SPDRM
WORKFLOW MANAGER

Tutorial

Execution and Design of processes

BETA
SIMULATION SOLUTIONS

Contents

Introduction

This document is a tutorial for the familiarization of the user with the process execution and the process design within SPDRM. It is intended to provide to new users insight on the tools used in order to

- execute and monitor existing processes, and
- design a ready-to-execute process from scratch.

The user should then be able to execute but also create workflows according to the requirements of the organization.

This tutorial is focused on the execution of three workflows:

- A workflow for the build and save of a Subsystem,
- A workflow for setting up a ready-to-run Simulation Run from the previously created subsystem,
- A workflow for solving the previously saved Run and post-processing the results.

Then, the tutorial is focused on the design of the first workflow, the Build Subsystem.

Prerequisites are the basic terms and GUI features of SPDRM. Reading the sections “Process Design” and “Process Execution” of the SPDRM User’s Guide is recommended. Reading the “SPDRM Introduction” is also recommended in order to obtain a familiarization with the SPDRM interface and terminology. Finally, a basic knowledge of ANSA functionality and Data Management principles is considered essential.

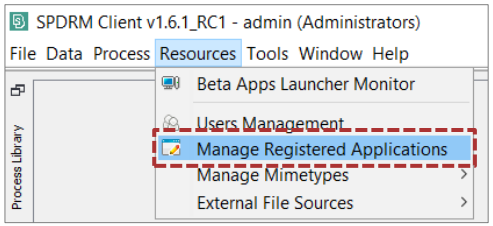
The files used in this tutorial are located in the directory **tutorials** which can be found inside the SPDRM installation directory. Throughout this document, all paths will be referred relatively to this folder.

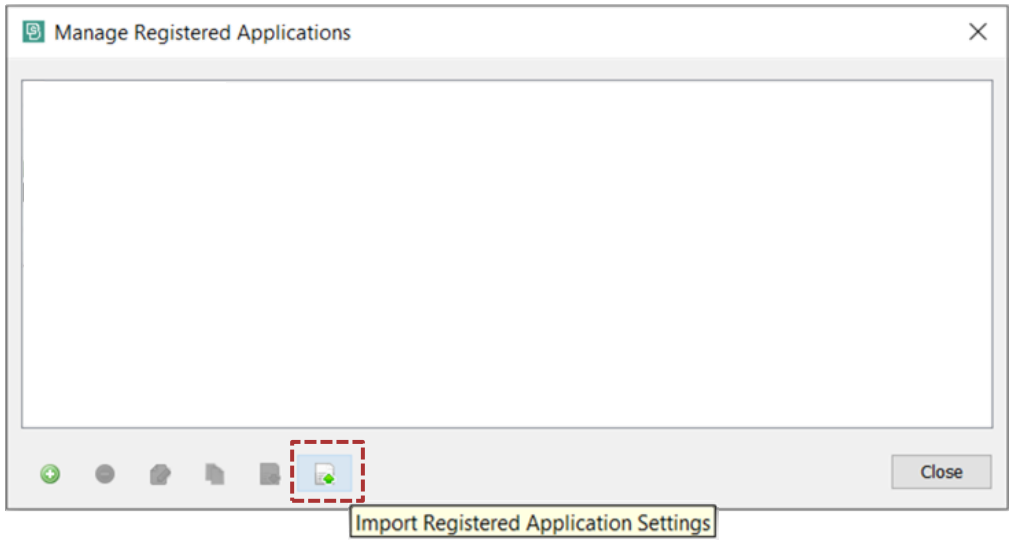
Import Data

Registered Applications

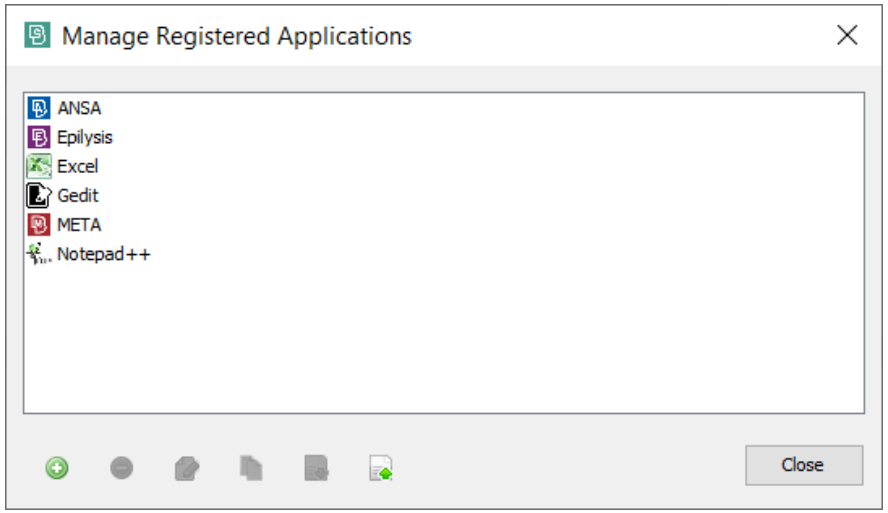
Before proceeding to the execution and design of the processes, we must prepare the environment by importing all the needed data and resources. First we will register the applications that will be used during the execution of the processes.

