BETA
SIMULATION SOLUTIONS

physics on screen

SPDRM
WORKFLOW MANAGER

User's Guide

v1.8.0

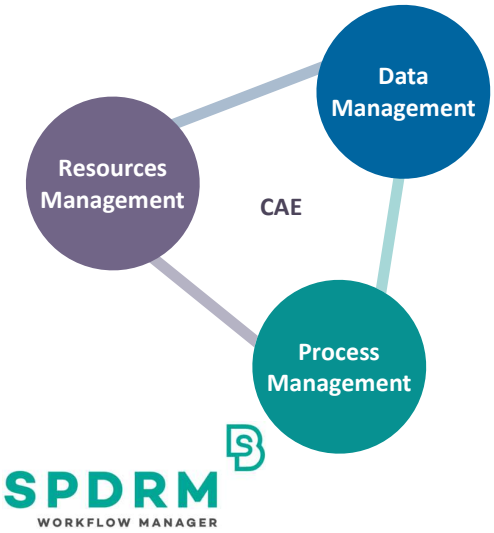
BETA CAE Systems International AG
D4 Business Village Luzern, Platz 4
CH-6039 Root D4, Switzerland
T +41 41 545 3650, F +41 41 545 3651
ansa@beta-cae.com
www.beta-cae.com

SPDRM
WORKFLOW MANAGER

Introduction

BETA
SIMULATION SOLUTIONS

Introduction to SPDRM



Product Name

SPDRM: Simulation Process, Data and Resources Manager

Function

Orchestrates all actors involved in the simulation process

Architecture

Client-server architectural pattern

Interfaces

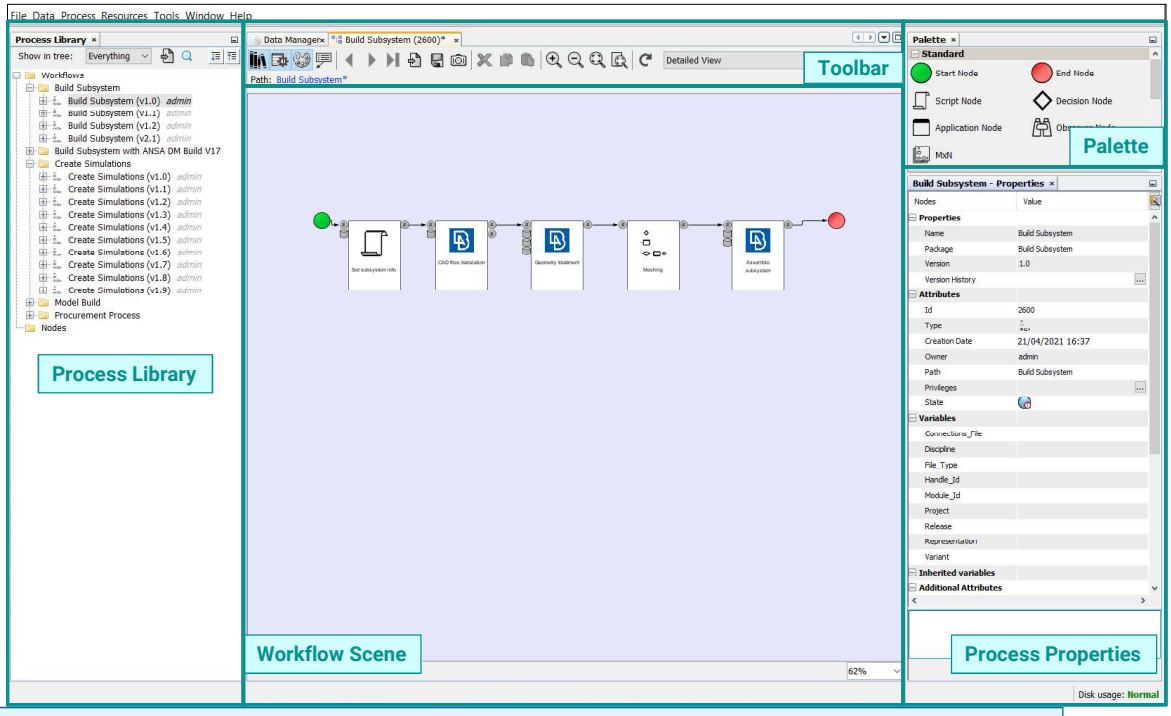
Offers API for communication of data to/from 3rd party applications
Built-in tight integration with ANSA and META

BETA SIMULATION SOLUTIONS

Introduction to process design through the SPDRM Client

A process is a group of nodes, which are connected with each other forming a workflow. A node can contain other nodes in the form of a process; this node is a sub-process. The design of processes is implemented in the Workflow scene window.

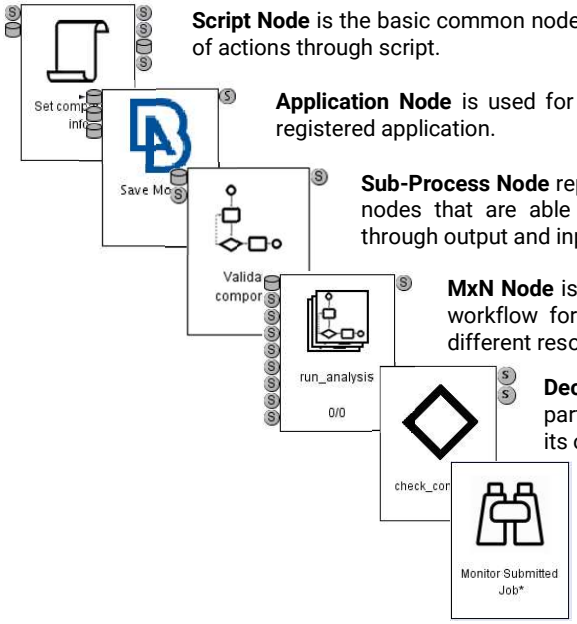
All workflows can be saved in the Process Library as Definitions. They can be retrieved then from this library in order to create instances of a workflow in the working space of clients connected to the SPDRM software. The ability to export Definition workflows (in the form of *.json files or *.ser binary files) is provided.



Go to Process > New, to open a new empty workflow scene.
Go to Process > Process Library, or press the respective button in the toolbar to access saved workflows.

BETA SIMULATION SOLUTIONS

Node and Input/Output Types



Script Node is the basic common node which is used for the execution of actions through script.

Application Node is used for actions that require the execution of a registered application.

Sub-Process Node represents a multiple-step task. It can contain sub-nodes that are able to communicate to the root (master) process through output and input streams.

MxN Node is a special sub-process node used to instantiate the same workflow for M different input data sets, optionally delegated to N different resources.

Decision Node is a special node used to drive the process towards a particular path, based on a condition statement. The node fills one of its output slots based on whether the statement turns True or False.

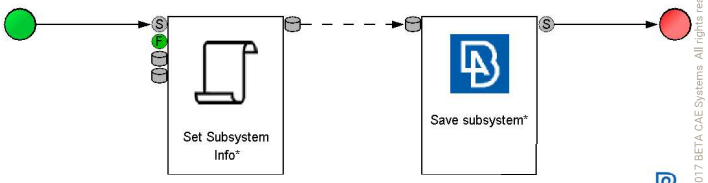
Observer Node is a special node used to monitor of a job submitted to HPC. Two types of observer node exist, **Script** and **Application Observer Node**, each of which embodies the functionalities of the respective simple ones, alongside their ability to enter idle mode, as long as they monitor the progress of the submitted jobs.

Supported types of variable for **Input** and **Output** slots are:

String, **Float**, **Integer**, **Boolean**, **Date**, **Single file**, and **List of files**.

Data can be provided to a node either through a connection with another node, or through the DM.

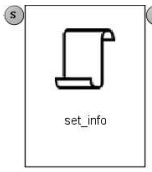
Files are communicated between nodes through dash connection lines, while the continuous connection lines represents the transfer of signals.



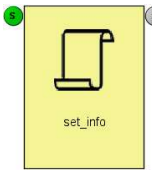
Color Coding of Nodes

A default color scheme is in place in SPDRM, that allows the user to quickly and easily identify the state of each process, node and inputs/outputs. During the execution of a process, the colors of a node and of its slots are changed, reflecting the transition from one state to another.

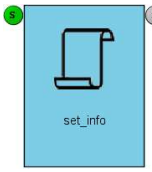
Basic node states



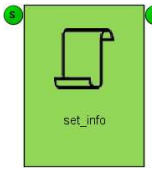
Not ready: Input Slots data are not available. Node cannot be executed.



Ready: Input Slot data are available. Node is ready to be executed.

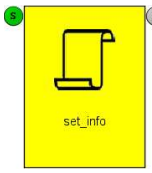


Running: Node is currently executed.

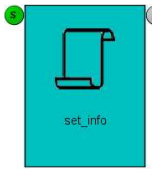


Finished: Node has finished its execution.

Additional (optionally acquired) node states

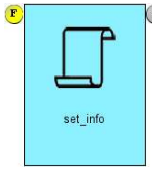


Scheduled or Paused: Input Slot data are available. Node is ready to be executed but it is scheduled to start on a particular time in future, or
node has been paused by the user (only for application nodes).



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

Observing: Node is in observing state and receives information about the progress of the submitted jobs. *Submitted Jobs* option from the context menu when in that state provides those information.



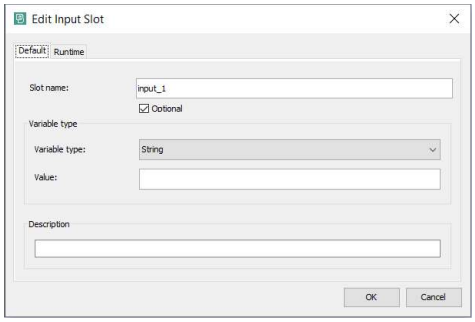
Polling input: Input slot of the node is polling for particular input data.

Color Coding of Slots

There are two types of input slots available:

- **Obligatory** input slot, that must be filled in order for the node to be executed.
- **Optional** input slot, that is not required to be filled in order for the node to start.

An input slot can be defined as obligatory or optional, by the *Optional* checkbox in the *Edit Input Slot* window. If the optional input slot is filled when the node is executed, then the slot becomes green, otherwise it remains as before (light purple) and the process continues.



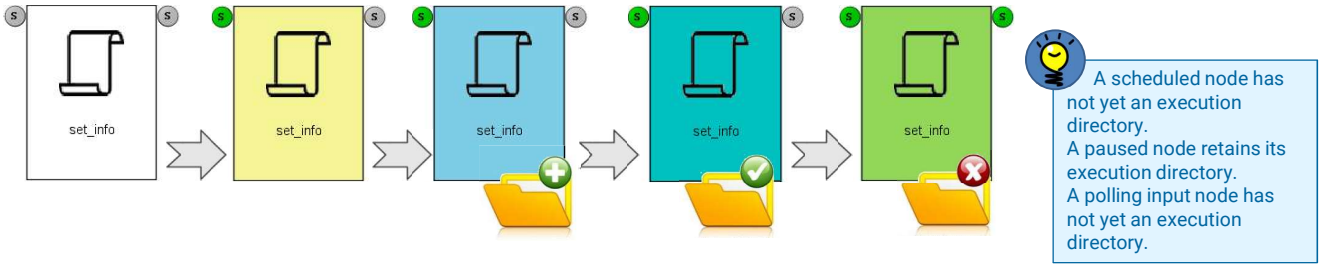
Input Slots:
- Data are not available (Obligatory slot)
- Data are not available (Optional slot)
- Data are available and ready to use

Output Slots:
- Data have not been provided by the node yet
- Data are available but they have not been released to the next node
- Data are available and they have been released automatically to the next node

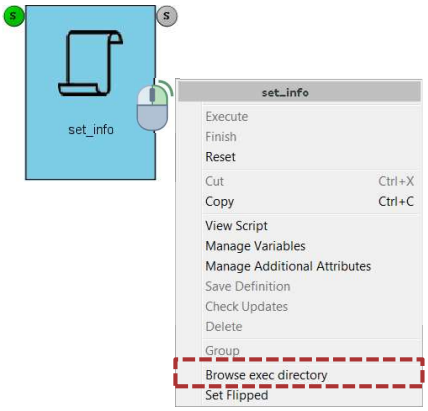
Node Execution Directory

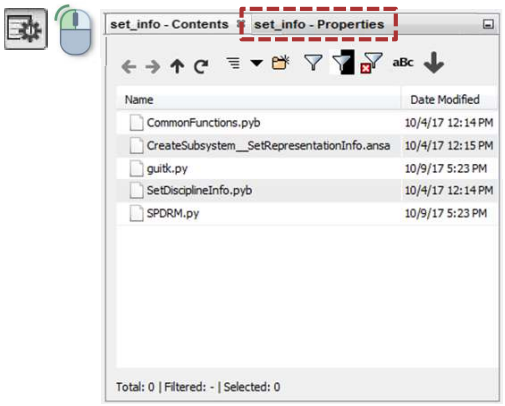
During the execution of a node, SPDRM creates a unique temporary directory, the Node Execution directory, for the communication of data between the client and the SPDRM database.

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

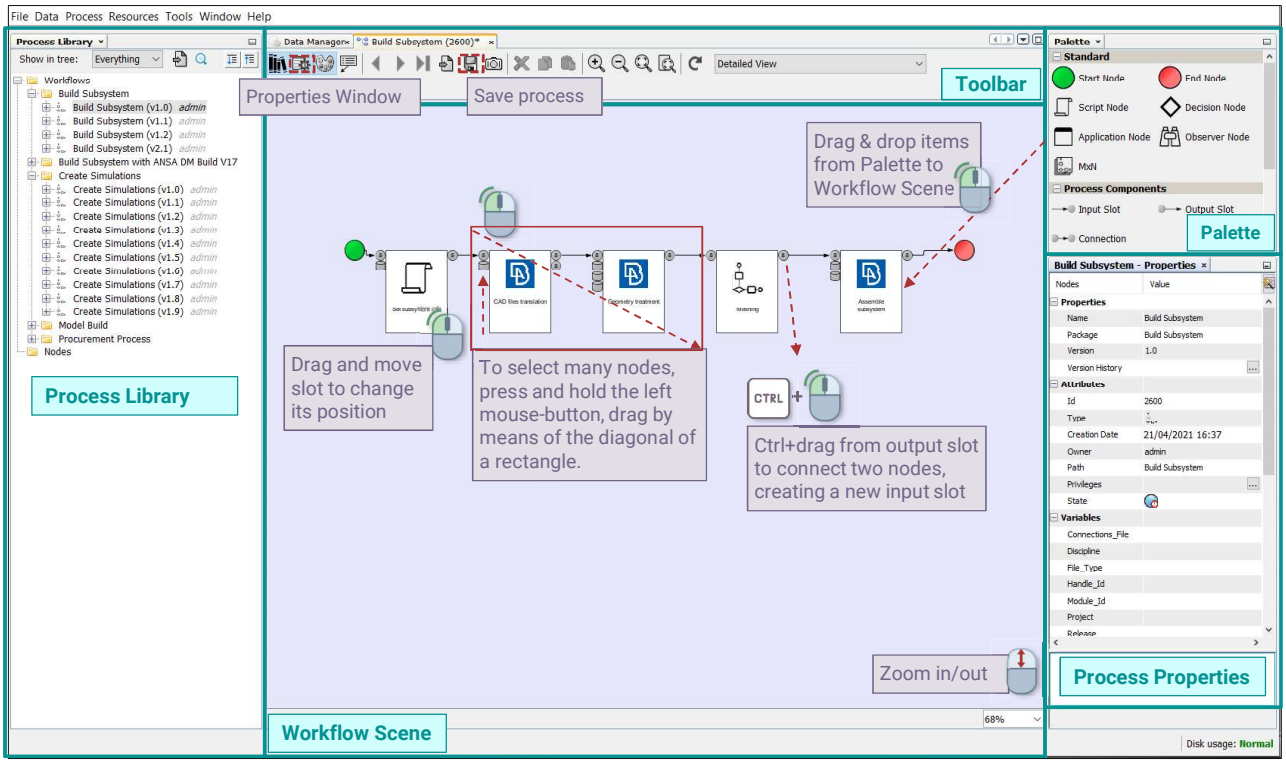


It is accessed through the **Browse exec directory** function of the context menu of the node, or through the *Contents* tab of the *Properties* sheet.





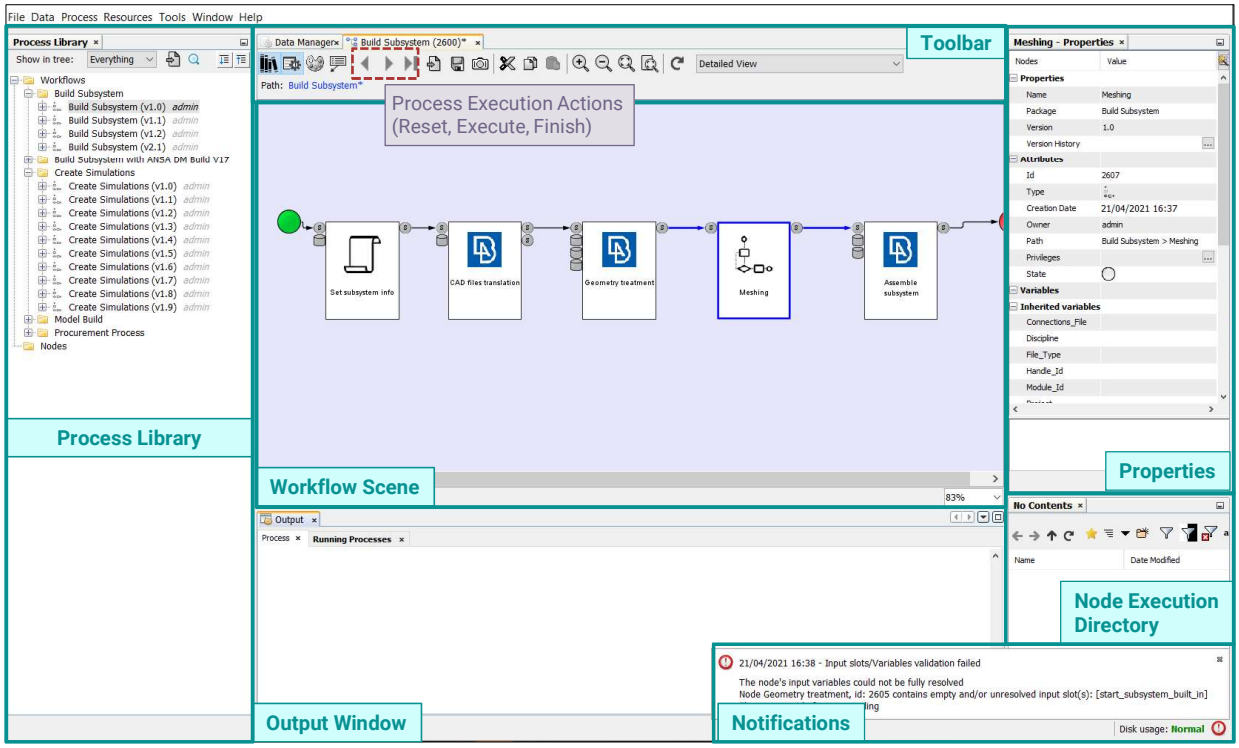
Basic actions during process design



Go to Process > New, to open a new empty workflow scene.
Go to Process > Process Library, or press the respective button in the toolbar to access saved workflows.

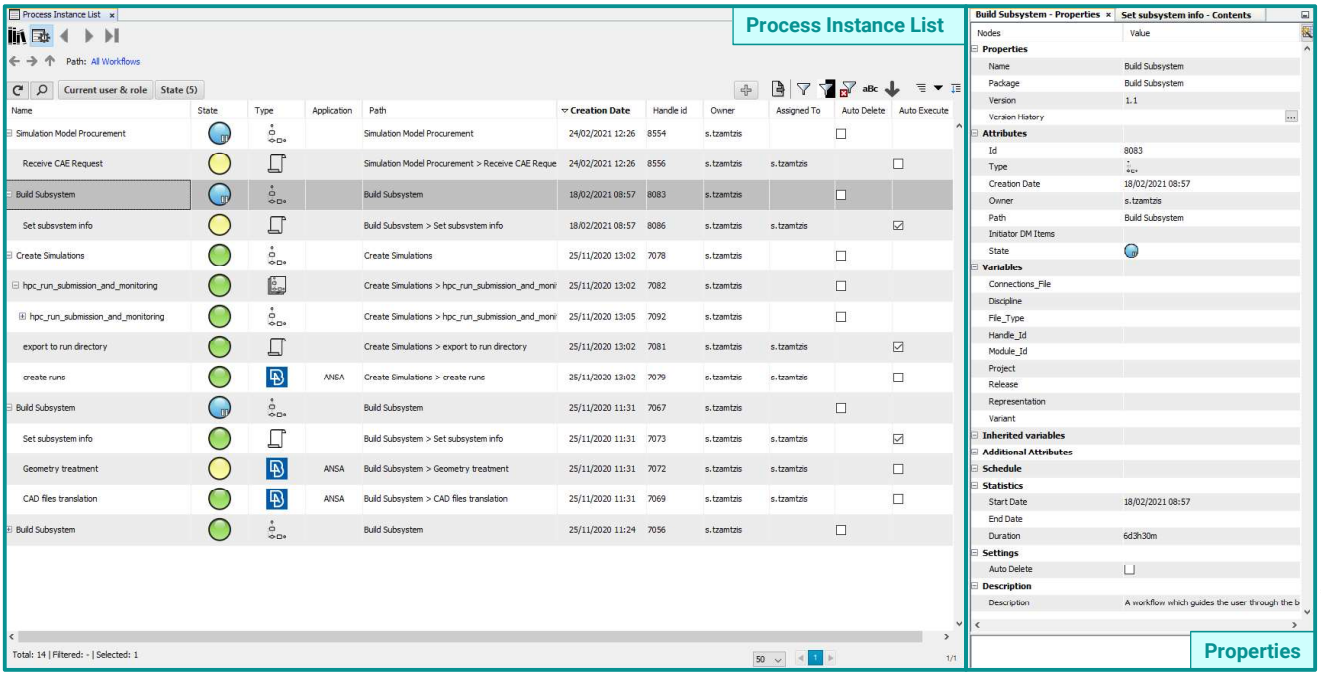
Introduction to process execution through SPDRM Client

The execution of a process can be performed by one or more Users or Groups that have been assigned with specific Nodes and can take place either at the *Workflow Scene* window or at the *Process Instance List* window.



Introduction to process execution through SPDRM Client

The execution of a process can be performed by one or more Users or Groups that have been assigned with specific Nodes and can take place either at the *Workflow Scene* window or at the *Process Instance List* window.



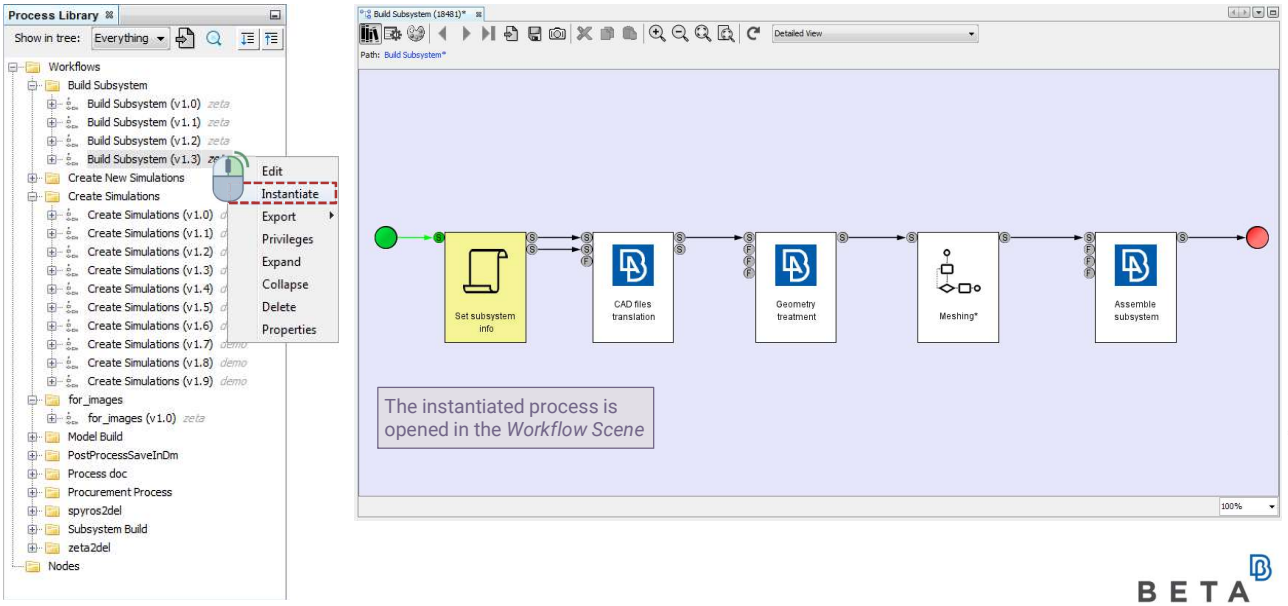
Execute a process from the Process Library to the Workflow Scene

The *Process Library* window (invoked from **Process > Process Library**) is a list of the available processes which are divided into:

- **Workflows** which contain all the process definitions that are accessible by the user.
- **Nodes** which can be used to search and re-use existing node definitions, during process design.

The user can instantiate a process from the library by pressing right-click on it and selecting the **Instantiate** option from the context menu.

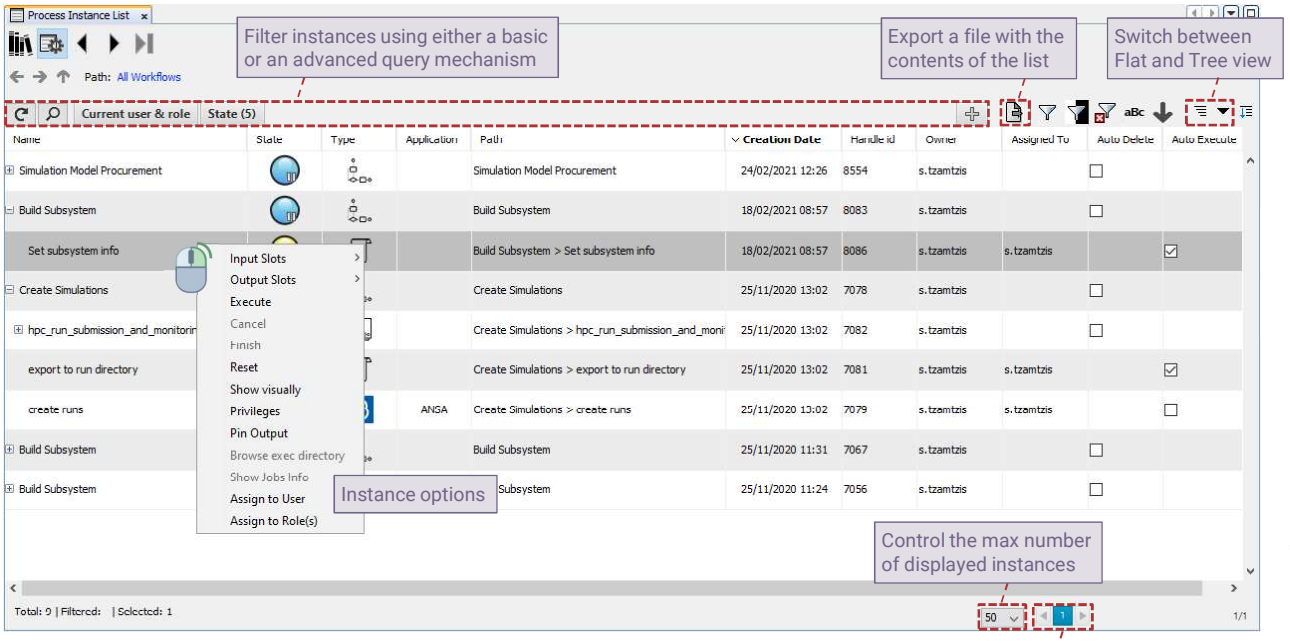
The process appears in the *Workflow Scene* window where the processes are executed.



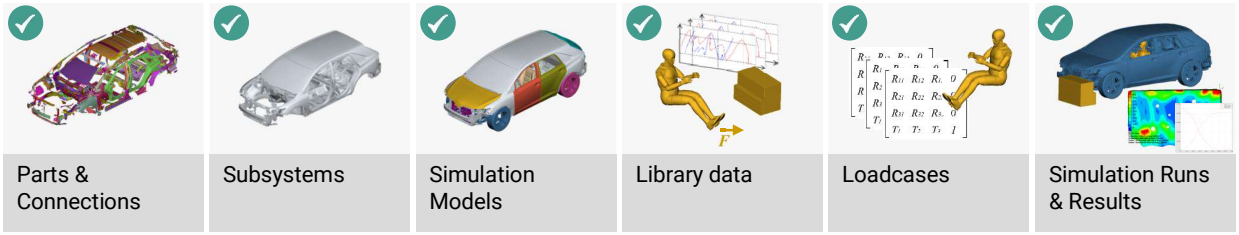
Execute a process through the Process Instance List

The process instances are managed through the *Process Instance List* window invoked from **Process > Process Instance List**.

Apart from process execution and information viewing, the Process Instance List comprises a dynamic tool for users, enabling them to have an overview of their "To Do" lists and manage their workload, using a search tool based on basic or advanced queries to filter the displayed instances.



Data Management Capabilities of SPDRM



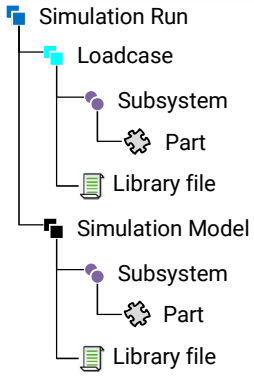
Parts & Connections	Subsystems	Simulation Models	Library data	Loadcases	Simulation Runs & Results

Key features:

- Data traceability
- Version control
- Security
- Sharing with team and external suppliers
- Data archival
- Customizable data types

Data Management Capabilities of SPDRM

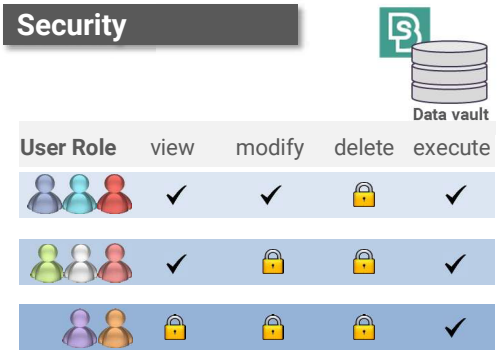
Data Model



The Data Model is the data organization schema used in SPDRM.
All the data types in SPDRM are defined through the Data Model.
The Data Model is :

- Enterprise specific
- Customizable

Security

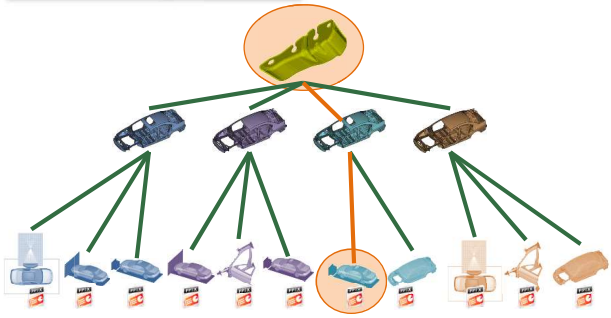


SPDRM controls the accessibility of each data object to protect enterprise data and ensure data integrity.

- Each data object has a list of permissions (**A**ccess **C**ontrol **L**ist)
- ACL is defined per User Role
- ACL can be specified per data object, data type or using default rules

Data Management Capabilities of SPDRM

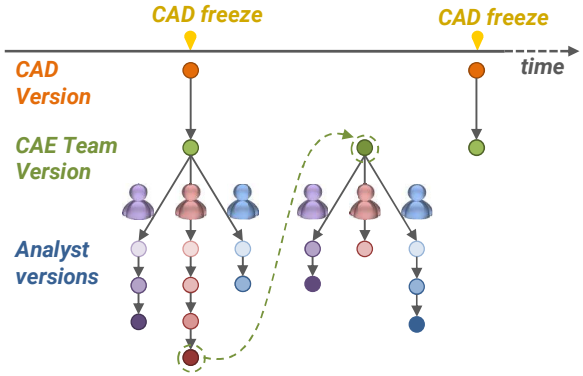
Traceability



SPDRM handles the relationships between data objects. Data tracking in SPDRM can be performed using different types of Links:

- **History Links**: Between different versions of the same object
- **Content-type Links**: When an object is contained in another object (e.g. a Part contained in a Subsystem)
- **Manual Links**: When an object requires another object as input (e.g. a Subsystem that has used a product definition .xml as input)

Version Control



SPDRM offers a Multi-Level Version Control system.
Several versioning schemes can be used to capture the evolution of a data object.

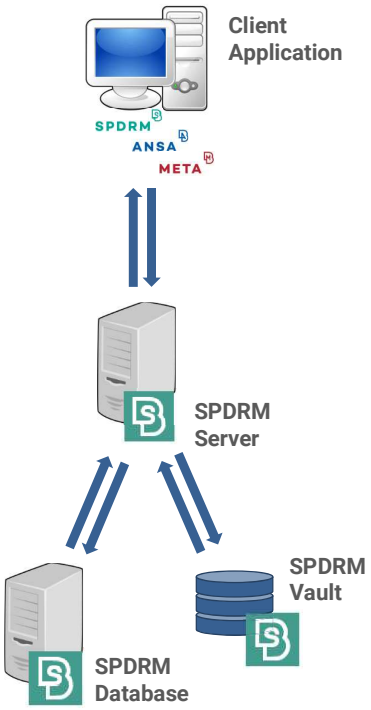
Data Management Capabilities of SPDRM

In SPDRM, the files related to data objects are stored in the Vault while the metadata related to each data object are stored in the Database.

Instead of directly accessing a file through the file browser, the user has to access the data object through the SPDRM Server. The Server fetches the right file from the Vault with the corresponding metadata from the Database.

The Server provides access to the contents of the Vault and Database to client applications. The applications that can act as clients are:

- SPDRM Client
- ANSA
- META
- KOMVOS
- 3rd party client applications



In case of **SPDRM Client**, Data I/O operations are performed through the Data Manager Window, or during Process execution.

In case of **ANSA**, **META** and **KOMVOS**, Data I/O operations are executed through the DM functionality of each application.

In case of the **3rd party client applications**, Data I/O operations are possible through Web Services.

Data Types in SPDRM

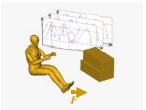
Any type of CAE data can be handled through SPDRM. The data types in SPDRM can be grouped in the following categories:

- **DM Items**

  These are customizable data types of primary data objects (ex. Parts, Subsystems, Simulation Models etc.).

- **Rich Library Items**

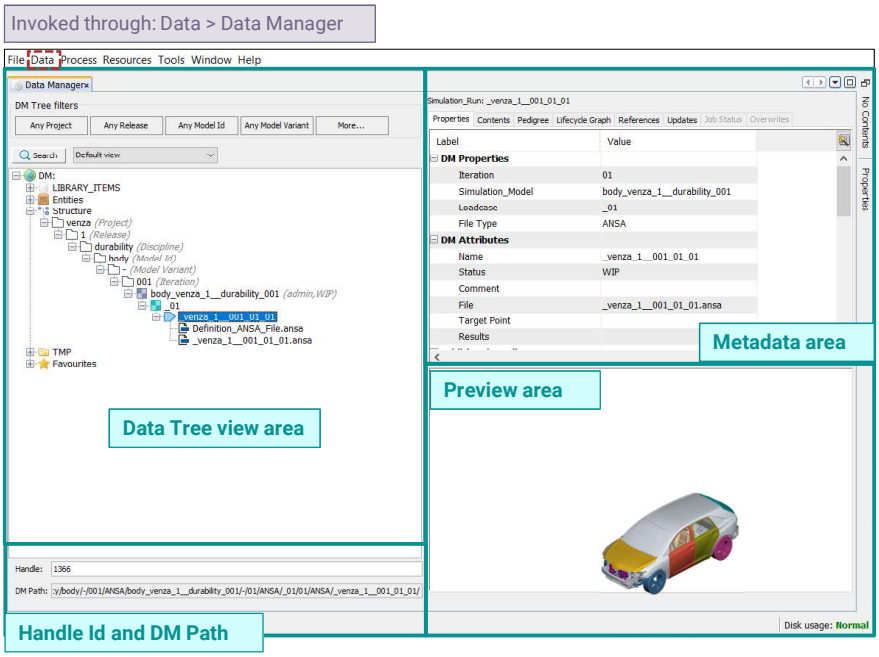
  These are customizable data types of auxiliary data objects (ex. Header files, Dummies, Material databases, etc.).



- **Files/Folders**

  The Properties of these data types are the File Name and DM Path. The Properties and Attributes these data types are fixed and are not customizable. Characteristic examples of simple file/folder data objects are the script files and external libraries.

Data Manager – Window Layout



The **Data Manager** window is the main tool used to access and browse the contents of the database.

In the **Data Tree View area** the contents of the database are displayed in tree format.

The **Metadata area** reports information about the selected data object on the left.

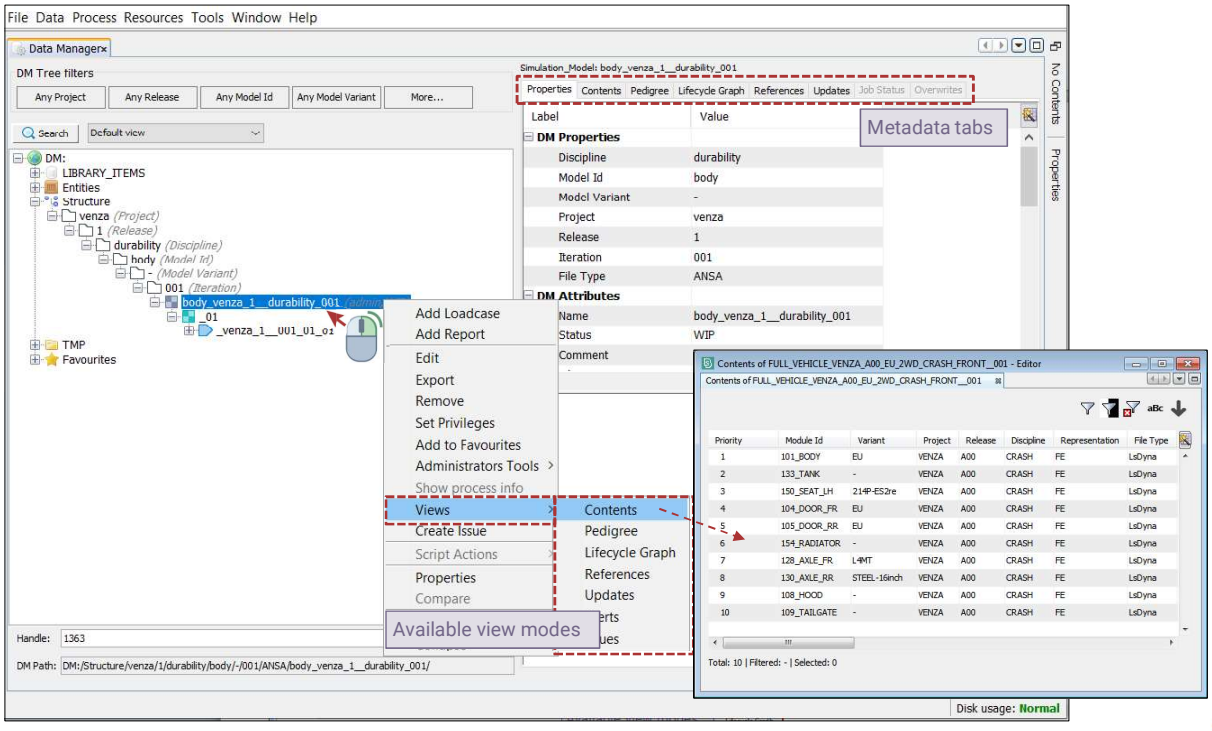
A snapshot of the selected data object is displayed in the **Preview area**, if this is available.

The unique ID number of the selected Data Object is reported in the **Handle** field.

The path of the selected Data Object in the tree structure is reported in the **DM Path** field.

View Modes

Each data type has certain view modes which can be displayed either as **Metadata tabs** or through the **Views** option in the context menu of the data object.



SPDRM
WORKFLOW MANAGER

Process Design

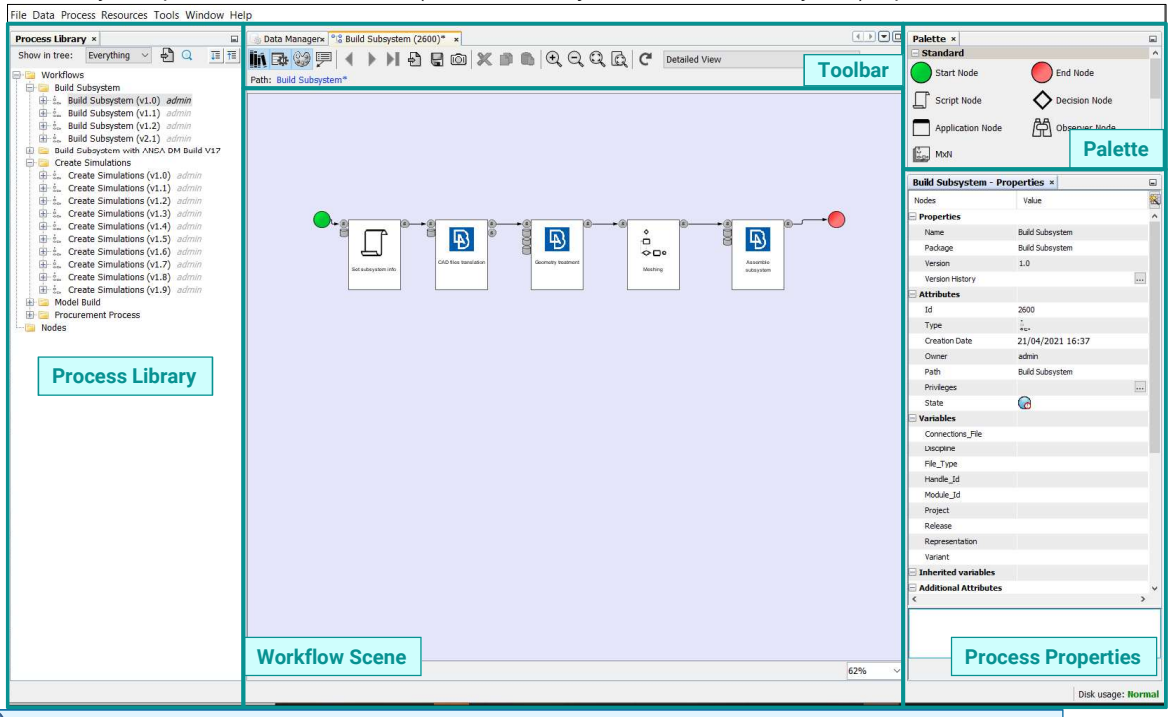
BETA
SIMULATION SOLUTIONS

Table of contents

Introduction to process design through the SPDRM Client

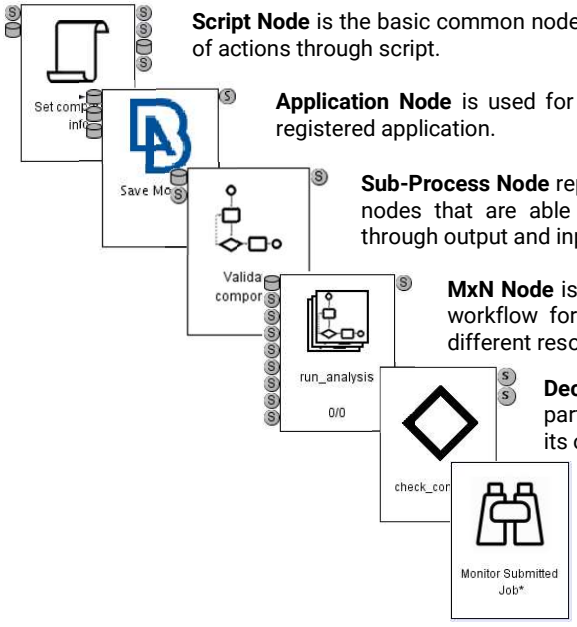
A process is a group of nodes, which are connected with each other forming a workflow. A node can contain other nodes in the form of a process; this node is a sub-process. The design of processes is implemented in the Workflow scene window.

All workflows can be saved in the Process Library as Definitions. They can be retrieved then from this library in order to create instances of a workflow in the working space of clients connected to the SPDRM software. The Workflow Definitions can also be combined, by using script functions that are available in the SPDRM Scripting API, to create new workflow instances on-the-fly. The ability to export Definition workflows (in the form of *.json files or *.ser binary files) is provided.



Go to Process > New, to open a new empty workflow scene.
Go to Process > Process Library, or press the respective button in the toolbar to access saved workflows.

Node and Input/Output Types



Script Node is the basic common node which is used for the execution of actions through script.

Application Node is used for actions that require the execution of a registered application.

Sub-Process Node represents a multiple-step task. It can contain sub-nodes that are able to communicate to the root (master) process through output and input streams.

MxN Node is a special sub-process node used to instantiate the same workflow for M different input data sets, optionally delegated to N different resources.

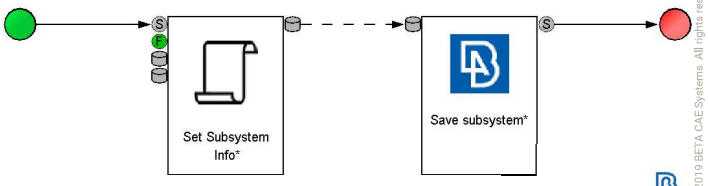
Decision Node is a special node used to drive the process towards a particular path, based on a condition statement. The node fills one of its output slots based on whether the statement turns True or False.

Observer Node is a special node used to monitor of a job submitted to HPC. Two types of observer node exist, **Script** and **Application Observer Node**, each of which embodies the functionalities of the respective simple ones, alongside their ability to enter idle mode, as long as they monitor the progress of the submitted jobs.

Supported types of variable for **Input** and **Output** slots are:
String, **Float**, **Integer**, **Boolean**, **Date**, **Single file**, and **List of files**.

Data can be provided to a node either through a connection with another node, or through the DM.

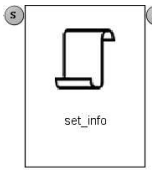
Files are communicated between nodes through dash connection lines, while the continuous connection lines represents the transfer of signals.



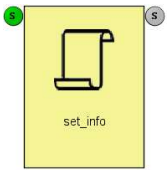
Color Coding of Nodes

A default color scheme is in place in SPDRM, that allows the user to quickly and easily identify the state of each process, node and inputs/outputs. During the execution of a process, the colors of a node and of its slots are changed, reflecting the transition from one state to another.

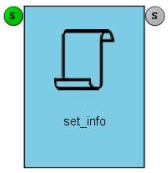
Basic node states



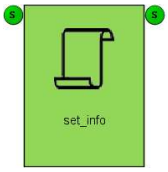
Not ready: Input Slots data are not available. Node cannot be executed.



Ready: Input Slot data are available. Node is ready to be executed.

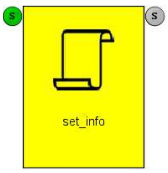


Running: Node is currently executed.

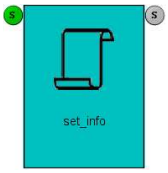


Finished: Node has finished its execution.

Additional (optionally acquired) node states

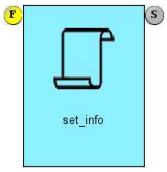


Scheduled or Paused: Input Slot data are available. Node is ready to be executed but it is scheduled to start on a particular time in future, or
node has been paused by the user (only for application nodes).



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

Observing: Node is in observing state and receives information about the progress of the submitted jobs. *Submitted Jobs* option from the context menu when in that state provides those information.



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

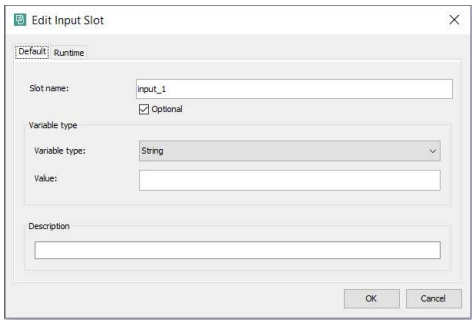
Color Coding of Slots

There are two types of input slots available:

Obligatory input slot, that must be filled in order for the node to be executed.

Optional input slot, that is not required to be filled in order for the node to start.

An input slot can be defined as obligatory or optional, by the *Optional* checkbox in the *Edit Input Slot* window. If the optional input slot is filled when the node is executed, then the slot becomes green, otherwise it remains as before (light purple) and the process continues.



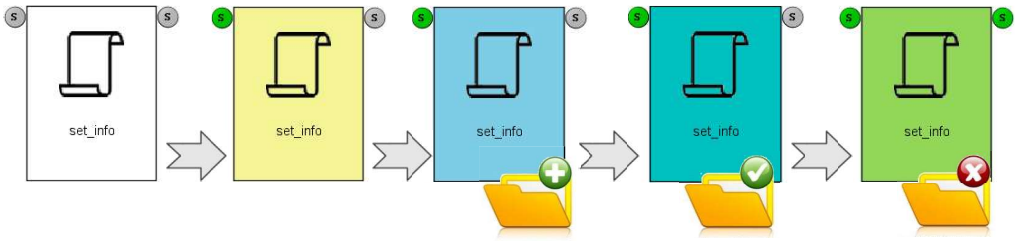
Input Slots:
- Data are not available (Obligatory slot)
- Data are not available (Optional slot)
- Data are available and ready to use

Output Slots:
- Data have not been provided by the node yet
- Data are available but they have not been released to the next node
- Data are available and they have been released automatically to the next node

Node Execution Directory

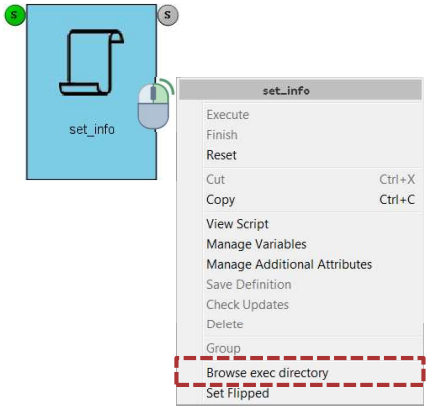
During the execution of a node, SPDRM creates a unique temporary directory, the Node Execution directory, for the communication of data between the client and the SPDRM database.

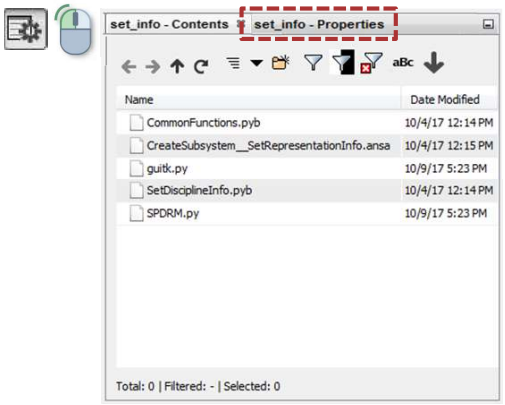
The Node Execution directory is automatically created when the execution of node starts, and it is automatically deleted when the node is finished and all of its output slots have been released.



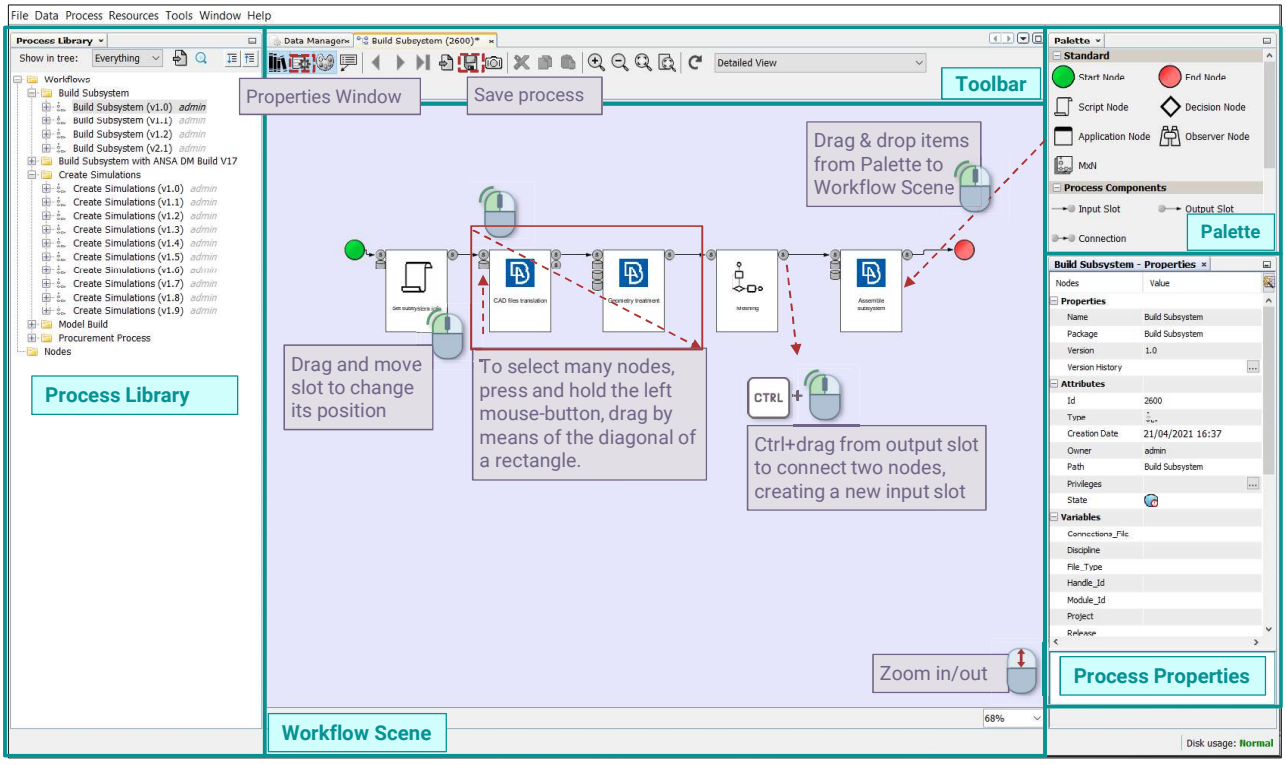
A scheduled node has not yet an execution directory.
A paused node retains its execution directory.
A polling input node has not yet an execution directory.

It is accessed through the **Browse exec directory** function of the context menu of the node, or through the *Contents* tab of the *Properties* sheet.



