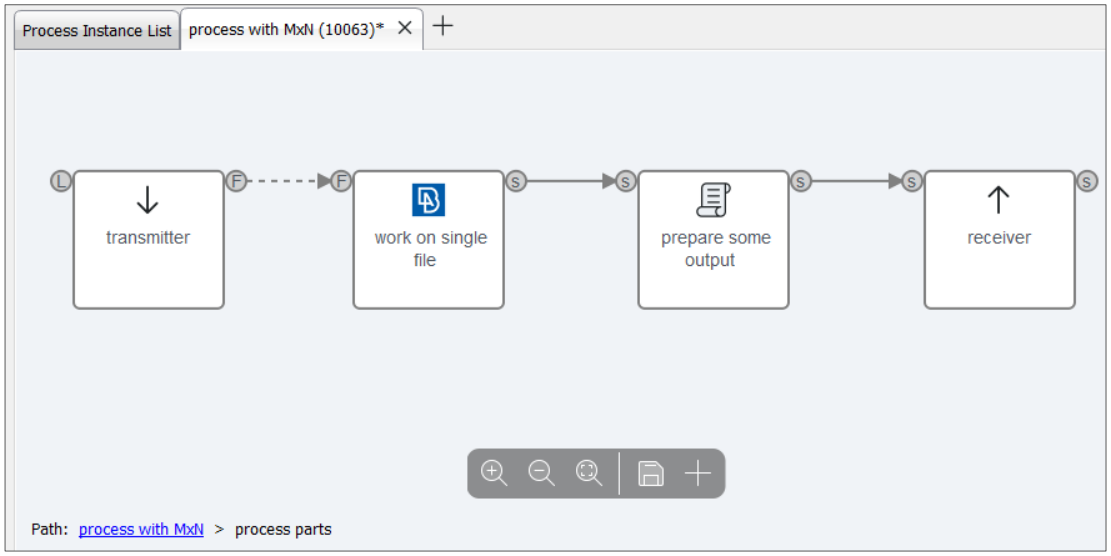
To design the sub-process one should navigate inside the MxN node. With double click on the node the user is directed in its contents.

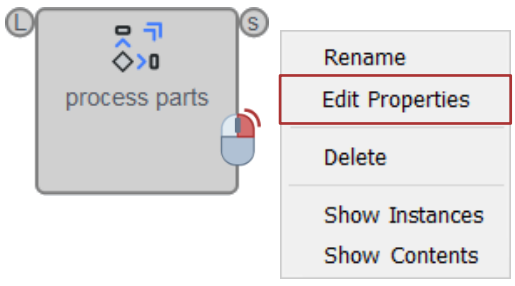


Inside a new MxN node, only its input and output slots will be found. These are represented with two special nodes that cannot be edited and exist only for visualization, the **transmitter** and the **receiver** nodes.

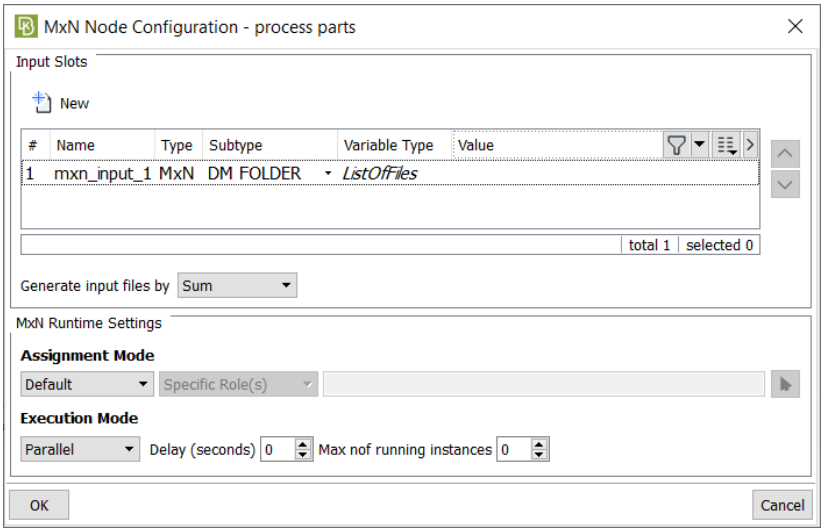
Notice that the *List of Files* input slot of an MxN node, is transmitted inside its sub-process as a single *File* input. This of course serves the purpose of the MxN node, which is to repeat the very same process as many times as the number of the input file.

Following the steps described in the previous paragraphs, the designer can create the desired sub-process that will handle each of the input files.

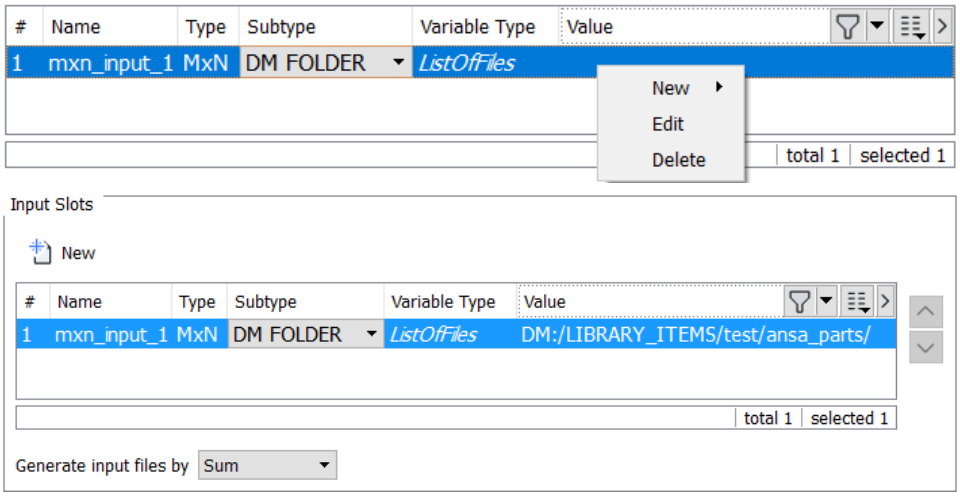




Next the input set of files must be defined and the MxN node should be configured to serve the use case scenario. Access the context menu of the node and select option **Edit Properties**. The *MxN Node Configuration* window opens.



By default the MxN node has at least one input of type *MxN*, which will be the input data set (in the form of an input slot of type *List Of Files*) that will drive the instantiation of the MxN sub-process. Access the context menu of this input to **Edit** it and select the input folder from the DM.



The selected folder is displayed in the *Value* column.

When executed, the MxN node will provide one file from this folder to each of the instances of its sub-process.

The MxN input data are arranged in data sets, thus it is possible to add more than one data set. When multiple data sets are used two use case scenarios can be served:

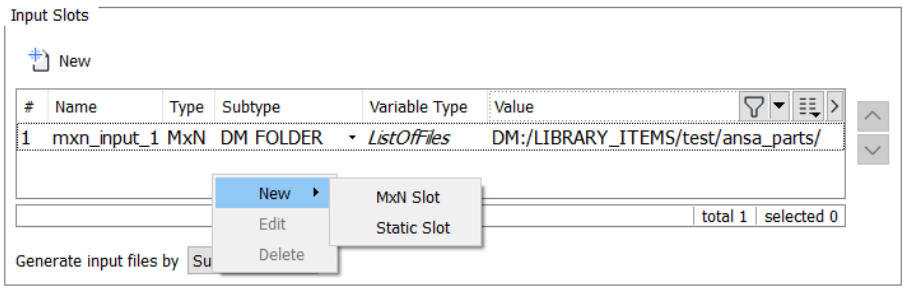
- either the data to be processed by the MxN sub-process are provided from different sources,
- or the files of more than one data sets must be combined to produce the desired result.

The use case scenario to be followed is controlled by the option **Generate input files by**, with two possible values according to the cases described above, **Sum** and **Combination**.

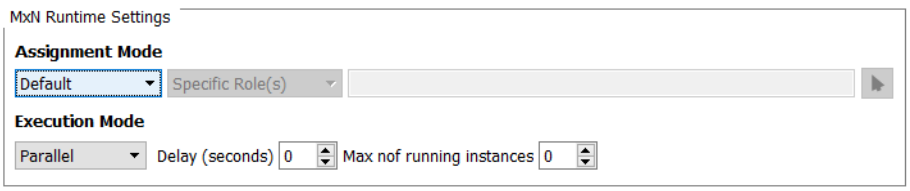
So, adding a data set of '*k*' files and another data set with '*j*' files, the data sets can form one data set which will be the union of the two ('*k+j*' files to be distributed), or they can be combined to form one big data set of '*k*j*' unique combinations of files to be distributed.

In an example of the first case, '*k*' ANSA files coming from one route of the process, and '*j*' ANSA files coming from a second route of the process, all of them need to be geometrically improved.

In the second case, for example the scenario to be served could be, '*k*' ANSA files that need to be meshed but using '*j*' meshing configuration files, with the target to produce '*k*j*' unique meshed files each of them meshed with all available meshing configurations.



Additionally to the *MxN* input data sets, the possibility to add one or more *Static* inputs is offered. These inputs (of any type) will be available to all instances of the MxN sub-process upon execution.



In the *MxN Runtime Settings* section, the designer can configure the execution mode of the MxN sub-process instances.

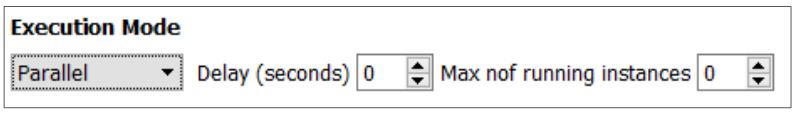
By default, all instances will be assigned for execution to the user who initiated the whole process (the parent process). Optionally, the instances can be assigned to be executed by multiple users.

For this purpose the following task assignment modes are supported:

- **Cyclic user**: Each instance will be assigned to one of the users of the selected role.
- **Cyclic role**: Each instance will be assigned to one of the selected role(s). Any user that belongs to the assigned role can execute the specific instance.

Additionally, the sub-process instances can be executed either in parallel or sequentially. This is configured with the following two options:

- **Parallel**: All instances will be executed simultaneously.
  - Optionally, a delay can be added to the parallel execution by activating the **Delay** option. In this case the instances will be executed with a time offset (delay).
  - A max batch size can be defined, in order to prevent saturation of resources.



- **Sequential**: The MxN instances will be executed one after the other.
  - Optionally a **Timeout** can be defined. When this time limit is reached all subsequent nodes will not be executed.

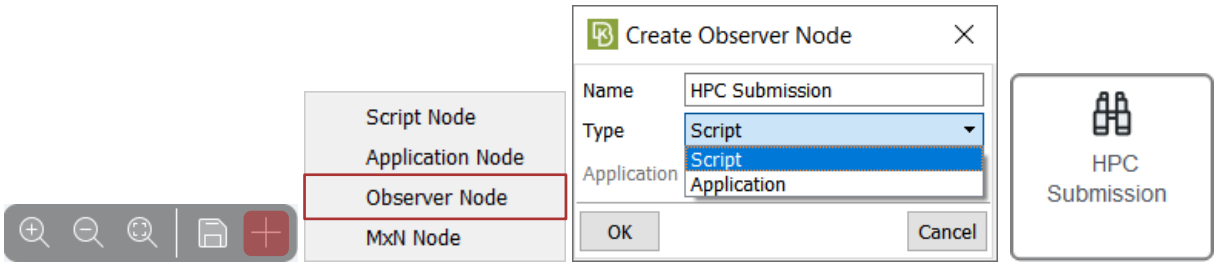
NOTE: Quite often, the process designer needs to return some output from each of the instantiated sub-process of the MxN node to the main process. This is implemented automatically through the output slot of the MxN node, that combines the outputs of all instances into a single json-like value as shown below:

```
{<handle id of instance 1 sub-process>:<value of its receiver output slot>,
 <handle id of instance 2 sub-process>:<value of its receiver output slot>,
 . . . . .
 <handle id of instance n sub-process>:<value of its receiver output slot>}
```

5.4.4.5. Observer Nodes

Observer Nodes are used for job submission and monitoring on HPC systems. There are two types of observer nodes, the **Script** and the **Application Observer Nodes**. Both these types submit a job, either through Script or by calling a Registered Application, register their interest for a particular job id and then get in an idle mode, until the respective job shows some progress.

A new Observer node can be created either with the **Create Node** option of the context menu on the *Process Diagram*, or by pressing the respective button of the toolbar. The designer must provide a *Name* and the *Type* of the node, *Script/Application*.



The **Script Observer Node** consists of three components: **Pre-run**, **Post-run** and **Cancellation** scripts.

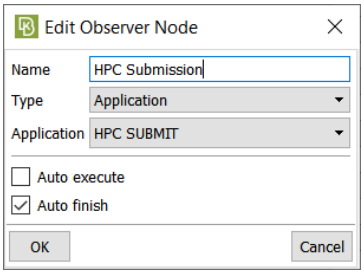
- In the Pre-run script any tasks needed prior to the submission should be conducted. The submission of the job should be done here, and the Job Id returned from the HPC should be assigned to the build-in variable `_correlationId`.
- In the Post-run script the designer can optionally define tasks that should follow the completion of the submission.
- In the Cancellation script, the designer should define all the needed actions and communication with the HPC so as to allow the cancellation of the submitted job.

In the phase between Pre- and Post-run scripts, the Observer Node is in monitoring state.

The **Application Observer Node** contains two additional components, the **Application** itself and the **Post-application script** which can contain any actions required upon completion of the submission. In this type of node the HPC submission takes place by using a dedicated application, so the assignment of the returned Job Id to the build-in variable `_correlationId` should be done in the Post-application script.

The `_correlationId` is the identity of each running Observer Node. Based on this unique id the node takes updates from the server about the status of the job it monitors. For more information about the HPC Jobs management please refer to paragraph 5.7 of this document.

An existing Observer node can be edited through the option **Edit** of its context menu. Here the **Name** and its **Type** can be changed. For an Application Obsrever node the executing **Application** can also be changed. Additionally the designer can manage the following options:



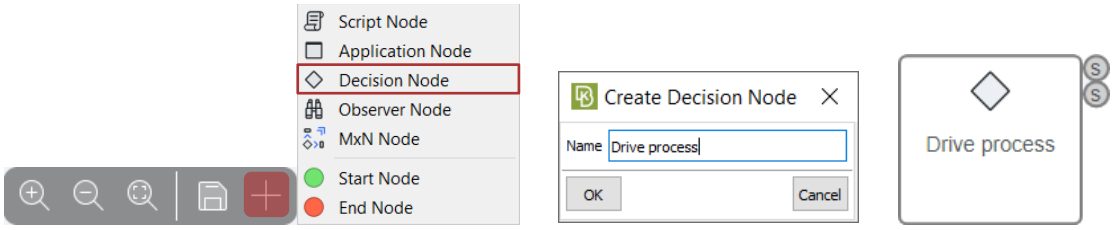
Auto execute: the node will be executed automatically as soon as its input data are available.

Auto finish: the node will be finished automatically when the monitored job gets one of its pre-defined finished states and its Post-run script finishes its execution.

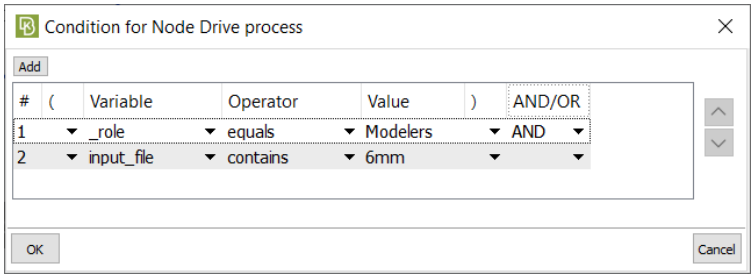
5.4.4.6. Decision Nodes

The Decision Nodes are used to drive the process choosing among two possible paths, based on a condition statement. The node fills one of its output slots based on whether the statement is evaluated to True or False.

A new Decision node can be created either with the Create Node option of the context menu on the Process Diagram, or by pressing the respective button of the toolbar.



After the creation of the Decision node the user will have to configure its condition criteria. Selecting the option **Edit Condition** of its context menu, the *Condition for Node…* window opens.

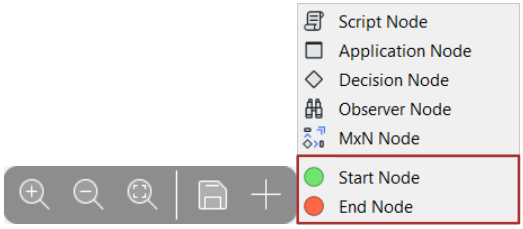


Here the user can add conditions using the build-in/global/input-output variables of the node and appropriate operators and values. The different conditions can be combined with AND/OR statements and parentheses can be also used for expressing complex criteria.

5.4.5. The Start/End nodes

It is a good practice for all workflows designs to contain one Start and one End node. These nodes signify the starting and ending of a process, and are considered by the calculating algorithm of the progress of a process.

The Start/End nodes can be added in the process, from the toolbar, similarly to the design of the rest node types.



5.4.6. Saving the Process

At any point, a designed workflow can be saved as **Definition** (Template) or as **Instance**.

A **Definition** of a process:

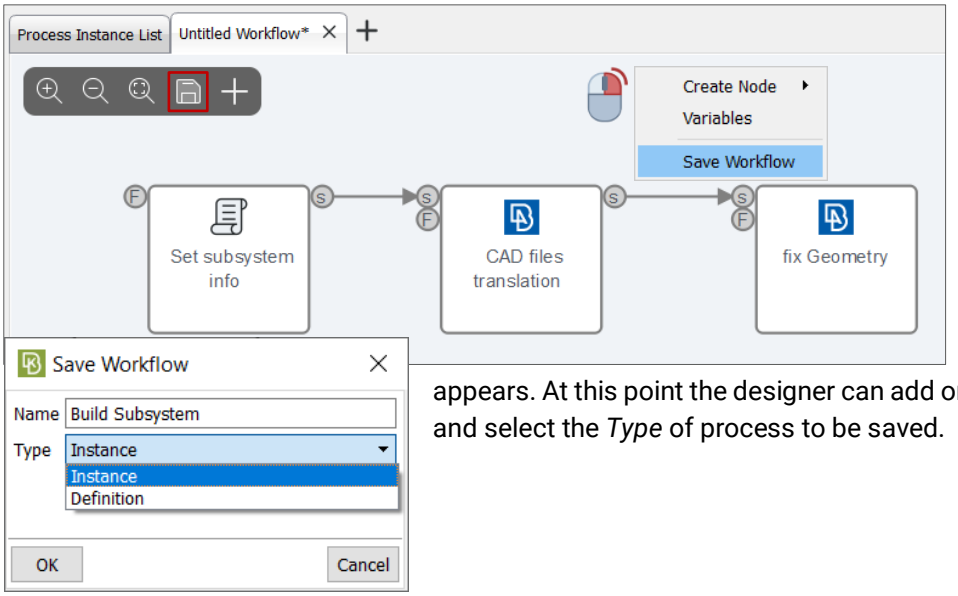
- is available for instantiation in the *Process Library*,
- is subject to versioning,
- can be exported to a file to be shared with other teams.

An **Instance** of a process:

- can be found in *Process Instance List*,
- does not necessarily need to exist as a Definition,

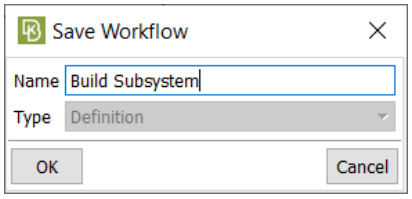
- can have multiple instances running at the same time,
- can be used for local testing and won't appear as a new version in the Process Library.

The processes that are marked with the "*" sign, have not been saved yet or were modified since their last save.



To save a process one can either activate the **Save Workflow** option in the context menu of the design area or press the **Save** button of the toolbar.

The *Save Workflow* window appears. At this point the designer can add or modify the *Name* of the workflow, and select the *Type* of process to be saved.



Note that, no such option will be given if a workflow has already been saved as an Instance.

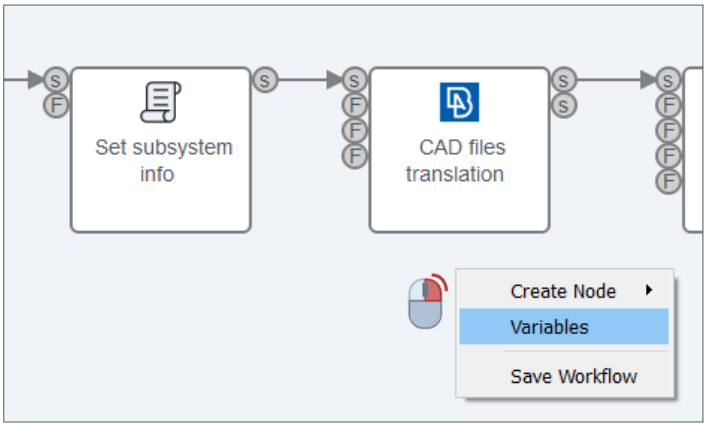
5.4.7. Variables

Processes and individual nodes can host *Variables*. The variables defined on a process are inherited to all its inner nodes. These variables can be used to:

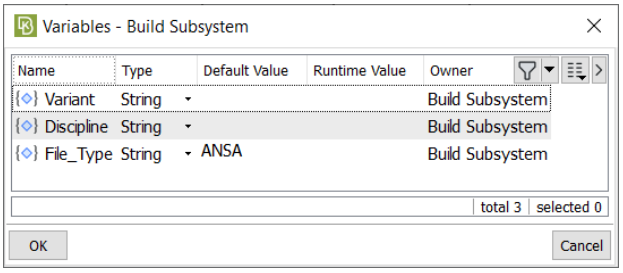
- facilitate the transfer of information between the nodes without the need of connections,
- enable the flow of information throughout the process, into inner sub-processes, independently of the number of sub-process levels,
- simplify and parameterize the workflow.

The variables can be of various types: **String**, **Integer**, **Float**, **Boolean**, **Date**. They can optionally have a default value given during process design. They will get their runtime value during execution.

5.4.7.1. Process Variables



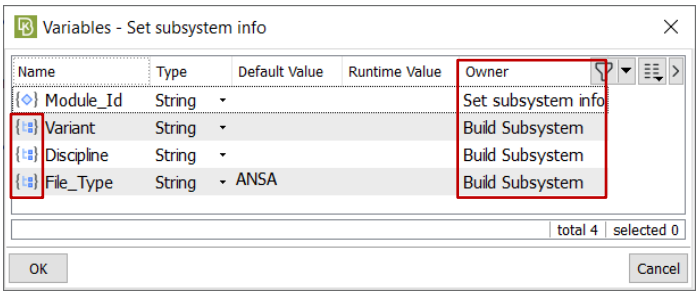
To add a variable on the parent workflow, right click on the design area and select the **Variables** option of the context menu.



A new process variable can be added with the option **New** of the context menu in this window. The *Type* can be selected from the combo-box in the respective column. In the *Default* column, a default value can be optionally added, which will be inherited by the *Runtime Value* the moment of execution. Alternatively, the *Runtime Value* can be set through a script executed by a process node.

A Process Variable is “visible” to all the nodes of the process, top-down.

Node Variables



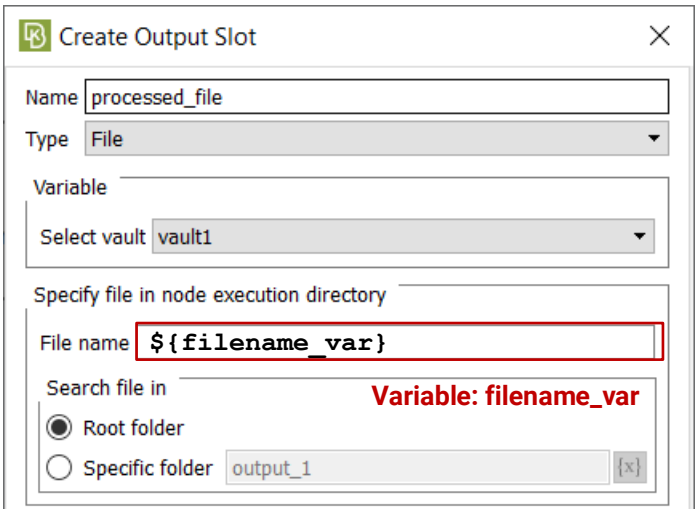
To add a variable on a node, select the option **Variables** of its context menu to open the *Variables* window. A new node variable can be added with the option **New** of the context menu in this window.

In this list one can also see the variables inherited from the process -if any, marked with a different icon.

The column *Owner* shows the name of the process component on which the variable was defined (here “Set subsystem info” is the name of the node and “Build Subsystem” is the name of the process).

Additionally to the inherited variables and the node own variables created by the designer, a node also holds some build-in variables that are useful to the designer. These include information such as the full path to the node's execution directory (`_nodeExecDir`), the unique ID of the node (`_nodeHandle`), the name of the executing user (`_execUser`), and many more. A list of these variables is found in the *Script Editor* of the node (refer to 5.4.4.1)

5.4.7.2. Parameterizing a process with the aid of variables



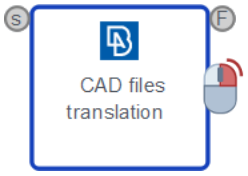
Process and Node variables can be used for the parameterization of a process.

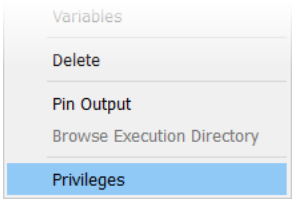
For example, an output slot can get a predefined value by the designer, or it can be defined to acquire the runtime value of a variable at the time of the execution. This variable can get a value by the executing user during the execution of the process or, usually through a script.

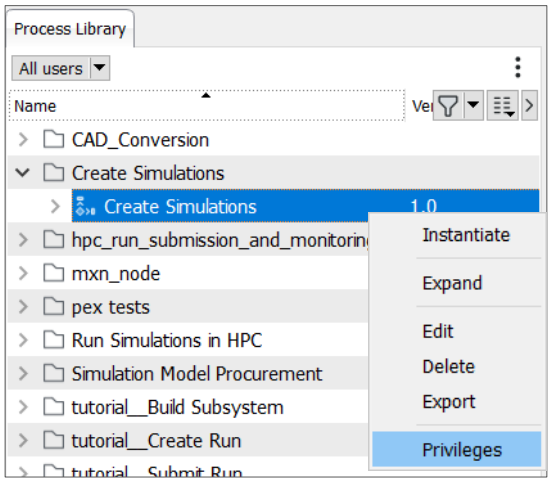
Code snippet from the script of the node

```
…
if Discipline == 'NVH':
        filename_var = 'door_f1_D256.nas'
elif Discipline == 'CRASH':
        filename_var = 'door_f1_D256.key'
elif Discipline == 'DURABILITY':
        filename_var = 'door_f1_D256.inp'
…
```

5.4.8. Privileges definition

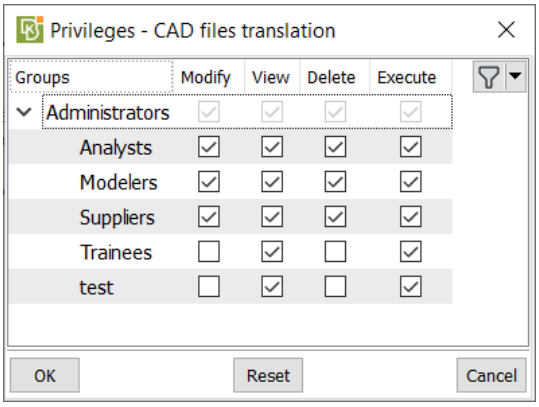






During the design of a node, the designer can define its privileges from the respective option of the node's context menu.

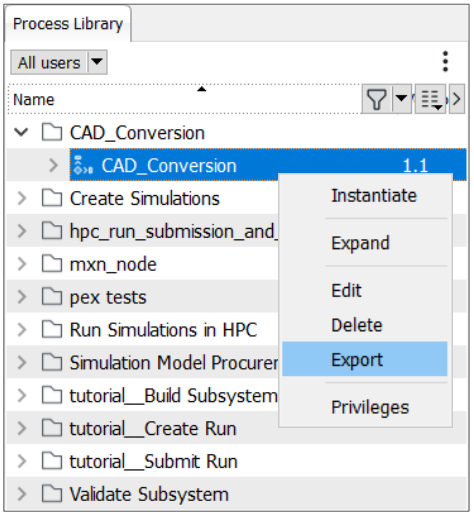
In the ACLs table the designer can set role-based access control to the specific node.



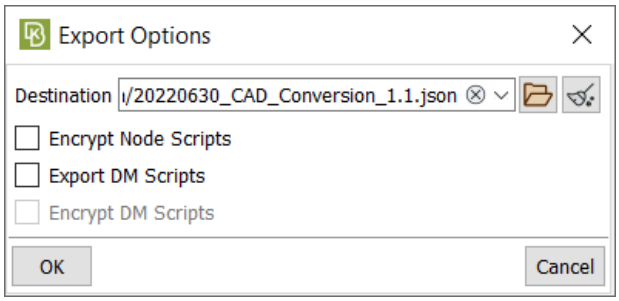
The privileges of a workflow saved in the *Process Library* can be set in a similar way.

5.5. Export processes

All processes listed in the *Process Library* can be exported in the form of *.json files or *.ser binary files for sharing with other teams or for transferring to other environments.

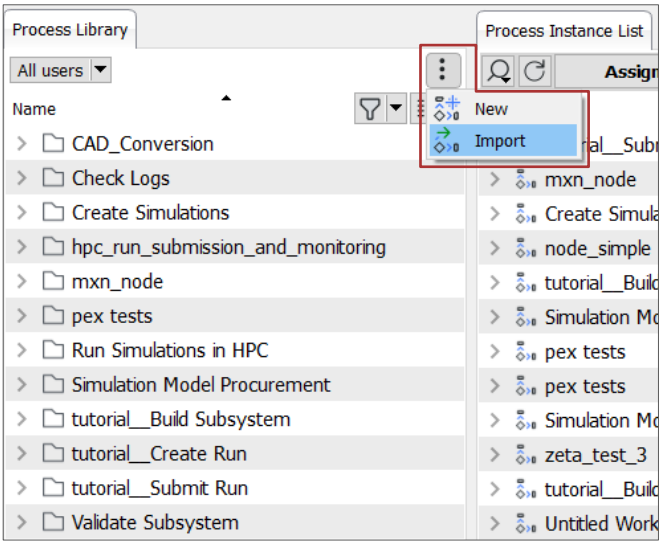


During the export, the user has the option to take along with the process all the referenced DM scripts. Additionally, both the process definition and all the related scripts can first be encrypted and then exported.



5.6. Import processes

Exported processes can be imported in the *Process Library*, using the **Import** option from the button  or from the context menu.



The user is prompted to select the *.json or *.ser files.

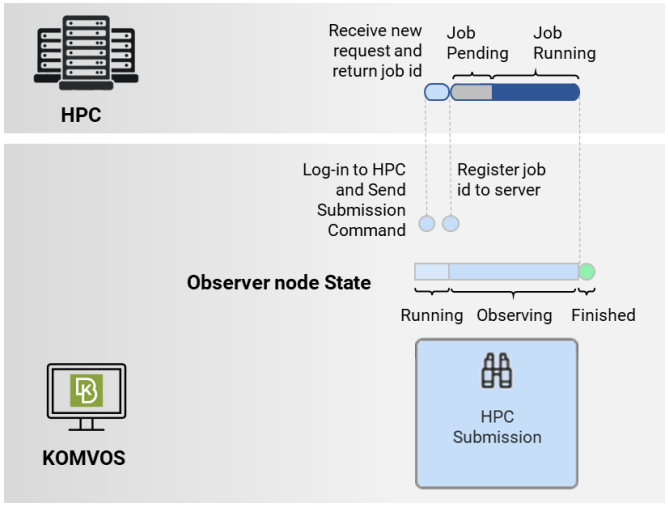
During the import, KOMVOS automatically creates new versions of nodes and process in case of conflicts.

5.7. HPC Jobs management

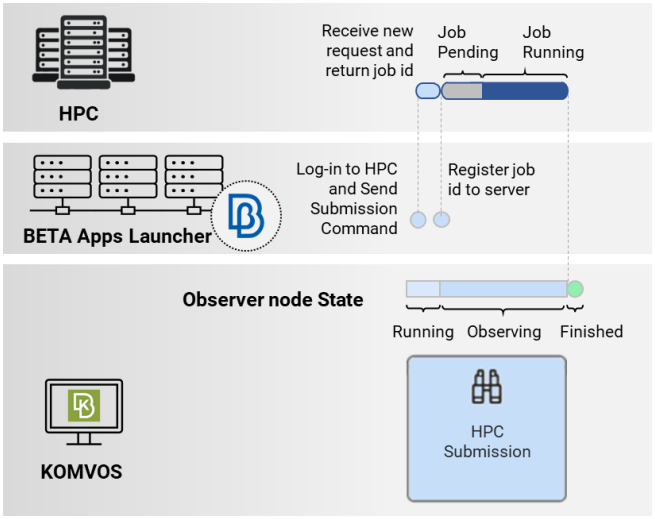
The management of HPC jobs is implemented within the framework of Process Management. It is the SPDRM back-end that offers the required infrastructure for the management of jobs on remote HPC systems, which includes:

- Submission of Jobs to HPC
- Monitoring of job status
- Notification on job completion and automatic advancement to the next steps of the process
- Possibility to cancel a job
- Possibility to perform Submission through remote resources

Through a process, job submission to the HPC is handled with an Observer node. The Observer node handles all the steps of the process, from the submission to the job termination. An insight into the actions handled by the Observer, is given in the diagrams below. More information on the execution of tasks on remote resources is given in paragraph 0.



Execution on local resources



Execution through remote resources

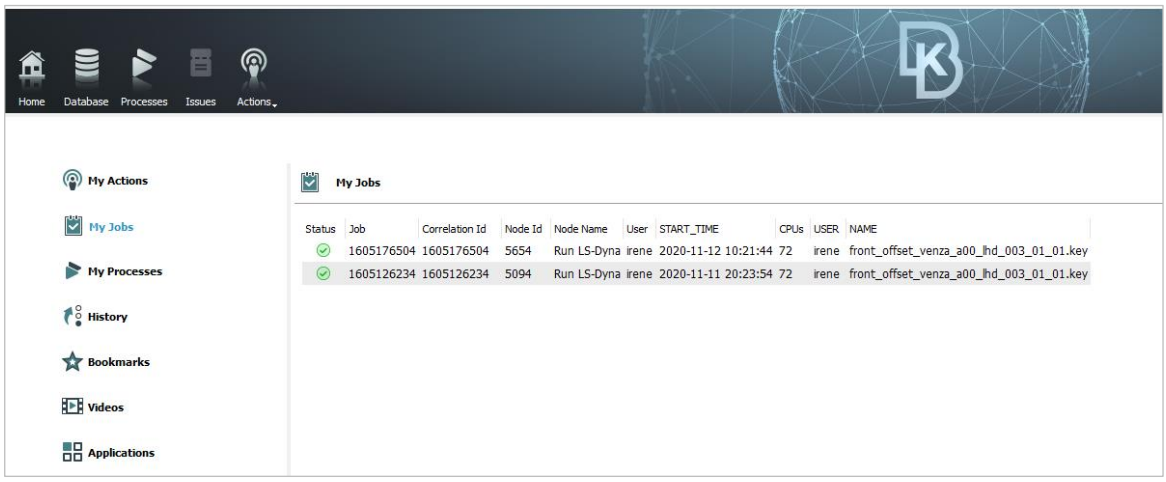
Within this sequence of actions, the step **Register job id to server** is crucial for the management of a job. During this step, the Observer node is linked to one job id on the HPC side. On the HPC side, this job id may correspond to a bundle of sub-jobs (e.g. sub-jobs that represent the different submission phases, sub-jobs in case of pedestrian protection simulations, sub-jobs in case of DOEs, etc.). SPDRM-side, this identifier is called the **correlation id**.

The management of HPC jobs requires one more **service**, that is responsible for making queries to the HPC at regular intervals, in order to retrieve the status of all the job ids registered through the Observer node above. The installation and configuration of this service is done by the SPDRM administrator. Part of the configuration includes the registration of the job statuses that signal job termination HPC-side as node completion signals SPDRM-side, so that all running Observer nodes whose job ids have reached termination, advance to Finished. With this service in place the SPDRM server is *aware* of the status of all submitted jobs.

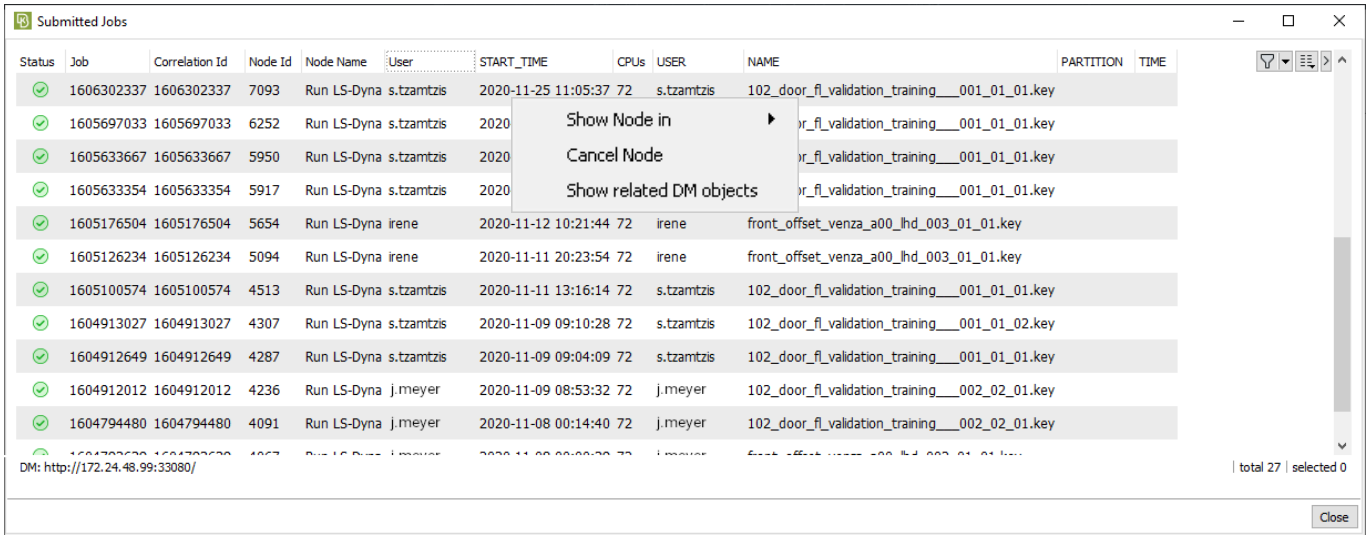
For the end-user, there are two key items related to the job submission process:

1. The HPC Submission **observer node**, whose status is monitored in the Process Instance List
2. The job id, whose status is monitored in the **My Jobs** tab in the Home workspace or in the **Submitted Jobs** list accessed through *Tools > Submitted Jobs*

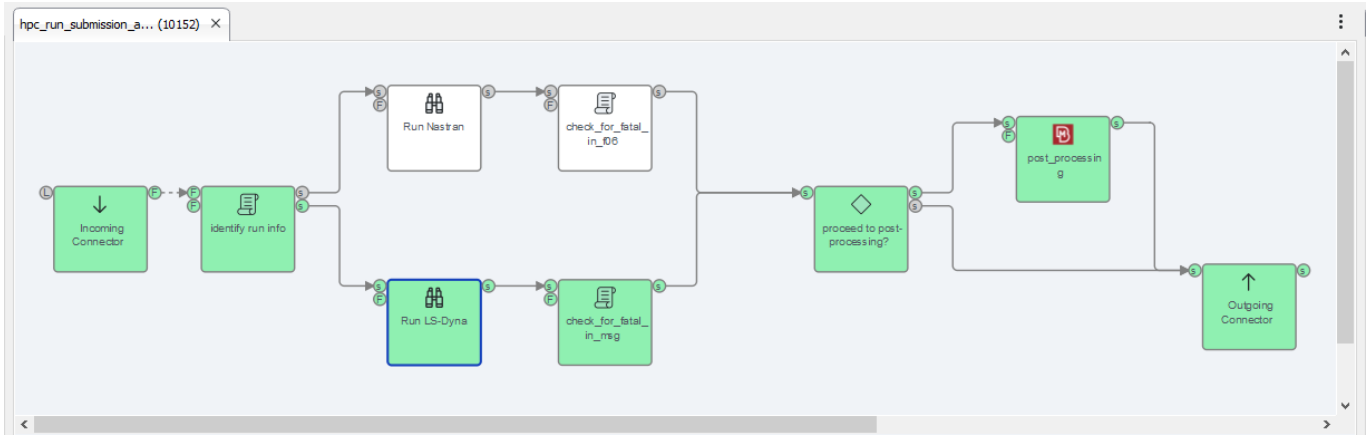
The **My Jobs** tab shows only the jobs of the logged-in user (by default, a job is still accessible one week after its completion).



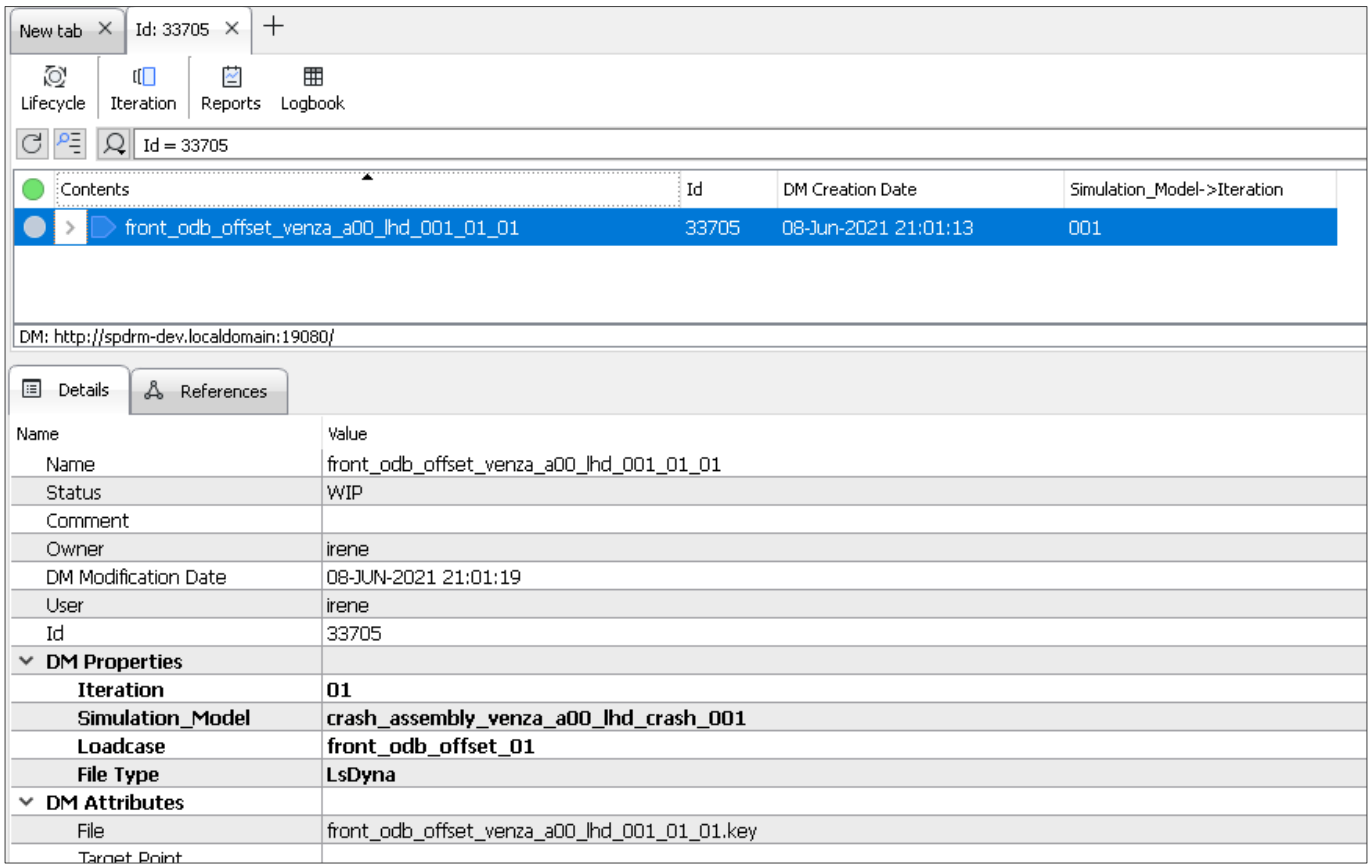
If the logged-in user belongs to the Administrators role, the **Submitted Jobs** window will show all the submitted jobs.



In both lists, it is possible for the user to be directed to the Observer node related to each job or to the Simulation Run DM Object that was submitted. Through the context menu option **Show Node in>Process Diagram** the corresponding Observer node will open in Diagram view.

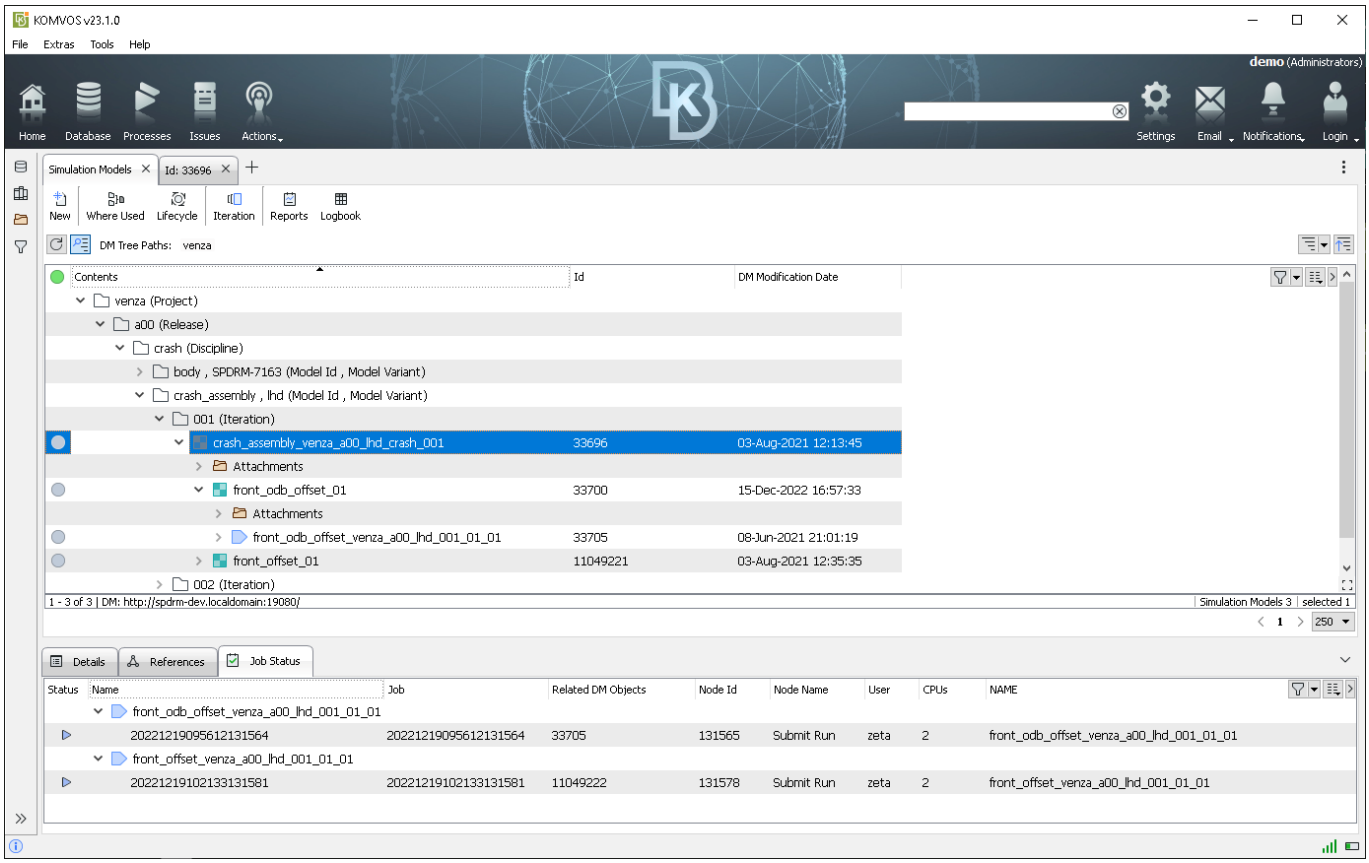


Through the context menu option **Show related DM Objects** the corresponding Simulation Run will open in the DM Browser list.



By default, a job is still accessible in the lists one week after its completion.

Moreover, it is possible for the user to be directed from a submitted Simulation Run to the Job.



In the Simulation Models list, accessing the **Job Status** bottom tab, lists all the related submitted Simulation Runs. The list offers in the respective columns, monitoring of the *Status* of the submitted jobs, as well as other information like the User who made the submission.

6. Issue Management

6.1. Introduction

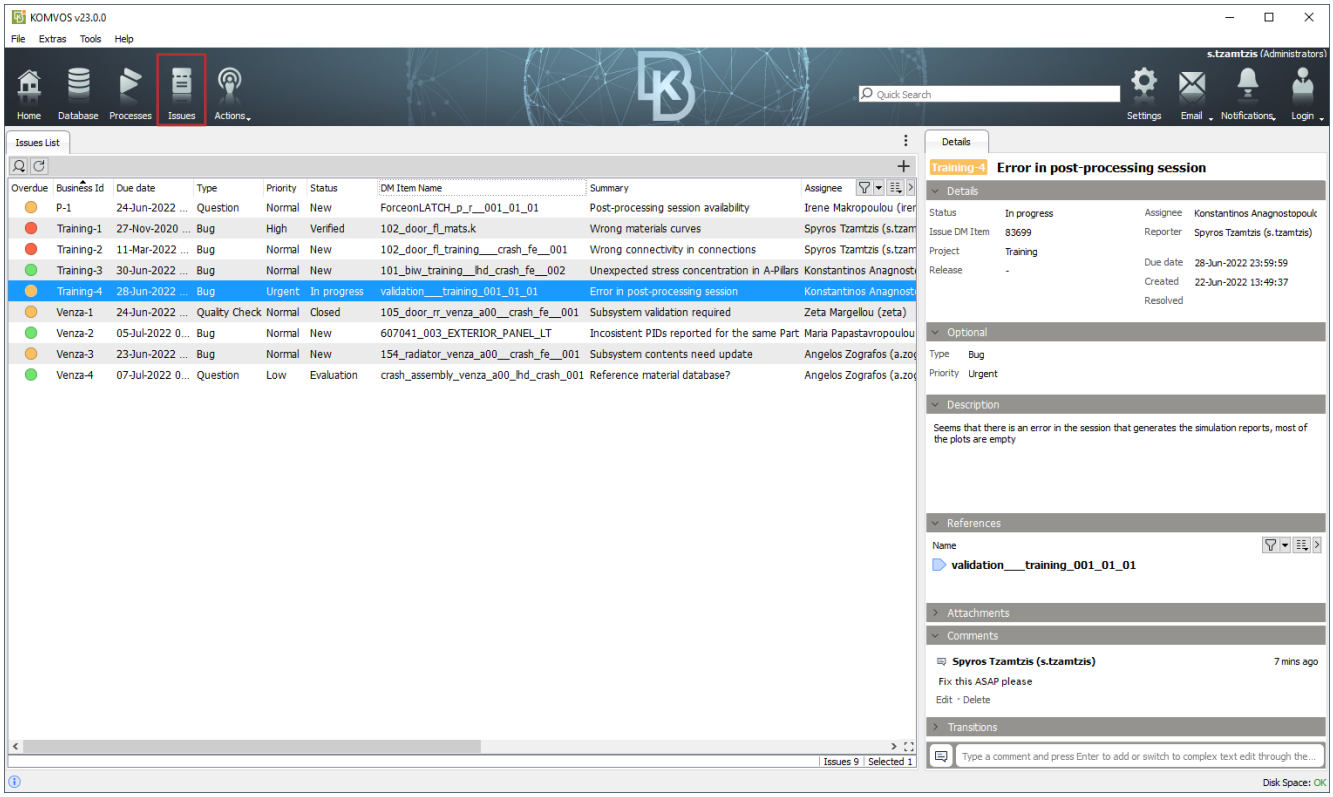
When KOMVOS is connected to an SPDRM backend, it offers an interface to SPDRM's *Issue Management* console. This is a solution that provides CAE engineers and analysts with structured procedures for the processing and management of quality issues that concern model and simulation data.

The main features of the *Issue Management* solution can be summarized as follows:

- It enables the report of issues concerning model or simulation data, directly in the SDM client.
- It offers a pre-defined dialog for the creation of an issue, through which all required data and information are captured.
- It comes with an embedded notification mechanism, ensuring that any user involved with an issue can receive an email notification when action is required.
- It offers a platform to search for simulation issues in the SPDRM database.
- It enables the automatic generation of alerts for objects that are associated with an issue. Such alerts are displayed in the Lifecycle Graph, facilitating the error impact analysis

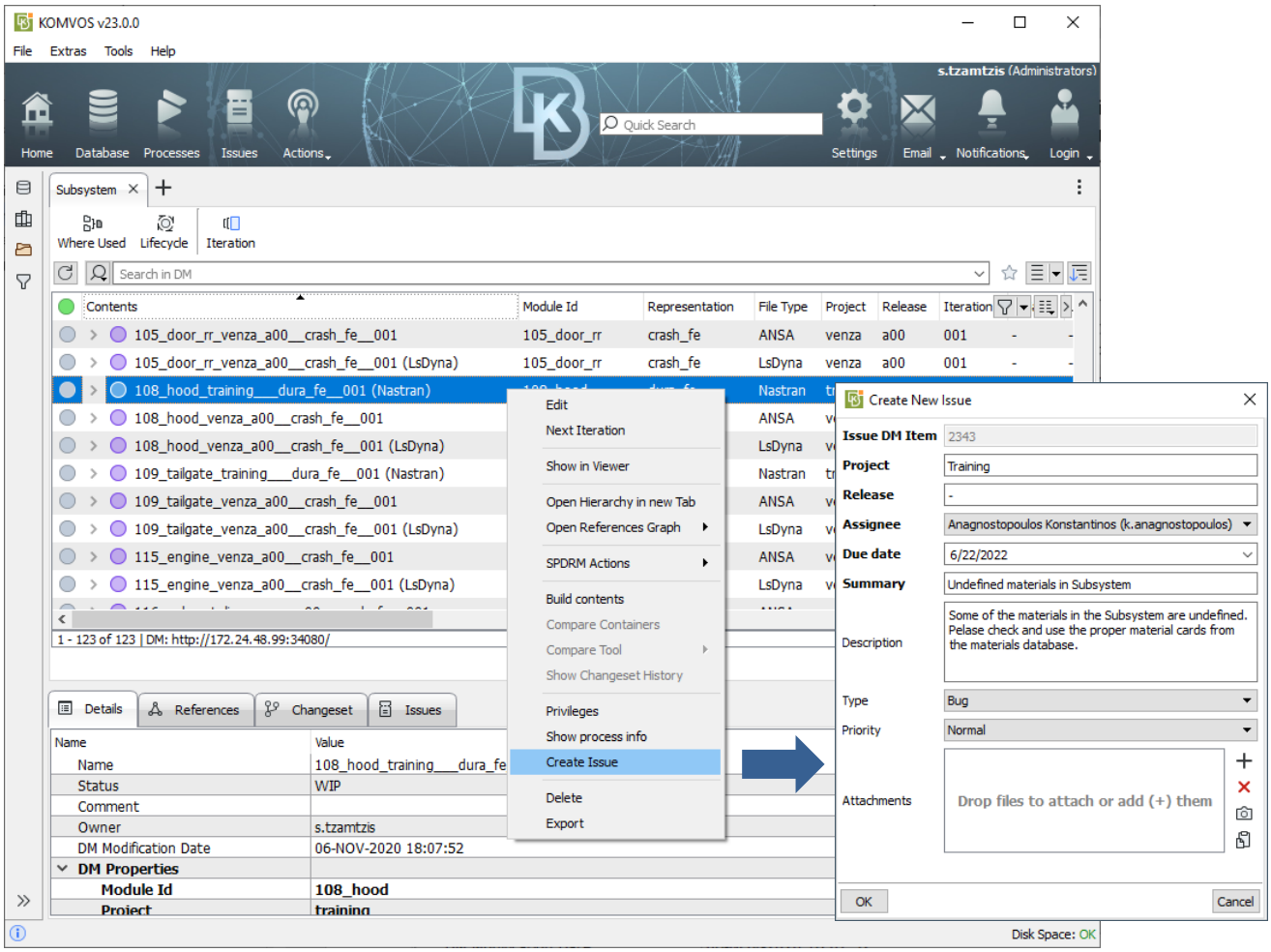
The SPDRM *Issue Management* is a fully customizable solution, which can be configured by an SPDRM system administrator to adapt to the requirements of each enterprise. A detailed description of the *Issue Management* configuration is given in the SPDRM Administrator's Guide.