1 Go to **Resources>Manage Registered Applications** and in the window that appears press the **Import** button.

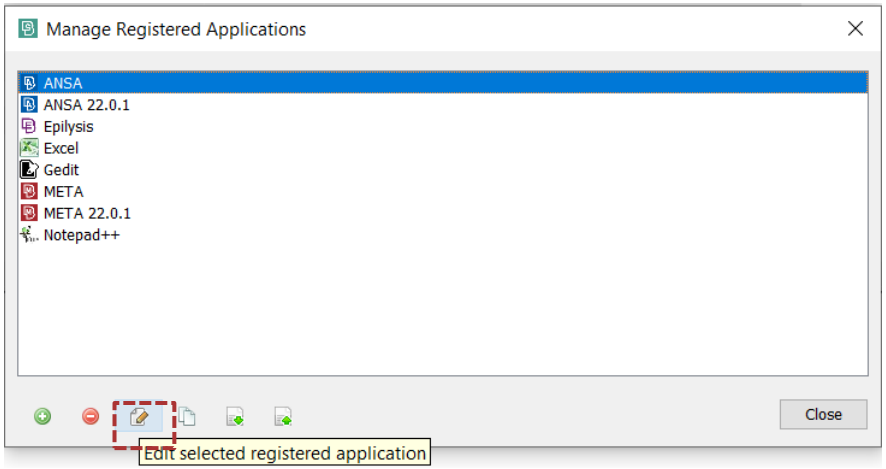




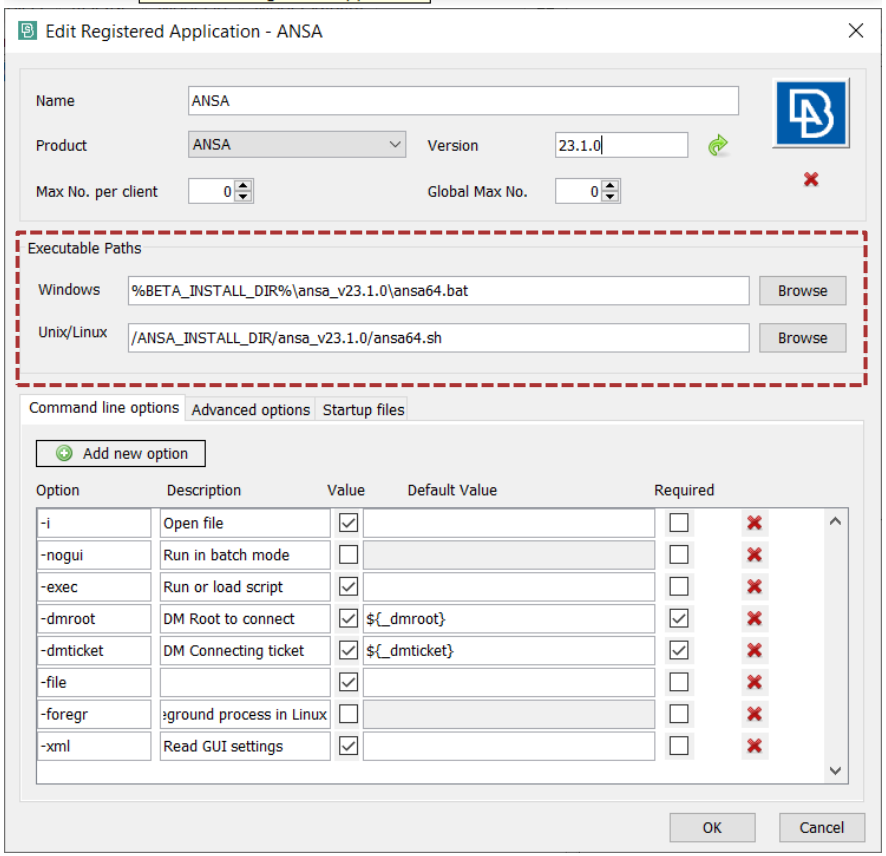
2 Select to import the **Registered_applications.json** file in the tutorials folder. The applications are populated in the window.