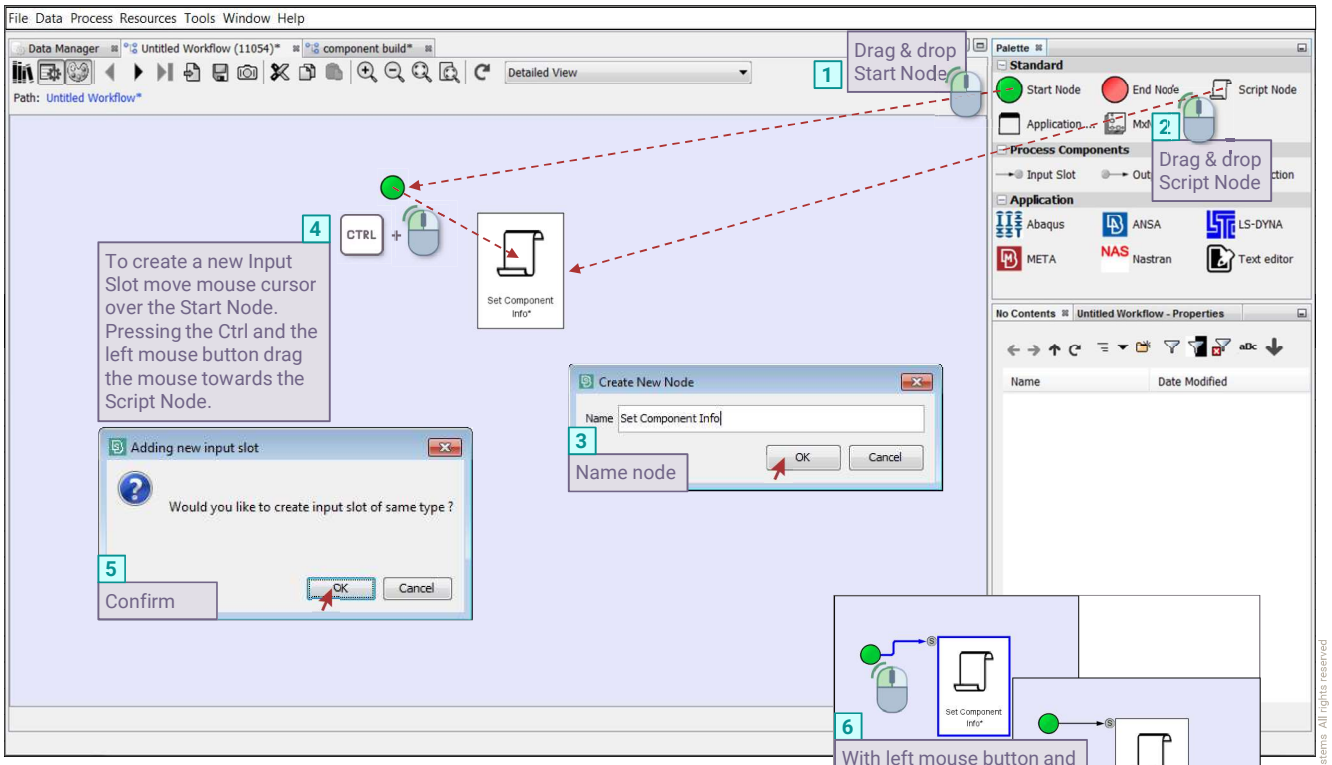


Basic actions during process design



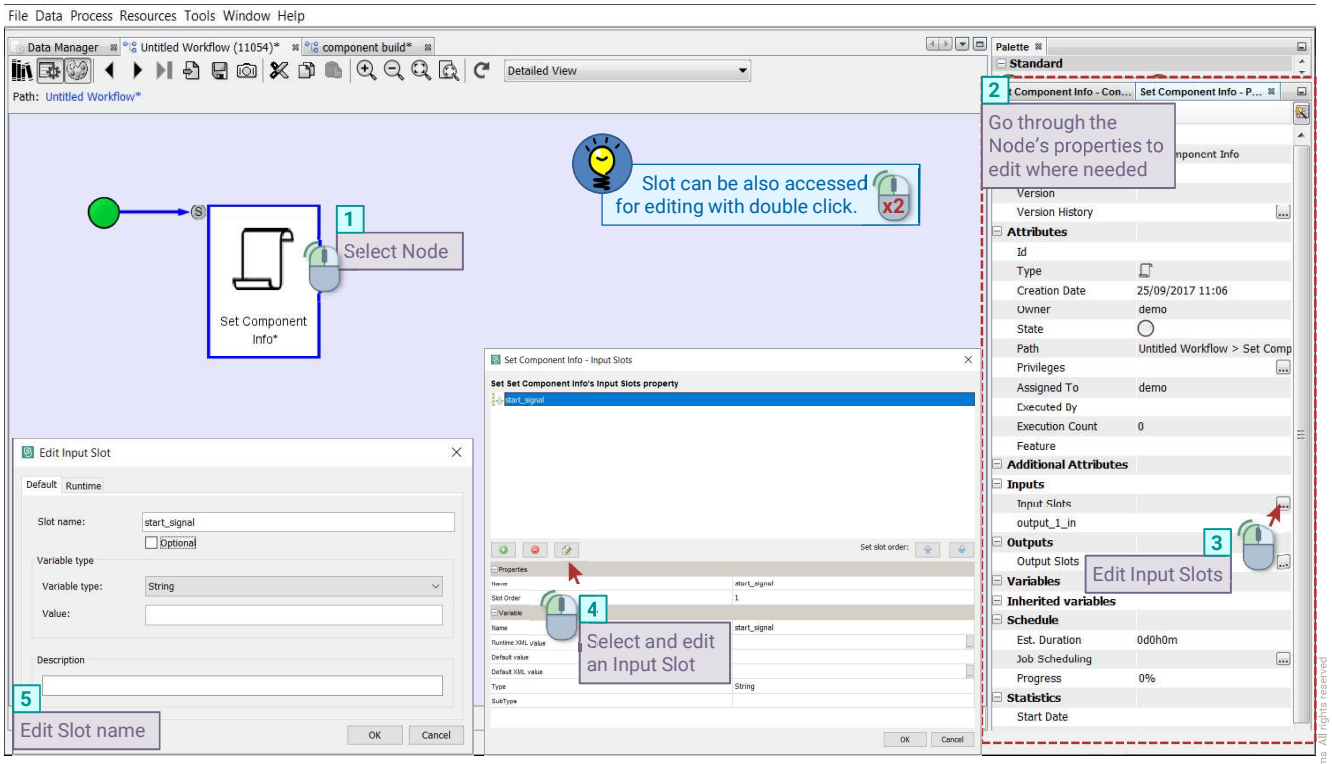
Go to Process > New, to open a new empty workflow scene.
Go to Process > Process Library, or press the respective button in the toolbar to access saved workflows.

Create and connect nodes



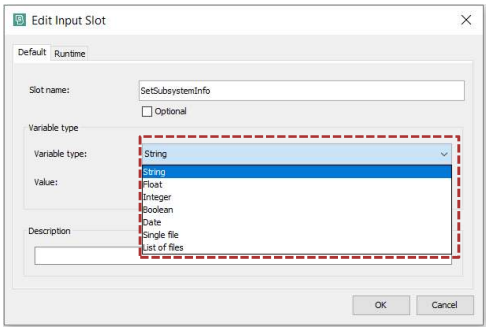
Copy/Cut and **Paste** capabilities for the nodes are available via their context menu.

Edit nodes - add/edit Input Slots



Activate **Auto Execute** option for the node to be executed automatically as soon as its input data are available.
Activate **Auto Finish** option for the node to be Finished when its job is done and omit the pre-finished state without user intervention.
Activate **Write to Output Window** option to allow printing of log messages in the output window, during the execution.

Input Slot definition - Single file type



Supported types of variable for **Input** slots are:

String, **Float**, **Integer**, **Boolean**, **Date**, **Single file**, and **List of files**.

A **Description** of the slot can be provided in the respective field for all types of slots.

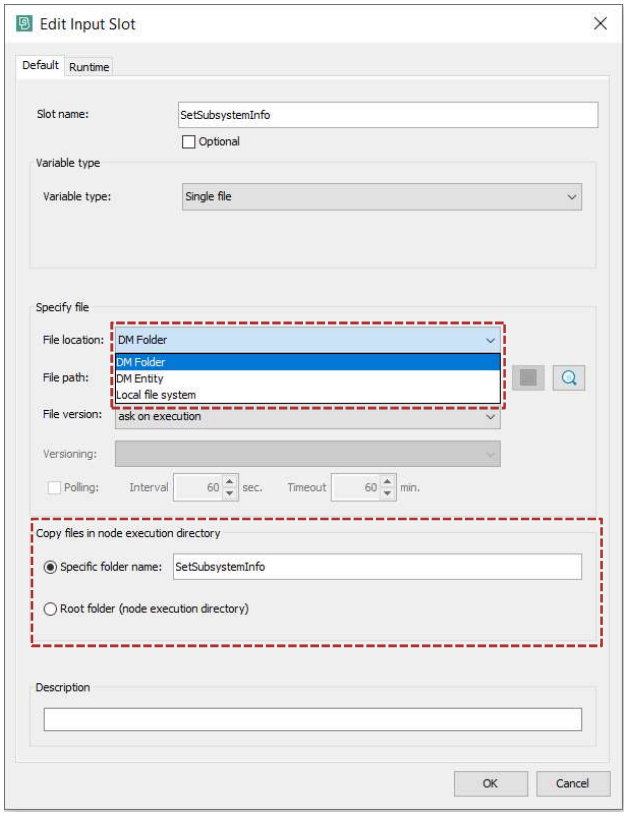
In case of type **Single file**, a file can be provided from:

a **DM Folder**, a **DM Entity**, or the **Local file system**.

Whatever the file location, the selected file will be downloaded in the node execution directory.

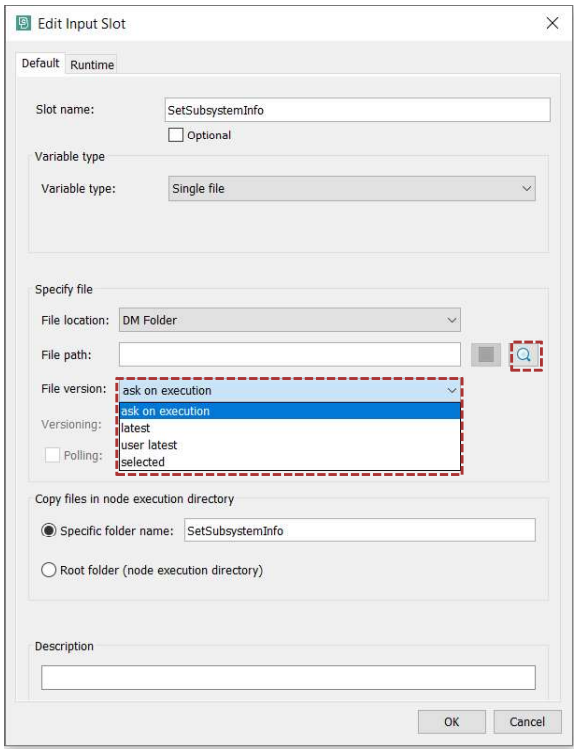
Activate the **Specific folder name** option, so as a new directory to be created inside the node execution directory, to host the downloaded file, upon execution. The name of the directory will be the one given in the edit field.

If **Root folder** option is active, the file will be downloaded directly in the node execution directory.

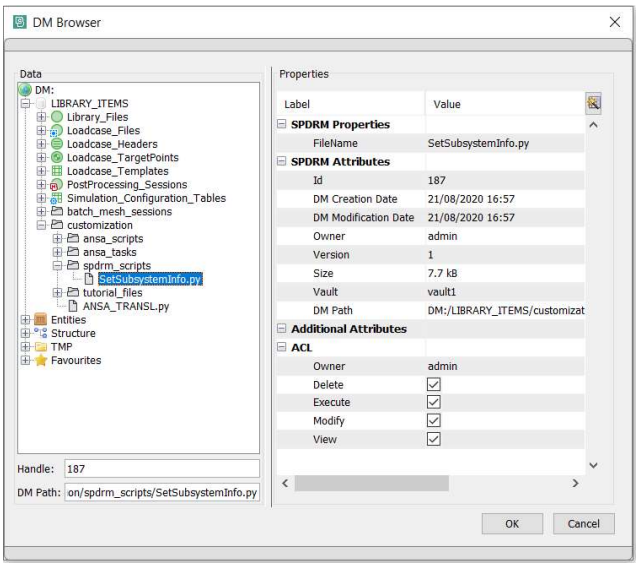


Input Slot definition - Variable type: Single file - File location: DM Folder

The file provided to the node by the input slot can reside in the SPDRM repository.



Pressing the **Browse** button the user can navigate into the folders of the DM and select the desirable file.



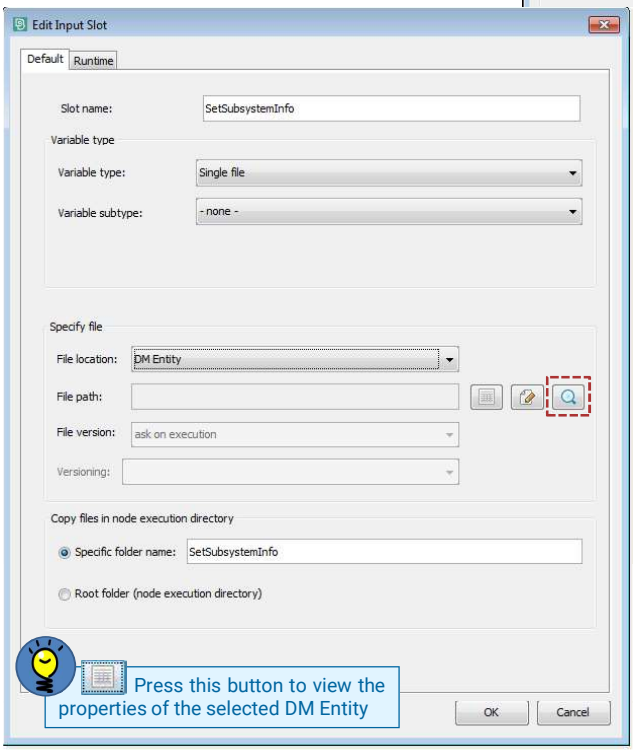
The version of the file that SPDRM will access, if multiple versions detected during the execution, can be:

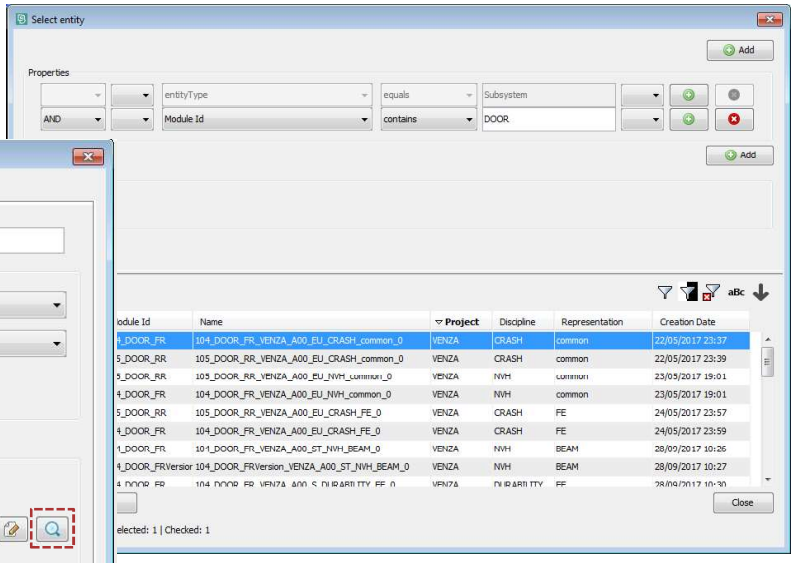
the one selected by the user: **ask on execution,** the **latest,** the created by the particular user latest: **user latest,** the **selected** one during the design of the workflow.

Input Slot definition - Variable type: Single file - File location: DM Entity / Local file system

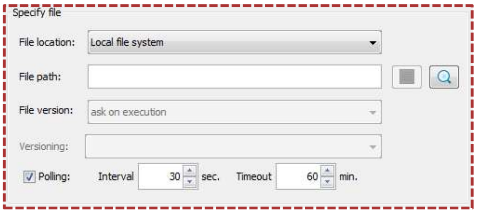
The file provided to the node by the input slot can be the attached file(s) of a DM Entity.

When **DM Entity** is selected, pressing the **Browse** button the user can search according to criteria and select the desired one.



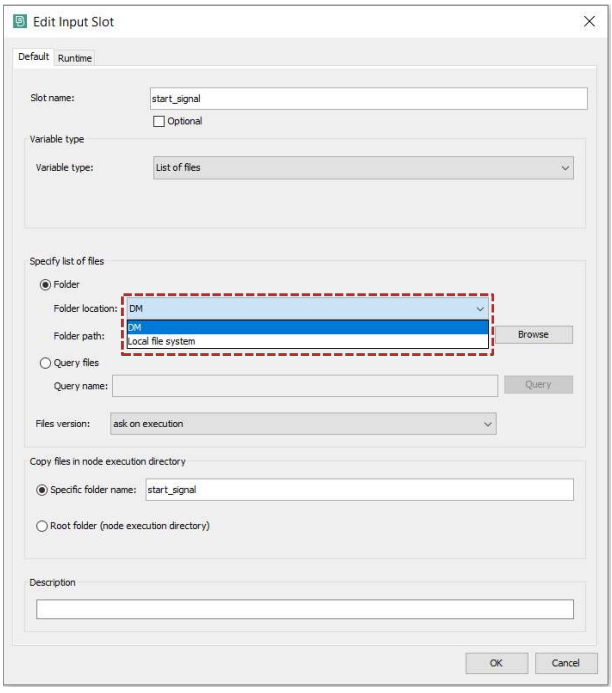


When **Local file system** is selected the user can navigate and select a file from the local disk storage. The option to poll for particular input data with a user defined interval is possible. When data become available the node will be ready to run.

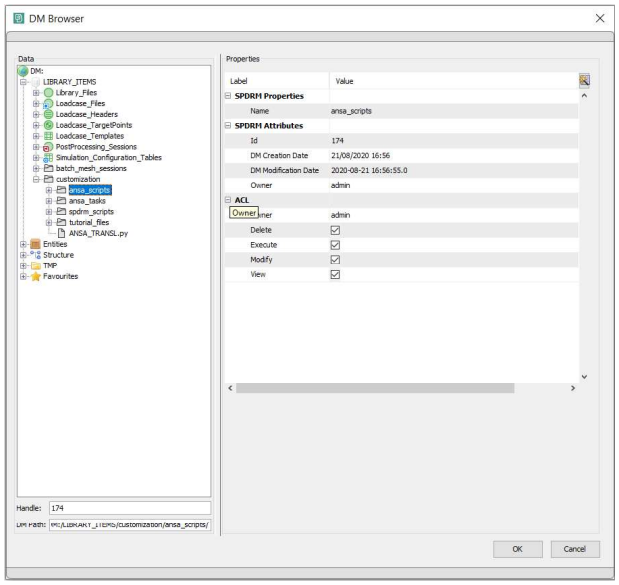


Input Slot definition - Variable type: List of files

On the course of the process it may be imperative for a Node to use more than one files as input (e.g. in the case of MxN Nodes). In this case the concept of **List of Files** is introduced in order to be used as an Input slot.



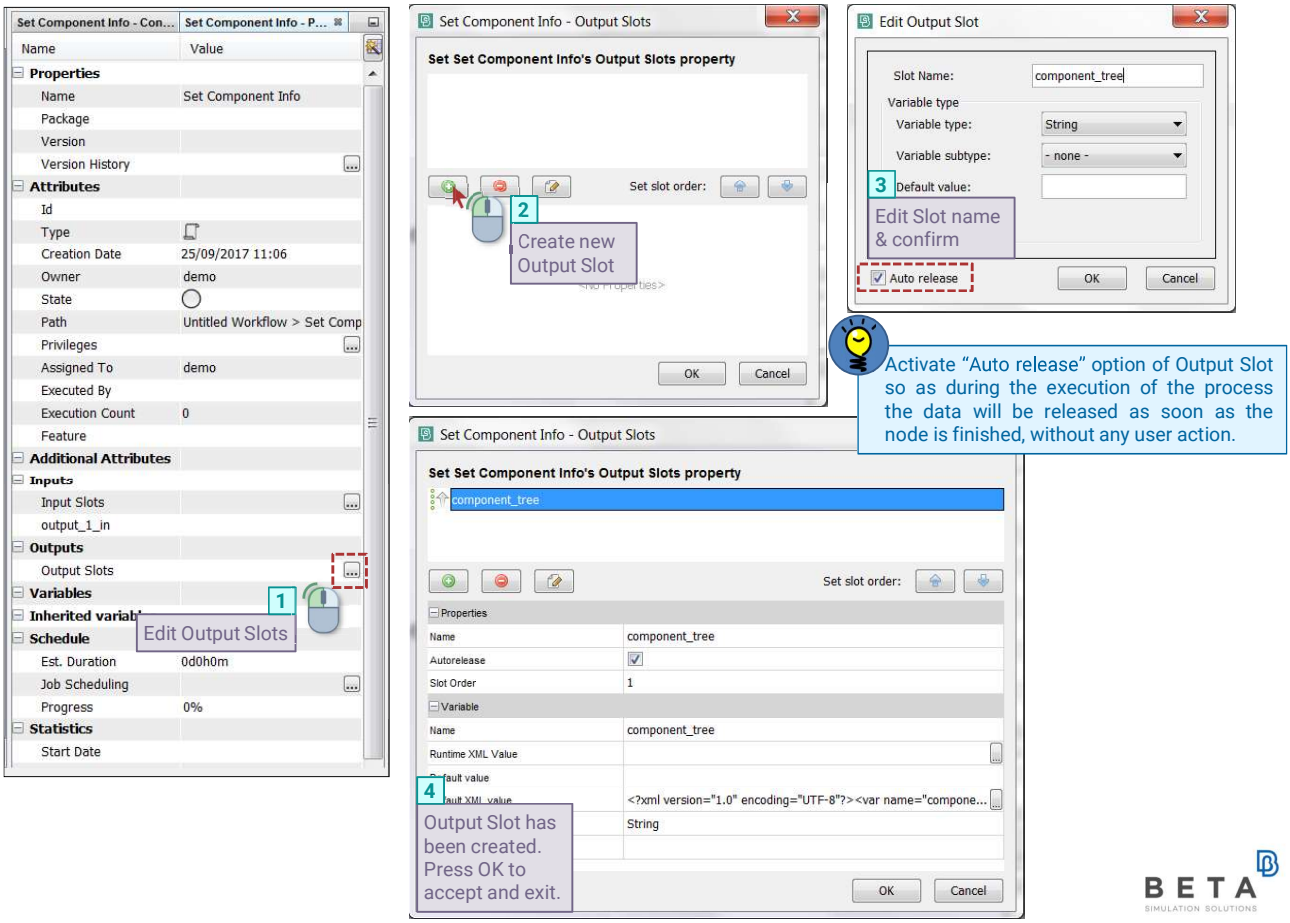
A List of files can be provided from: a **DM Folder** or the **Local file system**.



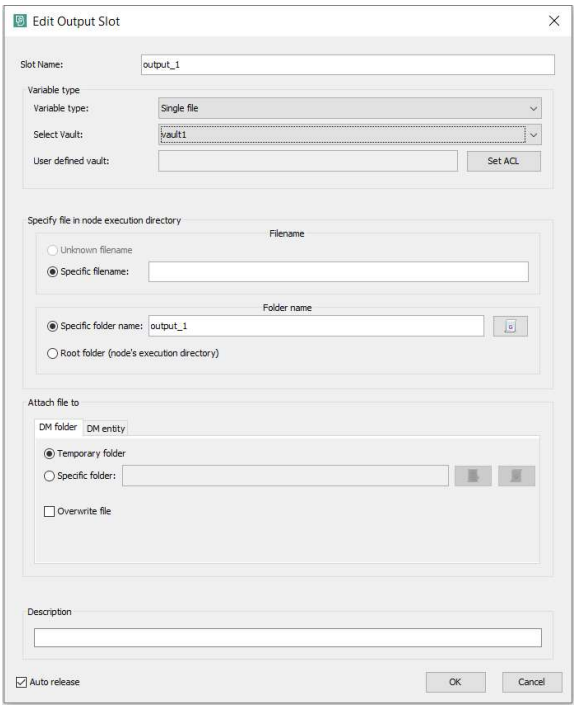
Press the Browse button to navigate in the DM or the local file system, according to Folder location option, and select the desired directory.

Activate option **Query files** to fill the input slot with files that fulfill an existing Query in the database. The Query can be selected by its name from within a list that appears when user presses the **Query** button.

Add/edit Output Slots



Output Slot definition - Variable type: Single file/List of files



In a similar manner the Output slot with variable type: List of files can be defined

In case that more than one vaults have been set during SPDRM installation the respective Vault path must be defined. This can be parametrically defined by selecting **User defined vault** and typing “*${_name of process variable}*”.

Access control can be optionally set on the output slot file pressing the **Set ACL** button.

A predefined filename can be given to the produced file, which can be located either directly in the node execution directory or in directory with a predefined name.

Press this button, to parametrically set the folder name, in dependence with the system set variables (if any).

The file can be attached to a **DM folder**, a **DM Entity**, or a **New DM Entity**.

Select DM folder and press button to attach a file inside a specific folder of the DM. Alternatively, the file can be attached to a temporary folder. Press to parametrically set the folder name.

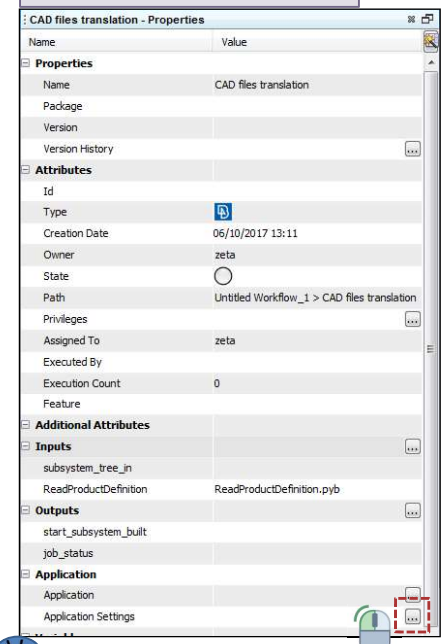
Select DM Entity and press button to attach the file to a selected DM Entity.

Select New DM Entity to create a new subsystem and attach the file to it.

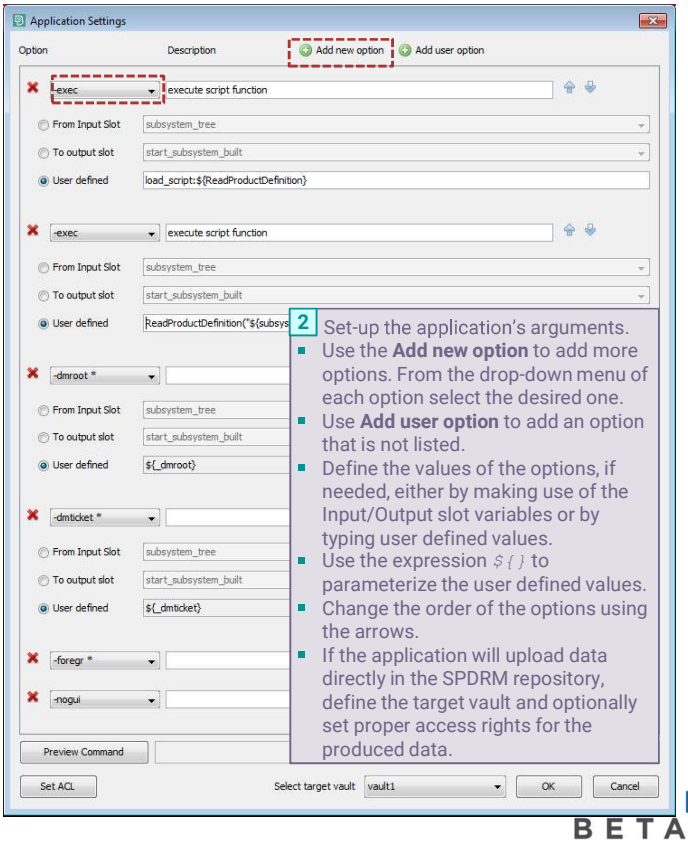
Application nodes: Set up the application settings

An Application Node can have several options that will affect the way the application will be executed. All available application options in the node definition depend on the set-up of the application itself (Resources > Manage Registered Applications).

1 Access the application settings from within the **Properties** window



Options that are marked as **Required** in the registration of the application, are shown automatically in the **Application Settings** window and cannot be removed by the user.



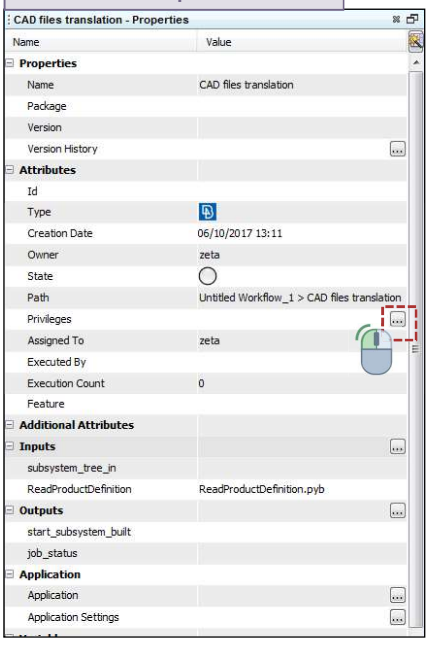
2 Set-up the application's arguments.

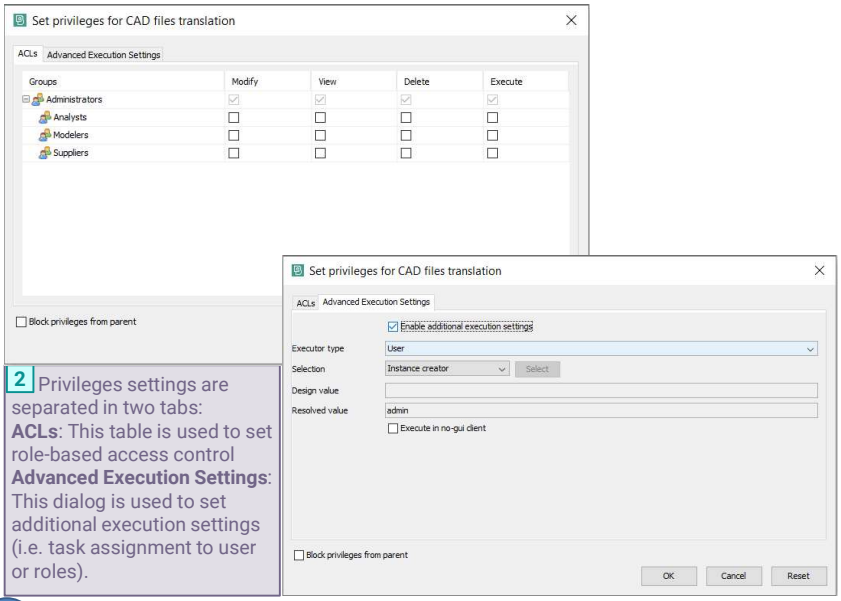
- Use the **Add new option** to add more options. From the drop-down menu of each option select the desired one.
- Use **Add user option** to add an option that is not listed.
- Define the values of the options, if needed, either by making use of the Input/Output slot variables or by typing user defined values.
- Use the expression *${}* to parameterize the user defined values.
- Change the order of the options using the arrows.
- If the application will upload data directly in the SPDRM repository, define the target vault and optionally set proper access rights for the produced data.

Privileges definition

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

1 Access the **Privileges** settings from within the **Properties** window





2 Privileges settings are separated in two tabs:
ACLs: This table is used to set role-based access control
Advanced Execution Settings: This dialog is used to set additional execution settings (i.e. task assignment to user or roles).

The additional execution settings (if any) override the ones displayed in the **Execute** column of the **ACLs** tab.

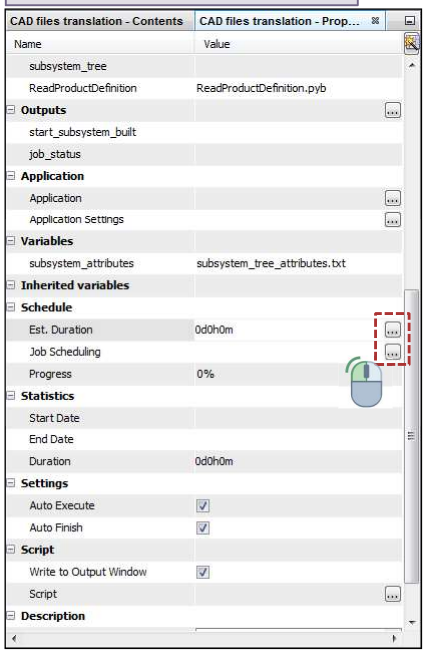
The privileges that are defined in a workflow, sub-process or MxN node can be (optionally) propagated to its children nodes.

Activate the option **Block privileges from parent** to freeze the privileges settings for this node and prevent any override from changes done in parent nodes.

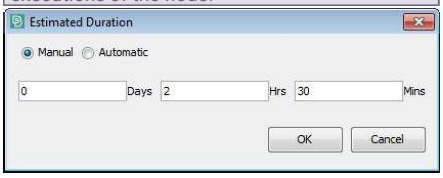
Estimated duration - Schedule execution

A node can host information about the duration and the progress of its execution. This information is obtained during the execution and the respective fields in the Properties window are updated. Nevertheless, during the design of a workflow the designer can optionally define an estimated duration for the node(s) in order to make a time scheduling for the workflow. Additionally, the designer of the workflow can schedule the time of execution of a node.

1 Access the **Est. Duration** settings from within the **Properties** window

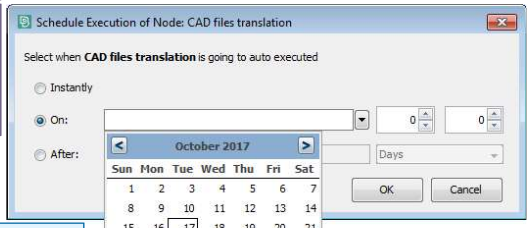


2 Two modes are available for the estimation of the node duration, the **Manual** and the **Automatic**. In the first case the user can set a particular duration in Days/Hours/Minutes. In the latter case the estimated duration of the node is calculated by SPDRM based on previous executions of the node.



3 Access the **Job Scheduling** settings from within the **Properties** window

4 In the Schedule Execution window the node can be set to start as soon as its input is available **(Instantly)**, or **On** a particular timeslot, or **After** a particular time interval.



A scheduled node with option **After**, will calculate the time of execution, the moment the node would become *Ready*.

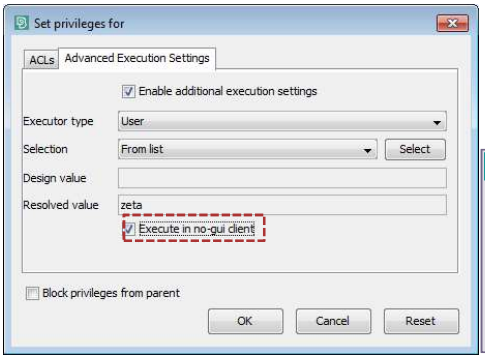
Assign tasks to No-GUI client

The assignment and execution of tasks in a No-GUI SPDRM client is possible. This capability can be used for the execution of heavy and time-consuming jobs, on remote resources (even on different sites of an organization), such as the job submission of a Simulation Run to the solver, the polling of the solver results, and the post-processing of the results.

The default setup enables the configuration of the SPDRM No-GUI Client in such a way that it runs on a machine different than the one that hosts the SPDRM application server, to avoid its overload. This can be done by adding a specific key in the server's configuration file.

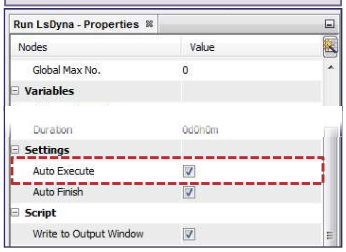
When a node that is set up to run in no-gui client is executed, SPDRM tries to detect an already running no-gui client launched by the current user. If none is found, SPDRM will launch a new no-gui client session in order to execute the node. So, in case of two or more such consecutive nodes, all nodes after the first one will use the opened by the first node no-gui client.

The launched by the previously described process no-gui client session will close when the no-gui session idle timeout session is met. The idle timeout can be configured in the no-gui client's configuration file.



1 When the process is on the design phase, the task to be executed in the No-GUI client is marked with the option Execute in no-gui client, in the *Set privileges for...* window of the respective node.

2 In the *Properties* sheet of the node the **Auto Execute** option is activated



Additionally to the default setup, any user can launch, locally or remotely, a No-GUI client so as to execute particular tasks by providing the respective Node Id. In this case, custom GUI, generated by script functions within the executed node, can be utilized so as the user to give a possible initial input to the process that will be executed from then on.

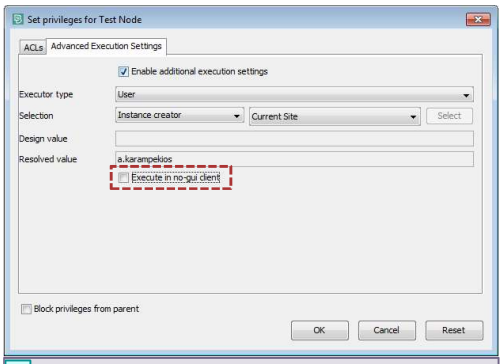
Assign tasks to No-GUI client using BETA Apps Launcher

If one or more BETA Apps Launcher have been configured, the *Advanced Execution Settings* window will be enriched with a special drop down menu that is activated when a node is configured to run in no-gui client.

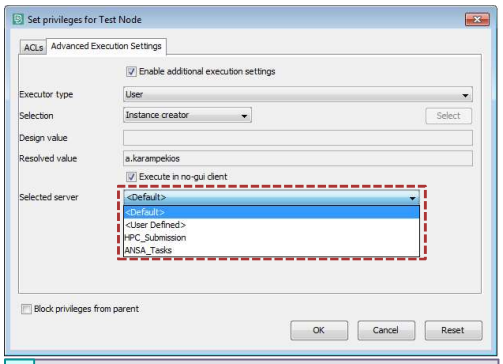
The new menu that appears alongside the **Selected server** label enables the selection of the BETA Apps Launcher where the no-gui client will be launched. The menu contains also the option to use the **Default** server or a **User Defined** one.

The **Default** option leaves to SPDRM the selection of the BETA Apps Launcher. SPDRM will select in that case the registered BAL based on the free memory of each server, if none of them has already launched a no-gui client on a previous node.

The **User Defined** option enables a text field that supports SPDRM process variables (e.g. ${process_var_1}). Thus, this options can be used for the parametrical definition of the BAL in which the task will be executed. This variable can be set in a previous step of the process, e.g. according to certain user input.



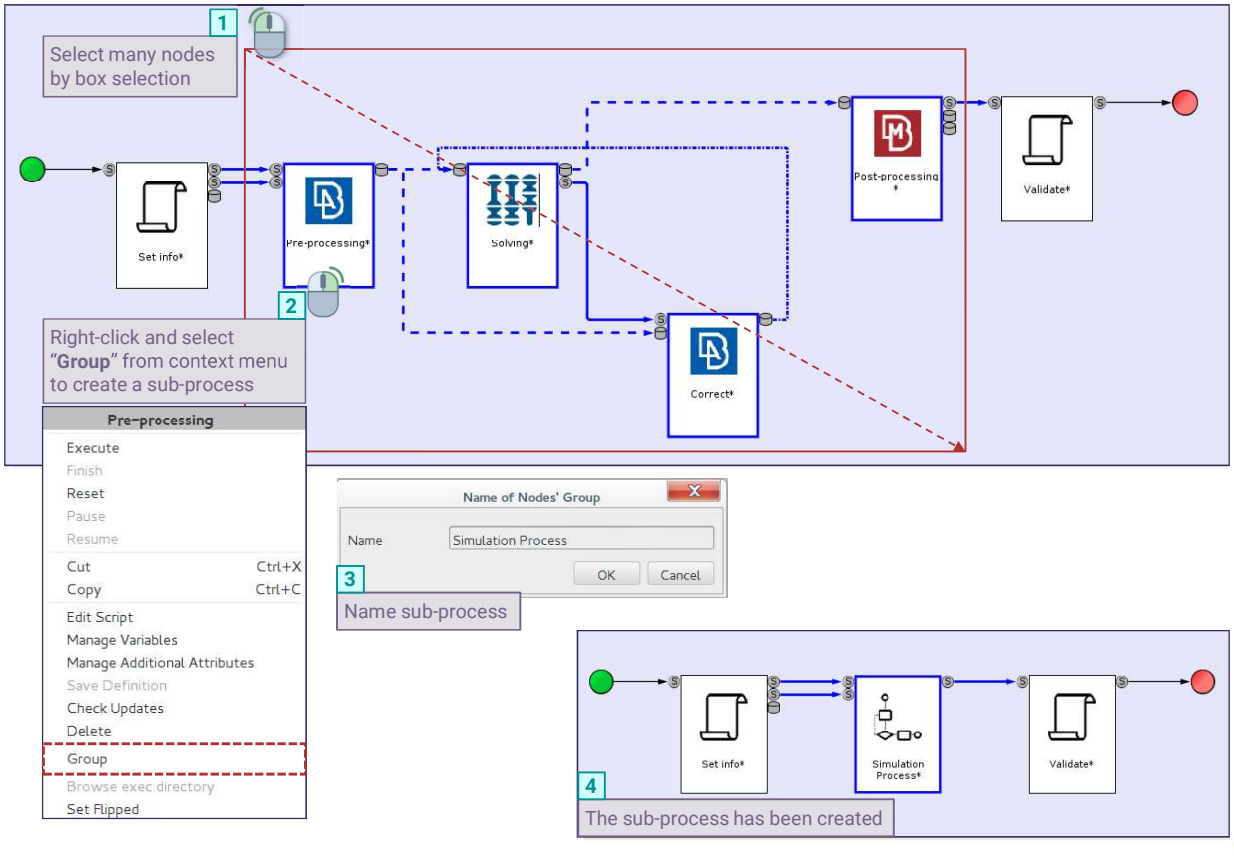
1 Before the selection of **Execute in no-gui client** option, the **Selected server** menu is hidden.



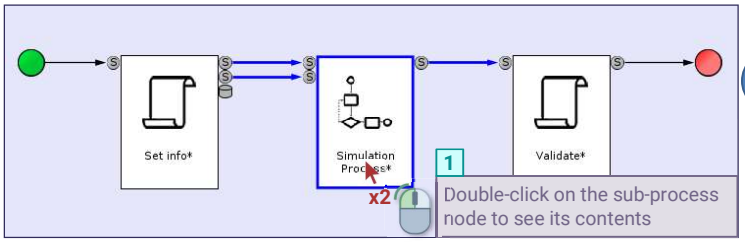
2 When the checkbox is selected, the user can select a pre-defined BAL or define parametrically the desired target server. HPC_Submission and ANSA_Tasks are on this example the registered BETA Apps Launchers.

The names of the BETA Apps Launcher that are displayed in the menu do not reflect the actual name of the servers, but an alias that has been given by the system administrator. Those alias make it much easier for the process designer to detect the server that is set-up to fulfill specific requirements.

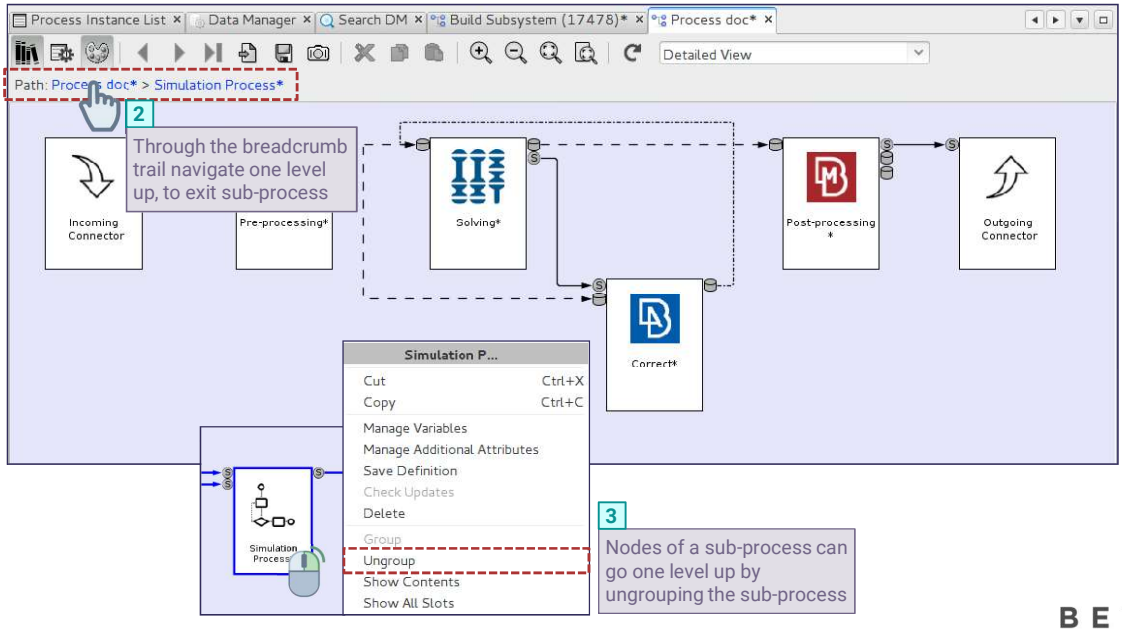
Create sub-processes



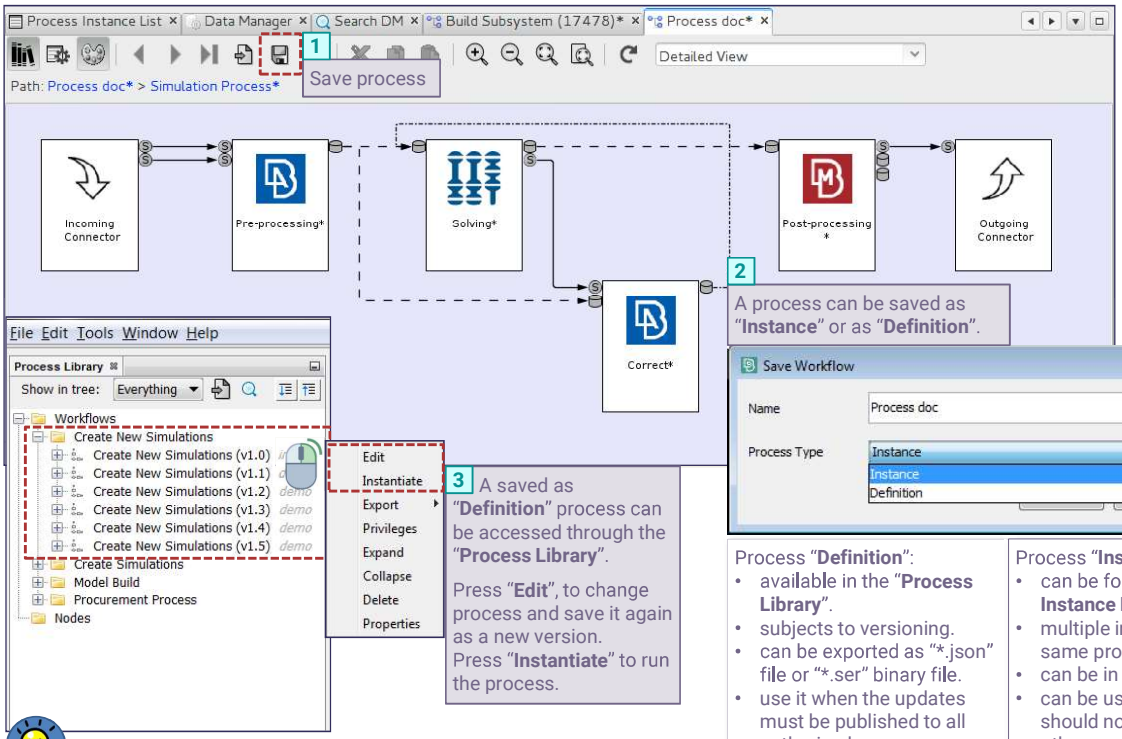
Handle sub-processes



Incoming & Outgoing Connector nodes that are visible inside a sub-process are not actual node entities, but visualization sketches that represent the connections of the sub-process with the rest of the workflow.



Save workflow



Process "**Definition**":
- available in the "**Process Library**".
- subjects to versioning.
- can be exported as "*.json" file or "*.ser" binary file.
- use it when the updates must be published to all authorized users.

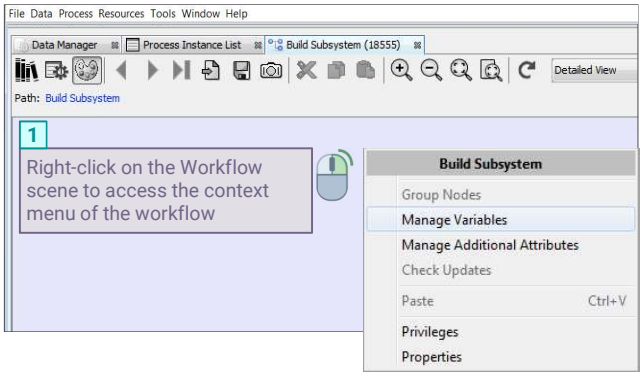
Process "**Instance**":
- can be found in "**Process Instance List**".
- multiple instances of the same process can exist.
- can be in Running state.
- can be used when updates should not be published to other users.

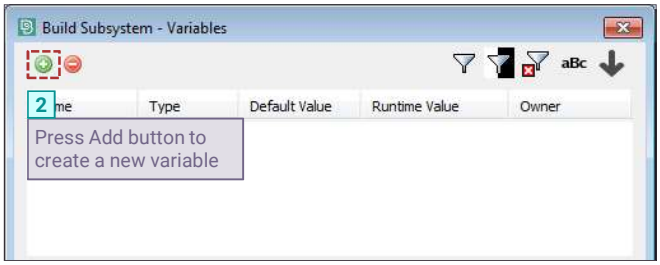
Nodes and Processes that are marked with the " * " sign have been changed and not yet saved.

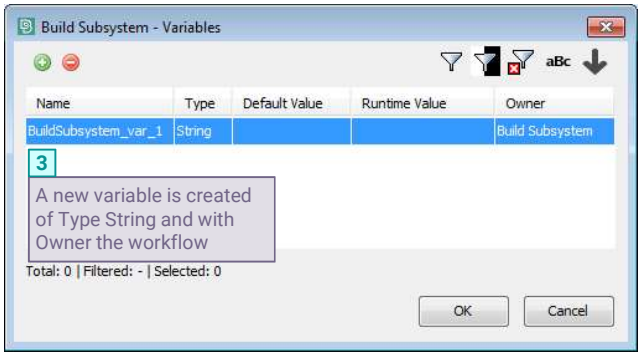
Saved workflows can be exported from the Process Library window by selecting **Export** from the context menu. Two types of file formats are supported, *.json and *.ser. The latter is a binary file.

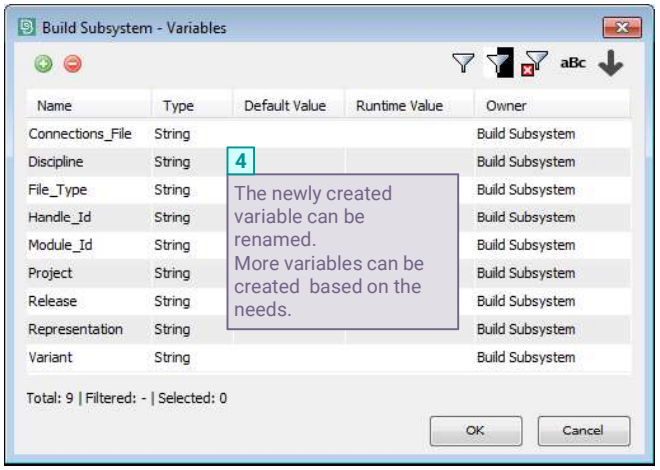
Manage variables

In order to facilitate the transferring of information between the nodes of a workflow with no need of connections and independently of the nodes' level, several variables can be defined on the workflow or/and on particular nodes. Variables are also used to simplify and parameterize the workflow.





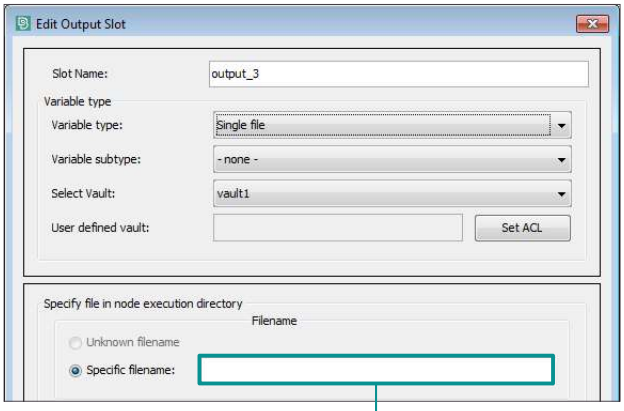




The variables can host values of various types: **String, Integer, Float, Boolean, Date**.

Parameterization of Process using the variables

Variables can be used for the parameterization of a process. For example, the value of an output slot of type 'Single file' can get a fixed predefined value or it can be linked to the value of a variable. The variable can get a value during the execution of the process or it can be determined through a script.



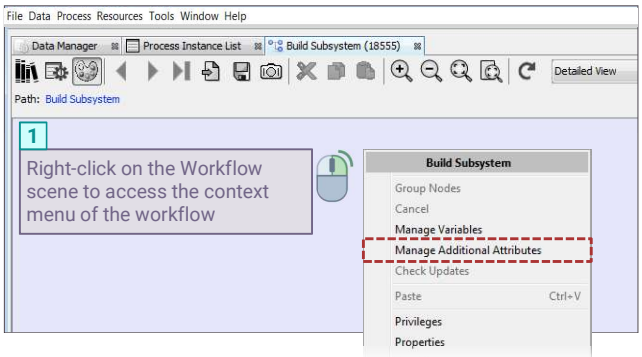
Variable — `${filename_var}` — **Fixed** — `Door_f1_D256.nas`

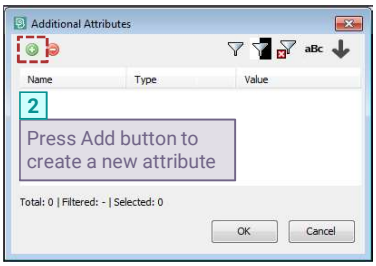
```
filename_var = 'door_f1_D256.key'
```

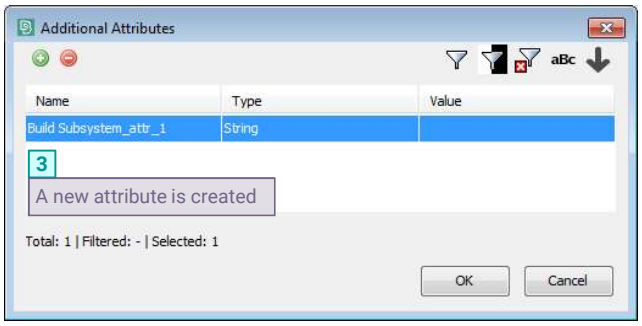
```
if Discipline == 'NVH':
        filename_var = 'door_f1_D256.nas'
elif Discipline == 'CRASH':
        filename_var = 'door_f1_D256.key'
elif Discipline == 'DURABILITY':
        filename_var = 'door_f1_D256.inp'
```

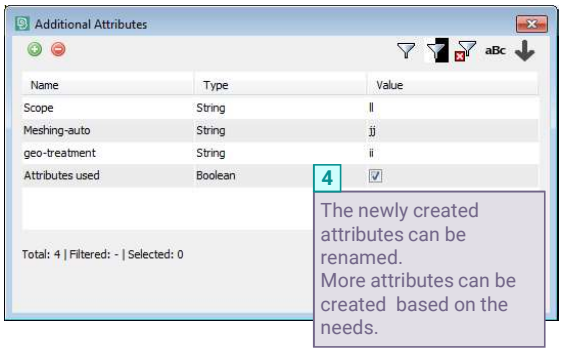
User attributes for nodes and workflows

The definition of user attributes for nodes and workflows is possible. These attributes are saved along the workflow when this is exported as meta-data.