In KOMVOS, a dedicated **Issues** workspace is activated when connected to an SPDRM backend of v1.8.0 or later, facilitating the display and search/filtering of existing issues.



6.2. Creating Issues

One of the key features of the *Issue Management* functionality is that it facilitates the creation of issues directly related to the model or simulation data of interest. Selecting a DM object in the *Database* workspace of KOMVOS, the context menu option **Create Issue** can be used to invoke the issue creation dialog.



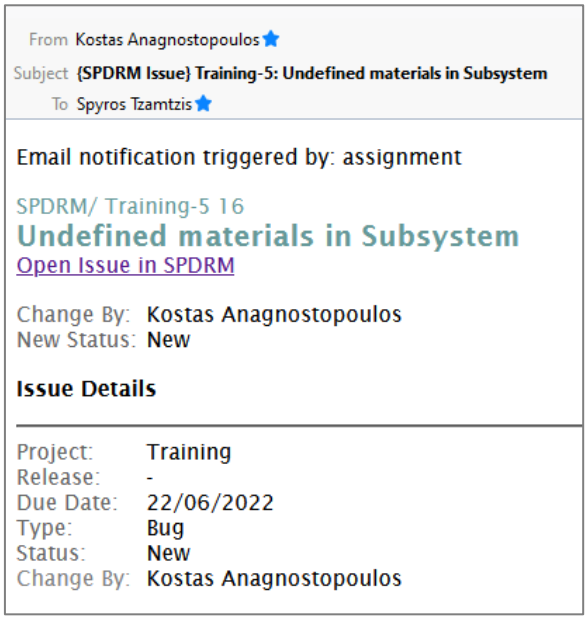
The fields of the *Create New Issue* window consist of a series of mandatory and optional issue attributes that are required for the creation of a new issue. Mandatory attributes are displayed in bold, while optional are displayed in a regular font. All fields can be pre-configured by an SPDRM system administrator, refer to the Data Management chapter of the SPDRM User's Guide for more details on the customization capabilities of the *Issue Management* functionality.

In the default *Issue Management* configuration, the following fields will appear in the *Create New Issue* window:

- The **Issue DM Item** is a key attribute, which associates an issue with a specific DM object and can trigger the inheritance of certain properties (i.e. the *Project* and the *Release*) to the issue. It is automatically filled in with the Id of the DM object for which the issue is created.
- The **Project** and **Release** are fields that should correspond to the respective attributes of the DM object for which the issue is created.
- The **Assignee** is the user responsible for the handling of an issue at each phase. Each time the assignee of an issue is modified, an email notification is sent to inform the respective user for the assignment of an issue and provide issue-related information. A notification also appears in KOMVOS, in the *Notifications* area.

- The **Due Date** facilitates the monitoring and management of issues and enables CAE engineers and analysts to prioritize their tasks,.
- In the **Summary** field brief information regarding the issue contents is provided, while a more detailed **Description** can be given using the respective field.
- The **Type** is a field that can be used to categorize the issues.
- The **Priority** field is used to characterize priority with which an issue should be handled
- The **Attachments** field may be used to attach any additional files that may be relevant to the issue (e.g. an image, a checklist, etc.)

Once all mandatory attributes are filled, the issue can be created pressing the **OK** button in the dialog. Alternatively, the creation of an issue can be aborted using the **Cancel** button.


From Kostas Anagnostopoulos
Subject {SPDRM Issue} Training-5: Undefined materials in Subsystem
To Spyros Tzamtzis

Email notification triggered by: assignment

SPDRM/ Training-5 16
Undefined materials in Subsystem
Open Issue in SPDRM

Change By: Kostas Anagnostopoulos
New Status: New

Issue Details

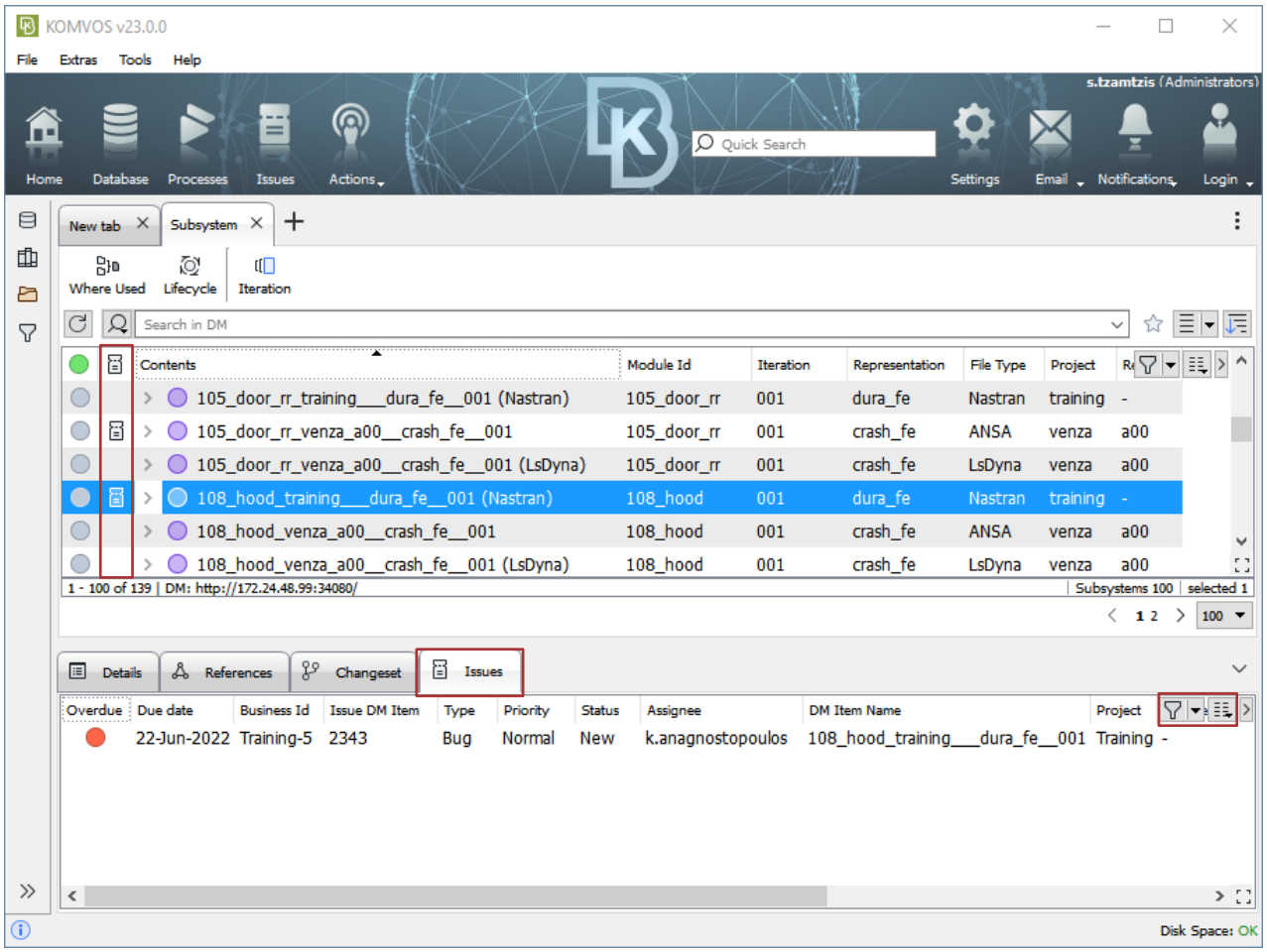
Project: Training
Release: -
Due Date: 22/06/2022
Type: Bug
Status: New
Change By: Kostas Anagnostopoulos

When an issue is successfully created, the *Assignee* will receive an email notification.

Both the subject and the main body of the email notification can be customized by an SPDRM system administrator, in order to convey to the users the relevant information for the assigned issues.

6.3. Identifying Issues

The existence of issues can easily be detected in the *Database* workspace.



The **Has Issues** attribute can be added as a column in the various lists of DM object types. It will be empty for entities that are not associated with any issues, and it will display a characteristic icon for entities that are associated with issues.

Switching to the **Issues** tab, the user may access all issues associated with the selected DM object. It is possible to filter the listed issues, as well as add more attributes of the listed issues as columns, in order to have direct access to the desired information. Except for the issues attributes that are configured by the SPDRM system administrators, some built-in default attributes are also listed in the *Issues* tab:

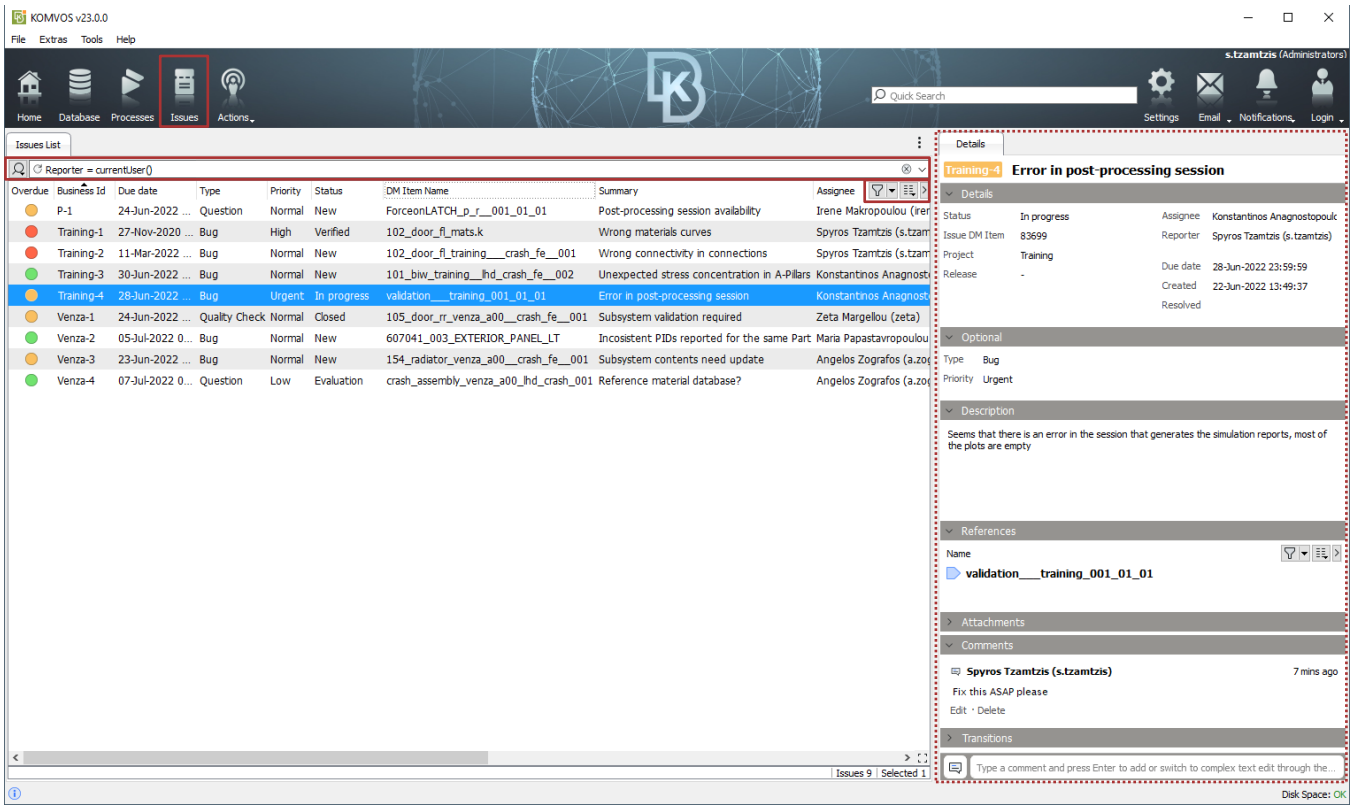
- The **Overdue** attribute assists in the management of issues and the prioritization of the issues to be handled. It displays a characteristic icon, directly linked with the *Due Date* issue attribute, which will be:
  - when the *Due Date* is more than 7 days from the current date.
  - when the *Due Date* is less than 7 days from the current date.
  - when the *Due Date* is past the current date.

  It should be noted that once an issue is resolved, the *Overdue* attribute is frozen to the state that it had on the resolution date.
- The **Business Id** is automatically generated using the issue *Project* attribute and a counter, to display the number of existing issues per Project.

- The **Created** attribute displays the issue creation date
- The **Resolved** attribute displays the issue resolution date.
- The **Reporter** attribute displays the user that created the issue.
- The **DM Item Name** attribute displays the name of the DM object that is defined as the *Issue DM Item*.

6.4. The “Issues” workspace

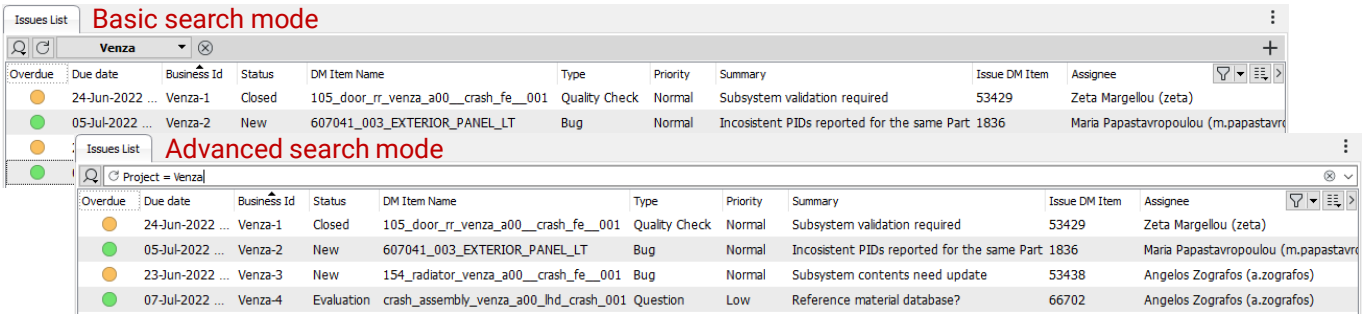
For a global overview of all issues, KOMVOS offers a dedicated workspace, which is only available when connected to an environment using SPDRM version 1.8.0 or later. Switching to the **Issues** workspace, all issues reported in the SPDRM environment can be listed. The user can search for issues, filter the listed issues, control the visibility of attributes of the listed issues as columns and view the issues of interest in detail.



6.4.1. Searching for Issues

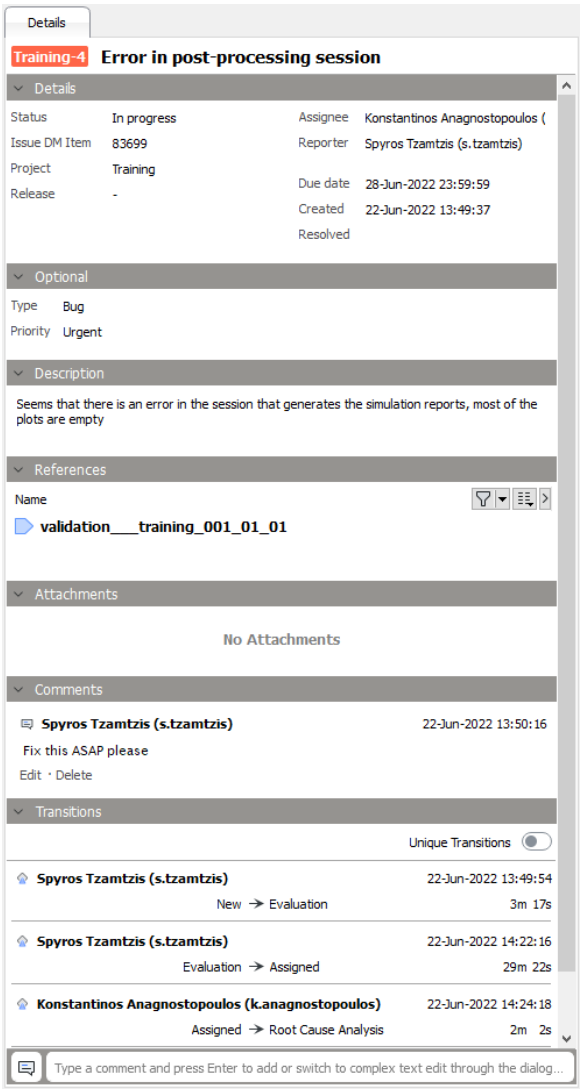
The search mechansim used in the *Issues* workspace is based on a powerful query language, the BETA Query Language (BETA QL), which is common for all BETA’s SDM solutions. In the issue management context, searches are performed based on the metadata of the issue items, in a similar manner to how searches are performed in the context of data management. A detailed description of the search functionality that enables the identification and focus on data of interest can be found in Chapter 3 of the Data Management reference guide.

In the *Issues* workspace, search is performed in two possible modes: *Basic* and *Advanced*. The mode of filtering can be selected by pressing the **Filter** button. Once the query is applied, the matching issues will be listed in the workspace.



6.4.2. Viewing Issues

Selecting any of the listed issues in the *Issues* workspace, the **Details** tab offers a detailed view of the issue attributes and additional information. The displayed information is organized in discrete sections:



- At the top, the issue *Summary* and *Business Id* appear.
- The *Details* section lists the mandatory issue attributes.
- The *Optional* section lists optional issue attributes.
- The *Description* section holds the issue description.
- The *References* section lists the issue DM item, as well as any other related DM objects.
- The *Attachment* section lists any issue attachments.
- The *Comments* section lists a full history of all comments exchanged between users in the scope of the issue.
- The *Transitions* section lists the full history of issue state transitions. It may be utilized as a project management tool, providing information on the time spent on each state. Activating the *Unique Transitions* flag, the tool displays the accumulated time spent on a transition between two states. A counter will also appear to indicate the number of times that this transition took place.
- At the bottom, new comments can be typed and added to the issue.

It is also possible to modify the Issue attributes through the **Details** tab. Where applicable, when the user hovers the mouse cursor over the attribute values, a characteristic icon will appear next to the fields that can be edited. Pressing the edit button, the field value can be modified and the changes can be applied or discarded, using the respective buttons that appear .

7. Actions

KOMVOS offers the option to define and execute custom actions, in order to provide users with additional capabilities, extending the built-in functionality for data and process management. Such custom actions need to be pre-configured by a system administrator.

There are two main types of custom actions:

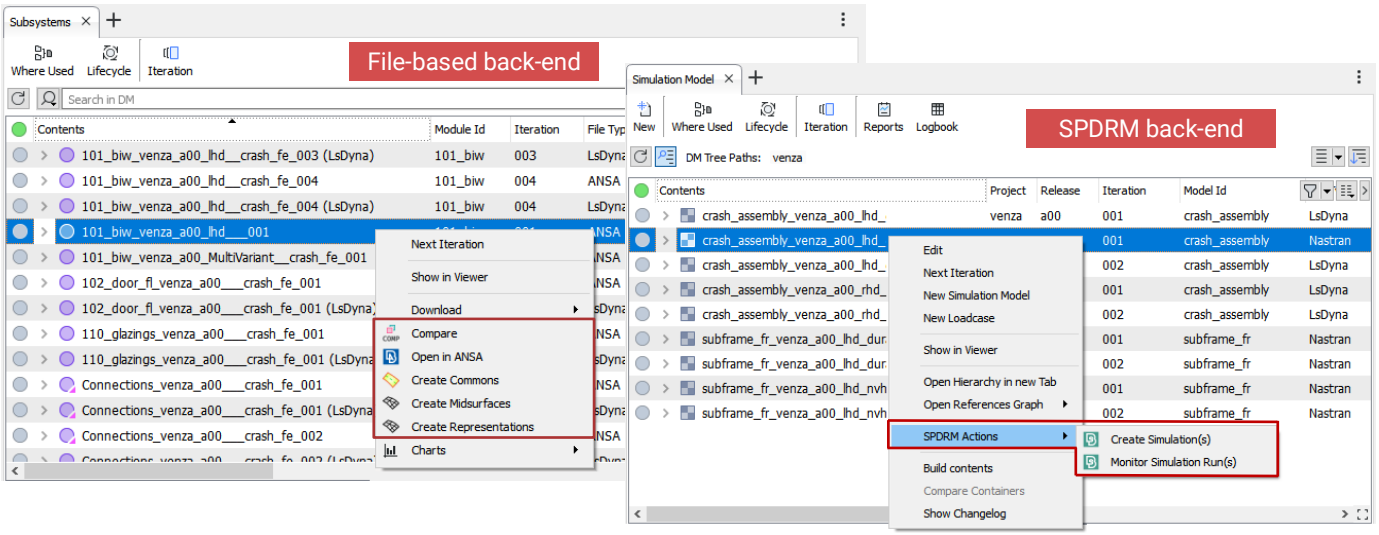
- Actions specific to particular DM object types, which are accessed through the context menu on DM Objects
- Generic actions, which do not apply to a specific DM object type and are accessed through the **Actions** button in the main toolbar

Regardless of their type, these actions are in essence scripts, which are executed when the respective option is selected. Two technologies are offered, depending on the SDM back-end that KOMVOS is connected to:

1. **KOMVOS Actions**: These custom actions:
    - Are written using the scripting API provided by KOMVOS (`sdm` Python module)
    - Are configured in KOMVOS application home (`$SDM_CONSOLE_HOME`)
    - Are executed by KOMVOS
    - Are applicable to any kind of SDM back-end, file-based or server-based
2. **SPDRM Actions**: These custom actions:
    - Are written using the scripting API provided by SPDRM (`SPDRM` Python module)
    - Are configured through the SPDRM Administrator Console
    - Are executed by PEX (SPDRM's Process Executor module). This means that these actions provide additional capabilities like for example SPDRM process instantiation, execution on remote resources, as well as Issue Management.
    - Are only applicable to SPDRM back-end

7.1. DM object actions

Actions specific to DM object types can be accessed through the context menu of the DM objects. Depending on the SDM back-end and their configuration, these actions will be listed either at the top level of the context menu, or under a grouping item labelled *SPDRM Actions*.



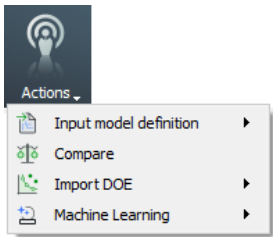
The Python scripts to be executed as KOMVOS Actions reside within the application home folder (`$SDM_CONSOLE_HOME`), which is, by default, the *config* directory of the KOMVOS installation. A custom action can be defined by associating a script function either with a DM object type, so that it's available in all views where DM objects of this type are shown, or with a particular data view (e.g. with the Default view of Subsystems but not with the Flat view). This association is done in the `dm_views.xml`. Information on how custom actions are registered on DM object types and views can be found in Chapter 5 of the Data Management Reference Manual.

KOMVOS comes pre-configured with a series of ready-to-run DM object actions, which are available in the default configuration used with file-based back-ends. A detailed description of how to use these actions is given in Chapter 8 of this guide. In this configuration, custom DM object actions are defined in the script file `right_click_actions.py` that can be found in the *config* directory in KOMVOS installation.

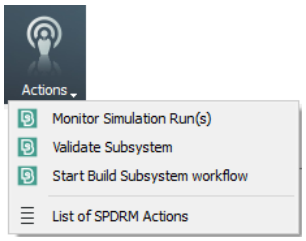
When KOMVOS is connected to an SPDRM back-end, the configuration of custom SPDRM DM object actions (i.e. *DMItem* and *LibraryItem Script Actions)* is performed by the SPDRM system administrator. A detailed description of how to set up SPDRM script actions is given in the SPDRM Administrator's guide.

7.2. Generic actions

Using the **Actions** button in the main toolbar of KOMVOS, various general actions that do not apply to a specific DM object type can be accessed. There are two categories of *Actions* in KOMVOS:



Generic Actions



SPDRM Actions

Generic actions are available when connected to either a file-based SDM back-end or a server based SPDRM back-end. These are defined in the `dm_actions.xml` file that is stored in the configuration folder of KOMVOS. A detailed description of how to configure generic actions is given in Chapter 10 of this guide.

SPDRM actions are available only when connected to an SPDRM back-end. They can be distinguished from generic actions by the characteristic SPDRM icon. These are defined by an SPDRM administrator and are part of the system configuration. A detailed description of how to set up SPDRM script actions is given in the *Additional Topics* chapter of the SPDRM Administrators guide.

Using the default configuration for a file-based back-end, four generic actions are supported:

- **Input model definition**: used in order to import a dataset exported from a PDM system and reflect the product structure in the Part Manager tool. For more information regarding this action, please refer to in Chapter 8 of this guide.
- **Compare**: used in order to compare two files in ANSA. The **Compare Tool** is used for the comparison between two models, ANSA files, solver input files (Nastran, LsDyna, Pam-Crash, etc.), connection files (VIP, VIP2, xml) or any combination of the above and the identification of differences in geometries, connections, solver entities and their attributes. Initially, the matching process classifies the models' entities into matched and unmatched, and then, the diffing process identifies the differences of matched entities. For an extensive description of the Compare Tool, please refer to the corresponding chapter of the ANSA User's Guide document, *Model Update and Comparison*.
- **Import DOE**:

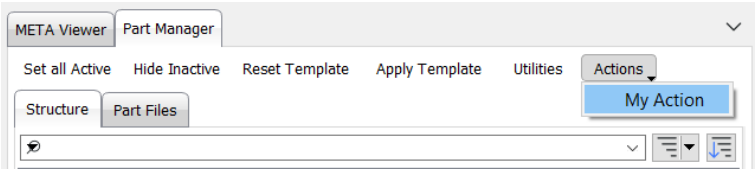
- ***Import DOE Study***: used in order to import to the current DM data from existing DOE studies stored in a *.txt or *.csv file.
- ***Import DOE Parametric Data***: used in order to import to the current DM data from existing DOE studies stored in a *.txt, *.csv or *.xlsx file format in cases where the parameterization was not done in ANSA.

For a more detailed description about how to manage DOE Studies in KOMVOS, please refer to chapter 9 of this guide.

- **Machine Learning**:
  - ***Train Embedded Clips Predictor (Local)***: used in order to train an Embedded Clips Predictor, which can learn to recognize the Clips provided as input to the action. For a more detailed description about how to use this functionality, please refer to chapter 9 of this guide.

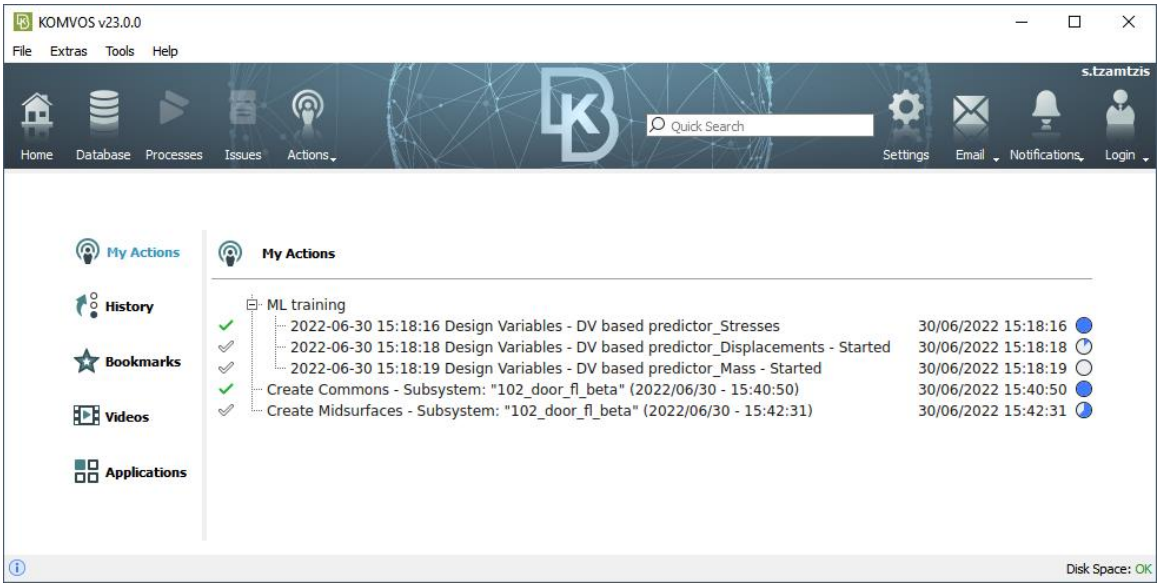
7.3. Part Manager Actions

It is also possible to show user defined generic actions in the Part Manager tool of KOMVOS under the **Actions** menu, as shown below. This can be achieved by configuring the `dm_actions.xml` file according to the guidelines given in Chapter 10.3 of this guide.



7.4. Monitoring the progress of KOMVOS Actions

KOMVOS offers progress monitoring functionality for KOMVOS Actions, that allows the users to check the execution status of the script actions they have initiated. This is done using the *My Actions* shortcut in the *Home* workspace.



In order for an action's progress to be monitored, the action script needs to create an action log file, following a fixed format. The action log file needs to first be registered in the SDM User Settings and is then parsed by KOMVOS regularly, in order to update the action status in the *My Actions* list in the *Home* workspace.

It is recommended to use as reference the scripts used by the custom actions of the default configuration for file-based back-ends (e.g. `right_click_actions.py`, `dm_scripts.py` and `general_fucntions.py`). These are located in the *config* folder in the installation directory of KOMVOS.

A code snippet with the formatting that is required for the creation of the action log file, its addition to the SDM User Settings and the update of the action progress is given below.

Code snippet

```
# Script function to be executed when action is selected
def my_custom_action():
    ...
    ...
    # write the action log file
    log_file = write_action_log_file(work_dir, action_name, descr)
    # add log file to SDM User Settings to enable progress monitoring
    addLogFileToMyActions(log_file)
    # update action progress
    updateLogFileInSDM(log_file, perc, message)
    ...

# Sample function for the creation of the action log file.
# A fixed format is required in the log file, so that KOMVOS can parse it and
# update the action progress.
def write_action_log_file(work_dir, action_name, descr):
    now = datetime.datetime.now()
    if not os.path.exists(work_dir):
        os.makedirs(work_dir)
    log_file_base_name = now.strftime("%Y_%m_%d_%H_%M_%S_") + action_name + ".log"
    full_path_name = os.path.join(work_dir, log_file_base_name)

    final_action_name = '{action} ({now_formatted})'.format(
        action=action_name,
        now_formatted=now.strftime("%Y/%m/%d - %H:%M:%S")
    )
    f = open(full_path_name, 'w')
    f.write("ACTION: " + final_action_name + "\n")
    f.write("USER: " + CURRENT_USERNAME + "\n")
    now_for_initialization = now.strftime("%Y%m%d%H%M%S")
    f.write("START_TIME: " + now_for_initialization + "\n")
    f.write("UPDATE_TIME: " + now_for_initialization + "\n")
    f.write("STATUS: " + "Started" + "\n")
    f.write("PERCENTAGE: " + "0%" + "\n")
    f.write("DESCRIPTION: " + descr + "\n")
    f.close()
    return full_path_name

# The action log file must be registered as an SDM User Setting. The setting key
# will hold the full path to the action log file, no value is required.
# Any SDM User Settings that end in ".log" and exist, are considered by KOMVOS as
# action logs. Their content will be parsed and their information will appear in
# the "My Actions" tab.
def addLogFileToMyActions(log_file):
    my_map = { log_file: "" }
    dm.SetUserSettings (my_map)


# Update the action progress in the action log file
def updateLogFileInSDM(log_file, perc, message):
    fs_write = open(log_file + "_copy", "w")
    fs_read = open(log_file)
    lines = fs_read.readlines()
    fs_read.close()
    for i in range(0, len(lines)):
        cur_string = lines[i]
        if cur_string[:10] == "PERCENTAGE":
            fs_write.write("PERCENTAGE: " + perc + "%\n")
        elif cur_string[:6] == "STATUS":
            fs_write.write("STATUS: " + message +"\n")
        else:
            fs_write.write(cur_string)
    fs_write.close()
    shutil.move(log_file + "_copy", log_file)
    return
```

8. Model Building Tools

KOMVOS comes with an out-of-the-box customization layer that facilitates CAE Model Build from CAD/PDM info. With this functionality, KOMVOS serves as an orchestrator for pre-processing tasks, which, powered by the high-end functionality of ANSA and META, can deliver models of high quality in the least possible time.

For the moment, the CAE Model Build functionality is only available for file-based SDM back-ends and all processing is done on the executing user's workstation. The customization is implemented with Python scripts that are available within the default home folder of the application, in the *config* directory of the installation. Engineering teams can modify the existing scripts to tailor the processes to their needs or replace them by their existing automations.

Within KOMVOS, processing is done on Part, Subsystem and Simulation Model level. The supported Model Build process includes the following steps:

- Import PDM/CAD data into the SDM back-end
- Process model data, starting with CAD files translation and going down to the creation of meshes
- Update the CAE model when an update of the PDM/CAD data becomes available

Additionally, a number of tools are available to provide overview of the running processes, comparison of models, identification of carry-over parts, etc.

This chapter describes all related tools and processes. Furthermore, a step-by-step description of the supported workflows is also given in the **tutorial** *Model build simplified in KOMVOS*, accessible through the documentation index.

8.1. Import Model

The first step in the model build workflow of KOMVOS is the creation of Subsystems. Subsystems represent the CAE subassemblies and consist of parts that are structured in a hierarchical way. In most of the cases, the data source for the creation of Subsystems is a product structure extracted from a PDM System.

KOMVOS offers functionality to import a product structure coming from a PDM system automatically and visualize its content. This can be done by importing a product definition file. Such files are usually exported by PDM systems and contain information regarding the product tree structure and the part attributes. Default supported formats include PLMXML, AP242 xml, VPM xml. Any other model definition can be read with the aid of a Python script.

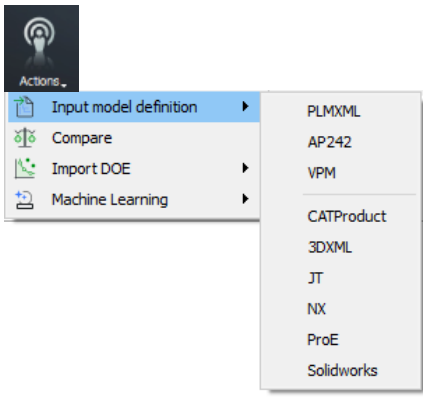
It is also possible to import assembly files of CAD systems like for example CATProduct, 3DXML, JT, NX, Solidworks, and ProE.

The model definition file is read through the **Actions > Input model definition** function.

When such a product definition file is read, the *Part Manager* window appears. This tool is used to preview the product hierarchy before importing it in KOMVOS and before further processing the referenced CAD data. Through the *Part Manager* it is also possible to preview the geometry of the referenced parts in the embedded META Viewer. Additionally, users can edit the part structure by excluding parts/ groups that are not needed in the CAE model to be created.

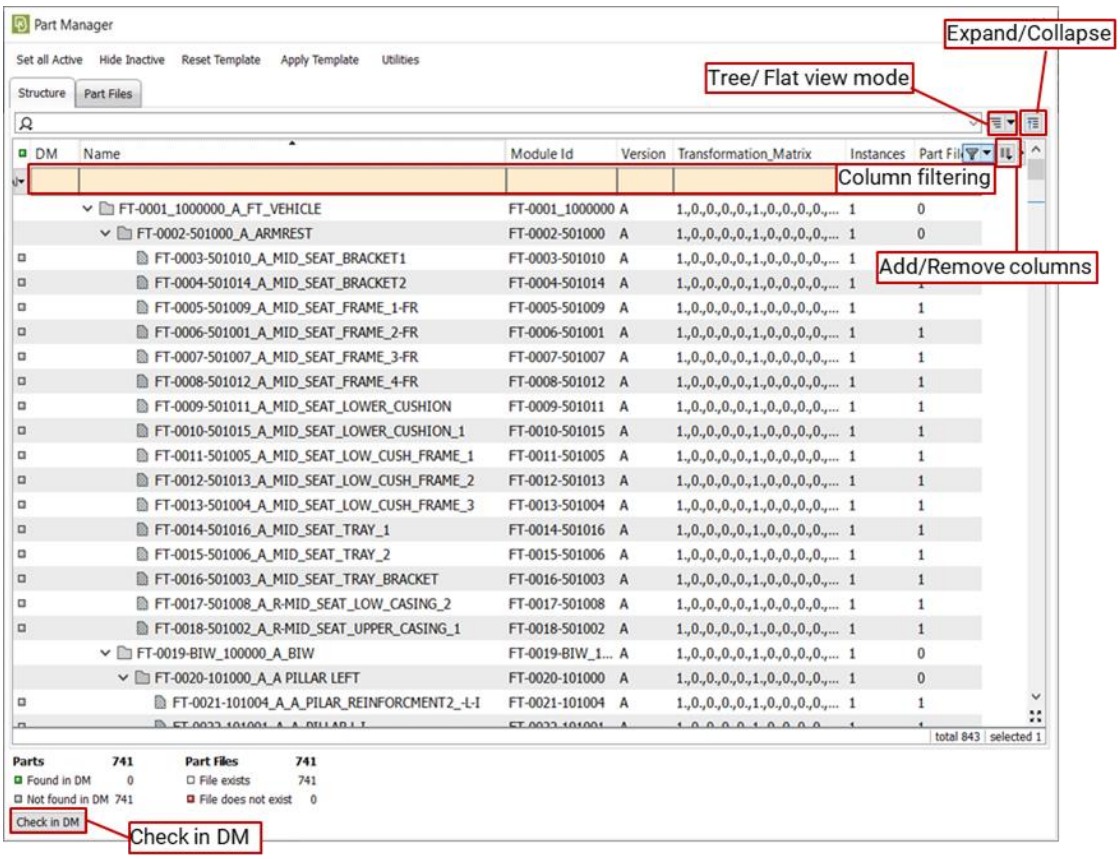
8.1.1. Reading the model definition

The assembly hierarchy information can be retrieved from the Actions button.



From **Actions > Input model definition** the product structure file can be selected with the aid of the File Manager. As soon as a file is selected, KOMVOS reads the file content and the *Part Manager* appears.

The window is separated into two main tabs, the “Structure” and the “Part Files”. In the *Structure* tab, the hierarchy of the model is displayed.



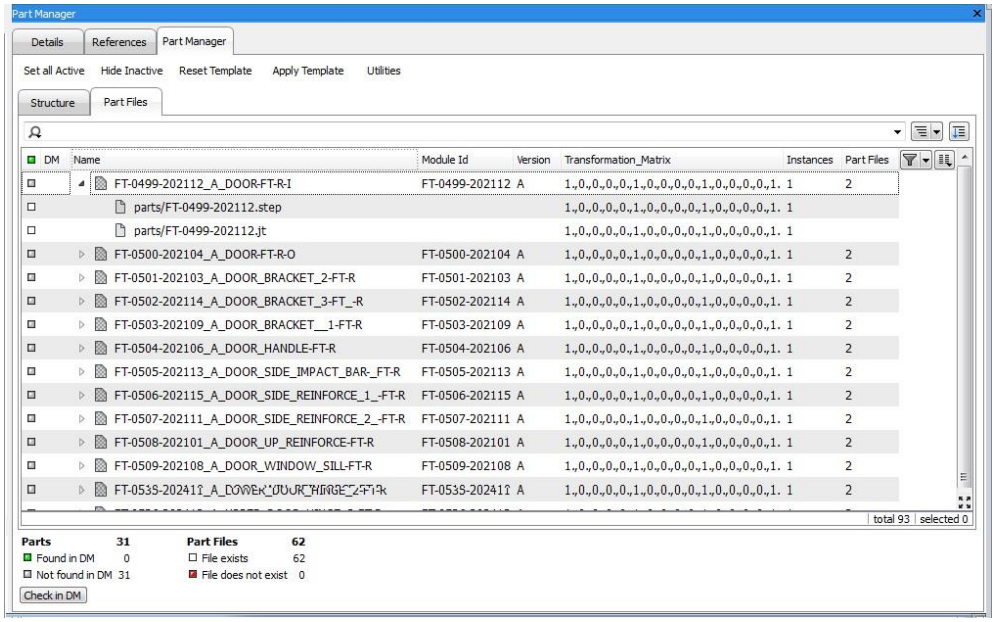
All functionalities applied to tree list windows are available here as well. The hierarchy can be expanded/ collapsed in order to have an overview of the assembly and columns can be added or removed according to user needs by pressing the respective button in the upper right area of the list.

All attributes written inside the product definition file are displayed here as columns. These will be exploited by KOMVOS and all information will be kept. The columns **Instances**, **Transformation Matrix**, **Version** and **Part Files** inform users about the number of instances of the particular part/ group that exist in the assembly, their transformation matrix, their Version and the number of physical files referred for the particular part/group, respectively.

By pressing the **Check in DM** button, KOMVOS checks in the DM repository for existing representations of the incoming parts and shows the existing representations as found from the ‘**Check in DM**’ function.

Switching to the *Part Files* tab, all unique parts that are referenced in the product structure file are displayed in a flat list, associated with their respective file(s).

Moreover, additional informative columns may be visible with metadata read from the product structure file and can be used for filtering and identification purposes. The available columns may differ, depending on the product structure format and its contents.



The moment the *Part Manager* is launched, KOMVOS checks the DM repository in order to identify which of the parts included in the assembly are already stored in DM with at least one representation. The result of this check is reflected in the status column and also in the DM column. When a part is found in the DM, its status appears in green color. Additionally, the names of the available representations appear in the DM column.

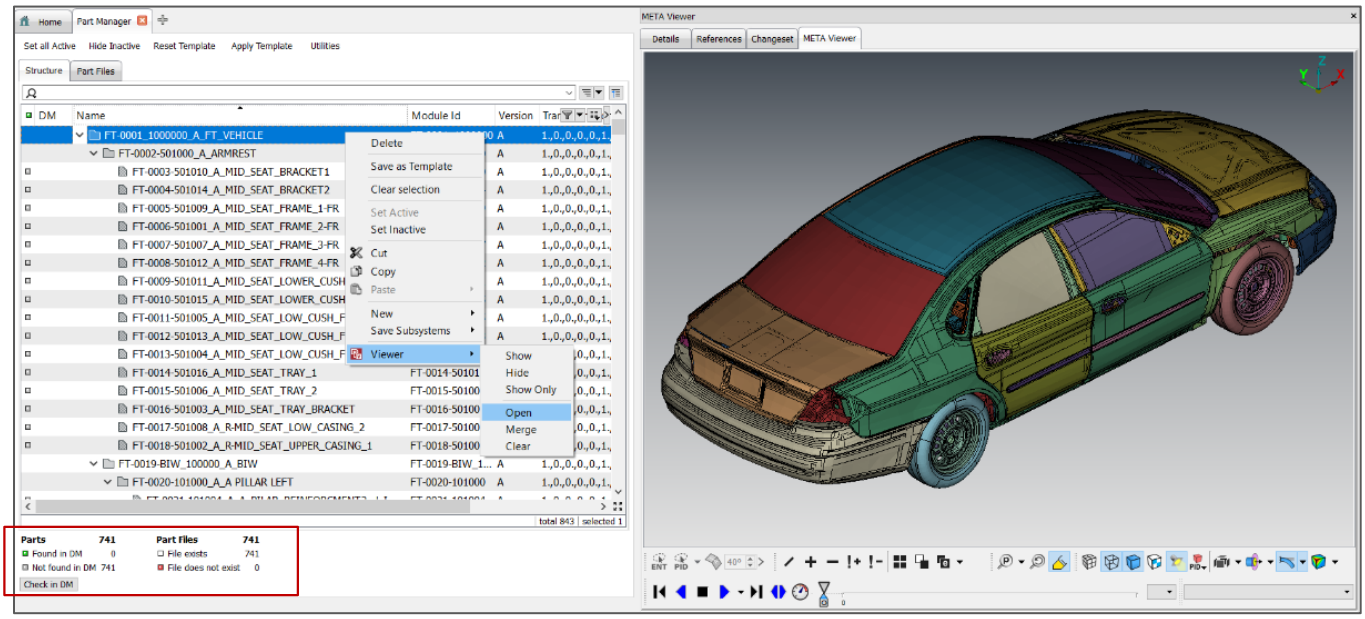
KOMVOS detects whether the CAD files exist in the physical location that is referred inside the product structure file. This information is provided in the status column, where the files that do not exist are marked with a red sign.

A summary of all the above information appears in the “Part Files” index in the lower area of the *Part Manager*.

8.1.2. Visual model inspection before Import

A graphical preview of the model is provided in KOMVOS, provided that a light weighted format of the CAD files is available in JT format.

By selecting the option **Viewer > Open** of the context menu of the outermost group or any other group of the hierarchy, the embedded *META Viewer* appears, displaying the selection. More information on the manipulation of models in the embedded META Viewer can be found in Chapter 3 of the Data Management Reference Manual.

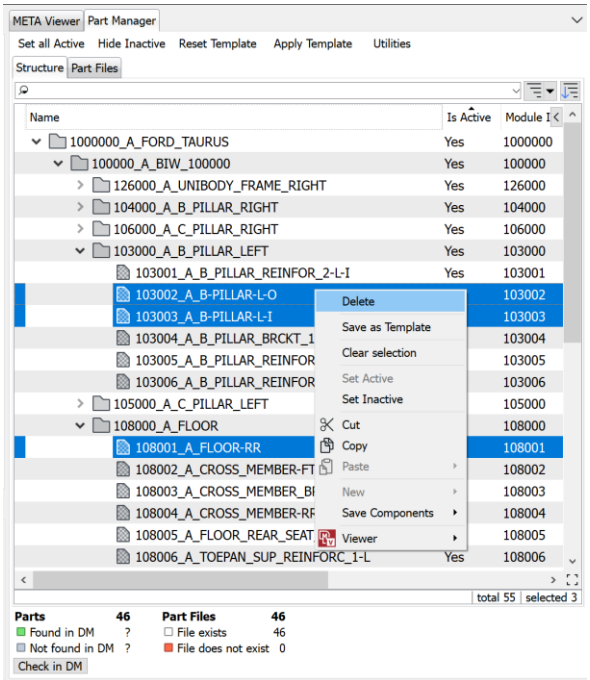


This visual inspection allows to check and verify if the correct model is displayed in Part Manager and to remove any unnecessary parts before the final import of the model into KOMVOS.

8.1.3. Modify the Product Structure

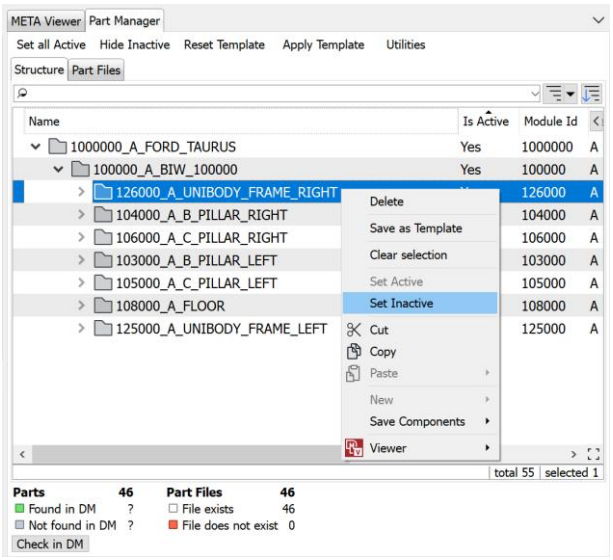
It is a common practice to edit the product structure exported by the PDM system by removing some parts/groups for the CAE model to be created. The *Part Manager* provides all the functionalities to facilitate the editing of the product hierarchy.

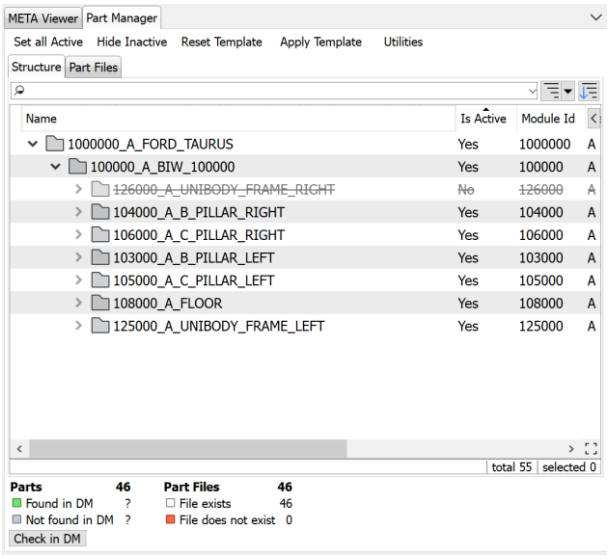
8.1.3.1. Manual modifications



Filtering operations can be performed in order to isolate parts/groups that are not needed and delete them in order to modify the hierarchy. Once the parts to be excluded have been selected, the option **Delete** from the context menu can be activated.

It is possible to manually set active or inactive specific parts/groups of the Product Structure by selecting the **Set Active** or **Set Inactive** context menu option, respectively. Depending on the state of the selected part/group the corresponding option is available. Specifically, if the group is active the context menu option available will be the Set Inactive and vice versa. The **Set all Active** toolbar option will activate all groups of the Product Structure.



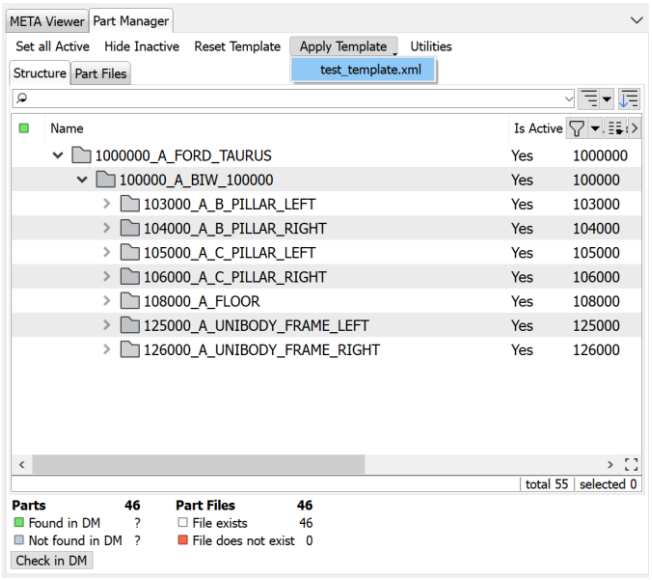


To hide all inactive parts/groups of the Product Structure press the **Hide Inactive** toolbar button. The button will remain pressed to indicate that all inactive parts/groups are not visible in the Product Structure. To show the inactive parts/groups press the button again.

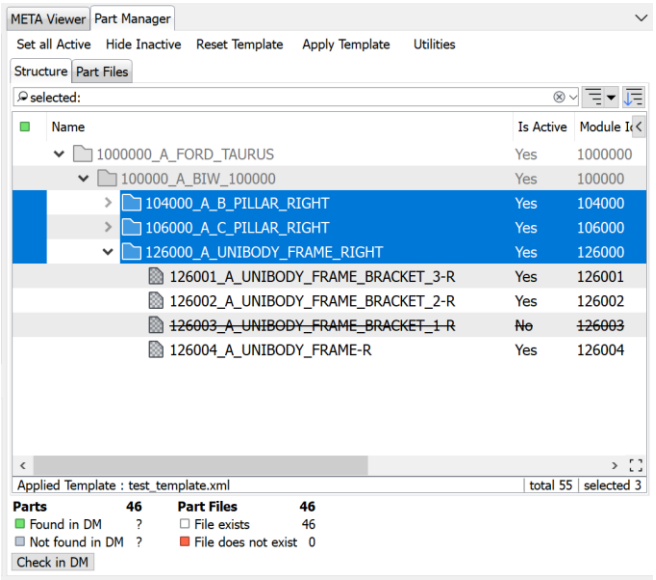
8.1.3.2. Modifications based on Subsystem Templates

To perform multiple modifications to a Product Structure according to a pattern, in a consistent and reproducible manner, the Subsystem Templates are used. Subsystem Templates are XML files that isolate the parts/groups of interest in the Product Structure, to end up with the part structure of a specific Subsystem. Subsystem Templates can be applied manually, through the **Apply Template** button, or through script. In the example below, a Subsystem Template is applied manually in the *Part Manager*.

Initial state

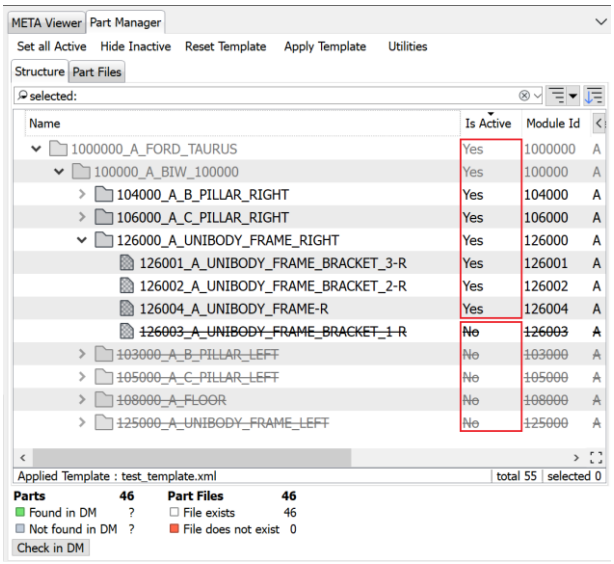


After Template application



To return to the initial state, the **Reset Template** button from the Part Manager toolbar can be selected.

To filter between active or inactive parts/groups of the Product Structure, the **"Is Active"** column can be used.



In order for a template to be listed under the **Apply Templates** menu, its XML file must be placed under the folder:

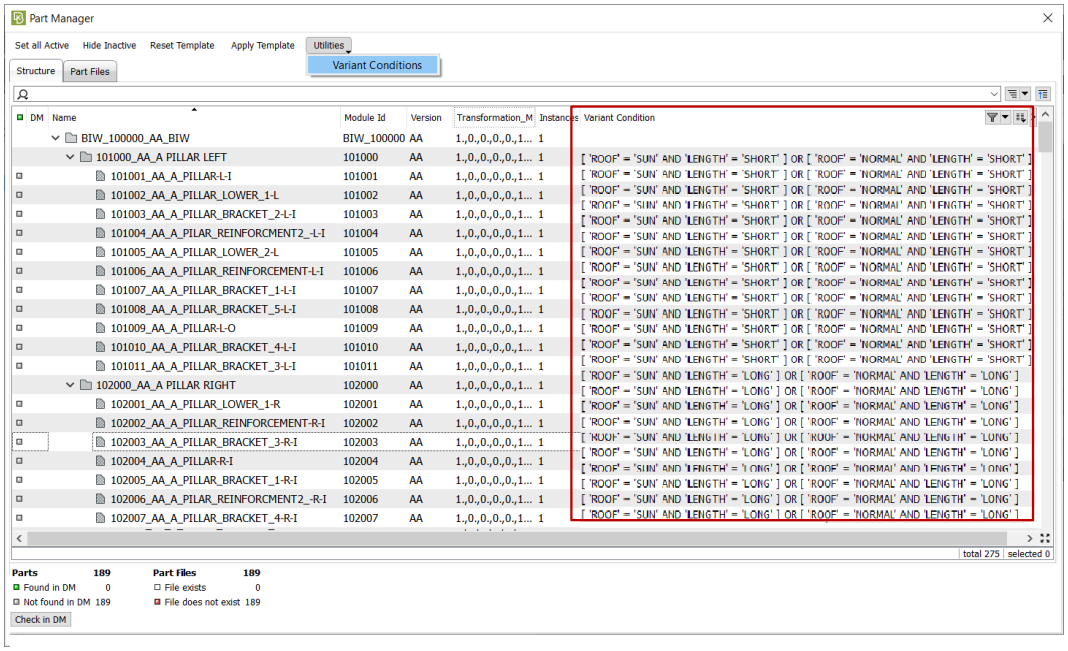
`$SDM_CONSOLE_HOME/Templates/Subsystems`

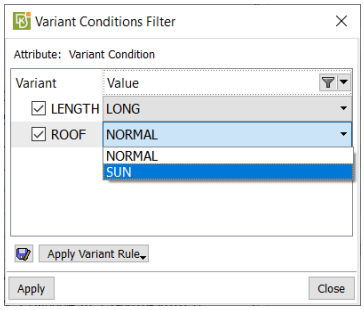
A Subsystem Template XML file can be created interactively, through the Part Manager, or with the aid of a script. More information on the format and creation of Subsystem Templates can be found in paragraph 8.6.