Update the settings of the imported applications



1 Go to **Resources > Manage registered applications**. The *Manage registered applications* window appears. Press **Edit** to open the *Application info* window for each application.

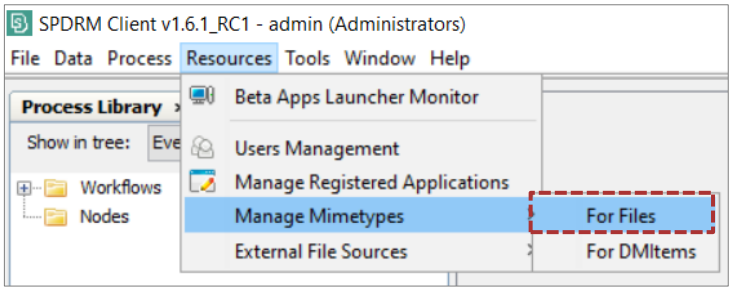


2 Edit the imported applications and correct their executable paths so as to point to valid executable paths of your system (suggested versions for ANSA/META/Epilysis: 23.1.0).

Mimetypes

Mimetypes are means through which SPDRM understands which applications are supposed to be used for opening particular kind of Files/DMItems. In order to set this up, the following steps need to be followed.

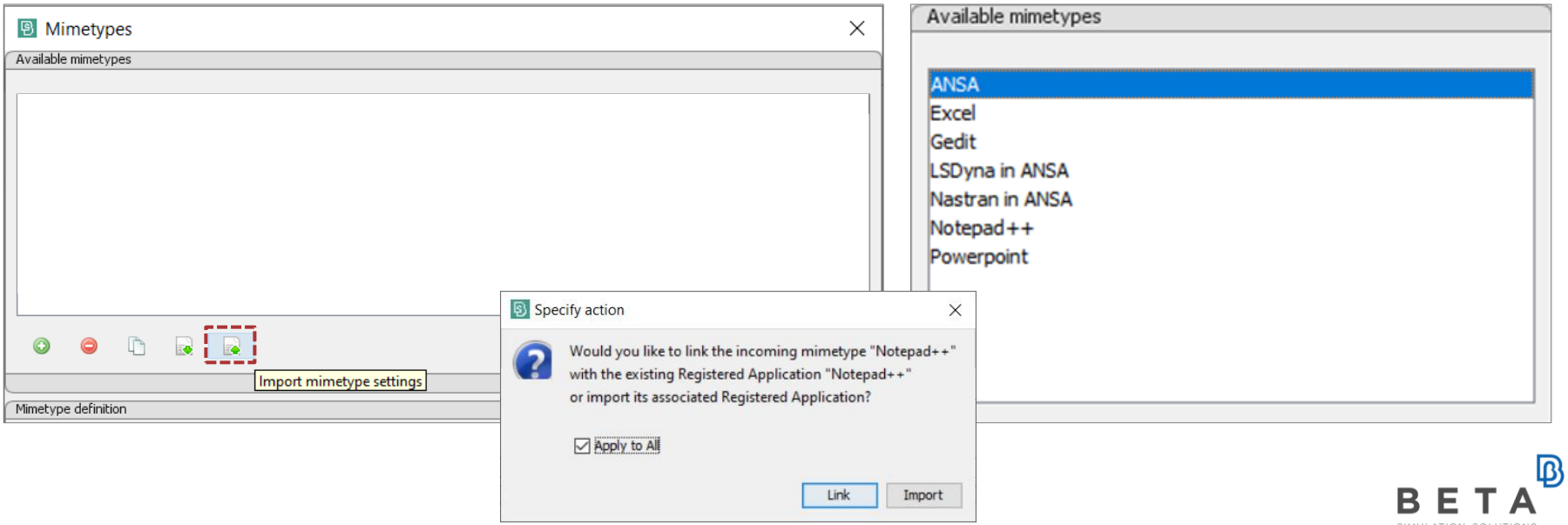
1 Go to **Resources>Manage Mimetypes>For files** and in the window that appears press the **Import** button.