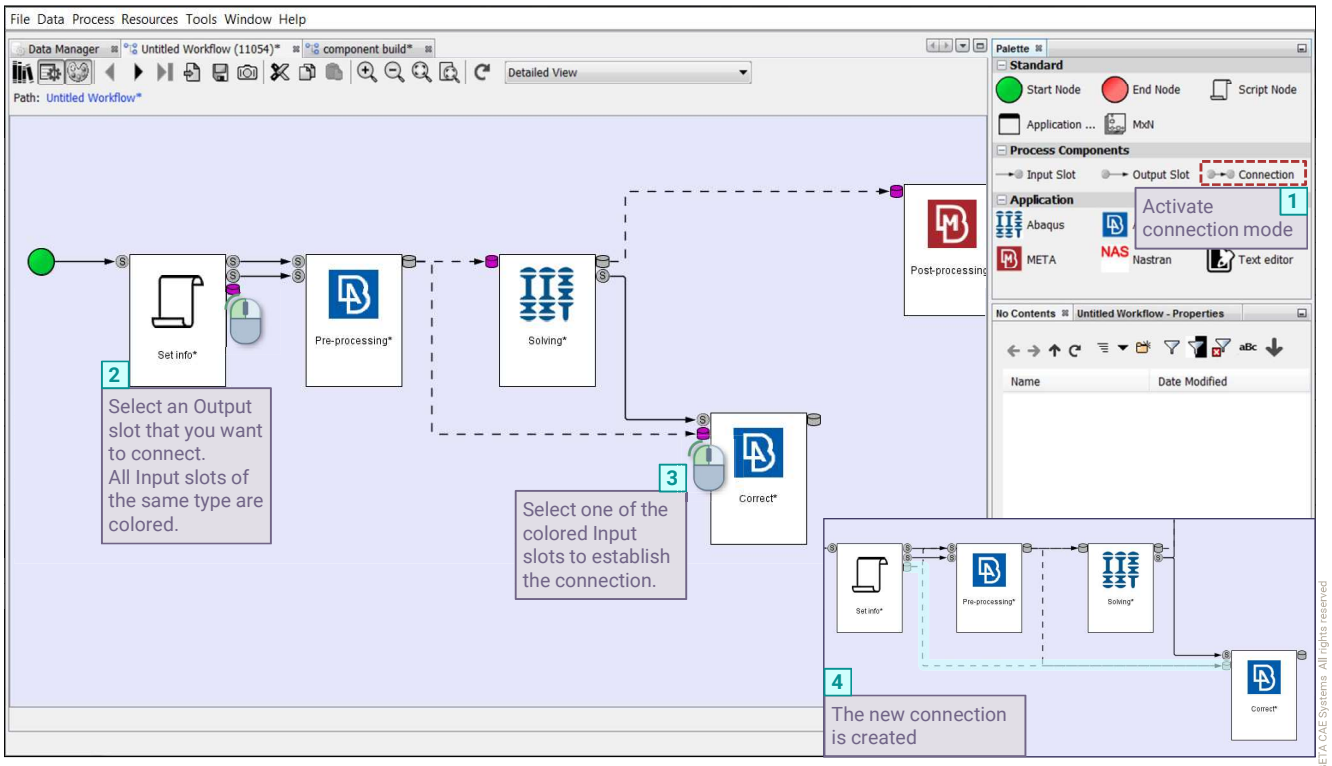




The variables can host values of various types: **String, Integer, Float, Boolean, Date**.

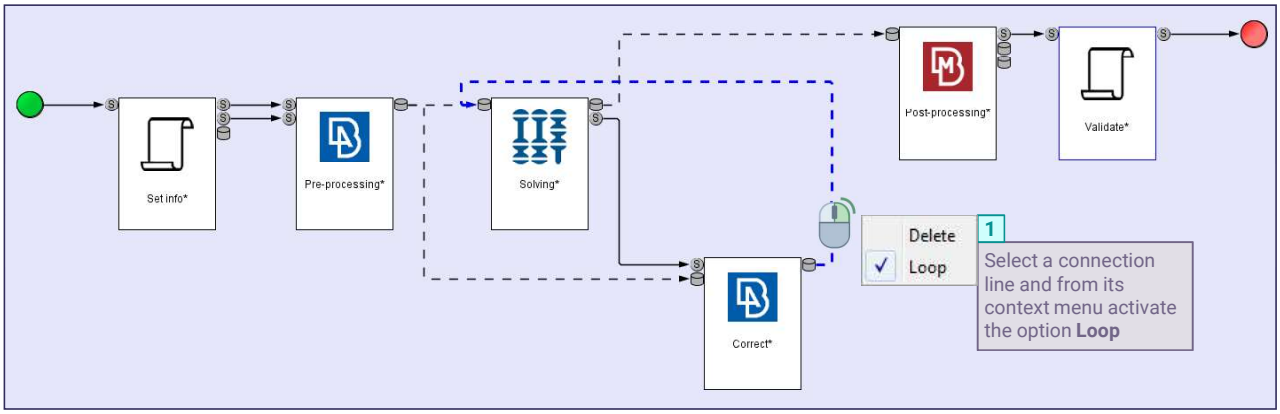
Additional capabilities – Connection mode

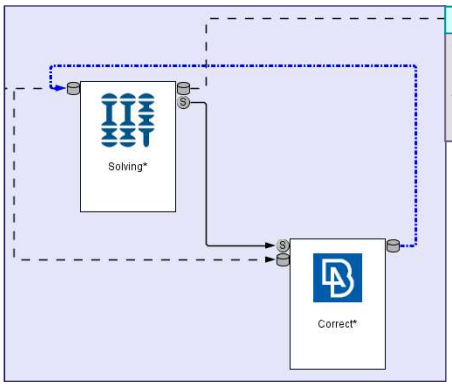
To facilitate further the design of a workflow and especially the connection of nodes SPDRM offers the **Connection mode**.



Additional capabilities - Loop connections

It is possible to create **loops** during the design of a workflow in order to direct a process back to a previous step.





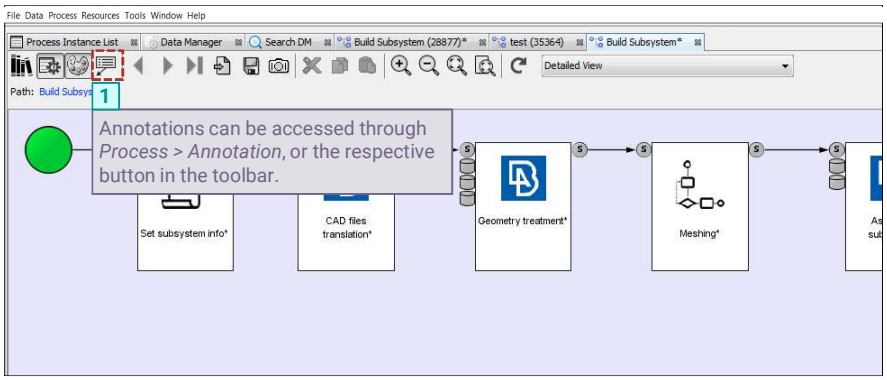
Loop connection lines can be changed back by deactivating the respective option from their context menu.

Connection lines of all types can be deleted by selecting **Delete** from their context menu.

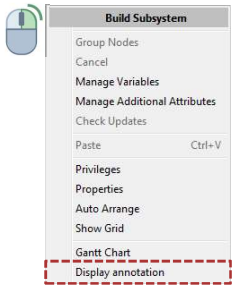
Additional capabilities - Annotations

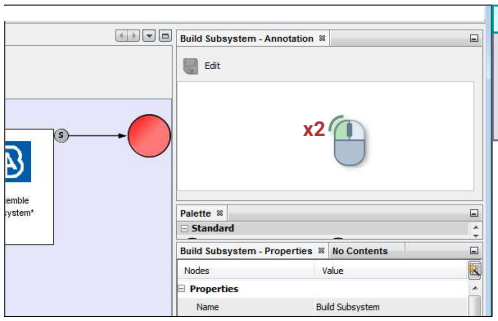
Annotations offer the possibility to display custom information related to a workflow.
The information that can be displayed is:

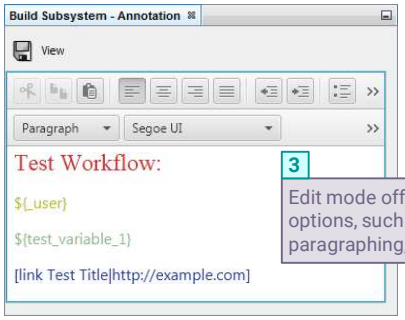
- Plain text information
- Built-in system variables
- Workflow variables synchronized with runtime values
- HTTP Hyperlinks



Annotations can be also accessed through the respective option in the context menu of the Workflow scene.

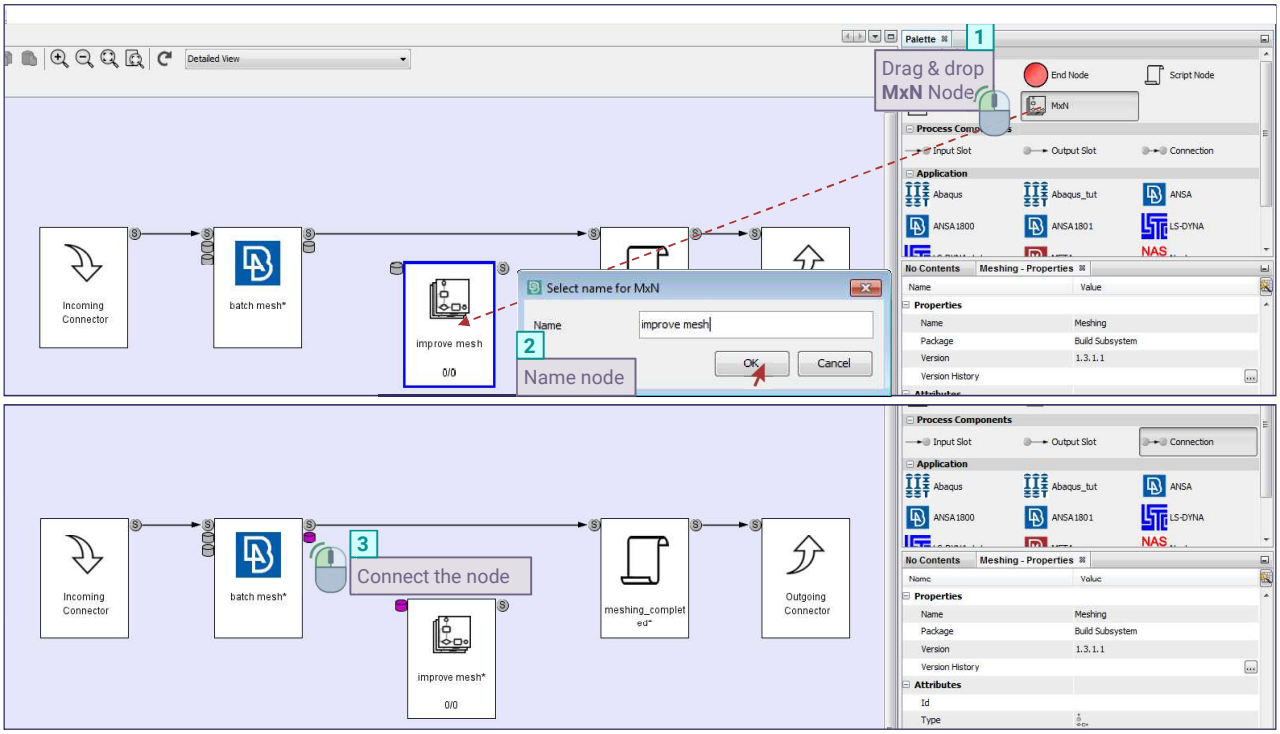




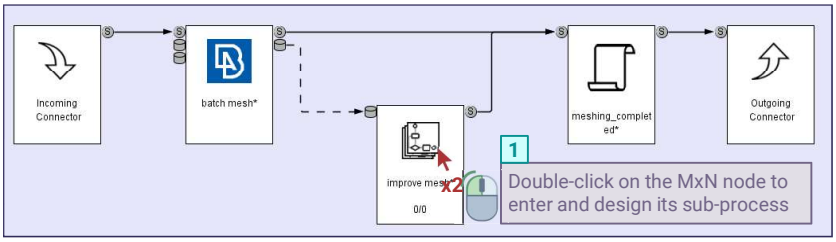


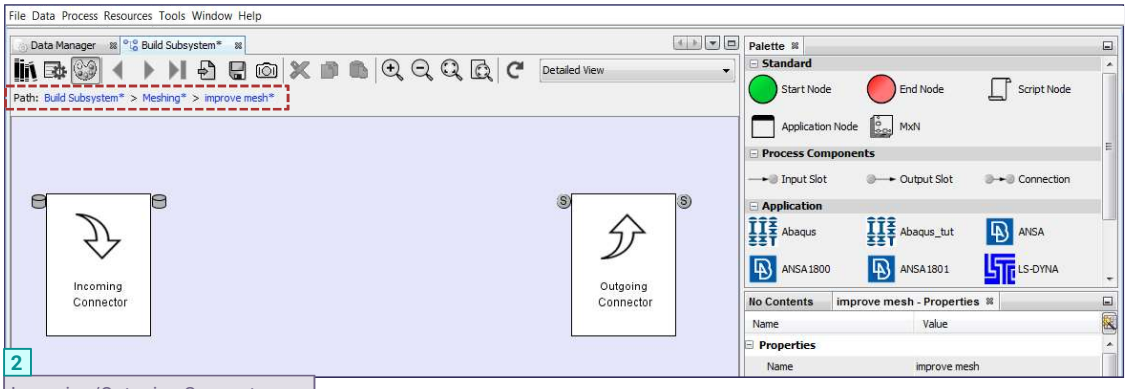
Creating MxN nodes (1/2)

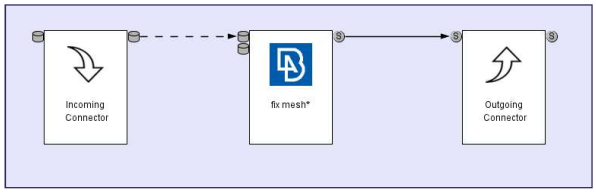
An **MxN Node** is a special sub-process node used to instantiate the same workflow for *M* different input data sets, optionally delegated to *N* different resources. The Input slot of such a node is always of type **List of files**.



Creating MxN nodes (2/2)

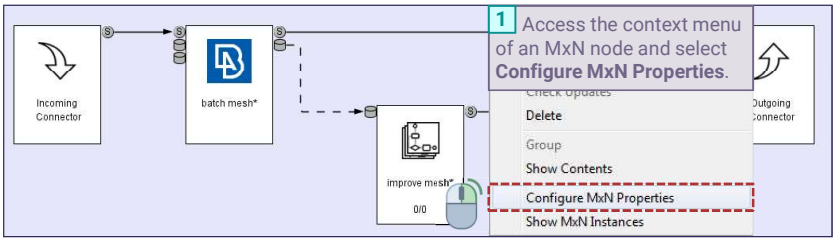


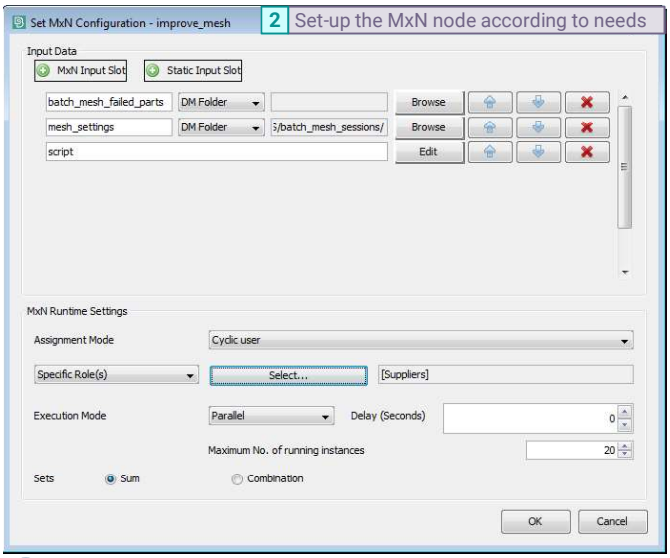




When an MxN node is executed, its containing process will be instantiated as many times as the number of files in its Input slot. Its instance will be fed by one of the files.

Configure an MxN node



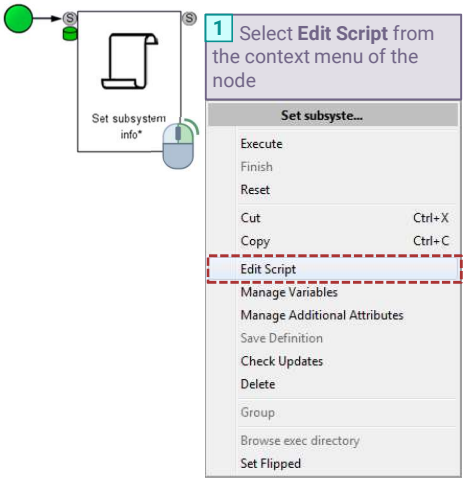


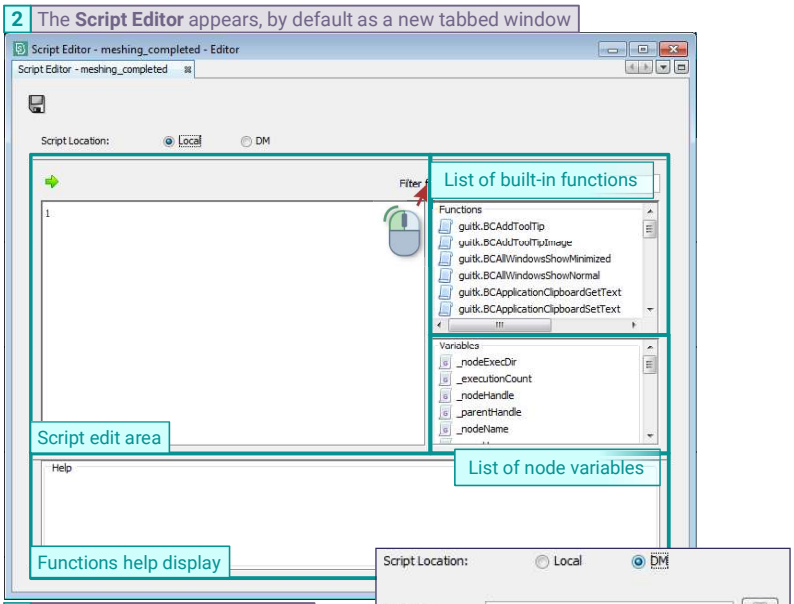
The **Static Input Slot** data set will be available as input to all MxN instances.

- The **Input Data** are arranged in data sets. So, it is possible to add a data set with some parts to be meshed, and another data set for some meshing scenarios to be used.
- In case more than one data sets have been defined, there is the option either to **sum** or to **combine** them. In this example, if we assume that the "batch_mesh _failed_parts" are 5 files, and the "mesh_settings" are 3 files, the **Sum** option will generate 5+3=8 MxN instances, with one MxN input file slot each, whereas the **Combination** option will generate 5x3=15 MxN instances, with two MxN input file slots each.
- The following execution modes are supported:
  Parallel: All instances will be executed simultaneously
  Parallel with delay: The instances will be executed with a time offset (delay).
  Sequential: The MxN instances will be executed one after the other
- The following task assignments modes are supported:
  Default: All instances will be generated being assigned to the user/role that is defined during the design of the MxN node. In case of role any user from this role will be able to execute any of the generated MxN instances.
  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.

Defining scripts on nodes

Script and application nodes can execute scripts and perform customized actions using the built-in script functions of SPDRM, and also python scripts. Such scripts can be defined inside the **Script Editor** of the node or imported to it. The user can type in the script edit area, search among the listed built-in functions of the list, and get help per function in the Help area of the Script Editor. Once the script is set at the Script Editor window it runs upon Node execution.





Application Nodes can execute one script right before the launch of the application, and one right after the termination of the application (**Pre-run** & **Post-run** scripts).

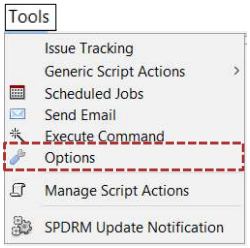
3 Click on a function from the list to see its help in the Help area. Double-click on it to copy it in the script edit area.

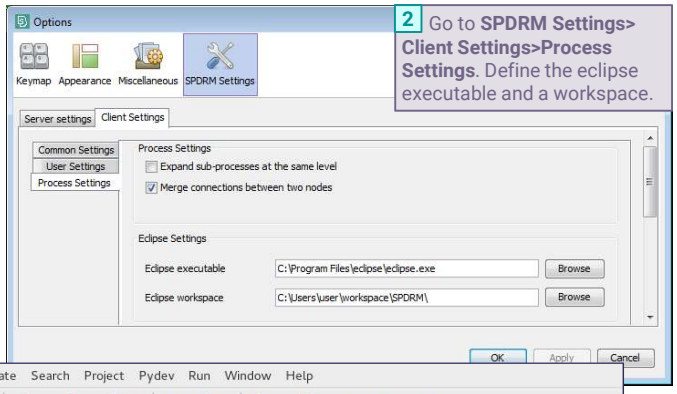
The script can be a stored file in the DM.

Node script debugging in Eclipse

It is possible to pair SPDRM client with the Eclipse IDE application, in order to write, edit and debug node scripts in the latter environment. The pairing of the two software can be achieved with simple configuration, described in detail in the chapter ***Integrating with Eclipse IDE,*** accessed through *Help -> Documentation Index -> User's Guides -> SPDRM Scripting API*.

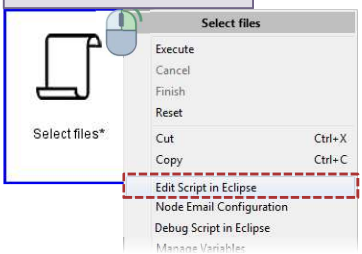
1 Go to **Tools > Options**

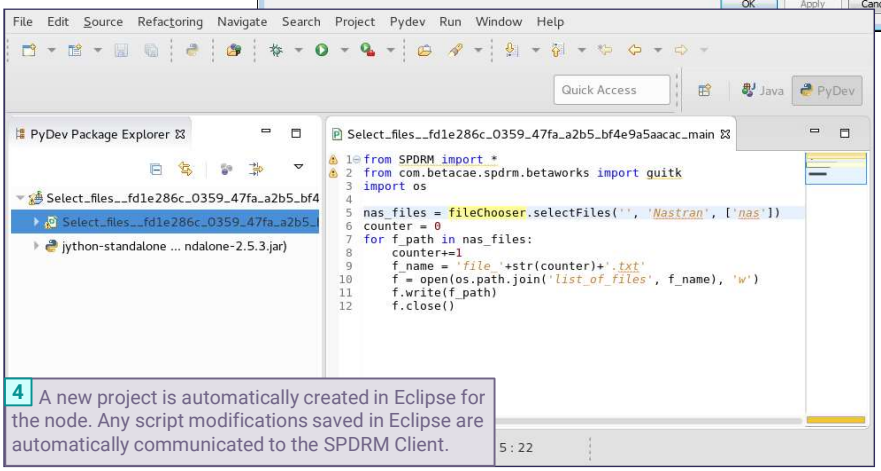




2 Go to **SPDRM Settings> Client Settings>Process Settings**. Define the eclipse executable and a workspace.

3 Activate the **Edit Script in Eclipse** option of the context menu of a node.





The option to debug a script in Eclipse is also available by selecting the respective option of the context menu of the node.

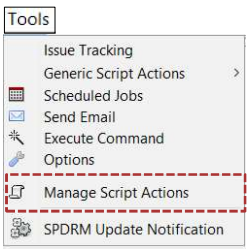
4 A new project is automatically created in Eclipse for the node. Any script modifications saved in Eclipse are automatically communicated to the SPDRM Client.

Script actions

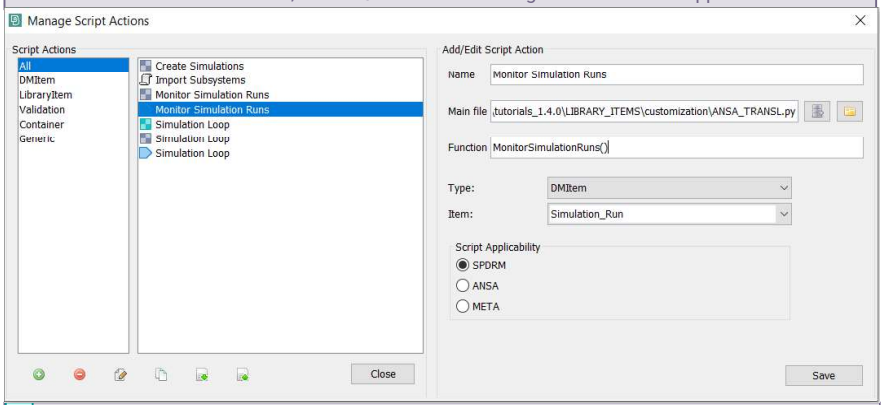
Generic scripts written in Jython/Python scripting language and making use also of the build-in script functions of SPDRM, can be stored in SPDRM and used as custom actions. These actions are executed either from the SPDRM menu in order to perform general tasks, or explicitly from the context menu of items of predefined type, to perform special tasks. For the script actions set-up on predefined item types, there is the option for the script to be executed directly in ANSA or META (apart from SPDRM).

Additionally, such actions can invoke the instantiation of workflows that will be executed automatically and will perform the designed tasks.

1 Go to **Tools > Manage Script Actions**



2 In the **Manage Script Actions** window select the type of script action to create from the options on the left. Press the **Add** button, the **Add/Edit** area on the right of the window appear.



3 In the **Add/Edit** area the user must define the **Name** of the action, the **Main file** of the script located either in the DM or the local system, and the name of the **Function** to be executed.
For the **Container**, **DMItem** , **Library Item** and **Validation** types of actions, select the **Item** of interest from the respective drop-down menu and from the **Script Applicability** menu select the BETA Application, where the script will be executed.
Script actions of type **Validation** are executed every time an item of the defined type is to be added in the SPDRM repository or is edited. Thus, using such a script action it is possible to define checks on certain types of items that if not successfully passed, the item will not be accepted by the system.

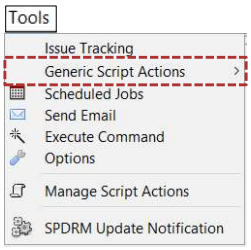
To invoke the instantiation of a specific workflow through a script action, the following build-in SPDRM script functions are available:
process.instantiateNode
process.instantiateWorkflow
There is the capability to activate for these nodes and workflows the option **Auto Delete**, and thus they will be deleted automatically after their completion.

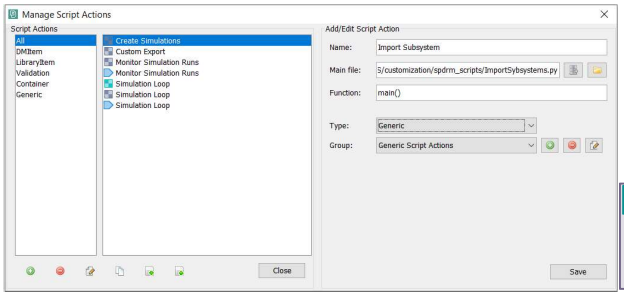
In order to execute a script action directly to ANSA or META, at least one ANSA or META Registered Application must have been earlier defined.

Generic Script Actions

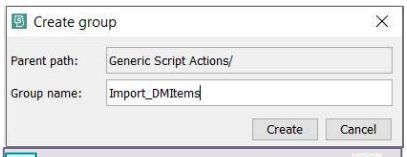
Generic Script Actions can be executed by the Tools menu in the main menubar.

Grouping of Generic Script Actions is supported, enabling better organization and shorter lists in case of large number of existing Script Actions.

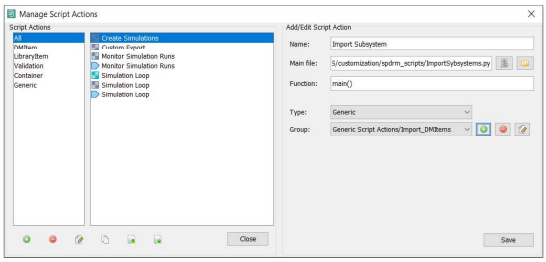




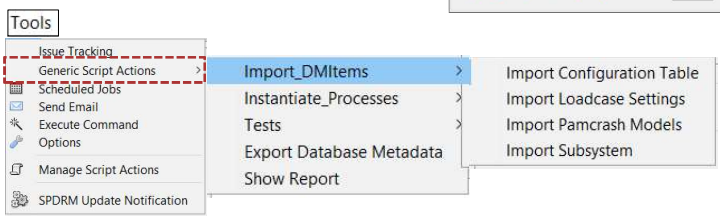
1 While creating a script action of type **Generic**, the default option is to create the action under the default Group **Generic Script Actions**.



2 By selecting the **Add group** button ( ), a new sub-group can be created for the Generic Script Actions enabling action classification in the main menu.



3 Sub-grouping is now displayed in the Group field when editing the script action.

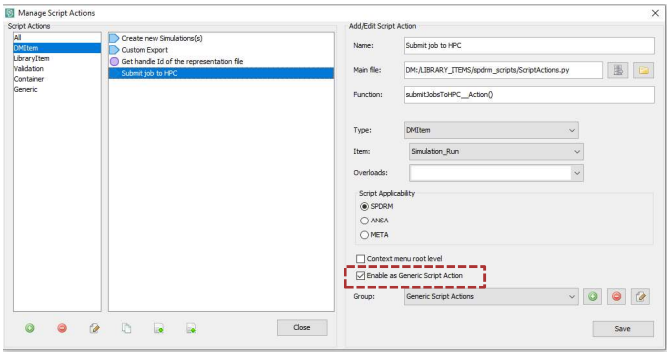


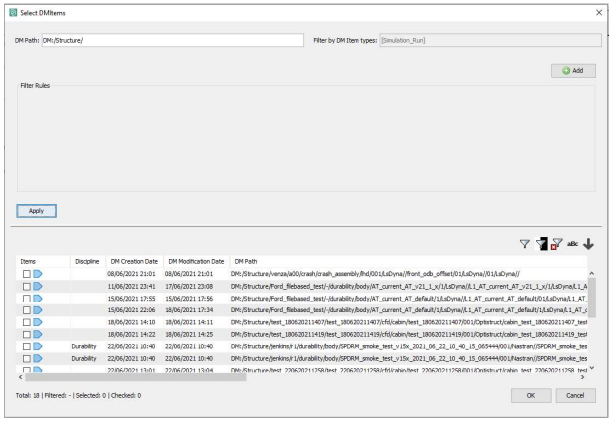
4 Generic Script Actions defined under a sub-group are now displayed properly in a sub-menu in the main Generic Script Actions menu, while rest of them are in root level.

DM Item Script Actions as Generic

DM Item Script Actions are Script Actions that can be executed from the context menu of a DM Item, based on it's type.

An option exists on that type of Script Actions that enables it's execution as Generic Script Actions. The functionality can be enabled by selecting the **Enable as Generic Script Action** checkbox in the Script Actions **Edit** menu.





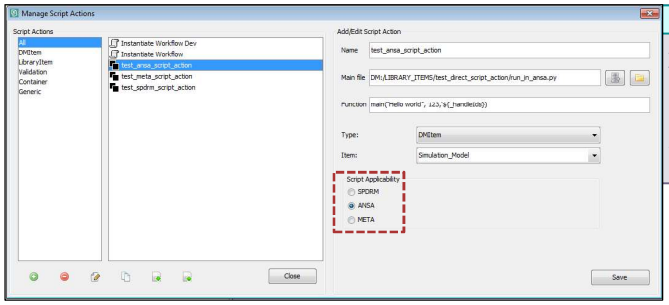
By enabling the option, the DM Item Script Action will be available for execution by the **Tools -> Generic Script Actions** menu.

If a DM Item Script Action is executed as a Generic Script Action, the **Select DM Items** window will appear, where the DM Item type of the corresponding Script Action will be prefilled in the **Filter By DM Item Type** field.

Custom **Filter Rules** can be applied on the window and when the user selects one or multiple DM Items from the list, the DM Item Script Action will be executed, using the selected items' ids as input.

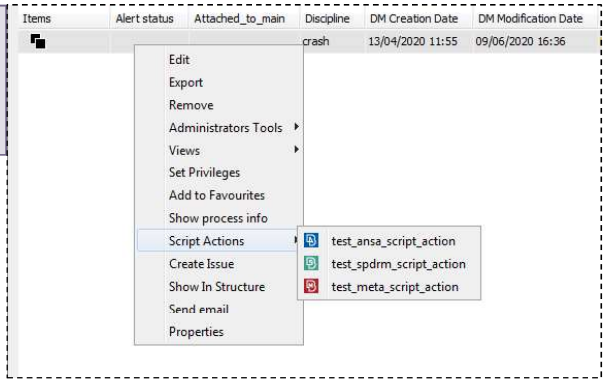
Script actions directly in ANSA/META

A script action can run directly in ANSA or META apart from SPDRM. The user can define a script action to be executed directly in ANSA or META from the context menu of an item. When the script action starts, the user is prompt to select one of the ANSA Registered Applications, where the script will be executed.



1 There is a radio button in **Manage Script Actions** window, where the destination of the script action is defined (SPDRM/ANSA or META). The same script can be used to execute a script action either in SPDRM or in ANSA, by defining different function of the script to be executed.

2 By navigating in the context menu of a DM Item, the menu for Script Actions can be seen, where the respective icon of each application indicates the target application (SPDRM, ANSA or META) of the script action.



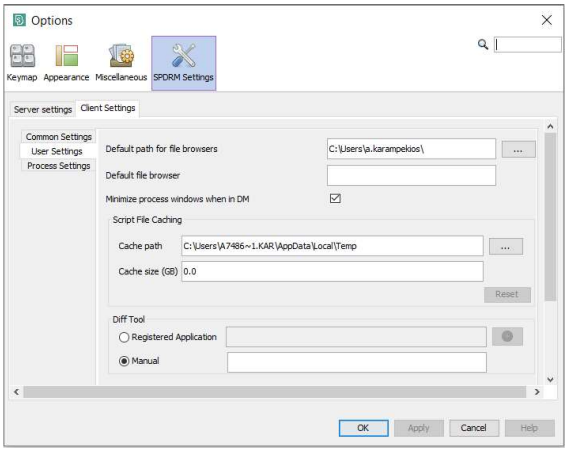
The script action functionality supports a build-in SPDRM variable (**${_handleIds}**), which can be used to resolve the handle id(s) of the selected DM Item(s), when a script action is triggered by DM Items. On such a way the handle id(s) of the item(s) can be given as input argument of the functions that will be executed in ANSA/META.

SPDRM Client DM File Caching mechanism

The SPDRM Client DM File Caching mechanism is a performance optimization mechanism that can be used to enhance the DM files read speed, when network delay exists between the SPDRM Client machine and the network share of the SPDRM vault. When enabled, the mechanism can significantly boost the execution performance of the Script Actions.

The performance is enhanced by searching the cache folder for already downloaded script by the user due to a previous execution of the Script Action. If the files are found in the cache folder, their local copy is used, eliminating the network delay. If not, the files are downloaded and cached for future used.

The Caching mechanism is handled by two keys that can be defined in the SPDRM Client's configuration file (*taxisprops.xml*), ***user_script_cache_size*** and ***user_script_cache_path***.



The first one is the key that enables the mechanism by setting a value greater than zero. The default value is of the key is -1 defining the mechanism as deactivated. If the defined cached size is exceeded, the mechanism deletes automatically files in opposite chronological order until the final size of the folder is less than 80% of the defined size. The value of the key corresponds to gigabytes, e.g.:

```
<entry key="user_script_cache_size">1</entry>
```

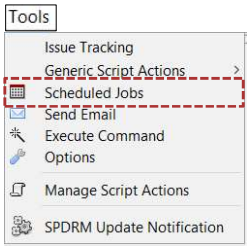
The second key defines the cache path for the cached files and if not defined the user's temporary path will be used by default. e.g.:

```
<entry key="user_script_cache_path">C:\Users\cache_folder</entry>
```

The two keys in SPDRM client's configuration file enable the File Caching mechanism for all users using the client. However, each user can enable or define custom values according to specific needs by the **User Settings** menu, by going to **Tools -> Options -> SPDRM Settings -> User Settings -> Script File Caching.** By using the Reset button, the user restores the settings to the ones defined in the SPDRM Client's configuration file.

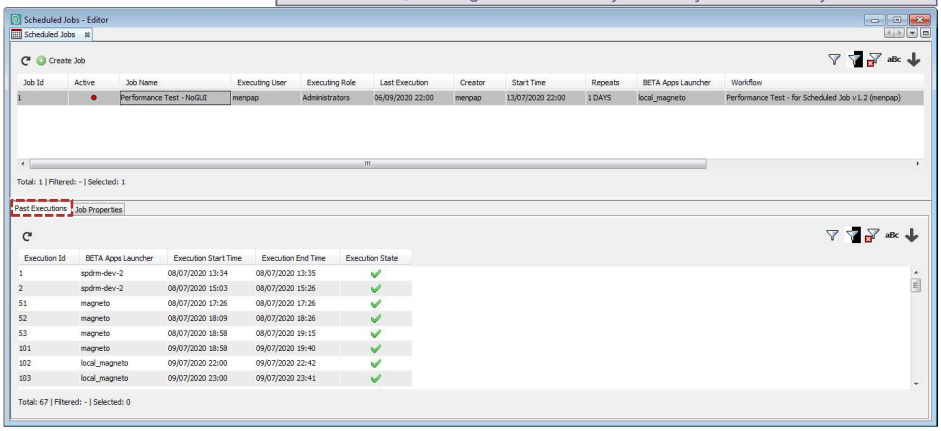
Scheduled Jobs

Scheduled Jobs is a functionality that enables the scheduling and the execution of recurring tasks. The jobs that can be scheduled can be associated either to Generic Script Actions or to Workflow Definitions.



1 The Scheduled Jobs functionality can be accessed by menu **Tools -> Scheduled Jobs**. When selected, the Scheduled Jobs monitoring window will open.

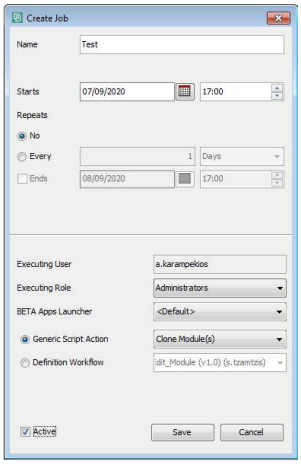
2 From the **Scheduled Jobs** window, a user can see all the existing Scheduled Jobs, created by all users, as well as their status (active/inactive). Apart from the status, several information are given such as the **creator**, the **workflow** or the **script action** that will be executed etc. In the **Past Executions** tab the user can see info about all the jobs that have been ran, starting back on the day that the job was initially scheduled.



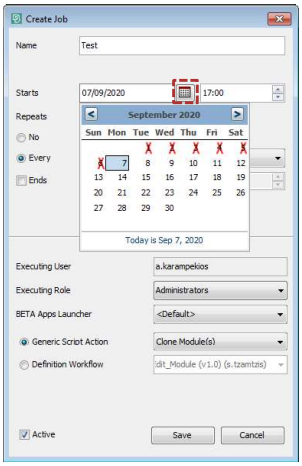
The execution of scheduled jobs takes place in an SPDRM No-GUI client session. Thus, this functionality requires proper set-up for the execution of SPDRM No-GUI clients.

Scheduled Jobs – Create New Job (1/2)

By selecting the Create Jobs option in the Scheduled Jobs window, the user can create a new custom Scheduled Job.

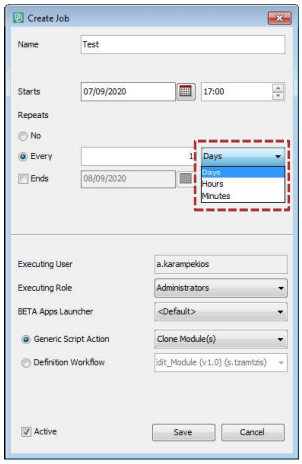


1 By selecting the Create Jobs option in the Scheduled Jobs window. From this window the user can set-up all the desired options for the execution of a Scheduled Job.



2 Besides **Starts** option, the user can define the exact Date and Time that the execution of the Job will start.

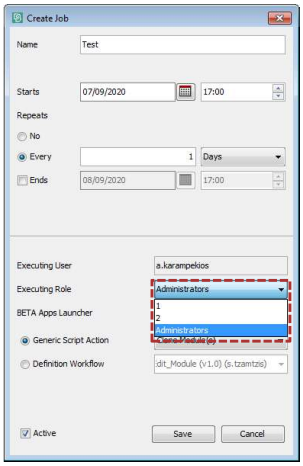
By clicking on the diary icon, the user can select the desired date by using the build-in calendar.



3 On the **Repeats** field, the user can define the repeatability of the Scheduled Job.

The user can select to run the Job only once, by selecting the **No** radio button, or can define the repeatability by:

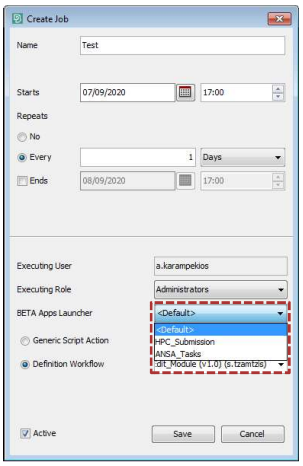
- Days
- Hours
- Months



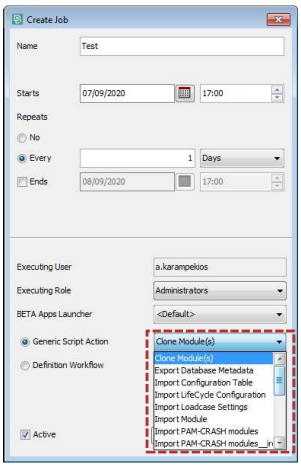
4 The user cannot select the **Executing User** of a Scheduled Job (the logged-in user is configured as Executing User).

However, the Executing Role can be defined, if the user belongs to more than one roles.

Scheduled Jobs – Create New Job (2/2)



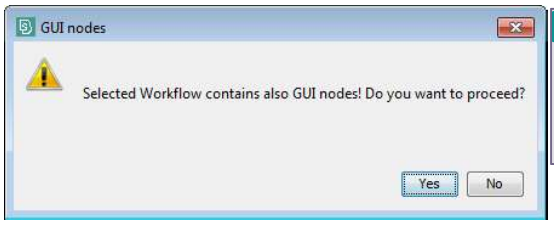
1 If multiple BETA Apps Launchers have been configured, the user can select the desired BAL in which the defined Scheduled Job will be executed. If none is configured, the respective field will remain greyed-out.



2 The last field of the Create Jobs card has a radio button where the user can select the executed item by the Scheduled Job that can be:

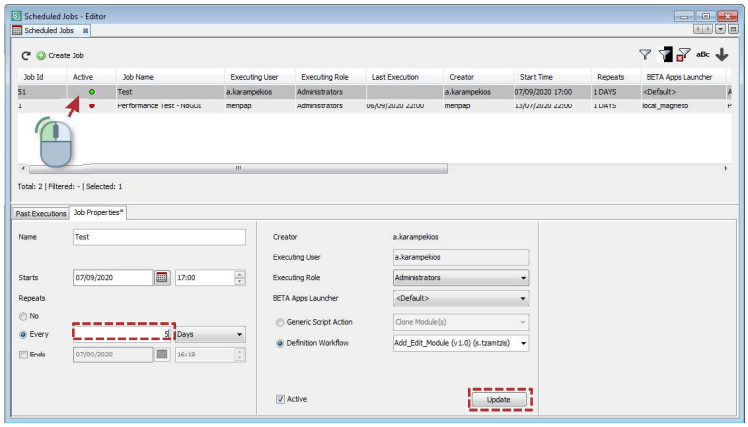
1. Generic Script Action
2. Definition Workflow

Each one of the two options enables the respective drop-down menu for the selection of the desired item.



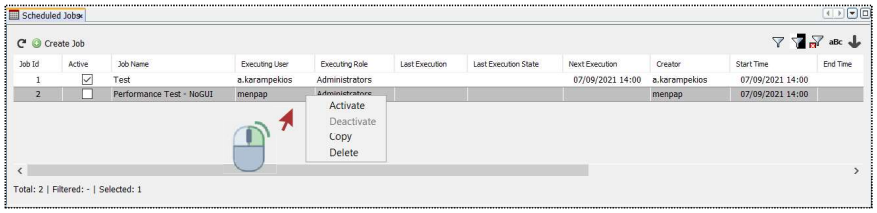
3 When the Save button is pressed in order for the Scheduled Job to be created, if a Definition Workflow has been selected, a check is performed by SPDRM, in order to verify that all nodes of the workflow are No-GUI nodes. Since Scheduled Jobs are executed only in SPDRM No-GUI session, if a GUI node is detected, a proper info message is displayed to inform the user.

Scheduled Jobs – Edit



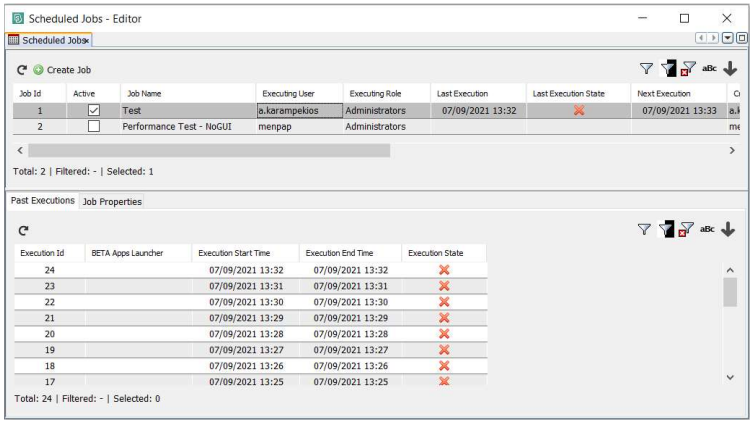
1 An already created Scheduled Job can be modified by the **Job Properties** tab of the Scheduled Jobs window. By selecting a job, the existing properties of the job are displayed in the Job Properties tab. If a value is changed, the Update button is activated and the user can modify the job and apply the changes by pressing it.

2 By right-clicking on a Scheduled Job, it's context menu appears, from which the user can **Activate** and **Deactivate** any existing job.
From the same context menu ,the user can **Copy** or **Delete** a Scheduled Job from the list.



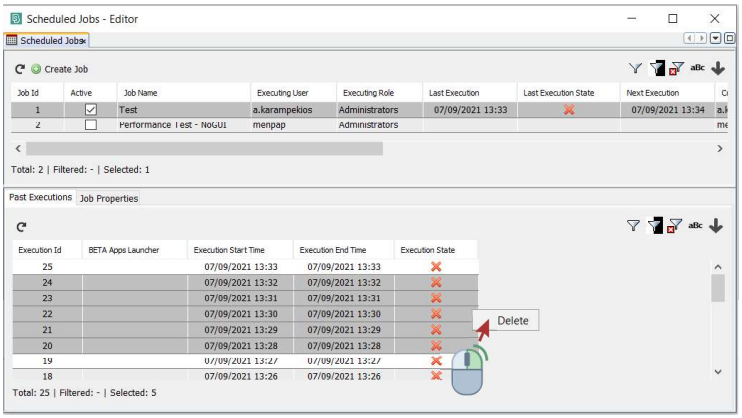
The Job Properties tab is editable only for the Scheduled Jobs that have been created by the logged-in user. The logged-in user can see all the properties of any Scheduled Job on that tab, but if another user is the owner of a job, then the respective Job Properties tab will be greyed out.

Scheduled Jobs – Past Executions



1 By selecting a Scheduled Job and by turning in the **Past Executions** tab, the user can monitor the result of **Past Executions** of a Scheduled Job. The **Execution State** indicates the result of the execution. More information for a Past Execution, such as **Start Time** and **End Time**, can also be found in the list.

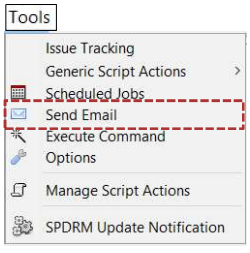
2 By right clicking on a single or multiple **Past Executions**, the user can clear Past Execution entries from the list, by using the **Delete** button from the context menu.
A Scheduled Job with short execution period can produce large number of such useless entries that must be removed.



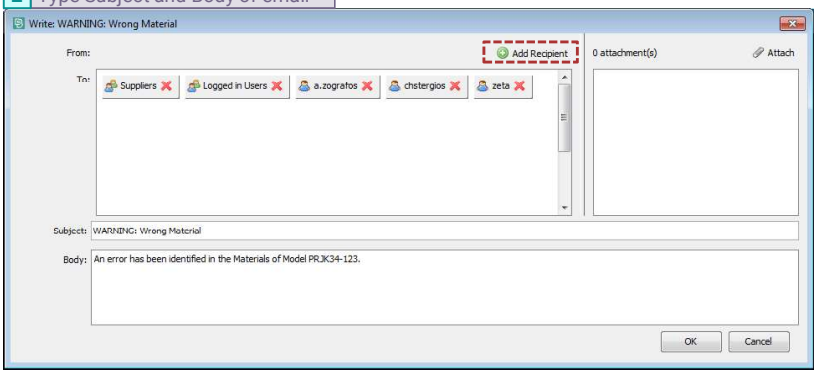
Set up notification sending - Send email

Communication between users is possible via sending emails to groups and individuals. The options to send email to the logged in users or to all users are also available. In order for this option to work properly, it requires the definition of email addresses (in the user's profile) for both the sender and the recipient(s).

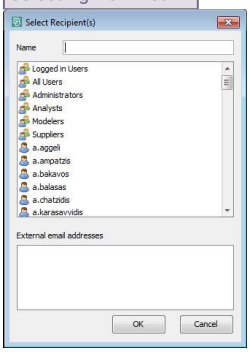
1 Go to Tools > Send Email



2 Type Subject and Body of email



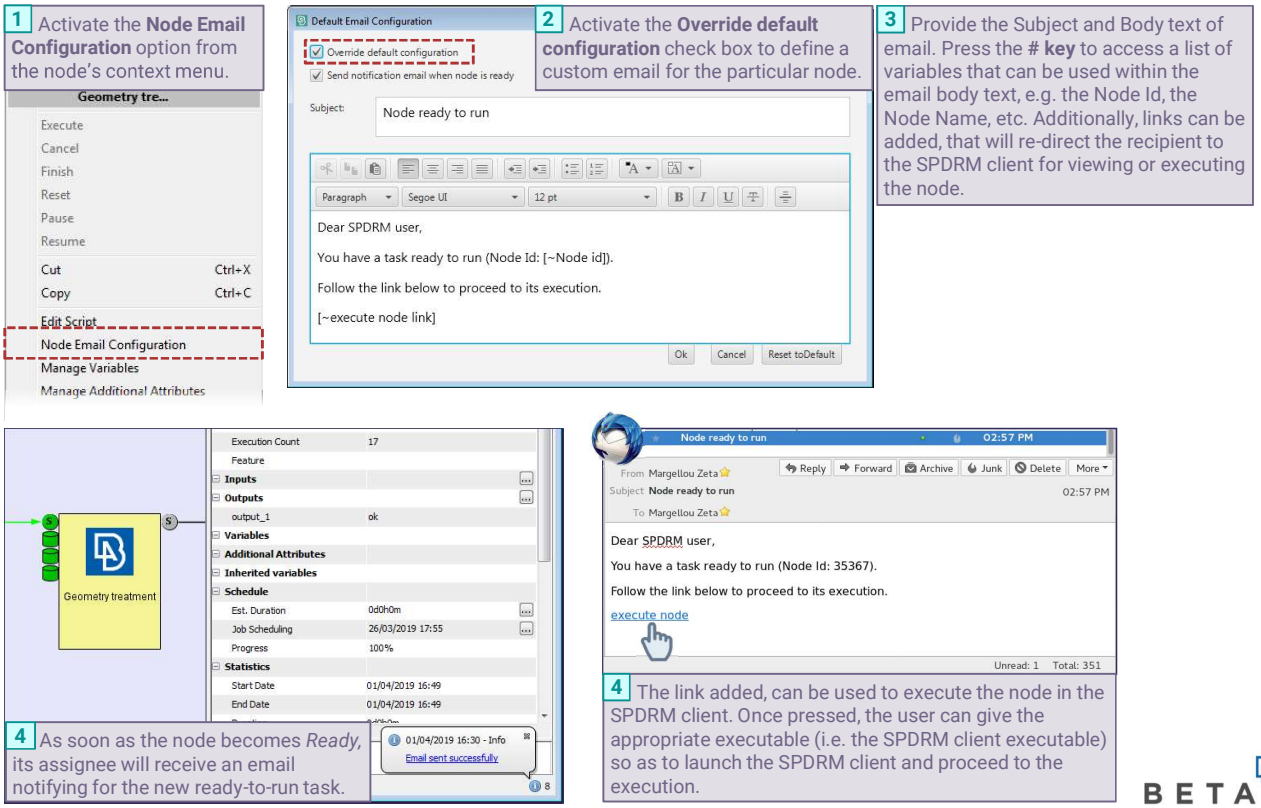
3 Add recipients by selecting from list



Notifications can be sent also automatically when a workflow is running by the use of the following built in script function in the used scripts: `emailer.sendMail`

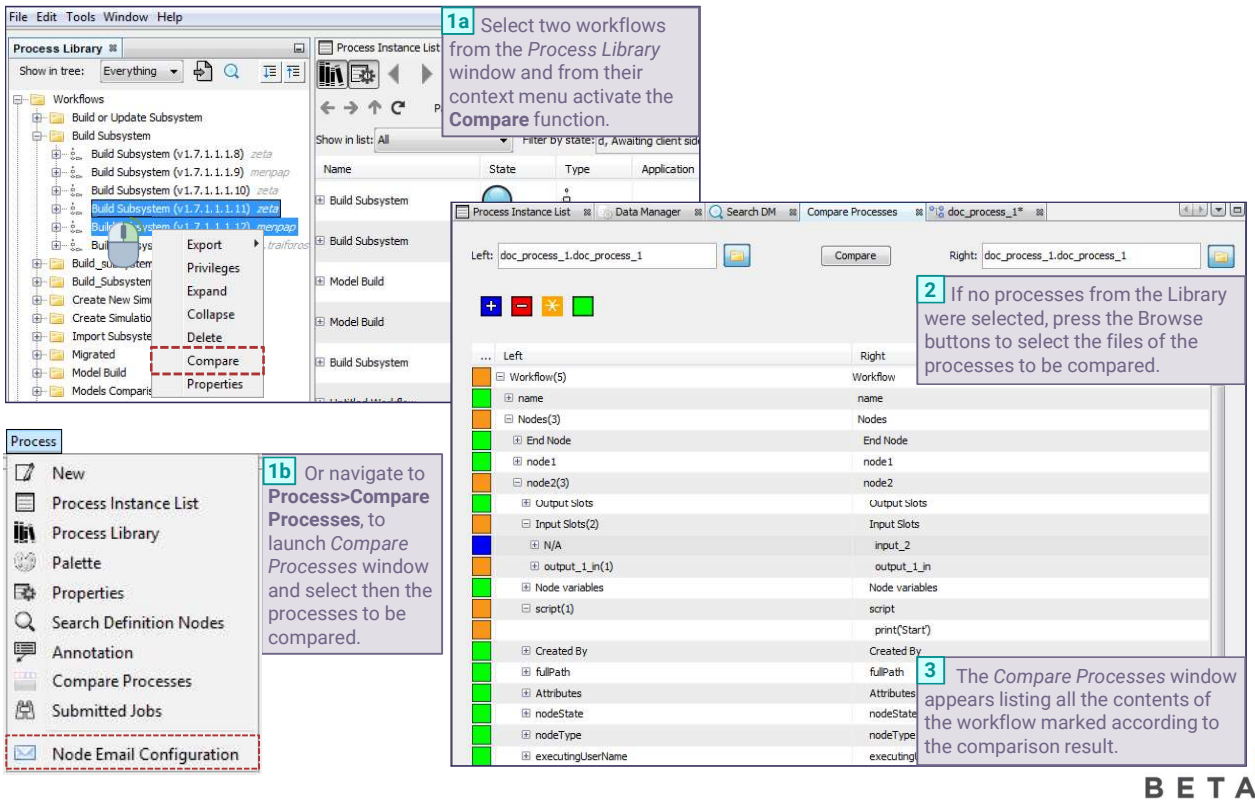
Set up notification sending - Automatically send email upon node status update

Additionally to the capability of sending email on demand, SPDRM can be set-up so as to send email automatically to the assignee of any node that becomes *Ready*. This capability is usually set up by the administrator and is active for all users. Nevertheless, the user can override the predefined message and define a custom email per node.