8.1.4. Handling model variants

KOMVOS enables the preparation of several different Variants of the same model in a single shot. The concept is to add in the DM repository parts that are used by the different Variants of the model (also known as “150% model”), and then, each time be able to retrieve the ones that form any single Variant (also known as “100% model) required. This way, time consuming processes concerning all common Parts of the different models variants are performed only once.

PLMXML model definition files can carry information that describes under which conditions a CAD Part participates to a model Variant (Variant Conditions). When such a model definition is read in KOMVOS, these conditions can be visualized by adding the **Variant Condition** column in the Structure tab. Then, using the function **Utilities>Variant Conditions** from the Part Manager toolbar, the *Variant Conditions Filter* window opens, through which particular variants can be isolated, by filtering out irrelevant parts.

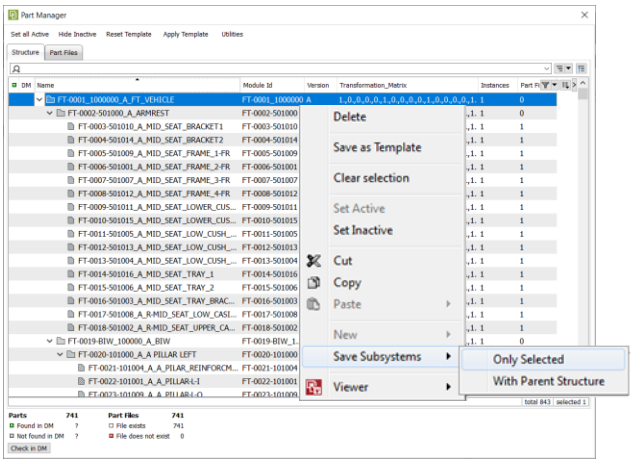




In the *Variant Conditions Filter* window the different Variants and their given Values are listed. The users can create a filter consisting of a combination of Variants' names / values which, on **Apply**, will be applied in the Parts Manager, selecting all the parts that satisfy it.

Such filters can be saved as Variant Rules using the  button. Existing Variant Rules can be reused.

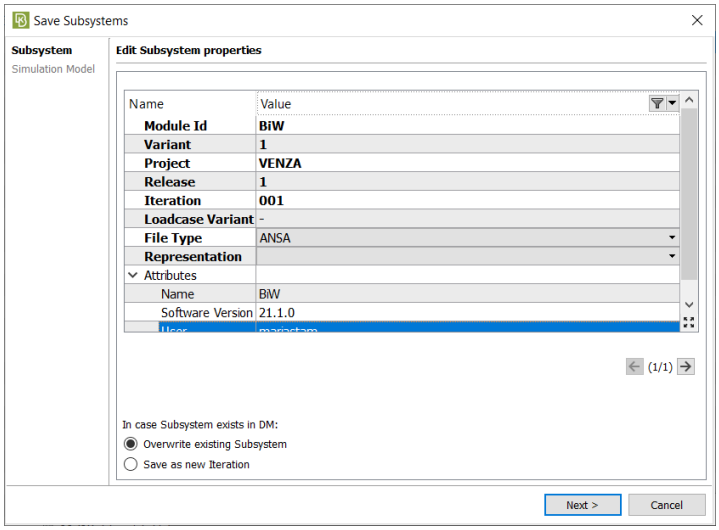
8.1.5. Create and Save Subsystems and Simulation Models



From the Structure tab of the Part Manager select the entries that should be used as content of the CAE Subsystem (e.g. the outermost group) and from the context menu select the option **Save Subsystems > Only Selected**.

! The option **With Parent Structure** is used in cases individual parts are selected (e.g. the result of a filter), in order to save the parts with their parent structure and not just as a flat list of parts.

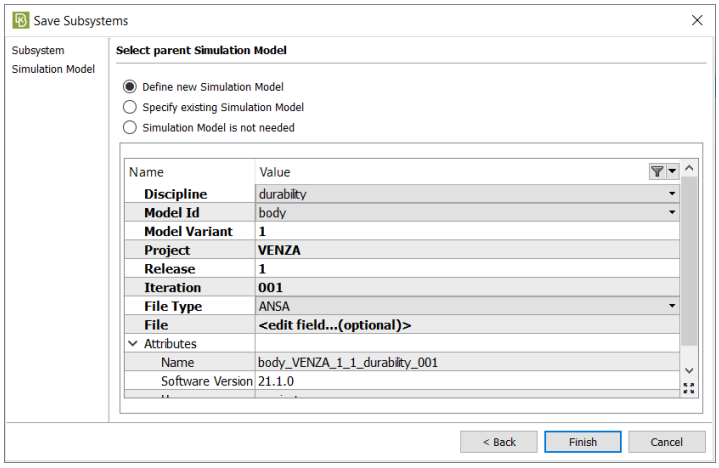
The *Save Subsystems* wizard opens. In the *Edit Subsystem* properties tab the Properties and Attributes of the Subsystem to be created are displayed.



At this point, the user should edit the displayed Properties and Attributes as desired

! A “blank” representation (empty **Representation** field) implies the “common” representation has been selected.

At the bottom, the user can define how a potential conflict should be resolved. While working with a new Subsystem select the option **Overwrite existing Subsystem** and press **Next** in order to proceed.



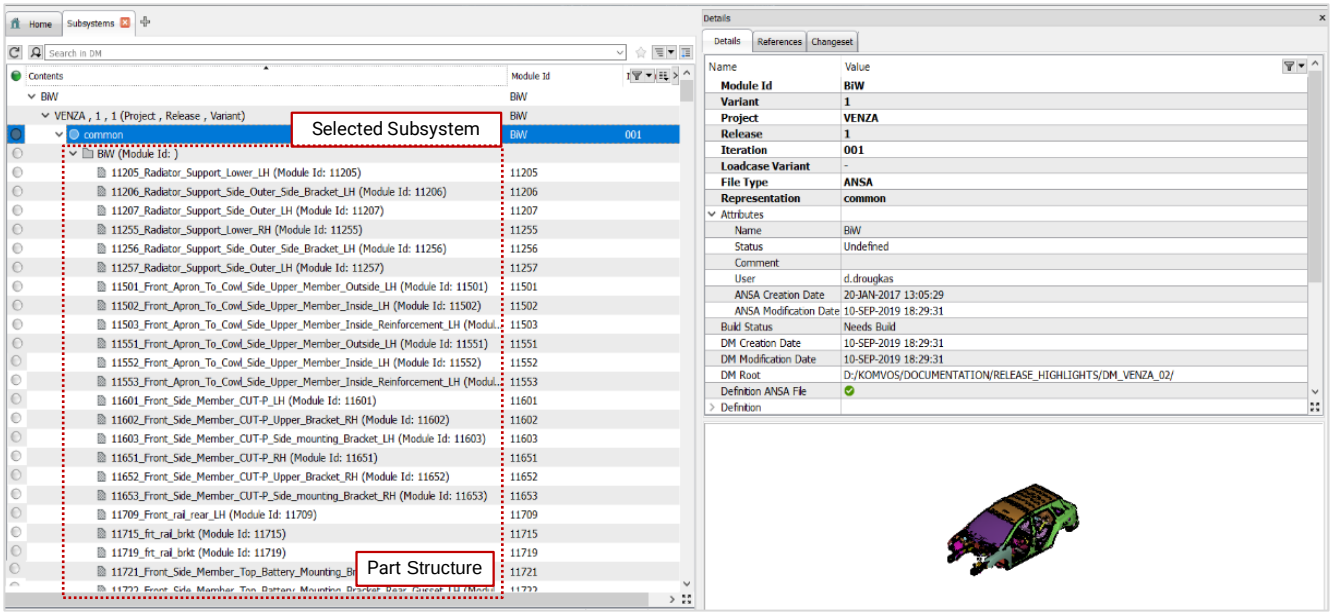
In the next page, it is possible to associate the Subsystem to be created with a Simulation Model. Leave the default selected option **Define new Simulation Model**.

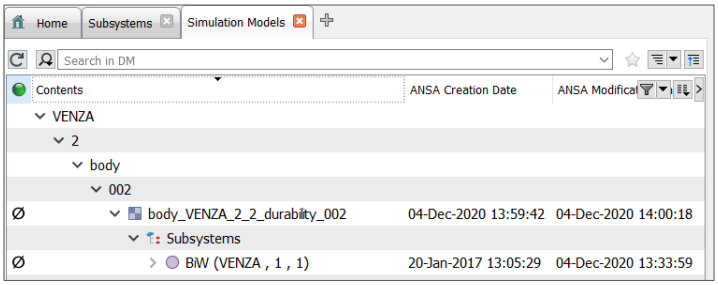
By filling the empty fields (optionally, the desired values from the respective drop-down menus can be inserted) and pressing **Finish** the process is completed.

Note that the Properties and Attributes of the Subsystem and Simulation Model are controlled by the data model of the SDM back-end. For more information on the data model and its configuration, please refer to Chapters 2 and 5 of the Data Management Reference Manual.

The respective Subsystem and Simulation Model have now been created in the DM repository.

The newly created Subsystem is displayed in the Subsystems tab after pressing the **Refresh** button. At this point, the representation file of the Subsystem that is stored in the DM contains only the hierarchy of the Subsystem.





Similarly, the Simulation Model is displayed in the *Simulation Models* tab.

In the Default view, Subsystems and Simulation Models are presented in a tree view, grouped according to their Property values. This grouping is customizable through `dm_views.xml`. For more information on the customization of views please refer to Chapter 5 of the Data Management Reference Manual.

Note that additionally to the Subsystems and Simulation Models created with the process defined above through the Parts Manager, Subsystems and Simulation Models can be created manually, either from **File > New > Subsystem** and **File > New> Simulation Model** respectively, or through the context menu options **New Subsystem** or **New Simulation Model** in Subsystem and Simulation Model tabs respectively (as described in paragraph 4.4.2.1 of this document).

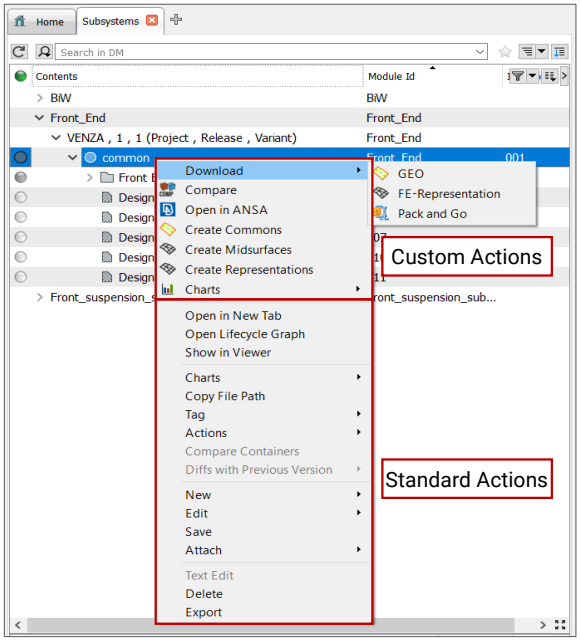
8.2. Model Preparation

KOMVOS comes pre-configured with a set of actions that facilitate the most common model building tasks, approaching the model preparation in a modular way. This functionality is accessible through the context menu of Parts, Subsystems and Simulation Models.

Model building actions can start on Subsystem level once Subsystems are created after model import as described in paragraph 8.1.

8.2.1. Actions on Subsystems

The default actions on Subsystems are shown below. Note that this functionality is only available when KOMVOS is connected to a file-based SDM back-end.



The actions are categorized in two main groups:

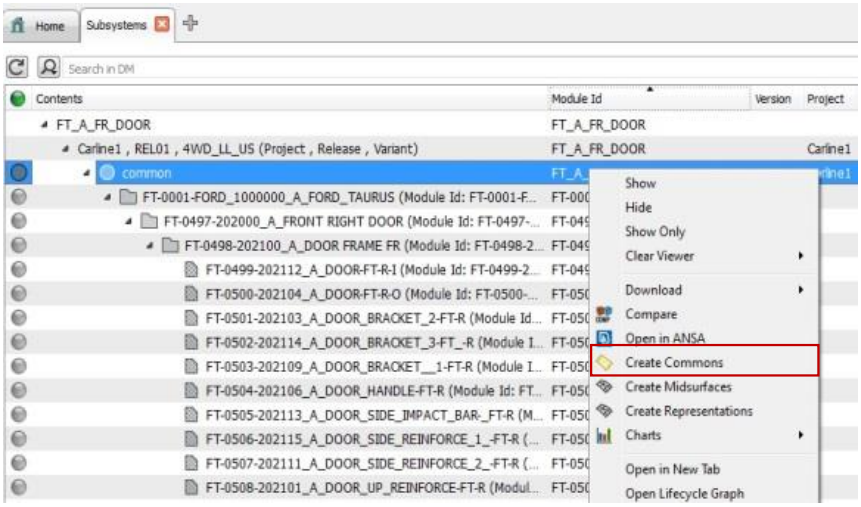
- **Custom actions**: These actions are part of KOMVOS' default configuration for model build with file-based DMs. They are made available in the default application home, the *config* directory of KOMVOS installation. This customization layer is based on Python scripts. More information on custom actions can be found in Chapter 7.
- **Standard actions**: These actions are hardcoded and cannot be modified by the users.

The process that is described in the following subchapters are based on the Custom Actions as defined in the default configuration of KOMVOS. This process starts with the creation of Common Representation, continuous with the Middle Surface extraction and closes with the generation of Mesh Representations.

8.2.1.1. Create Commons

During the **Create Commons** action, the CAD files associated to the part structure of the Subsystem are translated and stored in DM, the corresponding attributes (e.g. Module Id, Name, Version, etc.) are assigned to each part, the geometry of parts is checked and finally the parts are saved in DM with 'common' representation.

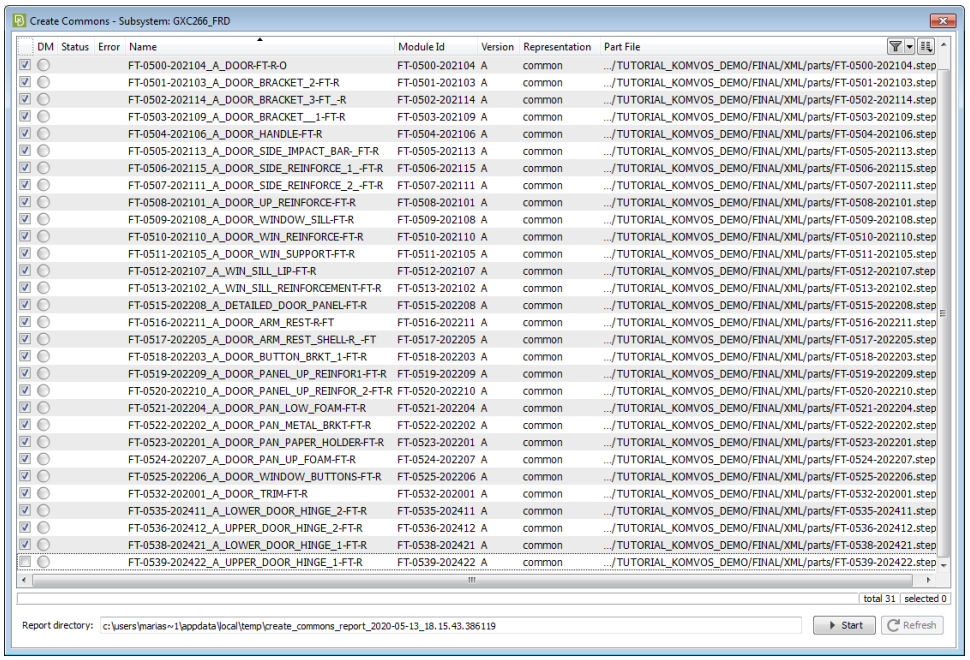
These parts will be the basis for the creation of any discipline-dependent mesh representation that will be generated in the following steps.



This action will employ a number of parallel ANSA workers to run the translation of the CAD files in the background. The number of these parallel jobs can be modified by the user as described in chapter 10.6.4.

To perform this step, right click on the Subsystem and select the **Create Commons** option from its context menu.

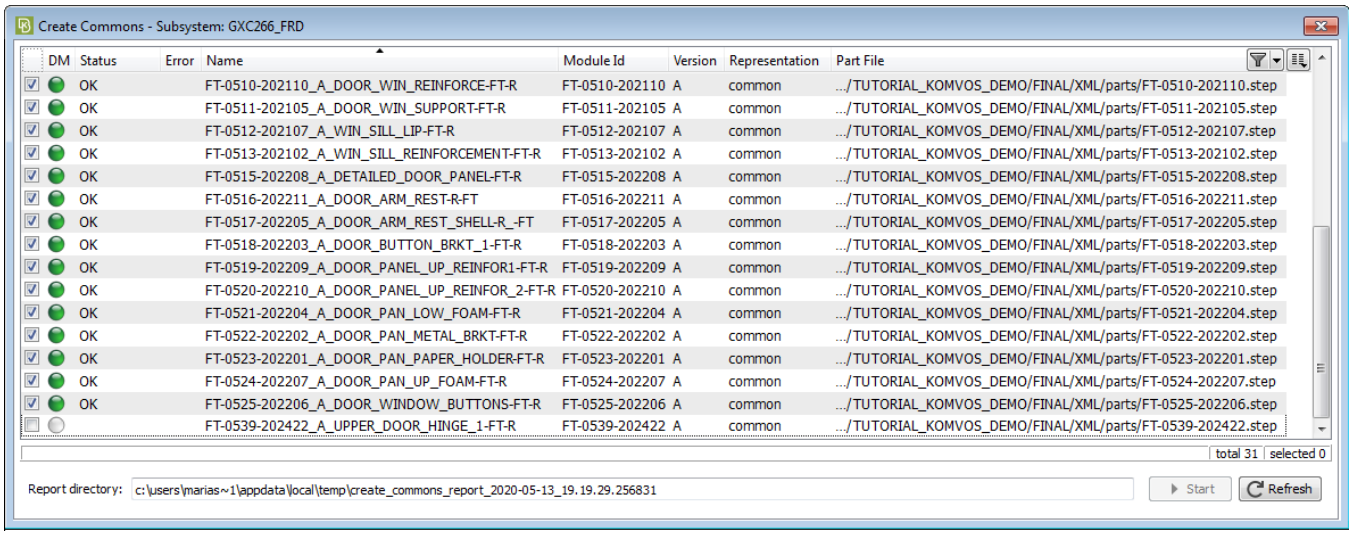
The *Create Commons* window is launched.



KOMVOS identifies and marks the parts that will be processed automatically. These are only the parts that do not have the 'common' representation in the DM repository.

Press **Start** in order to trigger the Create Commons action.

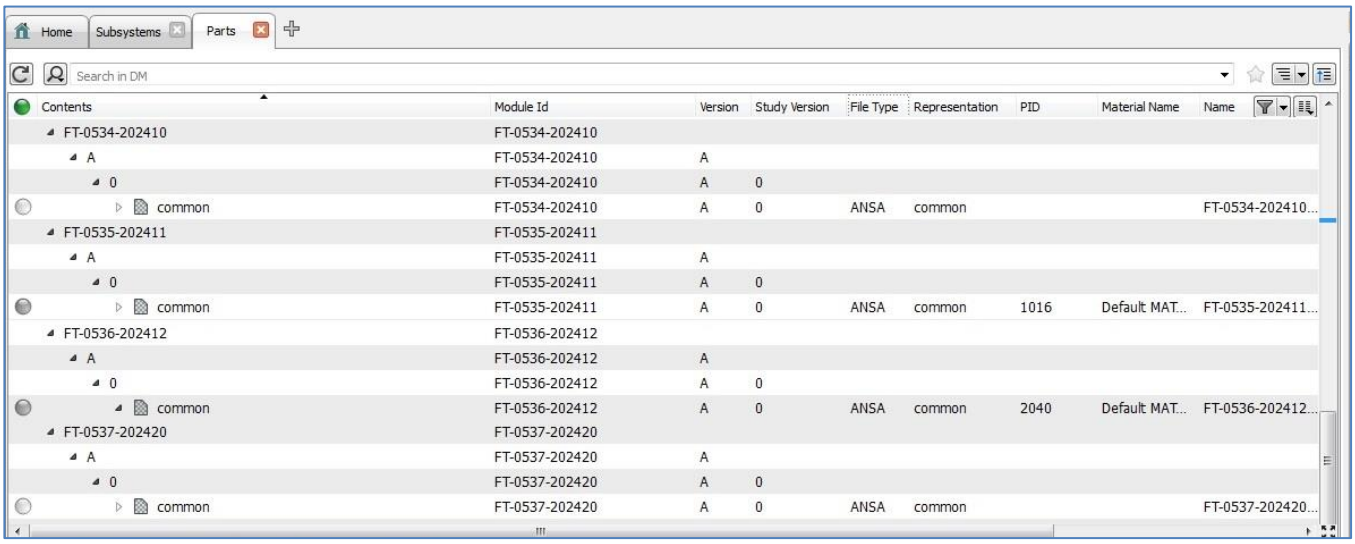
As the process progresses and 'common' representation of parts are created, the **Refresh** button can be used to update the parts status.



Through the command prompt window (Windows OS) or the terminal window (Linux), the user is informed about the state of the process. Once all actions have ended, information about the processed parts is printed and the Refresh button is no longer available. Afterwards, proceed with closing the Create Commons action window and select **OK** at the emerging confirmation window.

Pressing the **Refresh** button in the Subsystems tab will update its contents. The status of the Groups and Parts is also updated to WIP.

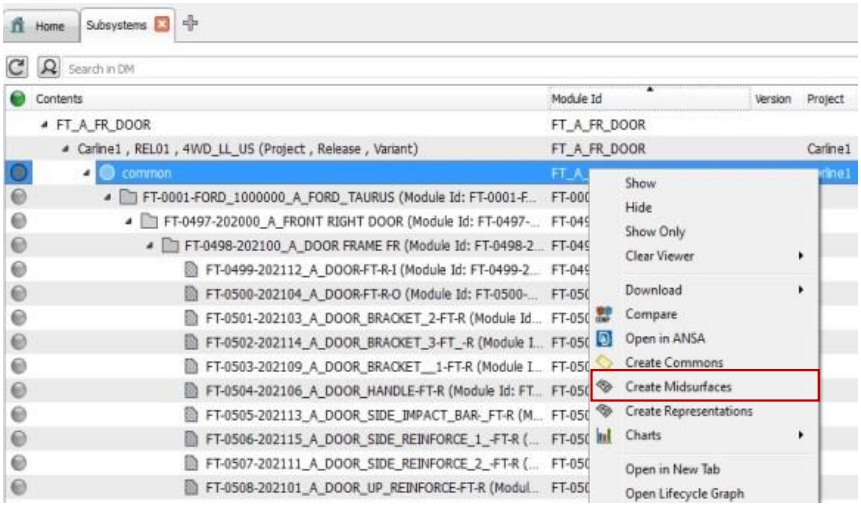
In the *Parts* tab the 'common' representation of the Parts is displayed.



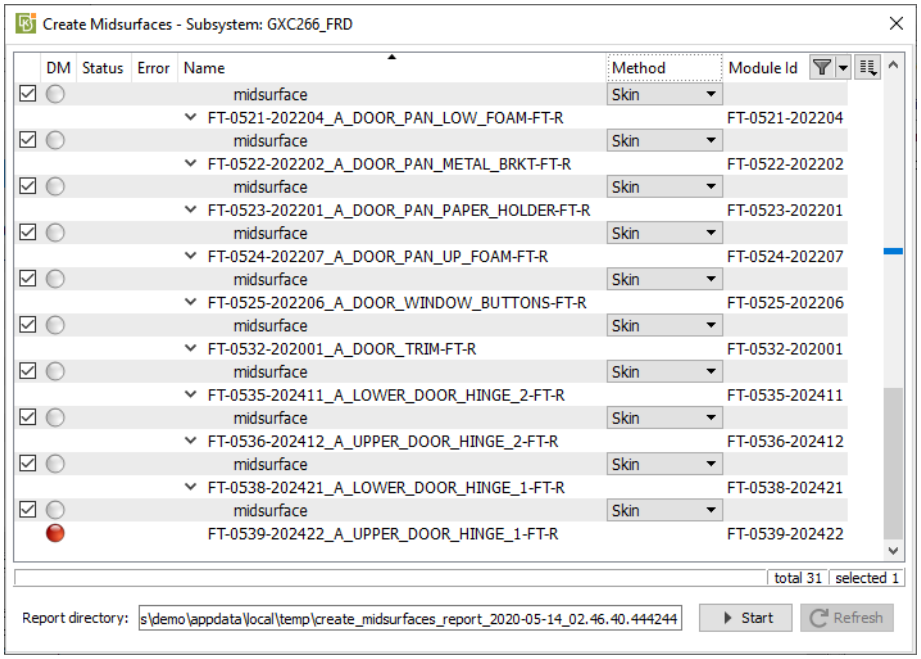
8.2.1.2. Create Midsurfaces

The **Create Midsurfaces** action is used to create the middle surface of the parts of the subsystem, so that parts with solid description get their thin shell description. The Skin or Casting function is performed on the parts that will finally be saved in the DM repository under the 'midsurface' representation.

This action will employ a number of parallel ANSA workers to extract the middle surface of the parts in the background. The number of these parallel jobs can be modified by the user as described in chapter 10.6.4.



To perform this step, right click on the Subsystem and select option from its context menu. The respective action window is launched.



KOMVOS automatically identifies and marks the parts where the action will be applied.

Only the parts with 'common' representation that do not have a 'midsurface' representation in the DM repository will participate in this action.

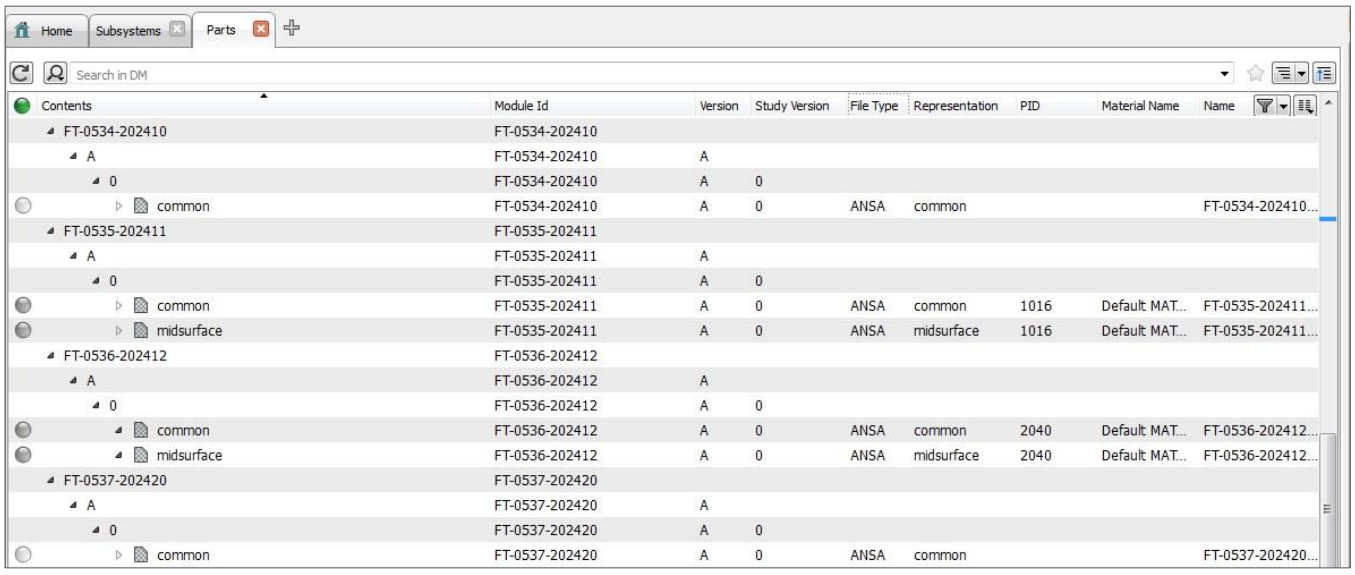
Two middle surface extraction methods are available in this Action: Skin and Casting. Further information regarding these methods can be found in chapters 8.24.1 and 8.24.2 of ANSA User Guide.

Pressing **Start** the process starts.

As the process progresses and 'midsurface' representation of parts are created, the **Refresh** button can be used to update the parts status.

Once the action is finished, the *Create Midsurfaces* action window remains visible, giving useful information about the status of the parts.

Close the *Create Midsurfaces* action window and select **OK**.



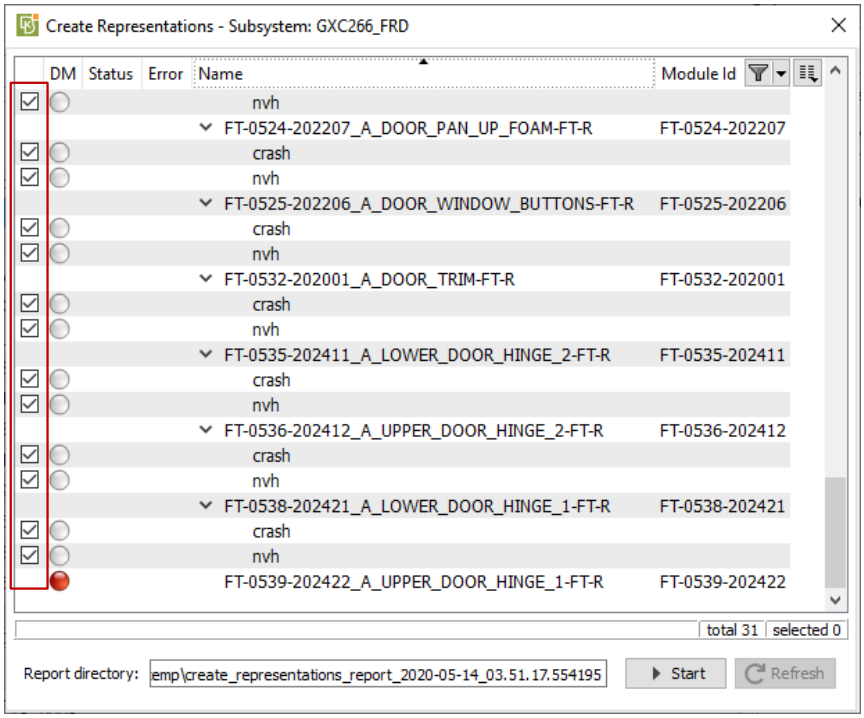
8.2.1.3. Create Mesh Representations

The **Create Representations** action is performed on the Subsystem in order to create mesh representations on the selected parts of the Subsystem, according to the target mesh values of the available mesh scenarios and save them back to the DM. This action will employ a number of parallel ANSA workers to create the mesh of the parts in the background. The number of these parallel jobs can be modified by the user as described in chapter 10.6.4.

The available meshing scenarios are those located in the 'batch_mesh_sessions' folder of the DM.

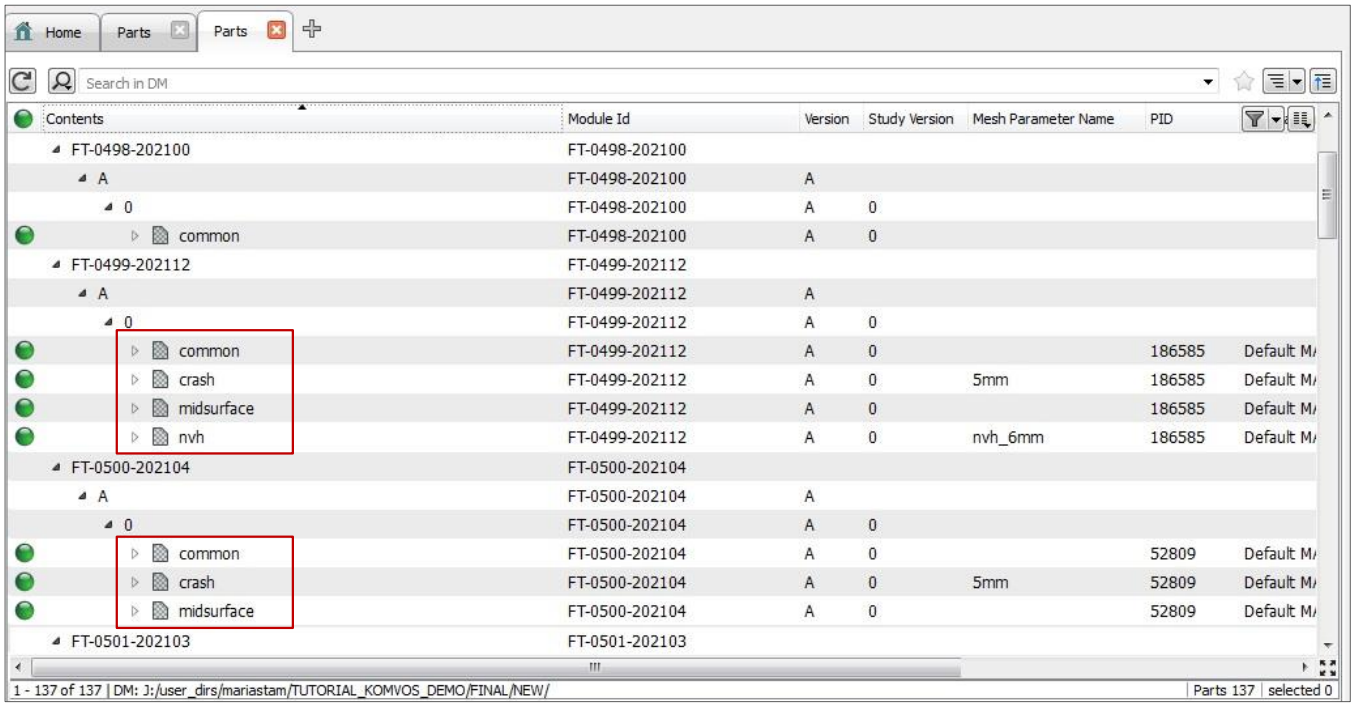
From the *Create Representations* action window, users can select the desired meshing scenarios to be executed, by marking/ unmarking the respective mesh representation. In this case, the middle surface representation was used to run the meshing scenarios.

Through the command prompt window (Windows OS) or the terminal window (Linux), the user is informed about the state of the process. Once all actions have ended, information about the processed parts is printed and the Refresh button is no longer available.



Once the action is finished, press **Refresh** and close the *Create Mesh Representations* action window.

By pressing **Refresh** in the Parts tab, all representations of the parts that have been saved in the DM repository (included the recently saved mesh representations) are displayed.


Home
Parts
Parts
Search in DM
Contents
Module Id
Version
Study Version
Mesh Parameter Name
PID
FT-0498-202100
FT-0498-202100
A
FT-0498-202100
A
0
FT-0498-202100
A
0
common
FT-0498-202100
A
0
FT-0499-202112
FT-0499-202112
A
FT-0499-202112
A
0
FT-0499-202112
A
0
common
FT-0499-202112
A
0
186585
Default M
crash
FT-0499-202112
A
0
5mm
186585
Default M
midsurface
FT-0499-202112
A
0
186585
Default M
nvh
FT-0499-202112
A
0
nvh_6mm
186585
Default M
FT-0500-202104
FT-0500-202104
A
FT-0500-202104
A
0
FT-0500-202104
A
0
common
FT-0500-202104
A
0
52809
Default M
crash
FT-0500-202104
A
0
5mm
52809
Default M
midsurface
FT-0500-202104
A
0
52809
Default M
FT-0501-202103
FT-0501-202103
1 - 137 of 137 | DM: J:/user_dirs/mariastam/TUTORIAL_KOMVOS_DEMO/FINAL/NEW/
Parts 137 | selected 0

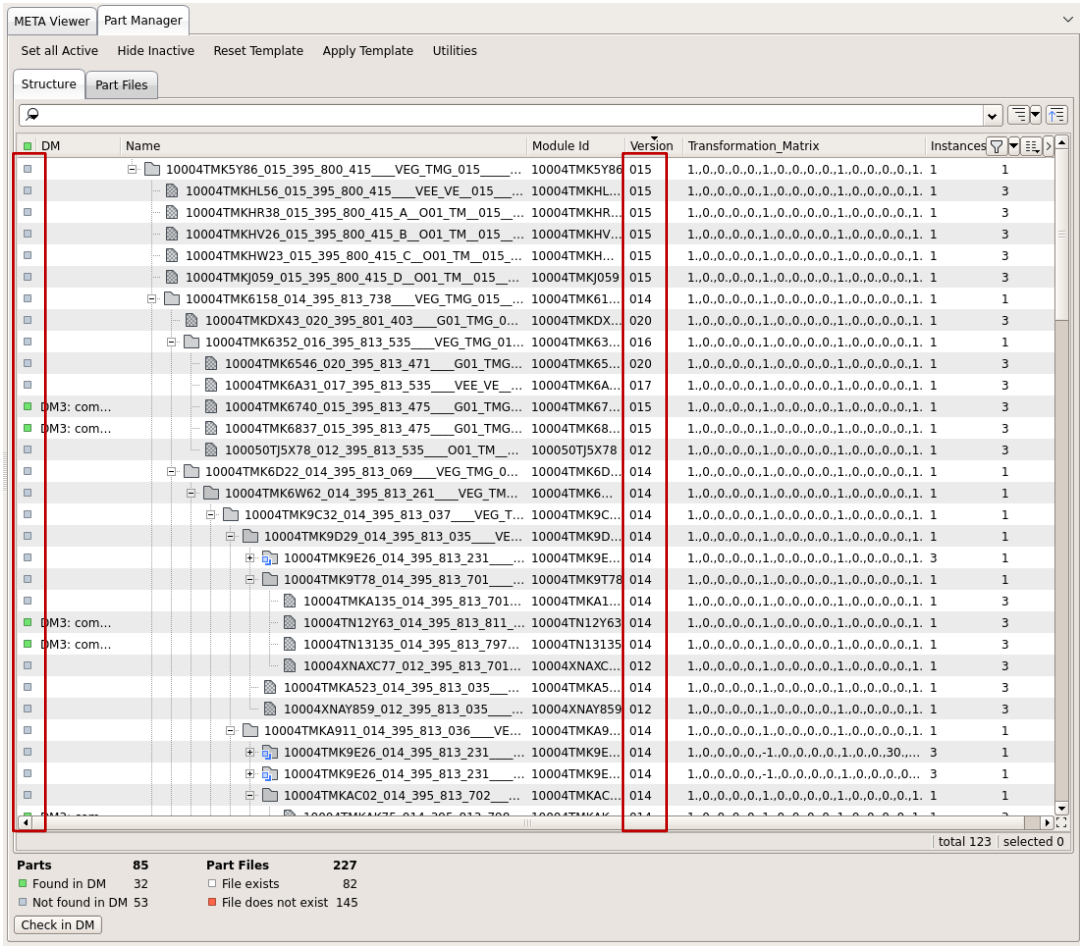
8.3. Model Update

When new CAD versions are released from the CAD department, the new parts and the new part versions should be added in the DM repository in order to be included in the CAE models for further analysis.

For this reason, the new product tree that includes the updates and any additional parts must be imported in KOMVOS.

8.3.1. Import the updated model definition

The new assembly hierarchy version can be imported and read in KOMVOS by following the procedure described in paragraph 8.1.1.



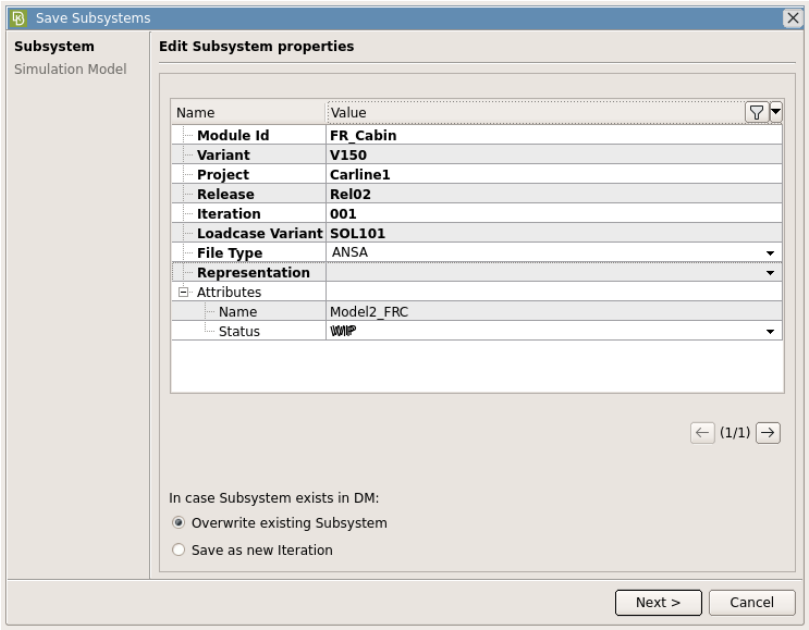
The *Part Manager* window is launched, displaying the updated assembly hierarchy information.

By pressing the **Check in DM** button, the DM column is automatically updated and the available representations of the parts already in DM are shown.

The parts that already exist in the DM repository are marked with green color. These parts exist in the repository with exactly the same signature (Module Id, Version, Part Name) with the one in the initial assembly hierarchy tree and do not need to be processed again.

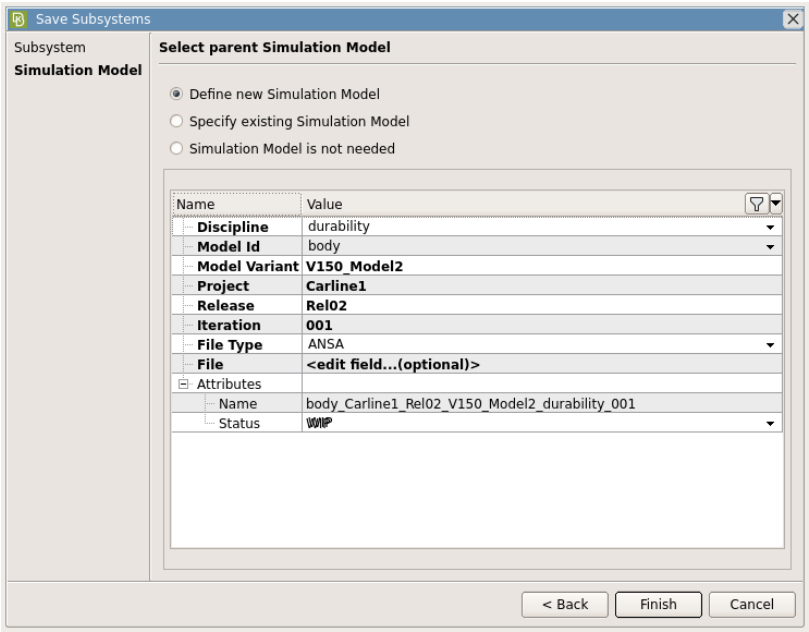
Some parts though are identified with a new version.

8.3.2. Create and save the updated Subsystem



At this point, the user should edit the displayed Properties and Attributes as desired.

Press **Next** in order to proceed.



In the next page, it is possible to associate the Subsystem to be created with a Simulation Model. Leave the default selected option **Define new Simulation Model**.

Press **Finish** to complete the process.

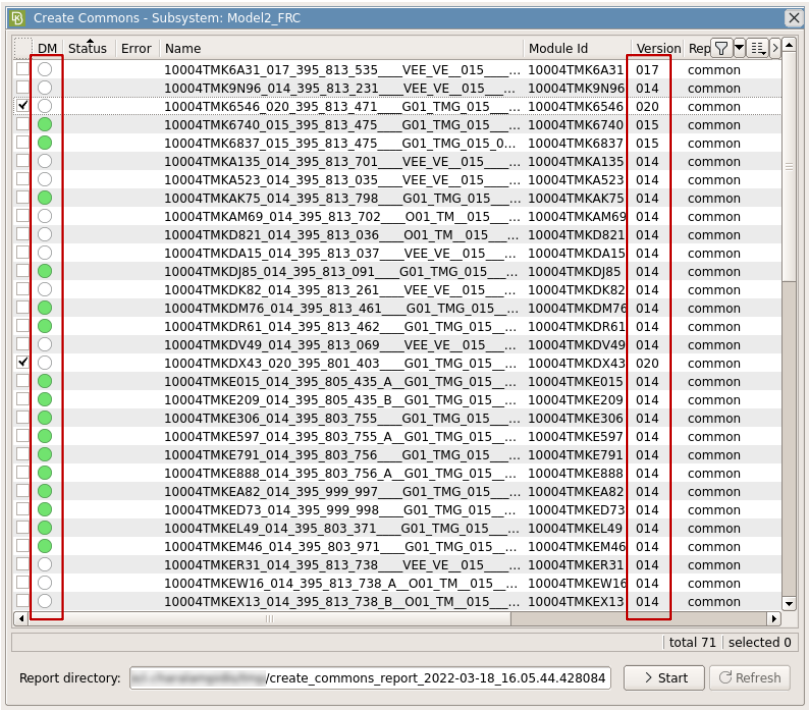
The respective Subsystem and Simulation Model have now been created in the DM repository.

The newly created Subsystem and Simulation Model are displayed in the respective tabs after pressing the **Refresh** button.

At this point, users can compare the two Subsystems in order to identify their differences, as described in paragraph 8.4.1.1.

8.3.3. Perform all actions on the updated data

Once the Subsystem with the updated part hierarchy has been created, the actions to create and save inside the DM folder the “common”, “midsurface”, “crash” and “nvh” representation of the parts can be executed.



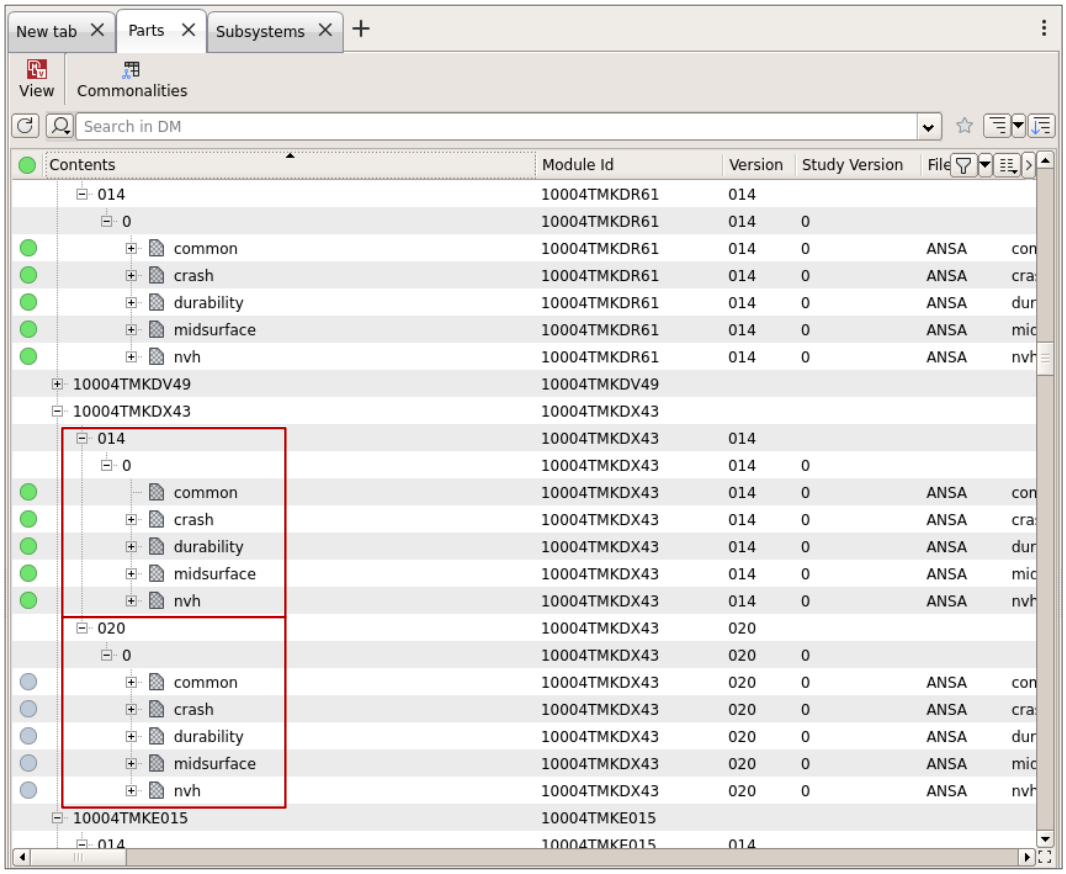
! Notice that before starting each action, KOMVOS fetches directly from the DM repository the parts that already exist under the specific representation.

So, the actions do not need to run on these parts again, thus saving considerable time.

For instance, in the case of **Create Commons** action, only the parts with new versions will participate in the translation and the generation of the ‘common’ representation.

Similarly, all actions will be performed only on the new/ updated parts.

After the actions on the updated Subsystem are completed, by pressing the **Refresh** button at the Parts tab, the representations of all parts and their versions that have been saved in the DM repository can be viewed.



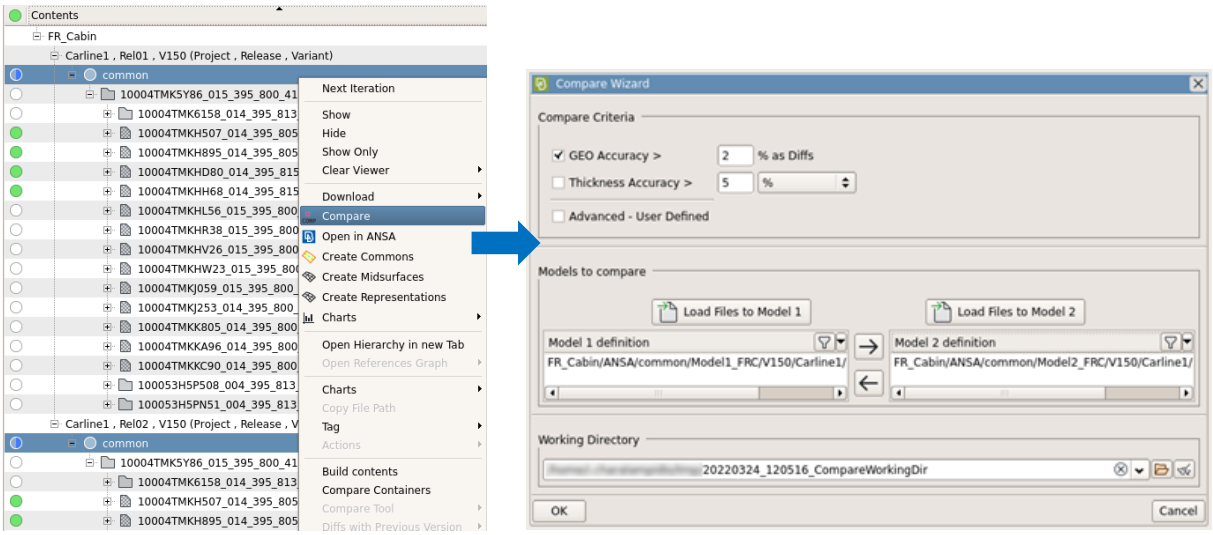
8.4. Tools for Model Review

KOMVOS offers several tools for model review, on Subsystem, Part and Simulation Model level. The available functionality is described in the paragraphs below.

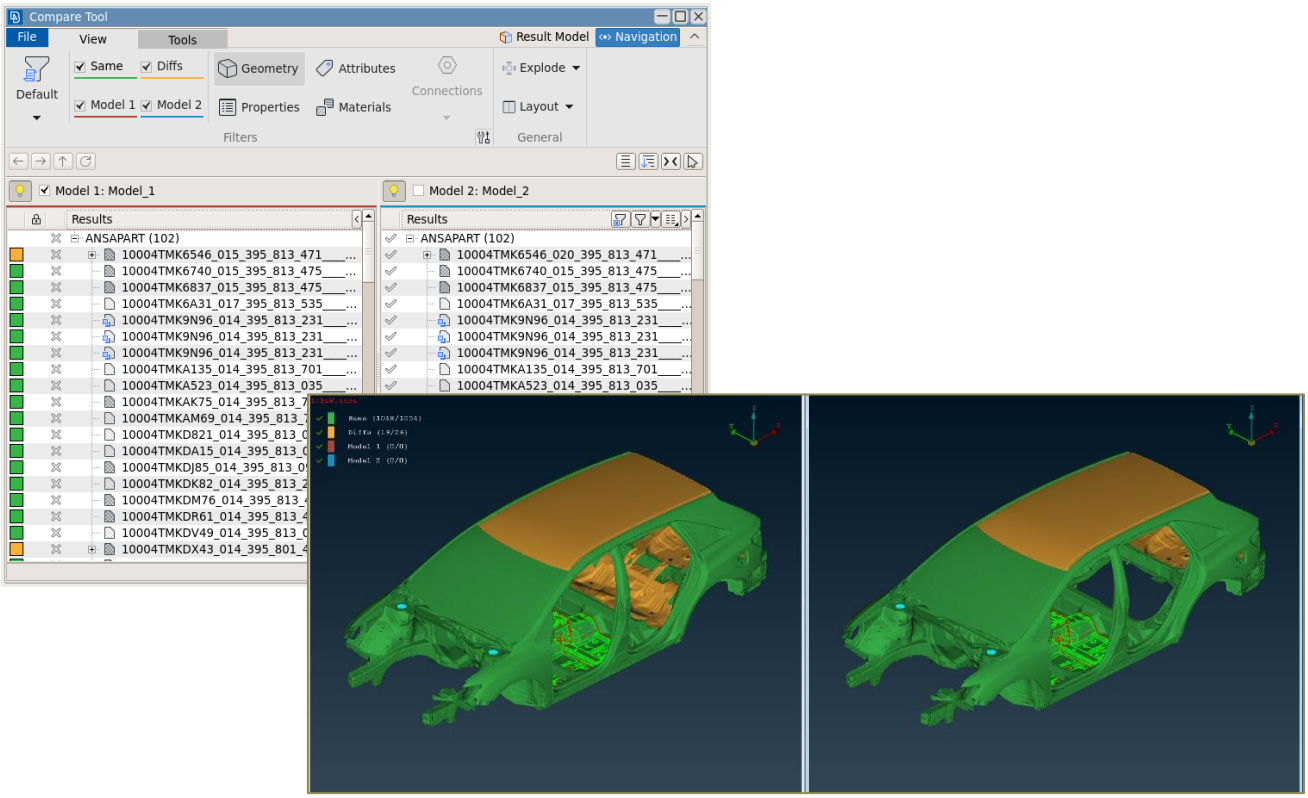
8.4.1. Model Review on Subsystem level

8.4.1.1. Compare

The Compare action allows the easy identification of differences between two selected Subsystems or between a Subsystem and an external file. Differences are identified in terms of geometry and attributes. The *Compare Wizard* window pops up where the user can define the models to be compared. On OK, ANSA is launched with the proper settings to run the comparison in the Compare Tool.



The *Compare Tool* window opens, displaying the comparison result. From this window, users can select the differences to focus on, filtering out the rest.



Note that apart from the Subsystems, the Compare action can also be applied on Parts.