2 Select to import the **Mimetypes_for_files.json** file in the tutorials folder. Link all the incoming mimetypes with the existing Registered Applications.

The list with the available Mimetypes is updated.

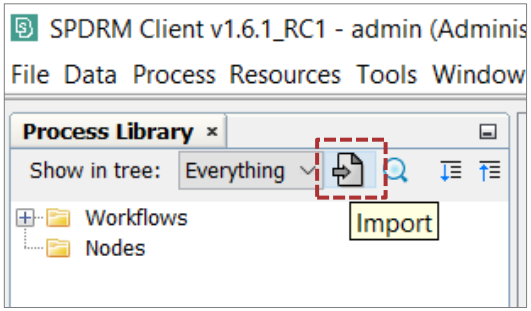
Repeat the process for the Mimetypes for the DM Items, using the file **Mimetypes_for_DM_items.json**.



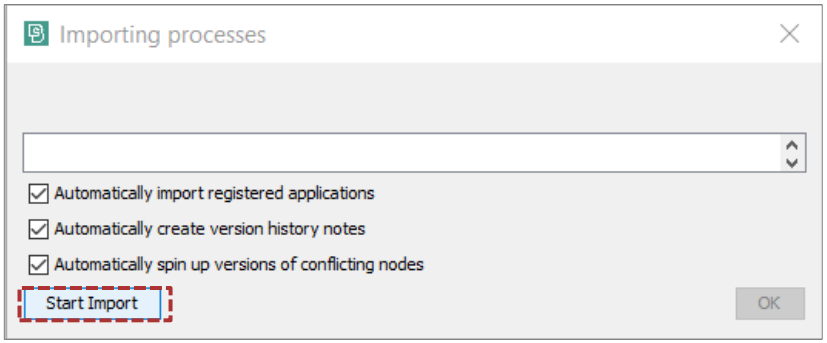
Process Templates

Processes templates are definitions of SPDRM workflows saved in .json files. In order to import the process templates for this tutorial follow the steps below.

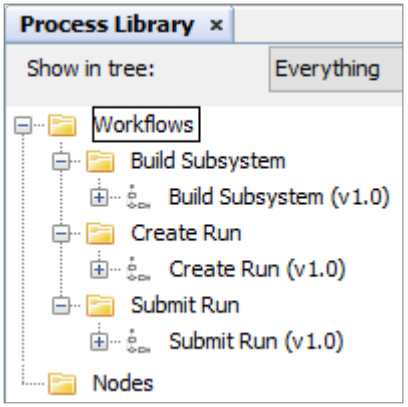
1 Go to **Process > Process Library**. In the *Process Library* window press the **Import** button.



2 The *File Manager* window appears. Select all the processes of the **process_templates** directory. The *Importing processes* window appears. With all the options active, press the **Start Import** button to import the processes.



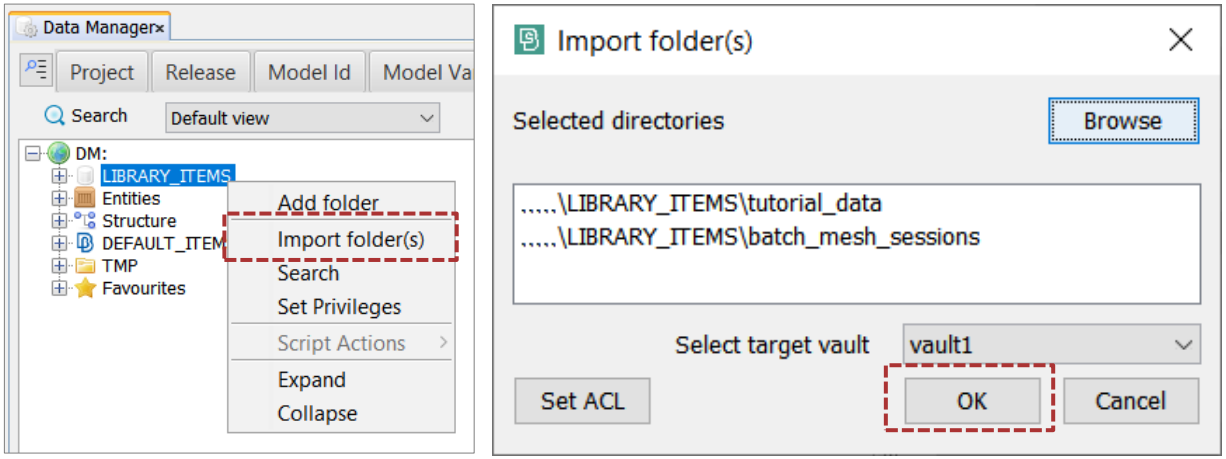
3 The processes are imported and appear in the Process Library tree.