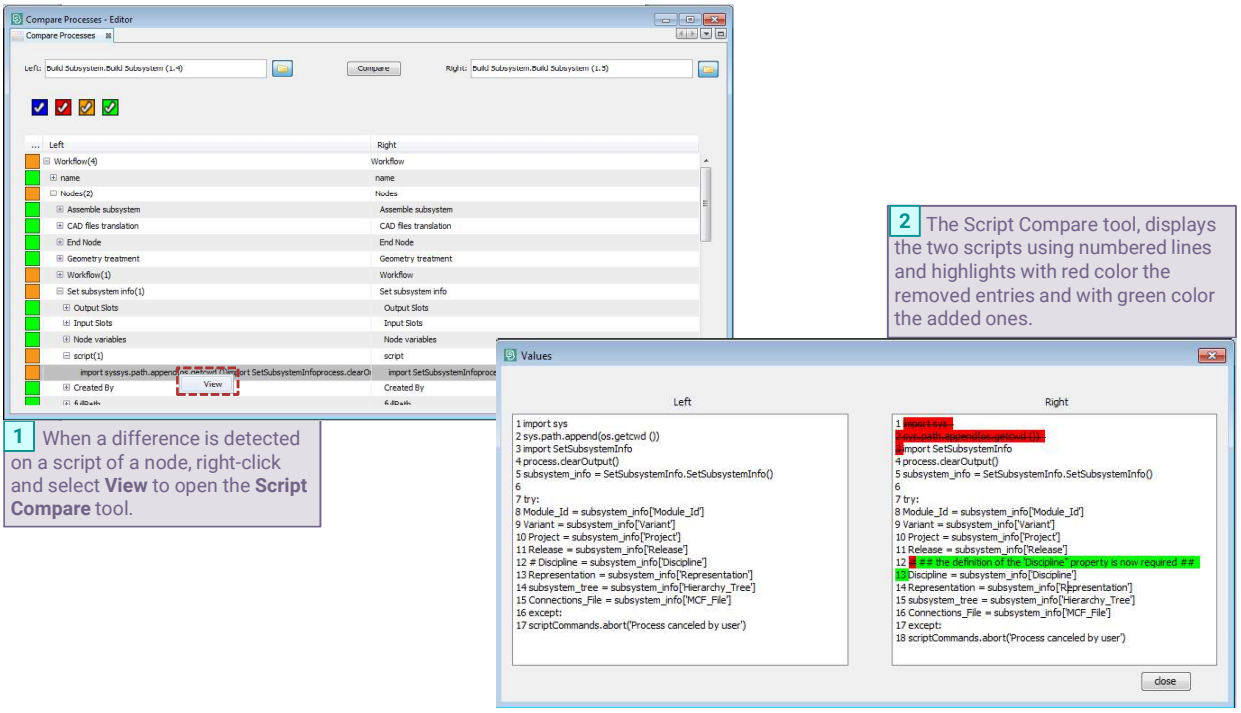
Compare workflows (1/2)

It is possible to compare two workflows in order to identify their differences. The user can select two processes from the Process Library, or select the files of two saved processes, and compare them. The comparison is done on all levels and contents of a workflow, that is, attributes, input/output slots, variables, node scripts etc.



Compare workflows (2/2)

A final extra level of comparison exists that enables the comparison of the scripts that are contained in a node:

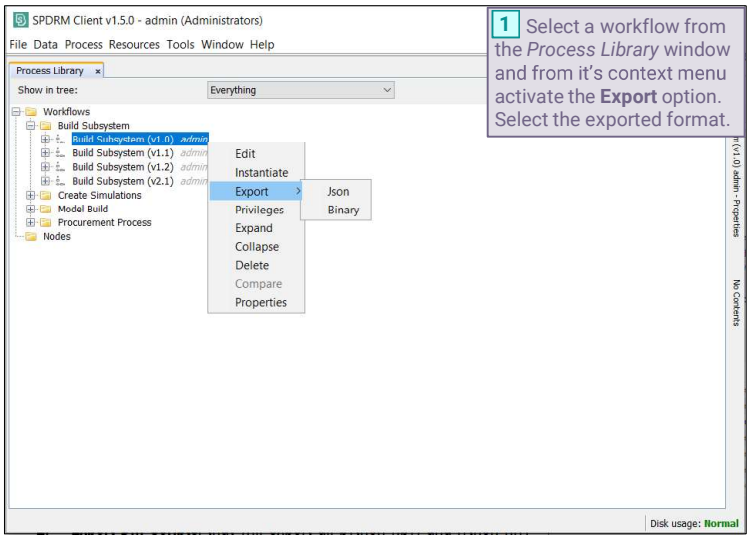


1 When a difference is detected on a script of a node, right-click and select **View** to open the **Script Compare** tool.

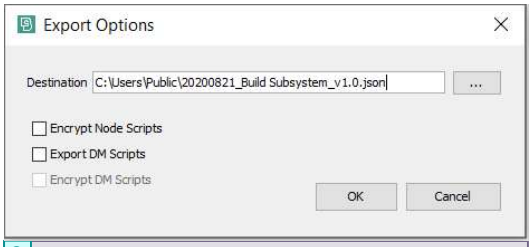
2 The Script Compare tool, displays the two scripts using numbered lines and highlights with red color the removed entries and with green color the added ones.

Export and Encrypt workflows

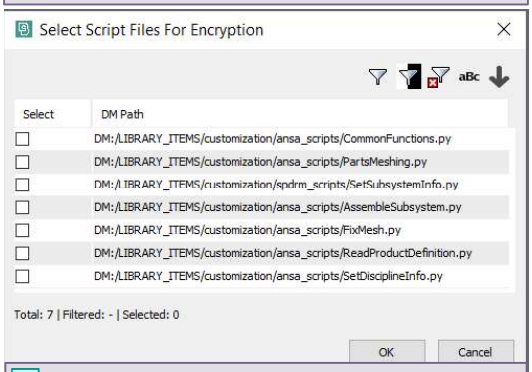
It is possible to export a workflow from the Process Library and encrypt all or some of the elements of the process.



1 Select a workflow from the *Process Library* window and from it's context menu activate the **Export** option. Select the exported format.



2 The **Export Options** window appears where the target folder for the export and the Advanced Export Options can be defined.



3 If Export DM Scripts and Encrypt DM Scripts is selected, the **Select Script Files For Encryption** window appears, where the user defines which of the contained in the workflow scripts will be exported as encrypted scripts.

The Advanced Export Options that are available on step 2 are:

1. **Encrypt Node Scripts**, that will transform all nodes of the process to encrypted nodes.
2. **Export DM Scripts**, that will export all python (.py) and jython (.jy) scripts defined as input slotes or DM scripts on the nodes of the process.
3. **Encrypt DM Scripts**, that will encrypt all manually selected (step 3) script to *.pyb* or *.jyb* files.

The final result will be two files named after the process name, a .json with the exported process and a .zip with the accompanying scripts.

Enable auto-reuse in launched ANSA – Application Node (1/2)

The mechanism described for auto-reuse of launched ANSA can be used for running ANSA Application Nodes in already opened ANSA session.

- The mechanism cannot be used for META Application Nodes, unlike the META MIMEType.
- The auto-reuse mechanism can be used for the design of a process only with ANSA versions 21.0.0 or later, because the script function **ansa.session.GetReadyForNewTask()** is needed.

There are three prerequisites for an ANSA Application Node to be defined for auto-reusability:

1. The **Enable Application Reusability** checkbox is selected in node's *Application Settings.*

2. The following keys must be contained in node's *Application Settings:*
   - *-dmroot*
   - *-dmticket*
   - *-i or –exec*

   (-exec must be used twice, one for load_script ANSA argument and one for the function's execution argument)

3. The function **ansa.session.GetReadyForNewTask()** must be set in the end of an automated script, in order that the ANSA session becomes available for re-use. The same function must be called by script editor or by using a user button that executes it when manual work is done in the ANSA session and the work done by the user is finished.

The auto-reusability mechanism can be used also for no-gui application nodes.

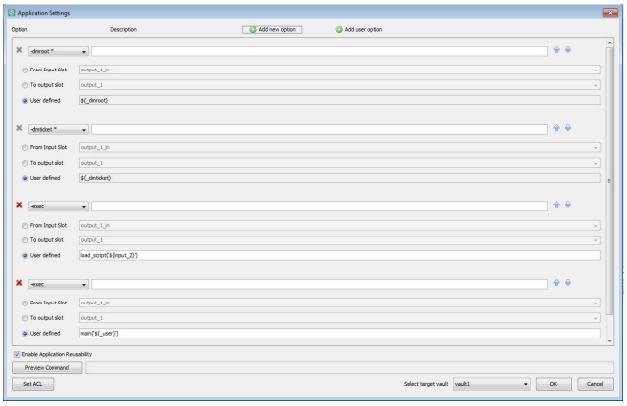
The *Enable Application Reusability* checkbox is enabled only for ANSA application nodes. The checkbox is greyed out in all other cases.

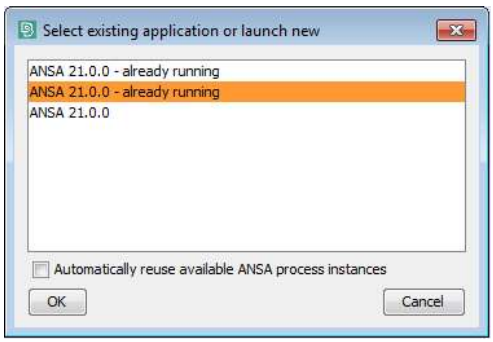
Enable auto-reuse in launched ANSA – Application Node (2/2)

The **Enable Application Reusability** button and properly configured Application Settings can be seen on the image on the right.

Another related configuration can be found in *Tools -> Options -> SPDRM Settings -> Client Settings -> Process Settings* menu.

When the checkbox **Automatically reuse available ANSA process instances** is checked, every node marked with *Enable Application Reusability* flag will be automatically executed in the first available launched ANSA session (system selection).





Otherwise, when the flag is off (which is the default value), each time an ANSA node with Enable Application Reusability flag is enabled, a pop-up window named Select existing application or launch new will appear and will ask from the user the selection of an already running ANSA.

The orange colored session indicates a running ANSA with GUI, while the untagged one a running no-gui ANSA

The *Automatically reuse available ANSA process instances* flag mentioned earlier can be directly enabled from that window apart from the Process Settings.

SPDRM
WORKFLOW MANAGER

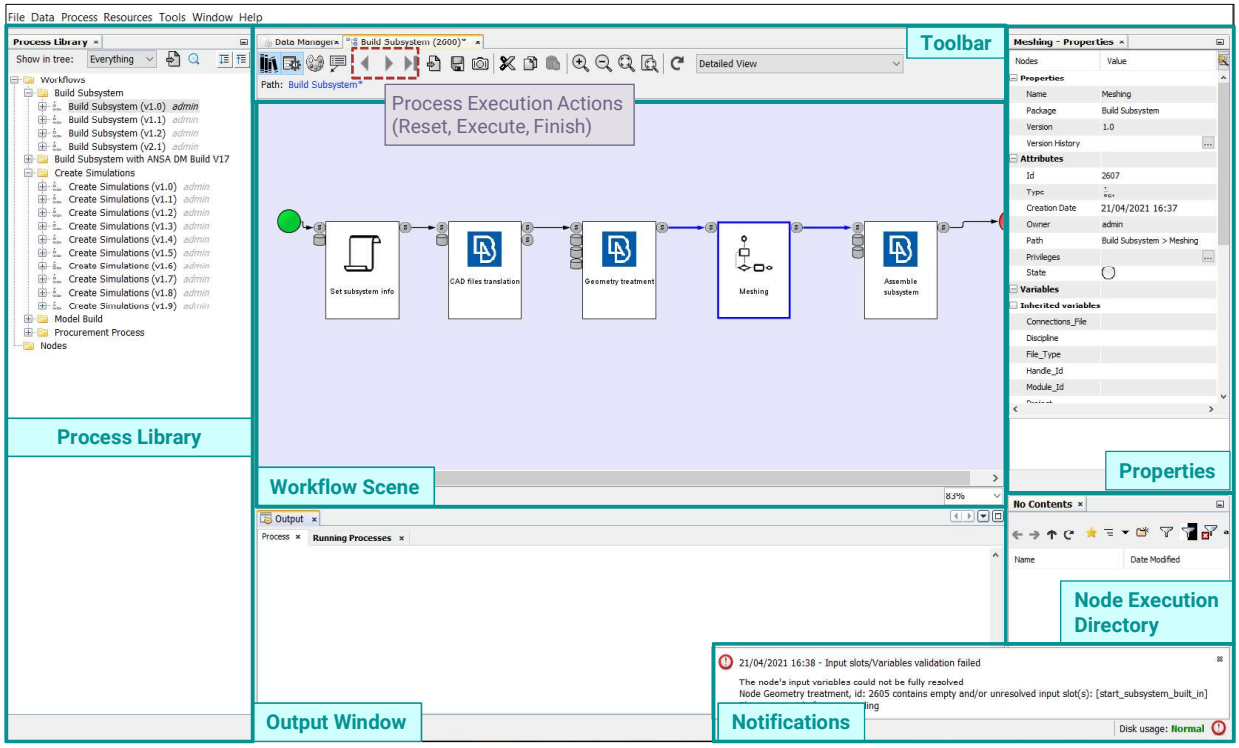
Process Execution

BETA
SIMULATION SOLUTIONS

Table of contents

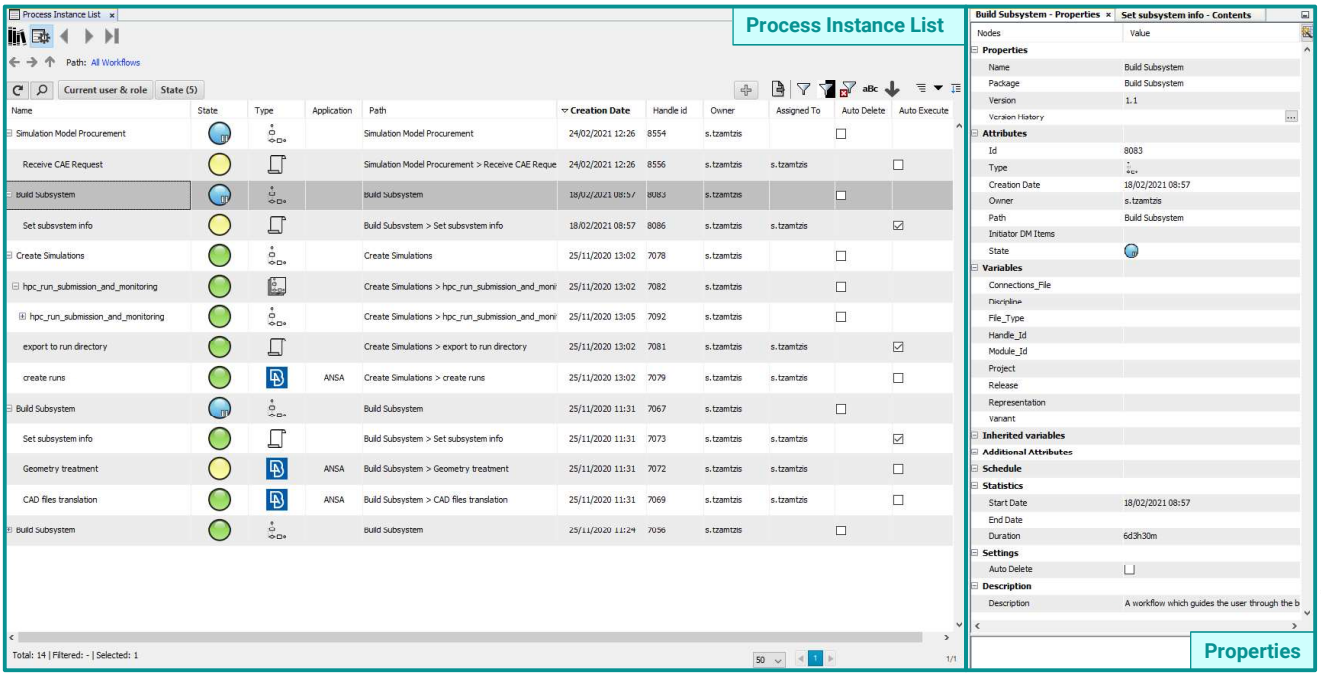
Introduction to process execution through SPDRM Client

The execution of a process can be performed by one or more Users or Groups that have been assigned with specific Nodes and can take place either at the *Workflow Scene* window or at the *Process Instance List* window.



Introduction to process execution through SPDRM Client

The execution of a process can be performed by one or more Users or Groups that have been assigned with specific Nodes and can take place either at the *Workflow Scene* window or at the *Process Instance List* window.



BETA SIMULATION SOLUTIONS
© Copyright 2019 BETA CAE Systems. All rights reserved

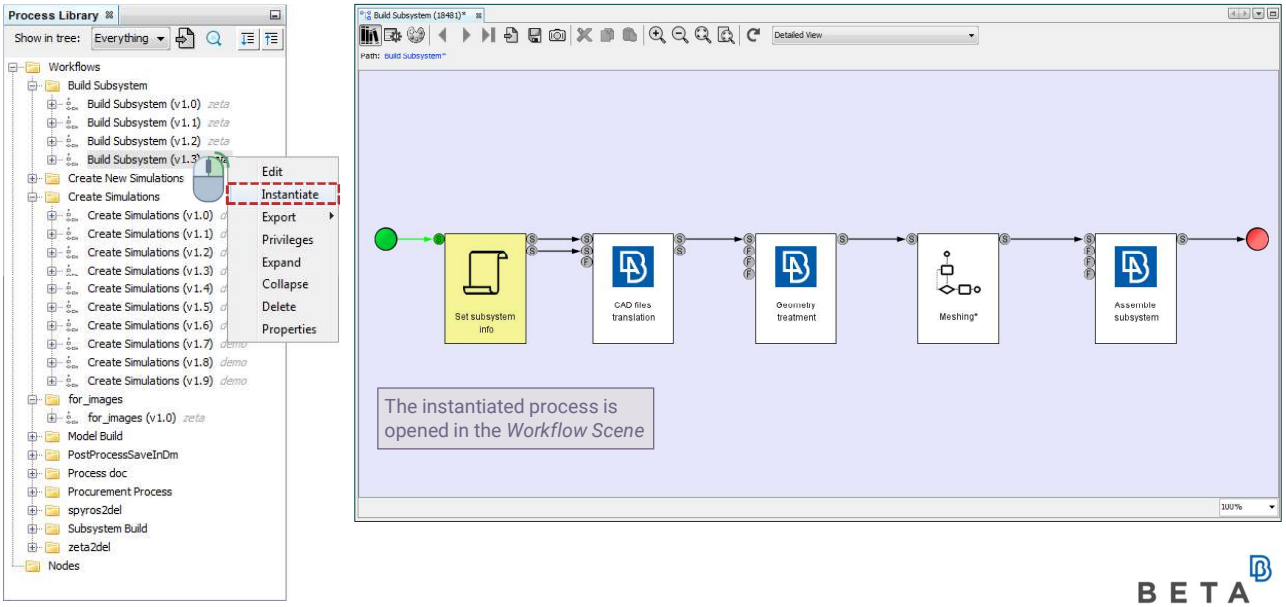
Process Library and Workflow Scene

The *Process Library* window (invoked from **Process > Process Library**) is a list of the available processes which are divided into:

- **Workflows** which contain all the process definitions that are accessible by the user.
- **Nodes** which can be used to search and re-use existing node definitions, during process design.

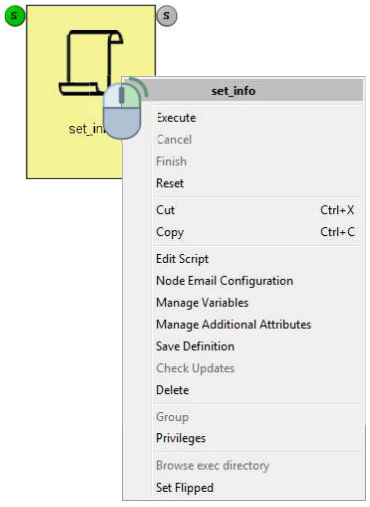
The user can instantiate a process from the library by pressing right-click on it and selecting the **Instantiate** option from the context menu.

The process appears in the *Workflow Scene* window where the processes are executed.



The Node Options in the Workflow scene

The options found in the context menu of a Node in the *Workflow Scene* window are activated or disabled depending on the node status. In general, the following options are available:

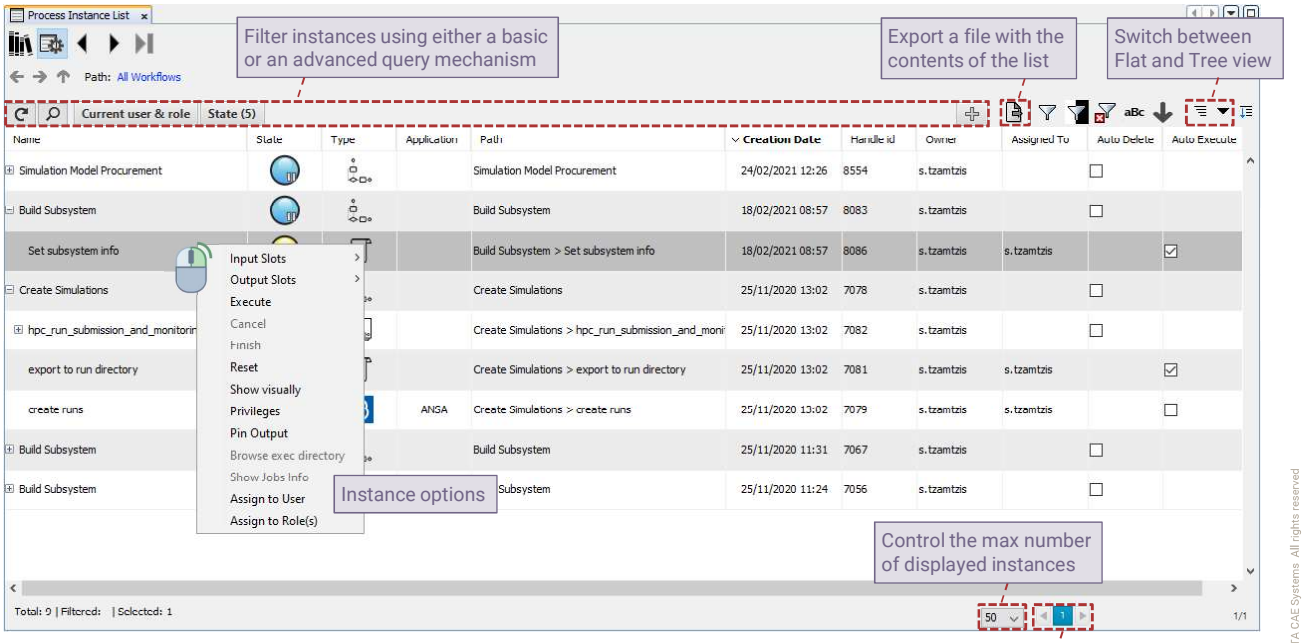


Execute:	Allows the manual execution of a *ready* node.
Cancel:	Cancels the execution of node. In case of application node, application is terminated and node gets in *paused* state.
Finish:	Allows the user to manually end the execution of a *pre-finished* node.
Reset:	Refresh the state of the node.
Cut/Copy:	Allows the user to cut/copy the node and reuse it in the design of a process (**Cut** is disabled when a node is *running*).
Edit Script:	Allows the user to edit the main script of script nodes or the Pre-run and Post-run scripts of application nodes (when a node is *running*, this options transforms to **View Script**).
Node Email Configuration:	Set-up the automatic email notification mechanism.
Manage Variables:	Allows the management (view, add, edit, remove) of node variables.
Manage Additional Attributes:	Allows the management (view, add, edit, remove) of node additional variables.
Save Definition:	Allows the user to save any modifications performed on the node as a new definition.
Check Updates:	Allows the user to check for updated definitions of a node.
Delete:	Allows the user to delete a node, provided that the user has the respective privileges.
Group:	Allows the user to select two or more nodes and group them into a sub-process.
Privileges:	Allows the user to access the privileges definition of the node.
Browse exec directory:	Allows the user to browse the node execution directory of a *running, paused,* or *pre-finished* node.
Set Flipped:	Allows the user to flip the drawing of a node and swap the position of input and output slots.

Process Instance List

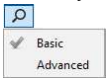
The process instances are managed through the *Process Instance List* window invoked from **Process > Process Instance List**.

Apart from process execution and information viewing, the Process Instance List comprises a dynamic tool for users, enabling them to have an overview of their "To Do" lists and manage their workload, using a search tool based on basic or advanced queries to filter the displayed instances.



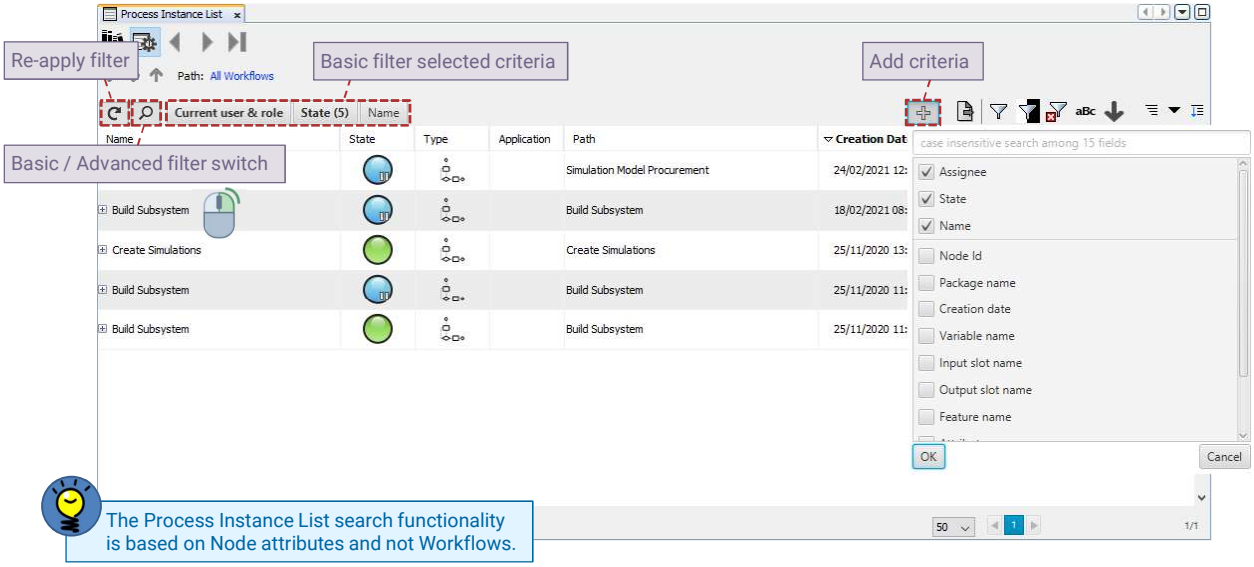
Process Instance List – Search mechanism

By default, the Process Instance List opens with an applied filter, showing the instances assigned to the **Current user & role** with any 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 SPDRM client session. In addition, the instances are listed by default from newest to oldest.



The Process Instance List search mechanism offers a **Basic** and an **Advanced** filter that can be switched using the respective button

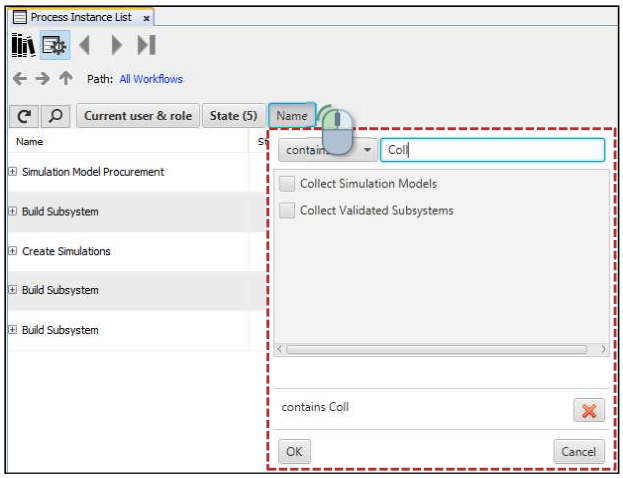
In the **Basic** filter, the applied criteria appear in bold and the unapplied in regular font. Adding more criteria to the query is done with the respective button.

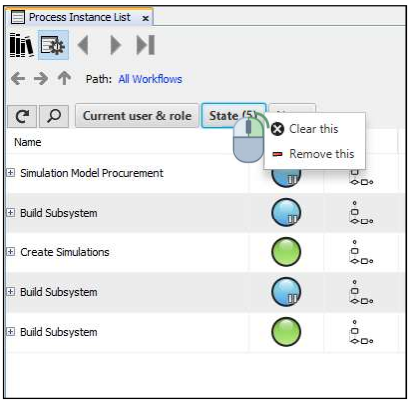


Process Instance List – Search mechanism

To select a value for one of the selected criteria and apply it in the **Basic** filter, the use can press on it with the left mouse button. One of the available values can be selected and activated through the respective checkbox, in which case a search for an exact match will be performed. Alternatively a search for a partial match with the provided text and operator can be performed.

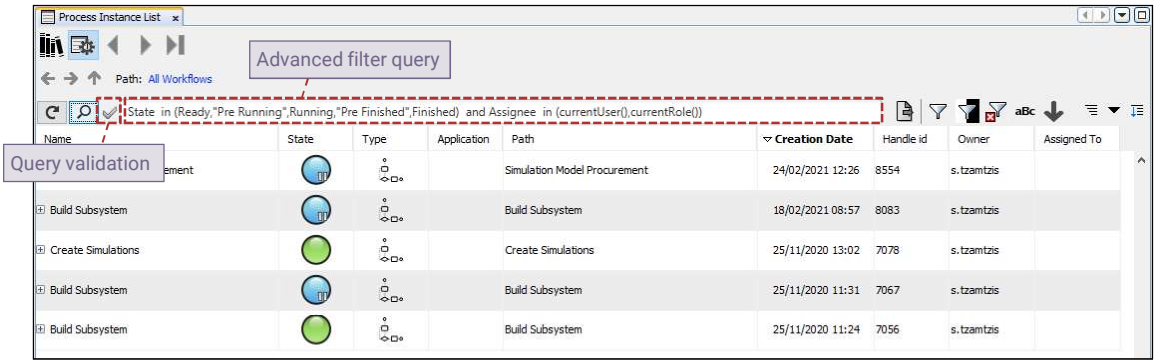
To **clear** an applied criterion or to **remove** it completely, the respective option from its context menu can be used.

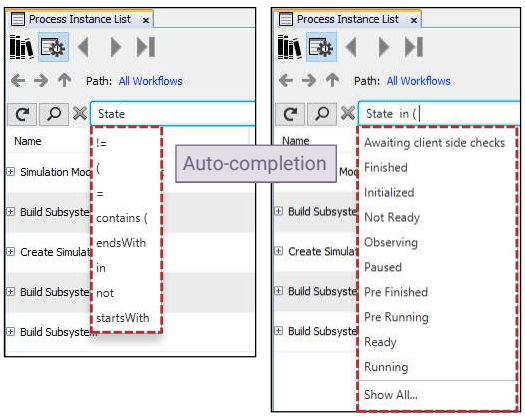




Process Instance List – Search mechanism

In the **Advanced** filter, the search query needs to be constructed by the user using specific syntax. SPDRM will validate the query, to inform the user for any possible syntax errors before applying.



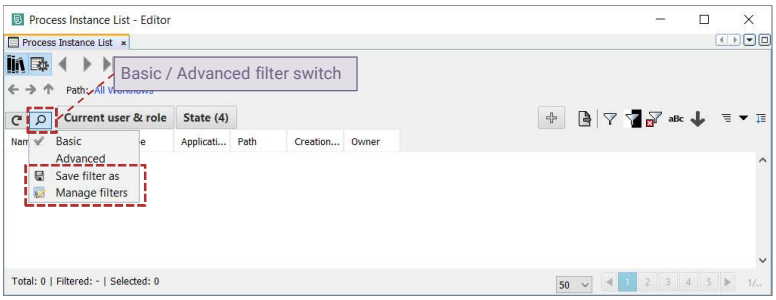


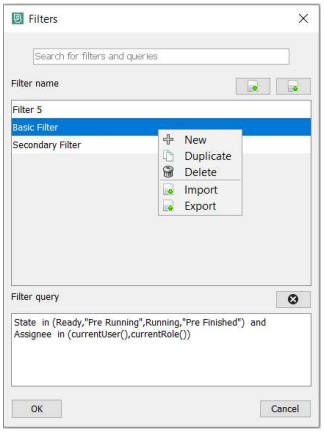
SPDRM offers auto-complete functionality, to help construct the query correctly. Auto-completion is available for all query components:

- Criteria
- Operators
- Values

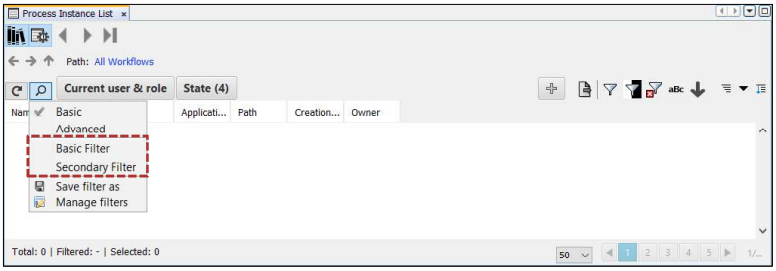
Process Instance List – Filters

Process Instance operates in **Basic** and **Advanced** Filter mode. The switch between the two modes can be done by clicking on the **Basic/Advanced filter switch** button. The options **Save filter as** and **Manage filters** exist also on the same menu, enabling the user to **Save** custom filters for future use. Both of the options will open the **Filters** management window.





Each custom filter that is saved by the user in the Filters management windows can be used directly by the **Basic/Advanced filter switch** menu. A renewed list of the saved custom filters is displayed each time a new filter is created.



Each user can use the context menu of the **Filters** management window to create **New** filters, **Duplicate** or **Delete** existing ones.

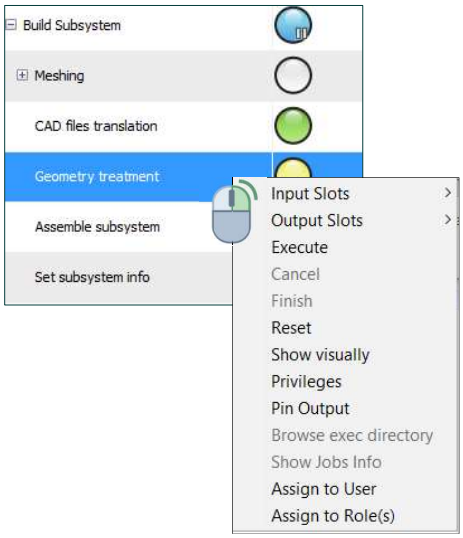
The user can also **Import** filters or **Export** saved one in json format by using either the context menu or the dedicated icons.

Filter query field displays the syntax of the saved filter.

The Node Options in the Process Instance List

The options found in the context menu of a Node in the *Process Instance List* window have some minor differences compared to those found in the *Workflow Scene*.

In general, the following options are available, which depending on the node state may be activated or disabled :



Input Slots / Output Slots: Allows the user to view the current or default value of the node input / output slots

Execute: Allows the manual execution of a *ready* node

Cancel: Cancels the execution of node. In case of application node, application is terminated and node gets in *paused* state.

Finish: Allows the user to manually end the execution of a *completed* node and mark it as *finished*

Reset: Allows the user to refresh the node and restore it to *ready* for execution

Show visually: Allows the user to open the selected node in the *Workflow Scene* window

Privileges: Allows the user to access the privileges definition of the node.

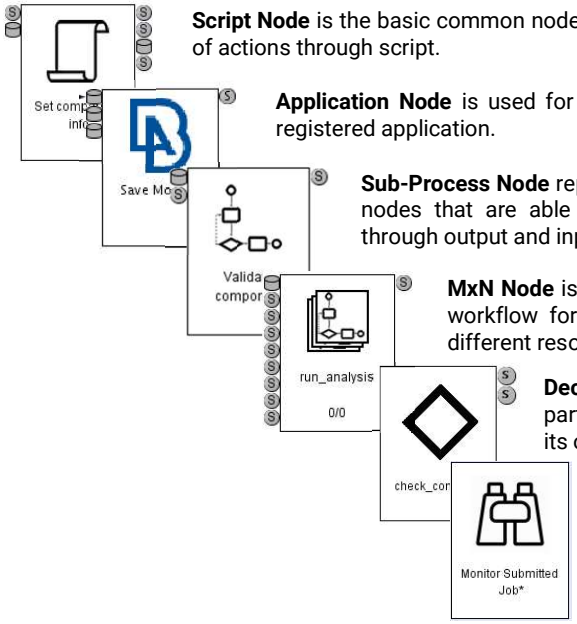
Pin Output Can be used to create a new pinned tab in the Output window, that will display the output of the selected process as "fixed" content, i.e. not in sync with the selected Process/Node.

Browse exec directory: Allows the user to browse the node execution directory of a *running* or *completed* node

Show Jobs Info Displays information for the submitted jobs of an Observer Node.

Set Assign to User / Assign to Roles: Allows the user to assign the selected node to a specific user or user role.

Node and Input/Output Types



Script Node is the basic common node which is used for the execution of actions through script.

Application Node is used for actions that require the execution of a registered application.

Sub-Process Node represents a multiple-step task. It can contain sub-nodes that are able to communicate to the root (master) process through output and input streams.

MxN Node is a special sub-process node used to instantiate the same workflow for M different input data sets, optionally delegated to N different resources.

Decision Node is a special node used to drive the process towards a particular path, based on a condition statement. The node fills one of its output slots based on whether the statement turns True or False.

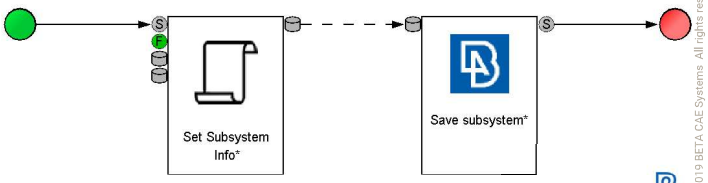
Observer Node is a special node used to monitor of a job submitted to HPC. Two types of observer node exist, **Script** and **Application Observer Node**, each of which embodies the functionalities of the respective simple ones, alongside their ability to enter idle mode, as long as they monitor the progress of the submitted jobs.

Supported types of variable for **Input** and **Output** slots are:

String, **Float**, **Integer**, **Boolean**, **Date**, **Single file**, and **List of files**.

Data can be provided to a node either through a connection with another node, or through the DM.

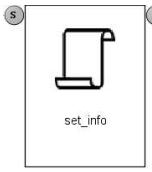
Files are communicated between nodes through dash connection lines, while the continuous connection lines represents the transfer of signals.



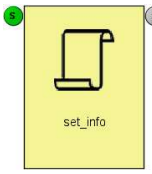
Color Coding of Nodes

A default color scheme is in place in SPDRM, that allows the user to quickly and easily identify the state of each process, node and inputs/outputs. During the execution of a process, the colors of a node and of its slots are changed, reflecting the transition from one state to another.

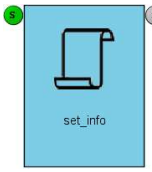
Basic node states



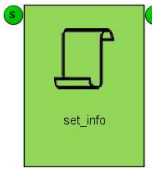
Not ready: Input Slots data are not available. Node cannot be executed.



Ready: Input Slot data are available. Node is ready to be executed.

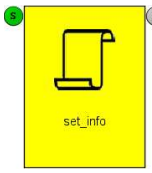


Running: Node is currently executed.

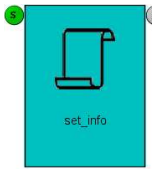


Finished: Node has finished its execution.

Additional (optionally acquired) node states

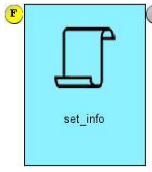


Scheduled or Paused: Input Slot data are available. Node is ready to be executed but it is scheduled to start on a particular time in future, or
node has been paused by the user (only for application nodes).



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

Observing: Node is in observing state and receives information about the progress of the submitted jobs. *Submitted Jobs* option from the context menu when in that state provides those information.



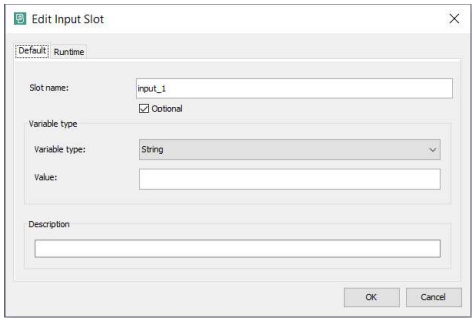
Polling input: Input slot of the node is polling for particular input data.

Color Coding of Slots

There are two types of input slots available:

- **Obligatory** input slot, that must be filled in order for the node to be executed.
- **Optional** input slot, that is not required to be filled in order for the node to start.

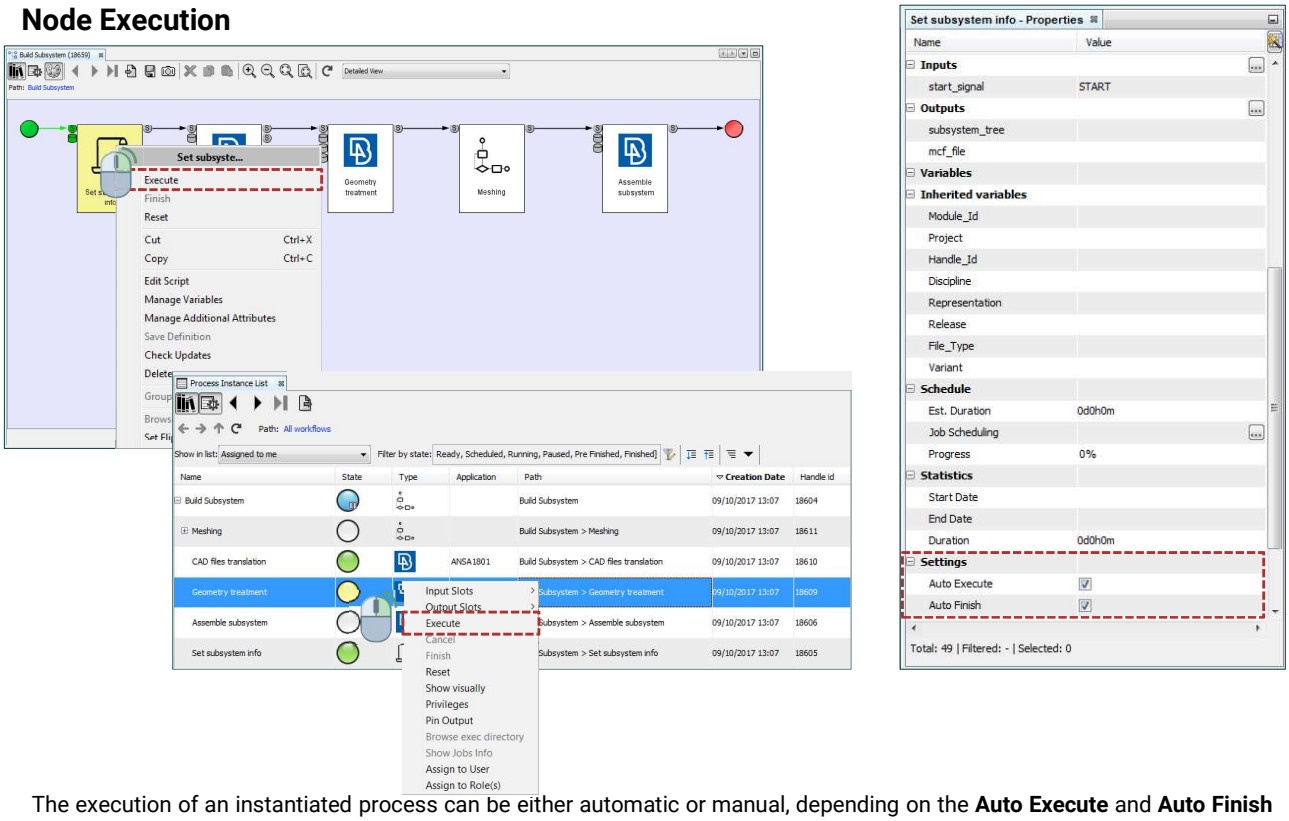
An input slot can be defined as obligatory or optional, by the *Optional* checkbox in the *Edit Input Slot* window. If the optional input slot is filled when the node is executed, then the slot becomes green, otherwise it remains as before (light purple) and the process continues.



Input Slots: Data are not available (Obligatory slot)
Data are not available (Optional slot)
Data are available and ready to use

Output Slots: Data have not been provided by the node yet
Data are available but they have not been released to the next node
Data are available and they have been released automatically to the next node

Node Execution



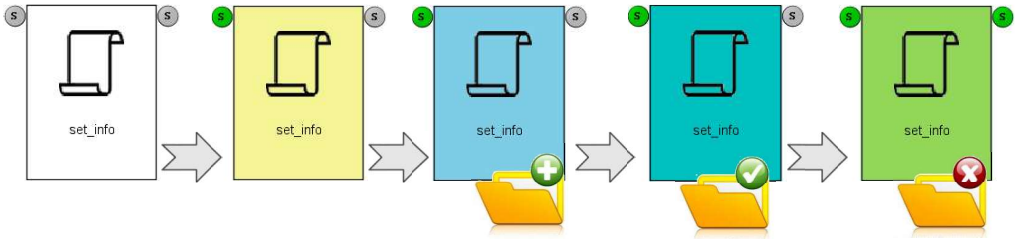
The execution of an instantiated process can be either automatic or manual, depending on the **Auto Execute** and **Auto Finish Settings** of each process node, as defined in the node *Properties* window.

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 Directory

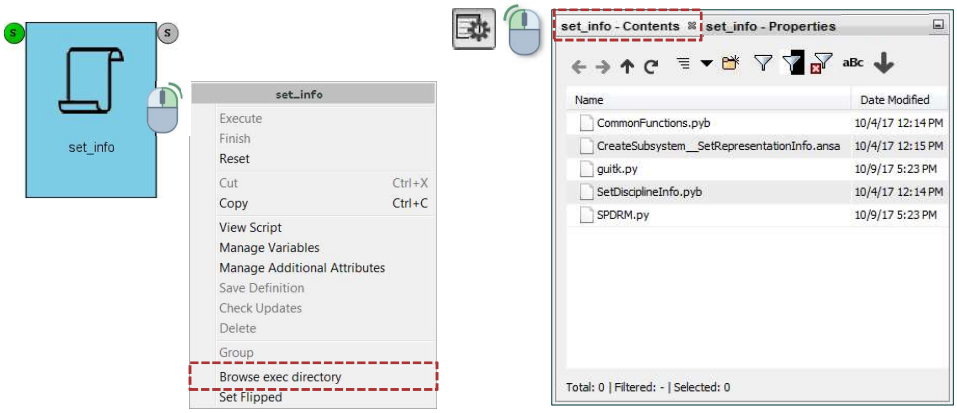
During the execution of a node, SPDRM creates a unique temporary directory, the Node Execution directory, for the communication of data between the client and the SPDRM database.

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



A scheduled node has not yet an execution directory.
A paused node retains its execution directory.
A polling input node has not yet an execution directory.

It is accessed through the **Browse exec directory** option of the context menu of the node, or through the *Contents* tab of the *Properties* sheet.

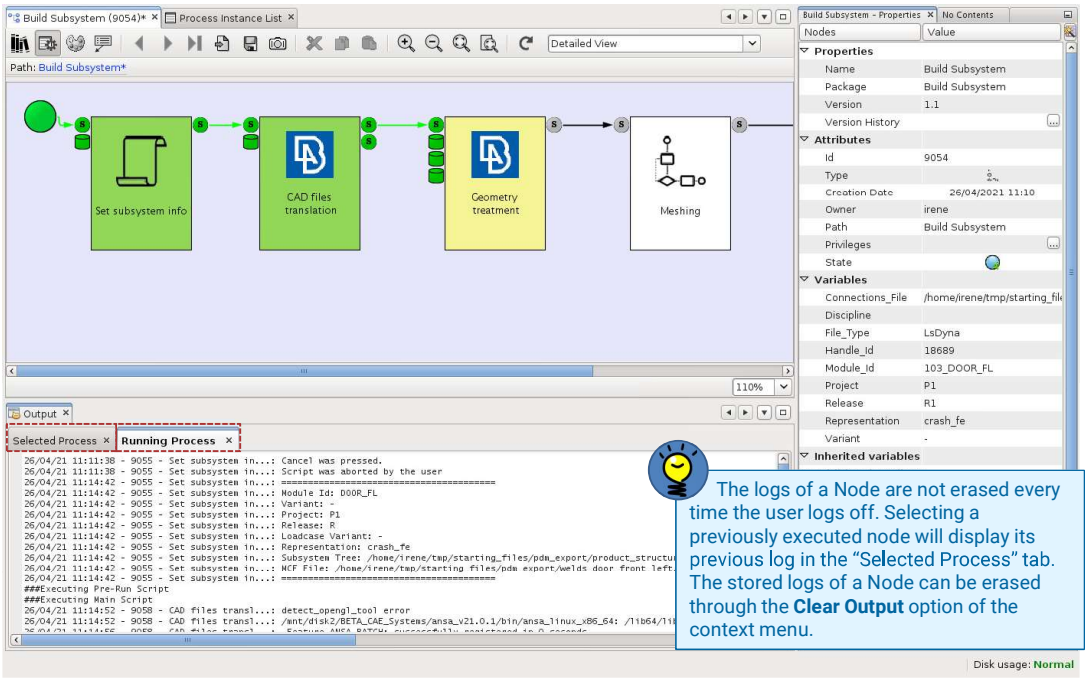


Output Window

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 SPDRM window. The Output is accessible in two different tabs:

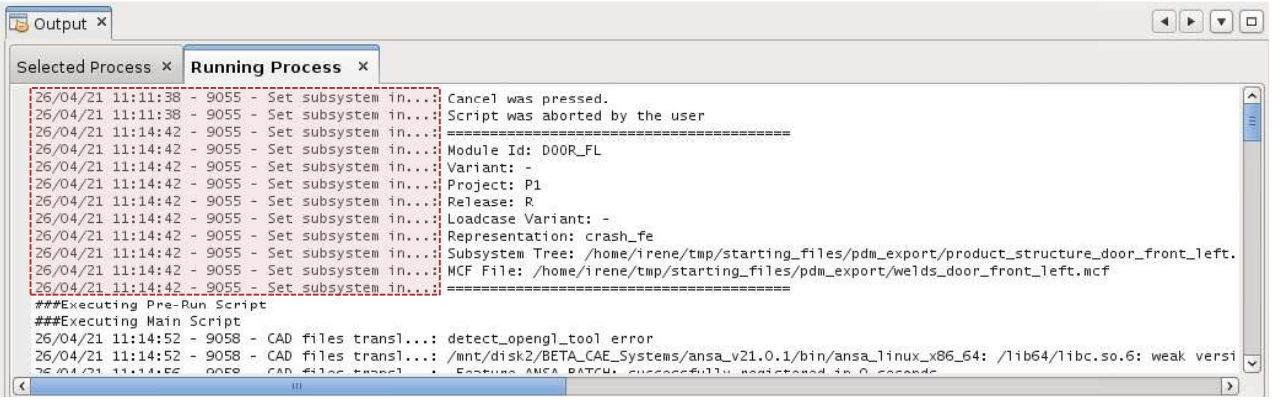
Selected Process: This tab is in sync with the selected node or sub-process, both in the Workflow Scene and in the Process Instance List. It shows the logs that have been produced by the selected node or sub-process.

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



The logs of a Node are not erased every time the user logs off. Selecting a previously executed node will display its previous log in the "Selected Process" tab. The stored logs of a Node can be erased through the **Clear Output** option of the context menu.

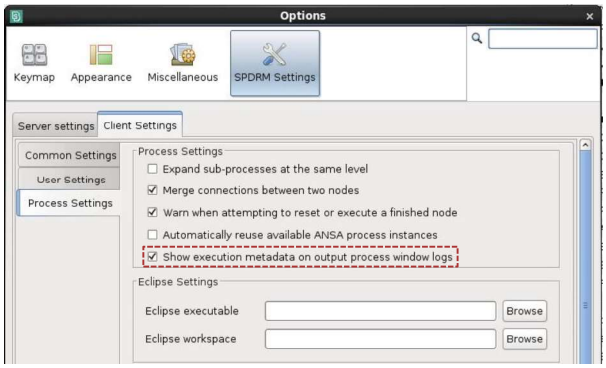
Output Window - Formatting



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 cases where different processes are running simultaneously, as successive output messages may come from different sources.

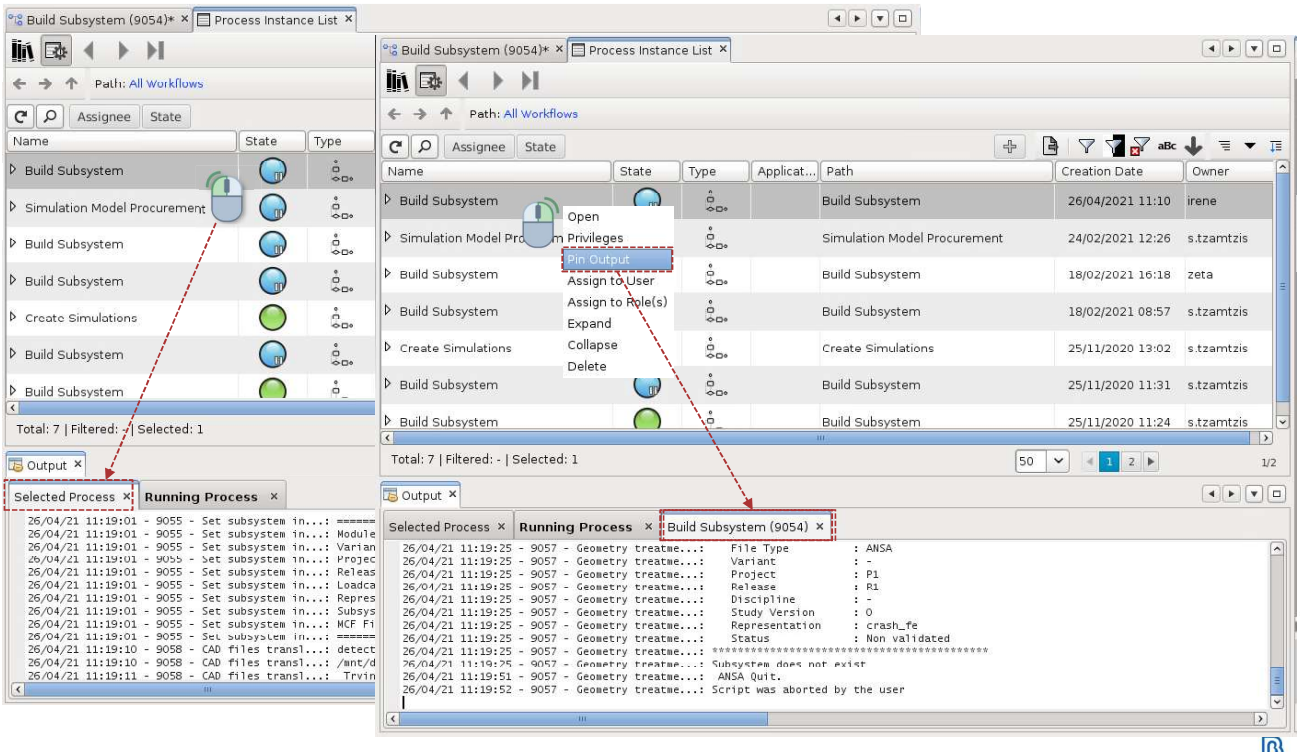
These execution metadata can be suppressed if needed, by deactivating the respective option in the Client Settings of the SPDRM Settings (*Tools -> Options -> SPDRM Settings -> Client Settings*).



The logs of a node from previous executions are stored in memory. Deactivating the **Show execution metadata on output process window logs** setting will only affect new logs and not the ones already produced.