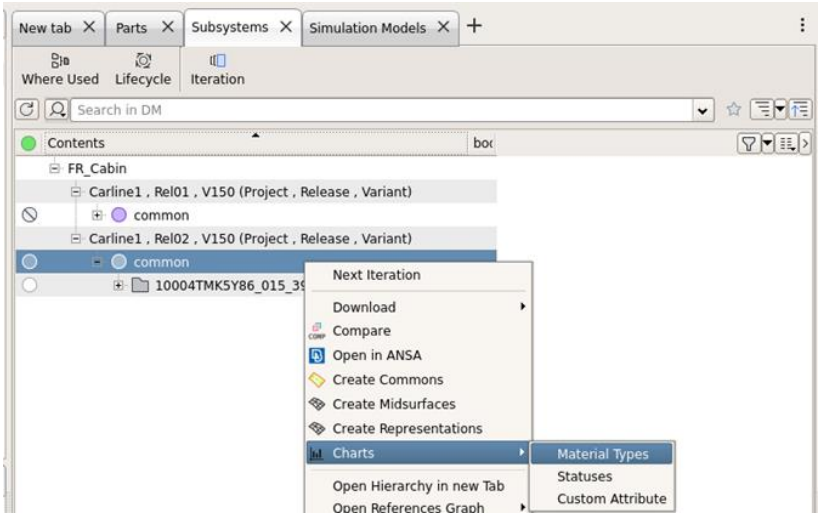
For more information on the Compare functionality, please refer to *Paragraph 19.2* of the ANSA User Guide

8.4.1.2. Open in ANSA

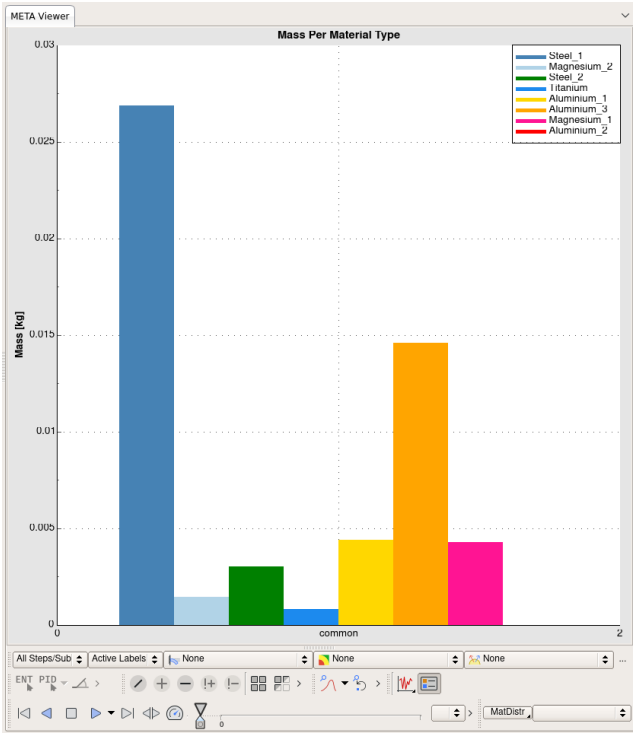
The **Open in ANSA** action allows opening the Subsystem in ANSA. All modifications that may be performed in ANSA, can be directly pushed back in the same DM repository with the Save in DM functionality of the Model Browser.

8.4.1.3. Charts

With the aid of the **Charts** action, an overview of certain characteristics of DM objects can be visualized in charts in the embedded *META Viewer*. The Charts functionality is available for *Subsystems*, *Parts* and *Simulation Models*, only if the *Parts* have already been saved as standalone DM objects.



In the Subsystems tab, select one of the Subsystems and click on the Charts action from its context menu. A sub-menu with the supported chart types pops up.

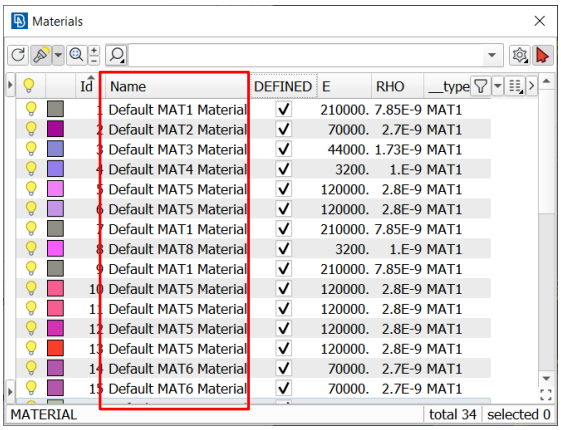


Material Types chart shows the mass per material type for the parts of the selected Subsystem.

This function relies on the configuration files that are described below, and they reside in the *config* directory. Each of these files can be customized to visualize the desired data.

`All_materials_types.xlsx` : through this Excel configuration file, the user can modify the list of the displayed material types. By default, the material list contained in this configuration file includes a column with the Material Names and a column with the Material Types.

The first column refers to the names of the materials as found in the Materials List of the ANSA database of each part. By modifying the Material Type column, the user can customize the material names that appear in the materials chart index. So, the **Charts > Material Types** function will look for the parts whose Material Name exists in the configuration file and will match them with the corresponding Material Type.



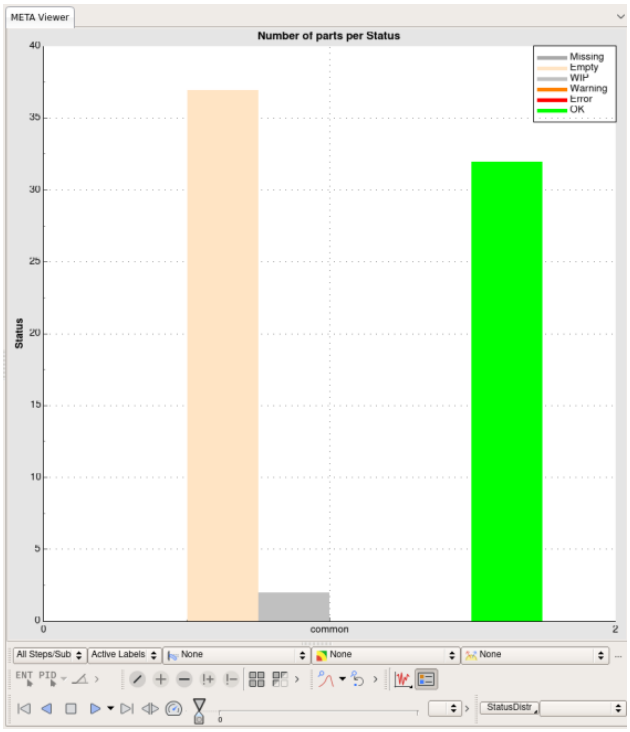
Material Names	Material Types
Default MAT1 Material	Steel
Default MAT2 Material	Aluminium
Default MAT3 Material	Magnesium
Default MAT4 Material	Plastic
Default MAT5 Material	Composite
Default MAT6 Material	Rubber
Default MAT7 Material	Glass
Default MAT8 Material	Silicon
Default MAT9 Material	Wood

`All_materials_types.xlsx`

`Palletes.xml` : Through this XML configuration file, the user can control the colours of the several parts' Statuses and Materials visualization. For example, in the palletes.xml part below, the RGB values for each Material Type are defined.

```
<colorMode name="Material" extraInfo="visibleEntities">
    <palette default="1" name="Material">
        <color name="Steel"                r="70"  g="130" b="180"/>
        <color name="Aluminium"              r="0"   g="128" b="0"/>
        <color name="Magnesium"              r="255" g="215" b="0"/>
        <color name="Plastic"              r="255" g="0"   b="0"/>
        <color name="Composite"              r="255" g="165" b="0"/>
        <color name="Rubber"               r="255" g="20"  b="147"/>
        <color name="Glass"                r="177" g="211" b="231"/>
        <color name="Silicon"                r="208" g="238" b="238"/>
        <color name="N/A Material Name" r="100" g="100" b="100"/>
        <color name="N/A Material Type" r="169" g="169" b="169"/>
        <color name="Multi-Material"     r="255" g="0"   b="255"/>
    </palette>
```

`Materials_management.py` : This script seeks for all_materials_types.xlsx file where it will find the Material Types available. Then it looks for all child parts of the selected items and it queries for those parts' mass and material names. It corresponds the material name of each part to a material type and finally, it sums up the weight of all material types found in each selected item and sends the result to META viewer. This algorithm depends on the assumptions each company uses for the material names it assigns to its data and to configure it, the user should customize this script as they see fit.

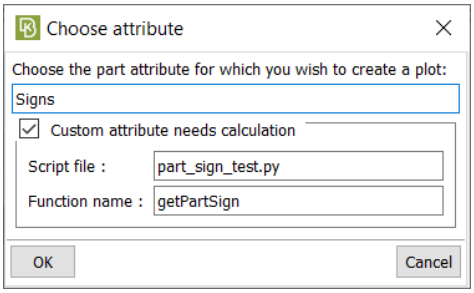


Statuses chart shows the number of Parts per status for the parts of the selected Subsystem.

Similar to Material Types Chart, the Statuses Chart can be customized according to the user's needs by modifying the Palletes.xml file and the Status_management.py script.

- `Status_management.py` : This script will search for the Status attribute of each child part and controls the colours per Status state. Likewise, the colour RGB values can be customized.

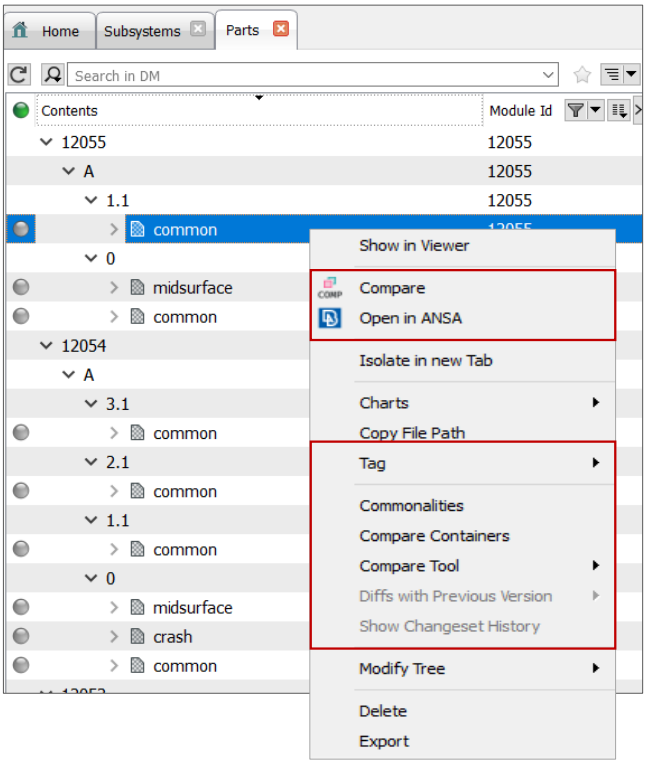
Custom Attributes



The user can produce their charts by using the *Custom Attributes* option and to do so, the necessary part attribute, script file and function name need to be defined. At the following example, the attribute "Signs" has been selected alongside the script file and the corresponding function name. This script resides in the config folder and controls the chart configuration. The user can proceed to customizing the example script in the way that fits their needs.

8.4.2. Model Review on Part level

Parts that are stored in the DM repository of KOMVOS come with a pre-defined set of built-in Actions, which appear by activating their context menu:



Similar to Subsystems, the actions are categorized in two main groups:

- **Custom actions**: These actions are part of KOMVOS' default configuration for model build with file-based DMs. They are made available in the default application home, the *config* directory of KOMVOS installation. This customization layer is based on Python scripts. More information on custom actions can be found in Chapter 7.
- **Standard actions**: These actions are hardcoded and cannot be modified by the users.

The functions used for model review are *Compare*, *Open in ANSA* and *Commonalities*.

8.4.2.1. Compare

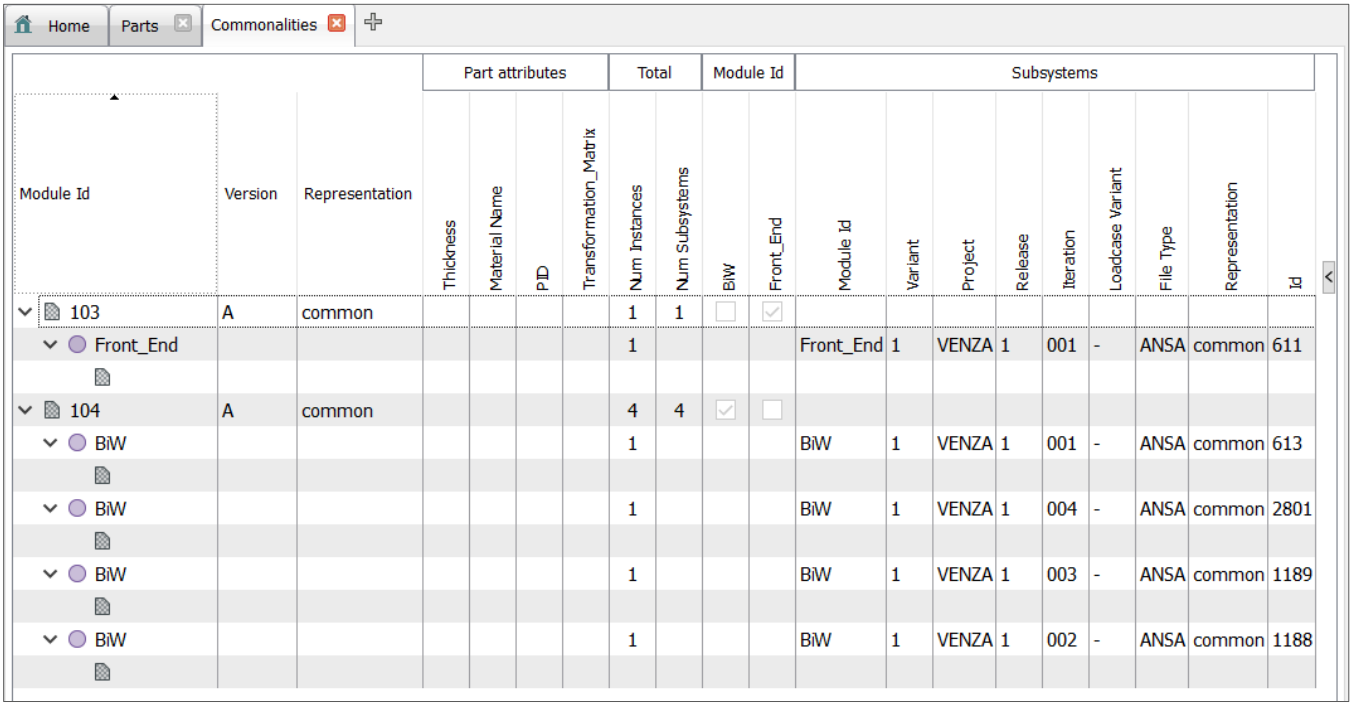
Compare on Part level is very similar to that of Subsystems. Please refer to paragraph 8.4.1.1

8.4.2.2. Open in ANSA

The **Open in ANSA** action allows opening the *Part* in ANSA. All modifications that may be performed in ANSA, can be directly pushed back in the same DM repository with the Save in DM functionality of the *Model Browser*.

8.4.2.3. Commonalities

The **Commonalities** action is performed on the *Part* in order to give information regarding in how many Subsystems the selected part participates, based on its Module id, Version and representation.



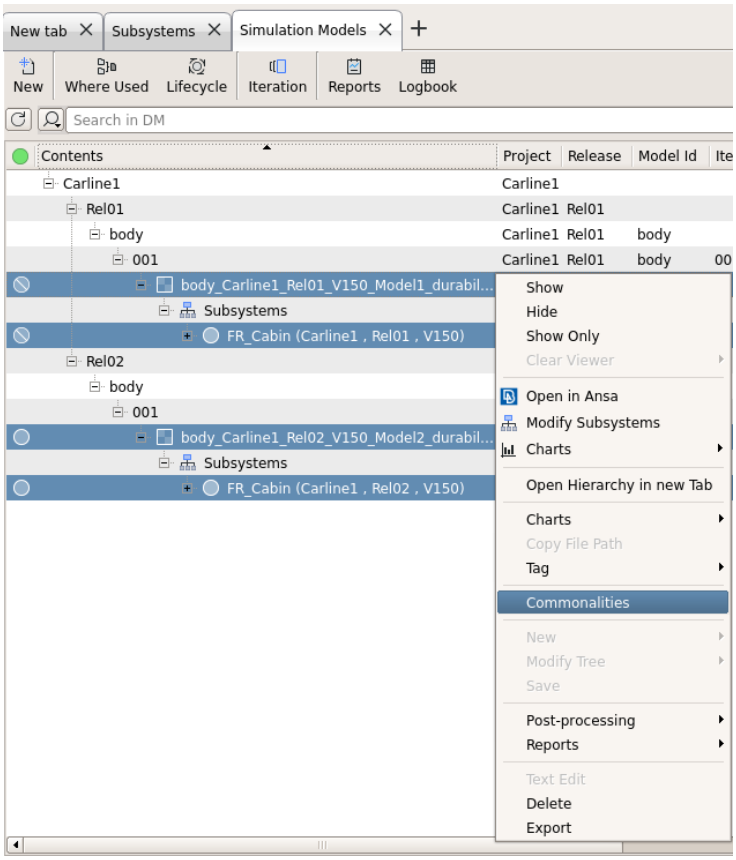
Module Id	Version	Representation	Part attributes: Thickness	Part attributes: Material Name	Part attributes: PID	Part attributes: Transformation_Matrix	Total: Num Instances	Total: Num Subsystems	Module Id: BiW	Module Id: Front_End	Subsystems: Module Id	Subsystems: Variant	Subsystems: Project	Subsystems: Release	Subsystems: Iteration	Subsystems: Loadcase Variant	Subsystems: File Type	Subsystems: Representation	Subsystems: Id
103	A	common					1	1	☐	☑									
Front_End							1				Front_End	1	VENZA	1	001	-	ANSA	common	611
104	A	common					4	4	☑	☐									
BiW							1				BiW	1	VENZA	1	001	-	ANSA	common	613
BiW							1				BiW	1	VENZA	1	004	-	ANSA	common	2801
BiW							1				BiW	1	VENZA	1	003	-	ANSA	common	1189
BiW							1				BiW	1	VENZA	1	002	-	ANSA	common	1188

In the first three columns the Properties of the parts are displayed, in the next three columns (i.e. Part attributes, Total and Module Id columns) the parts' attributes are shown, while in the Subsystems columns the properties of the referenced Subsystems where the belongs to appear. More information on the Commonalities functionality is given in paragraph 4.2.

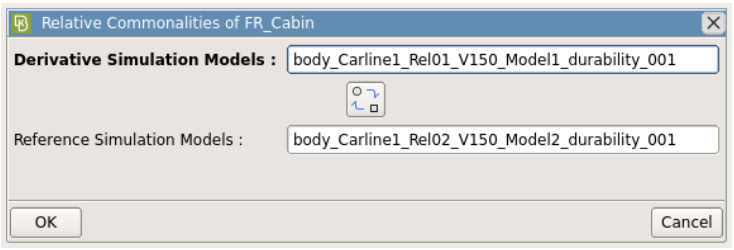
8.4.3. Model Review on Simulation Model level

8.4.3.1. Relative Commonalities

KOMVOS enables the identification of the common, shared parts among different versions or variants of the same model through the **Commonalities** function. This functionality, when executed on Simulation Model level, enables the identification of the parts that participate in two models, known also as 'carry-over' parts, so that the user can assess the effort required in order to build the new variant / version given that a base Simulation Model is already available.

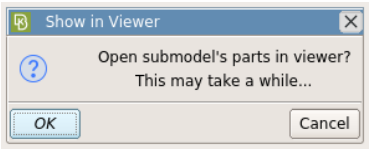


To trigger the action, in the *Simulation Models* tab select the Subsystems under each Simulation Model and the action **Commonalities** from the context menu.



The *Relative Commonalities* window appears in order to define which of the two Simulation Models is the reference and which the derivative.

The user can flip the models by pressing the **toggle** button.



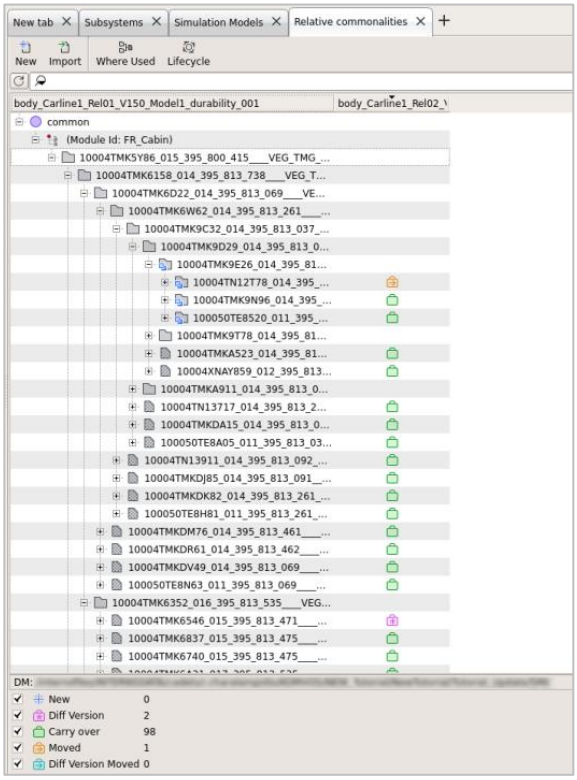
In this case, the models are correctly set, so press the **OK** button to proceed.

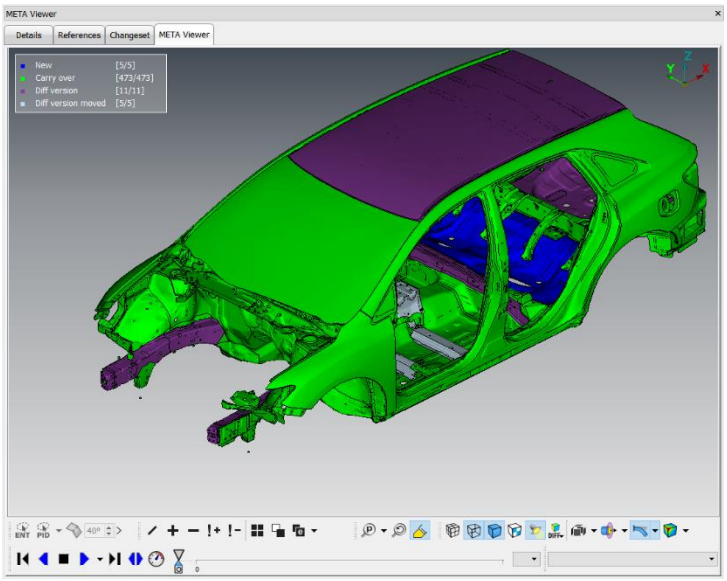
Press also **OK** to the confirmation window that appears.

By expanding the Subsystem and the columns of the Simulation Models, the results of the action are displayed and all parts are tagged with one of the following markings:

- **New**: the parts that are imported for the first time in the model
- **Diff. Version**: the parts that have only updated their version
- **Carry over**: the common parts of the Simulation Models
- **Moved**: the parts that have only a different position
- **Diff. Version Moved**: the parts that have a different version and a different position in the two models.

The results are also displayed in META Viewer, where users can select to isolate and display a specific type of parts by using the 'Show only' option on the legend.

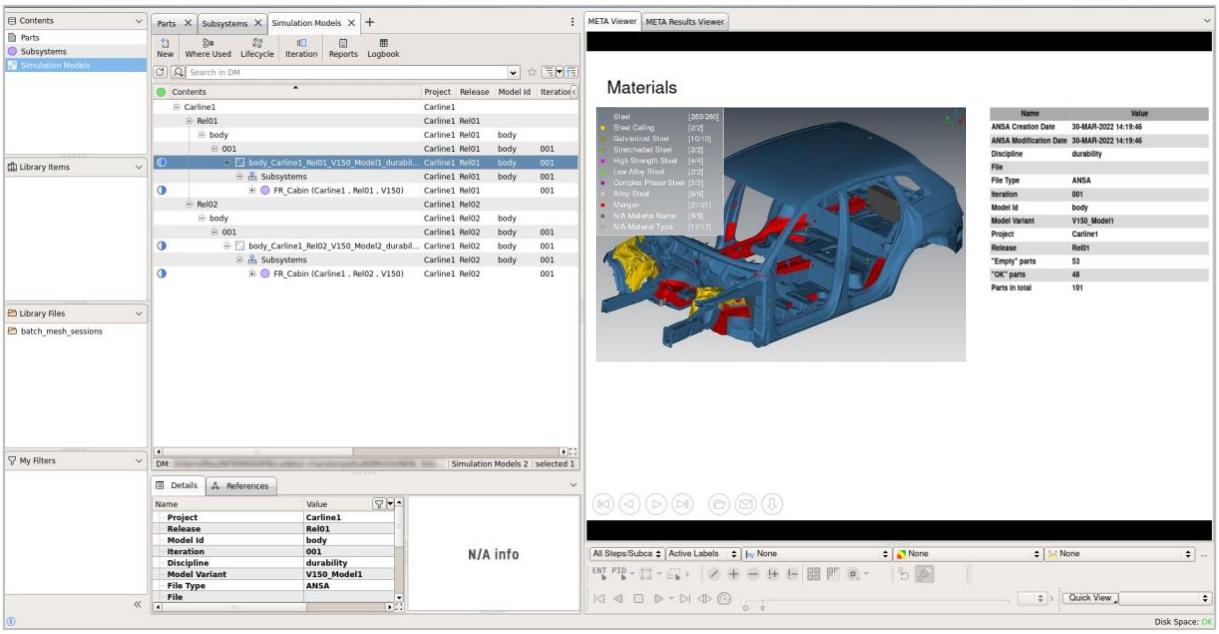




8.4.3.2. Simulation Model Reports

For the generation of model reports during model build, KOMVOS offers advanced reporting functionality that is powered by the report composition capabilities of META post-processor. Using the available functionality, Quick Reports can be generated on the fly and provide necessary information regarding the model. These reports, capturing the current status of the model, can be easily created in just one step, can be saved for reference and can also be shared with any interest party.

To generate the report, in the Simulation Models tab select **Quick View > Show** from the context menu of the Simulation Model and **OK** at the confirmation window, since the report will be created for the first time. The report is created and opened in the embedded META Viewer. It is also attached under the Simulation Model so that next time, it's ready for review.



8.5. Templates

KOMVOS supports the use of templates for the creation of Subsystems and Simulation Models. Subsystem Templates are used on a Product Structure, in order to deactivate branches of the tree structure that should not be part of the CAE Subsystem. Simulation Model Templates are used when a Simulation Model is created “from scratch”, in order to populate it with the right collection of Subsystems, following a given set of inclusion rules.

Templates are defined in the form of XML files. When these files are placed in the `$SDM_CONSOLE_HOME`, they are accessible through standard KOMVOS GUI functionality. However, templates can also be used through KOMVOS script API.

8.5.1. Subsystem Templates

8.5.1.1. Template format

The Subsystem template describes which branches of a Product Structure should be active or inactive in order to define the part structure of the Subsystem. This description consists of two parts:

a) The filter to be applied
b) The level of the tree structure where this filter should be applied

A basic structure of a Subsystem Template file is presented below. The *TopItem* and *ChildItem* keywords, of the template, describe the levels of the tree structure where a filter should be applied. To identify a branch of the tree structure as a *TopItem* or a *ChildItem*, the *MatchKeyword* line is utilized, which filters the tree structure using a set of keywords. Then, the scope of the filter, i.e. where it will be applied, is controlled by the *Depth* keyword. Finally, the action to be applied in the identified branch, that can either be an inclusion or an exclusion of the identified branch, is controlled by the *Action* keyword.

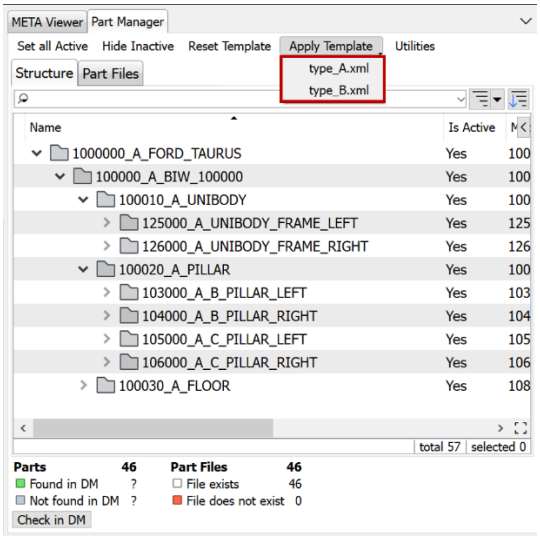
```
<?xml version="1.0" encoding="UTF-8"?>
<XMLData>
    <TopItem Id="1" Action="Set_Inactive" Depth="First_Level">
        <MatchKeyword search_in_attribute="*" after_string="*" for_keyword="*" use_delimeter="*" section_index="*" match_method="*"/>
        <ChildItem Id="2"/>
    </TopItem>
    <ChildItem Id="2" Action="Set_Active_Selected" Depth="All_Levels">
        <MatchKeyword search_in_attribute="Name" after_string="" for_keyword="RIGHT" use_delimeter="_" section_index="4" match_method="equals"/>
    </ChildItem>
</XMLData>
```

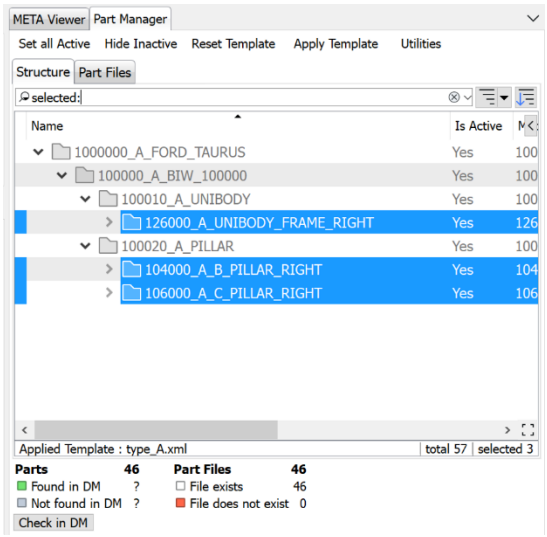
There are two filtering modes for the Subsystem templates, that differ in terms of format. **Mode A** starts with setting as inactive the whole Product Structure and then only activates the parts/groups of interest. **Mode B** starts with having everything as active and then only deactivates the parts/groups that should be excluded.

The mode used is determined by the action of the *TopItem*, which can be set to **Set_Inactive** or **Include**. In case **mode A** is utilized for the filtering of the Product Structure, the *Action* keyword should be set to **Set_Inactive**, as shown in the picture below. In case **mode B** is utilized for the filtering of the Product Structure the *Action* should be set to **Include**, as shown in the picture below.

Let's see the two modes in action through an example. In the example below, the goal is to modify the tree structure shown on the left in order to reach the result shown on the right, where the active branches are only those that contain "RIGHT" but not "FLOOR" in their name.

Initial tree structure **After Template application**





To achieve this with a template formatted according to mode A, the above template is utilized, which deactivates the first level group of the structure, in this case 1000000_A_FORD_TAURUS (TopItem), and then activates all child groups (ChildItem) which, when split by "_", the 4th token equals to the keyword "RIGHT".

Template format (Mode A)

```
<?xml version="1.0" encoding="UTF-8"?>
<XMLData>
    <TopItem Id="1" Action="Set_Inactive" Depth="First_Level">
        <MatchKeyword search_in_attribute="*" after_string="*" for_keyword="*" use_delimeter="*" section_index="*" match_method="*"/>
        <ChildItem Id="2"/>
    </TopItem>
    <ChildItem Id="2" Action="Set_Active_Selected" Depth="All_Levels">
        <MatchKeyword search_in_attribute="Name" after_string="" for_keyword="RIGHT" use_delimeter="_" section_index="4" match_method="equals"/>
    </ChildItem>
</XMLData>
```

To achieve the same result with a template formatted according to mode B, the above template is utilized, which filters the tree structure recursively in order to collect the groups of the structure that, when split by "_", the 2nd token equals the keyword "UNIBODY" (top item with id = 1). Then, for each group identified with this characteristic, filters its child groups recursively in order to deactivate the groups that, when split by "_", the 4th token equals the keyword "LEFT" (child item with id = 2). Similarly, the top item with id = 10 is filtered and its child groups identified by the corresponding filter are deactivated. All other groups of the tree structure that are not included in these two Include Top Items are removed from the resulted part structure.

Template format (Mode B)

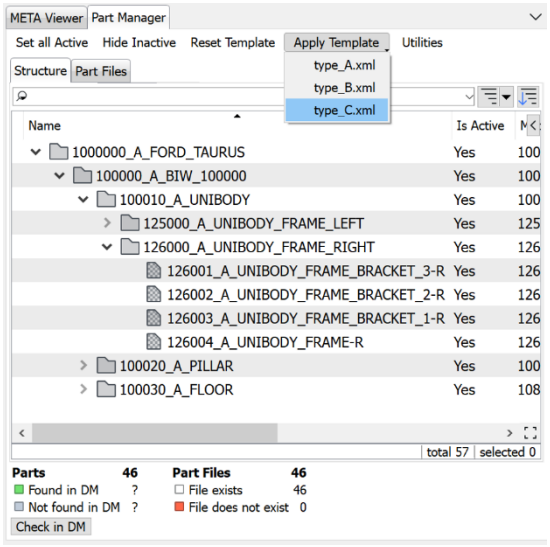
```
<?xml version="1.0" encoding="UTF-8"?>
<XMLData>
    <TopItem Id="1" Action="Include" Depth="All_Levels">
        <MatchKeyword search_in_attribute="Name" after_string="" for_keyword="UNIBODY" use_delimeter="_" section_index="2" match_method="equals"/>
        <ChildItem Id="2"/>
    </TopItem>
    <ChildItem Id="2" Action="Set_Inactive" Depth="All_Levels">
        <MatchKeyword search_in_attribute="Name" after_string="" for_keyword="LEFT" use_delimeter="_" section_index="4" match_method="equals"/>
    </ChildItem>

    <TopItem Id="10" Action="Include" Depth="All_Levels">
        <MatchKeyword search_in_attribute="Name" after_string="" for_keyword="PILLAR" use_delimeter="_" section_index="2" match_method="equals"/>
        <ChildItem Id="11"/>
    </TopItem>
    <ChildItem Id="11" Action="Set_Inactive" Depth="All_Levels">
        <MatchKeyword search_in_attribute="Name" after_string="" for_keyword="LEFT" use_delimeter="_" section_index="4" match_method="equals"/>
    </ChildItem>
</XMLData>
```

Additionally, it is possible to define a child item included in another child item, by adding a child item reference in the parent child item. This means that the Id of the child item should be defined below the match keyword filter of the parent child item, as shown in the example below.

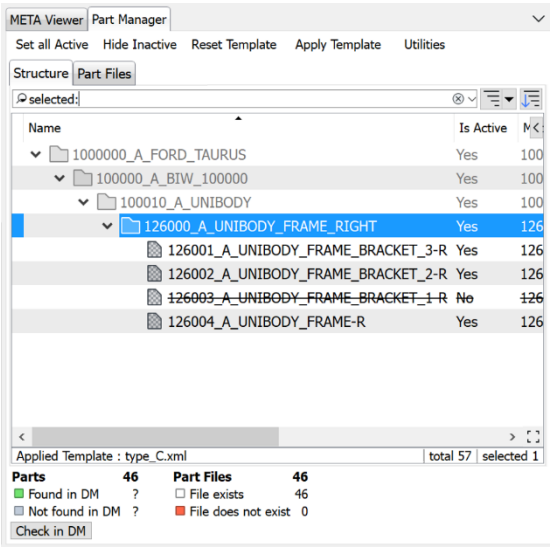
Template format example

```
<?xml version="1.0" encoding="UTF-8"?>
<XMLData>
    <TopItem Id="1" Action="Set_Inactive" Depth="First_Level">
        <MatchKeyword search_in_attribute="*" after_string="*" for_keyword="*" use_delimeter="*" section_index="*" match_method="*"/>
        <ChildItem Id="2"/>
    </TopItem>
    <ChildItem Id="2" Action="Set_Active_Selected" Depth="All_Levels">
        <MatchKeyword search_in_attribute="Name" after_string="FRAME" for_keyword="RIGHT" use_delimeter="_" section_index="1" match_method="equals"/>
        <ChildItem Id="3"/>
    </ChildItem>
        <ChildItem Id="3" Action="Set_Inactive" Depth="First_Level">
        <MatchKeyword search_in_attribute="Name" after_string="" for_keyword="1-R" use_delimeter="_" section_index="5" match_method="equals"/>
    </ChildItem>
</XMLData>
```

Initial tree structure



Tree structure after Template application



The different template settings with their possible values are described below:

For the top items, the supported **"Action"** keyword values are:

- **Set_Inactive/Set_Inactive_Selected** (mode A): sets the whole Product Structure as inactive in the Part Manager. In this case, it is recommended to set every keyword of the filter to "*", in order to automatically collect the top item of the tree structure. Also, it is mandatory to set the depth keyword value to "First_level", as shown in the picture above for mode A. If the **Set_Inactive_Selected** value is used, then the top item will also appear as selected in the Part Manager.
- **Include** (mode B): sets a group of the Product Structure as top item. This enables the user to define multiple top item definitions in a single template file, as shown in the picture above for mode B.

For the child items, the supported **"Action"** keyword values are:

- **Set_Inactive/Set_Inactive_Selected:** sets the item as inactive in the Part Manager. If the **Set_Inactive_Selected** value is used, then the child item will also appear as selected in the Part Manager.
- **Set_Active_Selected:** sets the item as active in the Part Manager.

The supported **"Depth"** keyword values are the same for both *TopItems* and *ChildItems*:

- **All_Levels**(default): the filter of the specific item will be applied to the matched item and to all its children recursively. If no depth value is set, then the "All_Levels" value will be assumed.
- **First_Level:** the filter of the specific item will be applied only on the first level contents of the item.

For the detection of items in the Product Structure the following keywords are utilized accordingly:

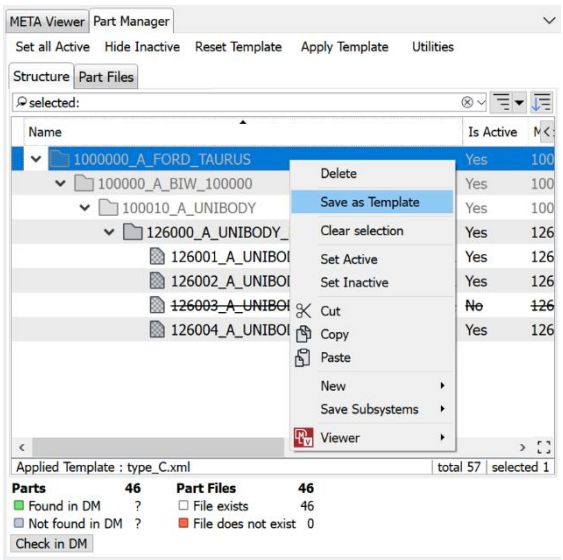
- **search_in_attribute:** collects the value of the specified attribute. The attribute can be one of “Name”, “Module Id”, “Project” etc.
- **after_string:** collects only the section of the value that follows the specified string
- **for_keyword:** The identifying key of the item among the other of the CAD Hierarchy.
- **use_delimeter:** the delimeter used along the attribute value to split the value into tokens. If the **after_string** keyword is not left as an empty string, then the delimiter used here will split the section of the attribute value, that is collected from the after_string keyword, and not the initial attribute value.
- **section_index:** After the split of the attribute value by the given delimiter, this is the index of the token to be used for filtering.
- **match_method:** the method that specifies how the identifying key specified in the for_*keyword* setting should be compared with the attribute token. If the match method is “equals”, the match should be exact. For values such as “*contains*” or “*ends_with*”, the identifying key must be contained in the calculated string.

8.5.1.2. Template creation through the Part Manager

To create a Subsystem Template through the Part Manager, the **Save as Template** context menu option is used. This option can be used on any parent item of the Product Structure in order to create a template xml file that captures the current state (active/inactive) of the selected item and its contents. Upon template application, the current state(active/inactive) of the parent item and its contents will be re-applied.

If the selected parent item has any parent items, those will be kept active. However, any other items of this Product structure, that are of the same level as the selected item, will be removed.

Before proceeding with the creation of Subsystem Templates, make sure to define a set of configuration settings in the `dm_views.xml` of KOMVOS. These settings are defined under the *General_settings* element and are utilized in order to detect items of the Product Structure. These settings are described in paragraph 5.2.1.7 of the Data Management Reference Manual.



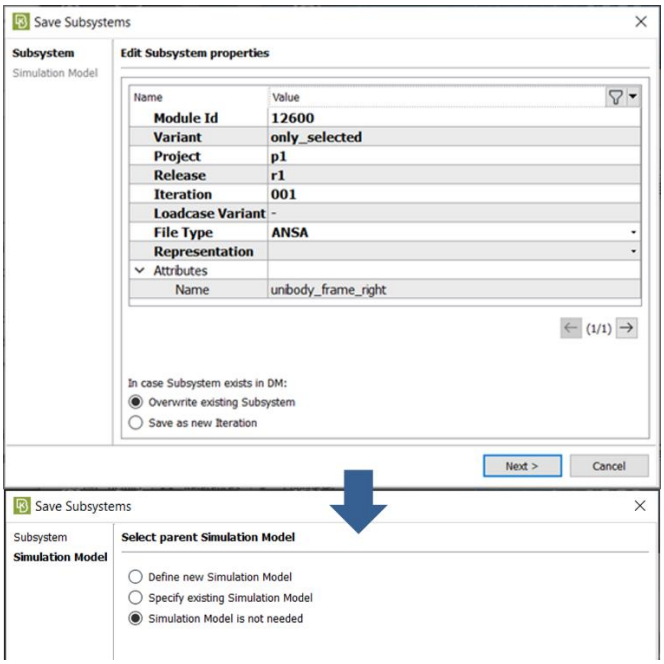
8.5.1.3. Subsystem creation through Templates

Manually

To create a Subsystem through Part Manager, the **Save Subsystems** context menu is utilized. First, a Template should be applied in the Part Manager, through the **Apply Template** toolbar option, to isolate the part structure of a specific Subsystem. Then, by selecting the **Save Subsystems** context menu option on the outermost group of the

isolated part structure, the *Save Subsystems* wizard opens. In the *Edit Subsystem* properties tab the Properties and Attributes of the Subsystem to be created are displayed.

At this point, the user should edit the displayed Properties and Attributes as desired. At the bottom, the user can define how a potential conflict should be resolved. While working with a new Subsystem select the option **Overwrite existing Subsystem** and press **Next** in order to proceed. In the next page, it is possible to associate the Subsystem to be created with a new or an existing Simulation Model.

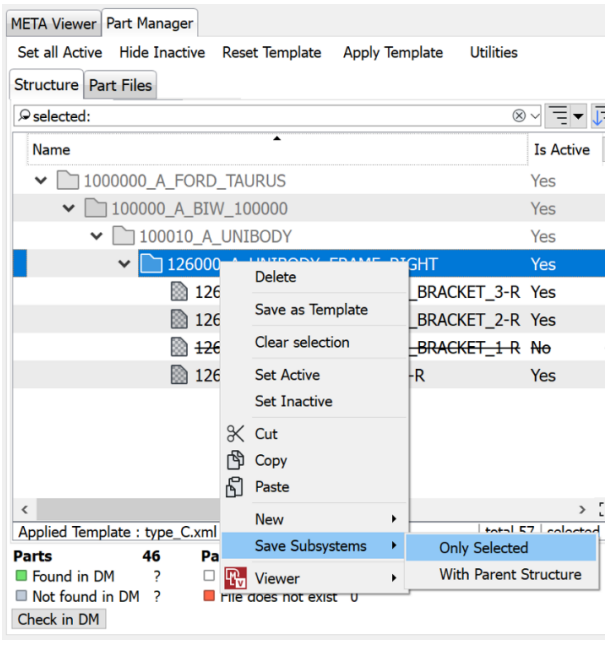


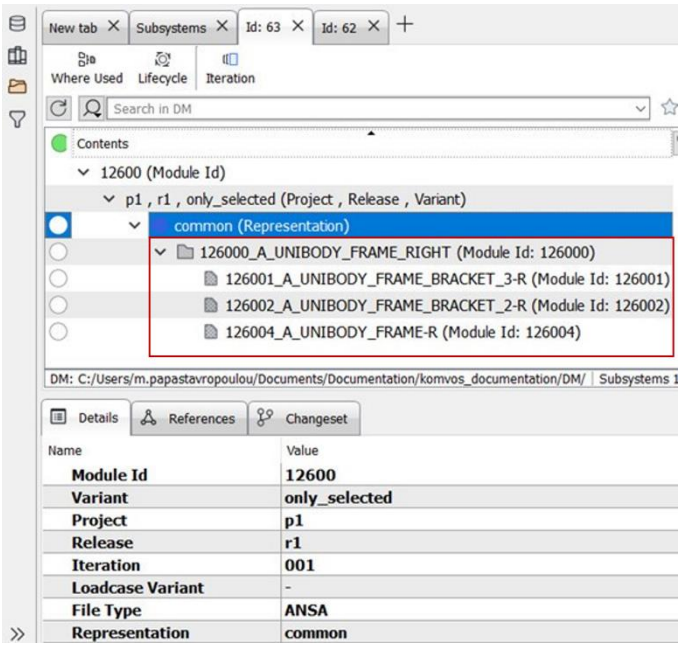
The options of the **Save Subsystem** menu are:

A. The **Only Selected**, which will save the Subsystem in DM with part structure the selected group or part.

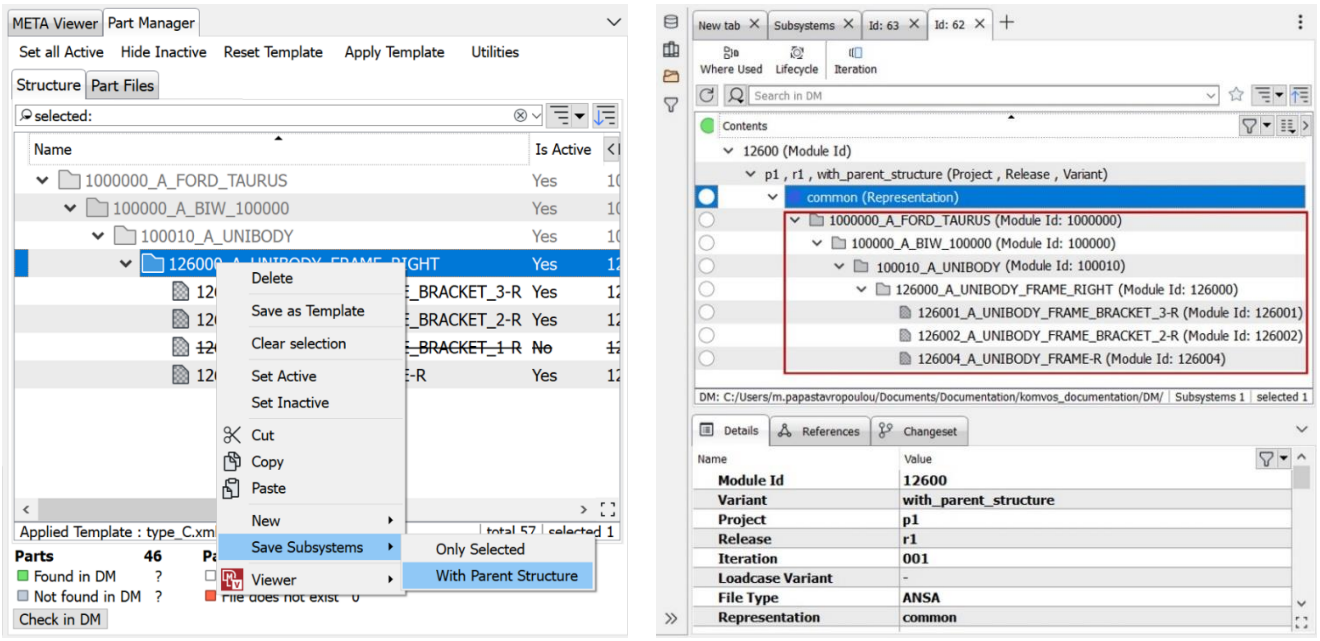
B. The **With Parent Structure:** this option will save the Subsystem in DM with part structure the selected group or part along with all its parent groups.

A. Only selected





B. With Parent Structure



With Script

It is possible to create multiple Subsystems in DM, that will have their CAD Hierarchy attached, by utilizing the `base.GenerateSubsystemsBasedOnTemplates` function. This function needs the following input arguments:

- A list of dictionaries, each one containing all the Properties of a Subsystem in a key, value pair format, having as key the property name and as value the property value.
- The file path of the model definition to be read by KOMVOS.
- A dictionary that will contain a key value pair for each Subsystem, having as key the Module Id of the Subsystem and as value the file path of the Subsystem Template. This file path can be an absolute path or a relative path to the Templates/Subsystems folder of the SDM_CONSOLE_HOME configuration folder.
- A set of settings for the import of the model definition specified with the help of the `base.InputProductTreeSettings`.

Upon completion, this function will save the Subsystems, with their CAD Hierarchy attached, in DM.

8.5.2. Simulation Model Templates

8.5.2.1. Template format

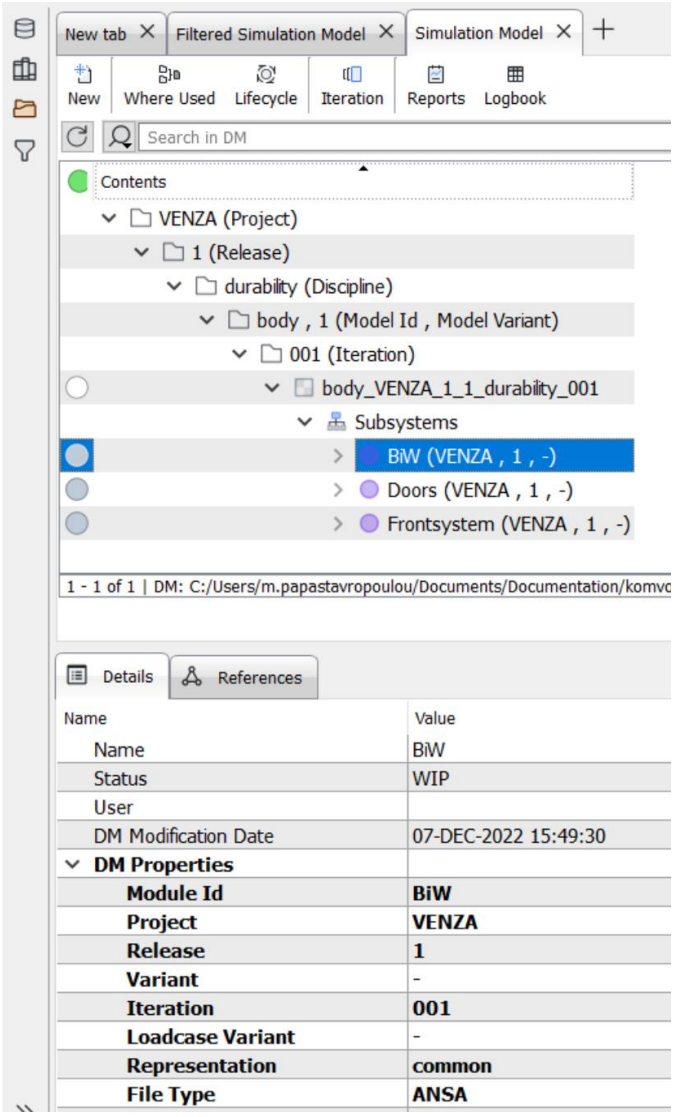
Simulation Model Templates are used when a Simulation Model is created “from scratch” in order to populate it with the right collection of Subsystems from DM, following a given set of inclusion rules. The inclusion rules are the Subsystem’s Properties.

The template file presented below will create a Simulation Model populated with three Subsystems. For each Subsystem its properties and attributes values are defined in the template file. If the Subsystem exists in DM, it will be added as a content of the Simulation Model. Otherwise, a new Subsystem will be created with these characteristics, as an empty placeholder.

In order to enable the definition of generalized templates, it is possible for Subsystem properties to be defined parametrically, based on Simulation Model properties. For example, if the Subsystems that must be collected under

a Simulation Model must have a Project value identical to that of the Simulation Model, then the "$" symbol followed by the Simulation Model property name ("$Project") can be used as value of the Subsystem property in the template file (see "Project" property below). Additionally, if a Subsystem property should have as value the default value specified in the data model, then the "$DefaultValue" keyword is used as value for this property in the Template file.

```
<?xml version="1.0" encoding="UTF-8"?>
<Definition>
    <TopItems ids="1"/>
    <Item id="1" type="Simulation_Model">
        <Children>
            <Child item="2"/>
            <Child item="3"/>
            <Child item="4"/>
        </Children>
    </Item>
    <Item id="2" type="Subsystems">
        <Properties>
            <Property name="Module Id" value="BiW"/>
            <Property name="Project" value="$Project"/>
            <Property name="Variant" value="$DefaultValue"/>
            <Property name="Release" value="$Release"/>
            <Property name="Iteration" value="$DefaultValue"/>
            <Property name="Loadcase Variant" value="$DefaultValue"/>
            <Property name="File Type" value="$DefaultValue"/>
            <Property name="Representation" value="common"/>
        </Properties>
        <Attributes>
            <Attribute name="Name" value="BiW"/>
        </Attributes>
    </Item>
    <Item id="3" type="Subsystems">
        <Properties>
            <Property name="Module Id" value="Doors"/>
            <Property name="Project" value="$Project"/>
            <Property name="Variant" value="$DefaultValue"/>
            <Property name="Release" value="$Release"/>
            <Property name="Iteration" value="$DefaultValue"/>
            <Property name="Loadcase Variant" value="$DefaultValue"/>
            <Property name="File Type" value="$DefaultValue"/>
            <Property name="Representation" value="common"/>
        </Properties>
        <Attributes>
            <Attribute name="Name" value="Doors"/>
        </Attributes>
    </Item>
    <Item id="4" type="Subsystems">
        <Properties>
            <Property name="Module Id" value="Frontsystem"/>
            <Property name="Project" value="$Project"/>
            <Property name="Variant" value="$DefaultValue"/>
            <Property name="Release" value="$Release"/>
            <Property name="Iteration" value="$DefaultValue"/>
            <Property name="Loadcase Variant" value="$DefaultValue"/>
            <Property name="File Type" value="$DefaultValue"/>
            <Property name="Representation" value="common"/>
        </Properties>
        <Attributes>
            <Attribute name="Name" value="Frontsystem"/>
        </Attributes>
    </Item>
</Definition>
```



8.5.2.2. Simulation Model creation through Templates

Simulation Model Templates can be applied manually, through standard KOMVOS GUI functionality, or through script.

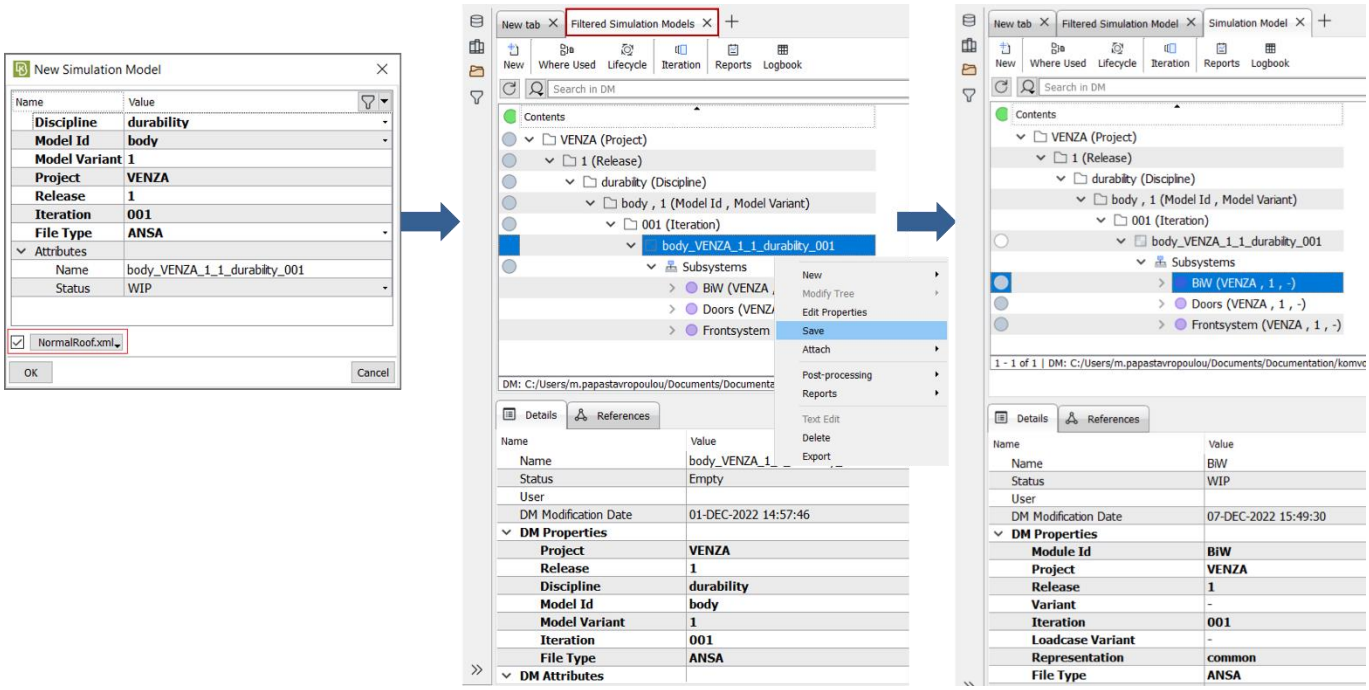
Manually

To create a Simulation Model utilizing a template in KOMVOS, select the option **File > New> Simulation Model.** The *New Simulation Model* window opens in order to define the properties of the Simulation Model. To select a template for the Simulation Model, activate the checkbox at the bottom of the window and choose a template file from the list, as shown below.