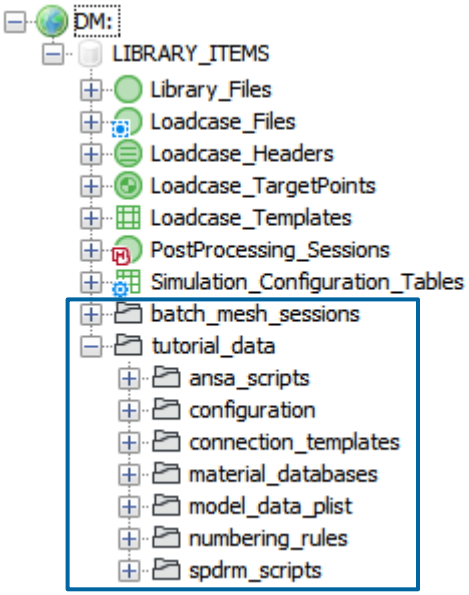
Files/Folders

The data required for the execution of the processes can be found in the **LIBRARY_ITEMS**. These will be imported under the *LIBRARY_ITEMS* container of the *Data Manager,* folder inside the installation directory.

1 Go to **Data > Data Manager**. Access the context menu of the **LIBRARY_ITEMS** container and select **Import folder(s)**.



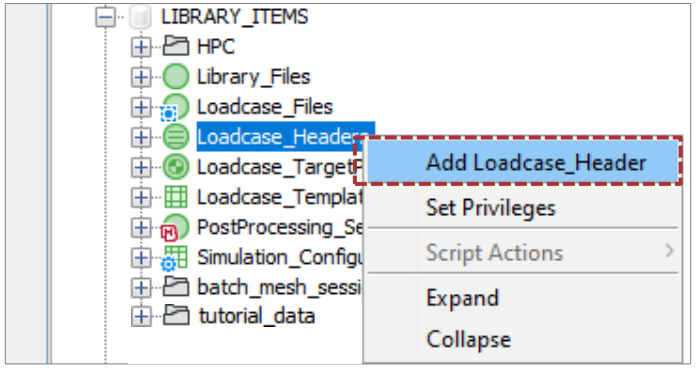
2 Import the content directories from **../LIBRARY_ITEMS**. The directories are imported and appear as sub-folders of the *LIBRARY_ITEMS* container.



Loadcase Header

For the execution of the Create Run process, a Loadcase_Header file will be needed. This is a Rich Library Item and must be imported as such with the correct type and properties. Rich Library Items appear also as sub-contents of the *LIBRARY_ITEMS* container, however they are not simple files but DM Items that are characterized by a unique set of Properties.

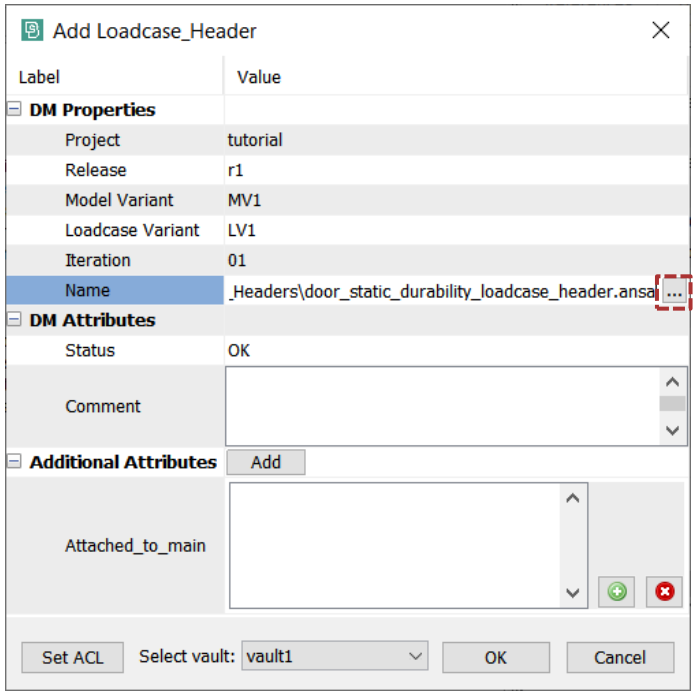
1 To import the Loadcase_Header go to the *Data Manager* and access the context menu of the respective container. Activate option **Add Loadcase_Header**.