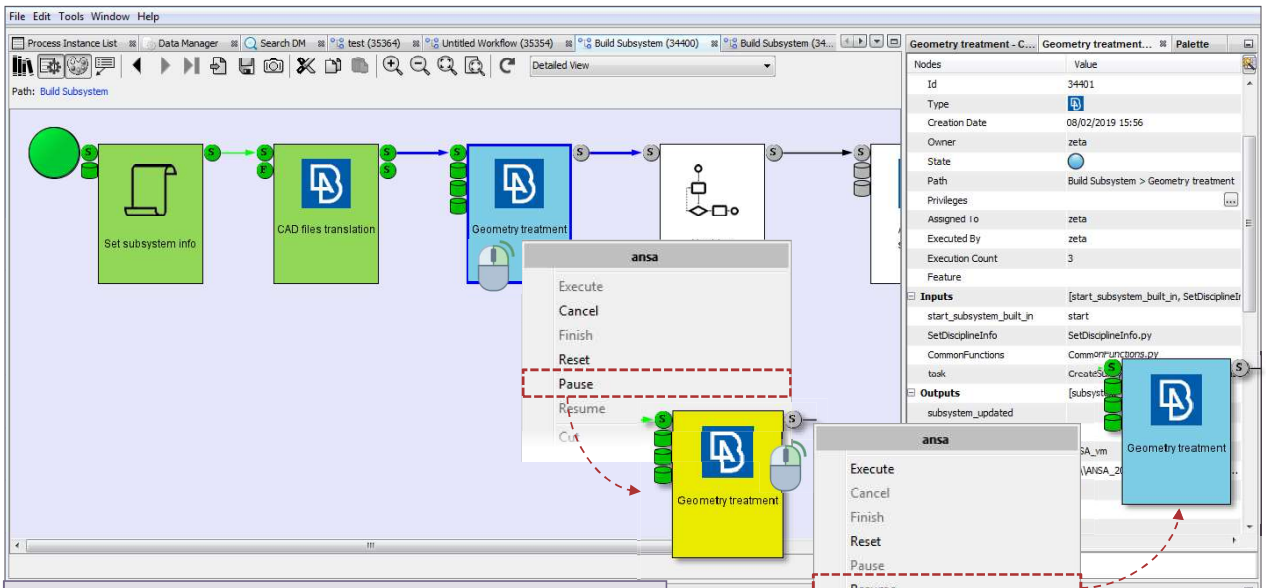
Output Window - Process Output

Selecting a whole process in the Workflow Scene or in the Process Instance List, all node outputs are collected and displayed in the **Selected Process** tab, in order of execution. Additionally, the option **Pin Output** can be used to create a new pinned tab, that will display the output of the selected process as "fixed" content, i.e. not in sync with the selected Process/Node.



Cancel/Pause/Resume the node execution

A running node can be paused or canceled. Script nodes can be canceled and application nodes can be either canceled or paused. The user can resume the execution of paused or canceled application nodes.

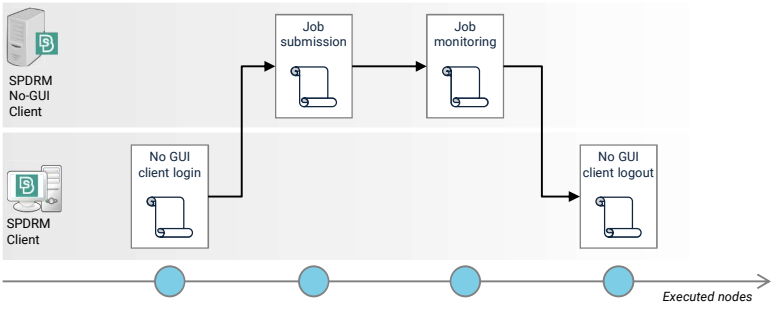


Script node:
Cancel -> node is back to *Ready* state.
Application node:
Cancel -> application is immediately terminated and node's state becomes *Paused*.
Pause -> application remains so as the user can save his/her work and exit. The node's state becomes *Paused*.
Resume -> the paused or canceled application node goes back into running state. If it is required the application will be launched again.

Execute tasks in No-GUI client

SPDRM offers the capability to execute tasks in a No-GUI client. This capability can be used for the execution of heavy and time-consuming jobs, on remote resources (even on different sites of an organization), such as the job submission of a simulation run to the solver, the polling of the solver results, and the post-processing of the results.

The default setup enables the configuration of the SPDRM No-GUI Client in such a way that it runs on a machine different than the one that hosts the SPDRM application server, to avoid its overload. In this case, the executor does not need to make any adjustments. The executing process will be set-up in such a way that the node, which is marked as auto-executed, will be executed by the user remotely utilizing the No-GUI client. The user will be automatically logged in before and logged out after the execution of the node.



Additionally, to the above usage of the No-GUI client, any user can launch, locally or remotely, a No-GUI client so as to execute particular tasks. This is done either by providing the Node Id of the node to be executed, or by providing the path of a DM script to be executed.

To execute a node, the command below should be executed in a command prompt/terminal:

```
./startClient-nogui.sh -J-DnodeIdToExecute=<Node Handle Id>
```

In this case, any custom GUI, generated by script functions within the executed node, can be utilized so as the user to give a possible initial input to the process that will be executed from then on.

Execute tasks in No-GUI client

To execute a script, the command below should be executed in a command prompt/terminal:

```
./startClient-nogui.sh -J-Dspdrm.script=<DM_Path_to_script> -J-
Dspdrm.script.function=<name_of_the_function_to_call>
```

In case several arguments are needed for the proper execution of the custom function, these can be given comma separated as shown below. In this case these are passed to the function as strings.

```
./startClient-nogui.sh -J-Dspdrm.script=<DM_Path_to_script> -J-
Dspdrm.script.function=<name_of_the_function_to_call> -J-
Dspdrm.script.arguments=argument_1,argument_2
```

A local file with arguments can be passed to the command in cases several arguments of various types are needed for the execution of the script. In this case the arguments should be written in json string format. For example, the content of such a file could be:

```
{arg_1':'value_1','arg_2':'value_2'}
```

And the command to be executed is:

```
./startClient-nogui.sh -J-Dspdrm.script=<DM_Path_to_script> -J-
Dspdrm.script.function=<name_of_the_function_to_call> -J-
Dspdrm.script.arguments=@<local_file_with_arguments_in_json_dict_format>
```

Note:

Depending on the OS and the local of the client's running workstation, double quotes may be needed as follows:

```
./startClient-nogui.bat "-J-Dspdrm.script=<DM_Path_to_script>" "-J-
Dspdrm.script.function=<name_of_the_function_to_call>"
```

Or

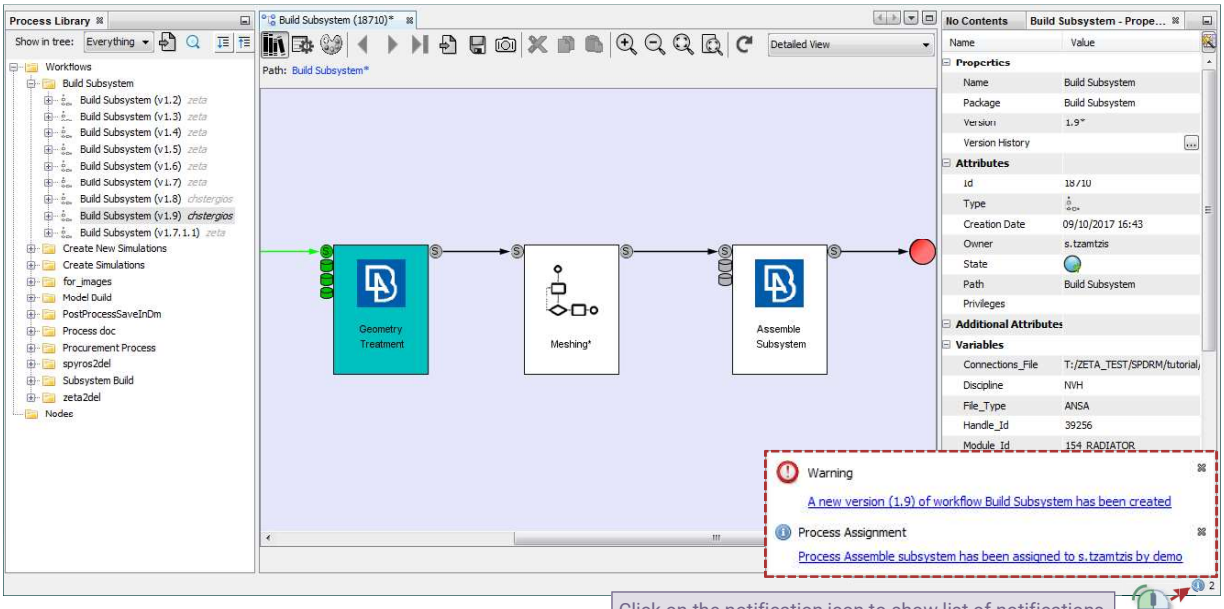
```
./startClient-nogui.bat -J-Dspdrm.script="<DM_Path_to_script>" -J-
"Dspdrm.script.function=<name_of_the_function_to_call>" -J-
Dspdrm.script.arguments="@<local_file_with_arguments_in_json_dict_format>"
```


Notifications

SPDRM offers a default notification mechanism to ensure that each user is notified for changes in the processes and data.

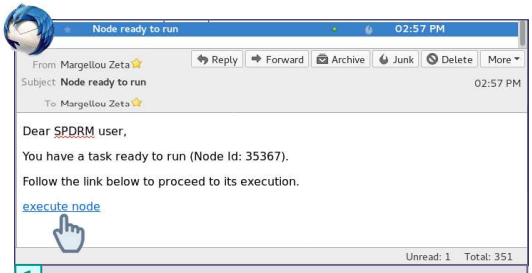
Whenever this notification mechanism is triggered, at the bottom right corner of the SPDRM client a notification icon will appear. By clicking on the icon, a pop-up window opens where different messages are printed notifying the users about:

- Updates of processes which are used by them
- Tasks that have been assigned to them



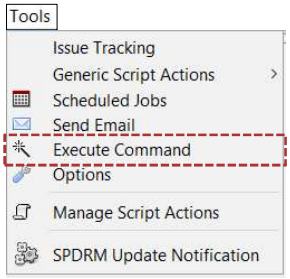
Notification via email upon node status update

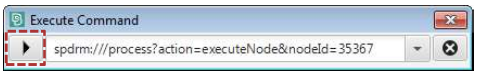
Notifications can be also received in the form of emails, so as to inform the user about an assigned node that has become ready to be executed. In this case the email can contain hyperlinks to view or/and execute the node.



1 The received email contains hyperlink that can be used to execute the node in the SPDRM client. Once pressed, the user can give the appropriate executable (i.e. the SPDRM client executable) so as to launch the SPDRM client and proceed to the execution.

2 Alternatively, the user can copy the link location of the SPDRM hyperlink, and paste it into the **Execute Command** tool of the SPDRM client, in order to execute the command.

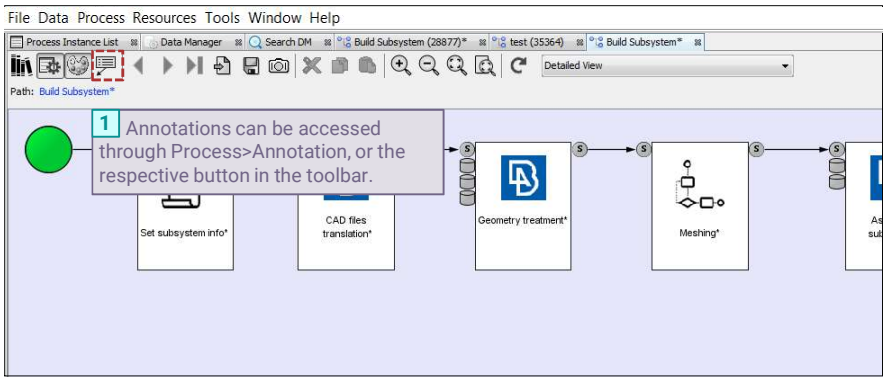




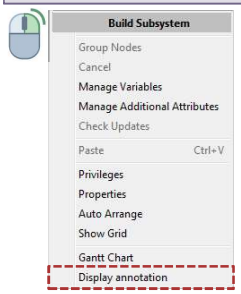
Annotations

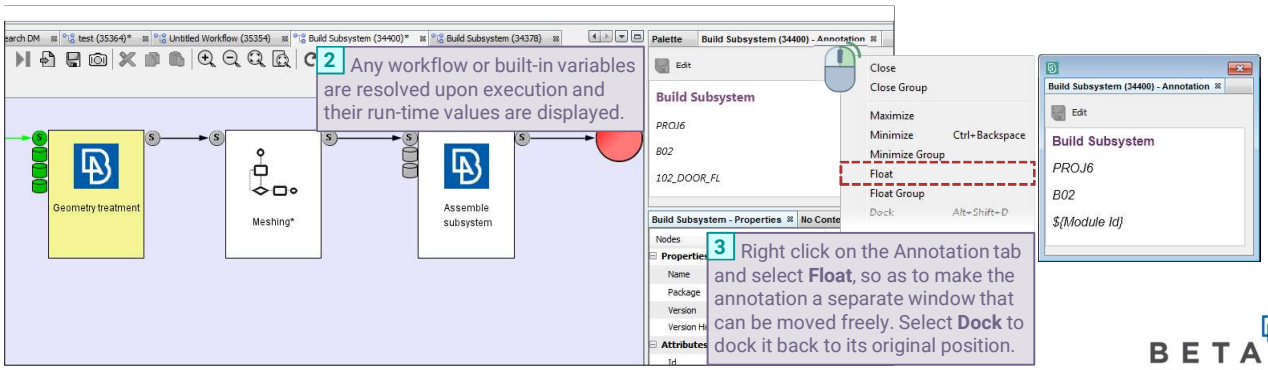
Annotations offer the possibility to display custom information related to a workflow.
The information that can be displayed is:

- Plain text information
- Built-in system variables
- Workflow variables synchronized with runtime values
- HTTP Hyperlinks



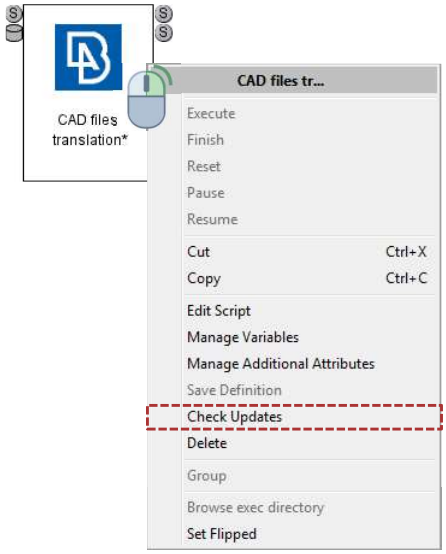
Annotations can be also accessed through the respective option in the context menu of the Workflow scene.

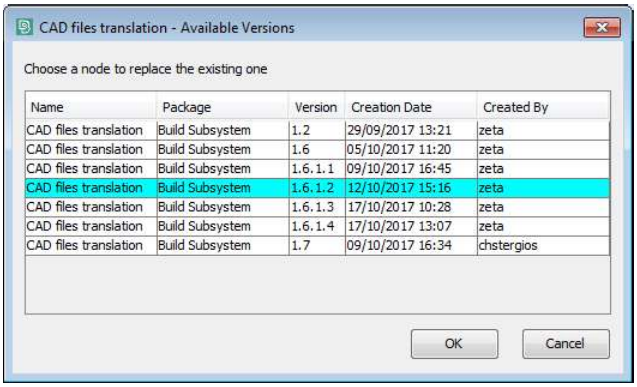




Check for Updates

The user can check for updated versions of a process node by selecting the option **Check Updates** from the node context menu in the *Workflow Scene* window.



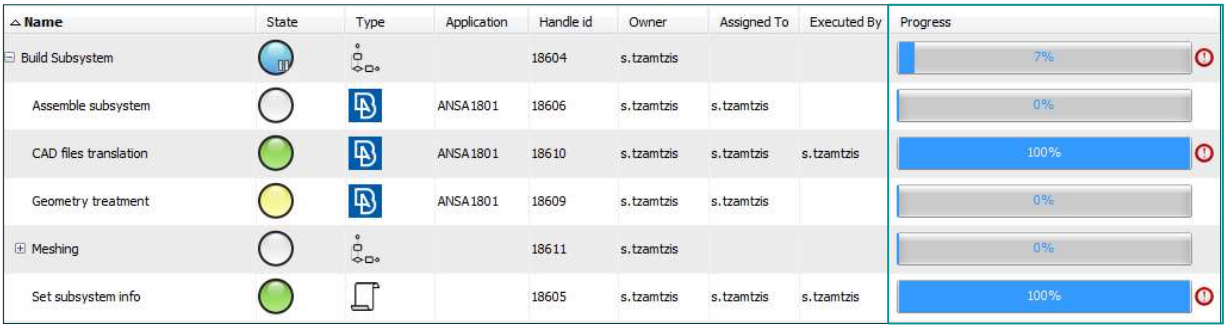


In the *Available Versions* window that appears, a list of alternative versions of the selected node is displayed.

The current version of the node is highlighted and the user can select any of the other available versions, created by any user, in order to update the process.

Progress monitoring

SPDRM offers progress monitoring functionality from the *Process Instance List* window. By enabling the *Progress* column, the user can have an overview of the overall process progress, as well as the progress of individual nodes in a process.



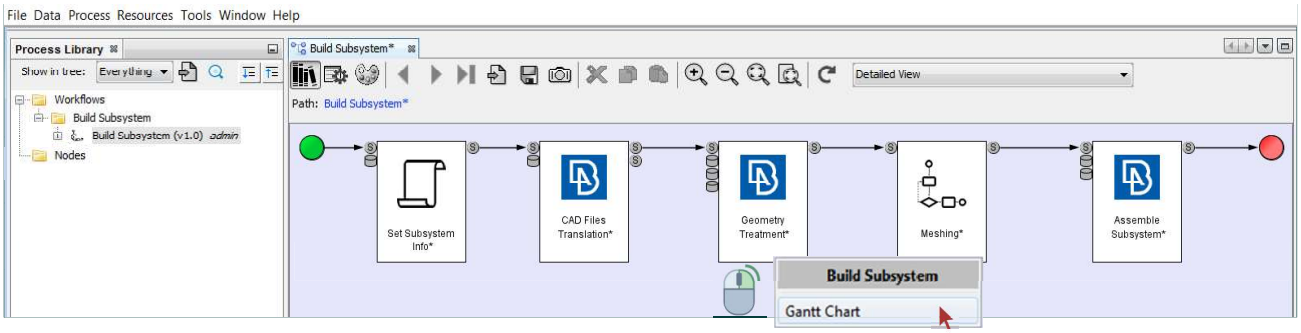
SPDRM will automatically calculate the progress of processes and nodes based on the following criteria:

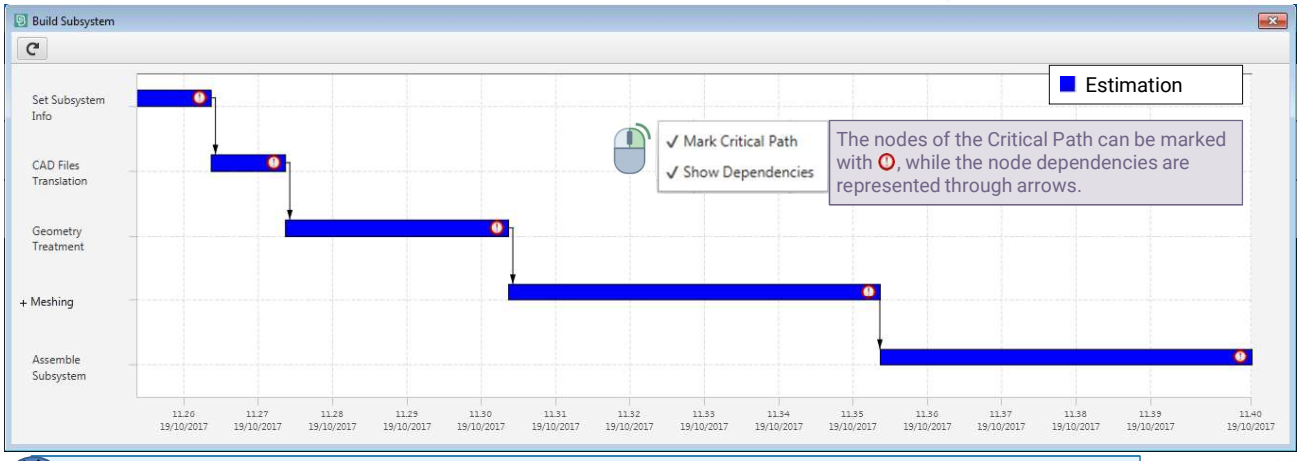
- If the *Estimated Duration* of a node was defined in the Properties card, SPDRM will calculate the node progress as a fraction of the execution time over the estimated duration.
- If a node has no *Estimated Duration* defined, SPDRM will automatically set the progress to 99% when the status of the 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 SPDRM using built-in algorithm based on the progress of its nodes and sub-processes.

When the execution time of a node exceeds its estimated duration, SPDRM will display a warning icon next to the node in the process instance list, to mark it as overdue. The progress of an overdue node remains at 99% while the node is R*unning* and changes to 100% when the node is *Finished*.

Process Planning

The Process Planning for a defined workflow can be done through the **Gantt Chart**, which is available in the context menu of the workflow. Based on the estimated duration of each node, the user has a preview of the time needed for the process execution.

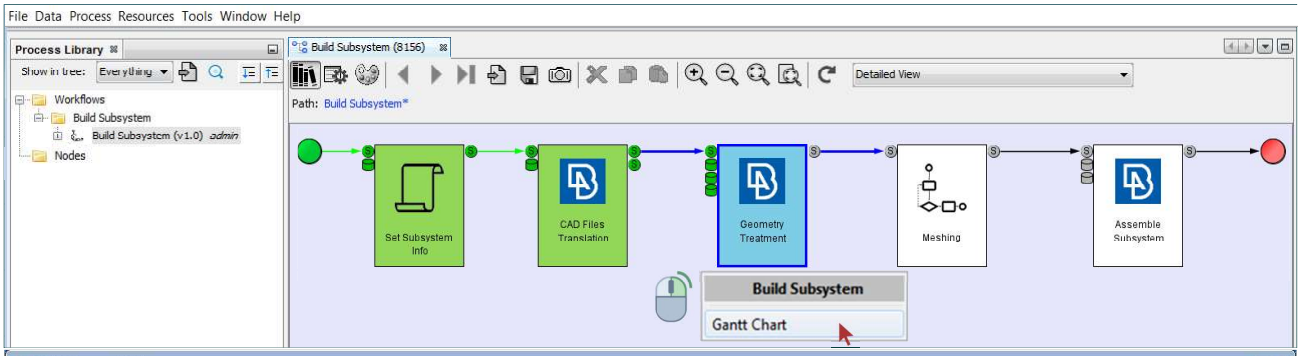


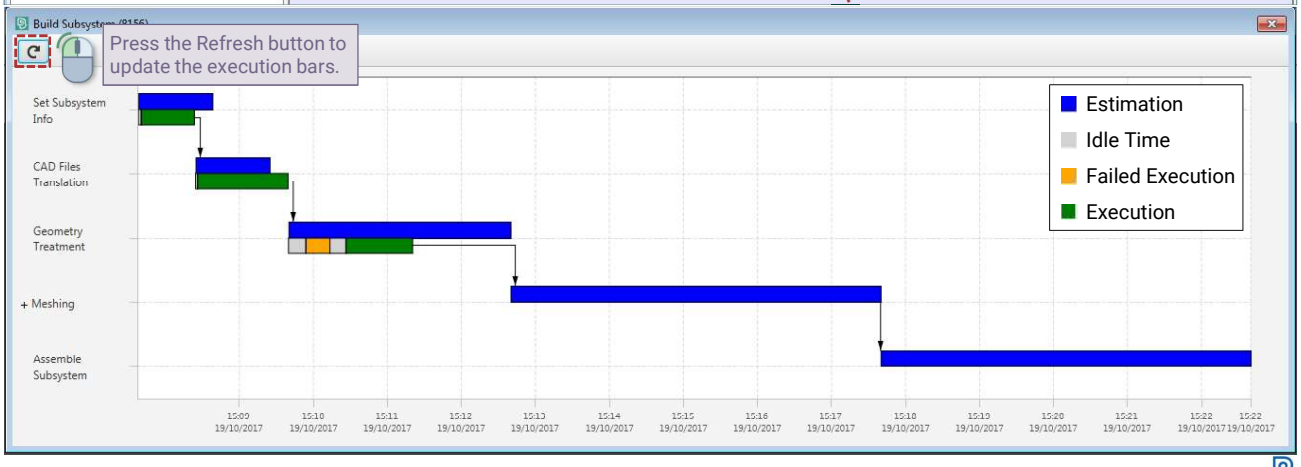


The estimated duration of each node can be either specified manually during the process design or calculated automatically from previous executions of the node.

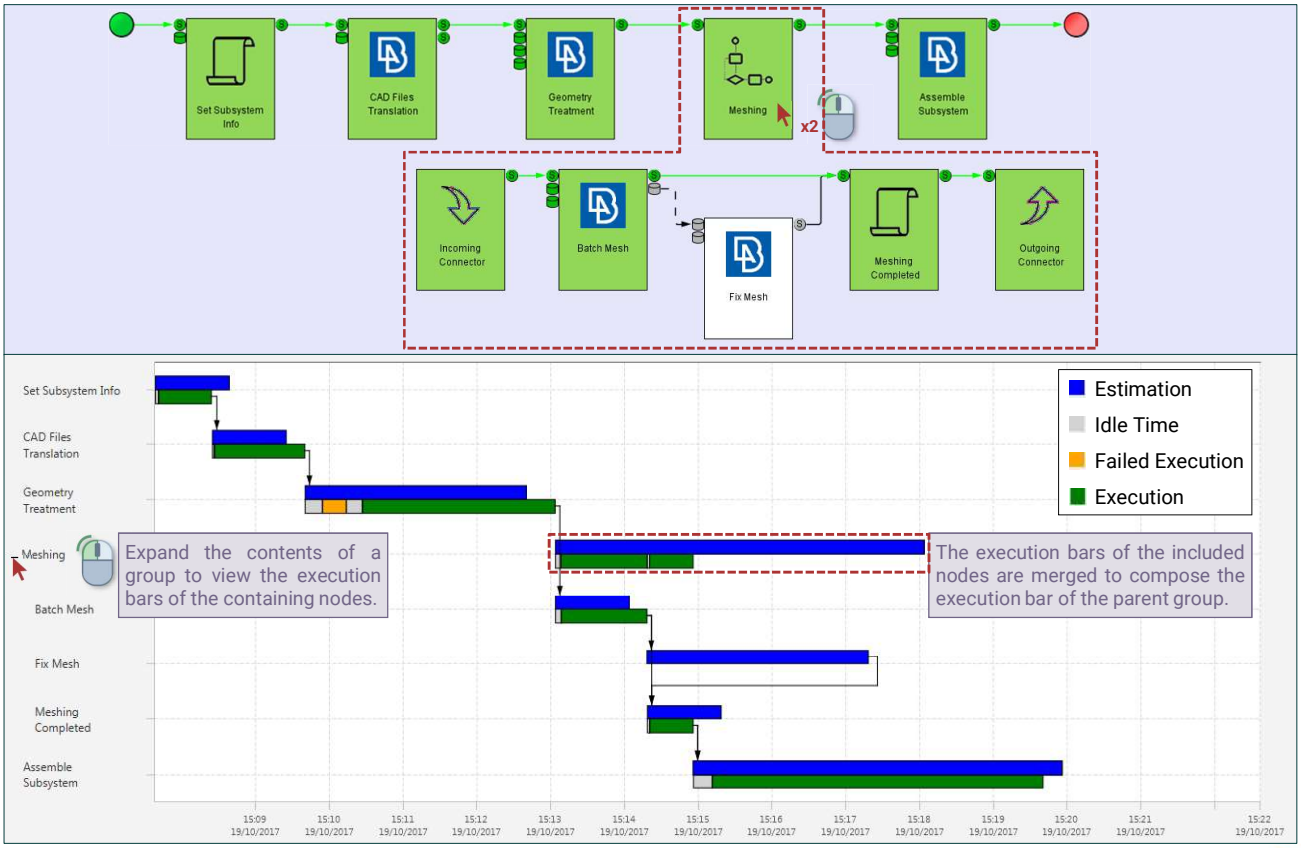
Process Management

The **Gantt Chart** of a running instance provides an overview of its progress. The execution of the completed and running nodes can be inspected, while all the bottlenecks, failures, and potential delays are easily identified.





Process Management



BETA SIMULATION SOLUTIONS

SPDRM
WORKFLOW MANAGER

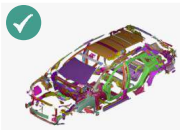
Data Management

BETA
SIMULATION SOLUTIONS

Table of Contents

Data Management Capabilities of SPDRM

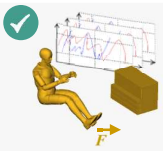
SPDRM



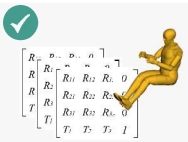
Parts & Connections

Subsystems

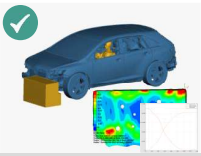
Simulation Models



Library data



Loadcases



Simulation Runs & Results

Key features:

- Data traceability
- Version control
- Security
- Sharing with team and external suppliers
- Data archival
- Customizable data types

BETA SIMULATION SOLUTIONS

Data Management Capabilities of SPDRM

Data Model

- Simulation Run
  - Loadcase
    - Subsystem
      - Part
    - Library file
  - Simulation Model
    - Subsystem
      - Part
    - Library file

The Data Model is the data organization schema used in SPDRM.
All the data types in SPDRM are defined through the Data Model.
The Data Model is :

- Enterprise specific
- Customizable

Security

User Role	view	modify	delete	execute
	✓	✓	🔒	✓
	✓	🔒	🔒	✓
	🔒	🔒	🔒	✓

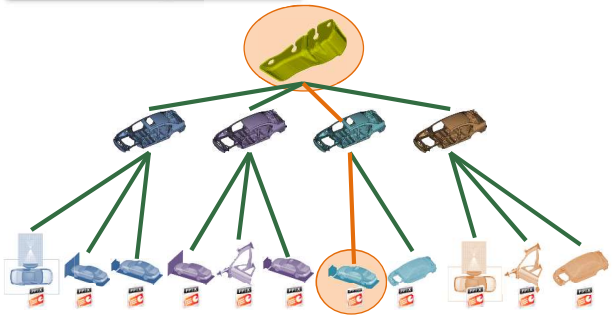
SPDRM controls the accessibility of each data object to protect enterprise data and ensure data integrity.

- Each data object has a list of permissions (**A**ccess **C**ontrol **L**ist)
- ACL is defined per User Role
- ACL can be specified per data object, data type or using default rules

BETA SIMULATION SOLUTIONS

Data Management Capabilities of SPDRM

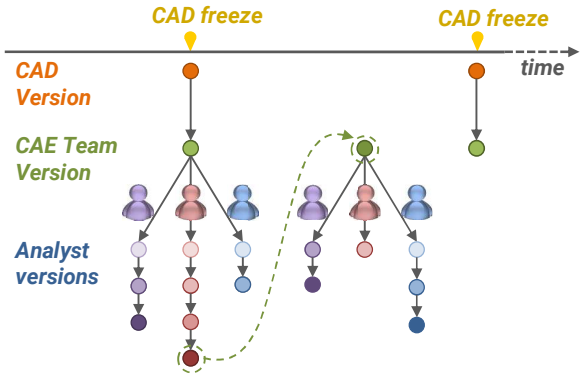
Traceability



SPDRM handles the relationships between data objects. Data tracking in SPDRM can be performed using different types of Links:

- **History Links**: Between different versions of the same object
- **Content-type Links**: When an object is contained in another object (e.g. a Part contained in a Subsystem)
- **Manual Links**: When an object requires another object as input (e.g. a Subsystem that has used a product definition .xml as input)

Version Control



SPDRM offers a Multi-Level Version Control system. Several versioning schemes can be used to capture the evolution of a data object.

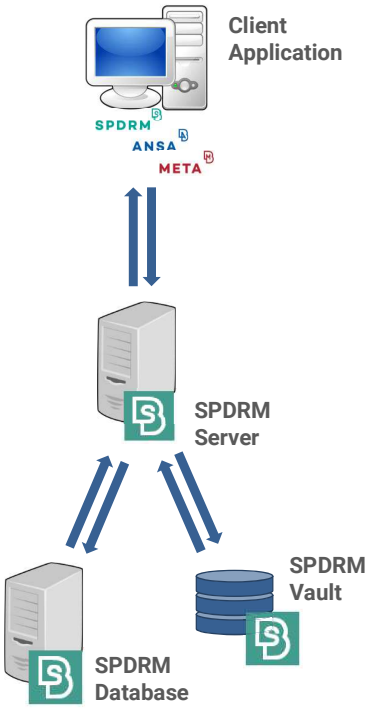
Data Management Capabilities of SPDRM

In SPDRM, the files related to data objects are stored in the Vault while the metadata related to each data object are stored in the Database.

Instead of directly accessing a file through the file browser, the user has to access the data object through the SPDRM Server. The Server fetches the right file from the Vault with the corresponding metadata from the Database.

The Server provides access to the contents of the Vault and Database to client applications. The applications that can act as clients are:

- SPDRM Client
- ANSA
- META
- KOMVOS
- 3rd party client applications



In case of **SPDRM Client**, Data I/O operations are performed through the Data Manager Window, or during Process execution.

In case of **ANSA**, **META** and **KOMVOS**, Data I/O operations are executed through the DM functionality of each application.

In case of the **3rd party client applications**, Data I/O operations are possible through Web Services.

Data Types in SPDRM

Any type of CAE data can be handled through SPDRM. The data types in SPDRM can be grouped in the following categories:

- **DM Items**

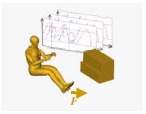
  These are customizable data types of primary data objects (ex. Parts, Subsystems, Simulation Models etc.).

- **Rich Library Items**

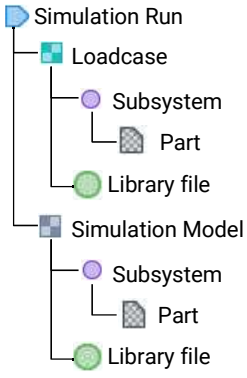
  These are customizable data types of auxiliary data objects (ex. Header files, Dummies, Material databases, etc.).

- **Files/Folders**

  The Properties of these data types are the File Name and DM Path. The Properties and Attributes these data types are fixed and are not customizable. Characteristic examples of simple file/folder data objects are the script files and external libraries.



Data Model Definition



Data Model : The data organization schema used in SPDRM. Each item in the structure corresponds to a data type.

Data Type : Class of data objects characterized by common Properties and Attributes.

Properties: Group of characteristics that constitute the unique identity of a data object in the Database.

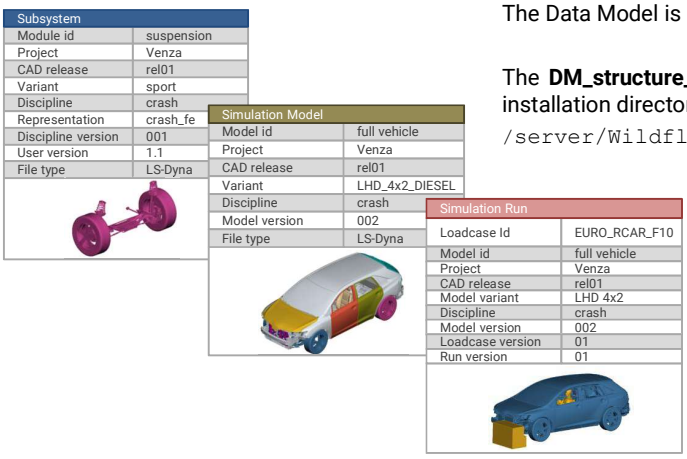
Attributes: Group of characteristics used for attribution.

Data Object : Instance of a specific data type.

The Data Model is defined in the **DM_structure_TBM.xml** file.

The **DM_structure_TBM.xml** file is located in following folder in the SPDRM installation directory:

`/server/Wildfly/standalone/configuration/`



Subsystem	
Module id	suspension
Project	Venza
CAD release	rel01
Variant	sport
Discipline	crash
Representation	crash_fe
Discipline version	001
User version	1.1
File type	LS-Dyna

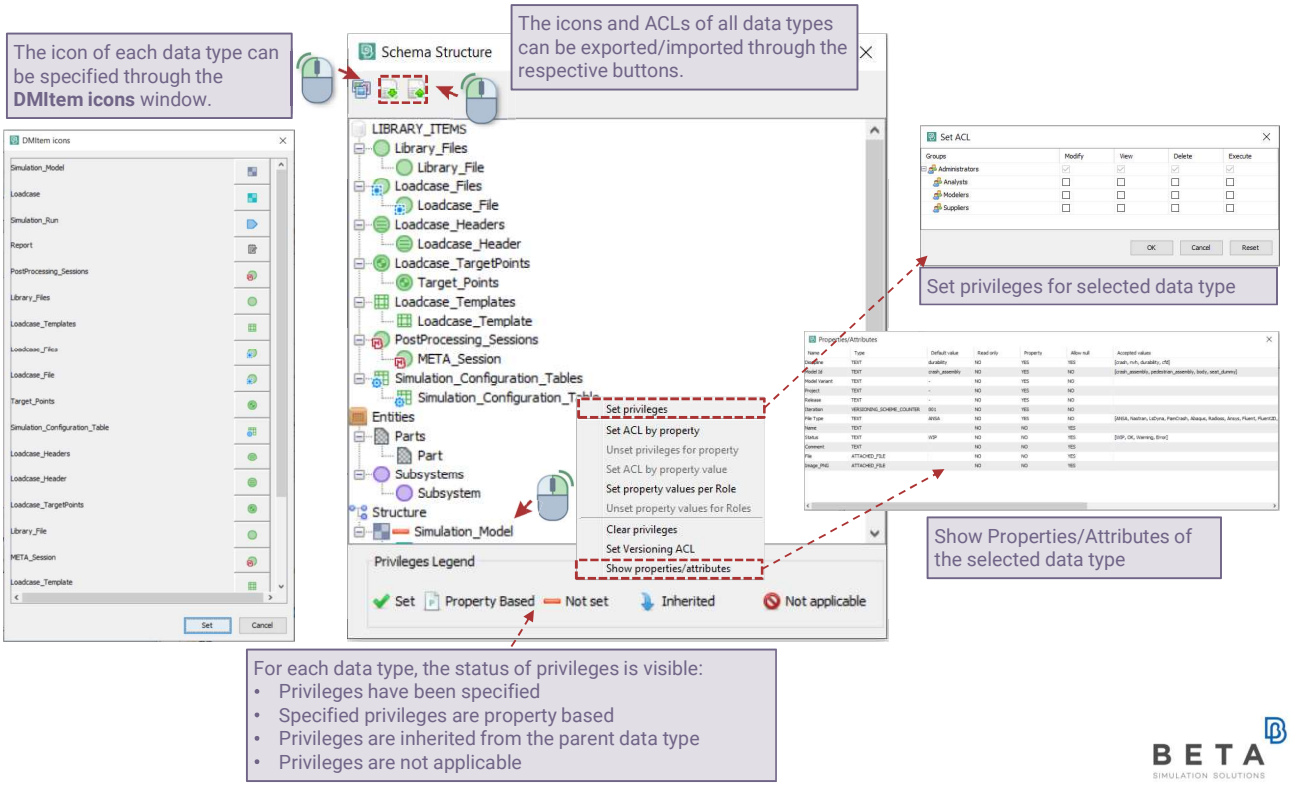
Simulation Model	
Model id	full vehicle
Project	Venza
CAD release	rel01
Variant	LHD_4x2_DIESEL
Discipline	crash
Model version	002
File type	LS-Dyna

Simulation Run	
Loadcase Id	EURO_RCAR_F10
Model id	full vehicle
Project	Venza
CAD release	rel01
Model variant	LHD 4x2
Discipline	crash
Model version	002
Loadcase version	01
Run version	01

The Data Model is communicated to the SPDRM server using the `CreateDMStructureWS` web service.

DM Schema Structure

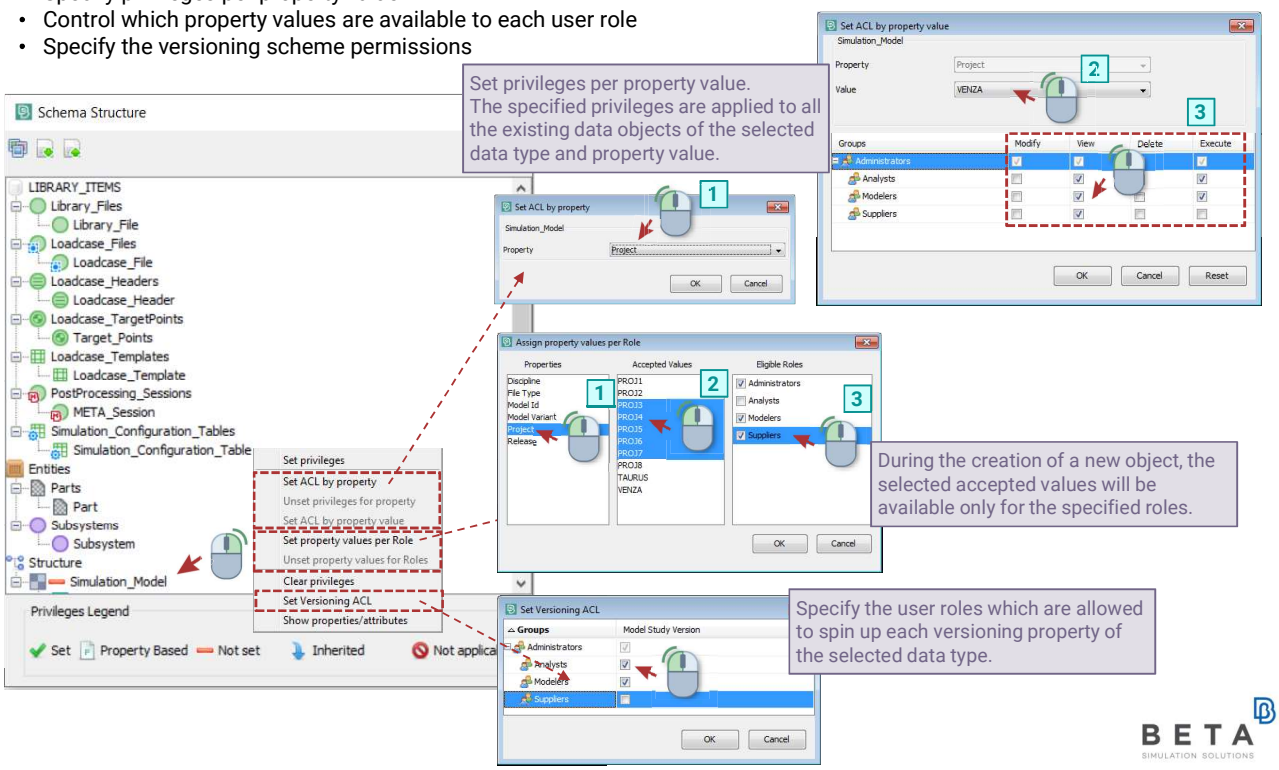
The Data Model can be previewed through *Data > DM Structure (Schema)*. In the **Schema Structure** window all the data types comprising the Data Model, along with their hierarchical relationships are schematically shown. The permissions of each data type under the Structure container can be defined.



DM Schema Structure

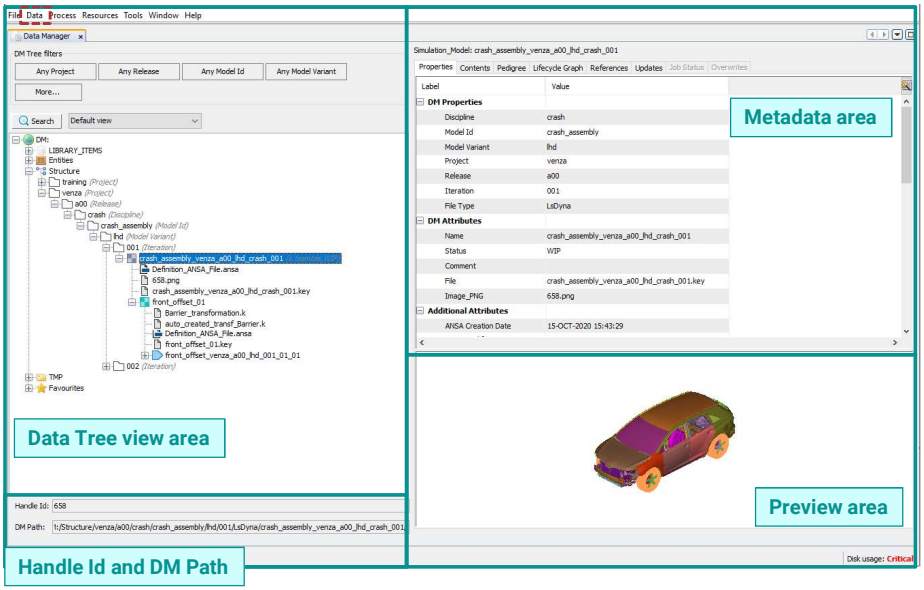
Apart from specifying general ACLs per data type, more advanced options are available through the DM structure (Schema) function:

- Specify privileges per property value
- Control which property values are available to each user role
- Specify the versioning scheme permissions



Data Manager – Window Layout

Invoked through: Data > Data Manager



The **Data Manager** window is the main tool used to access and browse the contents of the database.

In the **Data Tree View area** the contents of the database are displayed in tree format.

The **Metadata area** reports information about the selected data object on the left.

A snapshot of the selected data object is displayed in the **Preview area**, if this is available.

The unique ID number of the selected Data Object is reported in the **Handle** field.

The path of the selected Data Object in the tree structure is reported in the **DM Path** field.

Data Tree View Area

The Data Tree view area contains the following sections:

- **Library Items**

  This section contains simple files and Rich Library Items. Contrary to simple files that only have their file name and path for identification, Rich Library Items also have a set of properties that define their identity. This makes searching and identification much easier.

- **Entities**

  Serves as a pool through which Parts and Subsystems can be accessed.

- **Structure**

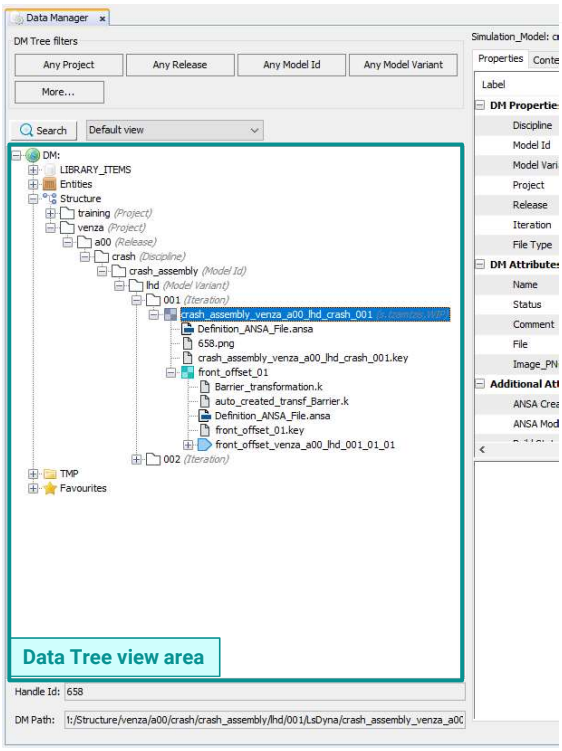
  The tree structure of Simulation Data, where Simulation Models, Loadcases and Runs are displayed.

- **TMP**

  Contains the data that have been produced during the execution of an SPDRM process.

- **Favorites**

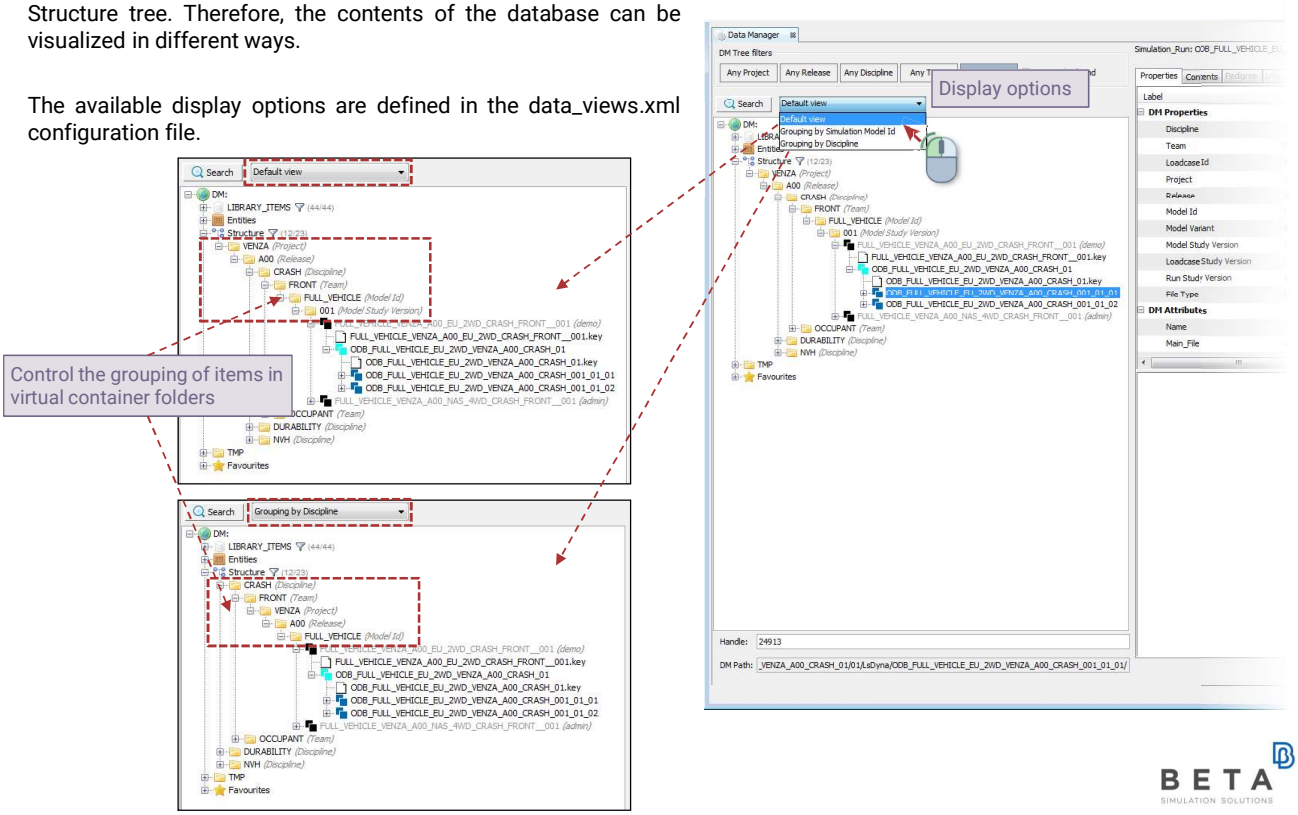
  Frequently used data objects can be placed in this folder, for easy access.



Data Tree View Area – Display Options

Through the available **Display options** it is possible to control the grouping of data objects in virtual container folders under the Structure tree. Therefore, the contents of the database can be visualized in different ways.

The available display options are defined in the data_views.xml configuration file.

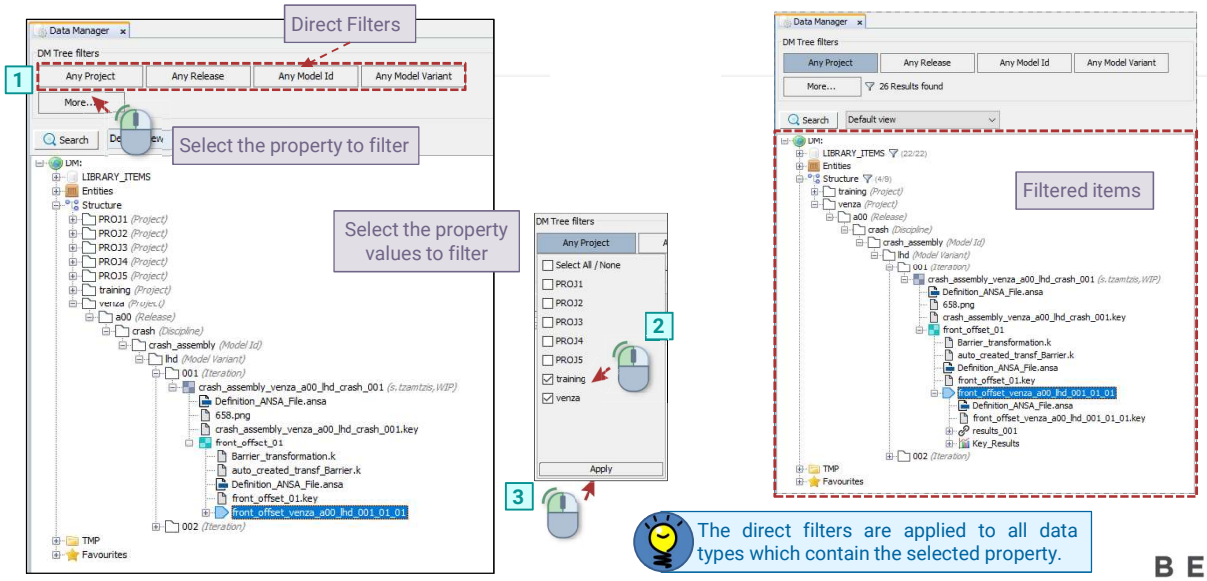


Data Manager – DM Tree Filters

Through the **DM Tree Filters** area it is possible to activate user defined filters for Library Items, Files and DM items that appear in the Data Tree view area.

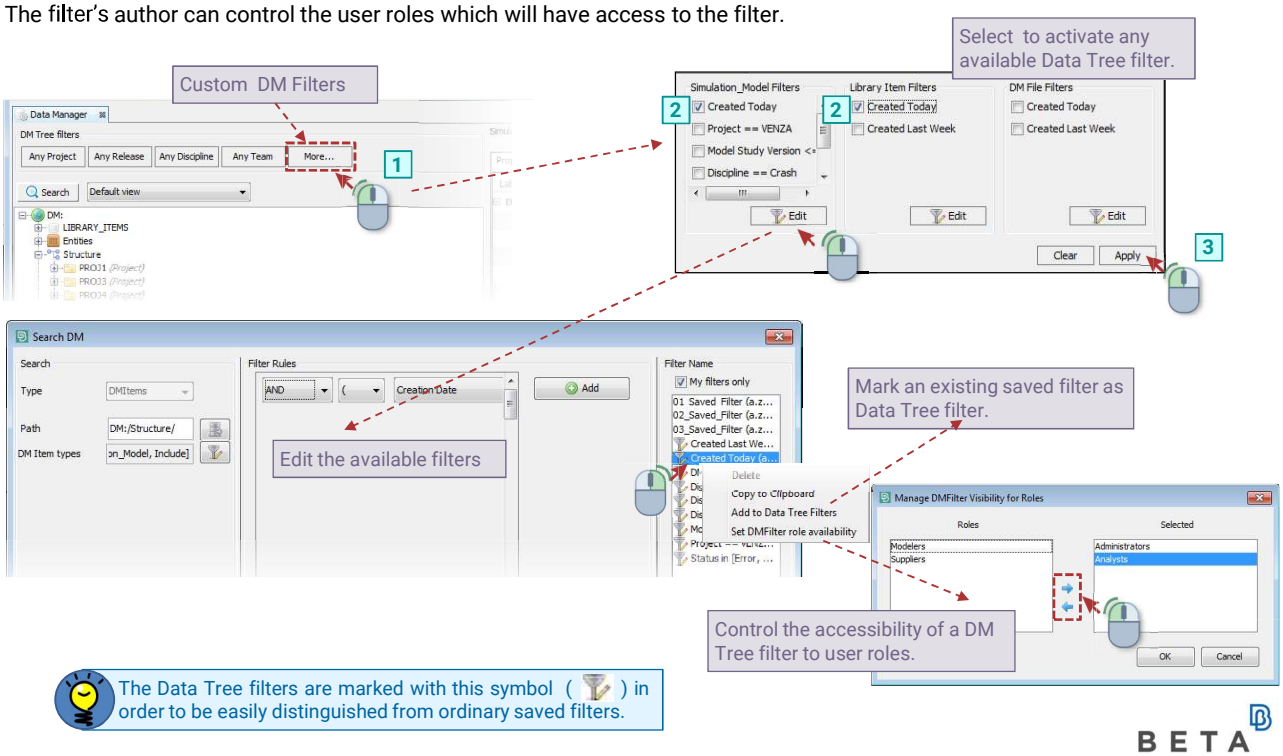
Two types of filters are supported:

- Direct Filters: These filters are property based and they are defined in the data model (dm_structure_TBM.xml) by the administrator. These filters are available to all users.
- Custom DM Filters: These are filters created by the user through the SPDRM client. The filter creator can decide the user roles that will be able to access the filter.