! Note that in order for a template to be listed under the menu of the *New Simulation Model* window, its XML file must be placed under the folder:

`$SDM_CONSOLE_HOME/Templates/SimulationModels`

Upon confirmation, a preview of the Simulation Model to be created is shown in a new *"Filtered Simulation Models"* tab, that contains all the Subsystems that will be included in the Simulation Model and are specified by the template file. To proceed with the creation of the Model select the option **Save** from the context menu.



With Script

It is possible to create a Simulation Model in DM, populated with the proper Subsystems, that have their CAD Hierarchy attached, by utilizing the `base.GenerateSimulationModelBasedOnTemplates` function. This function needs the following input arguments:

- A dictionary containing the Properties of the Simulation Model in a key, value pair format, having as key the property name and as value the property value.
- The Simulation Model Template file path to be used in order to create all Subsystems of the Simulation Model. This file path can be an absolute path or a relative path to the Templates/SimulationModels folder of the SDM_CONSOLE_HOME configuration folder.
- A dictionary that will contain a key value pair for each Subsystem of the Simulation Model, having as key the Module Id of the Subsystem and as value the file path of the Subsystem Template. This file path can be an absolute path or a relative path to the Templates/Subsystems folder of the SDM_CONSOLE_HOME configuration folder.
- A set of settings for the import of the model definition specified with the help of the `base.InputProductTreeSettings`.

Upon completion, this function will save the Simulation Model and the contained Subsystems in DM.

9. Optimization Studies and Machine Learning

9.1. Simulation Results

9.1.1. Introduction

9.1.1.1. What is Machine Learning

Machine learning (ML) is a field of Artificial Intelligence (AI), which includes a variety of algorithms that try to “learn” information from data. More specifically, the idea behind ML is that an algorithm can mimic the way humans learn, by feeding them with examples (e.g. similar to the case where a parent shows example images of a cow and indicates that this is a cow to a child) The objective of the ML algorithms is to “learn” a mapping function between inputs (the image of the cow in the previous example) and targets (the label cow). In order to approximate this mapping function, the ML algorithms are given access to a *training set* of labeled examples (i.e. inputs, labels pairs). This approximation is known as a ML model (not to be confused with the ANSA model, typically referred to as just model), and it can be used afterwards to predict the label of a new unseen example when given as input to it. Due to its predictive capabilities, we will refer to the ML model as a predictor from now on.

Machine learning algorithms can be categorized based on the types of labels they are trained on; first, we have the *Regression* algorithms where the label is a real continuous value (e.g. house selling prices, displacements, mass, etc.) and the *Classification* algorithms where the label is a categorical value/class (e.g. cat/dog/bird).

9.1.1.2. Benefits of using Machine Learning for predicting Simulation Results

Typically, ML models are trained so as to be used for automatically replicating (i.e. simulating) human actions (e.g. detecting objects in an image, transcribing text, etc). In our case, FEA solvers can already accurately perform the simulations and predict the real world outcome. So, the question is, what would be the benefit of using ML in this case. The main advantage of ML is that the ML models can provide almost instant predictions (in the matter of a few ms) when given a new unseen input, while a FEA solver might require up to several hours depending on the problem at stake.

9.1.1.3. The Machine Learning model lifecycle

The full ML model lifecycle consists of three major components, the training, the evaluation and the prediction process. During the training process, a training dataset of labeled examples (inputs/label pairs) is provided as input. The training process can then optionally apply feature engineering on the raw inputs to extract meaningful features tailored to the problem at stake. There is also the option to directly rely on the raw inputs, so as to be able to generalize to any kind of problem. Afterwards, feature pre-processing is applied, which includes feature selection and data normalization. Finally, the ML model is trained using the processed data. There are various ML algorithms and each of them has a plethora of hyperparameters that require tuning. The ML software developed for KOMVOS automatically selects the optimal combination of ML algorithm and hyperparamaters. The output of the training process is a full pipeline given a new input, it applies all the aforementioned steps (feature engineering, data preprocessing, ML model) and provides a predicted label as an output. In KOMVOS we refer to these pipelines as predictors.

The next step is the evaluation of a trained predictor. In order to evaluate the predictor, the user is given access to a set of performance indicators (KPIs) (e.g. the expected accuracy or average error). The training set is used to compute these KPIs, which are estimations of the predictor’s expected performance. It is worth noting that this estimation is more accurate for larger sizes of the training set. Finally, when the predictor is evaluated and validated

to be at an acceptable level of accuracy, it can be used to provide new predictions. Given a new set of inputs, the full pipeline of the first component is applied to them and their labels are predicted without having to go through the costly process of applying a FEA solver.

9.1.1.4. Machine Learning applications in KOMVOS

Since v21.0.0, an ***Integrated Machine Learning environment*** with KOMVOS has been introduced. Using this environment, a KOMVOS user can seamlessly train a predictor to predict any kind of output in a problem agnostic manner, i.e. by selecting the input/output pairs to be used for training. The integrated ML environment is responsible for selecting both the ML algorithm and parameters for train the optimal predictor tailored to the problem selected by the user, Such predictors are able to predict a theoretical model's behavior (keyvalue, 2d plots, 3d results) by changing design variable values (DV Based) or using a new FE model design (Feature based). More specifically, based on the types of inputs and outputs, three large groups of applications are available in KOMVOS:

1. Generic Design Variable based Prediction: A predictor can be trained using a model's (ANSA model) Design Variable values as inputs and any selected responses as outputs. The responses can be single scalar values, 2D curves or 3D field results. In this case, the training dataset consists of existing ANSA models (experiments) and their responses. Such datasets are created by ANSA's Optimization tool that can create a Design of Experiments (DOE) and automatically save them as Simulation runs in the selected DM. Utilizing the DM structure of Simulation Models and Simulation Runs, ML training can be performed on selected DOE Studies, which will train a predictor. The predictor can then be given a new set of Design Variable values and it will efficiently predict the responses for these new values.

2. Feature based Frequency Prediction: A predictor can be trained using directly a Finite Element (FE) model (Body in white) as input, while the outputs can be single scalar values related to the first torsional and bending mode frequency. Dedicated feature engineering techniques have been developed so as to extract descriptive features from the FE model tailored for predicting frequencies. More specifically, the extracted features contain information related to mass, materials, shape, etc. of the finite element model.

3. Mode Classification: A mode classifier can be trained using the normal modes result files as input, while the output is the type of each mode's shape. For example, torsional, vertical bending, lateral bending or local modes. Dedicated feature engineering has been developed to extract information from each mode/state of a result file. The features extracted for this process are based on the node displacements of the model for each mode.

9.1.1.5. Enabling the integrated ML environment

To use the integrated Machine Learning environment, the **ML Toolkit** has to be downloaded and installed, and the **ML_SERVER** license feature needs to be enabled. The ML Toolkit installs an environment that contains all the required latest libraries to perform Machine Learning (Tensorflow, Pandas, Numpy, etc.) as well as all the necessary tools to seamlessly train and use predictors in KOMVOS.

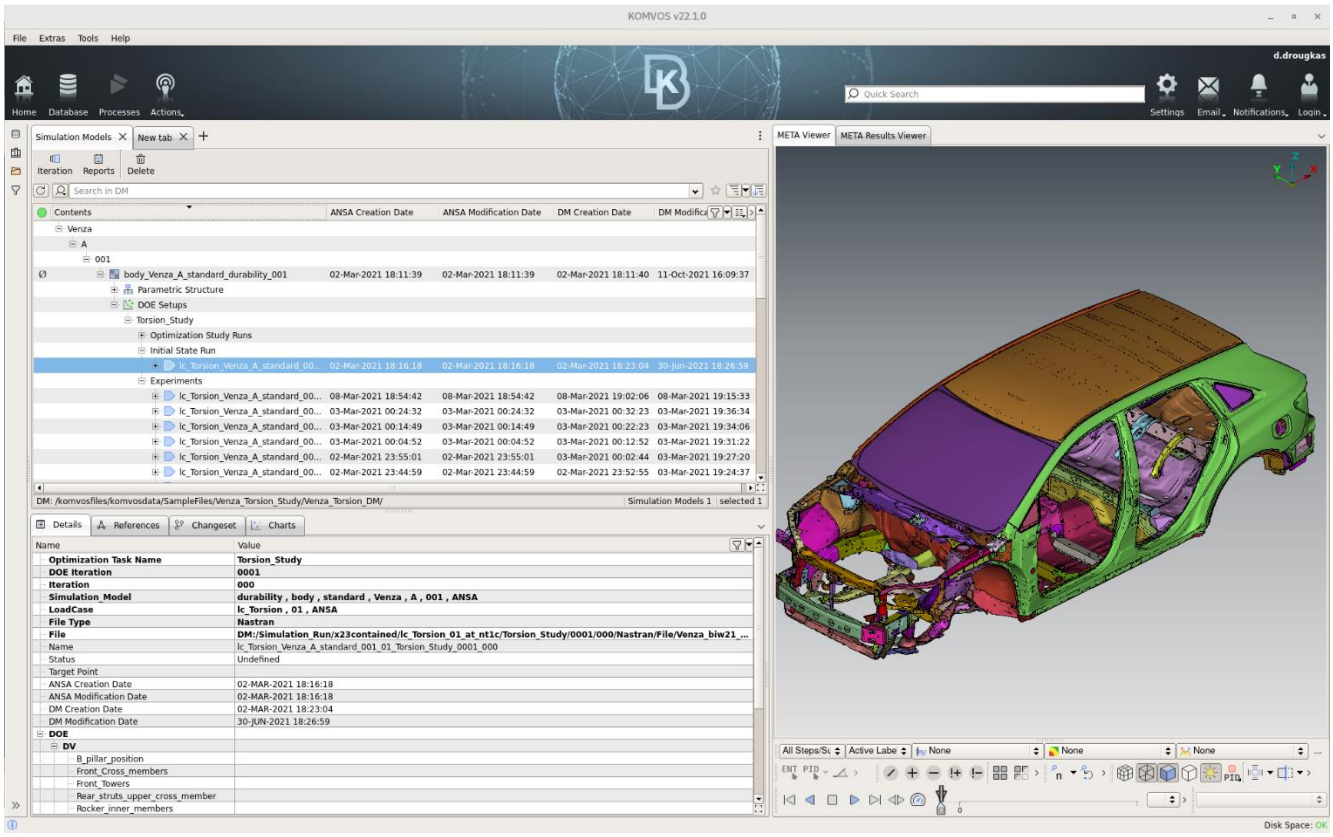
9.1.2. Training Machine Learning models in KOMVOS

9.1.2.1. KOMVOS ML View

Dedicated views have been introduced in KOMVOS in order to enable the three Machine Learning applications.

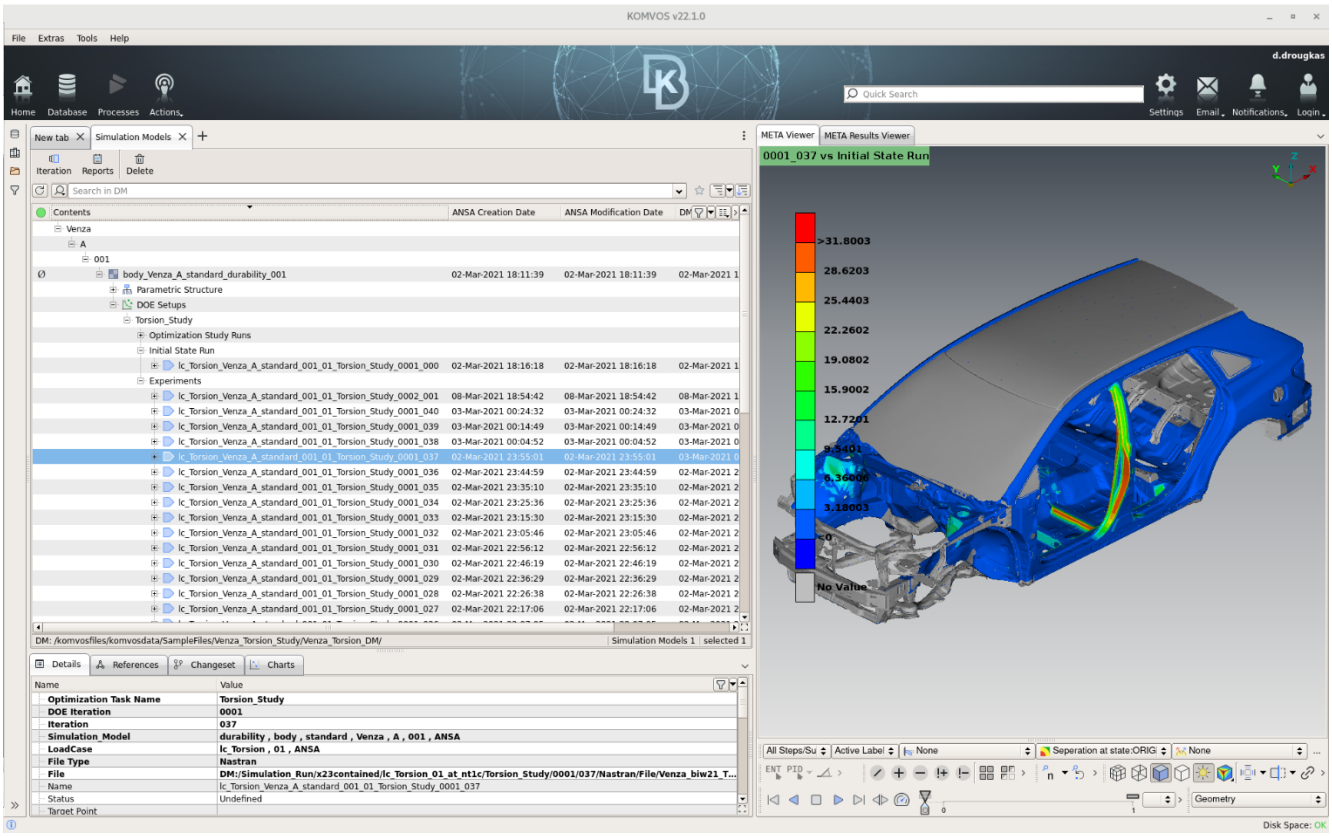
The **Optimization** view of Simulation Runs shows the existing DOE Setups that contain the Simulation Runs. These can be used in order to train the predictors. Only simulation runs that were generated from the same optimization task can be used to train predictors (i.e. one predictor cannot be trained on simulation runs generated by different optimization tasks)

The following image shows the Parametric Structure of the Simulation Model that consists of the Parametric Scenarios and Design changes that were used in order to generate the Simulation Runs.

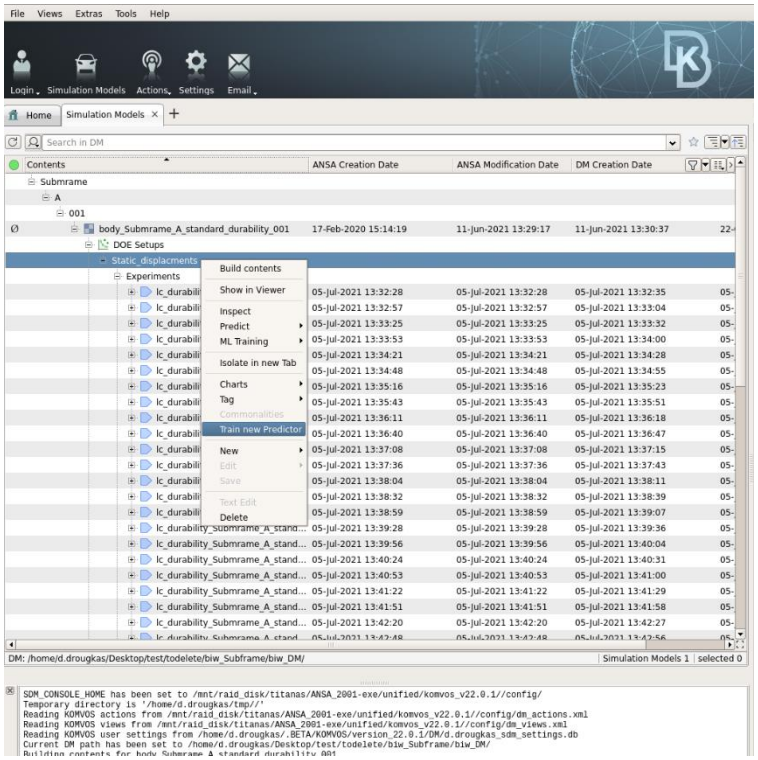


Through the **Simulate** option of the Parametric Scenario, it is possible to see live design changes on the model.

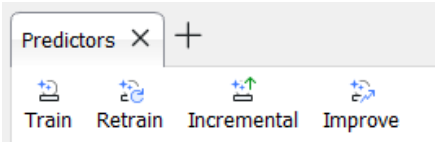
Selected Simulation Runs geometry differences can be compared with each other or directly with the Initial state run, through the *Separation Graph*.

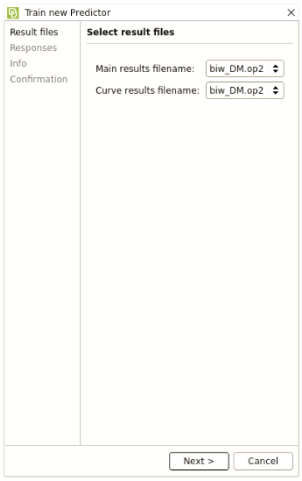


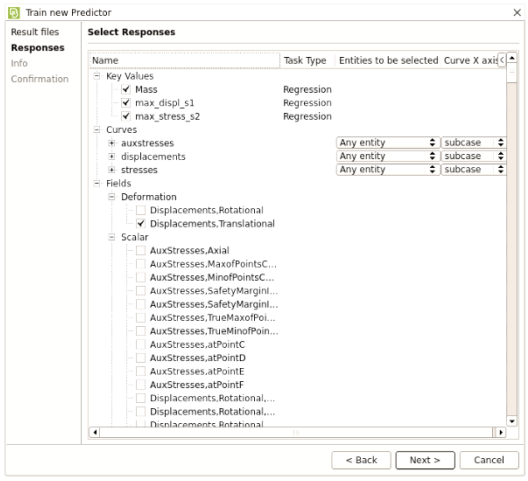
9.1.2.2. Machine learning for DOE Studies



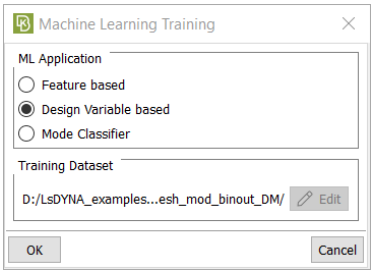
There are two ways to initiate the training of Machine Learning models. By selecting a number of simulation runs and selecting the **Train new Predictor** option of the context menu. Or by the respective **Train** button, in the Predictors tab.

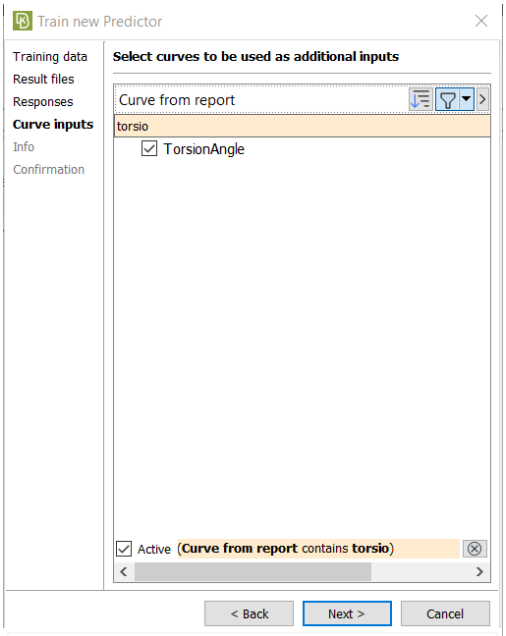






After the selection of the *Machine Learning Application*, the *Train new Predictor* wizard opens to select the respective options. The three available application options are described in paragraph 9.1.4.



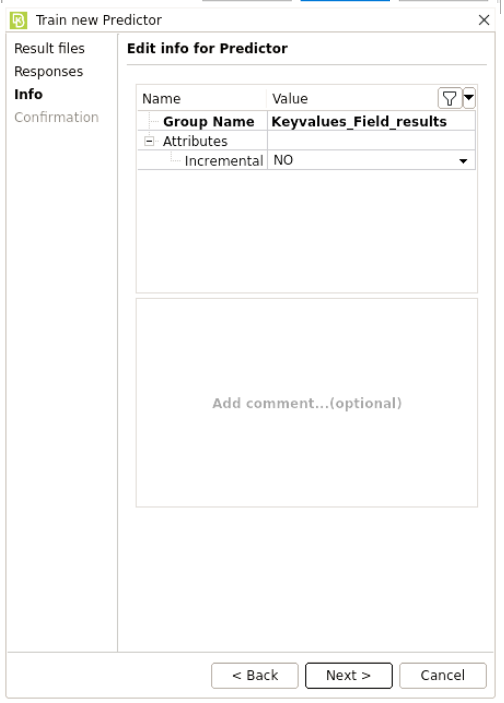


Solver result files containing *3 D (field)* and *2D (curve)* results need to be selected in order to get the full list of possible responses that can be used as outputs for predictors.

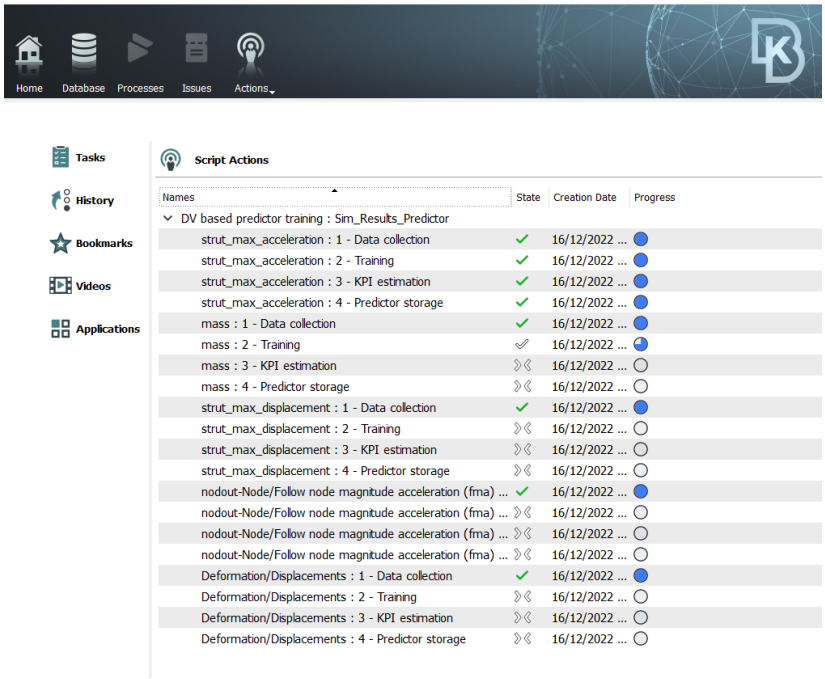
The available responses are presented in a list and multiple of them can be selected as predictor outputs. One **Predictor** will be created for each selected response and the predictors will be grouped in a single **Group.** A **Group name** and other attributes can be defined in the info page.

The **Curve Inputs** allows for the selection of 2D reports to be used as Design variables during the training of a predictor.

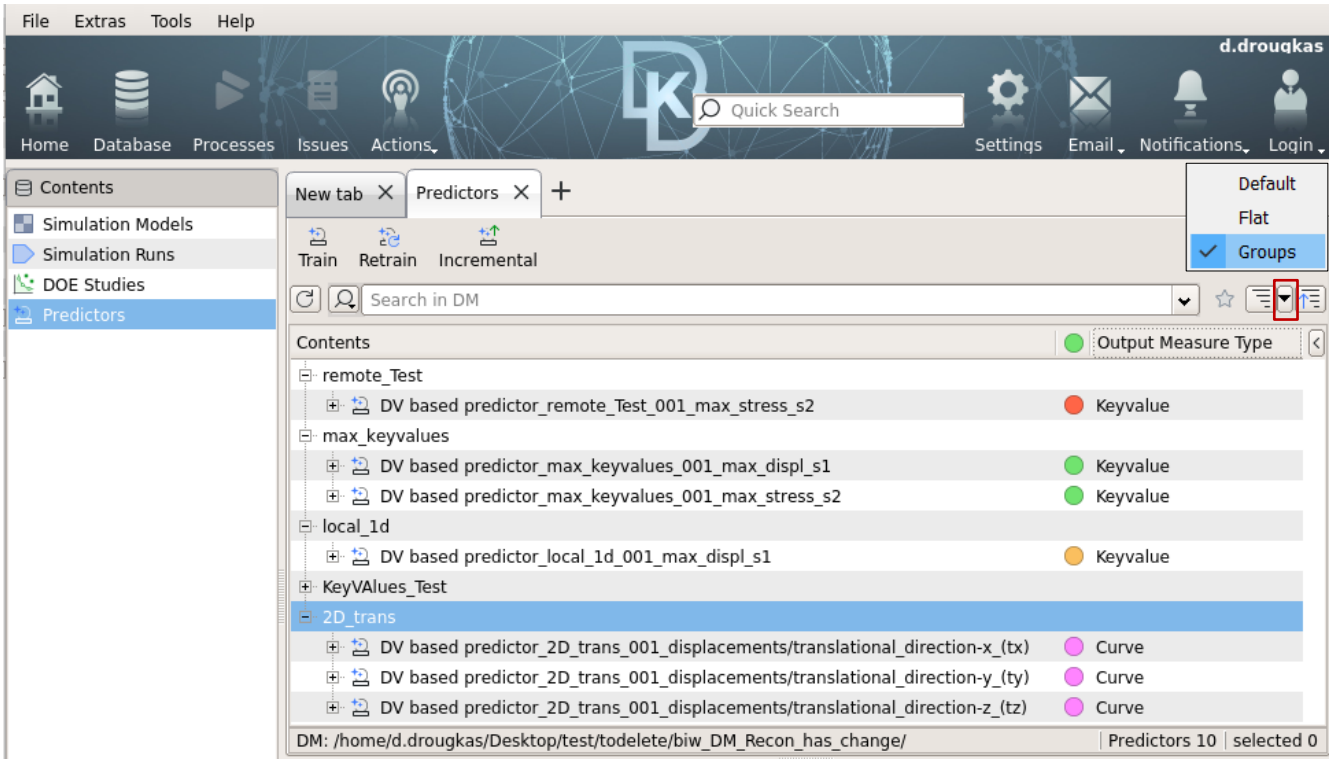
As a result if a predictor is trained to have curves as inputs, it will also require curves as input in order to make predictions of simulation results.



Once **Finish** is pressed, an asynchronous process will start and its status can be seen through the *My Actions* shortcut in the *Home* workspace.


Home
Database
Processes
Issues
Actions
Tasks
History
Bookmarks
Videos
Applications
Script Actions
Names
State
Creation Date
Progress
DV based predictor training : Sim_Results_Predictor
strut_max_acceleration : 1 - Data collection 16/12/2022 ...
strut_max_acceleration : 2 - Training 16/12/2022 ...
strut_max_acceleration : 3 - KPI estimation 16/12/2022 ...
strut_max_acceleration : 4 - Predictor storage 16/12/2022 ...
mass : 1 - Data collection 16/12/2022 ...
mass : 2 - Training 16/12/2022 ...
mass : 3 - KPI estimation 16/12/2022 ...
mass : 4 - Predictor storage 16/12/2022 ...
strut_max_displacement : 1 - Data collection 16/12/2022 ...
strut_max_displacement : 2 - Training 16/12/2022 ...
strut_max_displacement : 3 - KPI estimation 16/12/2022 ...
strut_max_displacement : 4 - Predictor storage 16/12/2022 ...
nodout-Node/Follow node magnitude acceleration (fma) ... 16/12/2022 ...
nodout-Node/Follow node magnitude acceleration (fma) ... 16/12/2022 ...
nodout-Node/Follow node magnitude acceleration (fma) ... 16/12/2022 ...
nodout-Node/Follow node magnitude acceleration (fma) ... 16/12/2022 ...
Deformation/Displacements : 1 - Data collection 16/12/2022 ...
Deformation/Displacements : 2 - Training 16/12/2022 ...
Deformation/Displacements : 3 - KPI estimation 16/12/2022 ...
Deformation/Displacements : 4 - Predictor storage 16/12/2022 ...

Once the training is completed, a Group of Predictor is created and it can be seen in the **Predictors** tab. The *Groups* view of the Predictors shows groups of Predictors as they were created during training.

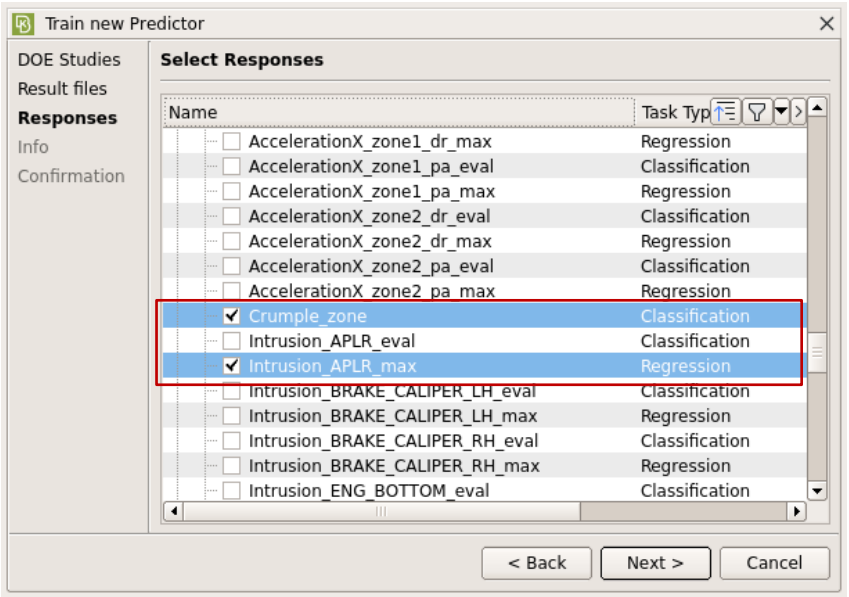


Initially the Predictor status is "*Pending Verification*", with a characteristic orange color. Each predictor contains **Reports (KPIs)** that describe its behavior, performance and accuracy. After manual inspection of the KPIs, a Predictor's status can be changed to "*Verified*" (green), using the context menu option **Verify > OK**, or "*Rejected*" (Red), using the context menu option **Verify > Failed**, if the predictor is of satisfying performance or not respectively. When the status color is white, it means the Predictor was created in previous versions and when the color is pink, there was an error during training. Gray color means the Predictor is currently under training.

Failed Predictors (red) cannot be used to make predictions.

9.1.2.3. Predictor Task Type

As mentioned in section 9.1.1 predictors can either be trained for regression problems or classification problems based on the selected output. This applies only to key value outputs (i.e. single scalar values). The categorization of *Regression* or *Classification* is done automatically during the definition of training, in the *Responses* page of the training wizard. The task type applies only for key values for both Feature based and Design Variable based predictors.

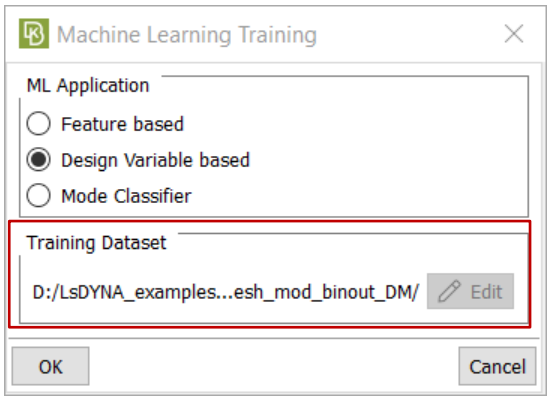


If the key values are purely numerical, then the Predictor will be of type Regression. Otherwise, in case of alphanumerical or purely string values (describing a state or status of the model) then the Predictor will be of type *Classification*.

My Experiments

Iteration	FrontRail1 (1.377-2.04)	FrontRail2 (1.377-2.04)	FrontRail3 (1.62-2.4)	FrontRail4 (1.62-2.4)	FrontRailSideMembers (1.134-1.68)	RailToBumper (1.701-2.52)	Crumple_zone	Intrusion_APLR_max
0001	1.7029	2.0175	1.8315	1.6332	1.1895	2.5061	Front	0.9829346
0002	1.4557	1.4781	2.1753	1.6861	1.4209	2.0203	Rear	1.24862591

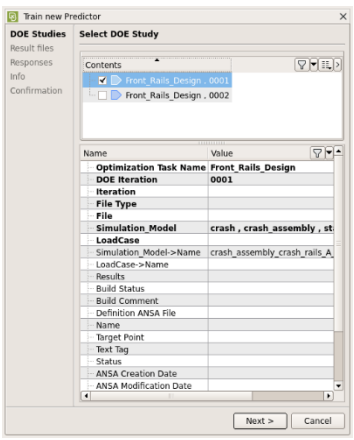
9.1.2.4. Remote Training

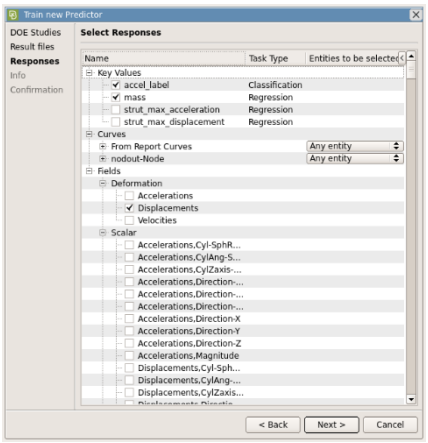


Training of Machine Learning predictors can be performed remotely if KOMVOS is connected to an SPDRM server.

Once the connection is established, the **Edit** button is activated and the training dataset can be selected.

NOTE!: The Machine Learning working directory (can be modified in the *Settings*), as well as the dataset, have to be accessible from the SPDRM Server. Additionally to perform remote training, **umask 0** must be defined to provide full permissions before starting KOMVOS.



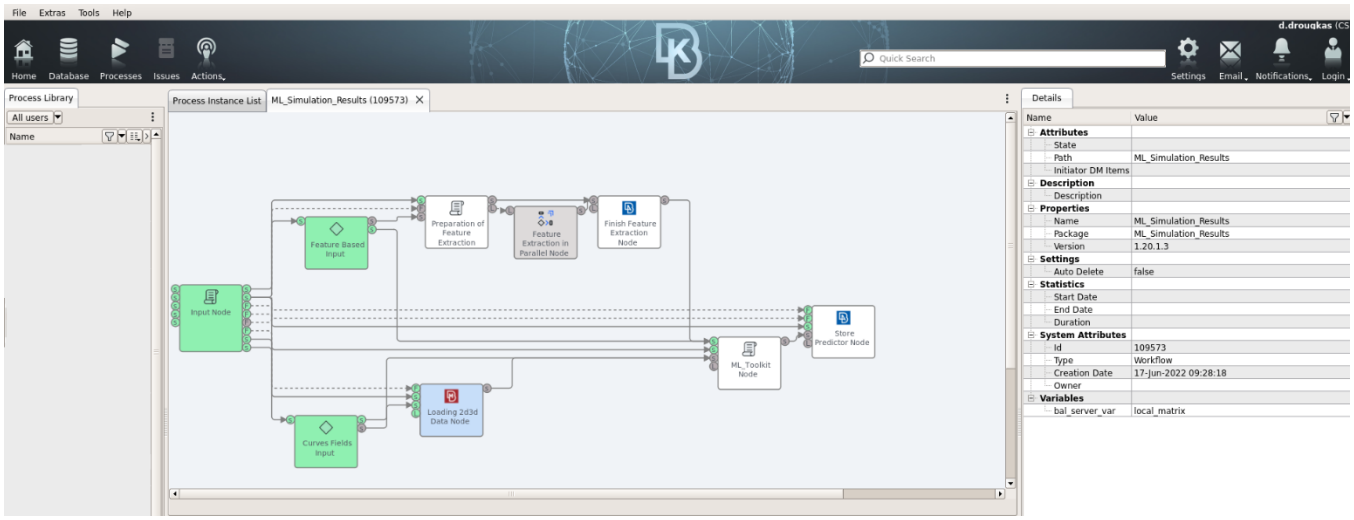


After the selection of the Training Dataset location, the *Train new Predictor* wizard opens to select the respective options.

Respective result files need to be selected for the training of Key Values, *3D (field)* and *2D (curve)* predictors.

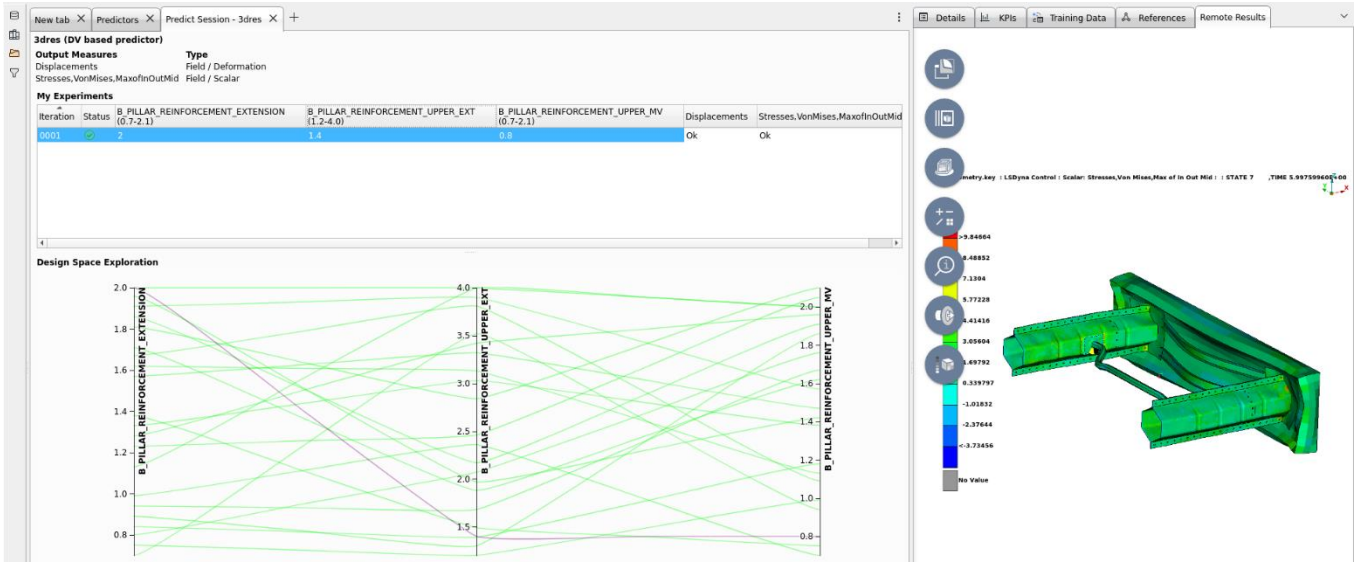
The required responses need to be selected from the available options and a name and other attributes can be defined in the info page.

Once **Finish** is pressed, an asynchronous process will start and its status can be seen through the *My Actions* shortcut in the *Home* workspace, while the SPDRM workflow can be seen in the *Process Instance* List through the *Processes* workspace.



9.1.2.5. Remote Prediction

Design variable based Machine Learning predictions can be performed remotely if KOMVOS is connected to an SPDRM server. When the prediction is done on the server machine the results are added in the respective lists and 2D plots or Field results are streamed back to KOMVOS through META Viewer.

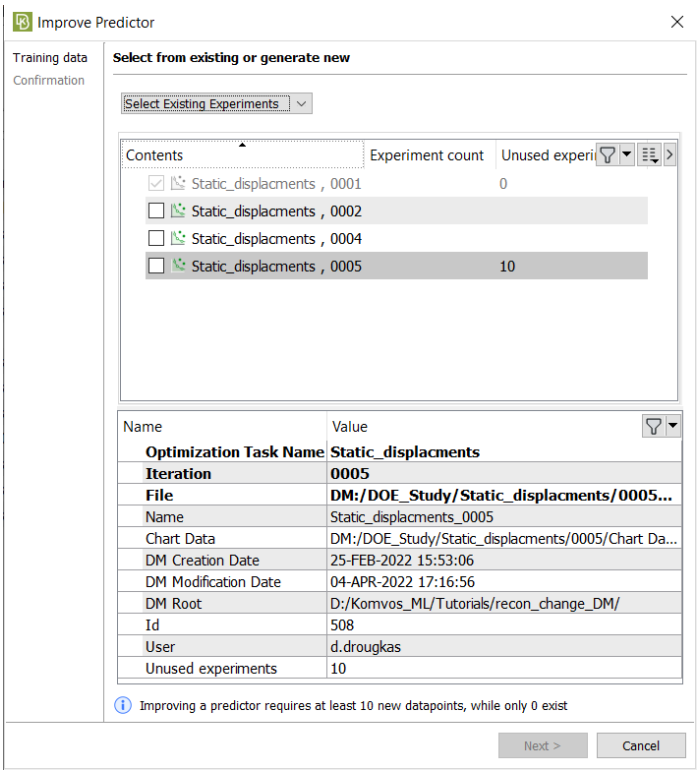


9.1.2.6. Improve

If the performance of an existing predictor is not satisfying, the Improve option offers tools to “update” predictors with additional data.

There are two ways to improve existing predictors; one by using experiments already existing in the DM and one by create new experiments.

In the **Training data** page, the option *Select Existing Experiments* will list DOE Studies that are available, the number of experiments in each DOE study and how many of these experiments are not used in the selected predictor. When a selection is done and after the confirmation, the predictor improvement will start. If the predictor supports **Incremental** learning, the update will be done this way. Alternatively the **Retrain** method will be used.

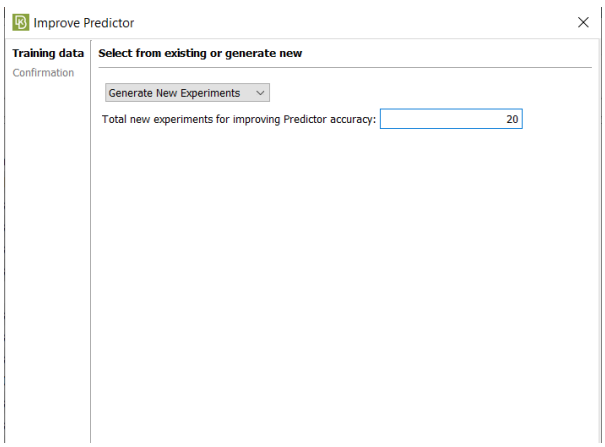


A predictor that supports **Incremental** learning can be “Improved” with new data. This is an option that can to be activated when the predictor is originally created. This means that the new predictor will be trained in the new data, while keeping all the information of the old predictor. This is a faster training process than re-training with all the data, old and new. However, not all ML algorithms support incremental learning, so the predictor will select the best configuration from the subset of ML algorithms that do support incremental learning.

Incremental learning can be used for both DV and Feature based predictors.

In case the selected predictor does not support incremental learning, the **Retrain** method will create a new predictor that will be trained with both the original data of the selected predictor and also new selected data. This process is slower than incremental learning; however it considers all possible ML algorithms in the best configuration selection process. The new predictor that will be created will be able to predict for the responses of the original predictor.

This method option can be used for both DV and Feature based predictors, basic or incremental



In case there are no new experiments (training data) to use, the **Generate New Experiments** option can be used. The new experiments will be selected through the process of Smart Sampling, i.e. selecting the experiments that will optimally improve the selected predictor.

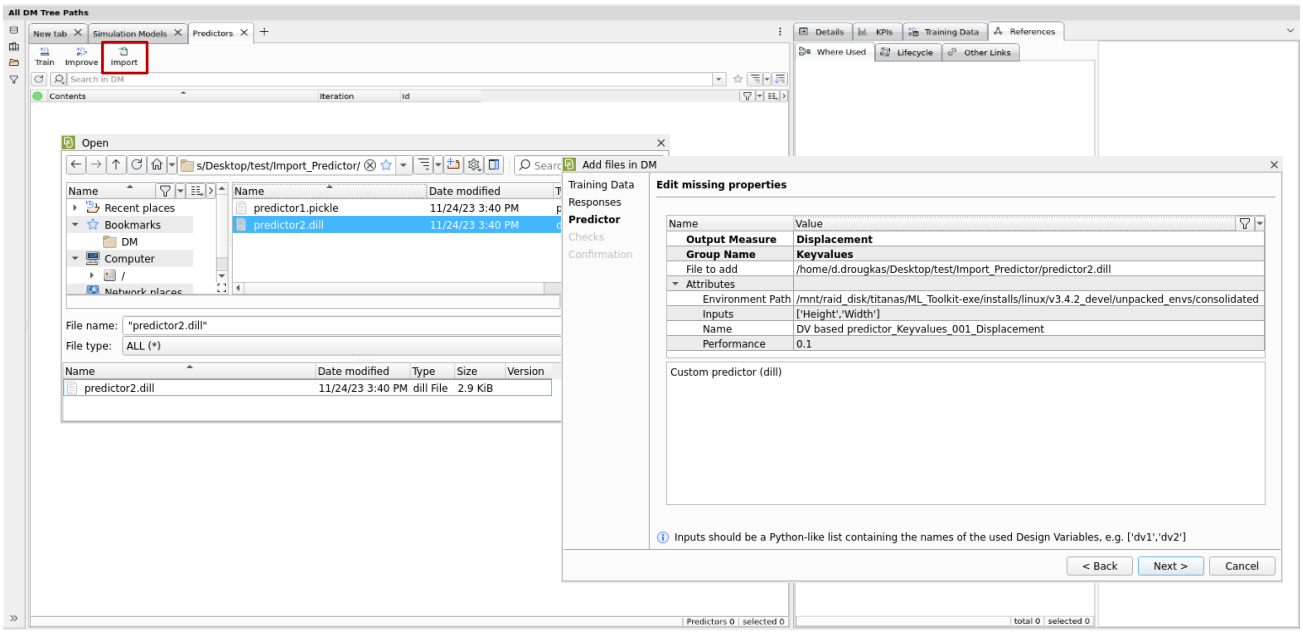
After selecting the *Generate New Experiments* option for a selected predictor, will ask as input for the number of new experiments to be added. Based on the predictor details (e.g. number of DVs) there will be a suggested size for the training set.

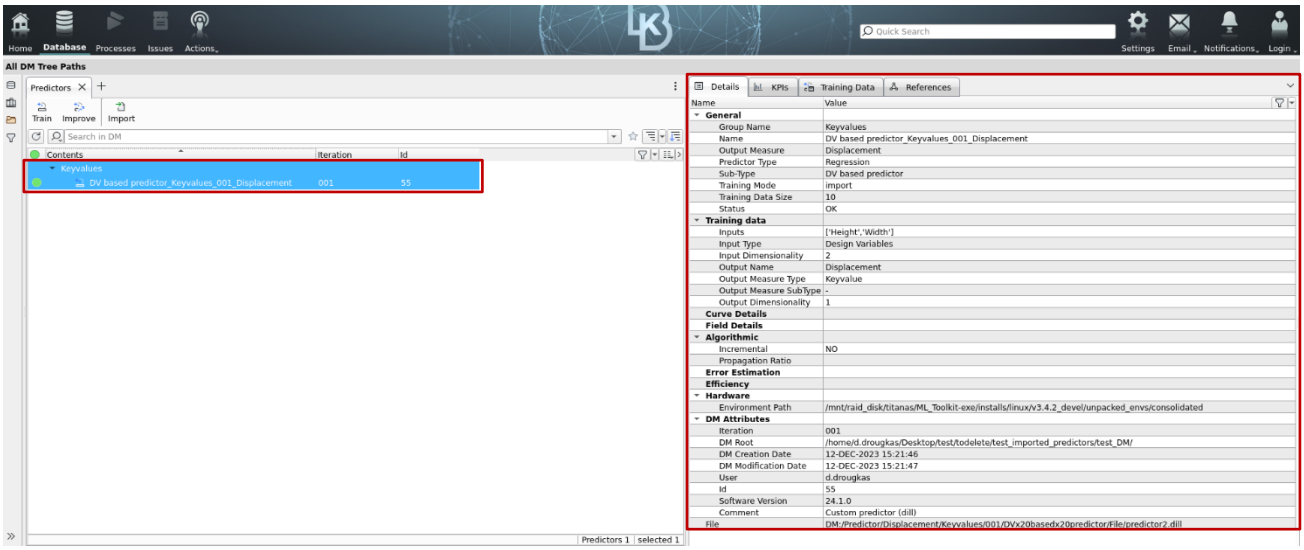
Note that in order for the new experiments to be created, a complete optimization task is required as created in ANSA (containing Pre Processing, Solver and Post processing). The new experiments will be added in a new DOE study.

This **Improve** option can be used only for DV based basic predictors (i.e. from train and retrain options). It does not support incremental predictors.

9.1.2.7. Import

User defined key value predictors created in a custom anaconda environment can be imported in KOMVOS and use the existing Predict functionality (Training Data Visualizations, Predict, Inverse Predict, Model Explanation) .This requires a predictor object saved in *.dill format (using the dill module). This predictor object should implement a function predict that takes as input the design variable values and returns the predictions in numpy arrays, following an interface similar to the sklearn Estimators. For the custom anaconda environment, it is required to have all the modules so that the imported predictor object can be un-dilled (including the dill module). Furthermore, in order to be able to use the model explanation functionalities, the dalex and plotly modules should be installed in the environment. Additionally two or more Simulation Runs should be available in the DM in order for the Predict session of KOMVOS to be able to present the correct design variable and responses names and values during predictions. An example of such an import along with an Interface Predictor class is available in the Machine Learning tutorial of KOMVOS.





9.1.3. Evaluating Predictors through KPIs

Every predictor contains Report items that include KPIs, that describe its behavior, performance and accuracy.

These reports are created during the training of the Predictor and are used for the evaluation of a Predictor.

Prior to the training, a percentage of the selected data (*Train/Test* ratio in Settings) is used for Training and a percentage is used for Testing. The data that will be used for testing are not used in the training.

Once the predictor training is finished, the test data are used in order to impartially test and produce these KPI reports that can be single valued (Accuracy metrics, or plot-based (Performance plots). These KPIs are analyzed in the following sections.

Note that in the following sections the terms simulation runs, datapoints and experiments are used interchangeably.

9.1.3.1. Accuracy Metrics

The prediction error is measured using the Mean Absolute Error (MAE) values. Please note that for Field and Curve responses, the MAE also averages over entities, states and curve points.

The Estimated Test MAE is computed either on a held-out test set (if there is sufficient amount of data) or using nested cross validation (if the amount of data is limited). The threshold for data below which nested cross validation will be used can be defined in the Machine Learning settings.

The Train MAE on the other hand is computed on the train data. It does not give a direct indication about the generalization capabilities of the predictor. However, if it is compared to the test MAE, it can be useful in order to detect if there is over fitting (possibly due to a limited amount of data). For instance, this would be the case if the train MAE is much lower than the test MAE.

Mean Absolute Error (MAE)

The MAE is computed as the average absolute error over a set of points. For instance, in the case of a single scalar keyvalue target, the MAE is computed as follows:

$$\mathrm{MAE} = \frac{1}{N}\sum_{i=1}^{N}(y_i - f(x_i))$$

where **N** is the number of simulation runs used for computing the KPIs (also referred to as datapoints), $\mathbf{y_i}$ the target value for datapoint **i** and $\mathbf{f(x_i)}$ is the predicted value for datapoint **i**.

Mean Squared Error (MSE)

The mean squared error is a metric corresponding to the expected value of the squared (quadratic) error or loss. If $\hat{\mathbf{y}}_\mathbf{i}$ is the predicted value of the $\mathbf{i^{th}}$ sample, and $\mathbf{y_i}$ is the corresponding true value, then the mean squared error (MSE) estimated over **n** samples is defined as:

$$MSE(y, \hat{y}) = \frac{1}{n_{samples}} \sum_{i=1}^{n_{samples}-1} (y_i - \hat{y}_i)^2$$

Root Mean Squared Error (RMSE)

The Root Mean Squared Error is the square root of MSE. MSE is measured in units that are the square of the target variable, while RMSE is measured in the same units as the target variable. Due to its formulation, MSE, just like the squared loss function that it derives from, effectively penalizes larger errors more severely.

Mean Absolute Percentage Error (MAPE)

The Mean absolute percentage error is a measure of prediction accuracy of a prediction method in statistics. It usually expresses the accuracy as a ratio defined by the formula:

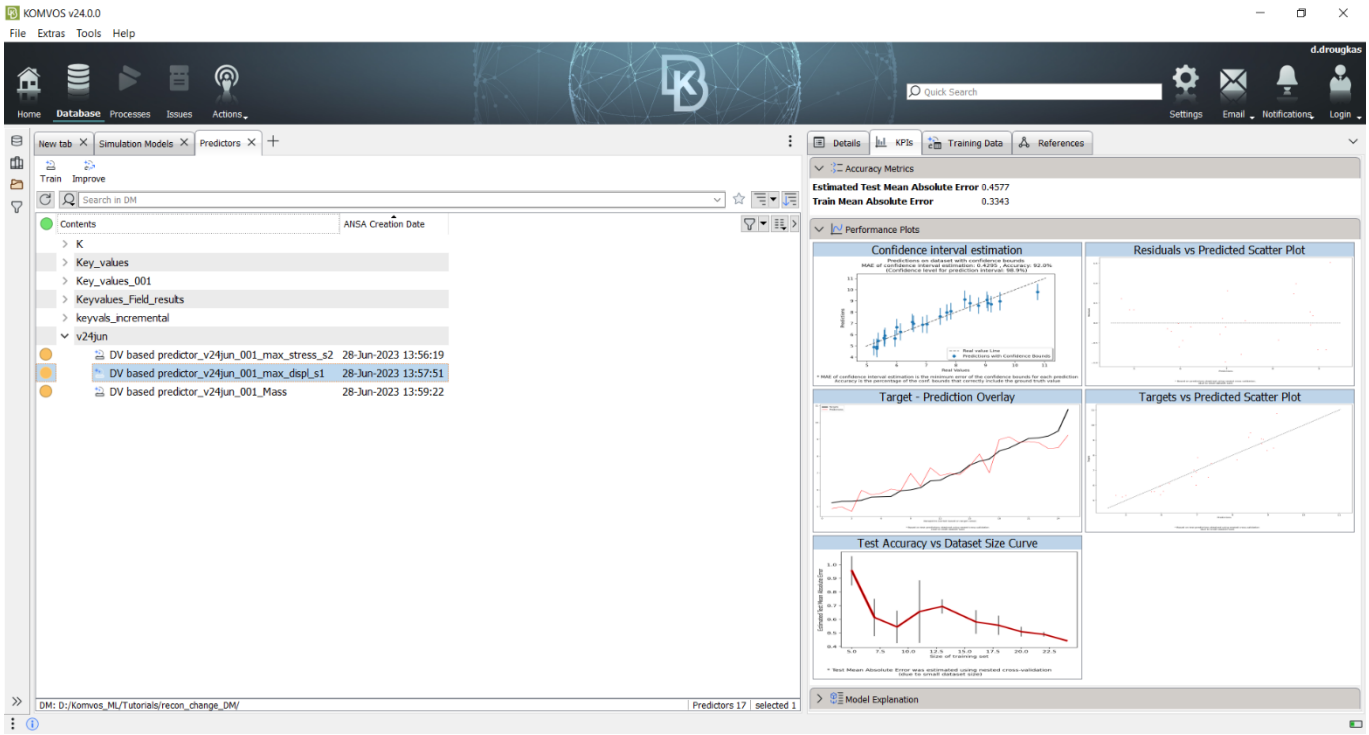
$$MAPE = \frac{1}{n}\sum_{t=1}^{n}\left|\frac{A_t - F_t}{A_t}\right|$$

where $\mathbf{A_t}$ is the corresponding true value and $\mathbf{F_t}$ is the prediction value. Their difference is divided by the corresponding true value $\mathbf{A_t}$. The absolute value of this ratio is summed for every predicted point in time and divided by **n**, the number of fitted points.