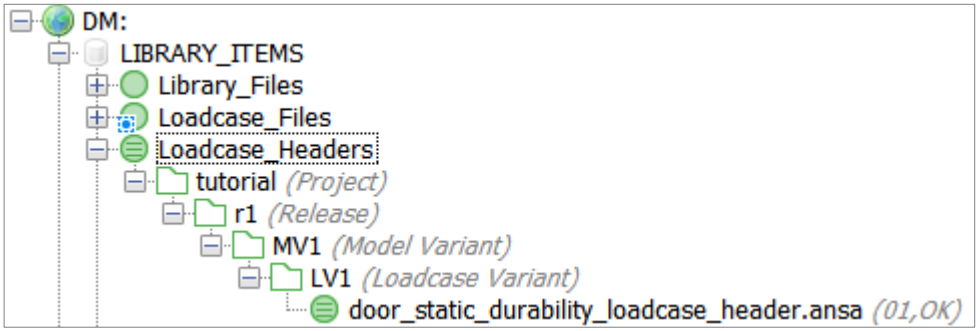


2 In the *Add Loadcase_Header* window that appears specify the properties as shown below. The *Name* property is the one that hosts the file. Press the **Browse** button and select the file

../Loadcase_Headers/door_static_durability_loadcase_header.ansa

When finished press **OK** to import the file. The structure is created based on the given properties and the file appears as attachment of the respective Rich Library Item.



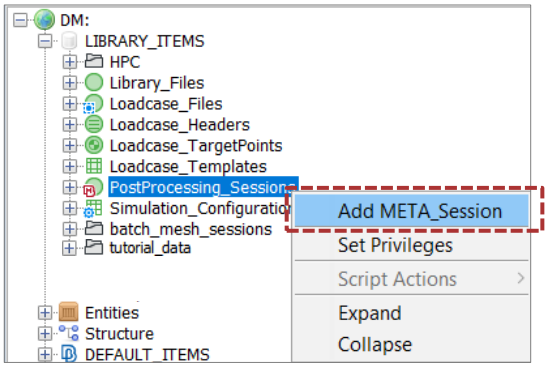


BETA SIMULATION SOLUTIONS

Session file

For the execution of the Post Process Noder in the Submit Run process, a PostProcessing_Session file will be needed. This is a Rich Library Item and must be imported as such with the correct type and properties. This DM Item is characterized by a unique set of Properties.

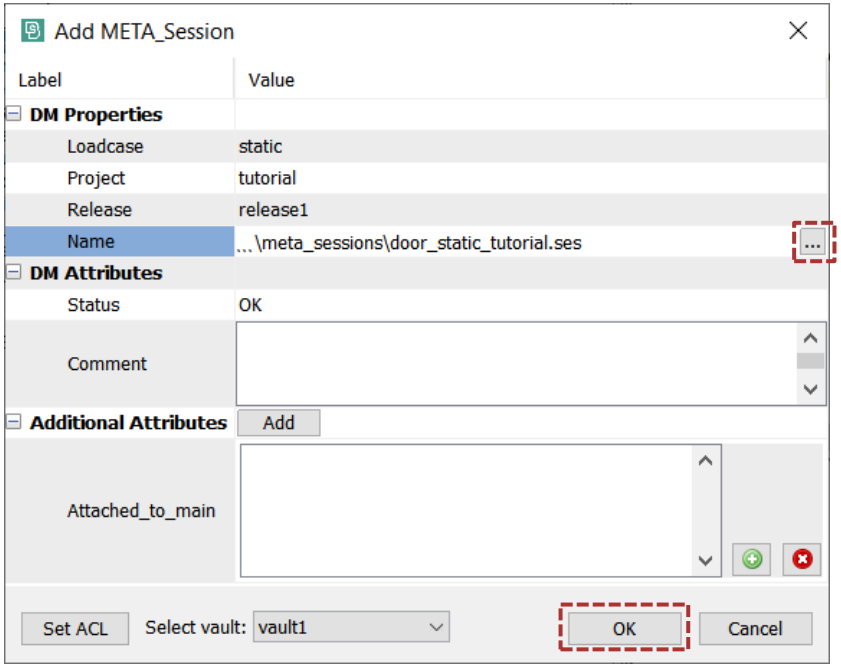
1 To import the Post processing session go to the *Data Manager* and access the context menu of the respective container. Activate option **Add META_Session**.