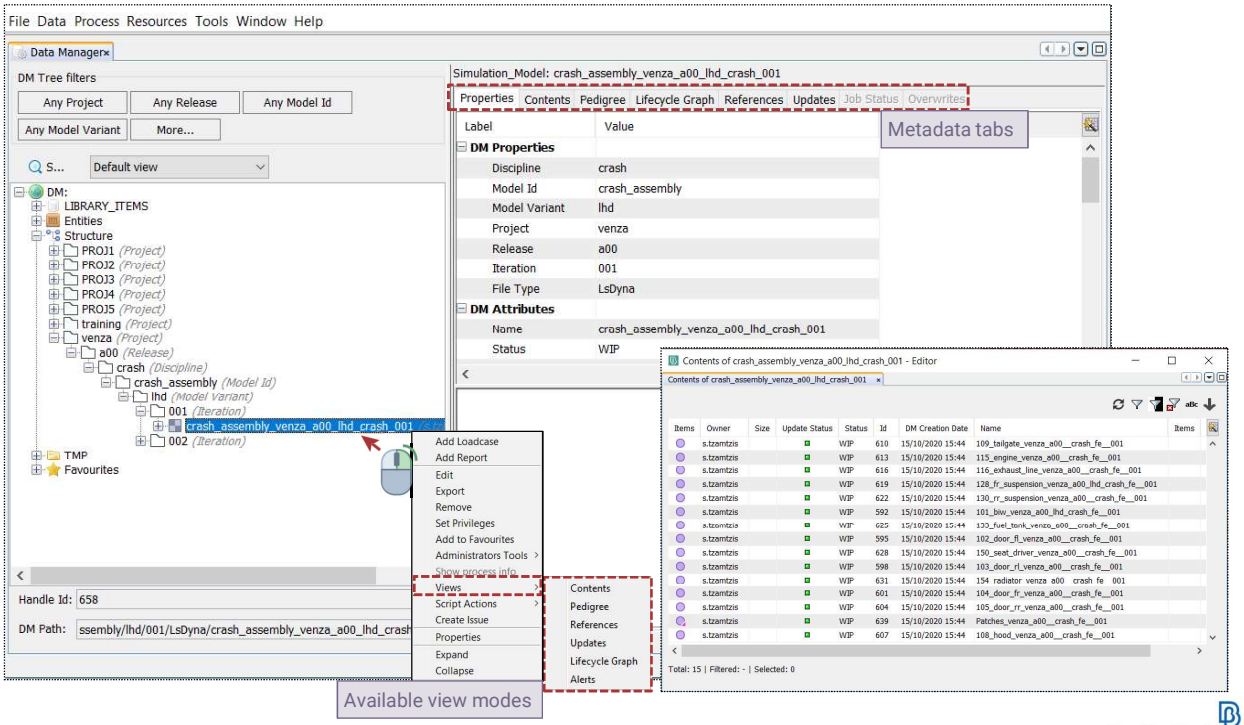
Data Manager – DM Tree Filters

The custom DM filters can be created by each user through the *Search DM* window.
Any saved filter for Files, Library Items or DM Items can be used as a DM Tree Filter using the *Add to Data Tree Filters* context menu option.
The filter's author can control the user roles which will have access to the filter.



View Modes

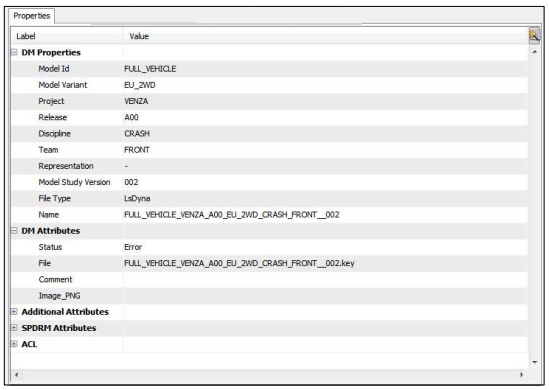
Each data type has certain view modes which can be displayed either as **Metadata tabs** or through the **Views** option in the context menu of the data object.



View Modes

The available view modes are *Properties*, *Contents*, *Pedigree*, *Lifecycle graph*, *References*, *Updates*, *Job Status*, *Overwrites*. The displayed columns in all view modes apart from *Properties* are customizable.

A detailed description follows :



- **Properties**

  The metadata of the selected data object are displayed in a table format under the following categories:

  i. DM Properties: Characteristics that constitute the unique ID of an object.

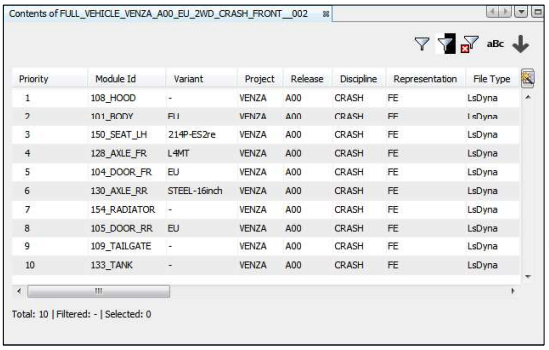
  ii. DM Attributes : Secondary attributes of each object, that are defined in the Data Model configuration file (dm_structure_TBM.xml).

  iii. SPDRM Properties: Default Properties for Files and Folders.

  iv. SPDRM Attributes: Default Attributes

  v. Additional Attributes: Attributes that are either added manually or during the creation of data in SPDRM.

  vi. ACL: Information on the access control of the selected data object.



- **Contents**

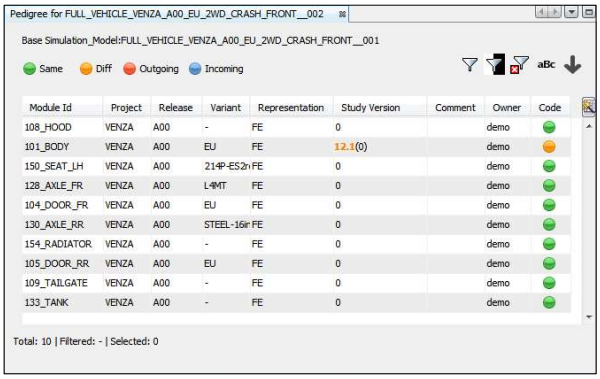
  Displays the contents of the selected data object. This view mode is applicable for data types that contain other data types. For example the contents of a Subsystem are Parts and the contents of a Simulation Model are Subsystems.

  The contents of items are defined automatically every time an object is created from ANSA, e.g. during saving of a Subsystem or Simulation Model. Additionally, contents can be set to a data object through scripting.

View Modes

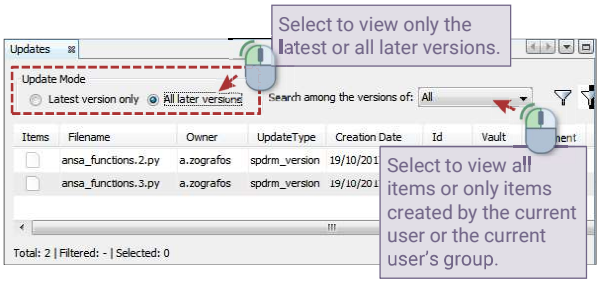
- **Pedigree**

  The pedigree view mode offers a comparison between the selected data object and its parent (previous version). The data objects are compared content wise. Thus, this view offers a quick identification of the similarities and differences between two versions of a data object.



- **Updates**

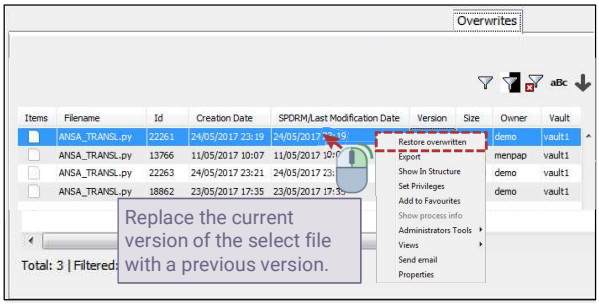
  The available updates of the selected object are displayed. Several filtering options are provided to control the displayed updates.



View Modes

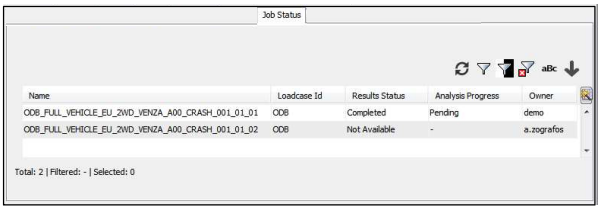
- **Overwrites**

  Displays previous versions of the selected file that are overwritten. An overwritten file can be restored and replace the current version of the file, through the *Restore overwritten* option.



- **Job Status**

  Displays the Simulation Runs that are performed based on the selected Simulation Model.

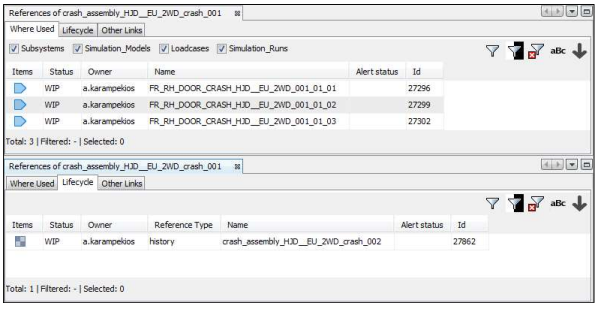


View Modes

- **References**

  Displays a list of all the data objects which are related to the selected item. The types of relations that are available are :

  i. Data objects using the selecting item (**Where Used**)
  ii. Data objects which have a inherited link to the selected item (**Lifecycle**)
  iii. Data objects which have other links to the selected item (**Other Links**)



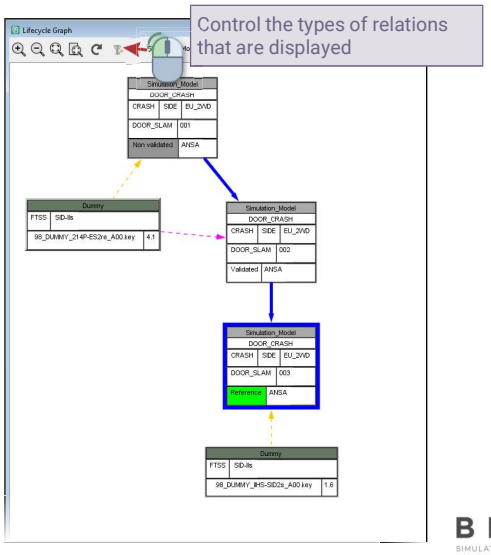
- **Lifecycle Graph**

  The Lifecycle graph displays the evolution of a data object.

  More specifically the lifecycle graphs shows the following types of relationships:

  i. History links
  ii. Contents
  iii. Manual links, which are further categorized in Direct and Inherited

  More information on Lifecycle graph can is available in the dedicated Lifecycle Management paragraph.

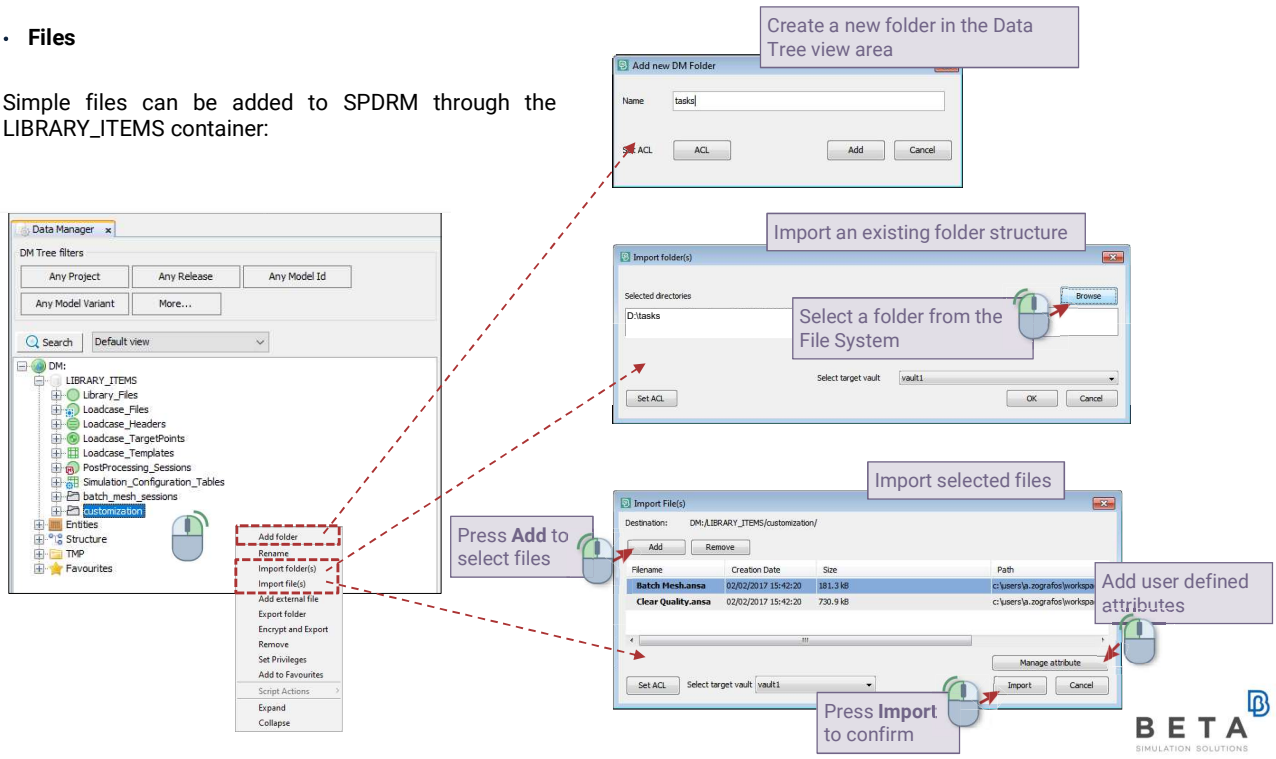


Adding data in SPDRM

Data objects can be added in SPDRM through the SPDRM Client, ANSA, META, Komvos and 3rd party applications. This document will focus on the addition of objects through the Data Manager window of the SPDRM Client.

- **Files**

Simple files can be added to SPDRM through the LIBRARY_ITEMS container:

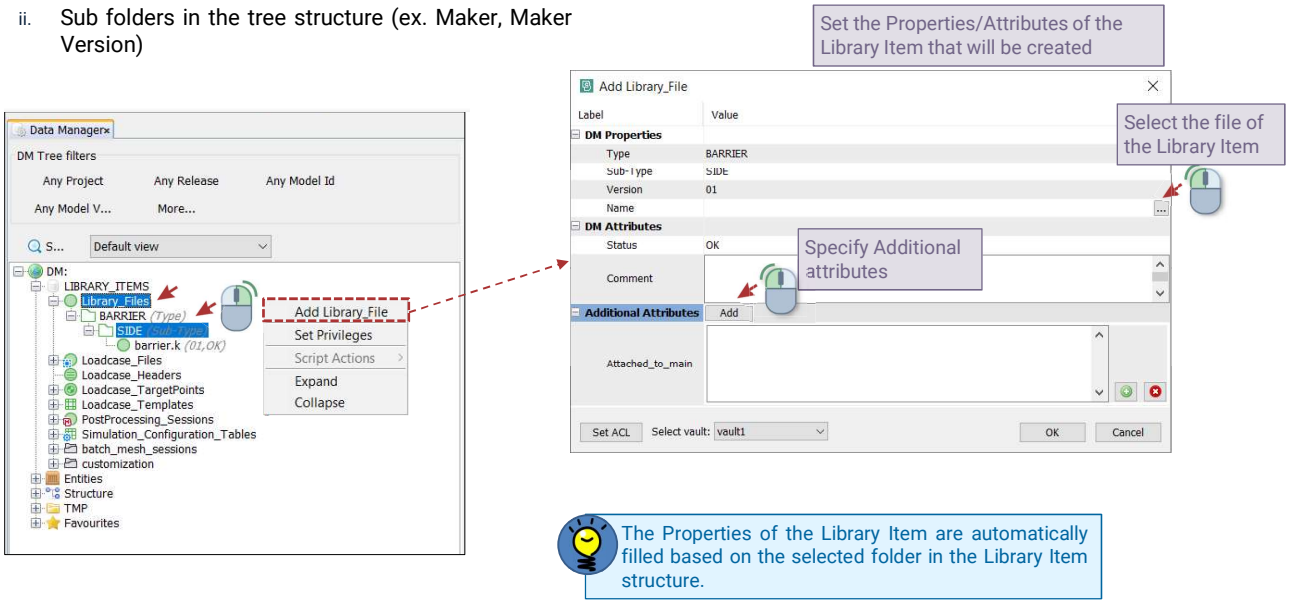


Adding data in SPDRM

- **Rich Library Items**

Rich Library Items can be added to SPDRM through :

i. Top level container that corresponds to each Library Item Data Type (ex. Barriers, Dummies).
ii. Sub folders in the tree structure (ex. Maker, Maker Version)

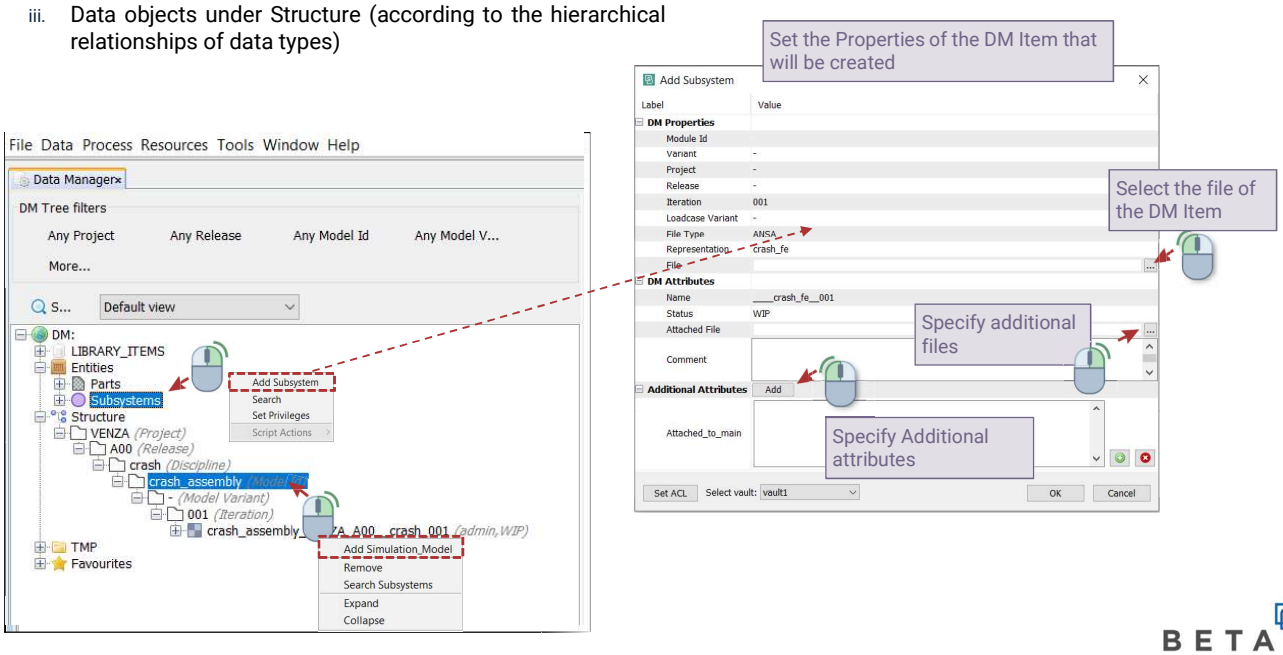


Adding data in SPDRM

- **DM Items**

Depending on the data type, DM Items can be added to DM through :

i. Containers under Entities pool (ex. Parts, Subsystems)
ii. Top level Structure container
iii. Data objects under Structure (according to the hierarchical relationships of data types)



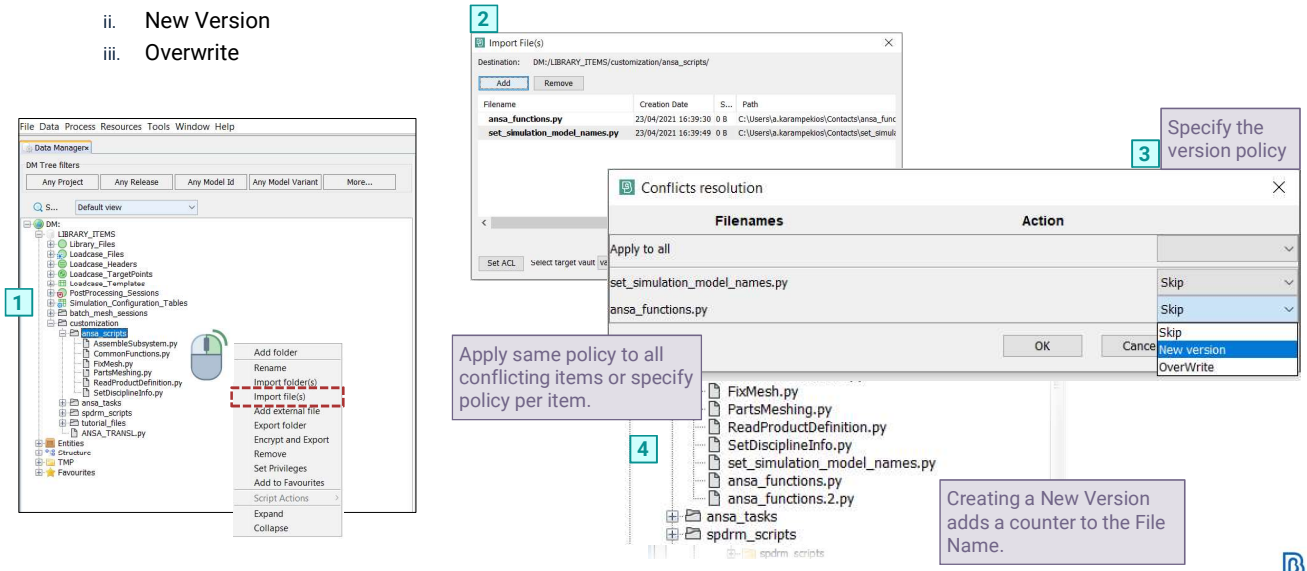
Version Control

- **Files**

During File/Folder import or File Editing, conflicts with existing Files are detected based on the File Name and DM Path.

The user can control the versioning policy through the **Conflicts resolution** window. The available options are:

i. Skip
ii. New Version
iii. Overwrite



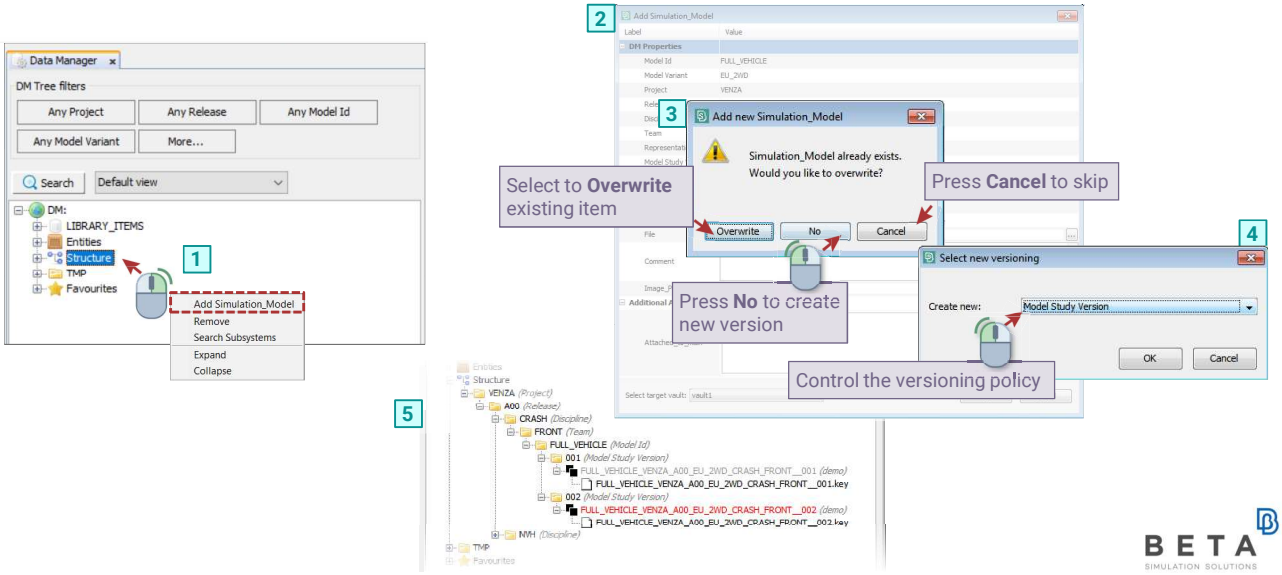
Version Control

- **DM Items and Rich Library Items**

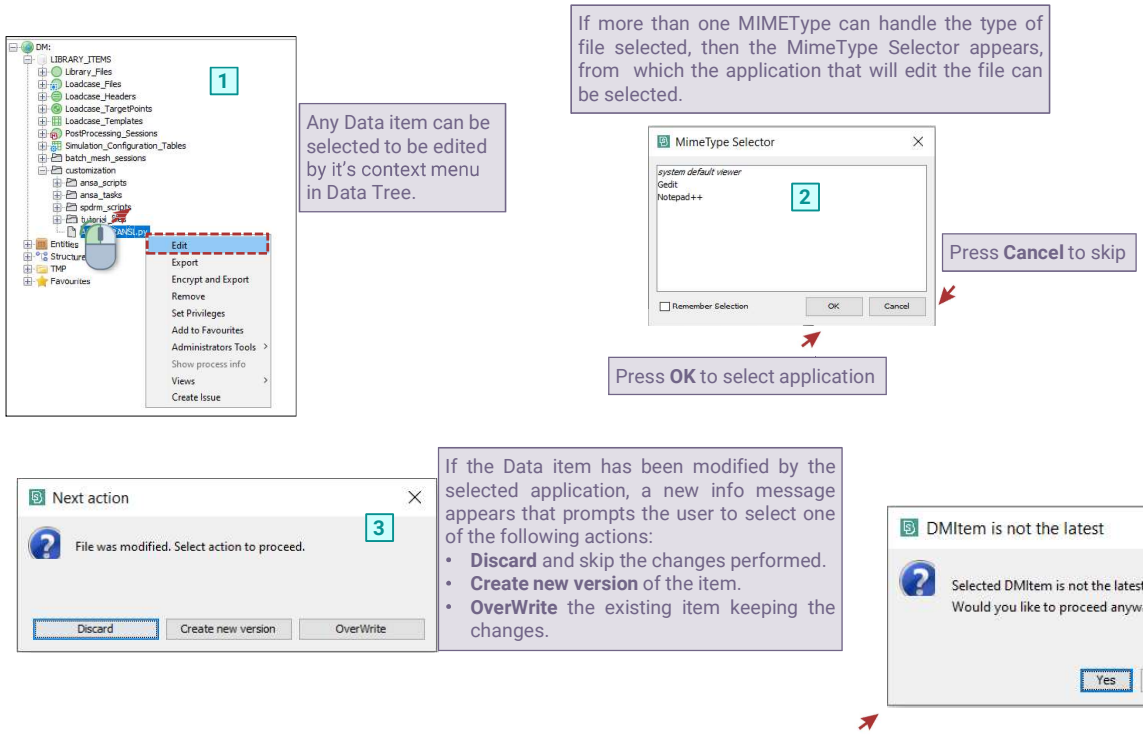
During the creation or editing of Rich Library Items or DM Items, conflicts with existing data objects are detected based on Properties. The user can control the versioning policy. The build-in available options are:

i. Skip
ii. Overwrite

Apart from the build-in options, depending on the available versioning attributes in the definition of the data type, other options may be available (ex. new Version, Revision, Study Version etc.)



Editing Data in SPDRM



A version check option can be defined in Data Model Configuration and can be used alongside any versioning property or attribute, that activates a warning window each time a user attempts to edit an item of that type and the selected item is not the latest version.

Editing Data in existing ANSA or META session

An internal mechanism exists that keeps track of every ANSA session that has been launched by SPDRM.Auto-reusability is supported in SPDRM when editing an item by it's context menu.

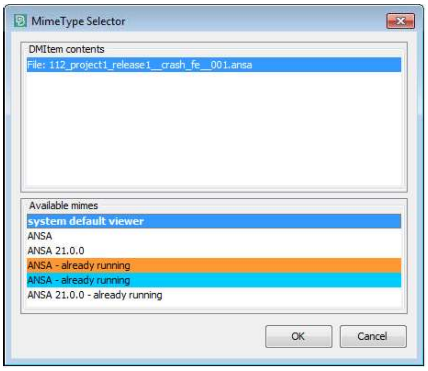
Auto-reusability of ANSA/META can eliminate the launching time of the ANSA and META applications.

When editing a DM Item and an open ANSA or META session already exists, the auto-reusability mechanism interferes and the MIMEType Selector looks like the one below:

In the image on the right the user has the following options:

1. *Edit in a new ANSA session* using ANSA or ANSA 21.0.0 MIMEType.
2. *Edit in existing ANSA session* using ANSA 21.0.0 that already runs or one of the two ANSA sessions that already run.

When option 2 is selected, the merging window of ANSA that was earlier mentioned appears.



The color indication in the MimeType Selector is identical to the color indication that each ANSA or META session has in the toolbar and helps the user detect the session in which the item will open, so that proper actions for the existing files can be performed.

If the selected item opens in an existing session with already existing models, a merging options window for each of the two applications appears, prompting the user to select the actions that would be performed. Those merging options windows, can be seen in the next image of that guide.

Drag 'n' drop data in ANSA or META

Drag 'n' drop functionality enables direct opening of the following data types in ANSA and META:

- Simple files
- Entities
- DM Items
- Rich Library Items

In simple file's occasion the file will be dropped, while in rest of occasions, the main file and the attached files will open in ANSA/META.

Drag 'n' drop can open a file or a DM Item in an already launched session from:

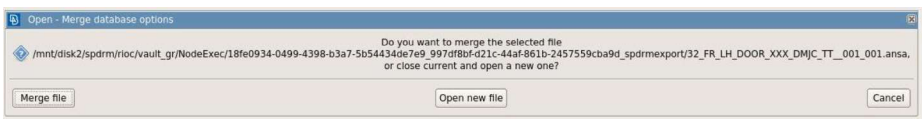
- Data Tree
- Search in DM tool

In cases where attached files or/and attached folders exist for a DM item, those will be dropped also in the selected application, on the same way this will happen for the main file.

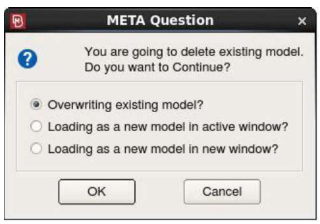
The action is triggered by pushing and holding down the left button of the mouse on the file directly in case of simple file or on the DM Item placeholder in case of DM Item. By dragging the item , when the mouse meets the draw area of the application and the drop icon appears, by releasing the mouse button the contents will be dropped inside the application.

If a file already exists in open session, the merging window of ANSA and META pops-up and user actions is required

ANSA



META



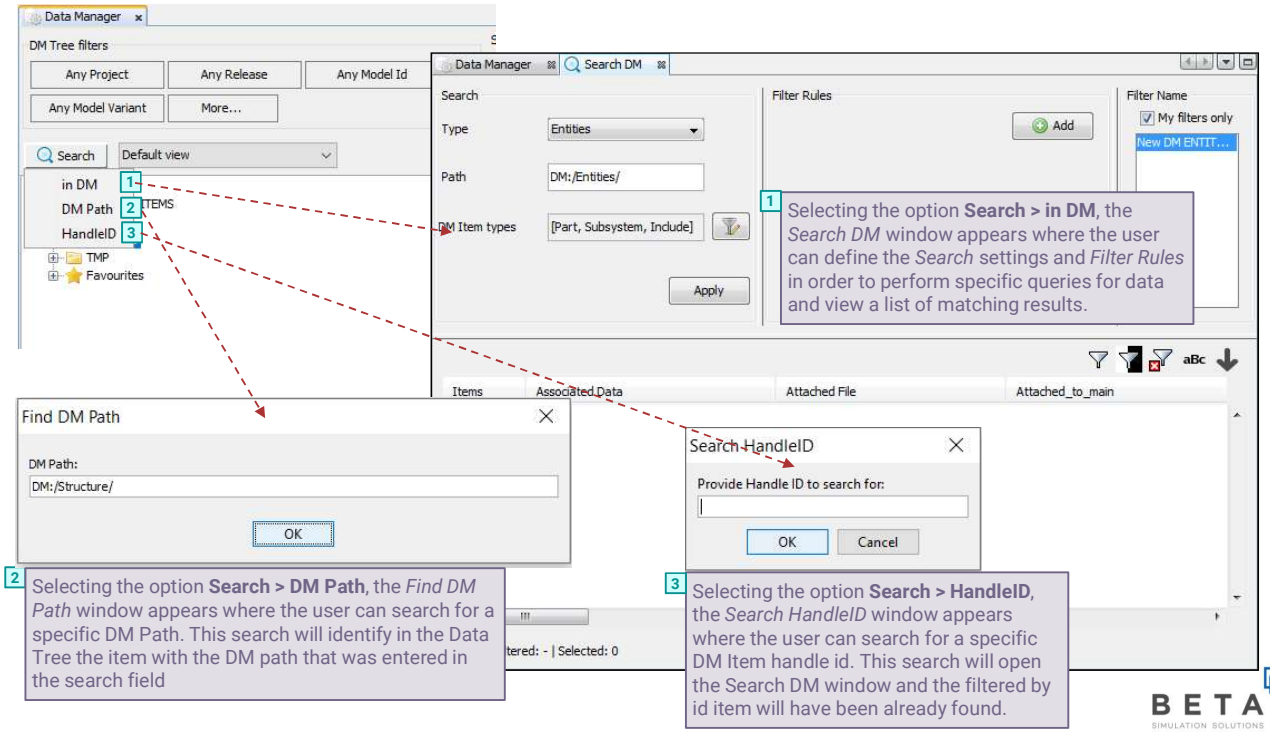
The drag 'n' drop functionality can be used on the exact same way to directly export items in any file system.

Data search

In SPDRM the identification of data by their characteristics is implemented through a search mechanism based on *Queries*.

Data search functionality is available for queries in the 3 different main pools, those of Entities, Library Items and DM Items, with the latter allowing the control of the search for particular data types. The possibility to search for a specific entity based on its DM Path is also provided.

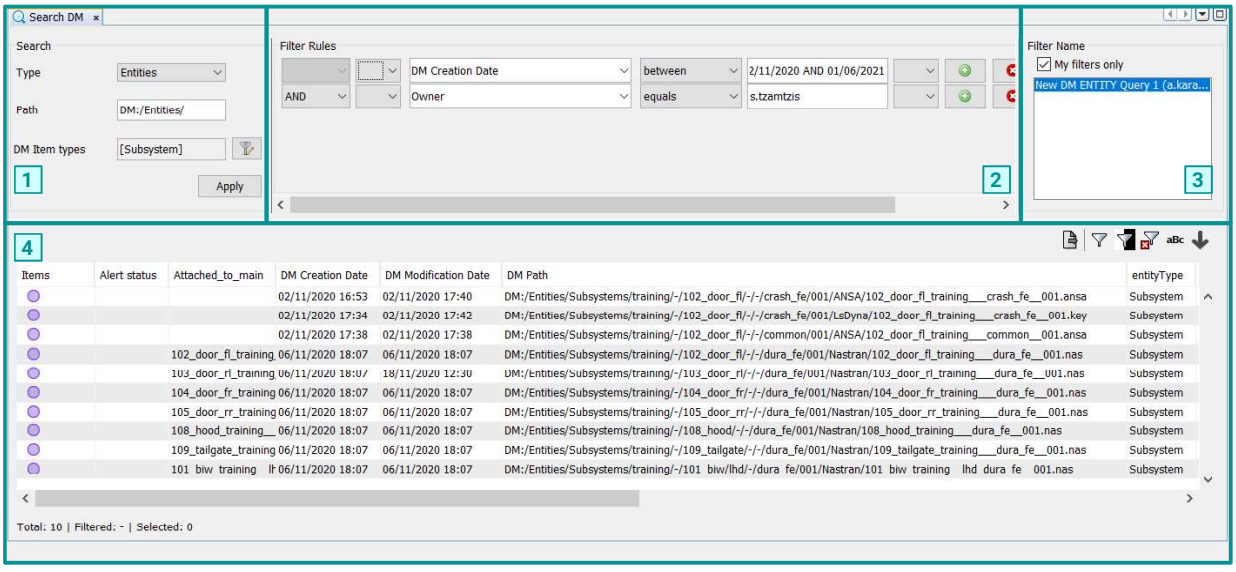
Such functionality is accessed through the **Search** button in the *Data Manager* window.



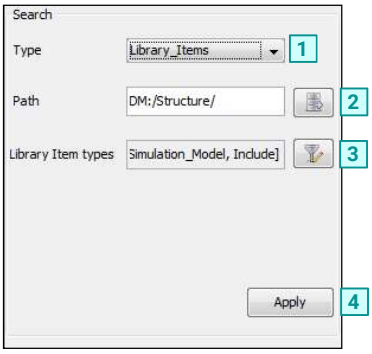
Data search

The *Search DM* window is split in four sections:

1. The *Search* section, where the user can define the search settings for the query execution and **Apply** the query
2. The *Filter Rules* section, where the user can define the filters for the query execution and **Add** new filters
3. The *Filter Name* section, where the user can save specific queries and view existing saved queries
4. The *Results* list, where the user can view the results of the executed query

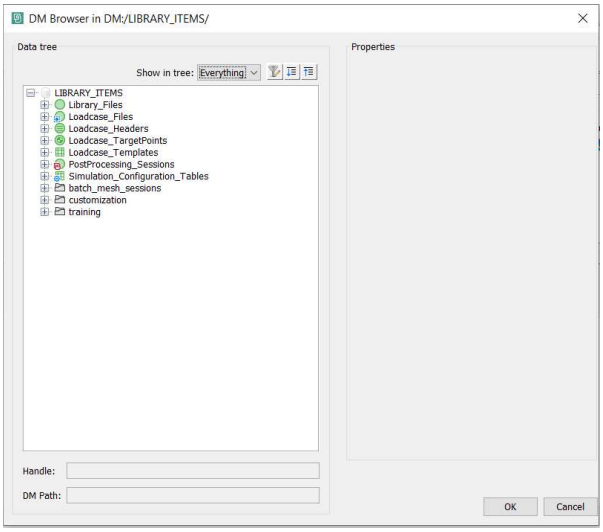


Data search

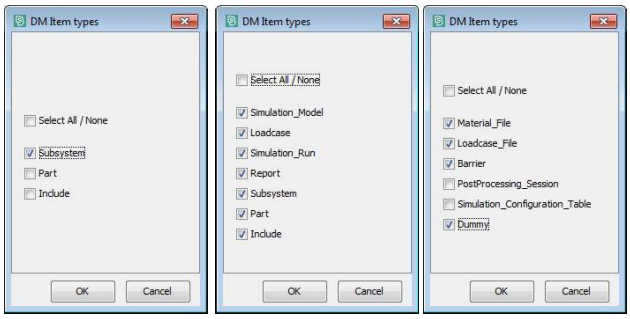


1 In the *Search* section the user can define the *Type* of the data on which the query will be executed.

2 By pressing the **Browse DM path** button, the user can further refine the search by selecting to execute the query on data under a specific DM path.



3 Based on the selected *Search Type*, the user can further refine the *DM Item types* or *Library Item types* on which the query will be executed, by pressing on the **Filter by DM Item type** button.

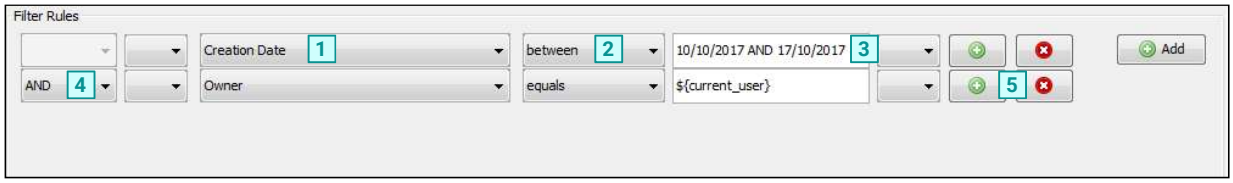


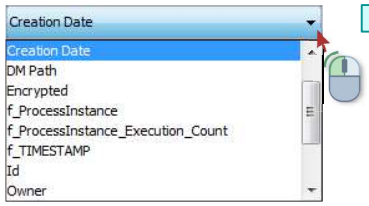
4 Pressing the **Apply** button the defined filters are applied and the SPDRM query is executed.

BETA SIMULATION SOLUTIONS

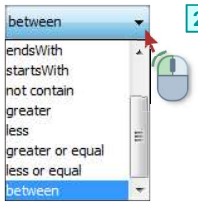
Data search

In the *Filter Rules* section the user can define the filter settings for the query execution.

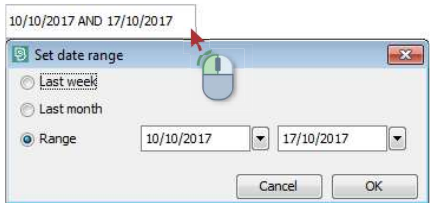




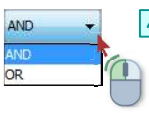
1 The *Properties/Attributes* to be filtered can be selected from a drop down list



2 The filter *Condition* can be defined



3 The user can define the *Property/Attribute Value* to be filtered



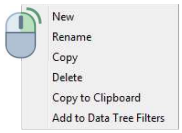
4 In case of multiple filters, the query *Operator* can be defined

5 Filters can be added or deleted using the **Add Filter Rule** (⊕) or **Remove Filter Rule** (⊗) button respectively.

BETA SIMULATION SOLUTIONS

Data search

In the *Filter Name* section a list of saved queries is displayed. The list can display all filters saved by any user in the system, or each user can select to display only his/her queries by activating the *My filters only* checkbox.

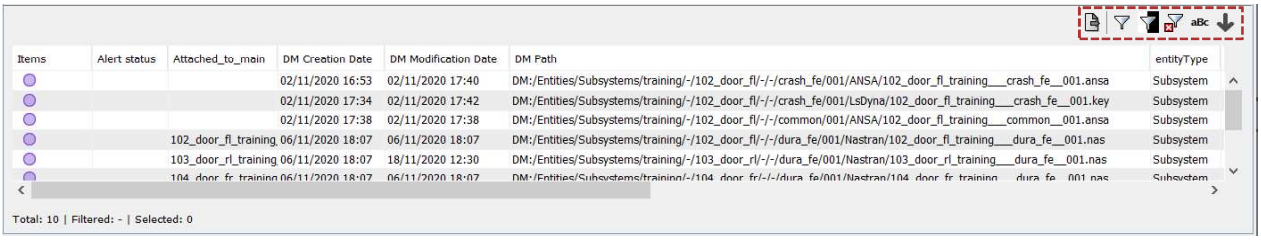


From the context menu of the *Filter name* section, the user can create **New** queries, as well as **Rename, Copy** and **Delete** existing queries.

With the **Copy to Clipboard** option the query command is copied and can be used by the user in scripting applications

Finally, with the **Add to Data Tree Filters** option, the user can add the selected query to the Data manager tree view filters.

In the *Results* list that displays the entities returned by the query execution, the user can browse the filtered entities, while further filtering functionality is available within the list using the respective buttons.



Data Export

All data can be exported from SPDRM to the File System through:

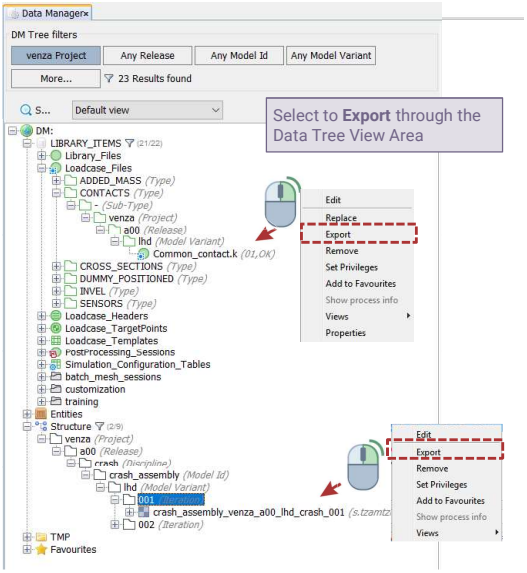
- Data Tree View Area

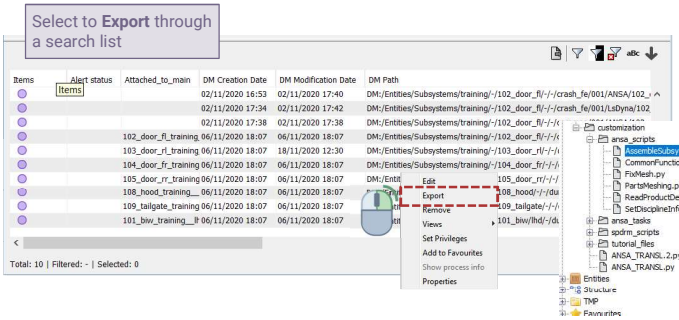
  The **Export/Export Files** options are available through the context menu of all items in the Data Tree View Area.

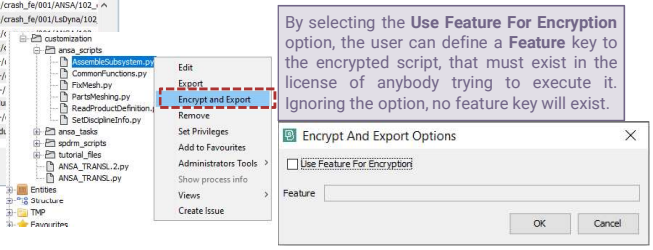
- Search lists

  The **Export** option is available through the context menu of all items that appear in the Data Manager search lists.

The File Browser is automatically launched, enabling the user to select the export location where the file(s)/ folder(s) associated with the selected data objects will be exported.







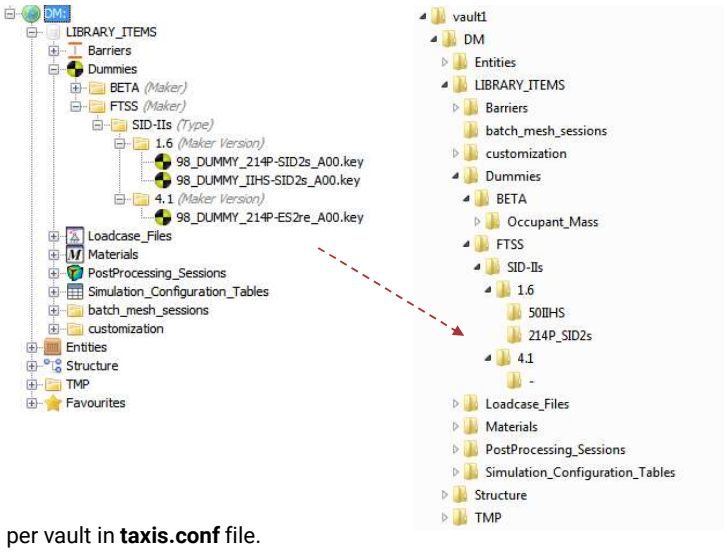
In case of python (*.py*) and jython scripts (*.jy*), an extra option exists in their context menu named **Encrypt and Export**, that encrypts the script (.pyb and .jyb files are exported respectively) apart from exporting it in the selected directory.

DM Structure Export - Introduction

All the files associated with data objects that are handled by SPDRM are stored in SPDRM vault. The SPDRM Client and other applications, like ANSA and META, access these files through the SPDRM Server.

SPDRM can optionally create and maintain a folder structure in the file system where selected components of the data structure will be exported. The exported file structure essentially provides direct access to the file contents of the SPDRM vault to users and applications.

Of course, the contents of the exported file structure are read-only.



The path to the exported file structure can be specified per vault in **taxis.conf** file.
The **taxis.conf** file is located in following folder in the SPDRM installation directory:

```
/server/Wildfly/standalone/configuration/
```

The respective element in the **taxis.conf** file is :

```
<entry key="export_[name of vault]">[path_to_exported_path]</entry>
```

Where :

name_of_vault = the name of a vault described in **taxis.conf**

path_to_exported_path = the path to the exported file structure

DM Structure Export – Control exported items

The contents of the exported file structure can be controlled per vault through the SPDRM client. The respective functionality can be invoked through *Data > DM Structure Export set-up*.

It is possible to select the components of the data structure that will be exported through the *Structure levels* button.

The exported file types can be controlled through the following options:

- **All**

All the items of the selected levels will be exported.

- **None**

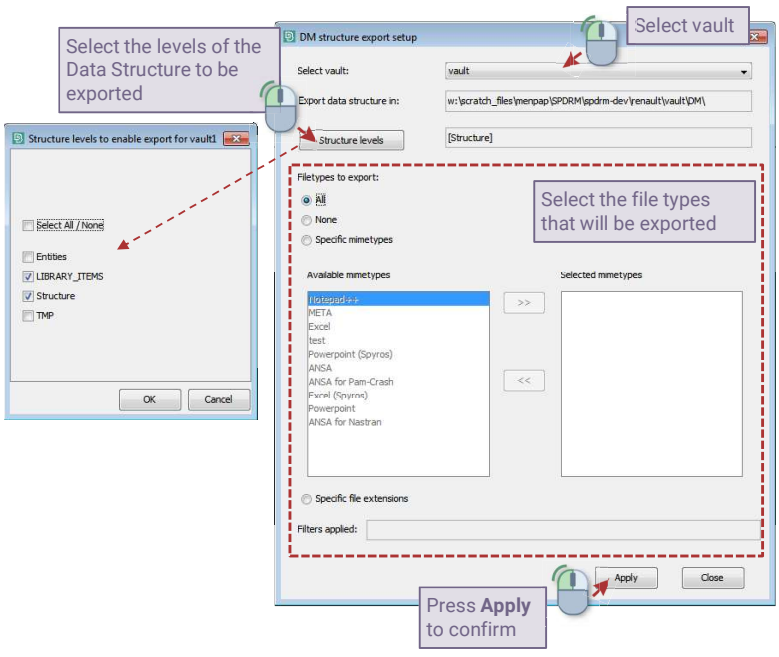
Deactivates the exported structure functionality.

- **Specific mimetypes**

Only files associated with the specified mimetypes will be exported.

- **Specific file extensions**

Only files having the selected extensions will be exported.

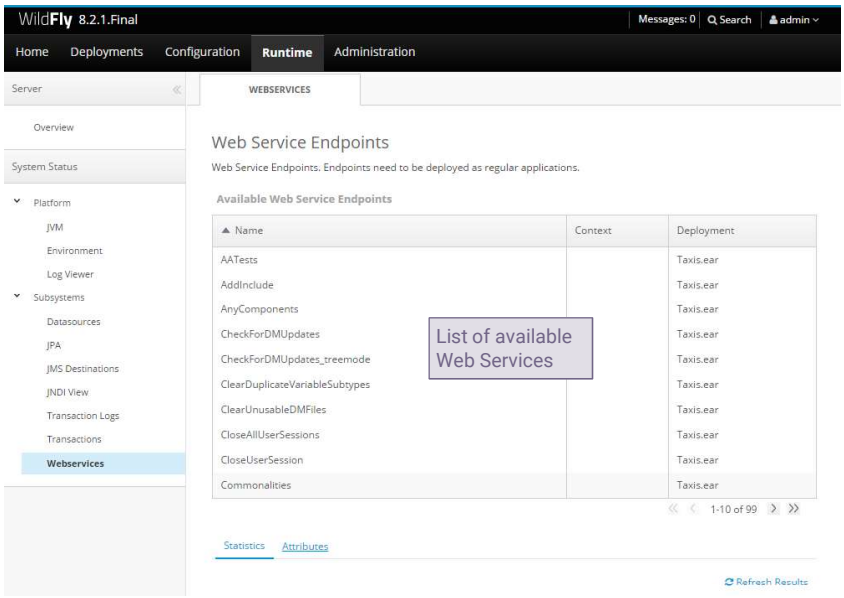


Web Services

All client applications communicate with the SPDRM Server through Web Services.

Apart from **ANSA**, **META** and **KOMVOS**, any 3rd party application can use the available Web Services in order to build an API to connect to SPDRM Server and access the SPDRM Database.

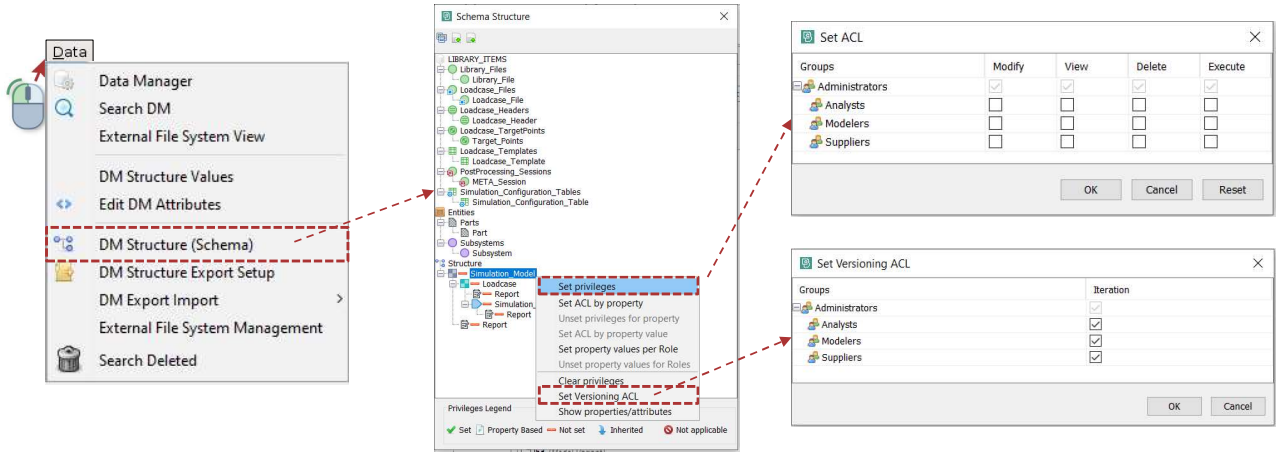
A list of the available Web Services is accessible through the web page of the SPDRM Server.
More information can be found in Administration chapter.



Data Security

In SPDRM, privileges on data are set with the aid of Access Control Lists (ACLs) which essentially control which user roles will be able to View, Modify, Delete or Execute particular data objects. For example, if the user's role doesn't have view privileges on a subsystem, it won't be possible to find it in the Search workspace. Similarly, without Modify privilege, it won't be possible to edit its file or modify its attributes and without Delete privileges it won't be possible to delete it.