Batches

In incremental training, the data arrive progressively, as a predictor is incrementally trained again and again with new sets of data (also referred to as batches). As new batches arrive, we want to know how the predictions for data seen in previous batches are affected. For instance we would like to see that our predictions for data from the first batch remain accurate after seeing the new data. For that reason, the test MAE is also computed for each batch that has been seen so far. A more in depth understanding of how predictions on previous batches are affected, can be obtained by examining the accuracy scatter plots on predictor's images, where the data from different batches are indicated with different colors.

9.1.3.2. Performance Plots



Residuals vs Predicted Scatter plot

This scatter plot shows the residuals between Predicted and FE result (target) values. Points closer to the 0.00 line, mean predictions are very close to the target values. The points could be created based on predictions obtained using cross-validation or nested cross-validation, based on the number of initial datapoints and the respective setting.

Target – Prediction Overlay

This chart shows the target values (black line) of the test datapoints and overlays the prediction values (red line) for these datapoints. A better "fit" of the red line on the black line suggests accurate predictions.

Targets vs Predicted Scatter plot

This scatter plot shows a comparison between the Prediction values and Target values. Points closer to the dashed line suggests accurate predictions.

Test Accuracy vs Dataset Size Curve

This chart shows how the estimated test error changes as the size of the training set increases (also known as learning curve). The estimates of the test error carry some uncertainty. The red line indicates the average test MAE, whereas the vertical gray lines indicate the estimated standard deviation around the average. For the calculation of this plot, the best configuration selected for the complete dataset is used to train predictors with data subsets of varying training sizes.

Confidence Interval Estimation

This chart shows test predictions with the corresponding confidence intervals. Ideally the predictions should be on the diagonal line or the confidence intervals should intersect it. Two performance measures are also shown in the plot: a) Accuracy measure, which shows the percentage of the confidence bounds that correctly include the target value, b) MAE of Confidence intervals, which shows the minimum error of the confidence bounds for each prediction.

Every single scalar Keyvalue prediction is followed by a confidence interval also known as Confidence bounds. This interval is generally narrower in areas where the predictor has seen a sufficient amount of data, whereas it is wider otherwise. This indicates how much a prediction can be trusted.

Test accuracy vs Improve iterations

This chart is only available for predictors that have been through an improve process at least once. It shows the estimated test error as experiments are solved and added in the training set. Similar to the learning curve, the uncertainty of the estimated errors are also displayed as vertical gray lines. This line has the same number of error indications to the number of experiments selected for the improve process (note that if you select to improve with just 1 experiment this plot will show only one point/dot). For the calculation of this line, for each experiment/datapoint added in the training set, the training process of selecting the best configuration is applied. This creates higher fluctuations on the estimated errors when each datapoint/experiment is added in the training set.

Test accuracy vs Group iterations

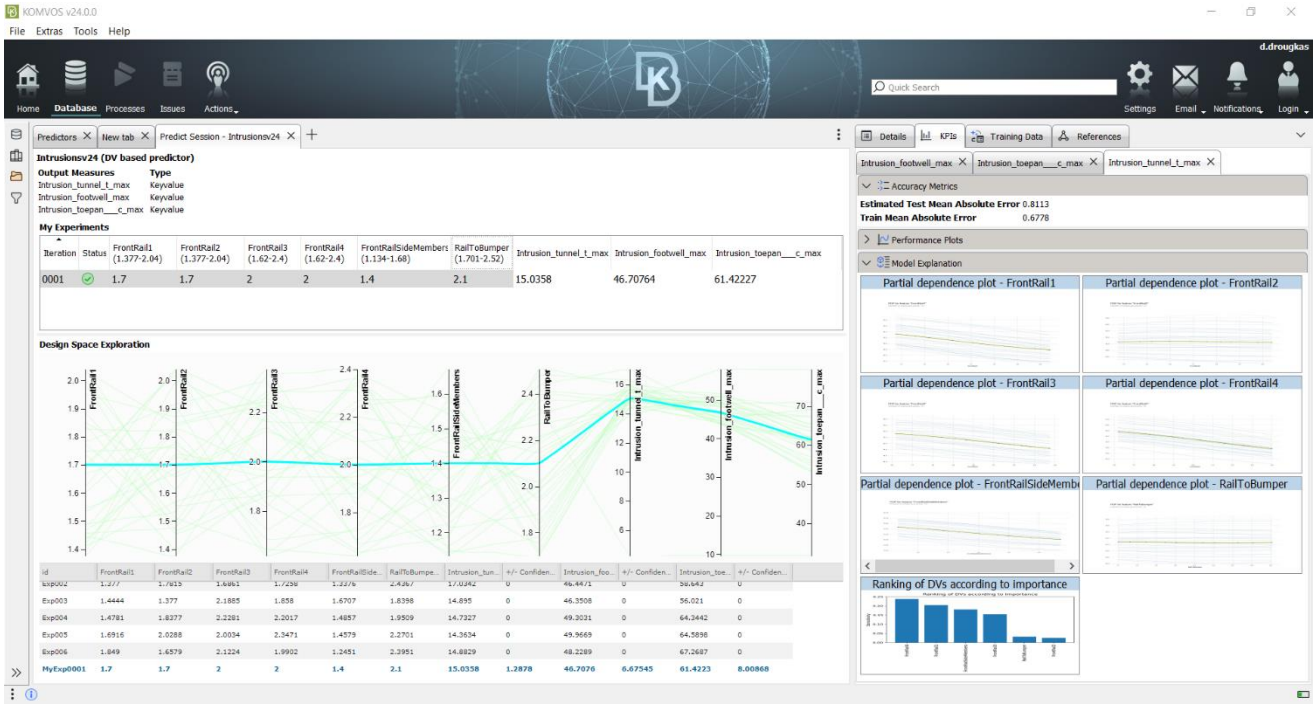
This chart is only available for predictors that have been through an improve process at least once. It shows the history of the estimated test error as the predictor evolved through iteratively applying improve processes. For the calculation of this line, we go through the lifecycle of the predictor and plot the errors of each of the predictors it is created from.

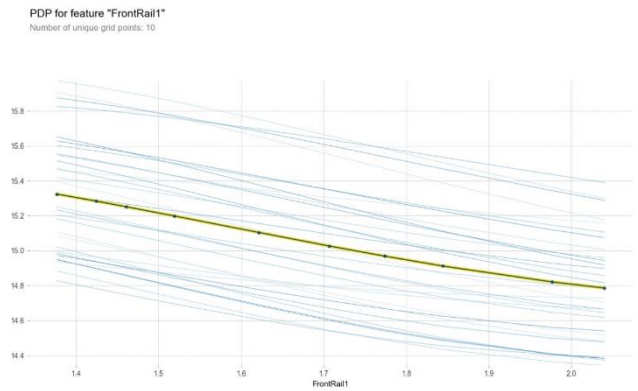
9.1.3.3. Model Explanation

To improve on understanding why a ML model is predicting the values it does, explanation plots have been added on Key Value predictors, either as KPIs in the Model Explanation tab, or on demand after the model is created, providing more information on the behavior of such ML models.

Ranking of DVs according to importance

This chart shows the sensitivity of the predictor output to its inputs (e,g how the predictor response is affected by changes of each design variable). Higher values for an input mean this input affects the predicted response the most.

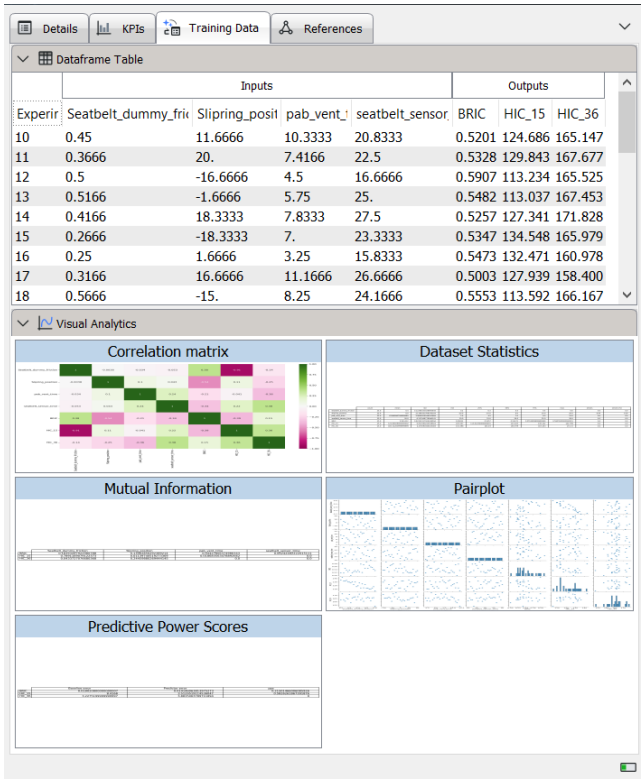




Partial dependence plots are created during ML Training. They appear in the Model Explanation tab of the Predictor's KPIs and show the expected prediction as the values of an input variable change. Every pale blue line corresponds to a data point from the training set. To create this line the dv values for the specific data point (experiment) are selected and predictions are taking place for this experiment by changing the value of one of the Design Variables (the one that this plot refers to). The values of the other Design Variables remain constant for the prediction.

The number of predictions is the number of discrete values that was used for this specific DV throughout the training dataset, and these are the dots that appear on the black-yellow line. This bold black/yellow line is the mean of all the other pale lines and shows the tendency of the response value as the Design Variable that we examine in this plot is modified.

9.1.3.4. Training Data



Data visualization charts help in better understanding a DV Based training dataset. These charts are created during DV Based ML Training or with the option **Explore Data** at the final page of the ML Training wizard.

Additionally the **Training Data** can be completed for predictors created with older versions, using the **Generate visual analytics plots** button in the **Training Data** tab

The *Correlation matrix* is one of the most popular data analysis techniques. It represents the relationship of two variables (e.g. a target and a DV) and takes values between -1,1. Values close to 0 mean that there is no linear relationship between the two variables.

Predictive power score (PPS) is an asymmetric, data-type-agnostic score that can detect linear or non-linear relationships between two columns. The score ranges from 0 (no predictive power) to 1 (perfect predictive power).

In the *Pair plots* the diagonal is the distribution of each DV/target and each other plot, is a scatter plot of the row DV/target and the column DV/target. The scatter plots can show various patterns connecting the variable in the row with the variable in the column.

Mutual information (MI) between two random variables is a non-negative value, which measures the mutual dependency between the variables. More specifically, it quantifies the "amount of information" obtained about one random variable by observing the other random variable.

In the *Dataset statistics* tables, Count, mean, min, max st. deviation and distributions can reveal important information about the dataset like possibility of outliers and experiments similarities.

9.1.3.5. Classifier Reports

In case of classification tasks, the produced reports are different from those of the predictors used for regression tasks, due to the nature of the two tasks. The same reports are extracted also in the cases of Mode Classifiers. The *'Ranking of DVs according to importance' and 'Test Accuracy vs Dataset Size Curve'* reports are the same for both task types. Besides these two similar KPI reports, the following are provided:

Accuracy

The accuracy of a classifier is the ratio of correct predictions it achieves:

$$\frac{TP + TN}{TP + FP + TN + FN}$$

, where:

- TP – True Positive: are the number of datapoints classified as positive and their ground truth label also being positive

- TN – True Negative: are the number of datapoints classified as negative and their ground truth label also being negative
- FP – False Positives: are the number of datapoints classified as positive and their ground truth label being negative

FN – False Negative: are the number of datapoints classified as negative and their ground truth label being positive

Precision

A classifier's precision denotes the ratio of positive predictions that are truly positive:

$$\frac{TP}{TP + FP}$$

Recall

Recall calculates the ratio of positive samples that are actually predicted as positive:

$$\frac{TP}{TP + FN}$$

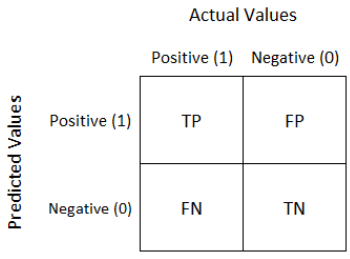
It is also called true positive rate or sensitivity.

In case of multiclass classification, the above metrics are computed as the mean value for all the classes weighted by the number of true instances of each class. In addition, these 3 metrics are computed for both the training and test datasets.

As with the MAE metric, the test metrics are computed either on a held-out test set or using nested cross-validation. The threshold for data bellow which nested cross-validation will be used can be defined in the Machine Learning settings.

Confusion Matrix

In this table, each row represents the true instances in a class, while each column represents the predicted instances. Many metrics may be derived from this chart including (but not limited to) the accuracy, precision and recall. The more diagonal this matrix is, the more accurate the classifier. More specifically, the confusion matrix for two classes can be seen bellow.



Classes Histogram

This chart shows the number of instances of each class. It is useful in order to visualize potential class imbalances in a dataset. Note that in the case of class imbalance, it is better to use the Precision and Recall metrics together, as the Accuracy metric may not be as representative of the predictor performance.

ROC (Receiver Operating Characteristic) Curve

The ROC Curve plots the true positive rate (recall) over the false positive rate for each class in the dataset.

A ROC curve close to the diagonal line (y=x) indicates low performance, while a ROC curve closer to the upper left-hand corner of the plot indicates better performance.

9.1.4. Predicting Simulation Results

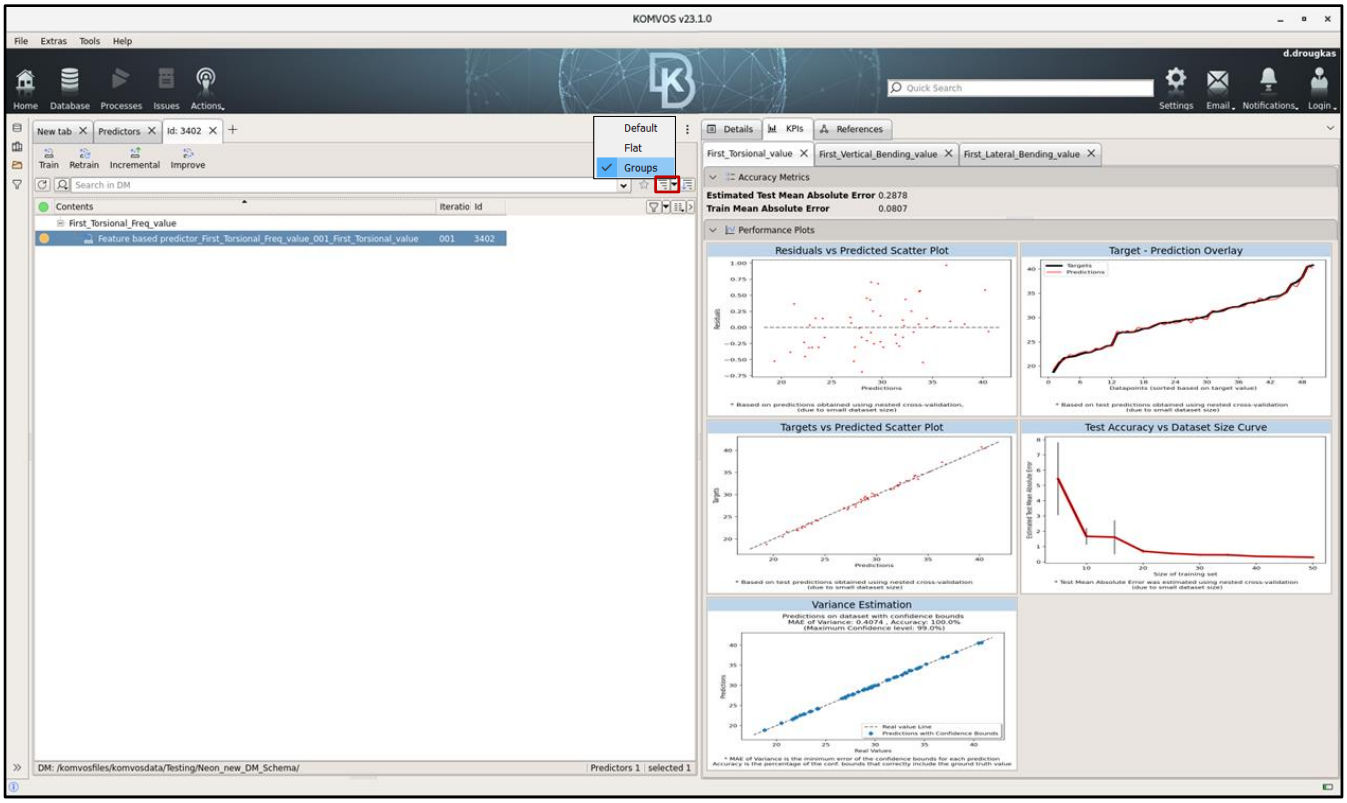
9.1.4.1. Feature based Predictors

Feature based Machine Learning uses as input the Finite element model and a selected response, and performs feature extraction. Through this process specific features of the FE model are captured and train the Machine Learning model. Feature based predictors are tuned to be trained and predict First Torsional, Vertical and Lateral Bending modes frequency values of a finite element body in white. No design variables are needed in order to train this type of Predictors.

Once the training process is finished, a Predictor will appear in the relative tab.

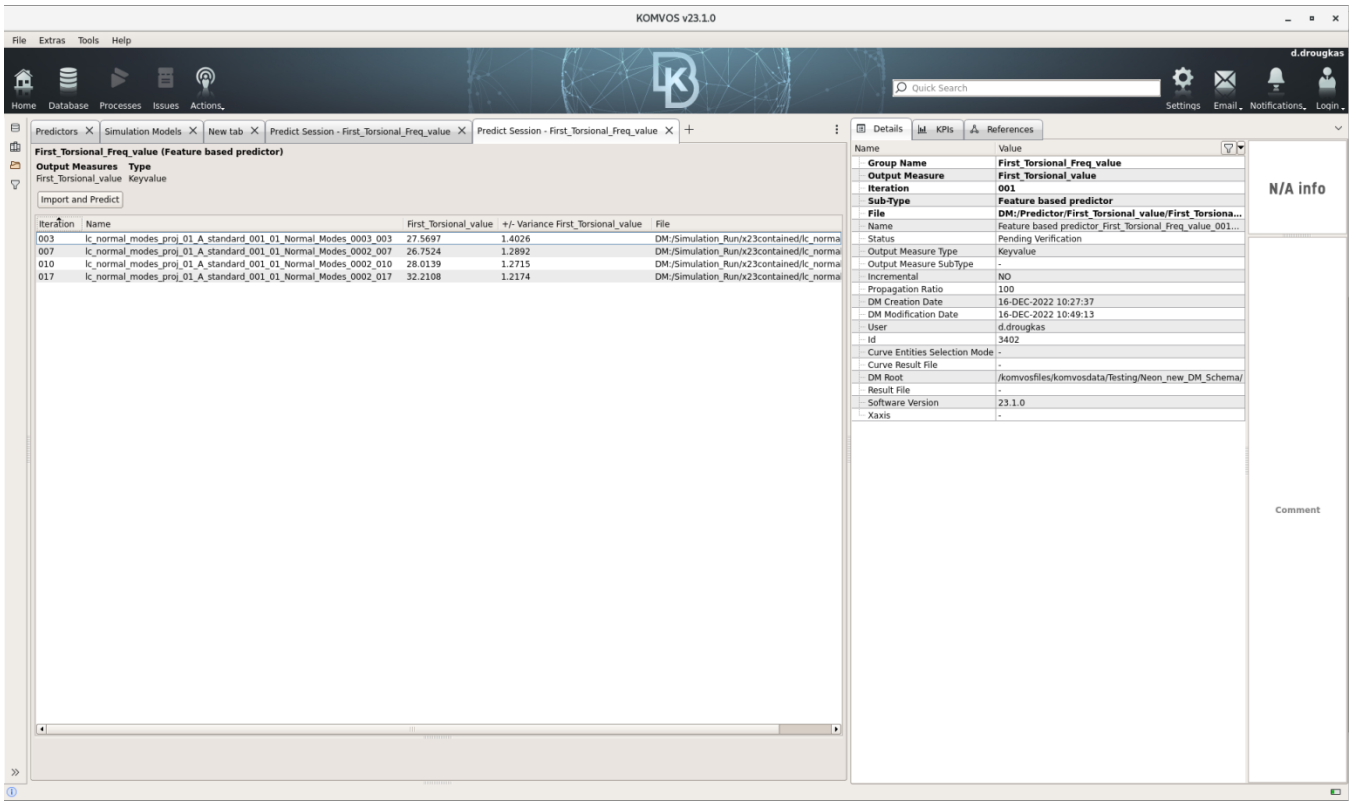
The *Groups* view of the Predictors shows groups of predictors as they were created during training.

Predictors contain reports (KPIs) of performance (accuracy charts, mean absolute error values, etc.) and can be used in order to make predictions for the responses they have been trained for.



Through the option **Predict session**, the predictions can be initiated they and appear in the relative window, for the respective trained responses.

For the Feature based Predictors, predictions can be performed for selected Simulation Runs existing in the DM, using the context menu option **Predict**, or through the **Import and predict** button of the Feature based Prediction window, for external body in white models.



9.1.4.2. Design Variable (DV) based Predictors

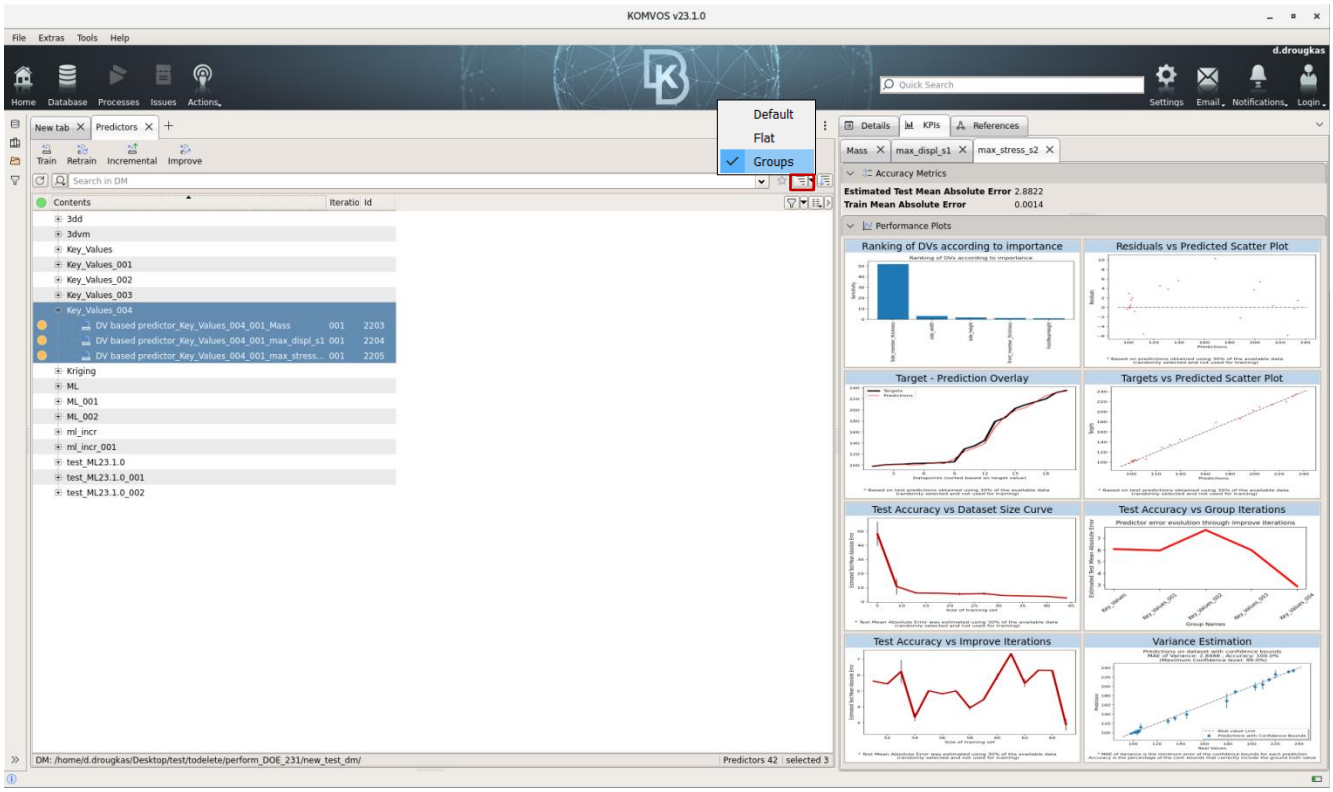
Design variable based Predictors are advanced data-driven machine learning models that can be used for any parametric study or exploration. DV based Predictors can be trained using as input the design variable values of a parametric model and its responses. These can be single scalar values (key values) 2d curves and 3d field results. Prediction of the responses is done through the DV Based prediction window, where values may be selected for the relative Design Variables, in order to define a “what if” scenario and predict the responses.

Responses for key values appear in a parallel coordinates chart, for ease of presentation and comparisons. 2D and 3D predictions create new Prediction Simulation Runs in a new *Predictions* tab. The created report of type *MetaProject* contains the results that are presented in the META Viewer. In order to allow for training of 3D results on models with mesh modifications, when creating the models in ANSA, enable **Changeset Management** for Parts, from the *Tools>Settings>Settings>Model Browser>General* menu.

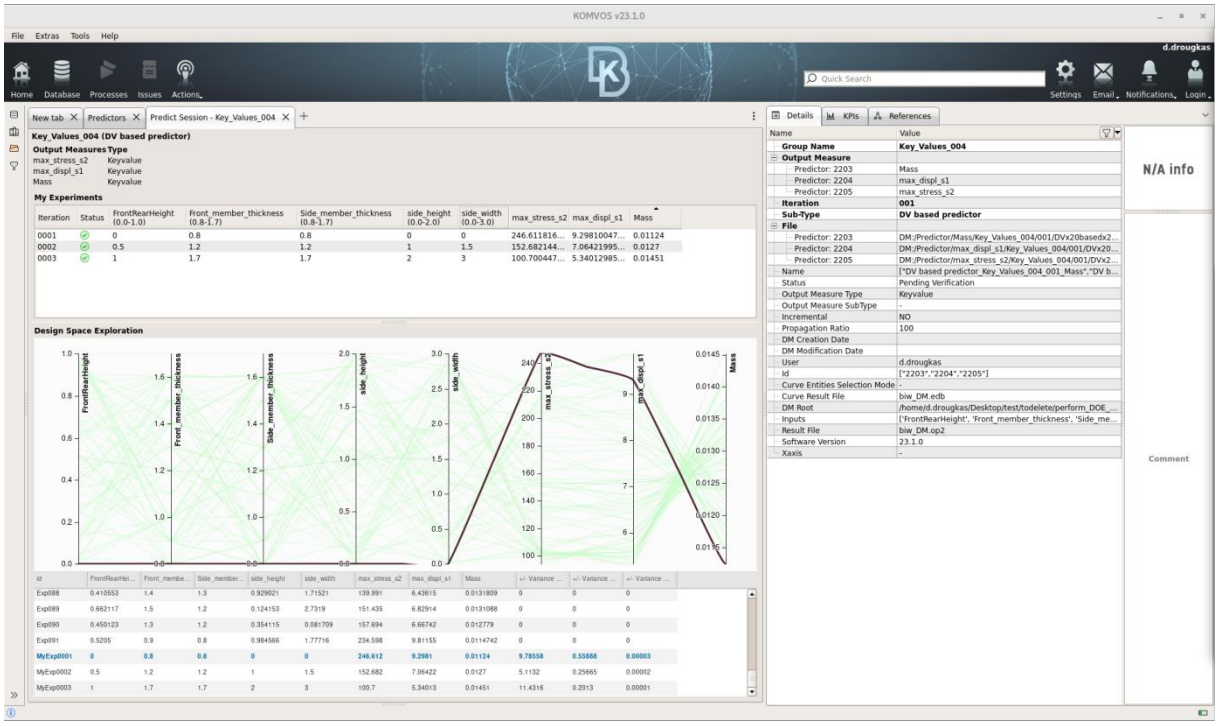
Once the training process is finished, a Predictor will appear in the relative tab.

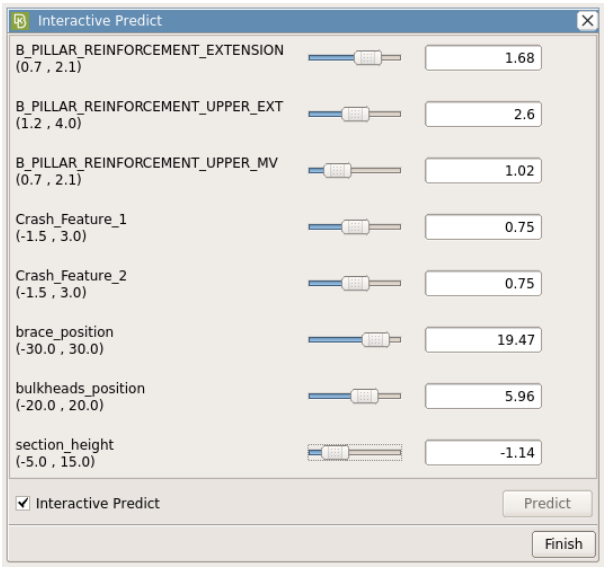
Predictors contain reports of performance (accuracy charts, mean absolute error values, etc.) and can be used in order to make predictions for the responses they have been trained for.

Through the option **Group Predict**, the predictions can be initiated and appear in the DV based prediction window, for the respective trained responses.



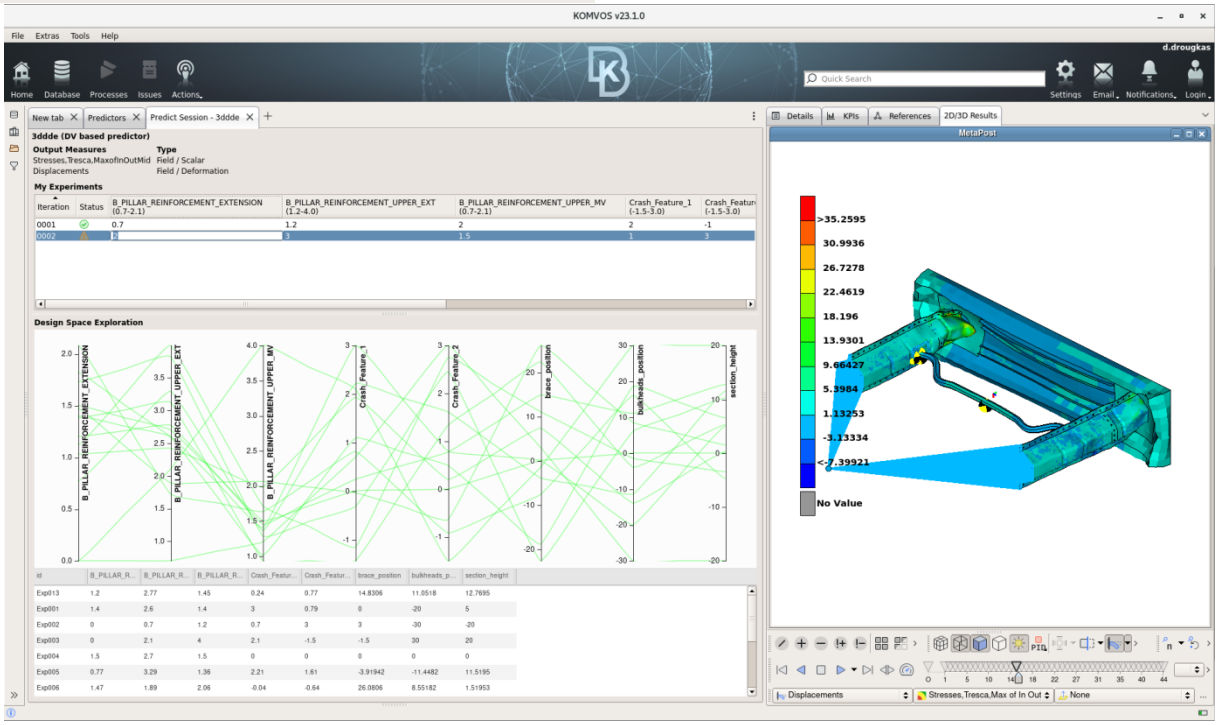
The Prediction can start by copy-pasting (Copy DV values) design variable values of existing Simulation Runs from the DM, by manually typing DV values in the *My Experiments* list and selecting **Predict** or by using the Interactive prediction option, that predicts live as the sliders are moved.



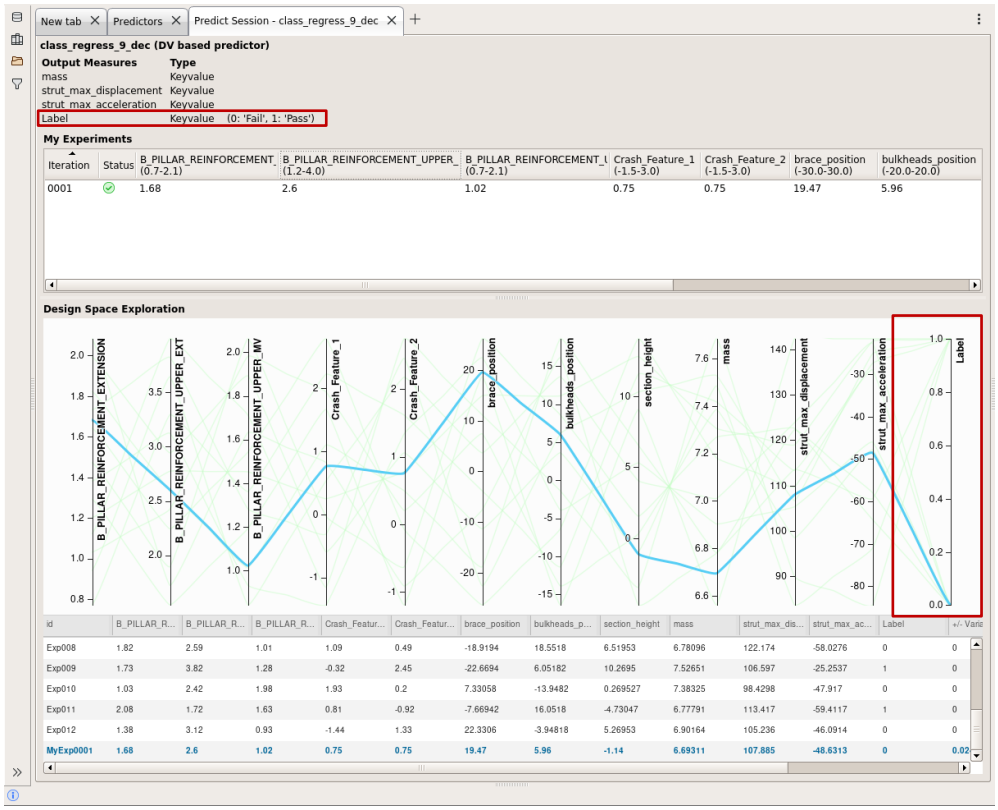


After the prediction is finished, a new Simulation Run can be created using the applied design variable values, as a validation of the prediction, using the Create New experiment in ANSA option.

This option opens the baseline model in ANSA and applies the DV values in the Optimization tool, ready to run and create a new Simulation Run in the DM.



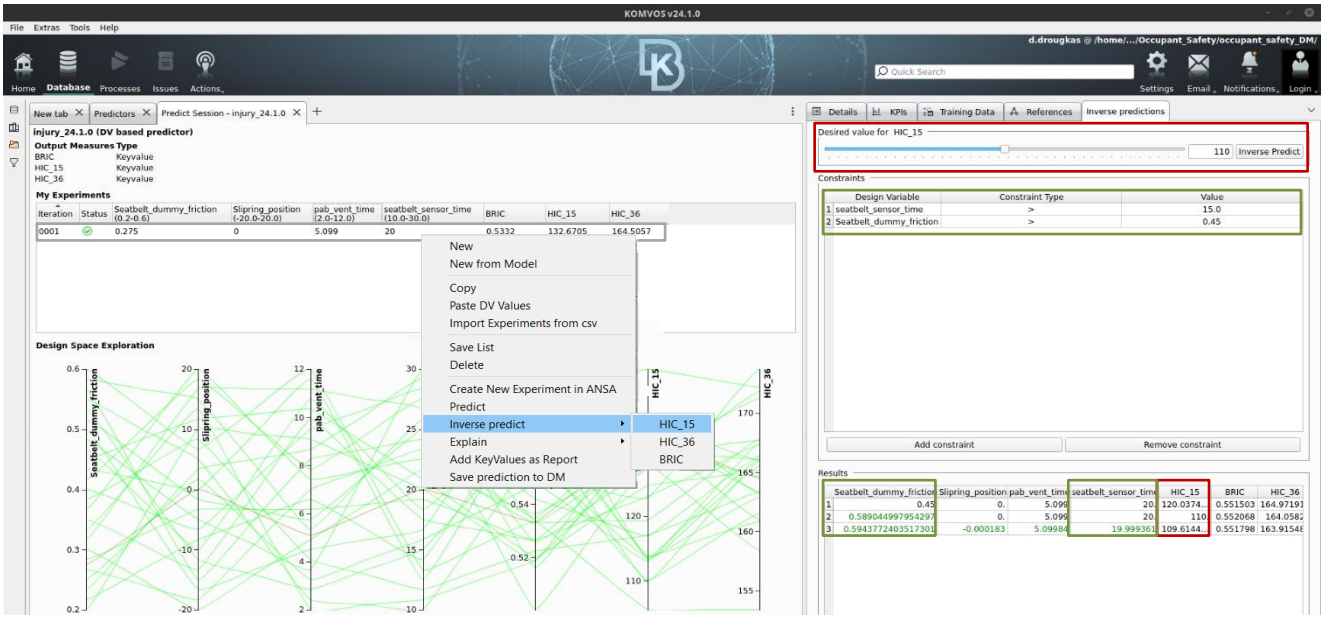
In case of a classifier prediction, the parallel coordinates chart is updated with a legend that explains which values in the chart column correspond to each class.



9.1.4.3. Inverse Prediction

Inverse Prediction returns the design variable values that could result in a specified response value. It is possible to define a target value for a response (a maximum stress or a HIC value for example) and the tool suggests a number of possible design variable input values that could achieve this target. Additionally Design Variables can be constrained to specific values, by adding Constraints. This allows for a more flexible design exploration.

There are multiple possible designs that could result in this response target and the tool will suggest designs that involve changing one design variable or all of them (changes seen in green).

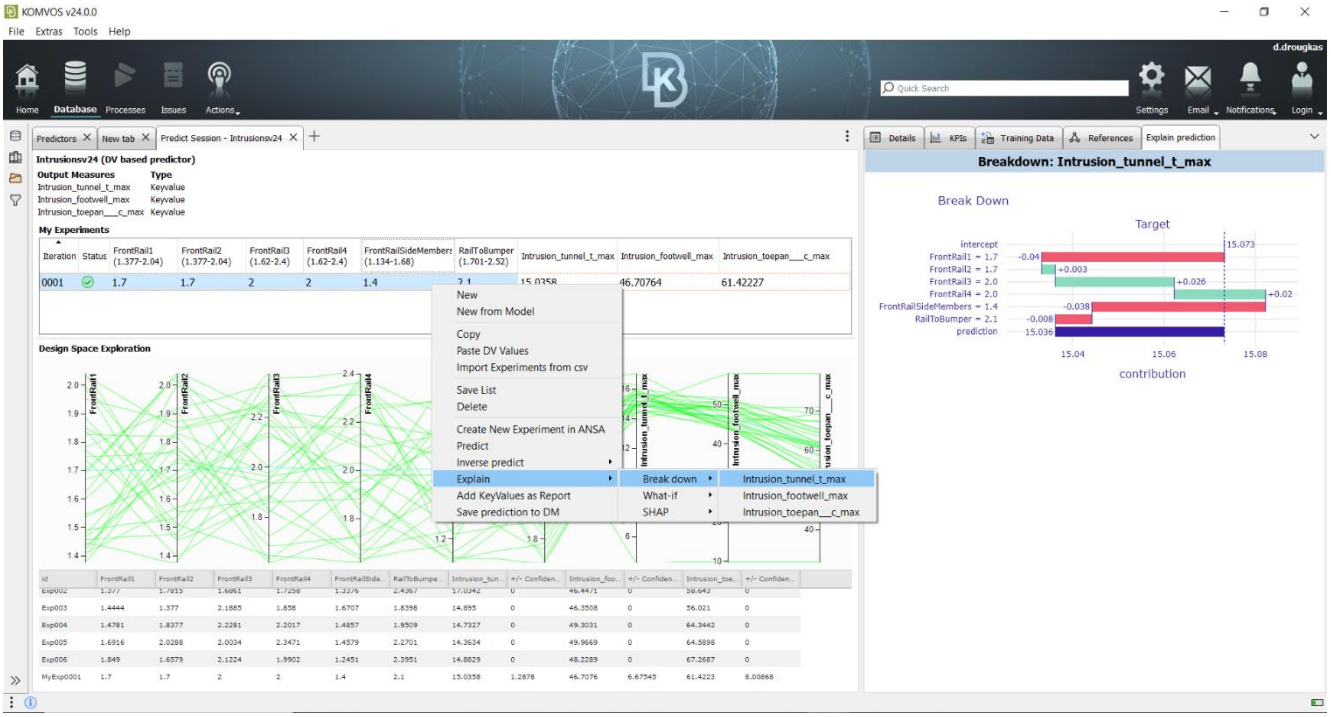


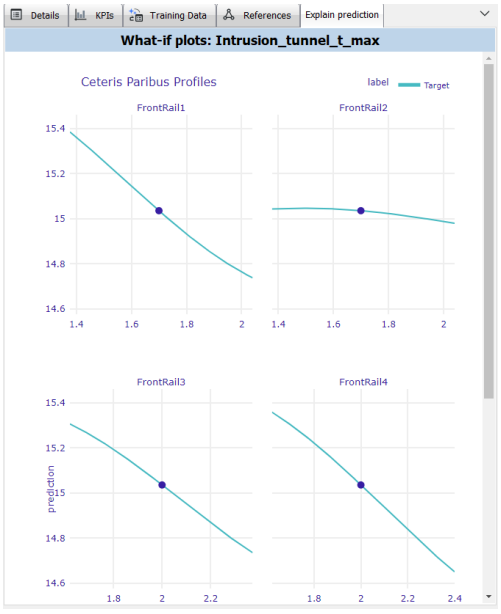
9.1.4.4. Explain

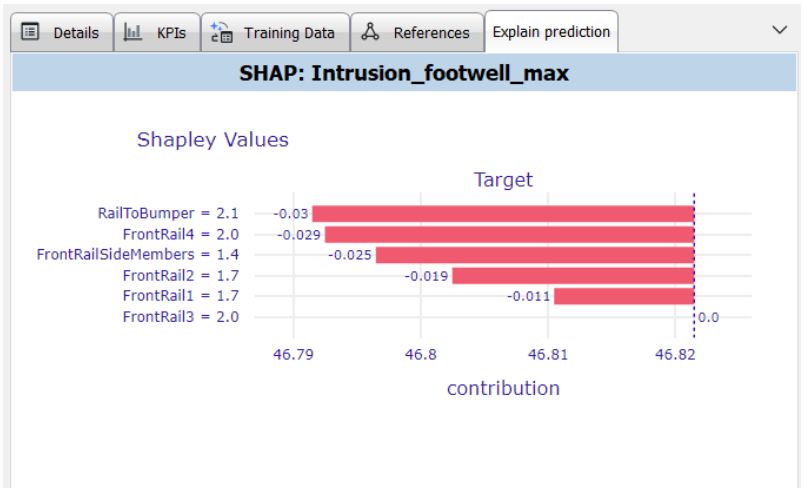
To improve on understanding why a ML model is predicting the values it does, explanation plots have been added on Key Value predictors, either as KPIs in the Model Explanation tab, or on demand after the model is created, providing more information on the behavior of such ML models.

Breakdown, *What-if* and *SHAP* plots are created on demand after a prediction was done.

Breakdown plots and SHAP plots express how much of a prediction can be attributed to each of the input values. This helps towards understanding the important variables for specific query points. The plots do not suggest that the effect of the design variables is additive and that they do not have any correlation, but is an indication of the effect each one has on the predicted response value. In the plots, the dashed line corresponds to the mean value of the response, and the effect of each design variable is added or subtracted in order to result to the final prediction.





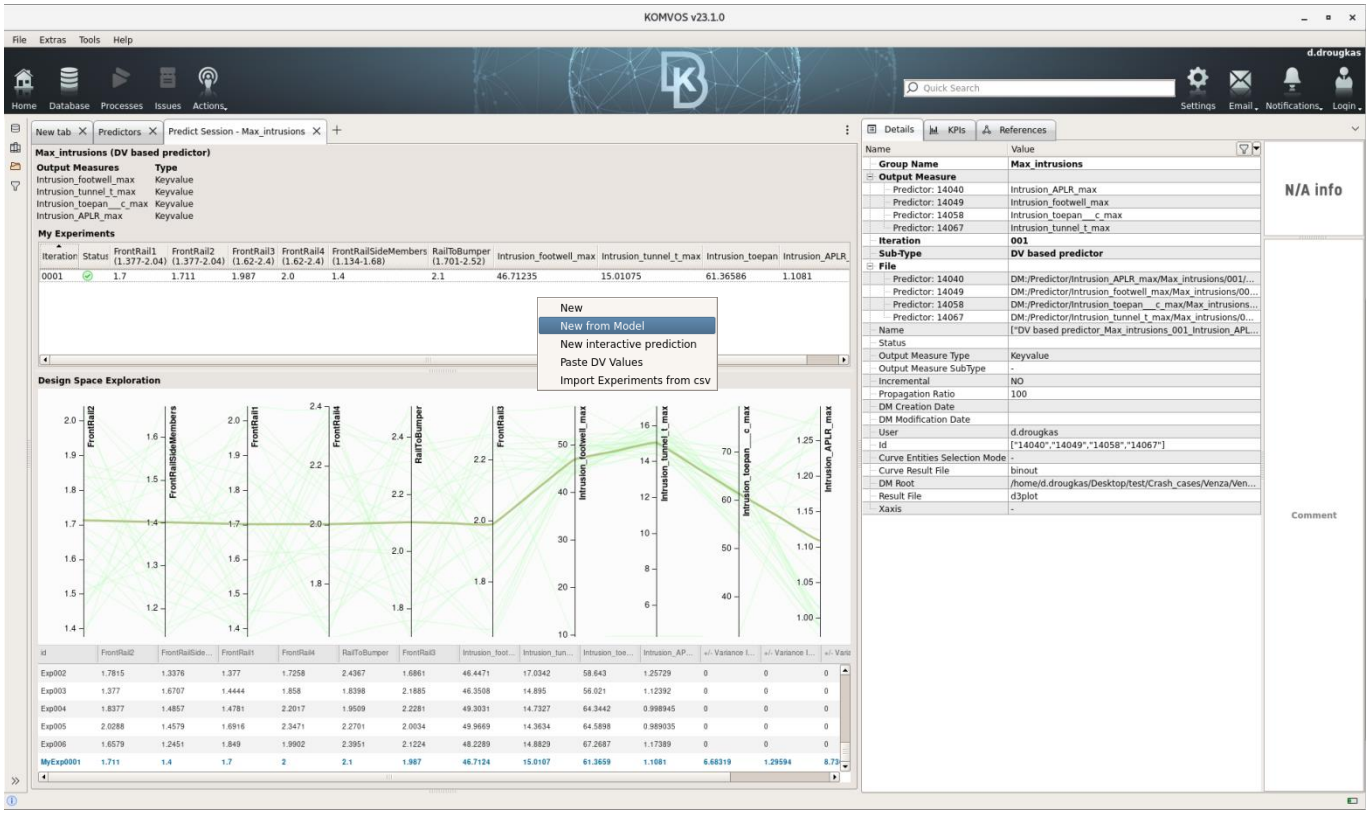


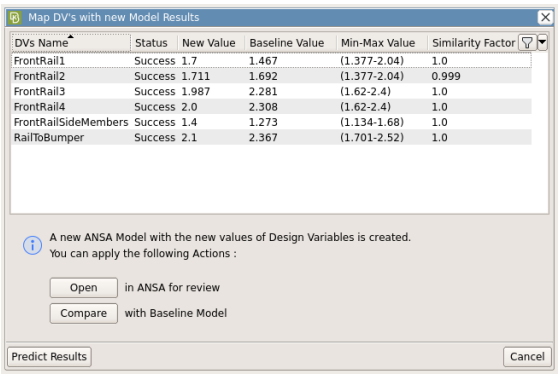
What-if plots show how the prediction for a query point would change, if we changed each input across the range of values. A plot for each input is created.

9.1.4.5. DV Based Prediction for a new FE model

Design Variable based prediction can be performed for FE models similar to the FE model that was used for the predictor training. If there is an existing predictor, it is possible to input a new FE model (similar[1] to the FE with which the predictor was trained), in order to make predictions.

The new FE model is selected with the **New from Model** option and a Design Variable mapping process takes place, mapping the Design Variables from the original, to the new FE model.



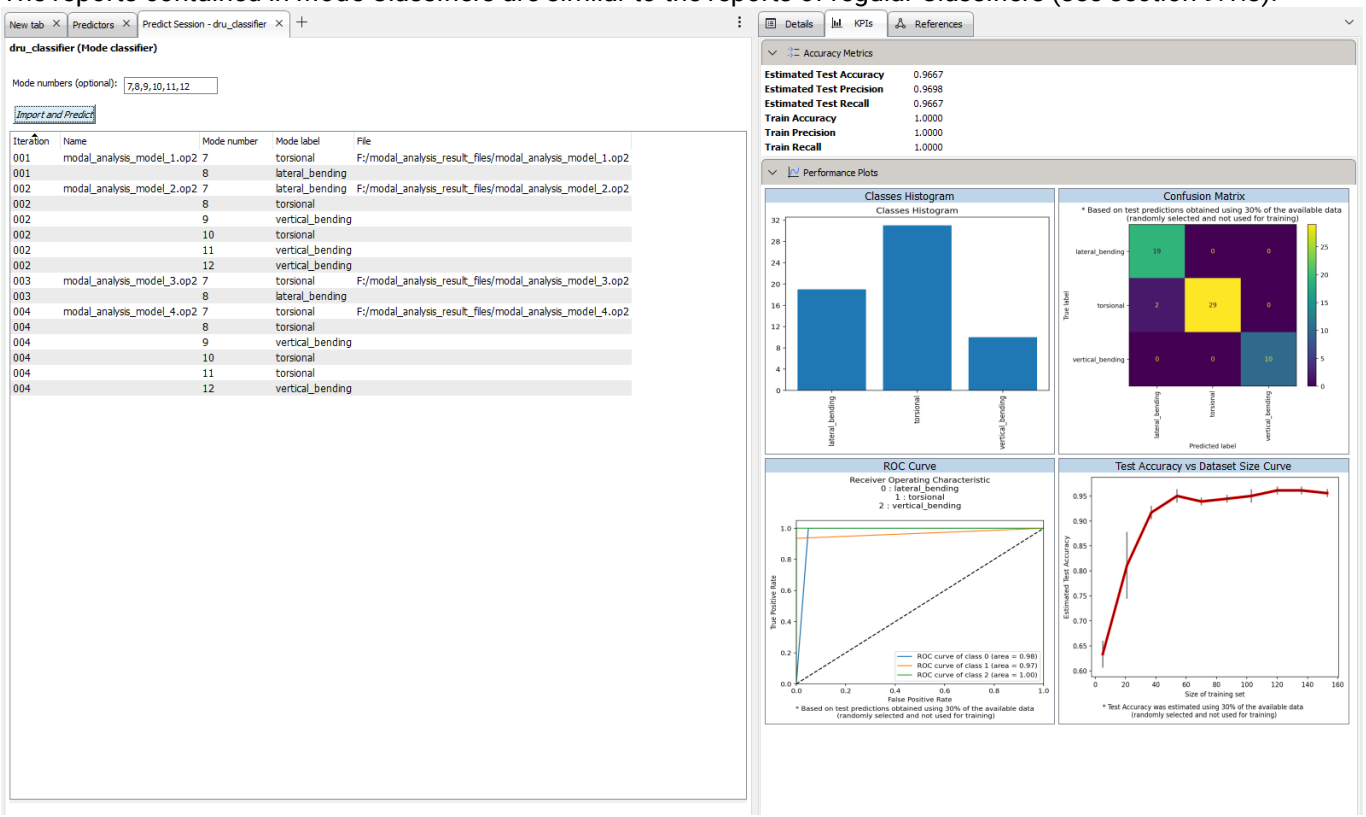


Once the mapping is done, if it was successful, the Similarity Factor will be close or equal to 1 and a prediction can be done for the new FE model.

[1] *A similar model could be a model with similar geometry but different mesh, prepared for a different type of analysis.*

9.1.4.6. Mode Classifier

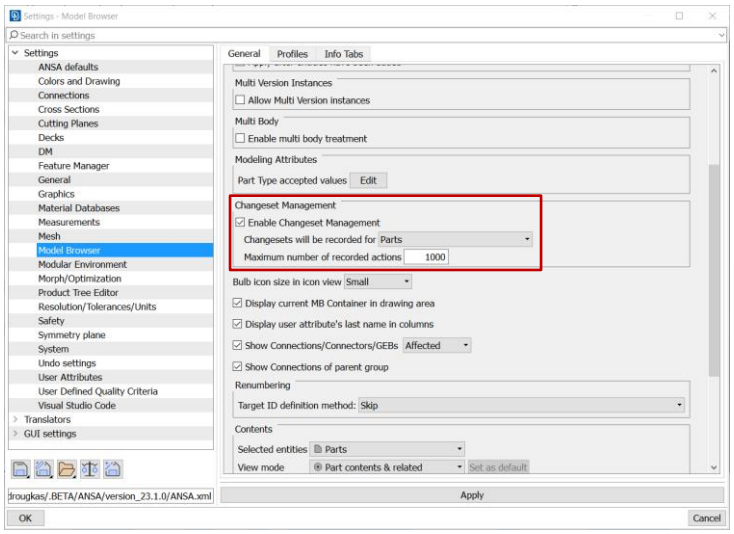
Mode Classifiers use the result files of FE normal modes analysis as input and perform feature extraction on those files. The features are designed to capture the shape of each mode with the help of the node displacements in order to be able to predict the type of the mode shape. For example, global mode types like Torsional, Vertical Bending and Lateral Bending can be used, but also local modes, e.g. on the roof or floor of the Body in White, as long as enough samples are provided.
No design variables are needed in order to train this type of Predictors so they can be used for cross model scenarios.
Data for mode classification can be imported in KOMVOS through the Import DOE action. (see section 9.1.6.5)
Once the training process is finished, a Predictor will appear in the relative tab.
The reports contained in Mode Classifiers are similar to the reports of regular Classifiers (see section 9.1.3).



For the Mode Classifiers, predictions can be performed for selected Simulation Runs existing in the DM, using the context menu option Predict, or through the **Import and Predict** button in a Predict Session, for external models. It must be noted, that the result files must be provided (e.g .op2) instead of the model files (.ansa, .nas, etc).
In case, only certain modes are of interest, these can be specified in a Predict Session, and predictions will be returned only for those modes, e.g 7,8,9.
If a file contains less modes, e.g only 8 modes, then the predictions of this file will be only for modes 7 and 8.