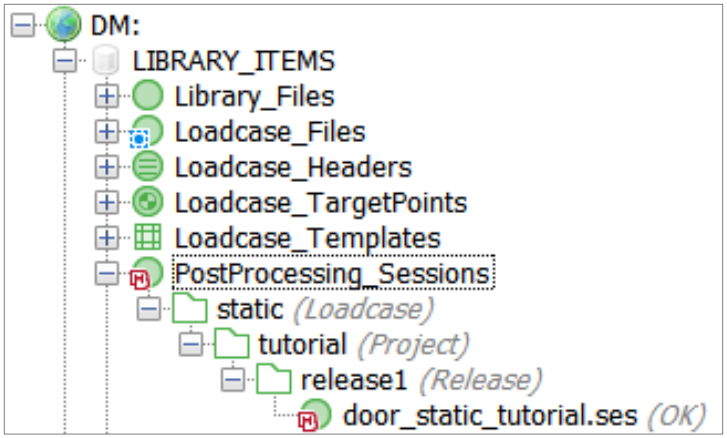


2 In the *Add META_Session* window that appears specify the properties as shown below. The *Name* property is the one that hosts the file. Press the **Browse** button and select the file

../ meta_sessions/door_static_tutorial.ses.

When finished press **OK** to import the file. The structure is created based on the given properties and the file appears as attachment of the respective Rich Library Item.

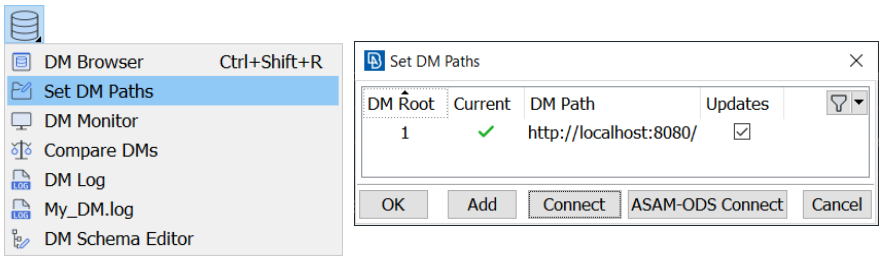




Modular Environment Profiles

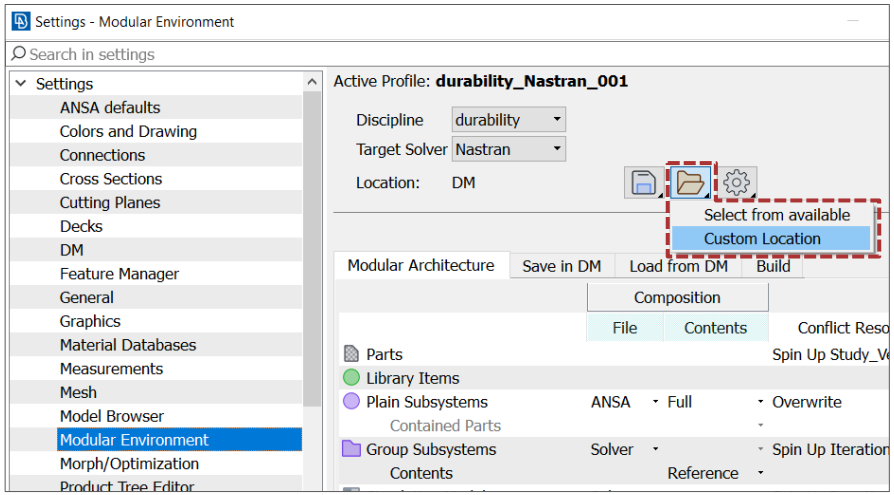
The Modular Environment Profiles (MEP) are sets of ANSA settings that will be used during the execution of the Build Subsystem process. In SPDRM these are found under *DEFAULT_ITEMS > Modular_Environment_Profiles* container. To import a MEP in the system, the user should save it through ANSA while connected to the SPDRM DM back-end.

1



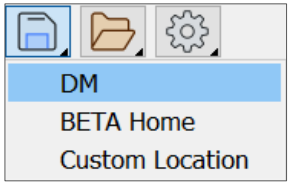
Open an ANSA v23.1.0 and connect to the SPDRM DM through **Lists>DM>Set DM Paths**, and the button Connect.