Default access control rules are defined in the server configuration file (*taxis.conf*). When it comes to data security in particular, generic access control rules can be set on DM item types. These rules control the default privileges that will be assigned to newly created DM items according to their type. To set the permissions on DM item types, the user can invoke the *Schema Structure* window through **Data > DM Structure (Schema)** from the SPDRM client menu bar.



In the *Schema Structure* window, the user can right click on the desired type and select the option **Set privileges**. This will invoke the *Set ACL* window where permissions for each group/role can be defined using the respective checkboxes.

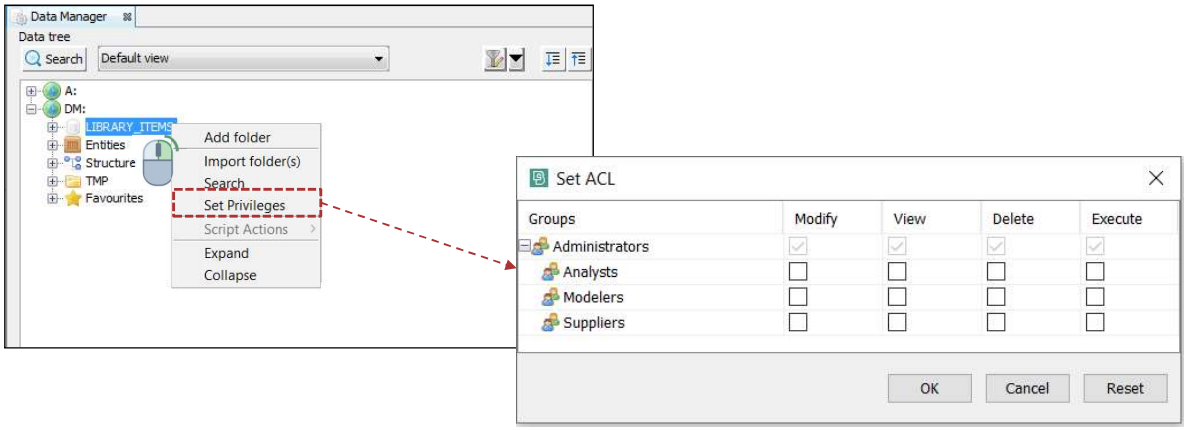
Similarly, using the option **Set Versioning ACL**, it is possible to define which user roles can spin-up particular versions of the selected DM object type.

Data Security

Access control can also be set on DM root items (i.e. *LIBRARY_ITEMS*, *Entities*, *Structure*) from the *Data Manager* window, to control who will be able to add, modify, view, or delete files, folders and/or DM items.

For example, if the user's role doesn't have view privileges on a folder, it won't be possible to view it in the DM Data tree. Similarly, without Modify privilege, it won't be possible its attributes. Without Delete privileges it won't be possible to delete it, while without Execute privileges it won't be possible to access its contents.

To define the privileges on DM root items, the user can select the option **Set Privileges** from the context menu in the Data Manager Tree and define the permissions for each group/role using the respective checkboxes in the *Set ACL* window that appears.



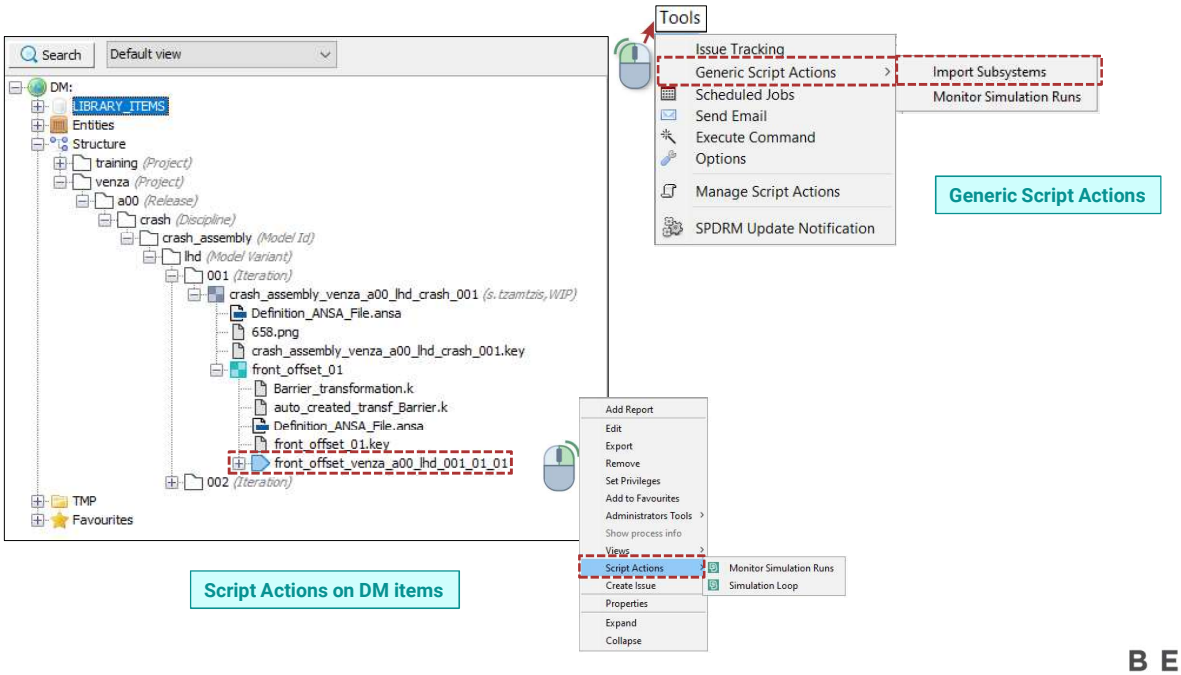
Custom Actions On Data

In SPDRM it is possible to perform customized actions on data, called *Script Actions*.

There are two types of Script Actions:

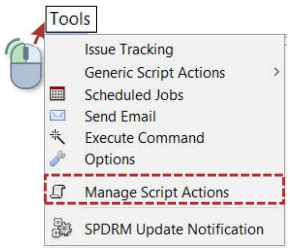
a) Generic Script Actions, which are launched from the SPDRM client menu bar using the option **Tools > Generic Script Actions**

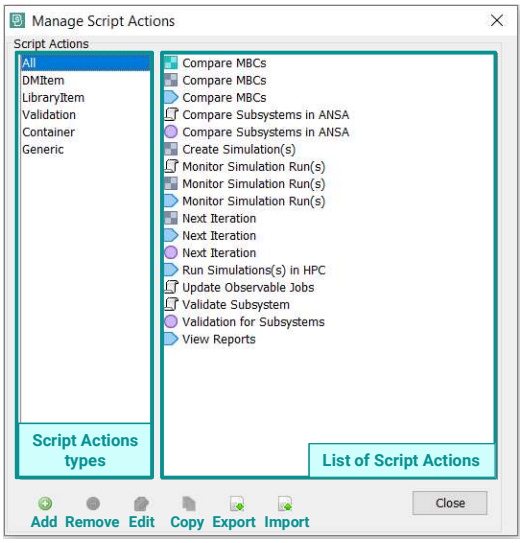
b) DM item-specific Script Actions, which are launched through the **Script Actions** context menu option of DM Items.



Custom Actions On Data

Management of the script actions is performed through the **Tools > Manage Script Actions** option from the SPDRM Client menu bar, which launches the *Manage Script Actions* window.





In the section on the left side of the *Manage Script Actions* window the available Script Action types are listed, while in the section on the right the available Script Actions of the selected type are displayed.

The available Script Action categories are:

All: Lists all available Script Actions, regardless of their type.

Container: Lists Script Actions defined on SPDRM Containers.

DMItem: Lists Script Actions defined on DM items (accessed from the context menu of each DM item).

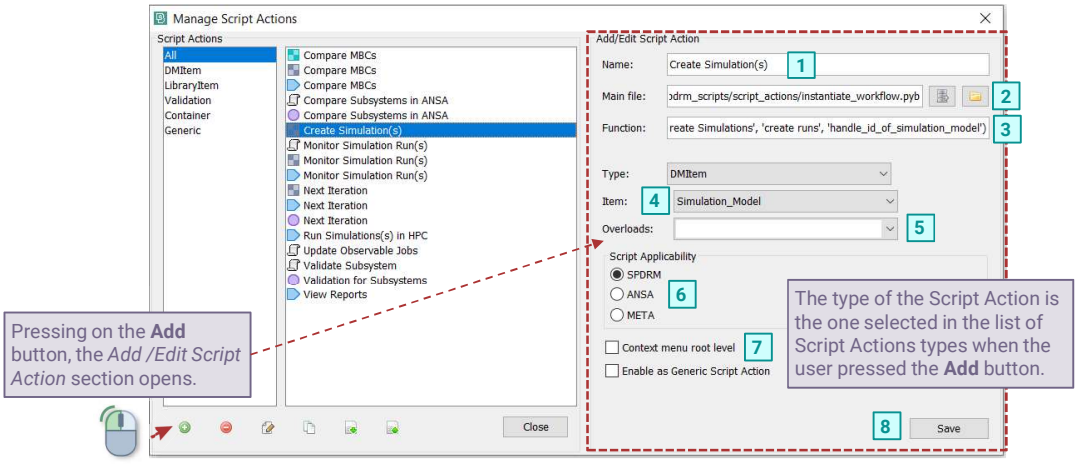
LibraryItem: Lists Script Actions defined on Library Items (accessed from the context menu of each Rich Library Item).

Generic: Lists generic Script Actions (accessed through Tools > Generic Script Actions).

Validation: Lists Script Actions used for the validation of actions on DM items or Rich Library Items.

Using the buttons at the bottom of the *Manage Script Actions* window the user can perform various actions for the management of Script Actions.

Custom Actions On Data



In order to **Add** ( ) a new Script Action, the following input is required in the *Add/Edit Script Action* section:

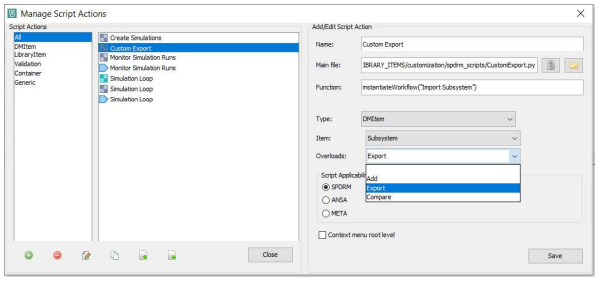
1. The **Name** of the Script Action, as it will appear in the Generic Script Actions list from the respective option in the SPDRM client menu bar or in the context menu of DM items.
2. The **Main file** of the script that will be used by the Script Action. The main file can be loaded either from the DM ( ) or from the local File System ( ) by pressing on the respective button.
3. The name of the **Function** in the main file that will be executed by the Script Action. For all Script Actions apart from the Validation Actions, the definition of the Function name requires the use of parentheses, while it is also possible to pass arguments to the function.
4. The Item (DM Item or Rich Library Item) or Container on which the Script Action will be applied, depending on the selected Script Action type being added.
5. A possible **Overload** of an existing context menu action (e.g. Export, Compare).
6. The BETA Application on which the script will be executed (SPDRM, ANSA or META).
7. The level of the action in DM Item's context menu. **Context menu root level** checked, sets the action in root level.
8. Pressing the **Save** button when all settings have been defined, the new Script Action is added.

Custom Actions On Data

The following options of the previous slide appear only when script action corresponds to type **DM Item** or **LibraryItem**:

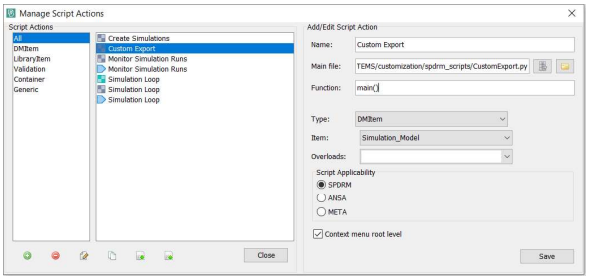
5 Overloads: The override of a built-in action of the context menu can take place by selecting a built-in action to be overloaded by the drop-down menu. The list of the available actions for overload pert type is the following:

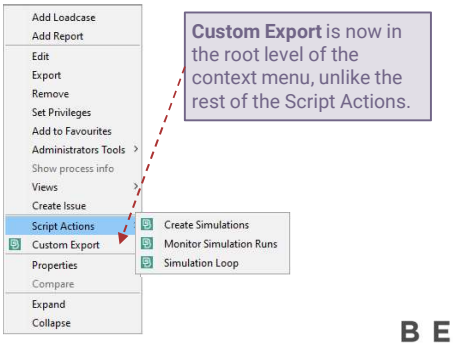
- **DMItem**
  - Export
  - Compare
- **LibraryItem**
  - Export
- **Entity** (Part, Subsystem)
  - Add
  - Compare
  - Export



6 Script Applicability: Defines the target application for the Script Action (SPDRM, ANSA, META). See sub-chapter **Script Actions directly in ANSA/META** in chapter **Process Design** for more information.

7 **Context menu root level:** The set-up of a script action in the root level of the context menu can take place by selecting the checkbox.



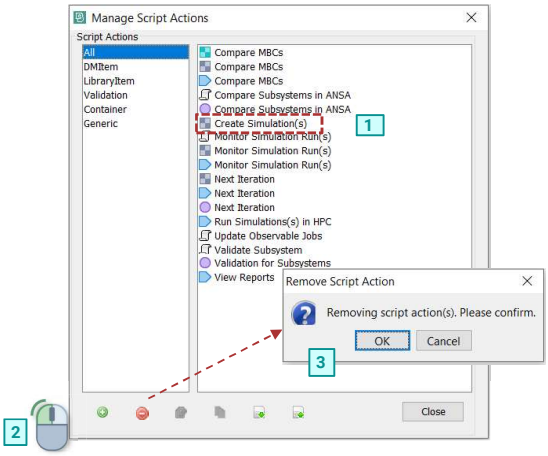


Custom Actions On Data

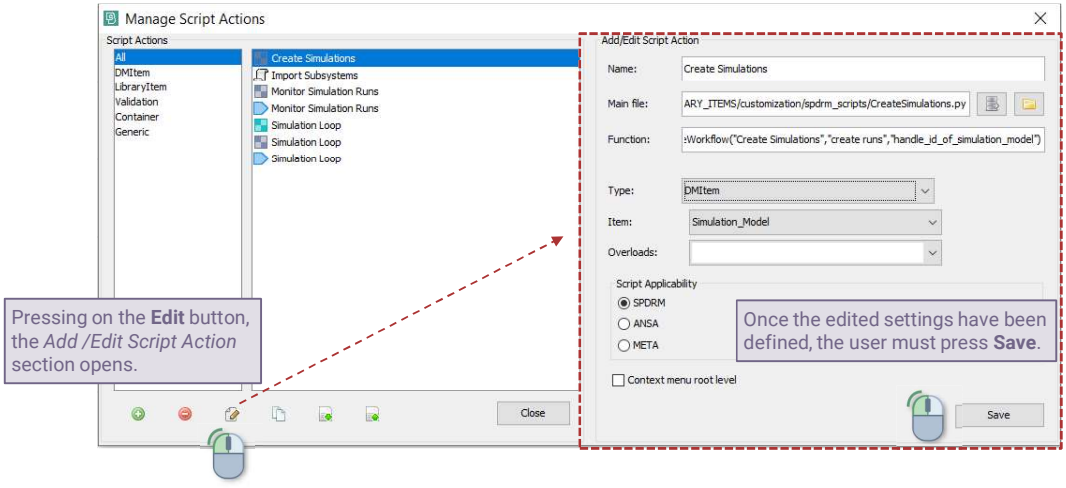
1 To remove one or more existing Script Actions, the user must first select them in the list of available Script Actions.

2 Pressing on the **Remove** (⛔) button, the *Remove Script Action* window opens.

3 The user must confirm or cancel the removal of the Script Action(s) in the *Remove Script Action* window.

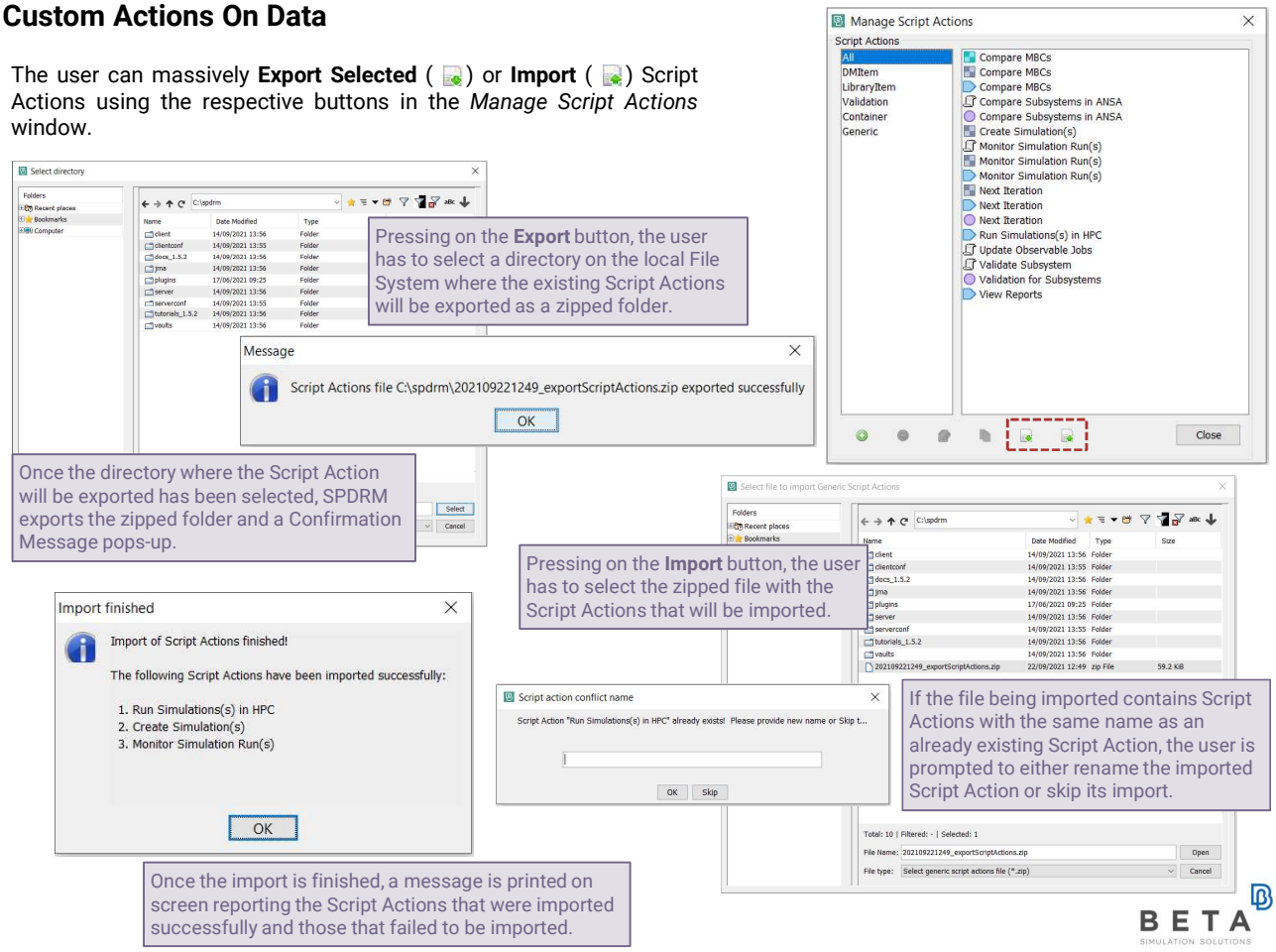


In order to **Edit** (✎) an existing Script Action, the user must define its new settings in the *Add/Edit Script Action.*



Custom Actions On Data

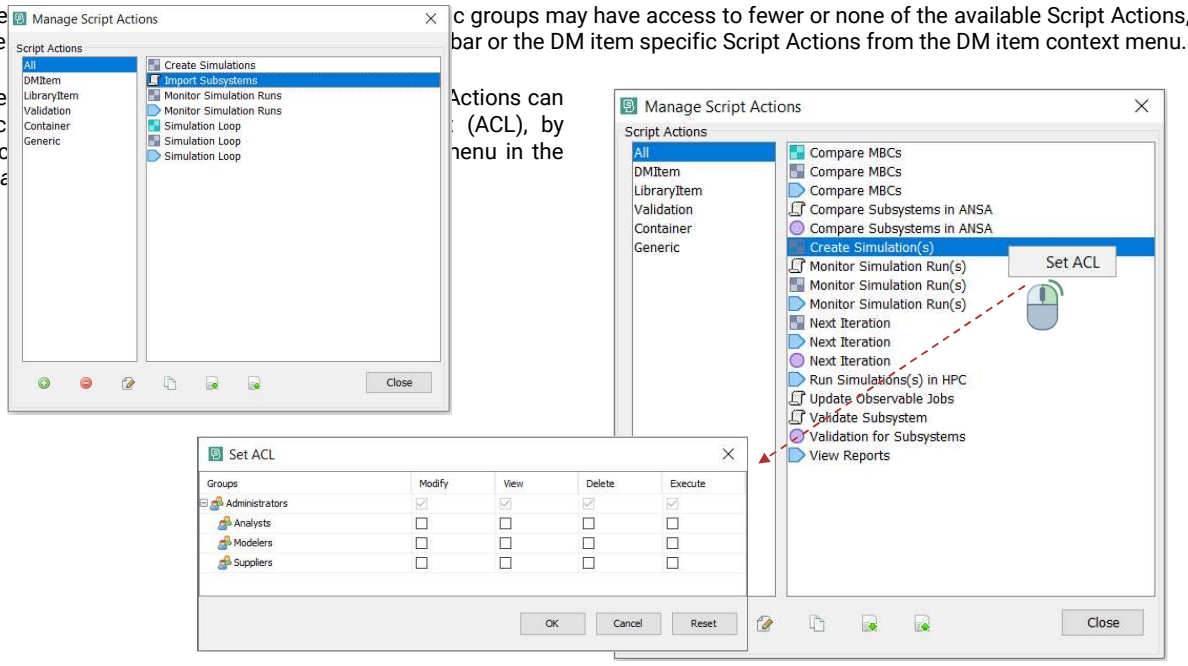
The user can massively **Export Selected** ( ) or **Import** ( ) Script Actions using the respective buttons in the *Manage Script Actions* window.



Custom Actions On Data

Depe c groups may have access to fewer or none of the available Script Actions, eithe bar or the DM item specific Script Actions from the DM item context menu.

Acce Actions can be c (ACL), by selec menu in the *Mana*



The *Set ACL* window appears where the user can define which actions (Modify, View, Delete and Execute) each user group can perform on specific Script Actions.

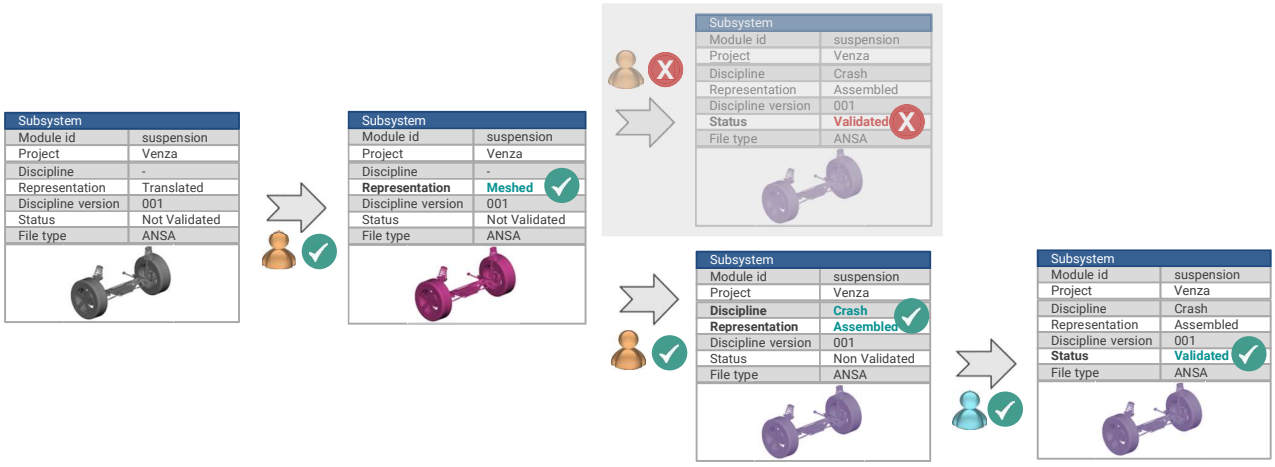
This feature is particularly useful for both system administrators who want to organize and manage the end user access rights to Script Actions, as well as end users who want to create and use private Script Actions.

Lifecycle Management

SPDRM supports the definition of a system of constraints that govern the evolution of a data object during its lifecycle. The lifecycle management scheme is enterprise specific and fully customizable.

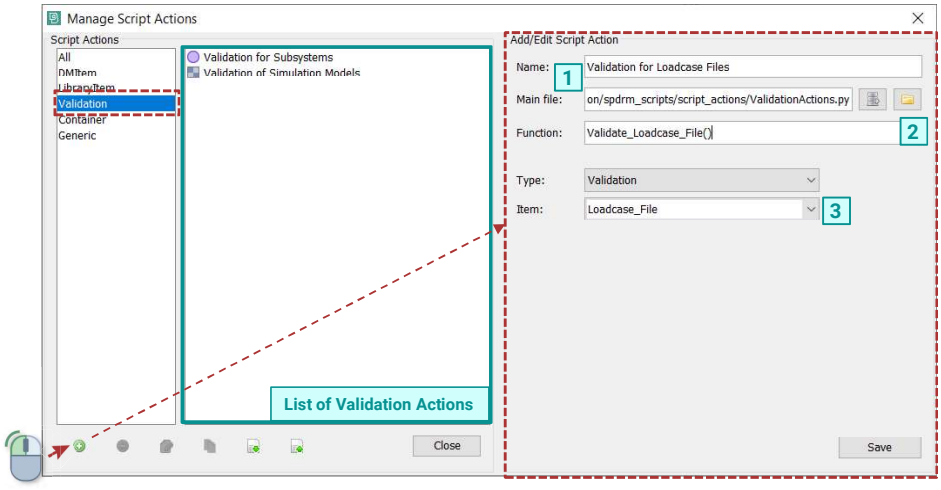
Through the lifecycle management scheme, SPDRM can control the possible states in the evolution of a data object based on:

- the data object's current status (e.g. control if the object can be overwritten or a new version will be created, based on its current status)
- the status of other data objects (e.g. control if a new object with a particular status can be created based on the status of its ancestors)
- the executing user (e.g. control which user roles have permissions to overwrite or spin-up the version of an existing object)



Validation Mechanism

To facilitate the lifecycle management, SPDRM employs a validation mechanism to ensure that any new object complies with the LifeCycle rules, automatically controlling the possible states in the evolution of a Data Object. This is enabled trough the definition of custom Validation Actions on the various Data Types.



1 The **Main file** is the custom Python script that will be used by the Validation Action. SPDRM currently supports the execution of Validation Actions only with scripts loaded from the DM ( ).

2 The user must specify the name of the **Function** in the main file that will be executed by the Script Action.
Contrary to other Script Actions, for the definition of the Function name for Validation Actions no parentheses should be used. In addition, no arguments can be passed to this function by the user.

3 The user must select the Item (DM Item or Rich Library Item) for which the Validation Action will be activated.

Validation Mechanism

Contrary to the other Script Action types, Validation Actions cannot be executed on demand by the users. When a Validation Action is defined for a specific Data Type, its execution is triggered automatically by SPDRM during the following operations:

- When a user tries to **Add** a DM object of this specific Data Type, using the respective option in the SPDRM Client.
- When a user tries to **Edit** a DM object of this specific Data Type, using the respective option in the SPDRM Client.
- When the user tries to update the a DM object of this specific Data Type, through the **Properties** card in the SPDRM Client.
- In particular for Rich Library Items, when the user tries to **Replace** the file of a DM object of this specific Data Type, or **Add New** version, using the respective options in the SPDRM Client.
- When any of the following SPDRM **script** functions are used in order to create a DM object of this specific Data Type or set its attributes/ additional attributes:
  - `createDMItem()`
  - `createLibraryItem()`
  - `setDMItemAttributes()`
  - `setDMItemAdditionalAttributes()`
  - `setEntityAdditionalAttributes()`
  - `setLibraryItemAttributes()`
  - `setFileAttributes()`
  - `setFileAdditionalAttributes()`

Validation Mechanism – Main script file

The Validation mechanism in SPDRM requires that the function of the main script file defined in the Validaton Action takes a specific argument and returns a specific object.

Validation function argument

When the validation action execution is triggered, SPDRM will pass to the validation script a dictionary as an input argument. This dictionary may contain the following keys:

Key	Key Type	Description
Mode	*String*	the mode of the validation script execution depending on the usage. Possible values: ***Add, Edit, SpinUp, Replace, Properties, Script***
Properties	*Dictionary*	the DM Properties of the DM object
Attributes	*Dictionary*	the DM Attributes of the DM object
Additional Attributes	*Dictionary*	the Additional Attributes of the DM object
DMObject Type	*String*	the Type of the DM object
DMObject Id	*Integer*	(only when SPDRM detects a conflict with an existing object) the Handle Id of the conflicting DM object
Parent Id	*Integer*	the Handle Id of the Parent container of the DM object in the Data Manager Tree
Edited Key	*String*	(only when the validation mode is *Edit)* the Property or Attribute that holds the file that is being edited

Therefore, the definition of the validation script function should have the format: *`functionName(argument)`*

```
def validateRLI(info):
    validation_mode = info['Mode']
    props = info['Properties']
    ...
```

Depending on the Validation Mode and the SPDRM conflict detection outcome, the "DMObject Id", "Parent Id" and "Edited Key" may not always exist as keys in the validation information PyDictionary.

Validation Mechanism – Main script file

Validation function return

The validation script should return to SPDRM a validation object that holds information related to the validation mechanism. This is a python object with the following attributes:

Attribute	Attribute Type	Description
is_valid	*Boolean*	takes the value ***True*** when then validation is successful or the vale ***False*** when the validation fails (*Default value: False*)
create_dm_item	*Boolean*	takes the value ***True*** when SPDRM should create a DM object or the value ***False*** when the validation script will handle the DM object creation (*Default value: True*)
invalid_props	*List*	a list of properties fields that are invalid and should be highlighted in the Add / Properties Card window (applicable when *is_valid=False*)
invalid_attrs	*List*	a list of attributes fields that are invalid and should be highlighted in the Add / Properties Card (applicable when *is_valid=False*)
invalid_addit_attrs	*List*	a list of additional attributes fields that are invalid and should be highlighted in the Add / Properties Card (applicable when *is_valid=False*)
error_message	*String*	a message that will be displayed in a pop-up window, instead of the standard SPDRM error message (applicable when *is_valid=False*)
conflict_resolution	*String*	the versioning policy (overwrite or spin-up) that will be applied for the validated object (applicable when *is_valid=True*). In case of spin-up, the spin-up property must be defined, i.e. Study Version.
modified_files	*Dict*	a dictionary that contains the paths of the files and/or folders that have been modified by the validation script (applicable when *is_valid=True*)
validatedObjectId	*Integer*	the handle Id of the object added to SPDRM by the validation script directly (applicable when *create_dm_item=False* and validation *Mode=Script*)

The class that defines the validation object is provided to the validation script by SPDRM and is exposed by a dedicated import in the script. Therefore the validation object must be instantiated in the validation script and at the end of the script execution, returned to SPDRM.

BETA SIMULATION SOLUTIONS

Validation Mechanism – Main script file example

```
# import all SPDRM script library
import SPDRM
# import the SPDRM ValidationResult class
from gr.betacae.clientexecutor.beans import ValidationResult

class LifecycleRules():
    conflict_resolution = {'Draft':'OverWrite', 'Validated':'Revision', 'Error':'Revision'}
    RLI_status_transitions = {'Draft':['Draft', 'Validated', 'Error'], 'Validated':['Validated', 'Error'], 'Error':['Draft']}

def validateRLI(info):
    # instantiate a validation object from the ValidationResult class
    res = ValidationResult

    # get validation information from the object passed as argument to the validation script
    validation_mode = info['Mode']
    props = info['Properties']
    attrs = info['Attributes']
    additional_attrs= info['Additional Attributes']
    try:
        conflicting_object_id = int(info['DMObject Id']
    except KeyError:
        conflicting_object_id = None

    if validation_mode == 'Properties':
        # the validation script will decide the conflict resolution based on the conflicting RLI Status
        conflicting_props_attrs_info = SPDRM.dm.getDMItemPropertiesAndAttributes(conflicting_object_id)
        conflicting_object_status = conflicting_props_attrs_info[2]['Status']
        res.conflict_resolution = LifecycleRules.conflict_resolution

        # the following determine if the Status transition from the conflicting value to the new value is allowed
        accepted_status_values = LifecycleRules.RLI_status_transitions[conflicting_object_status]
        if attrs['Status'] not in accepted_status_values:
            res.invalid_attrs.append('Status')
            res.error_message = 'Validator failed due to Status Transition Rule, the new value %s is not allowed!'% attrs['Status']

    # return the validation object back to SPDRM
    return res
```

BETA SIMULATION SOLUTIONS

Data relationships in SPDRM

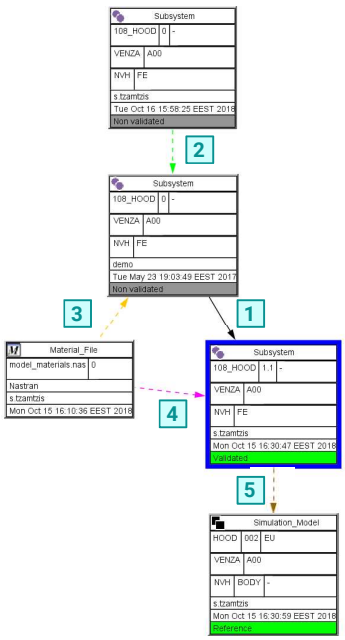
SPDRM offers complete tracking of the model history and associated data during model build. Links are used to capture the evolution of model data, in order to offer a global view of all relationships of a model and allow the identification of dependencies.

The following Links (relationships) are supported out-of-the box:

Link Type	Link Description	Examples
History Link	Used to connect two subsequent versions of the same object. This link is created automatically by SPDRM during the creation of a new version of an existing DM object	• A link between version 001 and version 002 of a Subsystem
Manual Link	Used to connect two different objects, where one has evolved from the other in its lifecycle or one has been used as input to produce the other. This link is created manually by the user, either from the SPDRM Client or via script, using either the SPDRM script function *dm.setDMItemReference* or the ANSA script function *base.ConnectDMObjectToDMObjects*.	• A link between the ANSA file type of a Subsystem and the solver keyword file • A link between a connections file library item and the common representation of a Subsystem
Content Link	Used to connect a DM object with its contents. This link is created automatically every time an object with contents is saved from ANSA. Additionally, content links can be created manually in SPDRM, using the script function *dm.setDMItemContents*	• The Subsystems used in a Simulation Model • The Library Items used in a Loadcase • The Simulation Model and Loadcase used in a Simulation Run

The SPDRM Lifecycle Graph

With the implementation of lifecycle management, SPDRM keeps track of the transitions and relationships of the data objects, enabling the creation of a pedigree tree. This pedigree is visualized in SPDRM in the Lifecycle Graph, displaying all links and dependencies for any DM object.



To maximize the traceability of the model data, the various data relationships can be visualized in the Lifecycle Graph, using links between the selected object and any other associated data objects:

1 **Lifecyle History Links** (——→)
They are used to connect previous and subsequent versions of the selected data object.

2 **Lifecycle Generic Links** (- - - -→)
They are used to connect different data objects of the same type (e.g. the ANSA and solver file of a subsystem).

3 **Input/Output Depencencies** (- - - -→)
They are used to connect data objects of different type that reference or are referenced by the selected data object (e.g. the connection file used to assemble a subsystem with the subsystem itself).

4 **Inherited Links** (- - - -→)
They are Manual links which are not created directly on the selected data object, but they are created on a previous version (History link) of the selected data object.

5 **Where Used Links** (- - - -→)
They are used to connect the selected object with the object(s) in which it is used.

6 **Content Links** (- - - -→)
They are used to connect the selected object with its contents.

It is possible to manually add references from the Lifecycle Graph of a DM object, using the respective context menu option that appears when the right mouse button is pressed anywhere on the white space of the graph. These will be Manual Links to objects that are referenced (used) by the selected object.

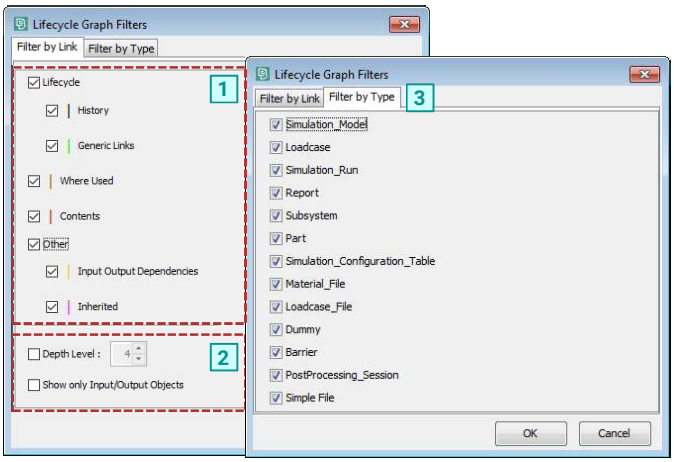
The SPDRM Lifecycle Graph

The visibility of the various data relationships in the SPDRM Lifecycle Graph can be controlled using built-in filters.

1 It is possible to filter the displayed relationships by link type, allowing the user to view the dependencies of the selected object in the desired context (e.g. view only the history evolution of an object, view only the contents of an object, etc.).

2 The displayed objects in the Lifecycle Graph can be limited by the depth level of connections, allowing the user to view direct relationships to the selected data objects or expand the pedigree tree to second level connections, third level connections, etc.
It is also possible to filter only the Input/Output relationships of the selected objects.

3 It is possible to filter the displayed relationships by object type, allowing the user to view specific data type dependencies of the selected object.



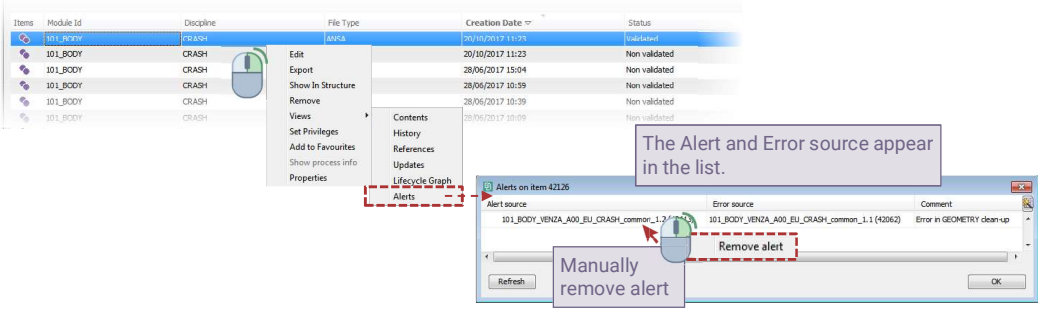
Alert Mechanism

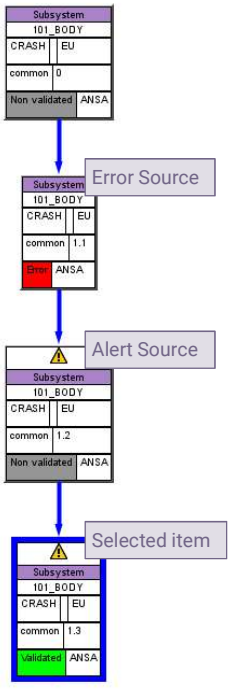
With the implementation of lifecycle management functionality, SPDRM offers the means to identify how a model was built and by whom. As part of this functionality an alert mehanism is offered, which, in case of errors, can raise alerts for the problematic objects and propagate them to all associated objects, offering the means to complete error impact analysis.

The generation of alerts is supported through two different mechanisms:

- By switching the value of an attribute to a pre-defined value (e.g. switching the Status of a dm object to Error). This will generate an alert for all the descendants of the object whose attribute was modified.
- By generating an *issue* for a particular object, using the Issue Tracking module. This built-in module enables the reporting and tracking of issues related to model and simulation data. SPDRM offers the possibility to generate an alert the moment an issue is created for a specific data object. This will generate an alert for the object for which the issue was created.

A list with all the alerts of the selected Data Object is available through the *Views>Alert* option of the context menu.





The alert mechanism can be customized per Data Type through the *dmAlerts.json* configuration file. It is possible to control the Attribute and Attribute Value which will trigger the alert for subsequent data objects, as well as the alert message. An example of the format of the file is given below:

```
{
"Subsystem":[{ "message_description": "${Comment}","attribute":"Status","alertValues": [ "Error"]}]
}
```

Issue Management overview

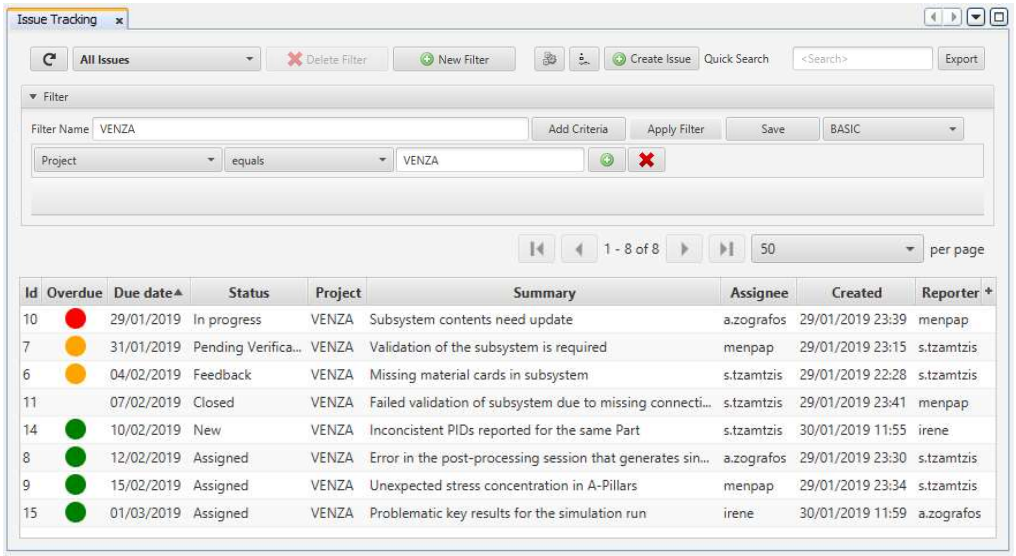
The SPDRM Issue Management console provides CAE engineers and analysts with a solution that offers structured procedures for the processing and management of quality issues that concern model and simulation data.

The main features of this solution are:

- It enables the report of issues concerning model or simulation data, directly in the SPDRM 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 will receive an email notification when action is required.
- It provides a pre-defined issue management workflow, ensuring that the correct steps for the handling of an issue are followed.
- 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

Issue Management workspace

SPDRM offers a dedicated workspace for the creation, display and searching/filtering of issues, called **Issue Tracking**. This workspace can be accessed through the option *Tools > Issue Tracking* from the SPDRM menu bar.



Id	Overdue	Due date	Status	Project	Summary	Assignee	Created	Reporter
10		29/01/2019	In progress	VENZA	Subsystem contents need update	a.zografos	29/01/2019 23:39	menpap
7		31/01/2019	Pending Verifica...	VENZA	Validation of the subsystem is required	menpap	29/01/2019 23:15	s.tzamtzis
6		04/02/2019	Feedback	VENZA	Missing material cards in subsystem	s.tzamtzis	29/01/2019 22:28	s.tzamtzis
11		07/02/2019	Closed	VENZA	Failed validation of subsystem due to missing connecti...	s.tzamtzis	29/01/2019 23:41	menpap
14		10/02/2019	New	VENZA	Inconcistent PIDs reported for the same Part	s.tzamtzis	30/01/2019 11:55	irene
8		12/02/2019	Assigned	VENZA	Error in the post-processing session that generates sin...	a.zografos	29/01/2019 23:30	s.tzamtzis
9		15/02/2019	Assigned	VENZA	Unexpected stress concentration in A-Pillars	menpap	29/01/2019 23:34	s.tzamtzis
15		01/03/2019	Assigned	VENZA	Problematic key results for the simulation run	irene	30/01/2019 11:59	a.zografos

© Copyright 2017 BETA CAE Systems All rights reserved

Issue Edit Card

Pressing the **Create Issue** button in the *Issue Tracking* window, a form (the issue *Edit Card*) opens that consists of several mandatory and optional properties which need to be filled in for the creation of an issue. Mandatory properties are marked in bold in the form and the issue cannot be created unless these are defined.

The ***Issue DM Item*** is a key property, which associates an issue with a specific DM item and can trigger the inheritance of certain properties (i.e. the ***Project*** and the ***Variant***) to the issue. Only one DM item can be defined as issue DM item.

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 ***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 from the SPDRM Issue Management console to inform the respective user for the assignment of an issue and provide issue-related information.

The ***Root Cause*** is a field that can be filled when the source of the error is known, either on issue creation or after the analysis has identified the error source. It is possible to define more than one DM items as the root cause of an issue.

The ***References*** table lists the DM items that are associated with each issue. These include the Issue DM Item and the Root Cause object(s), as well as any other DM items that are attached to an issue as supplementary objects. The *Reference Type* column of the table provides information on the relationship of each attached DM item with the issue.

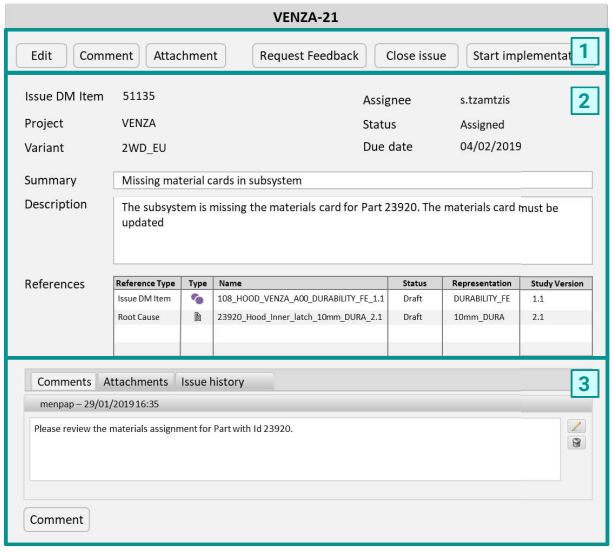
The ***Attachments*** field may be used to attach to any additional files that may be relevant to the issue (e.g. an image, a checklist, etc.)

The fields that appear in the issue Edit Card can be customized using the issue management configuration file.

Issue View Card

Once an issue is created, it can be viewed through the *Issue Tracking* window, by double-clicking on the issue or using the option **Open in new tab** from its context menu. This will open the issue *View Card* in a new tab in the *Issue Tracking* workspace.

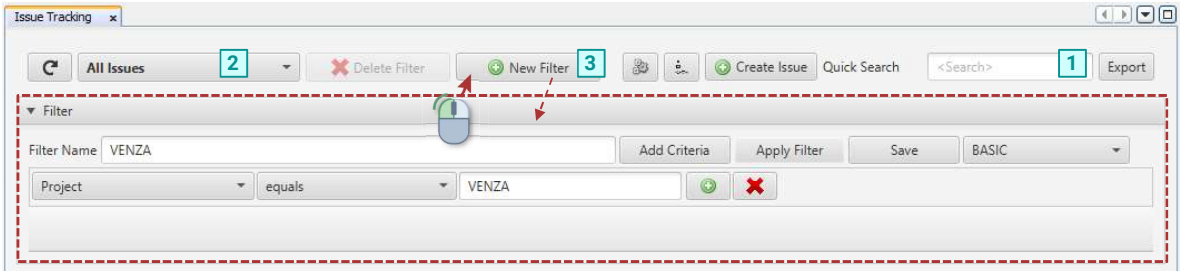
1. At the top of the issue View Card, action buttons are available to control the issue workflow (e.g. Start Implementation, Close Issue, etc.) or perform actions related to the issue management workflow (e.g. Edit the issue, Request Feedback, etc.),
2. The main body of the issue View Card displays the issue properties, which are read-only.
3. At the bottom section of the issue View Card, the user can view any comments added on the issue, the existing issue attachments and the issue history (a history of all actions on the issue and the user that performed each action).



The action buttons that appear in the issue View Card can be customized using the issue management configuration file.

Searching for issues

All created issues are displayed in the *Issue Tracking* window, where the user can search for issues using quick search keywords or filters.

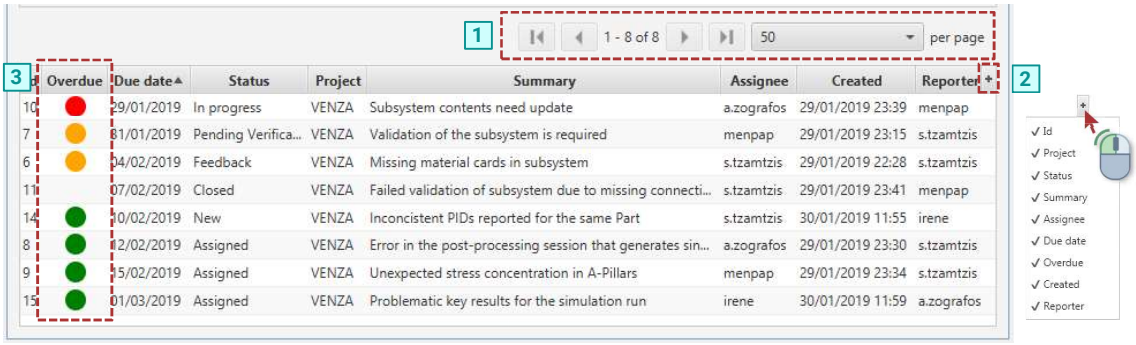


1 Using the **Quick Search** field, it is possible to search for issues that contain specific keywords in their *Summary* or in their *Description*.

2 Pre-defined filters are available out-of-the-box, controlling which issues will be displayed in the list: all issues, only issues reported by the current user or only issues assigned to the current user.

3 Pressing the **New Filter** button, it is possible to create user-defined filters using queries based on issue properties (e.g. only display the issues for a specific Project). Using the **Add Criteria** button, queries can be constructed that consist of one or more criteria and the filter can be applied using the **Apply filter** button. User-defined filters can be saved in order to be re-used as required, by specifying a *Filter name* and pressing the **Save** button.

Searching for issues



1 In the list of displayed issues, the user can easily identify the total number of issues, control the number of issues displayed per page and navigate to other pages.

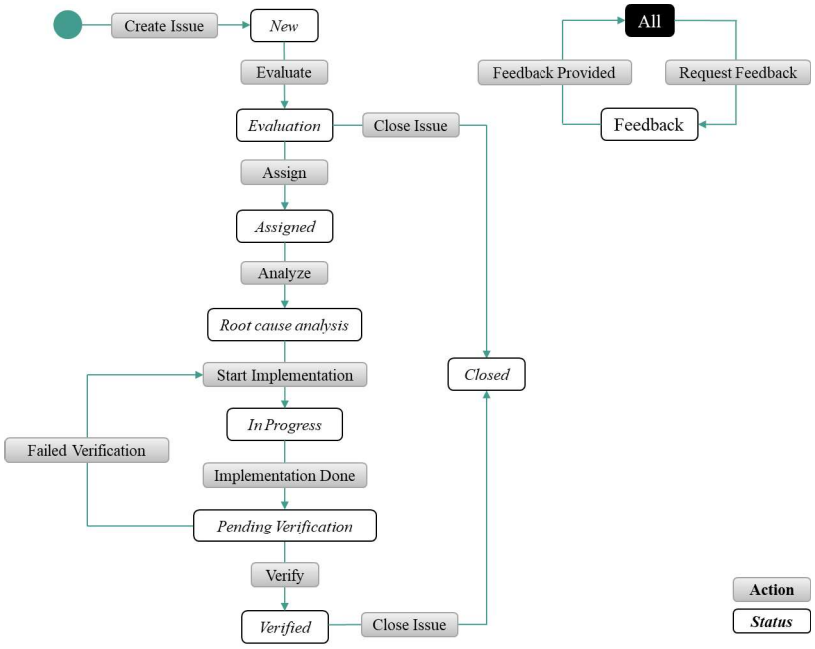
2 The properties that will be displayed as columns in the list of displayed issues can be configured. It is also possible to sort the issues per property value.

3 A quick visual indication of the time remaining until the issue due date is given in the *Overdue* column, using specific color coding to draw the user's attention as required.

With its built-in and user defined filters that control the display of issues in the Issue Management console, as well as the sorting capabilities offered within the list of displayed issues, SPDRM enables the efficient management of data quality issues.

Issue Management workflow

To accurately capture the issue management workflow, the definition of transitions between the different issue status values is required. These transitions are controlled through a set of user actions that drive the advancement of the issue to one of the alternative accepted status values. The default SPDRM issue management workflow offered by SPDRM can be seen below.

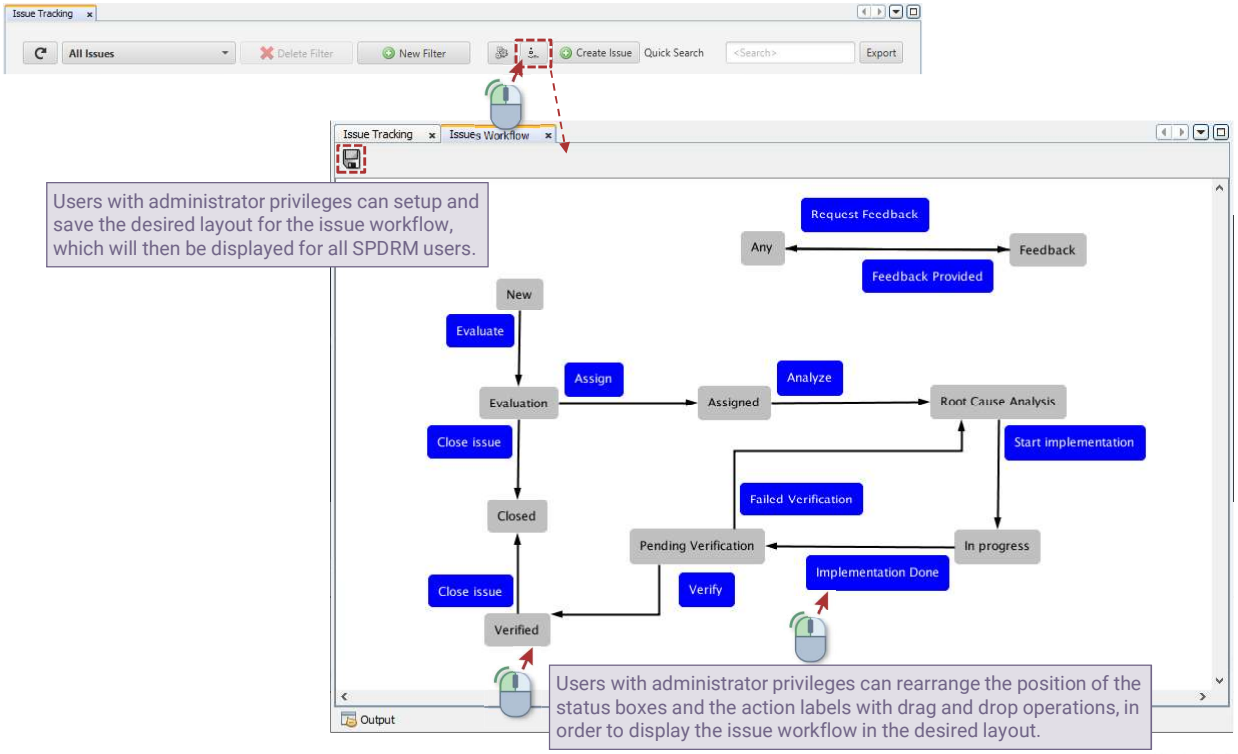


The definitions of the issue status values, the user actions that control each status transition, as well as the accepted transitions, are given in the issue management configuration file.