9.1.5. Dataset Generation

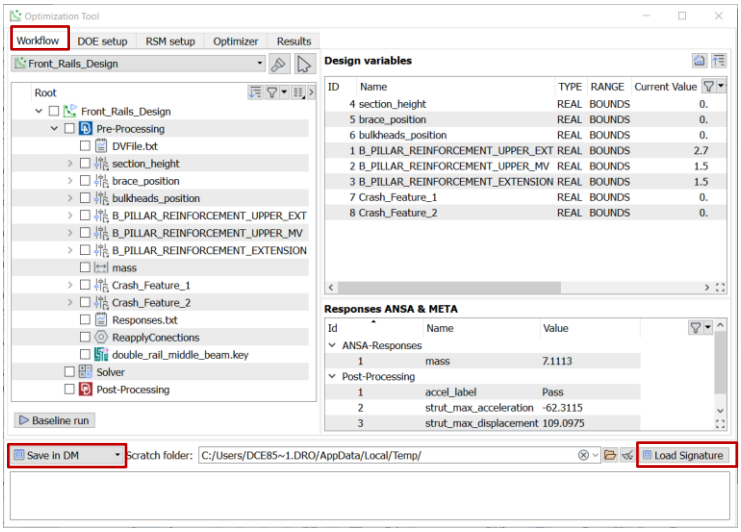
9.1.5.1. Model preparation in ANSA



In order to create the required training dataset, several Simulation Runs need to be created and added in a DM. This is done through the ANSA Optimization tool.

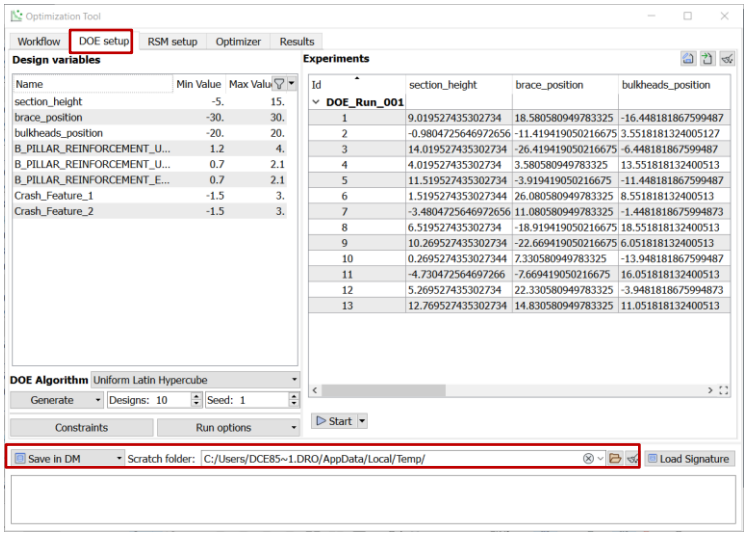
In order to allow for training of **3D results** on models with mesh modifications, enable **Changeset Management** for Parts, from the

Tools> Settings>Settings>Model Browser >General menu.

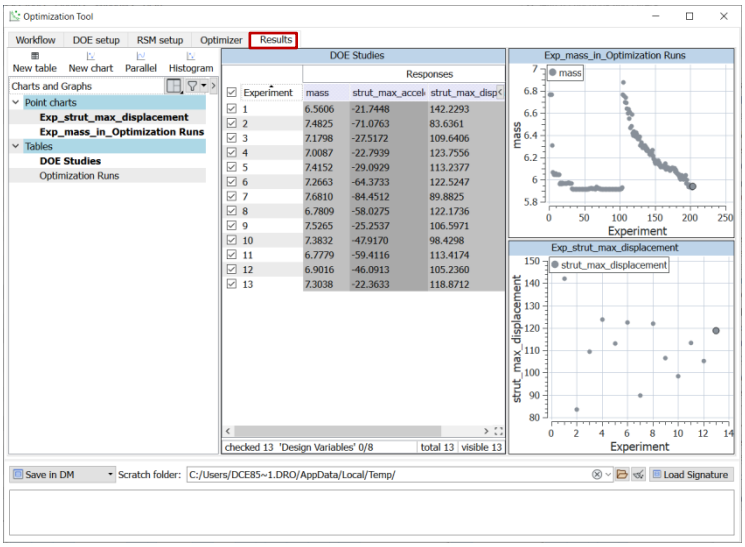


Once a DM path is defined, it is possible to define a complete Optimization task with Solver and post processing item, in the **Workflow** tab of the Optimization Tool.

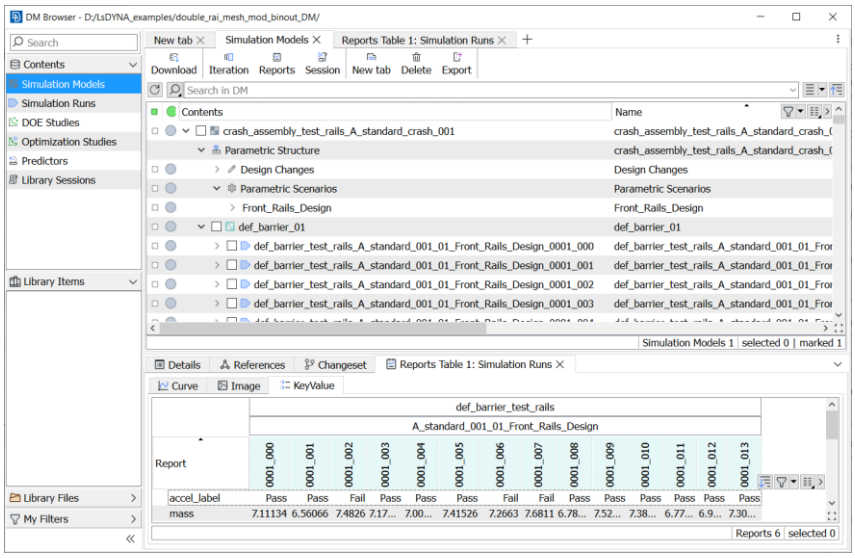
If there are existing Simulation Runs in the DM, the **Load Signature** button can specify which one will be loaded in the Optimization tool.



In the **DOE setup** tab, the option **Save in DM** means that when the DOE will start, it will automatically save a new Simulation model and as many Simulation Runs as are the defined experiments. Next to the Save in DM option, the Scratch folder is presented. The experiments will initially be created in the defined scratch folder and if they are successful, they will be copied in the DM. If any experiment fails, its respective files (output file, solver file, messages) will remain in the scratch directory for inspection.



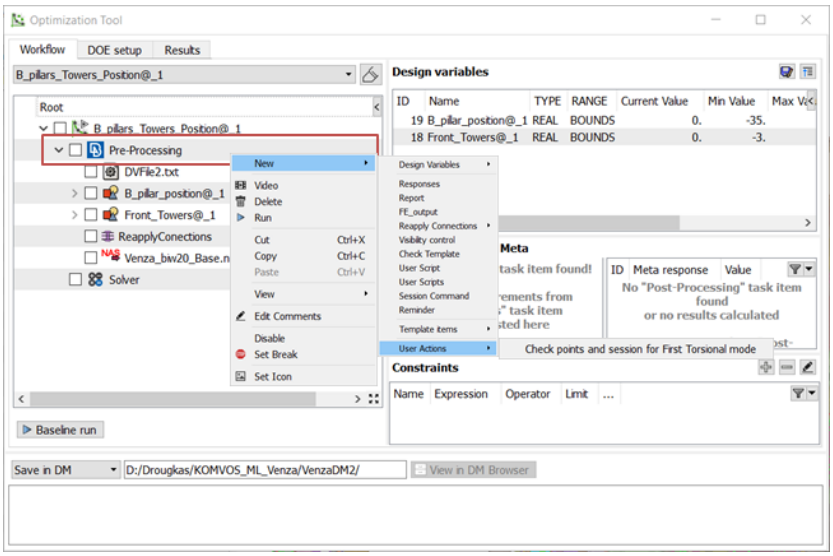
Once the DOE is completed, the results of each simulation can be seen and post processed in the **Results** tab of the optimization tool.



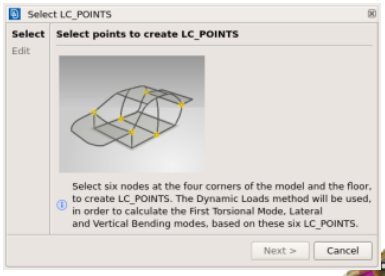
The Simulation Model and respective Simulation Runs and their information (attributes, reports, Design Variable information, etc.) are now saved in the DM and can be seen in the *DM Browser*.

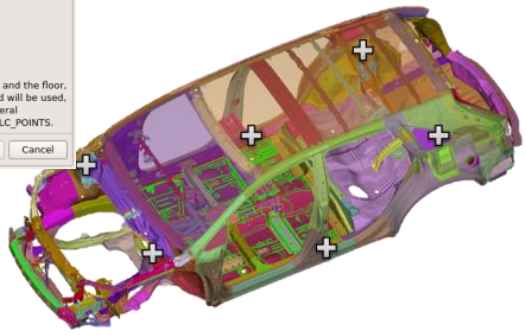
9.1.5.2. Preparation for first Torsional and Bending mode frequency values

In case a model needs to be prepared for Feature based Machine Learning, specific actions need to be taken in the preparation of the model. Specifically named Interface Points need to be created in the optimization task, which will pinpoint critical positions of the Body In white (BiW). These will be used by a dedicated post-processing session file, in order to identify the first Torsional and Bending mode frequency values.



A dedicated *User Action* is available that will check if the appropriate interface points are available. In case they are not defined, a wizard will appear, in order to complete the selection of the nodes where the interface points will be created.



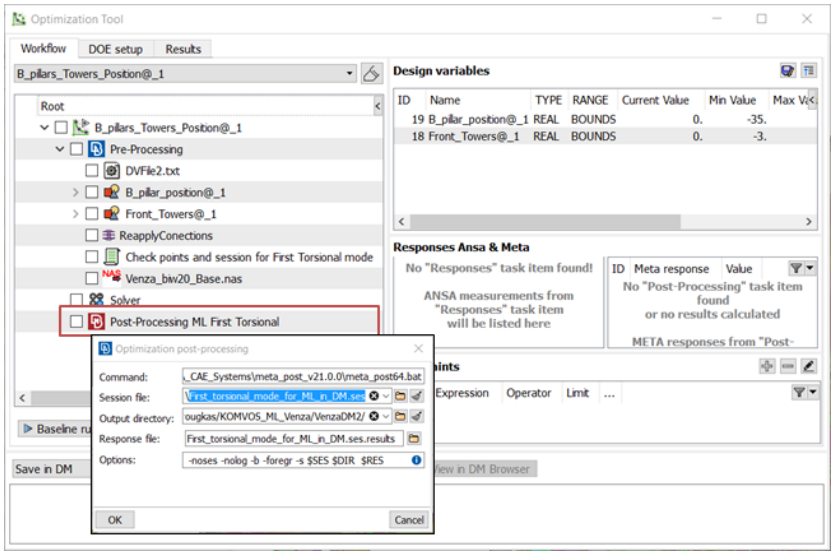


The 6 interface points need to be on the strut tower points and on stiff areas closer to the middle of the BiW. Interface points on panel areas are not recommended. The created interface points will be named according to their position.

F_LHS: Front left-hand side,

M_RHS: Middle right hand side, etc.

If the given names do not match the position, they should be renamed accordingly.

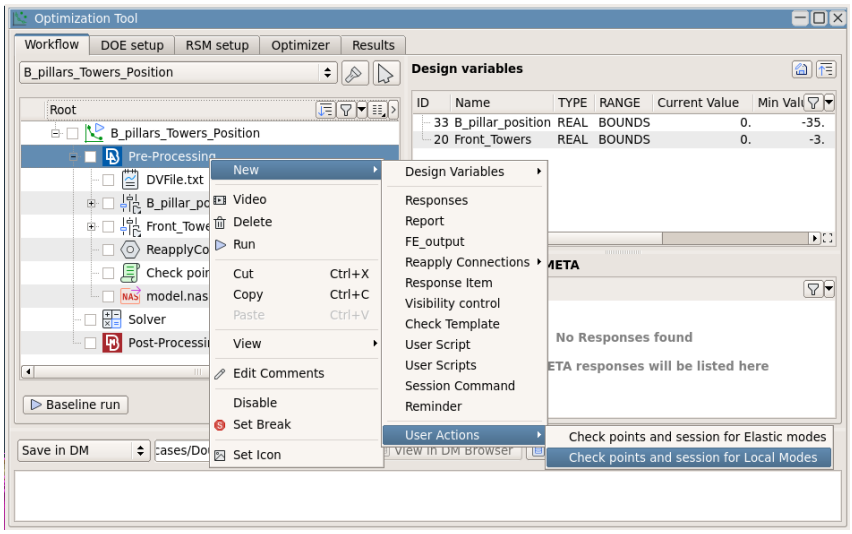


Once the interface points are created, a new post processing item will be created, using a specific post processing session that will produce the needed frequency value responses.

If 4 interface points are used at the corners of the BiW, they are enough to get the First Torsional mode value response. With the additional two middle points, first Lateral and Vertical Bending modes frequency values can be identified.

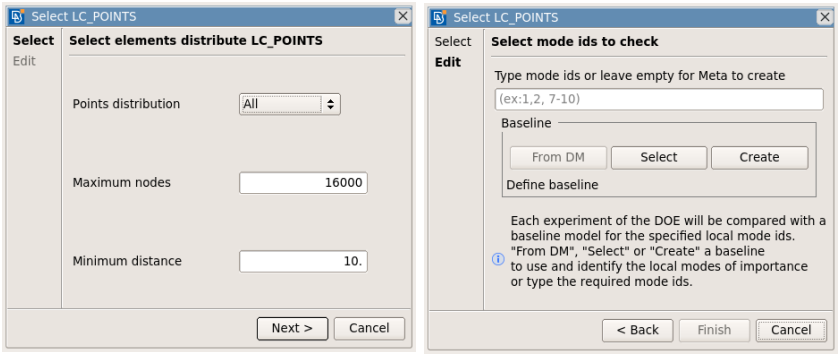
9.1.5.3. Preparation for local modes identification

Local Modes of a Body in White can be identified through a specific process in order to use Feature based Machine Learning. A new wizard in the Optimization task prepares the model and a specific post-processing session is able to identify the local modes and create responses. These can be used for training machine learning models.



A dedicated *User Action* is available that will check if the appropriate interface points are available. In case they are not defined, a wizard will appear, in order to complete the automatic definition of interface points.

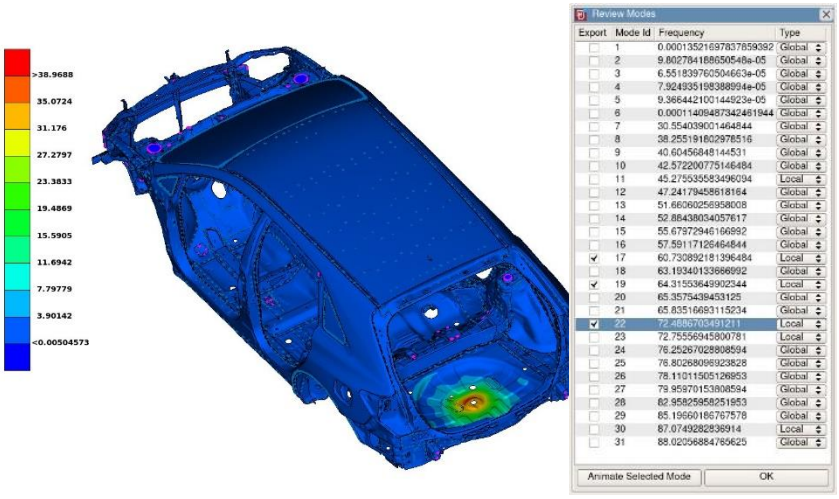
Select **Run** in the context menu to check if the model is prepared, or to start the wizard if it is not.



The maximum number of nodes will automatically be defined. The **Minimum distance** refers to the distance between interface points.

In the next step, the wizard requires a directory where **Baseline** results of the model exist. This will be used in order to open META for the baseline local modes selection.

If baseline results exist in the current DM select **From DM**. If results exist in a different directory, use **Select**. If results are not available, **Create** will start a new analysis when **Finish** is pressed (FE output and Solver items need to be defined in the Optimization task).



Once the desired local modes are selected, when META is closed, the interface points will be created, and also the dedicated post-processing item will be defined, with a session specific for identifying the selected local modes.

Finally, the local modes task needs to be **Reset**, using the respective context menu option, and the task is ready.

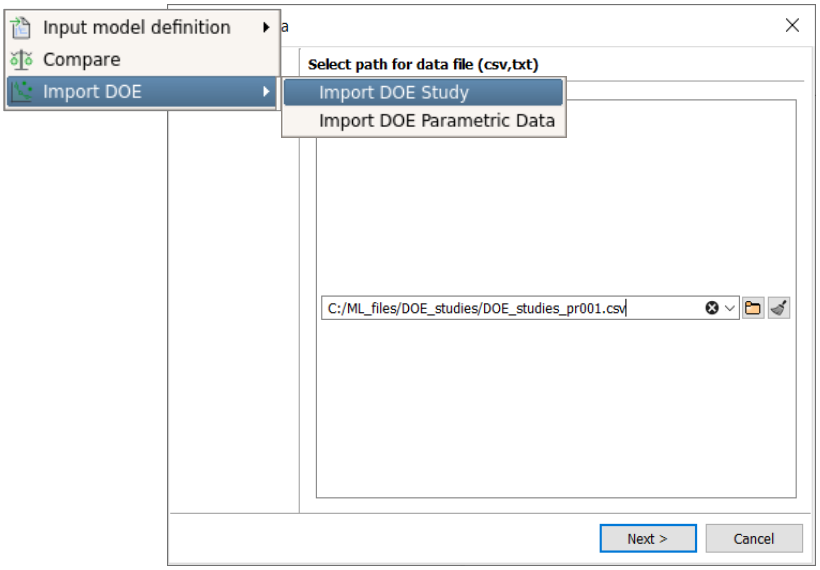
9.1.5.4. Preparation for Mode Classification

Training Mode Classifiers requires the result files of Normal modes analysis. Supported solvers are Nastran, Abaqus, Ansys and Pamcrash. No specific preprocessing process needs to take place in ANSA. Existing models with their normal modes results can be imported in a DM using KOMVOS' Import DOE action along with the mode shape type definition (see section 9.1.5.5.1). A special characterization of these "labels" will create the required Report items (Table report type), that will be used during training of Mode Classifiers. Alternatively for εχιστινγ simulation runs in a DM, a *.csv file can be used with specific format, in order to assign mode-shape type labels to the simulation runs.

9.1.5.5. Import DOE

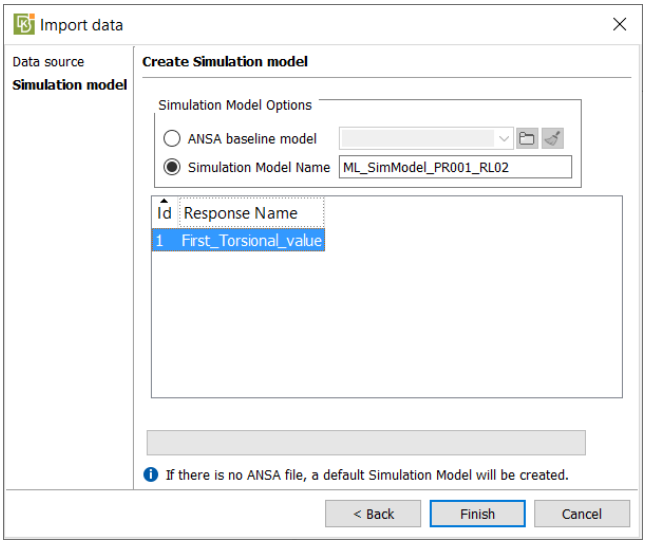
Machine Learning in KOMVOS can be performed even in cases where the "data set" was not created using ANSA's Optimization tool. Using the **Import DOE** action, it is possible to import experiments as Simulation Runs in a new DM and perform Machine Learning actions.

9.1.5.5.1. Import DOE Study



In case past DOE are available and the Feature Based Machine Learning will be used, it is possible to use the **Import DOE Study**, to store the data in the DM. This way Feature based Machine Learning will be possible.

Using the **Import DOE >Study** option from the **Actions** button in the toolbar, already executed DOE studies that are stored in a *.txt or *.csv file can be imported into the current DM System.



If there is no header in the *.txt or *.csv file, as seen bellow, the names in the *Responses name* column, a response name needs to be added, that corresponds to the value in the respective column after the file path. If there is a header, the responses names will be automatically filled.

If an ANSA file is available and the DOE was performed in ANSA, the design variable information can also be acquired.

The information will be stored automatically in the appropriate format under a Simulation Model, so that it can be used by the Machine Learning functionality. If result files exist in the same directory as the solver files

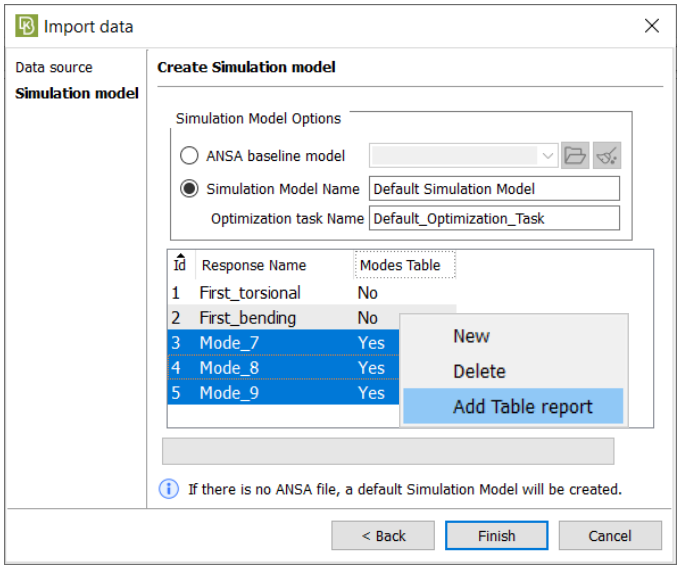
(*.nas, *.key, etc.) the results will also be added in the DM under each Simulation run, in order to be used for training of 3D predictors.

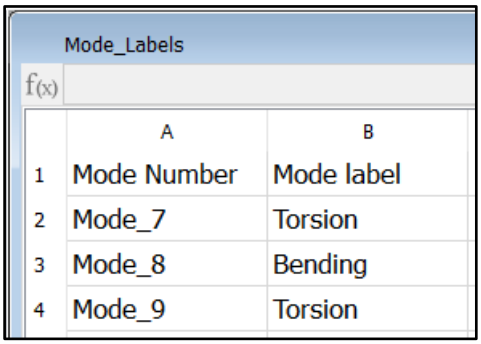
```
;/Neon/Neon_import_data/Neon_Parametric1.nas,25.1
;/Neon/Neon_import_data/Neon_Parametric2.nas,35.8
;/Neon/Neon_import_data/Neon_Parametric3.nas,40.0
;/Neon/Neon_import_data/Neon_Parametric4.nas,38.2
;/Neon/Neon_import_data/Neon_Parametric5.nas,24.2
;/Neon/Neon_import_data/Neon_Parametric6.nas,45.2
;/Neon/Neon_import_data/Neon_Parametric7.nas,52.0
;/Neon/Neon_import_data/Neon_Parametric8.nas,37.4
;/Neon/Neon_import_data/Neon_Parametric9.nas,40.8
;/Neon/Neon_import_data/Neon_Parametric10.nas,41.5
```

Additionally, if data for mode shape type classification exist, these need to be added as Table reports. This can be done by selecting the specific responses and selecting the **Add Table Report** option.

```
File,First_torsional,First_bending,Mode_7, Mode_8, Mode_9
D:\Komvos_ML\ImportDoeStudies\venza_data\Venza_biw20_Base1.nas,25.1,30.1,Torsion,Bending,Torsion
D:\Komvos_ML\ImportDoeStudies\venza_data\Venza_biw20_Base2.nas,35.8,31,Torsion,Torsion,Bending
D:\Komvos_ML\ImportDoeStudies\venza_data\Venza_biw20_Base3.nas,40.0,34,Torsion,Bending,Torsion
D:\Komvos_ML\ImportDoeStudies\venza_data\Venza_biw20_Base4.nas,38.2,32,Bending,Torsion,Torsion
D:\Komvos_ML\ImportDoeStudies\venza_data\Venza_biw20_Base5.nas,24.2,31,Torsion,Bending,Bending
D:\Komvos_ML\ImportDoeStudies\venza_data\Venza_biw20_Base6.nas,45.2,34,Bending,Torsion,Bending
D:\Komvos_ML\ImportDoeStudies\venza_data\Venza_biw20_Base7.nas,52.0,35,Torsion,Bending,Torsion
D:\Komvos_ML\ImportDoeStudies\venza_data\Venza_biw20_Base8.nas,37.4,35,Torsion,Torsion,Bending
D:\Komvos_ML\ImportDoeStudies\venza_data\Venza_biw20_Base9.nas,40.8,36,Bending,Torsion,Bending
D:\Komvos_ML\ImportDoeStudies\venza_data\Venza_biw20_Base10.nas,41.5,37,Torsion,Bending,Torsion
```

This option will create Table report items for each Simulation run that are required in order to perform training for Mode Classifiers.





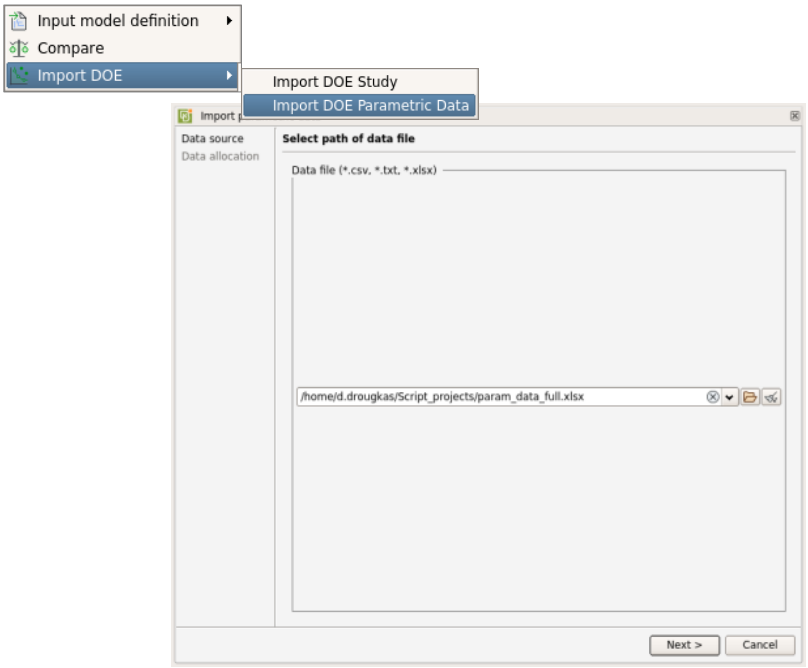
```
Sim Run Id, Mode number, Mode label
3, 7, torsional
3, 8, lateral_bending
6, 7, torsional
6, 8, vertical_bending
8, 7, torsional
8, 8, vertical_bending
10, 7, torsional
10, 8, vertical_bending
12, 7, torsional
12, 8, vertical_bending
14, 7, torsional
14, 8, vertical_bending
16, 7, torsional
16, 8, vertical_bending
18, 7, torsional
```

Alternatively if simulation runs in the DM do not have Mode labels Report items, the labeling information can be added during training through a *.csv file that will be requested by the training wizard. This csv file contains the mode labels information for each simulation run and it should have the following structure

Simulation Run id, Mode number, Mode label

9.1.5.5.2. Import DOE Parametric DATA

In case past DOE are available and the parameterization was not done in ANSA, it is possible to use the **Import DOE Parametric Data** option, to store the data in the DM. This way Design Variable based Machine Learning will be possible.

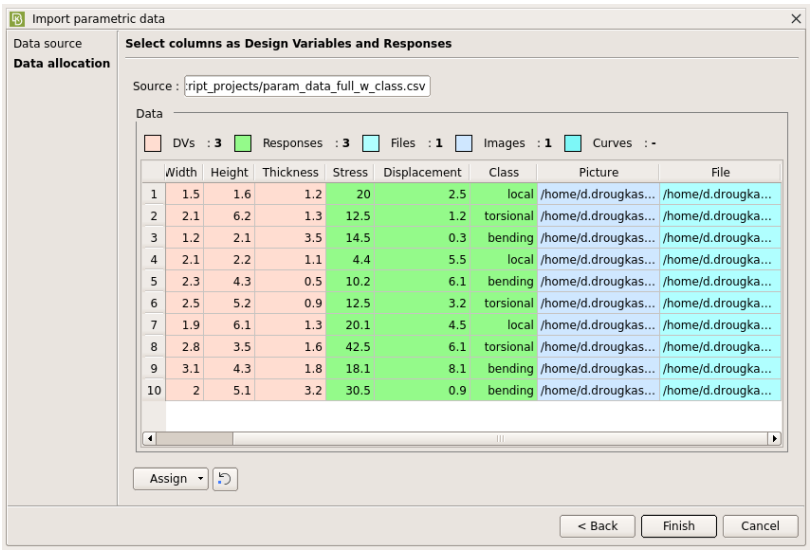


Using the **Import DOE > Import DOE Parametric Data** option from the **Actions** button in the toolbar, it is possible to import already existing parametric data from a spreadsheet (*.xlsx, *.csv) file.

	A	B	C	D	E	F	G	H	I
1	Exp id	Width	Height	Thickness	Stress	Displacement	Class	Picture	File
2	1	1.5	1.6	1.2	20	2.5	local	.../Image_001.png	.../biw1.nas
3	2	2.1	6.2	1.3	12.5	1.2	torsional	.../Image_002.png	.../biw2.nas
4	3	1.2	2.1	3.5	14.5	0.3	bending	.../Image_003.png	.../biw3.nas
5	4	2.1	2.2	1.1	4.4	5.5	local	.../Image_004.png	.../biw4.nas
6	5	2.3	4.3	0.5	10.2	6.1	bending	.../Image_005.png	.../biw5.nas
7	6	2.5	5.2	0.9	12.5	3.2	torsional	.../Image_006.png	../biw6.nas
8	7	1.9	6.1	1.3	20.1	4.5	local	.../Image_007.png	.../biw7.nas
9	8	2.8	3.5	1.6	42.5	6.1	torsional	.../Image_008.png	.../biw8.nas
10	9	3.1	4.3	1.8	18.1	8.1	bending	.../Image_009.png	.../biw9.nas
11	10	2	5.1	3.2	30.5	0.9	bending	.../Image_010.png	.../biw10.nas

The rows and columns of the column formatted file will be read and displayed in the data window.

Each column can be assigned as Design variable, Response, Response curve, Image or File.



	Width	Height	Thickness	Stress	Displacement	Class	Picture	File
1	1.5	1.6	1.2	20	2.5	local	/home/d.drougkas...	/home/d.drougka...
2	2.1	6.2	1.3	12.5	1.2	torsional	/home/d.drougkas...	/home/d.drougka...
3	1.2	2.1	3.5	14.5	0.3	bending	/home/d.drougkas...	/home/d.drougka...
4	2.1	2.2	1.1	4.4	5.5	local	/home/d.drougkas...	/home/d.drougka...
5	2.3	4.3	0.5	10.2	6.1	bending	/home/d.drougkas...	/home/d.drougka...
6	2.5	5.2	0.9	12.5	3.2	torsional	/home/d.drougkas...	/home/d.drougka...
7	1.9	6.1	1.3	20.1	4.5	local	/home/d.drougkas...	/home/d.drougka...
8	2.8	3.5	1.6	42.5	6.1	torsional	/home/d.drougkas...	/home/d.drougka...
9	3.1	4.3	1.8	18.1	8.1	bending	/home/d.drougkas...	/home/d.drougka...
10	2	5.1	3.2	30.5	0.9	bending	/home/d.drougkas...	/home/d.drougka...

If solver result files exist in the same directory as the solver files (*.nas, *.key, etc.) the results will also be added in the DM under each Simulation run, in order to be used for training of 3D predictors.

The data will be added in the DM structure and will be available to be used for Design Variable based Machine Learning.

NOTE!: The first column of the file needs to contain the ids of the experiments. The first row of the file is the header and needs to contain the type of value it contains (names of DVs or responses, pictures, files).

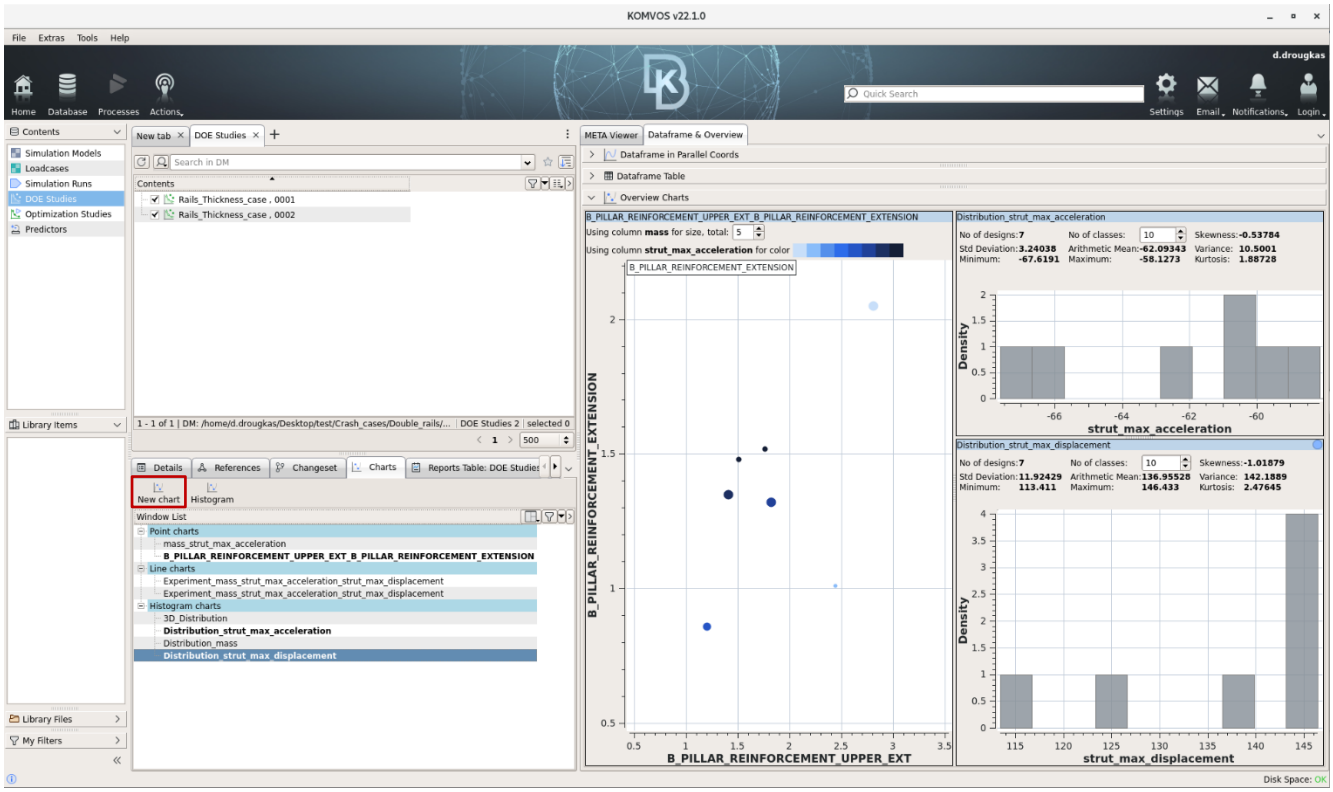
9.1.5.6. DOE Studies

The DOE Study entity contains DOE data existing in the DM. Such entities are automatically created when a DOE Study is created from ANSA and saved in the DM. The DOE Studies list hosts the DOE Study entities existing in the DM.

DOE Study entities can be created for previous versions through the Simulation Runs list (DOE View), using the Create DOE_Study action on a DOE Iteration.

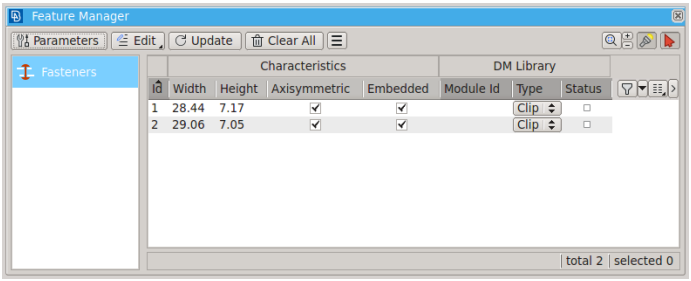
Through the **Dataframe & Overview** tab it is possible to explore the experiments' Design Variable values and Responses. Filtering can take place in the Parallel Coordinates plot in order to distinguish desired experiments.

Filtered and checked experiments can be added in a new Chart (line, point, histogram, etc.) using the respective button in the **Charts** tab for post processing.



A reports table can be generated for checked experiments (in *Dataframe Table* right click> **Reports Table**), to show the reports for each one in the respective **Reports Table** tab that is created.

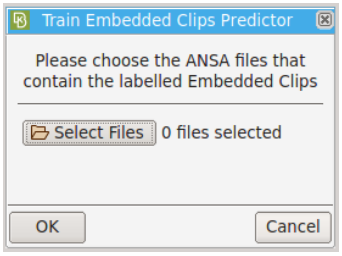
9.2. Train Embedded Clips Predictor



KOMVOS can be used to train an Embedded Clips Predictor, which can learn to recognize the Clips provided as input to the functionality. The input data need to be under a specific format. Then, this predictor can be used within ANSA for the recognition of similar clips in new databases.

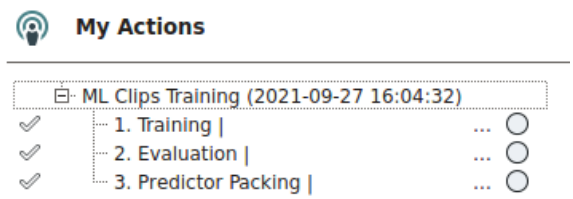
This functionality can be utilized through SPDRM as long as KOMVOS has been connected to an SPDRM back-end. This way, the training can take place on remote resources, depending on the SPDRM configuration. Otherwise, the *(Local)* option will be available and the training will be performed in the local machine. There is a minimum requirement of a 6GB GPU with CUDA 11.0.

Use the **Machine Learning > Train Embedded Clips Predictor** option from the **Actions** button in the toolbar. The user is prompt to select the files that will be used for the predictor's training.



NOTE!: A working directory must first have been specified under **Settings > Machine Learning Options > Working Directory**. This directory will be used for the output of the *tmp* files. The directory should have at least 10 GB available and (fast) read/write access. Setting the working directory is required only for local training.

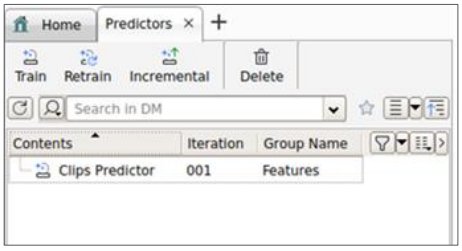
NOTE!: A path for an ANSA executable must be specified under **Settings > BETA apps > ANSA > Path**.



After the files have been selected, the progress of the training can be monitored through the *My Actions* shortcut in the *Home* workspace.

NOTE!: An important detail is the format of the files that will be used for training. They should be ANSA databases containing **Fastener Feature Entities**. These entities can been defined and handled via **Feature Manager** :

- One Feature Entity must exist for each Clip.
- The entire geometry of the parts where Clips are embedded on should exist in the database.
- Feature Entities should exist for all Clips, even for geometrically similar ones.



When the training is finished, the Clips Predictor will automatically be added in the connected DM and can be directly used in ANSA.

10. Configuration

KOMVOS is highly configurable in terms of both user-interface and core data management functionality. Depending on case, configuration can be made centrally, affecting all users, and on user level.

Central configuration of KOMVOS is done in its “home” directory, which is the directory indicated by the environment variable `SDM_CONSOLE_HOME`. By default, this environment variable is set in the KOMVOS start script and points to the **config** folder in the installation directory. If this environment variable has some value already before launching KOMVOS, that value prevails.

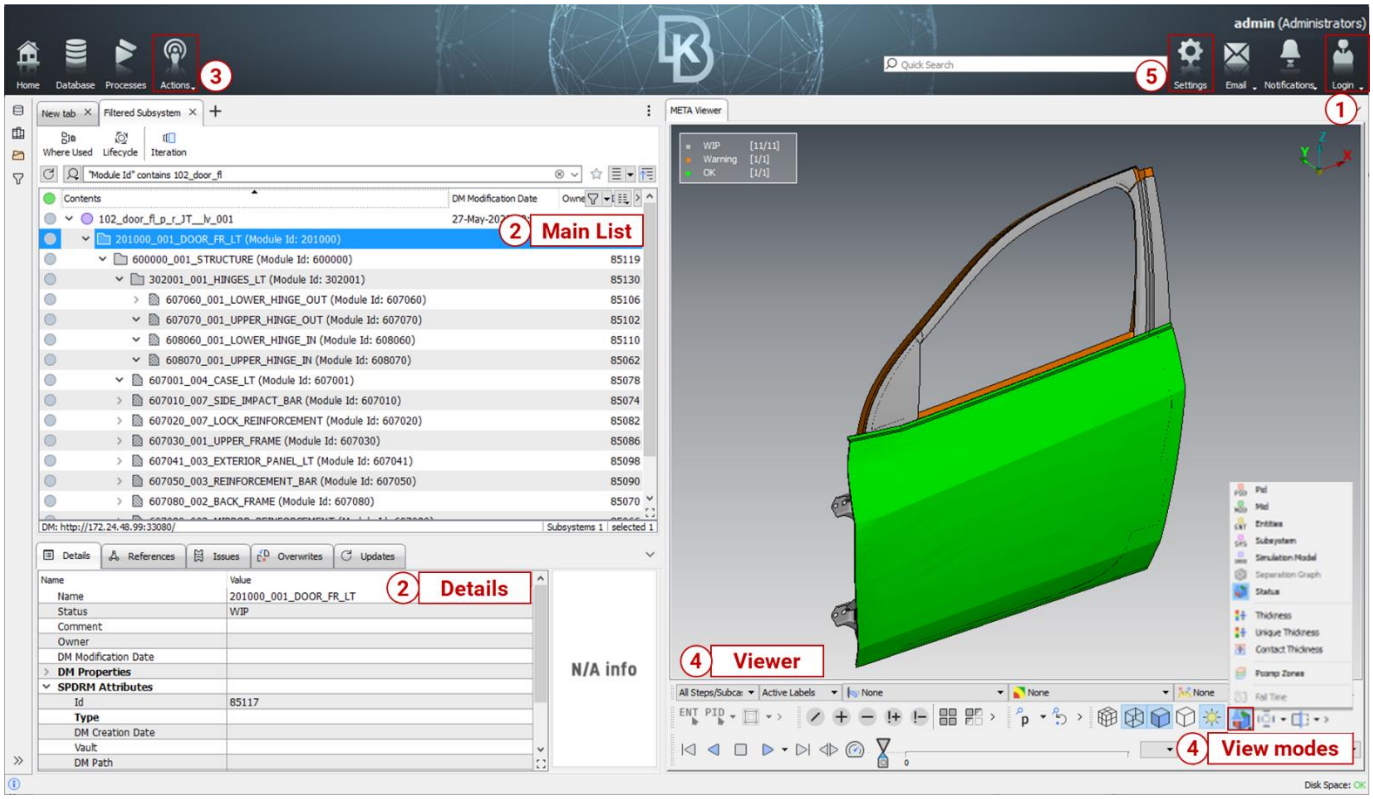
If the `SDM_CONSOLE_HOME` directory contains a folder named **SPDRM**, then this folder will be used as source for all configuration files when KOMVOS is connected to an SPDRM back-end. If this folder does not exist, the source for all configuration files will be the `SDM_CONSOLE_HOME` even in case of SPDRM back-end.

User configurations are stored in the user’s home folder under:

```
.BETA/KOMVOS/version_XX.X.X
```

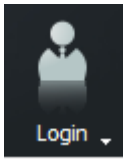
The different configuration possibilities of KOMVOS are listed below.

1. **SDM System**: See paragraph 10.1
2. **Database** workspace: See paragraphs 10.2, 10.6.11 and 10.7
3. **Actions**: See paragraph 10.3
4. **Viewer**: See paragraph 10.4
5. **Settings**: See paragraph 10.6

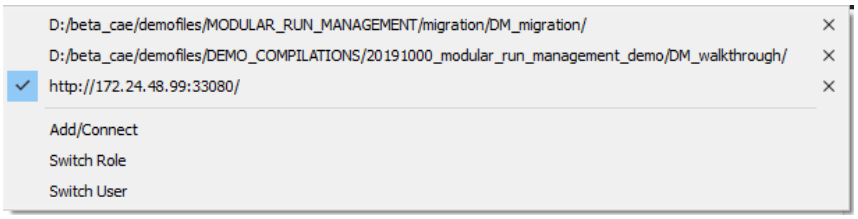


10.1. SDM System

In order to get started with KOMVOS, the first thing to be configured is the SDM back-end that will be used. This is done either with the command line argument `-dmroot` or through the **Login** option in the main toolbar.

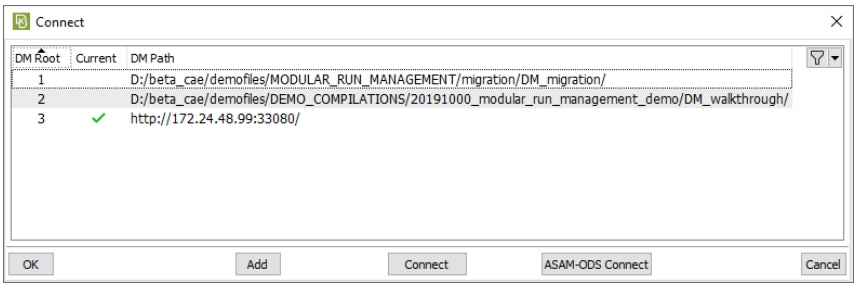


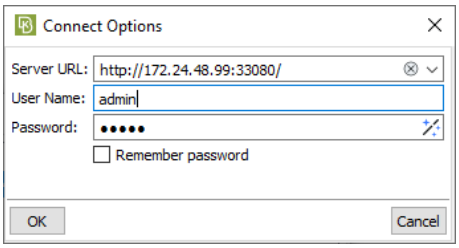
The **Login** button triggers a pop-up menu that provides login settings related to the last SDM systems used and the currently used SDM back-end.



In the example on the left, this menu shows the SDM back-ends used last, with 2 of them being file-based DMs and one being an SPDRM server.
The SPDRM server is the current DM, as it's marked with the tick mark on the left.

The **Add/Connect/ASAM-ODS Connect** options can be used to connect to a new DM through the **Connect** dialog:



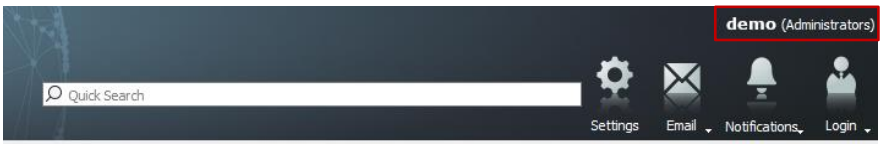


Information on the authentication of KOMVOS to server-based back-ends can be found in Chapter 7 of the Data Management Reference Manual.

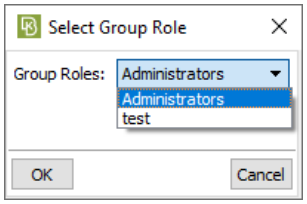
The moment KOMVOS gets connected to an SDM back-end, several settings are read that considerably affect the content, functionality and overall layout of the application. More specifically:

- The data model of the SDM back-end, that affects several behaviors within KOMVOS, like for instance Version Control or Lifecycle Management-related operations. Information on the configuration of the data model is given in chapter 5 of the Data Management Reference Manual.
    - In case an empty folder is specified as a file-based DM, any dm_structure.xml file that may exist within the `SDM_CONSOLE_HOME` directory will be copied within the new DM folder. If no such fule exists in KOMVOS' home, the hard-coded default data model will be used.
- In case the SDM back-end is SPDRM:
    - The Process and Issues toolbar buttons will be enabled if the SPDRM meets the minimum version requirements (1.6.0 for Process Management and 1.8.0 for Issue Management)
    - SPDRM's Generic Script Actions will be added under the **Actions** menu at the top
    - SPDRM's DM object type-specific Script Actions will become available in the context menu of listed items
    - SPDRM's Registered Applications will become available in the Applications Launchpad in the Home workspace
    - SPDRM's Mimetypes (file editors per DM object type and file extension) will be used as editors of files

When a user is connected to a server-based SDM back-end, his/her user name and role are displayed at the top right corner of the KOMVOS window.

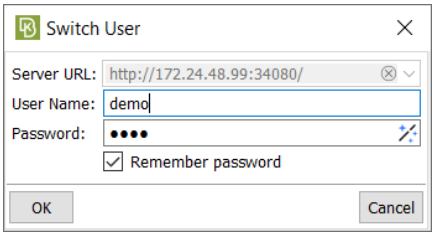


In this case, the options **Switch Role** and **Switch User** are also available in the **Login** menu.



Switch Role can be used in case the current user belongs to more than one different roles (that will be probably granted different privileges) and would like to change the current role.

Switch User can be used in order for the connection to SPDRM to be established with another user's credentials.



10.2. Data Views

The configuration of data views controls the way data are presented in the Data Management workspace and more specifically in the main lists and in the Details bottom tab. Additionally, through the data views configuration, the team leaders can load custom, Python script actions, in the context menu of DM objects.

Data Views are defined within the file `dm_views.xml` and can only be configured **centrally**, i.e. by placing this file in the `SDM_CONSOLE_HOME` directory.

The data views are highly dependent on the data model of the SDM back-end used. The use of KOMVOS with different SDM back-ends that have different data models will require the definition of different `SDM_CONSOLE_HOME` directories and subsequently the use of different start scripts.

More information on the configuration of the data views is given in chapter 5 of the Data Management Reference Manual.

Note that certain User Interface characteristics of the Database workspace can be configured differently for different user personas with the aid of Profiles, on top of the central configuration coming from the `dm_views.xml`. For more information on Profiles, please refer to paragraph 10.7.

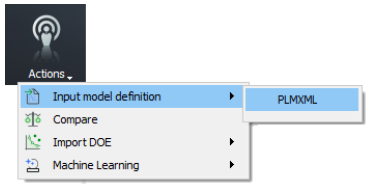
10.3. Actions

The configuration of the generic DM Actions, which are accessed through the **Actions** button in the main toolbar of KOMVOS, is performed with the file `dm_actions.xml` and can only be configured **centrally**.

The DM actions xml describes the characteristics of the different actions. Within the xml file, each action is described in a block like the one shown below:

File structure	Sample
Actions	<Actions> <Action> <action_name>PLMXML</action_name> <action_menu>Input model definition</action_menu> <action_menu_image>file_model_definition_input.svg</action_menu_image> <script_file_path>dm_scripts.py</script_file_path> <function_name>ReadPLMXMLInPManager</function_name> <keep_alive>YES</keep_alive> </Action> </Actions>
Action	
action_name	
action_image	
action_menu	
action_menu_image	
script_file_path	
function_name	
keep_alive	
add_separator	

Preview

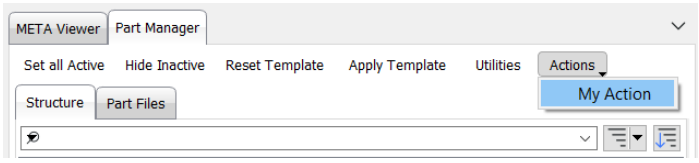


To load custom actions in the Part Manager of KOMVOS, the `dm_actions.xml` needs to be configured accordingly. A new **PManager_Buttons** block should be added to indicate that the referenced actions will be shown in the Part Manager window.

Within the *PManager_Buttons* block, one or more actions can be defined by adding *Action* blocks as the one shown below:

File structure	Sample
PManager_Buttons	
Actions	<SDM_Interface> <PManager_Buttons> <Actions> <Action> <action_name>My Action</action_name> <script_file_path>my_action.py</script_file_path> <function_name>run_action</function_name> </Action> </Actions> </PManager_Buttons>
Action	
action_name	
script_file_path	
function_name	

Preview



The **action_name** key is required and defines the name of the action that will be displayed in the list of actions.

The **action image** is optional and enables the use of an icon in front of the action name.

The **action_menu** key is optional and enables the grouping of actions under a grouping item in the list of actions.

The **action_menu image** key is optional and enables the use of an icon in front of the actions grouping item.

The **script_file_path** key is required and defines the file path, relative to the `SDM_CONSOLE_HOME` directory, of the python script that will be executed when the action is selected.

The **function_name** key is required and defines the script file function that will be executed when the action is selected.

The **keep_alive** key is optional and controls whether the action script will be automatically reloaded or not, each time the action is selected. If *keep_alive=YES,* the script will be loaded only the first time it is executed, unless the user explicitly reloads the script (default value = NO).

The **add_separator** key is optional and enables the addition of a separator under the action in the list of actions (default value = NO).

Actions specific to DM object types can be accessed through the context menu of DM objects depending on the SDM back-end. For SPDRM back-end, information on how to manage script actions is found in paragraph 7.3.1 of the SPDRM Administrator's Guide. For file-based DM, information on how to set-up context menu script actions is found in paragraph 5.2.1.5 of the DM Reference Manual.

10.4. Viewer

The configuration of the embedded 3D viewer enables the control of its overall layout (e.g. visible areas, docked or floating windows, background color, etc.) and the control of its available view modes (e.g. draw parts by their Status or Representation or Material type).

The layout of the viewer is defined with the file `viewer.xml` which is made available **centrally**. Any modifications done by the user are stored in the **user's home** directory on KOMVOS quit.

The different view modes are defined with the file `palettes.xml` and can only be configured **centrally**.

More information on the configuration possibilities of the Viewer can be found in paragraphs 3.6 of the Data Management Reference Manual.

10.5. Role-based customization

When the SDM back-end of KOMVOS is SPDRM, it is possible to control centrally the tools and workspaces that will be available for users, based on their role. The restrictions can be applied in the *SPDRM Administrator Console*, through the “Manage Tools/Windows” option of the context menu of roles, in the *User Management* workspace.

The restrictions that are possible through the *SPDRM Admin Console* and their corresponding effect in KOMVOS are listed in the table below.

Tool reference in the SPDRM Admin Console	**Effect of the restriction in KOMVOS**
`Data Manager`	The whole *Database* workspace is not visible
`DM Structure (Schema)`	The option *DM Schema Editor* is not visible under the *Tools* menu
`Search DM`	The *Search tool* appears deactivated in all lists of the *Database* workspace
`New (Workflow)`	Options to create new processes are not available in the *Processes* workspace
`Palette`	Opening of processes for editing is not possible in the *Processes* workspace
`Process Library`	The *Process Library* panel in the *Processes* workspace, is not visible
`Process Instance List` `Process Library*`	The whole *Processes* workspace and the *Tasks > Processes* tab of the *Home* workspace are not visible
`Submitted Jobs`	Option *Tools>Submitted Jobs* and *Tasks>HPC Jobs* tab of the *Home* workspace are not visible
`Issue Tracking`	The whole *Issues* workspace is not visible
** When both tools are restricted*	

10.6. Settings

KOMVOS settings can be configured through the *Settings* window, accessed either through the **Settings** button on the main toolbar or through *Tools>Settings* from the top menu bar (Ctrl+I shortcut).

Settings defined in this window are stored in the **user's home** directory the moment the related window closes with OK. The settings are stored within the file `sdm_settings.db` under `.BETA/KOMVOS/version_XX.X.X/DM`.