Issue Management workflow

The issue management workflow can be viewed directly from the SPDRM Client, using the **View Issues Workflow** button .

A new tab opens, called *Issue Workflow*, where the Status values, workflow Actions and status transitions are displayed.



Issue Management configuration

Issues in SPDRM are pre-configured with a set of properties, transitions and actions; however their definition can be customized to adapt to the needs of the various engineering teams. Defining additional/alternative status values, user actions and transitions, the built-in workflow can be enhanced or a fully customized issue workflow can be defined.

Through the Issue Management configuration file called *issues.dat*, it is possible to customize several of the issue characteristics, such as:

- The issue properties, as well as the attributes of each property (e.g. the property display name, the accepted values, etc.)

```
properties = project, dueDate, issueDmItem, summary, description, release, priority, type, rootCause, assignee, status
project.displayName = “Project”
project.acceptedValues = PROJ1, PROJ2, PROJ3, PROJ4, PROJ5, PROJ6, PROJ7, PROJ8
project.defaultValue = PROJ1
```

- The definition of properties as mandatory or optional for the creation of an issue, as well as if the value of each issue property can be edited or not after its creation.

```
project.mandatory = true
project.editable = false
```

- The issue states, the workflow transitions and the user actions that drive each transition.

```
state.new.displayName = “New”
state.assigned.displayName = “Assigned”
action.assign.displayName = “Assign”
action.assign.from = new
action.assign.to = assigned
```


Issue Management configuration

- The issue default and end states and notification actions triggered in state transition:

```
default.state = state
end.states = closed
emailOnStateTransition = true
```

- The necessity for commenting on state change:

```
assignComment.mandatory = true
provideFeedback.mandatory = false
```

- Custom script actions that can be triggered by an issue transition:

```
action.assign.scriptAction = myAction1
scriptAction.myAction1.mainFile = DM:/LIBRARY_ITEMS/scripts/issue_mgmt.py
scriptAction.myAction1.function = doSomething
```

- The date format:

```
date.format = dd/MM/yyyy
```

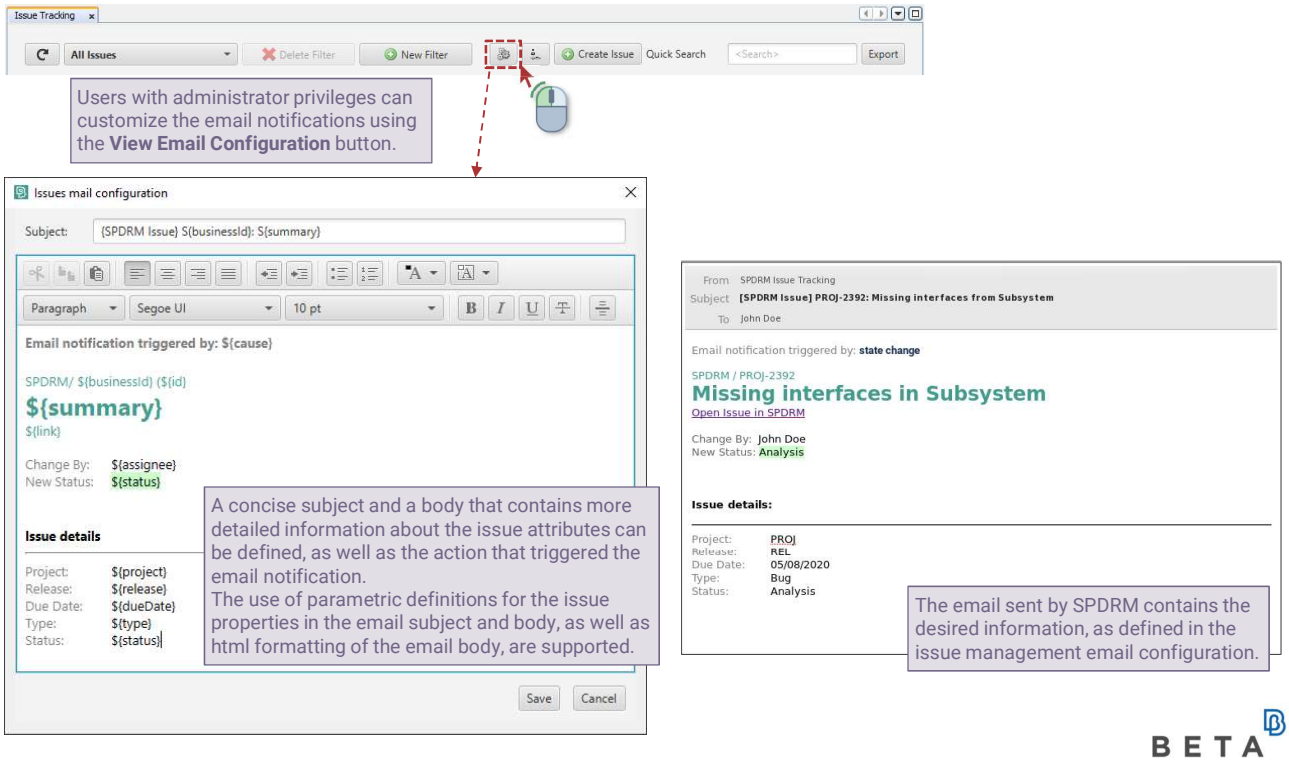
- The columns that will be displayed in the references table of each issue:

```
dm.columns = Id,Name,Status
```

The configuration settings for the Issue Management functionality are included in the ***issues.dat*** file that is located in: `[SPDRM_SERVER_DIR]/wildfly/standalone/configuration/spdrm/.`

Issue Management email configuration

The SPDRM Issue Management comes with an embedded notification mechanism, ensuring that any user involved with an issue will receive an email when action is required. This email notification content and layout can be customized in order to convey all necessary information in the right context.



Deletion Mechanism overview

SPDRM employs a deletion mechanism to facilitate the cleanup of the database from old and temporary data, with features to prevent accidental data loss and enable data restoration. The deletion of each object is governed by a set of rules based on its existing dependencies.

Data that the user attempts to delete are classified by SPDRM in two categories:
- Objects with dependencies (i.e. history links), which will be partially deleted
- Objects without references, which will be permanently deleted

With the SPDRM deletion mechanism, all data that are deleted by the users in the SPDRM client end up in the same pool, which can be browsed through the *Search Deleted* list. The Search Deleted list is a tool accessible only by administrator users, with which::
- Deleted data can be identified
- Deleted data can be restored
- Deleted data can be manually purged

In addition, a history of *Overwrites* is maintained for the files/folders of DM items, RLIs and simple library files. The user is able to identify overwritten files and restore them either through the Overwrites tab or the Search Deleted list.

Another key feature of this mechanism is that it enables the deletion of attached files/folders, while preserving the associated SPDRM objects and their metadata. It is possible to permanently delete such attached files/folders or archive them before their deletion, in order to enable their retrieval if required.

The advantages offered by the SPDRM deletion mechanism include:
- Prevention of accidental deletion of data
- Efficient restoration of deleted data
- Automatic or on-demand cleanup of the disk

Deletion Rules

The rules for the deletion of objects in SPDRM can be summarized as follows:

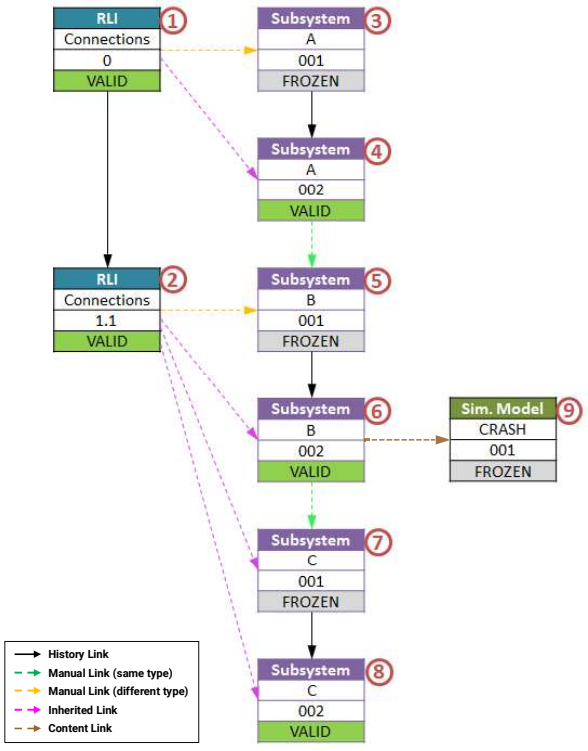
1. **No deletion**
   Objects cannot be deleted from the system when any of the following conditions apply:
   - The selected object has a **manual link** with one or more descendants
     (*e.g. a Subsystem with ANSA file type and a Subsystem with solver file type*)
   - The selected object is **content** of one or more descendants
     (*e.g. a Subsystem and a Simulation Model*)

2. **Partial deletion**
   Objects are partially deleted from the system when they are linked with their descendants by **history link** only
   (*e.g. version 001 and version 002 of a Subsystem*).
   Partially deleted objects have the following characteristics:
   - They can be browsed using the Search Deleted list
   - They are displayed(as metadata) in the Lifecycle Graph and the References lists, but not in the Search DM lists.
   - Conflicts with partially deleted objects are detected by the system

3. **Permanent deletion**
   Objects are permanently deleted from the system when any of the following conditions apply:
   - The selected object has **no descendants**
   - The selected object has **only deleted descendants**

   Permanently deleted objects have the following characteristics:
   - They end up in the Search Deleted list, with the exception of objects without a representation file, which are instantly purged from the system
   - No conflicts are detected with permanently deleted objects

Deletion examples



1. No deletion

Content of descendants

- Object 6 (Subsystem) cannot be deleted because it is content of object 9 (Simulation Model)

Descendants with reference link

- Object 4 cannot be deleted due to descendants with reference link (i.e. object 5)

2. Partial Deletion

Descendants with history link only

- Objects 3, 5 and 7 can be partially deleted (their metadata will be kept) because they have only history links to descendants

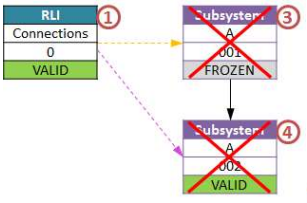
3. Permanent Deletion

No descendants

- Object 8 can be deleted permanently because it does not have any descendants
- Object 9 can be deleted permanently because it does not have any descendants

Deleted descendants

- Assuming that no other relationships existed, Object 1 can be deleted permanently, if objects 3 and 4 were deleted first



Permanently deleted objects and the Search Deleted list

Objects without any references can be deleted permanently in SPDRM. To prevent accidental loss of data, it is now possible to add a configurable time frame to delay the deletion of the data from the system.

This is controlled through the *taxis.conf* using the *delete_time_frame_days* key:

```
<entry key="delete_time_frame_days">number_of_days</entry>
```

This way, deleted data will always end up in the *Search Deleted* list, from where the system will automatically remove them after a specified number of days (a "cleaning" mechanism is executed nightly, removing objects with a deletion due date tag equal to the mechanism execution date).

(e.g. for data to be permanently removed from the system 2 days after their deletion by a user, the following entry should be added to the taxis.conf: *<entry key="delete_time_frame_days">2</entry>*)

If the **delete_time_frame_days** key is missing from the *taxis.conf*, by default SPDRM will automatically delete the data after 30 days.

If a negative value is given for the **delete_time_frame_days** key (i.e. -1), objects that can be permanently deleted will be directly removed from the system, without ending up in the *Search Deleted* list.

For objects that are permanently deleted, SPDRM by default removes their database entries, but not the actual files from the file system (firstDir). This behavior can also be controlled through the *taxis.conf* using the *enable_permanent_delete_of_files* key:

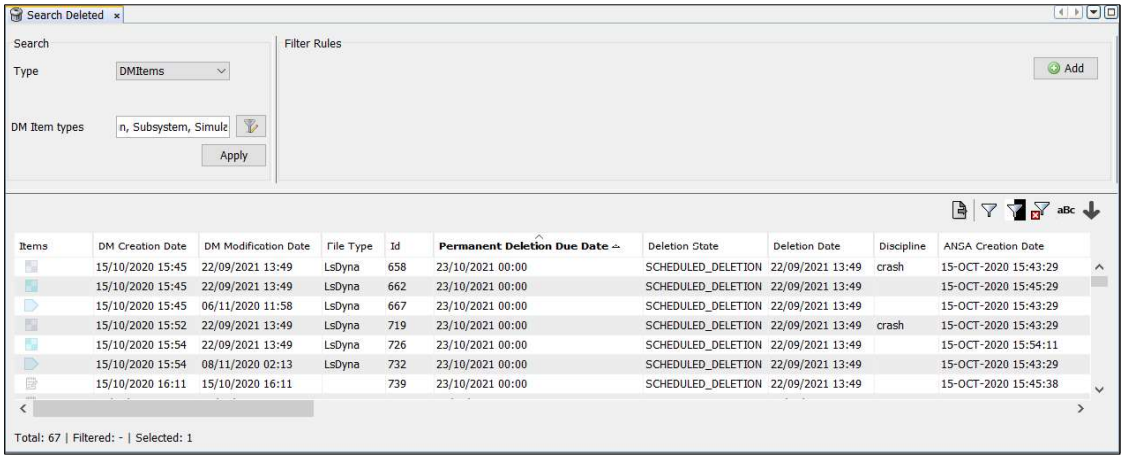
```
<entry key="enable_permanent_delete_of_files">boolean_value</entry>
```

This way, depending on the value (true/false) of the *enable_permanent_delete_of_files* key, deleted data can also be deleted from the firstDir.

If the **enable_permanent_delete_of_files** key is missing from the *taxis.conf*, by default SPDRM will **not delete** the data from the firstDir.

The Search Deleted list

The **Search Deleted** list is a tool that was developed to offer functionality related to the management of deleted objects and the data storage. It is a dedicated tool, accessible only by system administrators. It is accessed use the option **Data > Search Deleted** from the SPDRM menu bar.



Key features of the Search Deleted tool are:

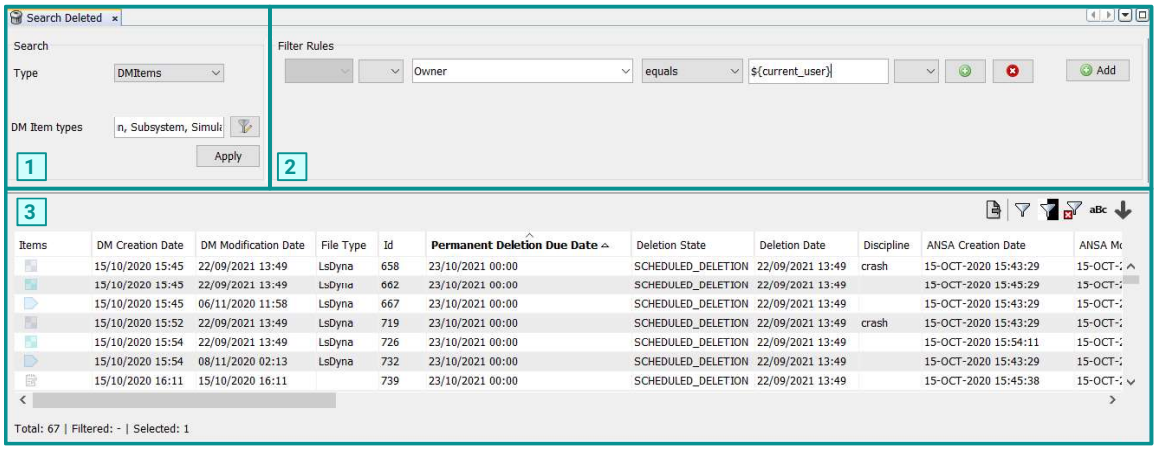
- It enables the restoration of deleted objects [1]
- It enables the manual purging of deleted objects [2]
- It offers dedicated views for the References, Contents and Lifecycle Graph of deleted objects
- It offers information regarding the deletion of each object (i.e. Deletion User, Deletion Date, etc.)

[1] Conditions apply (i.e. it would not be possible to restore a Simulation Model if any of its contents have been deleted)

[2] Conditions apply (i.e. it would not be possible to manually purge a partially deleted object, if any of its descendants are still "live")

The Search Deleted list

The *Search Deleted* list has similar search functionality as the *Search DM* window, described in the Data Search section.



1 In the *Search* section, the user can define the search settings for the execution of filter queries and **Apply** the filters

2 In the *Filter Rules* section, the user can define the filters for the query execution and **Add** new filters

3 In the *Results* list, the user can view the results of the applied filter and search within the displayed results

The Search Deleted list

The ***Deletion State*** and the ***Permanent Deletion Due Date*** columns offer information regarding the deleted data and their scheduled removal from the system.

Items	Owner	Size	Creation Date	Version	Id	Deletion State	Permanent Deletion Due Date
	demo	3.3 kB	04/07/2018 16:11	1	51281	SCHEDULED_DELETION	02/09/2018 11:12
	demo	0 B	04/07/2018 16:12	1	51283	OVERWRITTEN	
	s.tzamtzis	1.9 MB	09/03/2018 13:24	1	46713	SCHEDULED_DELETION	02/09/2018 11:12
	s.tzamtzis	26.2 kB	09/03/2018 14:34	1	46761	SCHEDULED_DELETION	02/09/2018 11:12
	s.tzamtzis	77.3 kB	19/09/2017 13:33	1	4369	SCHEDULED_DELETION	02/09/2018 11:12
	demo	44.1 kB	14/06/2018 15:17	1	50763	OVERWRITTEN	
	demo	206.0 kB	14/06/2018 15:17	1	50762	OVERWRITTEN	

The Deletion State column has three possible values:

- ***OVERWRITTEN***, for objects that exist as overwrites of existing objects. These objects can be purged manually by an administrator at any time.
- ***HISTORY_LINK***, for objects that have history links with other objects. These objects can be purged manually by an administrator only if all their descendants are deleted.
- ***SCHEDULED_DELETION***, for objects that will be automatically purged by the system on a scheduled date. These objects can be purged manually by an administrator at any time.

The Permanent Deletion Due Date column displays the exact date when the scheduled purging of an object will automatically be performed by SPDRM.

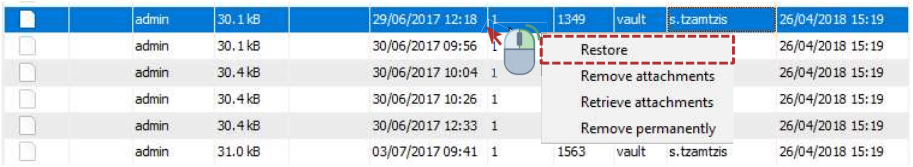
© Copyright 2017 BETA CAE Systems All rights reserved

The Search Deleted list

The ***Deletion Date*** and the ***Deletion User*** columns can assist the administrator in identifying deleted data.

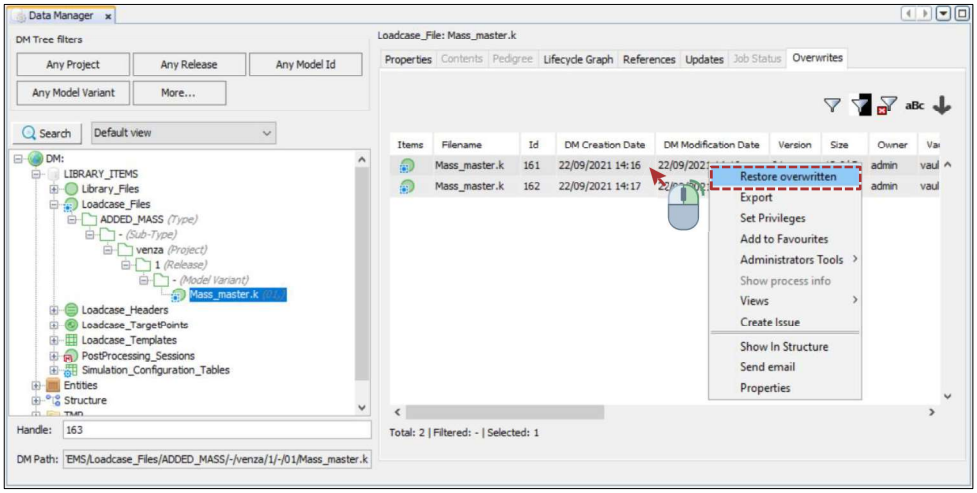
Items	Status	Owner	Size	DM Path	Creation Date	Version	Id	Vault	Deletion User	Deletion Date
		demo	123.6 kB		20/07/2017 18:27	1	3715	vault	demo	14/06/2018 15:17
		demo	124.2 kB		20/07/2017 23:15	1	3838	vault	demo	14/06/2018 15:17
		demo	124.2 kB		21/07/2017 16:56	1	3963	vault	demo	14/06/2018 15:17
		demo	133.4 kB		28/08/2017 12:07	1	4116	vault	demo	14/06/2018 15:17
		demo	134.6 kB		26/09/2017 10:43	1	7942	vault	demo	14/06/2018 15:17
		demo	30.1 kB		29/06/2017 12:18	1	1349	vault	demo	26/04/2018 15:19
		demo	30.1 kB		30/06/2017 09:56	1	1361	vault	demo	26/04/2018 15:19
		demo	30.4 kB		30/06/2017 10:04	1	1362	vault	demo	26/04/2018 15:19
		demo	30.4 kB		30/06/2017 10:26	1	1372	vault	demo	26/04/2018 15:19
		demo	30.4 kB		30/06/2017 12:33	1	1521	vault	demo	26/04/2018 15:19

If the data was accidentally deleted, the administrator can proceed to the restoration knowing who the deletion user was and when the data was deleted



Overwrites history

SPDRM keeps a history of **Overwrites** for the files/folders of DM items, RLIs and simple library files. When an attached file/folder of an object is replaced and the object is overwritten, the replaced file will appear in the Overwrites tab of the new file/folder, as well as the Search Deleted list.

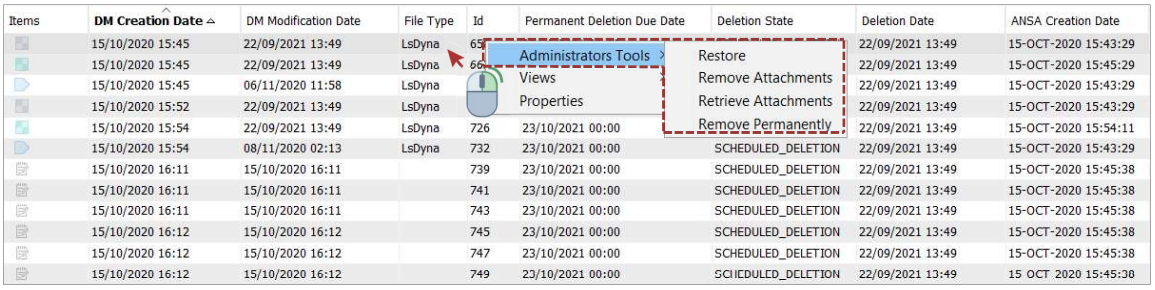


The user is able to restore overwritten files/folders either from the Overwrites tab or the Search Deleted list. This will automatically swap the existing and the restored file/folder in the Overwrites tab and in the Search Deleted list, provided that the DM object has no dependencies .

Overwrites are not automatically deleted by the system, however they can explicitly be permanently removed by the administrator from the Search Deleted list. Once an overwritten file/folder is permanently removed, it is also cleared from the history of Overwrites of existing files/folders.

Removal of attached files/folders

The SPDRM deletion mechanism enables the removal, archival and retrieval of attached files /folders for all DM objects. The defintions of the DM objects and their metadata will remain in the system, while their attachments will be removed to facilitate storage space management. This functionality is available through the respective context menu option, accessible only by system administrators.



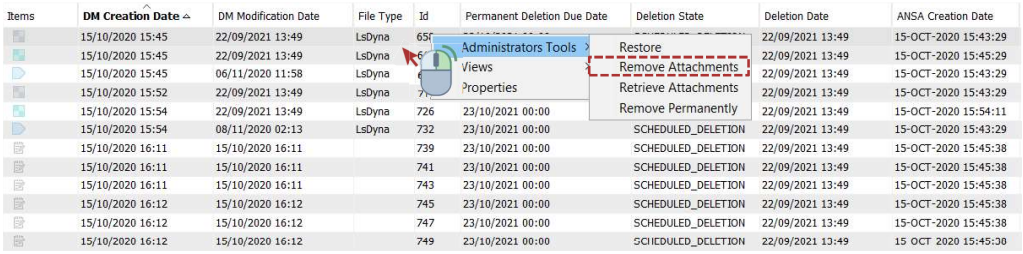
When attachments are being removed, it is possible for SPDRM to archive them in a pre-defined location on the file system, in order to enable their retrieval. This is controlled through the *taxis.conf* using a dedicated key:

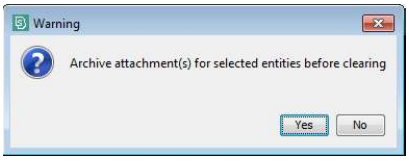
```
<entry key="<vaultName>ArchiveDeleted">/....filepath/</entry>
```

Once an archival path is defined, SPDRM will support the archival of attachments being removed, if requested by the administrator. A *.zip file named after the handle Id of the object whose attachments are being removed, containing all its attachments, will be saved in the archival directory. This can be retrieved on demand, provided that the *.zip file has not been modified after its creation.

SPDRM requires the definition of a separate archival path per vault. Therefore, in order to enable the archival functionality, the entry key should be written per vault in the *taxis.conf*. (e.g. to enable the archival functionality for objects stored in a vault named "*MyVault*" to an archive directory in the filepath such as "*/home/demo/SPDRM/archiveVault/*", the following entry should be added: `<entry key="MyVaultArchiveDeleted">/home/demo/SPDRM/archiveVault/</entry>`)

Removal of attached files/folders

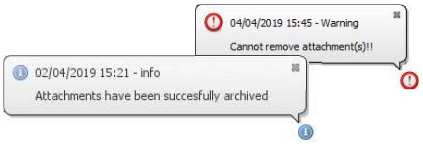




1 Selecting the option **Remove attachments** for a DM object, an SPDRM warning window will appear asking the user if the attachments should be archived before the removal.

If the archival functionality is not enabled through the *taxis.conf*, this warning window will not appear

2 Before removing the attachments of a DM object, an SPDRM window will appear to inform the user that attachments which have not been archived cannot be retrieved if required; and ask for confirmation that the attachments of the selected object(s) should be removed.

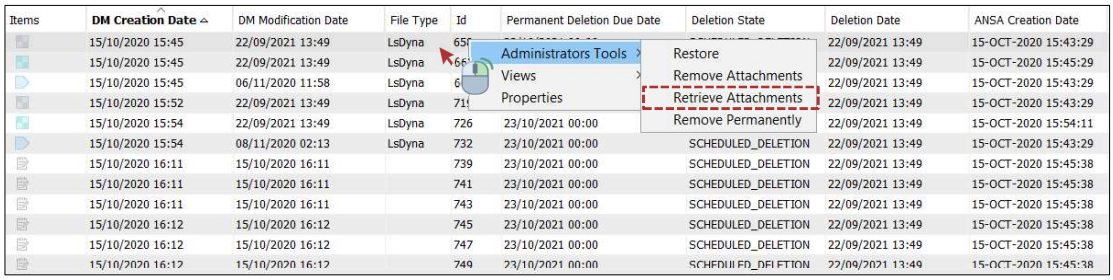


3 A notification balloon will appear on the notification displayer section of the SPDRM client, informing the user for the success / failure of the removal of the attachments.

It is possible to define custom icons for objects whose attachments have been cleared, in order to facilitate their visual identification in the SPDRM Client.

Removal of attached files/folders

Selecting the option **Retrieve attachments** for a DM object whose attachments have been removed, SPDRM will look in the archive directory for a .zip file named after the handle Id of the selected object and will try to restore the attachments.



A notification balloon will appear on the client, informing the user for the success / failure of the retrieval of the attachments.

SPDRM will fail to retrieve the attachments of a DM object if the .zip file containing them does not exist, or if it has been modified since its creation.

The functionality for the removal/retrieval of attached files/folders is also available via script, using the following SPDRM script functions:

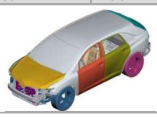
- `dm.clearAttachments(itemid, archive)`
- `dm.retrieveAttachments(handle_ids)`
- `dm.deleteArchivedAttachments(handle_ids)`

Please refer to the SPDRM scripting guide for a detailed description of the script functions.

Data Model Format Description

The Data Model is defined in the **DM_structure_TBM.xml** file. Each data type is defined as an element within the top level **XMLData** element. In this definition, the attributes of each data type are specified. The primary attributes, which constitute the identity of an object, are called **Properties** and are necessary for the definition of a data type:

Simulation Model	
Model id	full vehicle
Project	Venza
Release	rel01



```
<Simulation_Model>
      <Properties>
                  <Property name="Model id"  read_only="NO" type="TEXT" position="1"/>
                  <Property name="Project"   read_only="NO" type="TEXT" position="2"/>
                  <Property name="Release"   read_only="NO" type="TEXT" position="3"/>
      </Properties>
</Simulation_Model>
```

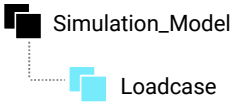
Apart from **Properties**, all the other elements in the **DM_structure_TBM.xml** file are optional.

The **Attributes** element is used to specify a list with all the secondary attributes of the data type. These attributes do not constitute part of the identity of a data object and are used only for attribution.

```
<Simulation_Model>
      …
      <Attributes>
                  <Attribute name="Name"  read_only="NO" type="TEXT" allow_null="NO" />
                  <Attribute name="Status" type="TEXT" default_value="WIP"/>
                  <Attribute name="Comment" type="TEXT" default_value="" multirow="YES"/>
      </Attributes>
</Simulation_Model>
```

The hierarchical relationships between different data types are described through the **Containing_Items** element. The **Containing_Items** element lists all the data types that are contained within the current type.

```
<Simulation_Model>
      …
      <Containing_Items>
                  <Containing_Item type="Loadcase"/>
      </Containing_Items>
</Simulation_Model>
```



A hierarchical relationship declares that a data object of the lower level data type requires a data object of the higher level data type in order to be defined.

Data Model Format Description

Different data types of Library Items can be defined within the **LIBRARY_ITEMS** element :

```
<LIBRARY_ITEMS>
	<LIBRARYITEM type="Simulation_Configuration_Table" container_name="Simulation_Configuration_Tables">
		<Properties>
			<Property name="Project" type="TEXT" default_value="-" />
			<Property name="Release" type="TEXT" default_value="-" />
			<Property name="Revision" type="VERSIONING_SCHEME" default_value="0" />
			<Property name="Name" type="ATTACHED_FILE" default_value=""/>
		</Properties>
		…
	</LIBRARYITEM>
</ LIBRARY_ITEMS >
```

It is possible to specify rules for the auto-completion of a **Property/Attribute** or control the list of the 'accepted_values' of a **Property/Attribute**, through the **Rules** element.

```
<Simulation_Model>
	<Properties>
		<Property name="Name" type="TEXT" default_value="" generated_value ="YES" />
		<Property name="Discipline" type="TEXT" default_value="-" accepted_values="-,CFD"/>
		<Property name="Team" type="TEXT" default_value="-" accepted_values="-"/>
		…
	</Properties>
	<Rules>
		<Rule name="Name" generated_value="[Model Id]_[Project]_[Release]/>
		<Rule condition_name="Discipline" condition_value="CFD" name="Team" accepted_values="-,AERO,THERMAL"/>
	</Rules>
</Simulation_Model>
```


Data Model Format Description

The allowed transitions between the values of an Attribute, can be controlled through the **ConditionalAttributes** element.
In the example below it is specified that the available transitions of the Status attribute of a Simulation_Model can be:
WIP ➔ OK ➔ Error, whereas the Error Status can not be modified.

```
<Simulation_Model>
	…
	<ConditionalAttributes>
		<condition id="1">
			<attribute>Status</attribute>
			<currentValue>WIP</currentValue>
			<edit>true</edit>
			<acceptedValues>
				<acceptedValue>WIP</acceptedValue>
				<acceptedValue>OK</acceptedValue>
			</acceptedValues>
		</condition>
		<condition id="2">
			<attribute>Status</attribute>
			<currentValue>OK</currentValue>
			<edit>true</edit>
			<acceptedValues>
				<acceptedValue>OK</acceptedValue>
				<acceptedValue>Error</acceptedValue>
			</acceptedValues>
		</condition>
		<condition id="3">
			<attribute>Status</attribute>
			<currentValue>Error</currentValue>
			<edit>False</edit>
		</condition>
	</ConditionalAttributes>
</Simulation_Model>
```

Data Model Format Description

The following elements can be optionally described within the definition of a Data Type and allow further customization of the Data Model. Elements that start from "SPDRM_" are only used by the SPDRM Client.

Element	Description
SPDRM_Gui_Action	Define a default action that should be available for Data Objects of this type in their context menu.
SPDRM_Content_Updates	Automatically modify a Property/Attribute of a parent Data Object based on the value a Property/Attribute of a child Data Object.
SPDRM_Conditional_DMACLs	Automatically modify the access control list (ACL) of a Data Object based on the value of a Property/Attribute.
SPDRM_AutoAdd_DMItem	Define the Data Types of a lower level in the structure which should be added together with this Data Type.
SPDRM_Lock	Allow locking of Data Objects to a specific User Role.
PropertyVaultMapping	Control the Vault where all the Data Objects of the specific Data Type, which match the specified condition, will be stored.
SPDRM_TargetVault	Control the Vault where all the Data Objects of the specific Data Type, which match the specified condition, will be stored.
SPDRM_Export	Control the path to the exported file structure, where all the Data Objects of the specific Data Type, which match the specified condition, will be exported.

Data Model Format Description

Each XML element in the **DM_structure_TBM.xml** file supports several XML attributes. The items in bold are necessary for the definition of a Property/Attribute XML element.

XML attribute	XML Element	Description
name	Property, Attribute	Define the Name of the Property/Attribute.
	Rule	Specify the Name of the Property/Attribute on which a Rule is applied.

type	Property, Attribute	Define the type of a Property/Attribute.
		• Supported types for DM Properties:
		  • TEXT, VERSIONING_SCHEME, VERSIONING_SCHEME_COUNTER, ATTACHED_FILE
		• Supported types for DM Attributes:
		  • TEXT, VERSIONING_SCHEME, VERSIONING_SCHEME_COUNTER, ATTACHED_FILE, ATTACHED_DIRECTORY, LINK_DIRECTORY, INTEGER, FLOAT, DOUBLE

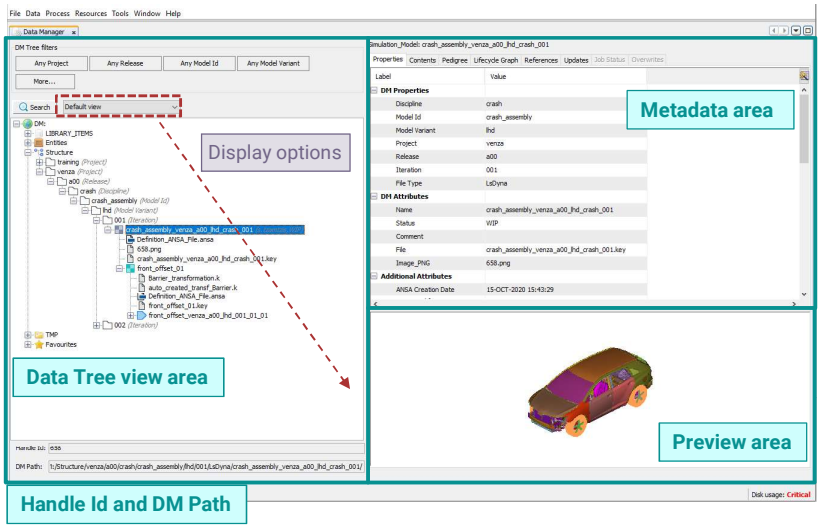
	LibraryItem, Containing_Item, SPDRM_AutoAdd_DMItem	Specify Data Type
default_value	Property, Attribute	The default value of this property, if nothing is defined by the user. Required in case of VERSIONING_SCHEME and VERSIONING_SCHEME_COUNTER properties.
accepted_values	Property, Attribute	Specify a list with the accepted values of the specific Property/Attribute. Applicable only for XML elements of type TEXT.
	Rule	Specify a list with the accepted values of the Property/Attribute on which the Rule is applied. Applicable only for XML elements of type TEXT.
generated_value	Property, Attribute	Specify whether the value of the Property/Attribute will be automatically completed based on a Rule or not. Can be: - YES - NO (Default)
	Rule	Specify the rule for the generation of the selected Property/Attribute. References to Properties/Attributes of the same Data Type are made by embracing the Property/Attribute name in brackets, e.g. [day]. References to Properties/Attributes of parent Data Types are made by preceding the Property/Attribute name by the data type name, using "." as separator. For example, ["type1.Name"].

© Copyright 2017 BETA CAE Systems All rights reserved

Data Model Format Description

XML attribute	XML Element	Description
read_only	Property	Controls whether this Property is editable or not. Can be : - YES - NO (Default)
allow_null	Property, Attribute	Controls whether this Property/Attribute can remain blank or not. Can be : - YES (Default) - NO
position	Property, Attribute	Control the position of a Property/Attribute in the Properties card of Data Object.
format	Property, Attribute	Controls the formatting of the Property value. Is written as a java format specifier, e.g. "%03d" for a 3-digit integer
multirow	Property, Attribute	Controls whether a Property/Attribute of type TEXT will get a text box or a single line-edit for its definition. Can be : - YES - NO (Default)
condition_name	Rule	The name of Property/Attribute of this or of another Data Object whose value will be used to evaluate the condition.
condition_value	Rule	In order to evaluate a Rule, the value of the Property/Attribute defined in "condition_name" should match this value.
discarded_chars	Rule	Specify the characters that should be discarded. This option is applicable in case of a Rule that contains a 'generated_value'.
container_name	LibraryItem	Specify the name of the Container that will hold the Library Item Data Type.

Introduction to Data Views



The form of the Data Structure tree in the Data Manager is controlled through the selected **Display option**.

The 'Default view', as well as any other **Display option**, is fully customizable and can be defined through the **Data_Views.xml** file.

Each data type has certain view modes which are displayed as **Metadata tabs** or through the **Views** option in the context menu of the data object.

The available view modes and the contents of each view mode can be customized through the **Data_Views.xml** file.

The **Data_Views.xml** file is located in following folder in the SPDRM installation directory:
`/server/Wildfly/standalone/configuration/`

It is communicated to the SPDRM server using the `CreateDMStructureWS` web service.

© Copyright 2017 BETA CAE Systems All rights reserved

Data Views Format Description

The top level element of the **Data_Views.xml** file contains the following XML elements :

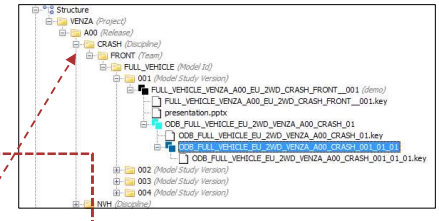
- dm_item_views
- library_item_views
- view_settings
- lifecycle_view

The **dm_item_views** and **library_item_views** sections contain **view** XML elements that define the visibility options of each data type in the Data Manager. Each **view** element is associated with a data type.

Through the following contents of the **view** element, it is possible to control :

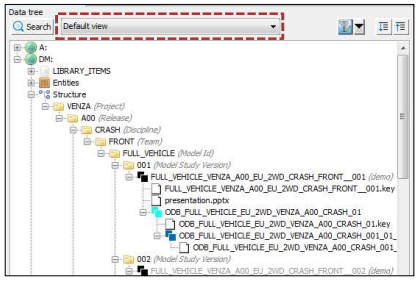
- **data_tree_view_settings** : The default grouping of objects of this data type in the tree structure in Data Manager window
- **info_view_settings** : The information which appears in each view mode for objects of this data type
- **icon** : the icon that marks objects of this data type

```
<dm_item_views>
      <view type="Simulation_Model">
            <icon>SimModel.png</icon>
            <info_view_settings>view_for_sim_models</info_view_settings>
            <data_tree_view_settings>
                  <grouping attribute="Project" position="1" icon="dir.png"/>
                  <grouping attribute="Release" position="2" icon="dir.png"/>
                  <grouping attribute="Discipline" position="3" icon="dir.png"/>
                  <grouping attribute="Team" position="4" icon="dir.png"/>
                  <grouping attribute="Model Id" position="5" icon="dir.png"/>
                  <grouping attribute="Model Study Version" position="6" icon="dir.png"/>
            </data_tree_view_settings>
      </view>
<dm_item_views>
```



Data Views Format Description

Apart form the 'Default view' Display option which is controlled through the **data_tree_view_settings** elements, it is possible to define additional Display options.



The additional Display options are defined through the **custom_tree_views** grouping element. Each Display option is described by a **custom_tree_view** element.

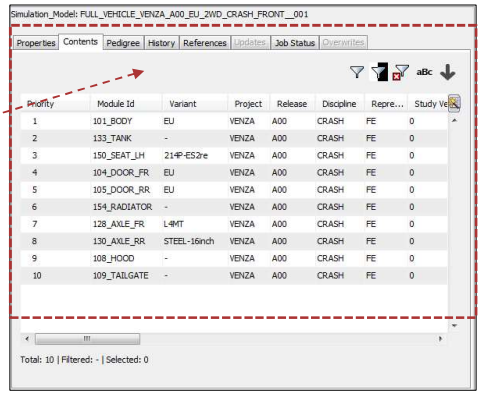
```
<dm_item_views>
      ...
      </custom_tree_views>
            <custom_tree_view name="Grouping by Discipline">
                  <grouping attribute="Discipline" position="1" icon="dir.png"/>
                  <grouping attribute="Team" position="2" icon="dir.png"/>
                  <grouping attribute="Project" position="3" icon="dir.png"/>
                  <grouping attribute="Release" position="4" icon="dir.png"/>
                  <grouping attribute="Model Id" position="5" icon="dir.png"/>
            </custom_tree_view>
            ...
      </custom_tree_views>
<dm_item_views>
```

Data Views Format Description

The view settings associated with each Data Type through **dm_item_views > view > info_view_settings** must be defined in the **view_settings** XML element.

The **info_view_settings** element controls the information that appears in the view modes of Data Objects of the respective Data Type.

```
<dm_item_views>
      <view type="Simulation_Model">
            <icon>SimModel.png</icon>
            <info_view_settings>view_for_sim_models</info_view_settings>
            <data_tree_view_settings>
                  …
            </data_tree_view_settings>
      </view>
<dm_item_views>
<view_settings>
      <info_view_settings index="view_for_sim_models">
            <show_tab>Properties</show_tab>
            <show_tab>Contents</show_tab>
            …
            <tab_content name="Contents">
                        <list_view_settings type="Subsystem">
                        <show_column>__order__</show_column>
                        …
                  </list_view_settings>
            </tab_content>
            <tab_content name="History">
                  …
            </tab_content>
      </info_view_settings>
</view_settings>
```



Data Views Format Description

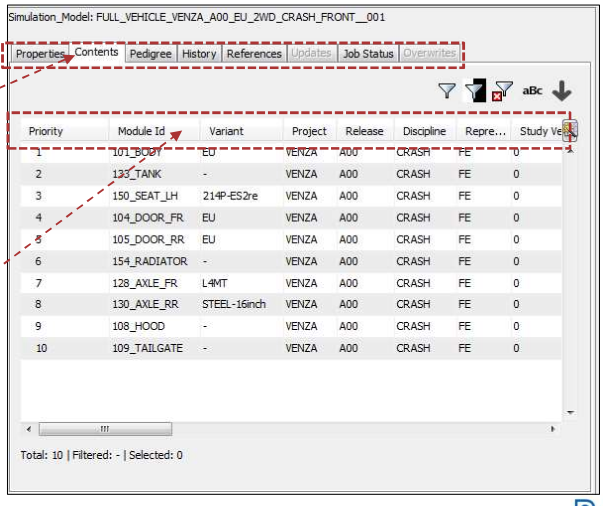
The available view modes are reported below :

- Contents
- Pedigree
- References
- Job Status
- Lifecycle graph

The format of all view modes is a list view, apart from the Lifecycle graph. The activation of each view mode for the specific data type is controlled through the **show_tab** element.

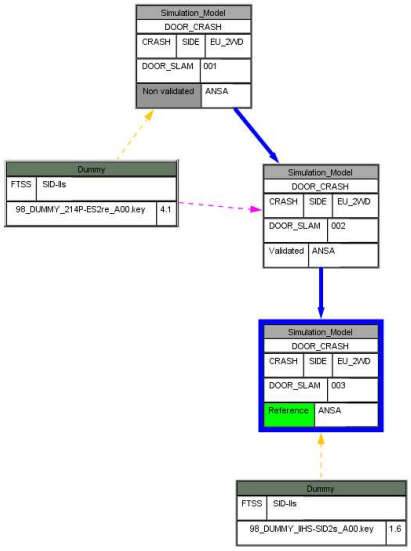
For list view modes it is possible to control the default visible columns for data objects of each data type, through the **show_column** element.

```
 <info_view_settings index="view_for_sim_models">
      <show_tab>Properties</show_tab>
      <show_tab>Contents</show_tab>
      …
       <tab_content name="Contents">
            <list_view_settings type="Subsystem">
                  <show_column>__order__</show_column>
                  <show_column>Module Id</show_column>
                  <show_column>Variant</show_column>
                  <show_column>Project</show_column>
                  <show_column>Release</show_column>
                  <show_column>Discipline</show_column>
                  …
            </list_view_settings>
       </tab_content>
       ..
</info_view_settings>
```



Data Views Format Description

The Lifecycle view is available through the context menu of all data objects. The Lifecycle graph displays the evolution of a data object. However, apart from history links, the Lifecycle graph also displays references to other items.



More specifically the Lifecycle graph shows the following types of relationships:

- History links
- Manual links which are further categorized in
    i. Direct
    ii. Inherited

Through the **lifecycle_view** element it is possible to control how objects of each data type will appear in the Lifecycle graph. For each data type a **dm_object** element must be defined.

```
<lifecycle_view>
      <dm_object name="Simulation_Model">
            <table rows="5" columns="3">
                  <cell_content row="1" column ="1" name="Type" value_type="spdrm_attribute"/>
                  …
                  <highlight_value_cell name="Type">
                        <condition>
                              <value>Simulation_Model</value>
                              <color red="170" green="170" blue="170"></color>
                        </condition>
                        …
                  </highlight_value_cell>
                  …
            </table>
      </dm_object>
</lifecycle_view>
```

SPDRM
WORKFLOW MANAGER

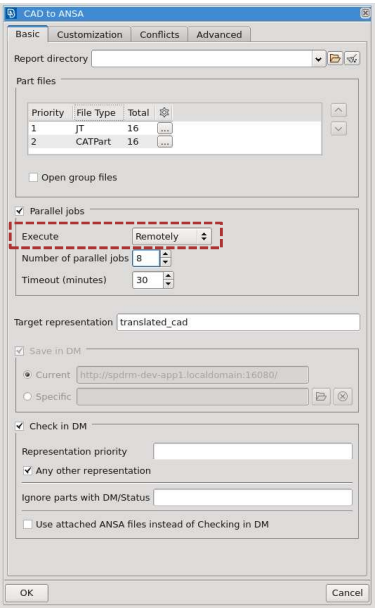
Plugins

BETA
SIMULATION SOLUTIONS

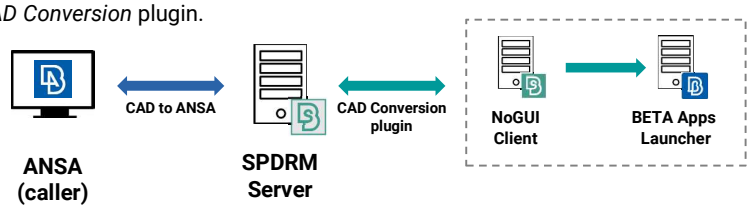
Table of contents

Introduction to the CAD Conversion plugin

CAD to ANSA is a tool offered by ANSA to facilitate the translation of CAD files, importing them in the current ANSA model and saving the target representation in the data management repository. The process can be executed sequentially in the existing ANSA session or using parallel ANSA workers for the translation of Parts.



Using **ANSA v22.0.0** or later, the user can enable the execution of *Parallel jobs* and select to execute the process *Remotely*. This requires that the ANSA session is connected to a properly configured environment where SPDRM version **1.5.2** or **1.6.0** (or any other later version) is deployed and the ***CAD Conversion*** SPDRM plugin is installed (as described in the SPDRM Installation Guide). In that case, the translation will be executed on remote resources (i.e. a *BETA Apps Launcher*), utilizing the built-in *CAD Conversion* plugin.

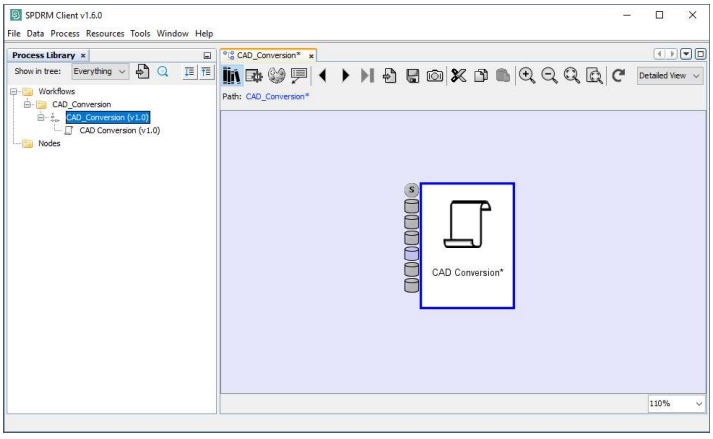


The current ANSA session (i.e. caller) will instantiate the *CAD Conversion* plugin workflow, which will be executed automatically in a NoGUI SPDRM Client. The CAD files will be translated in a remote ANSA session (i.e. worker), in a properly configured BETA Apps launcher, imported back to the model in the ANSA caller and based on the CAD to ANSA settings, saved in the SPDRM data vault with the target representation.

The execution of the plugin is performed automatically in SPDRM, no user action is required.

CAD Conversion plugin workflow

The *CAD Conversion* plugin workflow is available as a template in the Process Library and consists of a single script node.



CAD Conversion node variables

BAL_server *(Type=String, Default value=<empty>)*

A string variable that can be used to explicitly define the BETA Apps launcher to be used by the CAD conversion plugin. Its value needs to be filled with the *Name* of the desired BETA Apps launcher.

If empty, the plugin process script will identify an appropriate BETA Apps launcher based on the value of the *BAL_ansa_app_identifier* variable *(see below)*.

BAL_ansa_app_identifier *(Type=String, Default value=<empty>)*

A string variable that can be used to explicitly define the ANSA application in the BETA Apps launcher to be used by the CAD conversion plugin. Its value needs to be filled with the *Title* of the desired application.

If empty, the plugin process script will identify an appropriate application based on the running value of the *starting_ansa_version* input slot.

CAD Conversion node Input Slots

starting_ansa_version

A string passed to the workflow from the ANSA caller. It describes the ANSA version in use and can be utilized to find an appropriate application in the BETA Apps launcher.

ansa_defaults

The ANSA defaults file used by the ANSA caller. It will be used by the ANSA worker that will be launched in the BETA Apps launcher to perform the CAD translation

translator_defaults

The translator defaults file used by the ANSA caller. It will be used by the ANSA worker that will be launched in the BETA Apps launcher to perform the CAD translation.

input_data

A json file containing the CAD to ANSA settings from the ANSA caller and the part attributes of the files to be translated

post_actions_script (optional)

A python script file that may be defined in the CAD to ANSA settings of the ANSA caller. It will be used by the ANSA worker that will be launched in the BETA Apps launcher to perform the CAD translation, in order to perform a post-translation treatment per part.

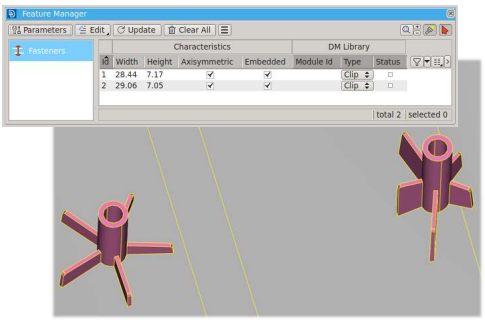
ansa_script

A python script file from the SPDRM library, executed by the ANSA worker that will be launched in the BETA Apps launcher to perform the CAD translation.

process_script

A python script file from the SPDRM library that drives the execution of the CAD Conversion SPDRM plugin

Introduction to the ML Predictor Training plugin



The ***ML Predictor Training*** plugin can be used to train an Embedded Clips Predictor, which can learn to recognize the Clips provided.

It involves the execution of a workflow that takes a training dataset of annotated Clip Feature Entities as input from the end user. Training gets executed in a remote machine with the use of a BETA Apps Launcher.

The output of the workflow is a Predictor DM object that is automatically saved to the DM.

The plugin execution is available using **ANSA** or **KOMVOS v22.0.1** or later, connected to a properly configured environment where **SPDRM v1.6.1** (or any other later version) is deployed and the ***ML Predictor Training*** SPDRM plugin is installed (as described in the SPDRM Installation Guide).

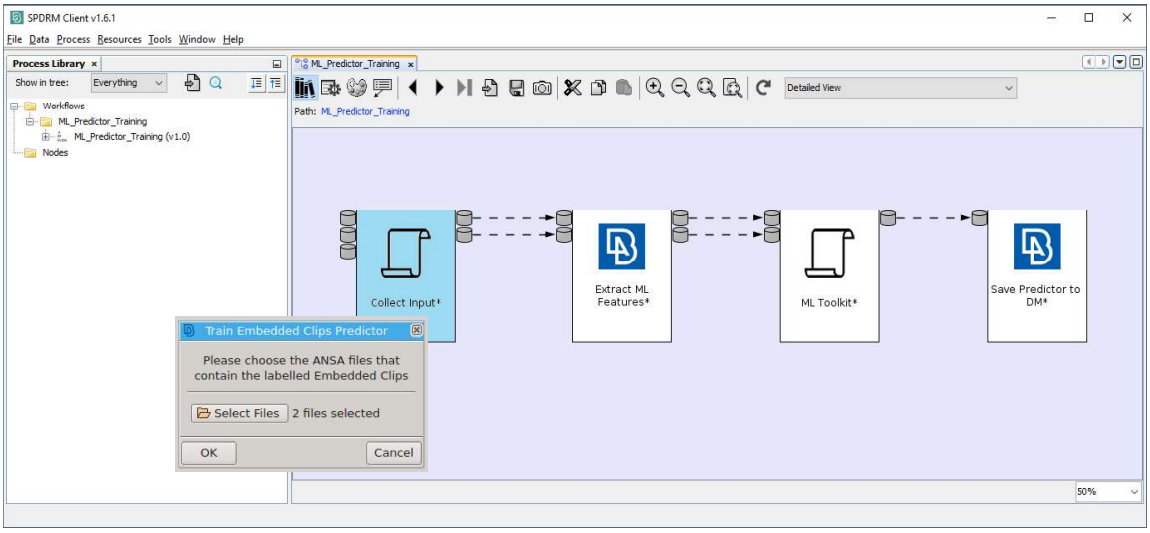
The workflow execution can be triggered through:

- ANSA, using the *DM Browser* action **Script Actions > Train Embedded Clips Predictor (remote)**
- KOMVOS, using the **Machine Learning >Train Embedded Clips Predictor (remote)** action in the *Actions Menu*
- SPDRM, using **Tools > Generic Script Actions > Machine Learning >Train Embedded Clips Predictor (remote)**

Depending on the input dataset size and the GPU memory and compute power of the remote machine, execution can take from several hours to several days.

ML Predictor Training plugin workflow

The *ML Predictor Training* plugin workflow is available as a template in the Process Library and consists of a combination of script and ANSA application script nodes.



All process nodes are automatically executed. Once the training dataset of annotated Clip Feature Entities is provided as input during the execution of the first workflow node, no other user interaction is required through the execution of the plugin.

It should be noted that all process nodes **input** and **output slots** are either scripts loaded from the SPDRM Data Library or data handled internally by the process and **should not be tampered with** by the user.

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