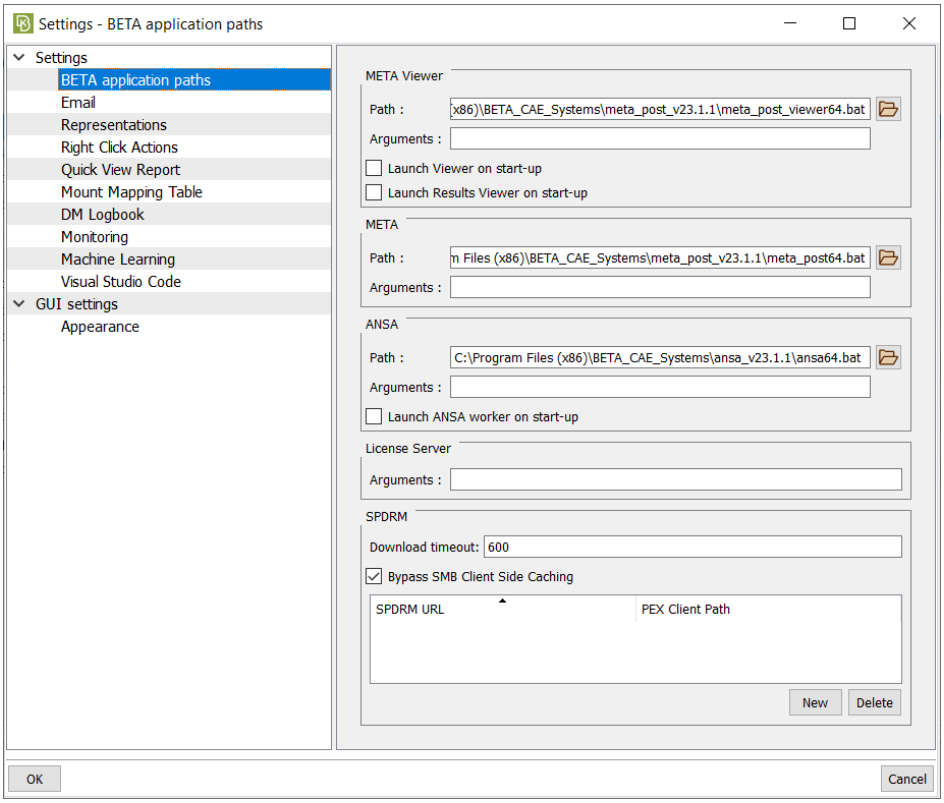
To enable central control of the settings, KOMVOS administrator can first save the settings in his/her home directory and then copy it centrally, within the `SDM_CONSOLE_HOME` directory, with the name `user_sdm_settings.db`.

When KOMVOS is launched, it will first read the settings from the central location and then read the user's settings from within the user's home directory. In all cases, the user's settings prevail.

In the *Settings* window, settings are organized into two main categories, the **Settings** and the **GUI Settings**, with the latter to host some configuration of the GUI. Note that some settings might differ according to the DM back-end, whether KOMVOS uses a file-based DM or it is SPDRM connected.

10.6.1. BETA apps

In the *BETA application paths* category the user can configure the set-up for the BETA Applications that may be launched through KOMVOS.



The **META Viewer** settings concern the embedded viewer that KOMVOS uses every time data are send for visualization by the user. Here, the executable path to be used can be configured, as well as optional arguments for the command. By default, when this path is empty, a META Viewer executable of the same software version with the running KOMVOS session will be searched in the installation directory.

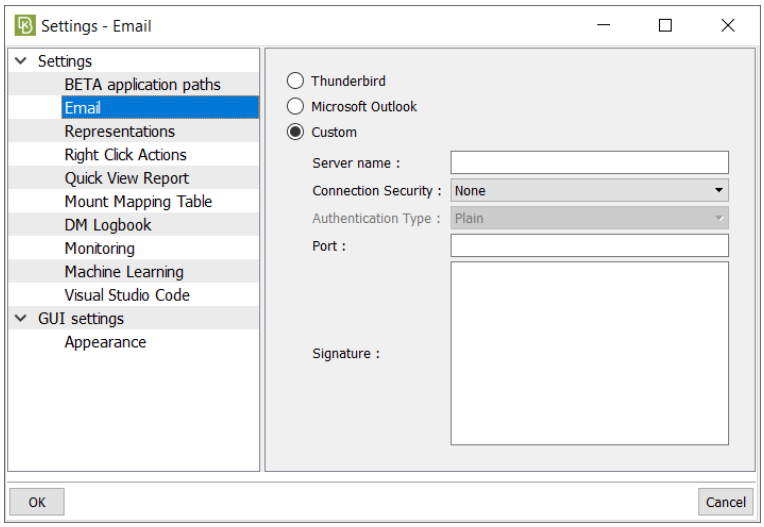
Activating the two options **Launch … on start-up**, the user can select to launch the viewer the moment KOMVOS is launched. Otherwise, the viewer is launched the moment visualization of data is requested by the user, (e.g. **View** action on some Key Results).

The **META**, **ANSA** groups of options will be used in a similar way, when data are requested to be viewed in the respective application. These paths will also be used by the ANSA/META shortcuts in the *Applications Launchpad* of the **Home** workspace. Note that in case of SPDRM back-end, the ANSA and META applications used as file editors and through the *Applications Launchpad* of the **Home** workspace, are those configured as Registered Applications by the SPDRM administrator. However, the ANSA executable defined here will still be used as a worker whenever required (e.g. during the automatic indexing that takes place when data are checked into the system).

In the **License Server** group, arguments that will be passed to all BETA applications, relatively to the license server, can be given (e.g. *lm_retry*, *lm_retry_timeout*, etc.).

Finally, in the **SPDRM** group of settings, a specific PEX, the process executor of SPDRM, can be configured. PEX is the background application used for the execution of SPDRM processes and SPDRM scripts and is included in the SPDRM client installation package (SPDRM version 1.6.0 and later). The path to the PEX is provided to KOMVOS by the SPDRM server automatically on connect. However, in cases where the centrally installed PEX is not "close" to the KOMVOS client, a local execution of PEX is recommended. In such cases, this table can be used in order to map SPDRM server URLs with local PEX paths.

10.6.2. Email

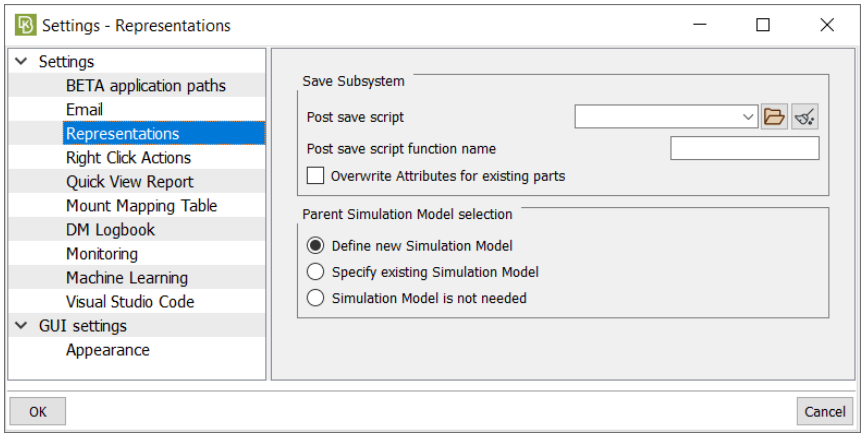


In the *Email* category, the appropriate email account must be set up, so that the users can compose e-mails directly through KOMVOS. Interfacing with Thunderbird and Microsoft Outlook accounts needs no further configuration and when the composition of an e-mail is requested, a *New E-mail* window of these e-mail clients will pop-up with the proper attachments in each case.

If neither Thunderbird nor Microsoft Outlook is available, KOMVOS can serve as an e-mail client with the proper configuration of a custom mail account.

10.6.3. Representations

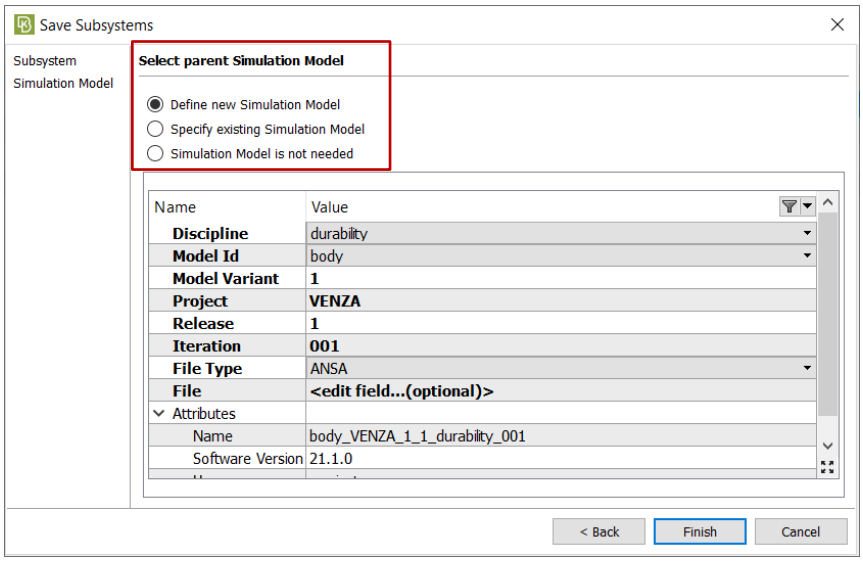
The *Representations* category collects settings related to the default supported workflow for Model Build that is available for file-based DMs (see Chapter 8).



Save Subsystem: A script function can be defined to be used as a post-save callback when the action **Save Subsystem** is used. Such a callback can be used to define or modify some attributes of the saved Subsystem.

Parent Simulation Model selection: Controls the default policy for the creation of Simulation Models when a new Subsystem is created.

For example, if the Define new Simulation Model option is selected, then, at the Save Subsystems window that appears when a new Subsystem is created, this option will appear as the default.



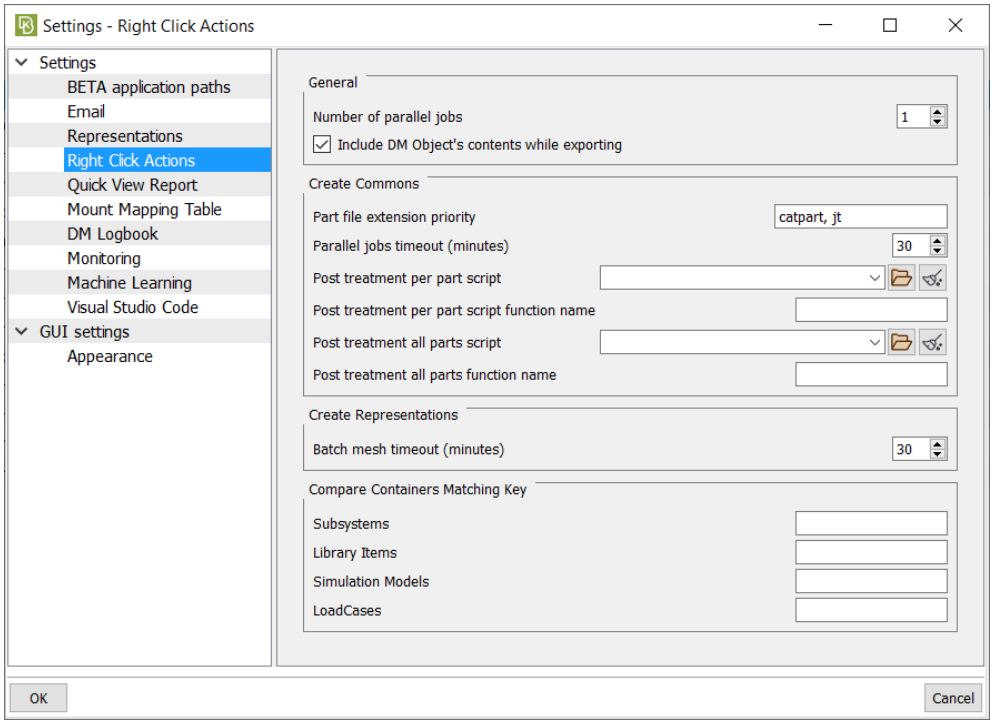
Define new Simulation Model: A new parent Simulation Model will be created along with the new Subsystem.

Specify existing Simulation Model: The new Subsystem will have as a parent Simulation Model an existing one which can be specified from a list that appears in the Save Subsystems window.

Simulation Model is not needed: The new Subsystem will not belong to any Simulation Model.

10.6.4. Actions

In the *Right Click Actions* category we find settings related to actions on data available as options in their context menu.



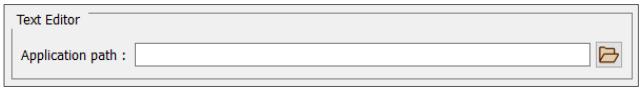
Number of parallel jobs
The number of parallel ANSA workers to be employed during translation and meshing tasks. Note that each worker will acquire one ANSA license. The maximum value for this setting is determined automatically based on the number of processors of the workstation.

Create Commons
These settings control the **CAD To ANSA** process that is executed through **Create Commons** action. For more information, please refer to paragraph 30.8.4 of the ANSA User's Guide. For more information on the CAD files translation action through KOMVOS, please refer to Chapter 8 of this User Guide.

Batch mesh timeout: Set up the time in minutes after which the worker should be killed.

Compare Containers Matching Key: These settings control the behavior of the Compare Containers function that compares two selected DM Objects in terms of metadata and contents. The settings determine which property of each DM object type will be used as a matching key during the comparison of contents. For example, running Compare Containers on two Simulation Models, the Subsystems Matching Key is the property that will be used to match the Subsystem of one model with the Subsystem of the other model. This property should be the Module Id. By default, these settings are blank, implying that the first property (in the order of their definition in the data model)

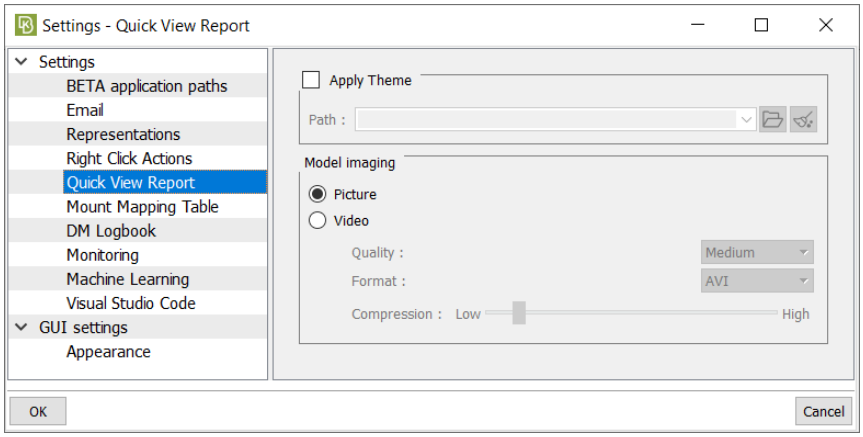
should be used for matching. However, after customization of the data model the order of properties may change, and the default rule may not be applicable.



Text Editor: Visible only while working with a file-based DM. An optional path can be given to the executable of a text editor, to be used whenever the option **Text Edit** is applied on specific data. If left blank, KOMVOS will launch the default system text editor.

10.6.5. Quick View Report

In this tab, user preferences regarding the creation of Quick View Reports are defined. Quick View Reports are available in the default configuration of KOMVOS for file-based DMs.

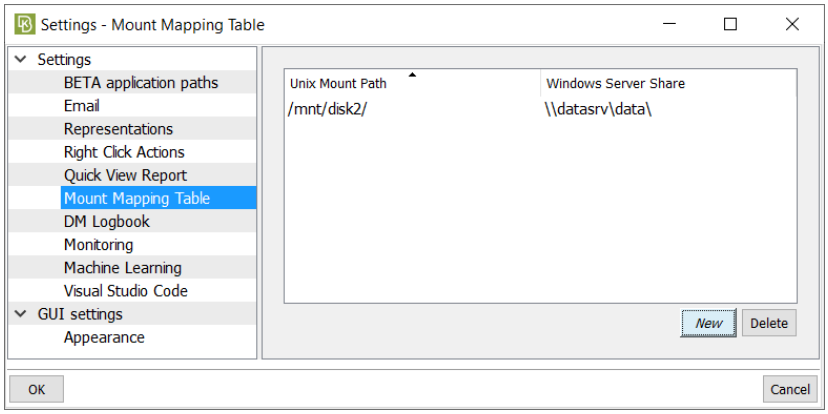


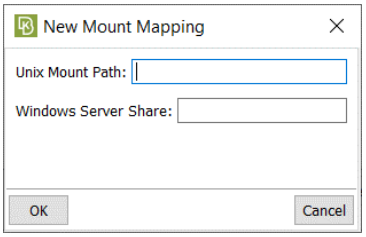
Apply Theme: Activate this setting to set a specific Powerpoint theme (*.thmx) to be used by the PPTX report composer.

Model imaging: Control whether Pictures only or also videos will be created upon the report generation.

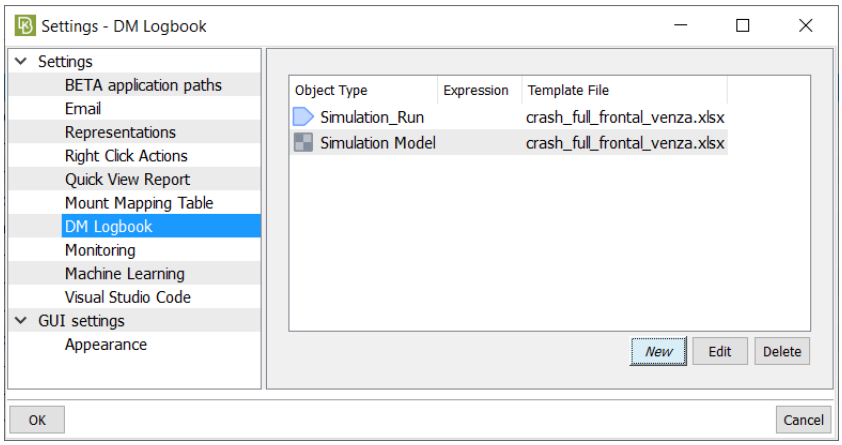
10.6.6. Mount Mapping Table

When working in environments with users using Linux and Windows, it is possible to define mapping of Unix Mount Paths to Window Server Shares in order to use it through custom actions.





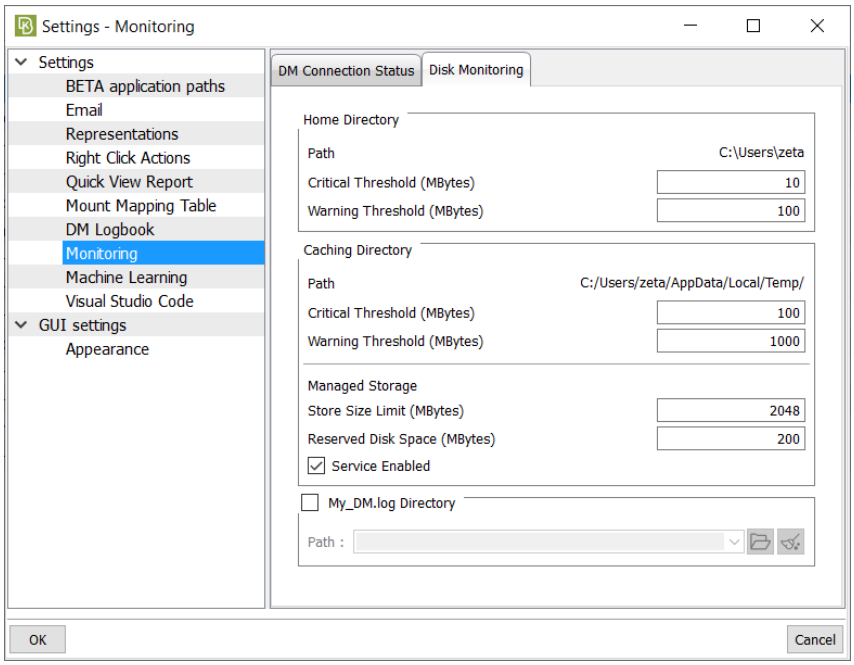
10.6.7. DM Logbook



This tab collects the settings for the DM Logbook functionality. Different Logbook template files can be defined as xlsx files, to enable the Logbook view for Simulation Data. For more information, please refer to paragraph 4.3.3 of the Data Management Reference Manual.

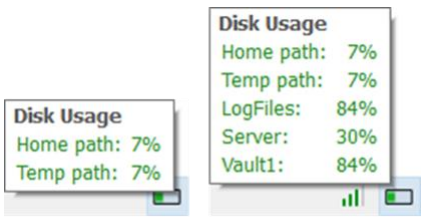
10.6.8. Disk Monitoring

In this tab, specific preferences regarding disk usage are defined.



Home Directory: The thresholds of free disk space below which, a low disk space warning or error should be shown in the bottom Status bar.

Note that according to the DM back-end, file-based or server-based, the indications differ. Furthermore, in case of an SPDRM back-end, the thresholds for the SPDRM Vault utilization are defined in the SPDRM configuration folder. Please refer to the SPDRM Administrator's Guide (paragraph 5.9.2.1) for more details.



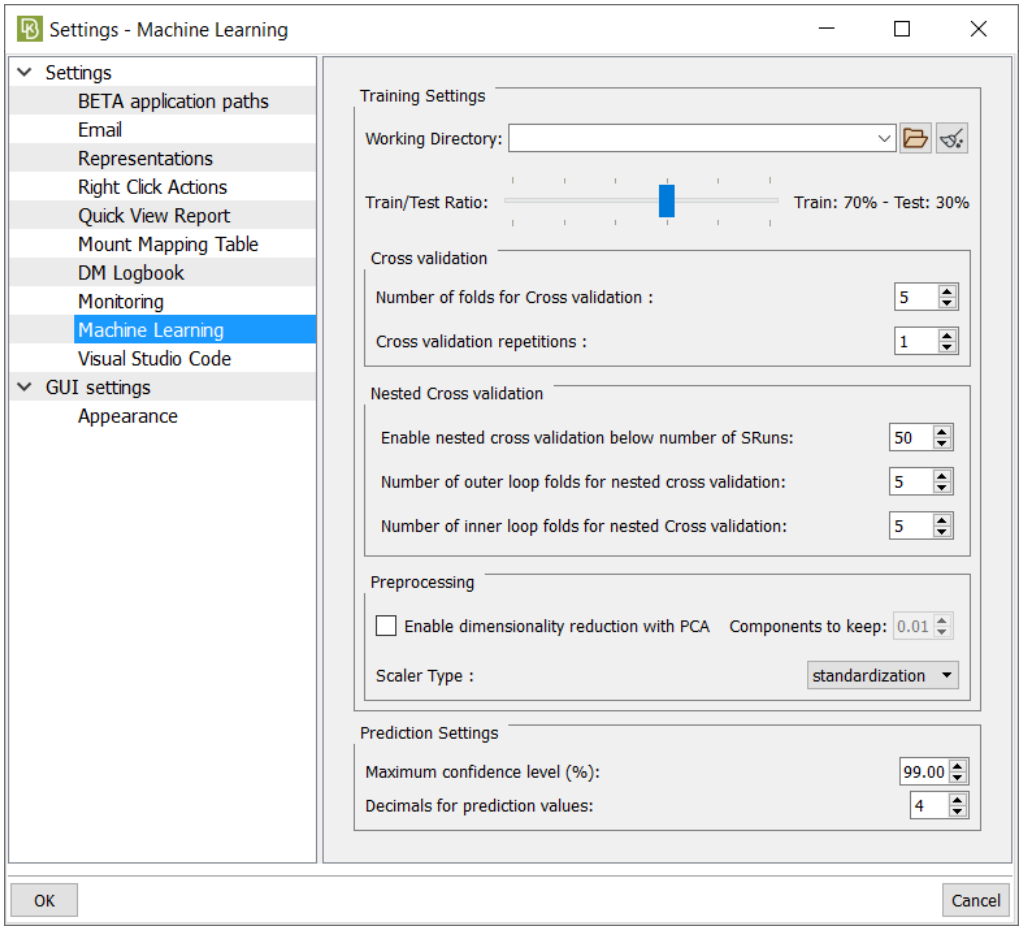
File-based DM *SPDRM back-end*

Caching Directory: Settings for the footprint of the cache storage used for client-side caching. For more information on file caching, refer to paragraph 2.10 of the Data Management Reference Manual.

My_DM.log Directory: Custom location for the My_DM.log file. More information on client-side logging is available in paragraph 7.2 of the Data Management Reference Manual.

10.6.9. Machine Learning

Specific settings on how to run the Machine Learning functionality can be controlled within this category.



Train/Test Ratio: Defines the percentage of data to be used for training and testing the Machine learning algorithm. From the selected data, the Train percentage will be used specifically for the training of the algorithm. The test percentage will be "unknown" to the algorithm and will be used in order to validate the algorithm's accuracy and produce the performance indicators (Predictor Reports). Nested Cross Validation uses a different methodology which don't use the Train/Test ratio and it is explained below.

Cross validation: During the training of the Machine Learning Algorithm (predictor), a cross validation process takes place where datapoints from the dataset are selected in order to be used as validation datapoints.

- **Number of folds for Cross Validation** option refers to the number of times the initial dataset will be split (creating folds), from which the selection of validation datapoints will be done.
- **Cross validation repetitions** option refers to the number of times the cross validation process will take place. More repetitions will take more time but suggest more accuracy.

Nested Cross Validation: Nested cross validation process is a process that takes places when the initial dataset is small, in order to improve the accuracy of the algorithm.

It has two phases. In the first phase, the dataset is split in a number of folds (outer loop folds). In every repetition, one fold is selected as the test set and the others as the training set. In the second phase, for each of the above repetitions, the selected training set is split in a number of folds (inner loop folds) where simple cross validation takes place.

- **Enable nested cross validation below number of Simulation Runs**: This option sets the threshold of datapoints/Simulation Runs below which the Nested Cross Validation will take place.
- **Number of outer loop folds**: During this process, the dataset is separated to a number of partitions. The algorithm will use each time a fold for test set and the others for training. This way more data can be used for training the algorithm in order to get more accuracy.
- **Number of inner loop folds**: This is similar to the number of folds mentioned on the cross validation option, but now referring to each outer loop.

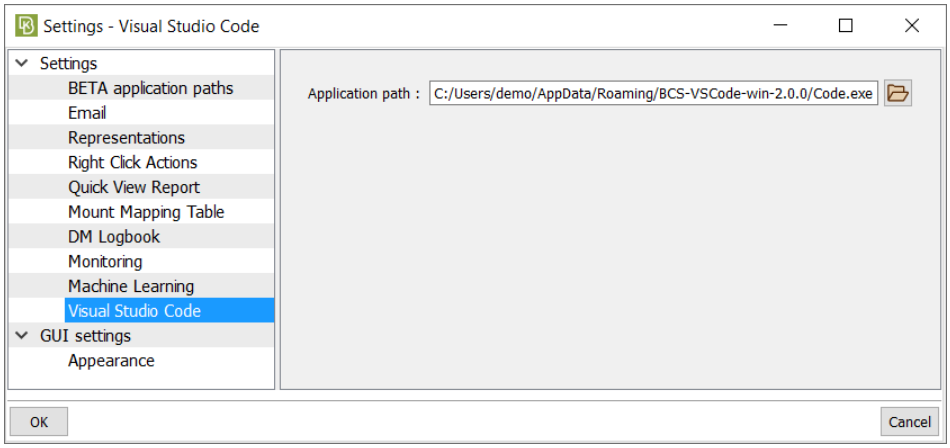
Preprocessing:

- **Enable or disable dimentionality reduction with PCA** (Principal Component Analysis), to reduce the number of dimensions of the dataset before training starts.
- **Components to keep** is a float number between 0.01 and 1.0. The number of components is the percentage of original features.
- **Scaler Type** is the scaling method for the learning process. Scaling is a method used to normalize the range of data features.

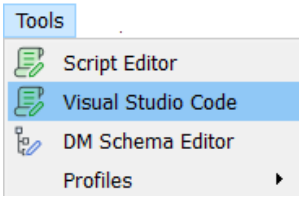
Prediction:

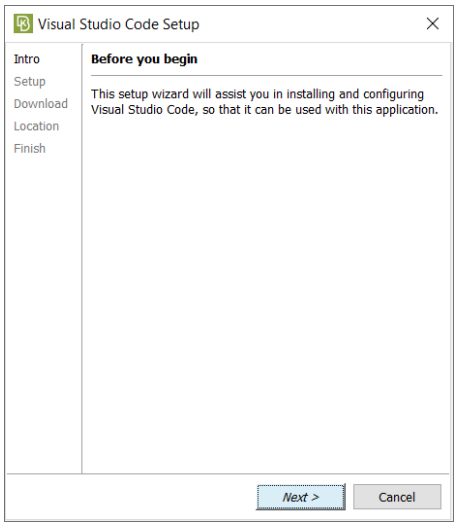
- **Maximum confidence level** percentage for the variance of predictions: Smaller percentage indicates smaller variance of prediction but also decreased accuracy of variance estimation.

10.6.10. Visual Studio Code



In this tab, it is possible to define the executable of Visual Studio Code to be used as a script editor or for debugging purposes.



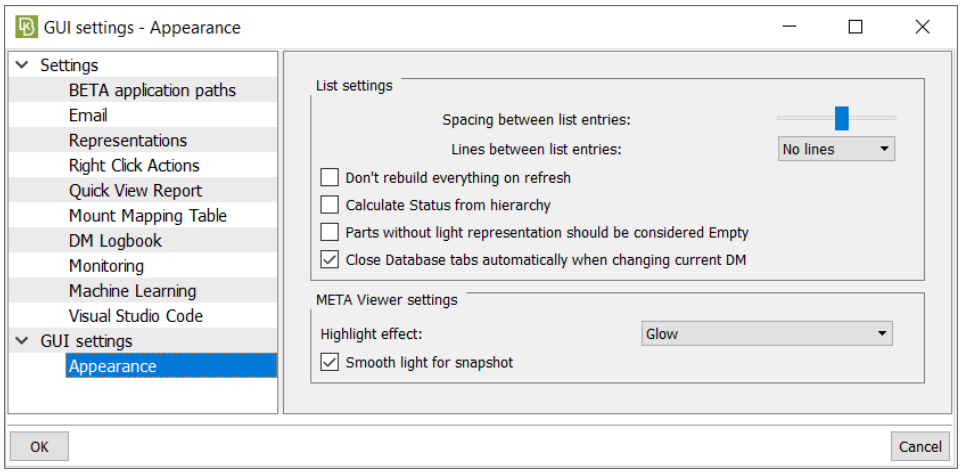


The Visual Studio Code can be also set up from the Visual Studio Code Setup Window that appears when selecting Tools > Visual Studio Code. In this case, the path from Preferences tab is defined automatically.

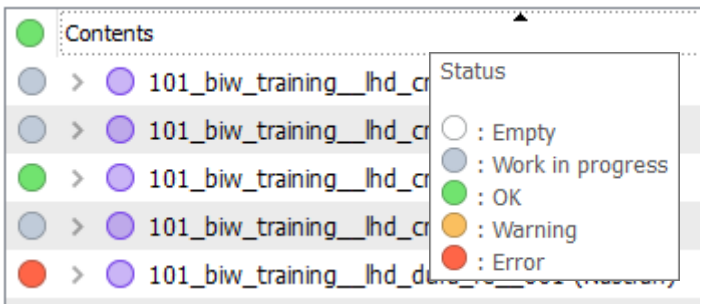
More information about Visual Studio Code Setup can be found at:
Help > Documentation Index > Automation Guides > BETA Suite API > Application Development > IDE Integration > BCS Development Environment with Microsoft Visual Studio Code.

10.6.11. Appearance

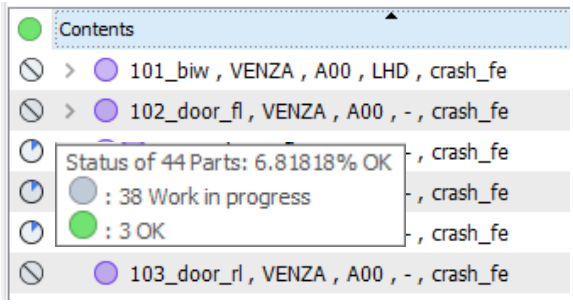
In the *Appearance* category of the *GUI Settings* group, the preferences regarding the visual depiction of entities are defined.



Calculate Status from hierarchy controls whether the Status column in the main data lists will be filled according to the value of the Status attribute or as an accumulation of the status values of the DM items contained in each object (e.g. the Status of a Subsystem could be accumulated from all the parts it contains).



Calculate Status from hierarchy: **OFF**



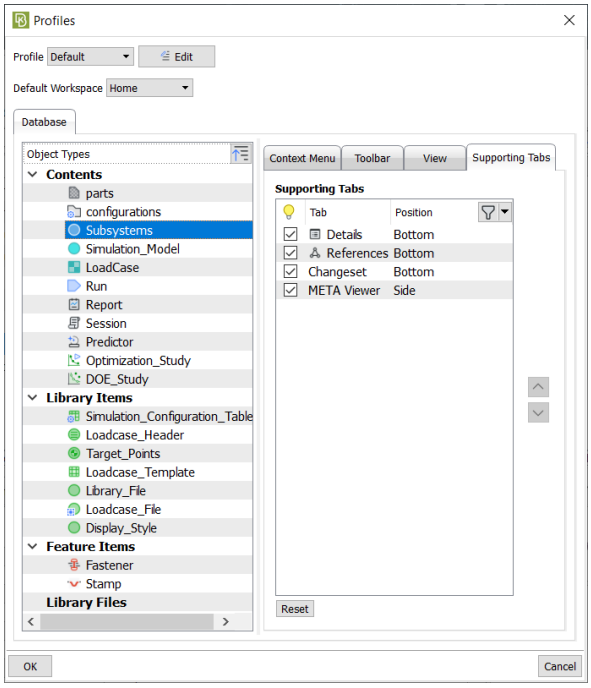
Calculate Status from hierarchy: ON

10.7. Profiles

As KOMVOS can be used by different user personas that have different needs in terms of information extraction or needed functionality, a *Profile Editor* is available in order for the administrator of KOMVOS to tune such settings **centrally**, for different user profiles. For the time being, the scope of Profiles is limited to the tuning of the Database workspace. Therefore, the *Default* profile, if unchanged, contains the default set-up which is only affected by the Data Views configuration (`dm_views.xml`).

A Profile currently controls:

- The **default workspace** of the user (Home, Database, Process, Issues, etc.)
- The layout of the **context menu** per DM object type
- The layout of the **top toolbar** in each DM object type's tab
- The **default view** of the main list per DM object type
- The default visible **supporting tabs** (bottom info tabs and right-side panel tabs) and their order



The Profile Editor opens through the menu-bar option **Tools > Profiles > Manage**.

Within the Profile Editor, one or more custom profiles can be created for each user persona. Depending on the SDM System that is used as back-end (SPDRM or file-based DM) different options may be available.

All information regarding **Profiles** is stored in the form of an xml file named `dm_profiles.xml`.

10.7.1. Profiles configuration file

All information regarding **Profiles** is stored in the form of an xml file named `dm_profiles.xml`.

This definition of profiles within this file is done in different sections for file-based DM and SPDRM back-end. This means that the Default profile for SPDRM back-end may have a different definition from the Default profile used for file-based DM.

10.7.1.1. Reading the profiles configuration file during start-up

The `dm_profiles.xml` is read from:

- The KOMVOS application home defined through `$SDM_CONSOLE_HOME` (by default the **config** directory in the installation)
  - In case of SPDRM back-end, KOMVOS looks for the `dm_profiles.xml` first in the folder `$SDM_CONSOLE_HOME/SPDRM`. In case no `$SDM_CONSOLE_HOME/SPDRM` folder exists, it looks into `$SDM_CONSOLE_HOME`
  - In case of file-based DM, KOMVOS looks for the `dm_profiles.xml` in `$SDM_CONSOLE_HOME`
- If the file is not found in the KOMVOS application home, KOMVOS looks in the current working directory (the start-in directory on Windows OS)
- In case a `dm_profiles.xml` exists in both locations, the file found in the central location (`$SDM_CONSOLE_HOME`) prevails

10.7.1.2. Updating the profiles configuration file

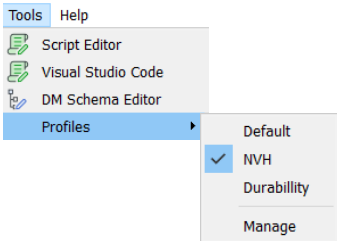
The default installation of KOMVOS does not include any `dm_profiles.xml` in the **config** directory. However, the first time the *Profile Editor* window opens and closes with OK, a `dm_profiles.xml` will be created in the current working directory (start-in directory on Windows OS).

If the user has write privileges within the `$SDM_CONSOLE_HOME`, the generated file can be moved there so that it's read by all users during the next KOMVOS launch.

If the KOMVOS session was launched with a `dm_profiles.xml` read from the `$SDM_CONSOLE_HOME` and the profiles are modified, the moment the *Profile Editor* closes with OK:

- The central `dm_profiles.xml` will be updated within `$SDM_CONSOLE_HOME`, if the user has write privileges on the file
- The message "`cannot open <absolute_path_to/dm_profiles_xml>: Permission denied`" will be printed and any changes will be lost.

10.7.2. Selecting the profile to be used



From within the KOMVOS GUI, a user can change the current Profile through the option **Tools > Profiles** by selecting any of the available custom profiles of the list.

During KOMVOS start-up, the Profile name to be used can be defined through the command line argument: **`-profile <profile_name>`.**

! NOTE: The profile that was active just before KOMVOS termination is saved as a setting in the user's home directory:

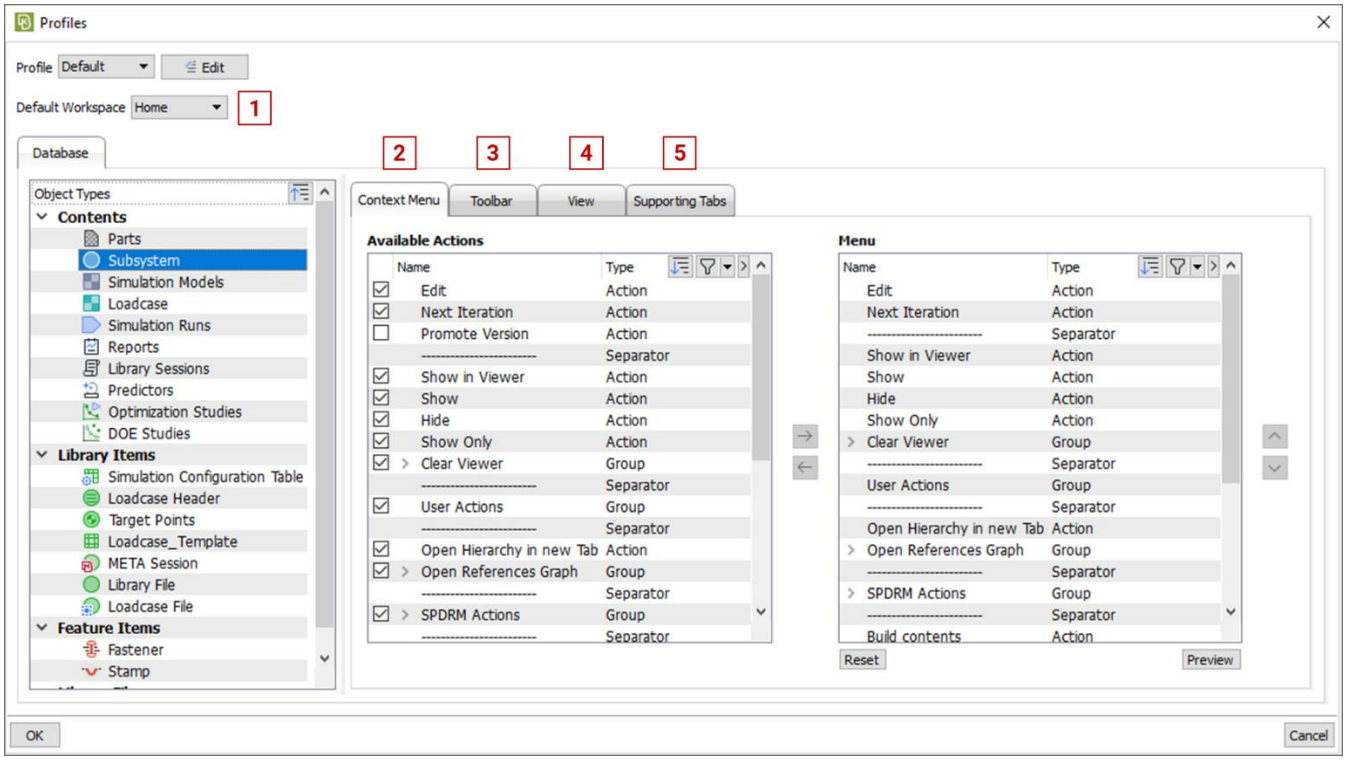
```
.BETA/KOMVOS/version_XX.X.X/KOMVOS.xml
```

This way, on KOMVOS start-up, the last used profile is re-used.

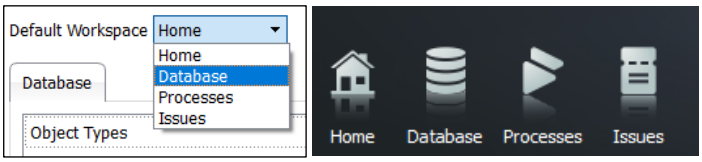
10.7.3. Editing a profile

Profiles are edited within the *Profile Editor* (**Tools > Profiles > Manage**).

Editing an existing profile is a 5-step process, as shown in the screenshot below and is described in the following paragraphs.



10.7.3.1. Defining the Default Workspace



The **Default Workspace** controls the workspace the user sees when KOMVOS is launched. Must be set to one of the available options of the top toolbar.

! Note that for file-based DMs, the options Processes and Issues are not available.

10.7.3.2. Defining the content of the Context Menu

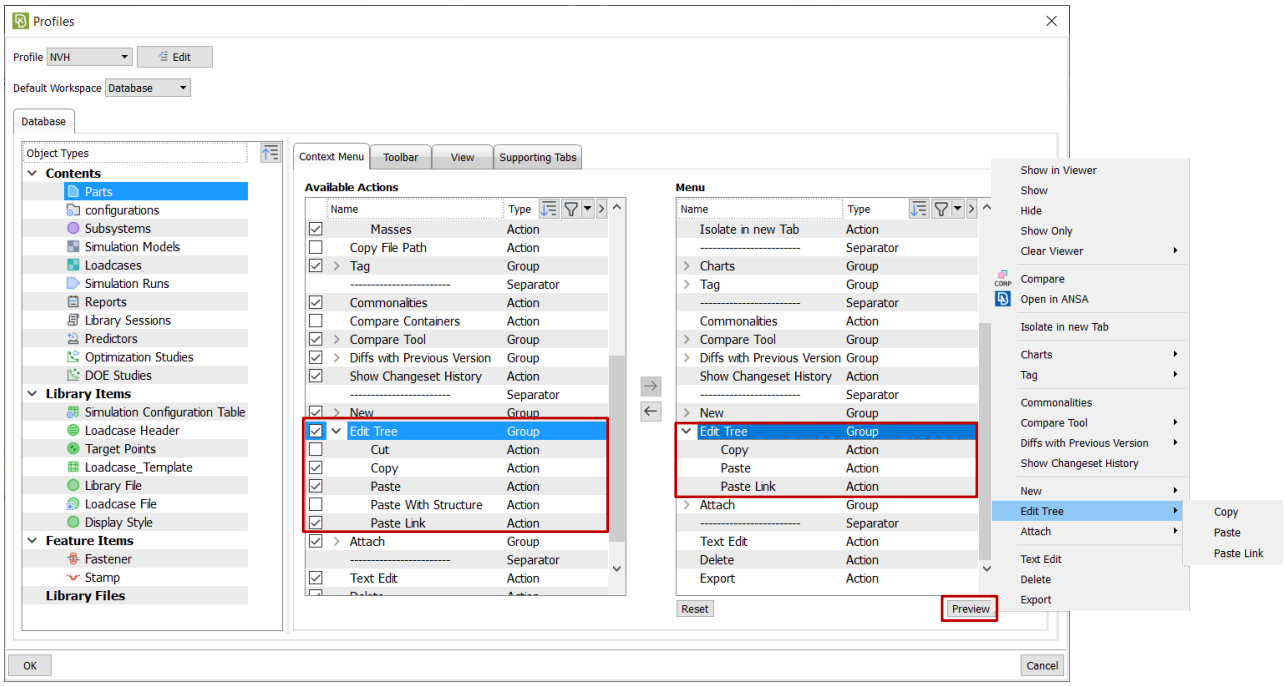
The options available in the default context menu of the DM Object Type are an accumulation of:

- The hard-coded actions or grouping items
- The custom actions defined through the `dm_views.xml`, that can be defined either per DM object type or per DM object type and view
- The custom actions defined in the SPDRM Administration Console as DM Item or Library Item Script Actions

Through the *Profile Editor*, any of the default visible actions can be removed. Additionally, actions can be re-ordered and separators can be added.

In the **Context Menu** tab, all possible actions for the DM Object type are listed in the *Available Actions* list on the left, and the actions in the configured context menu are shown in the *Menu* list on the right. In the *Available Actions* list, the check-box on the left indicates whether an action is included in the context menu of the selected DM Object type or not. An action can be added/removed from the target menu content either through this check-box or with the arrow buttons in the middle.

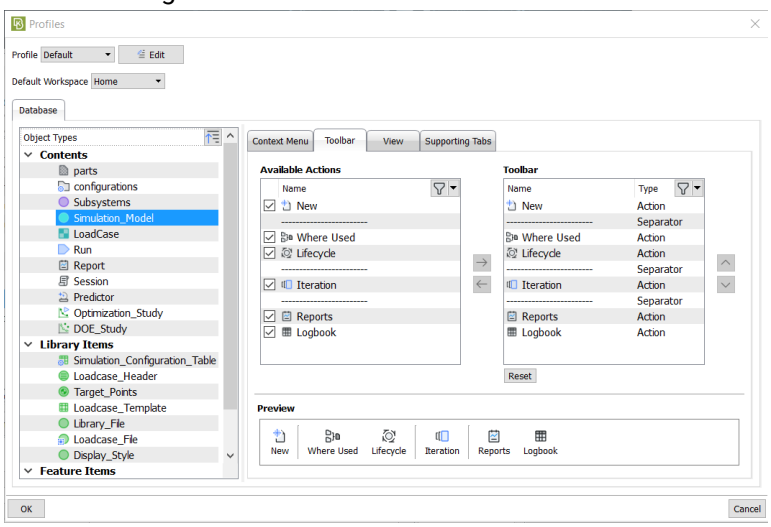
A preview of the new context menu can be seen with the use of the **Preview** button. The **Reset** button restores the default settings.

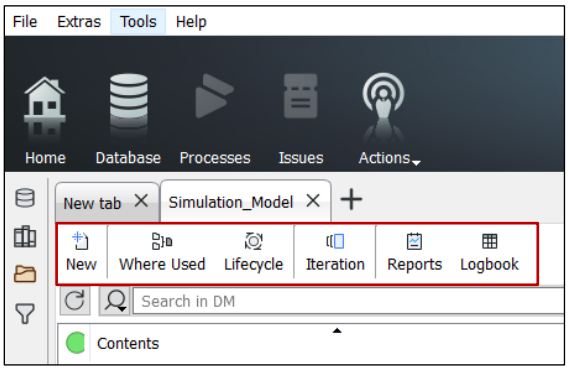


10.7.3.3. Defining the content of the Toolbar

In the **Toolbar** tab, it is possible to tune the content and layout of the default toolbar of the selected type by removing buttons, adding separators and re-order the buttons.The *Available Actions* list on the left shows all supported actions. The *Toolbar* list on the right shows the actions of the customized toolbar. Actions can be added/removed through the check-box in the *Available Actions* list or through the arrow buttons in the middle.

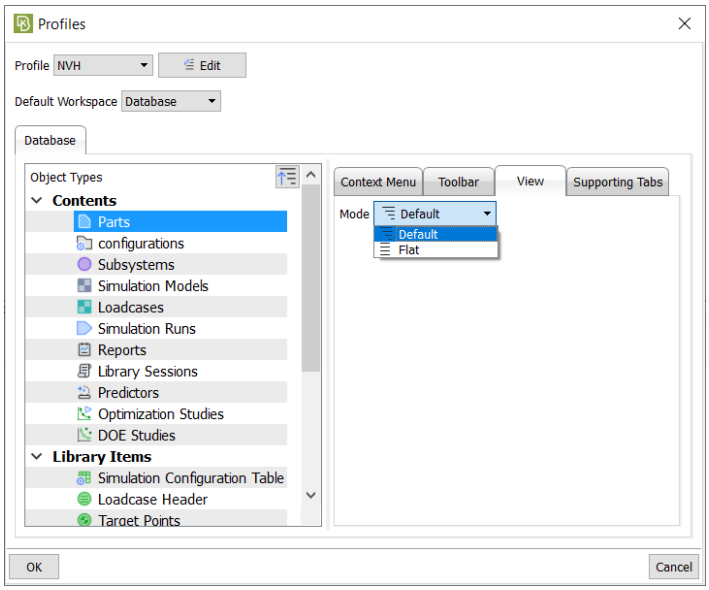
A **Preview** is available here too. The **Reset** button restores the default settings.





10.7.3.4. Defining the default list view

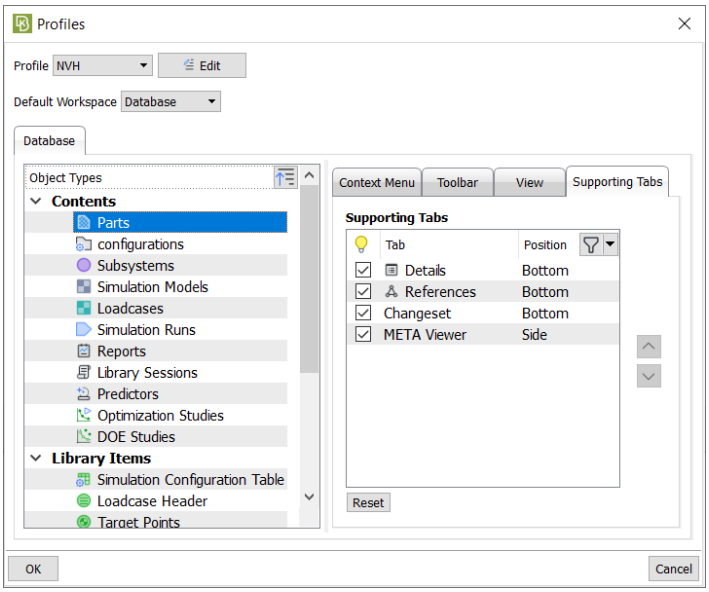
In the **View** tab, the user can select the default list View mode per-type.



The options listed in the combo-box are the different views of the selected type as these are defined in the `dm_views.xml`.

10.7.3.5. Defining the default supported tabs

In the **Supporting Tabs** tab, the visibility and order of the bottom/side tabs can be controlled.

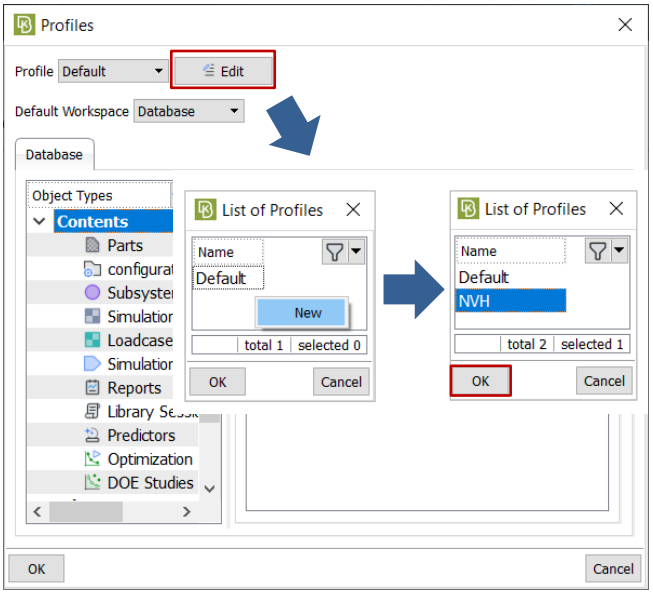


Even if a Supporting Tab is by default hidden, the user can still add it to the view through the visible tabs controller, on the right of the tab view.

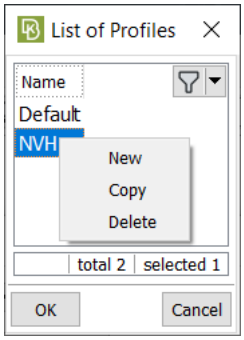
! Note that different bottom tabs are available for file-based DM and SPDRM back-end.

10.7.4. Creating new profiles

New profiles can be created through the *Profile Editor* window.



By pressing the **Edit** button next to the Profile name, the *List of Profiles* window pops up and the **New** option of the context menu can be used to create a new profile and set its name, e.g. NVH.

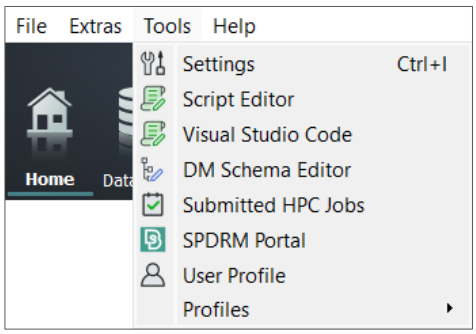


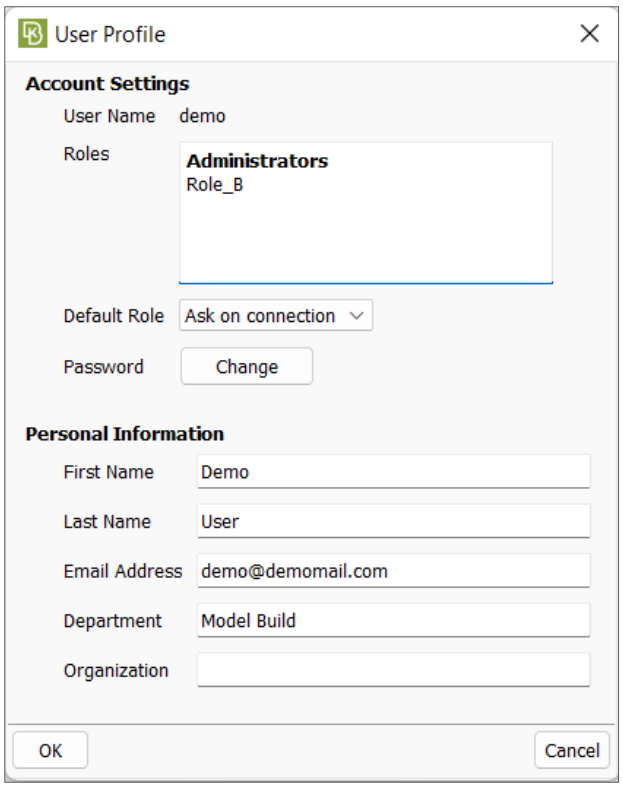
It is also possible to create a new profile by selecting the option **Copy** from an existing one. To remove a profile, select the **Delete** option.

To start modifying the newly created profile select it through the *Profile* combo box.

10.8. Account settings

When the SDM back-end of KOMVOS is SPDRM, the users can manage their User's Profile through the option *Tools>User Profile*. In the *User Profile* window that shows up, one can change the password and default login Role, as well as the personal information.





10.9. Command line arguments

KOMVOS can be launched pre-configured for certain operations with the aid of the following command line arguments:

Argument name	Description
`-dmroot`	Defines the SDM back-end. Can be followed by an absolute or relative path for file-based DM or the server address (URL) for SPDRM or MSC SimManager. Even in case of ASAM-ODS, a string with all connection information can be used for direct connection during launch through `-dmroot`.
`-profile`	Defines the KOMVOS user profile. Must be followed by the profile name
`-nogui`	Indicates that this will be a batch session of KOMVOS
`-exec`	Start KOMVOS and execute specified script commands. Often used combined with `-nogui`.

The complete list of command line arguments can be accessed by launching KOMVOS with `--help`

BETA CAE Systems International AG
D4 Business Village Luzern, Platz 4
CH-6039 Root D4, Switzerland
T +41 41 545 3650, F +41 41 545 3651
ansa@beta-cae.com
www.beta-cae.com

physics on screen