2



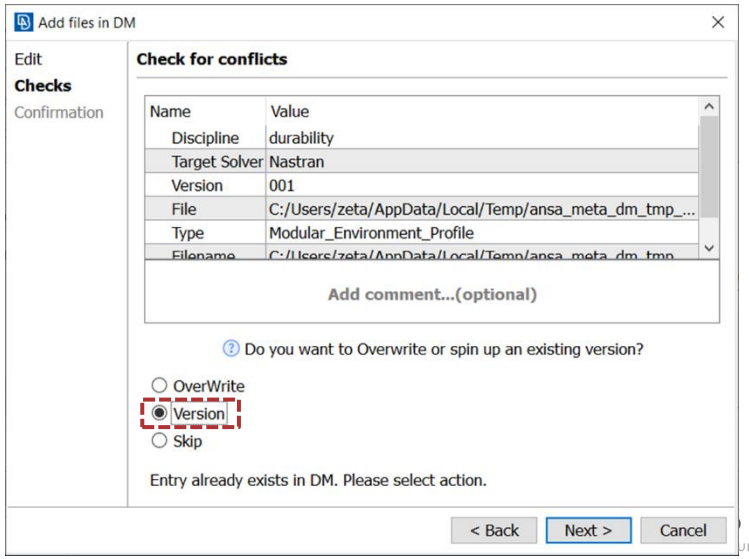
Go to **Tools>Settings>[Settings->Modular Environment]** and select to read the MEP from a **Custom Location**. Select the folder `../meps/durability_Nastran`.

3



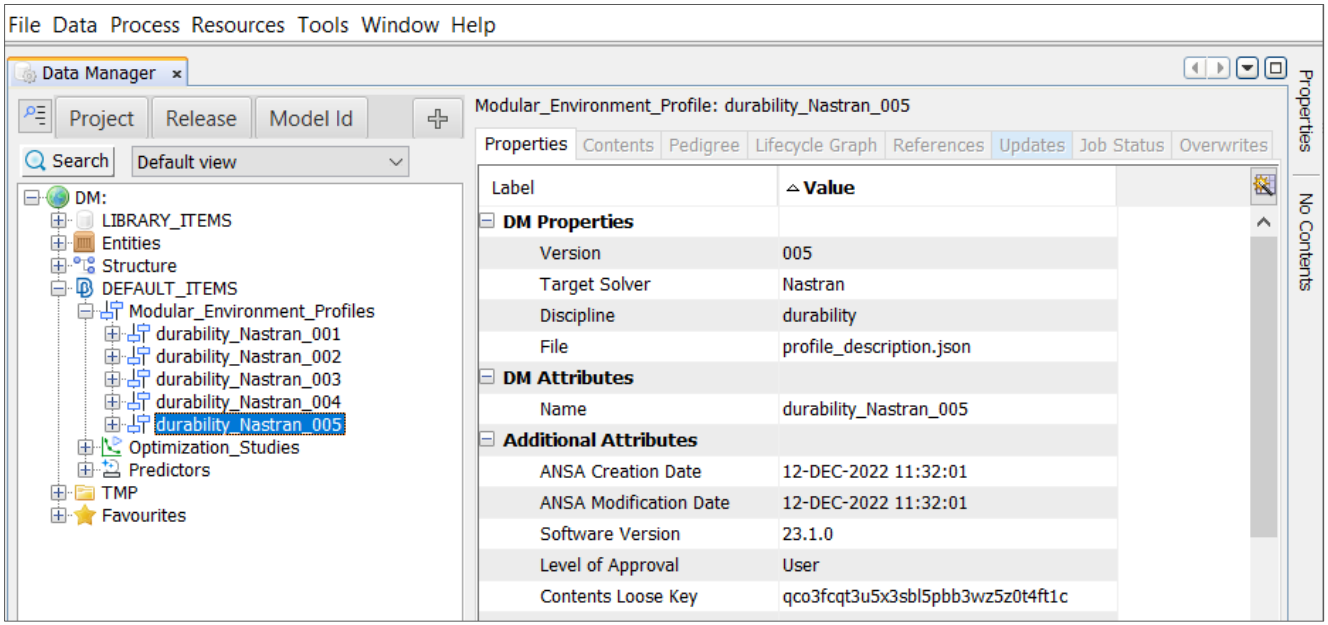
Proceed with saving the MEP to **DM**, following the saving procedure.

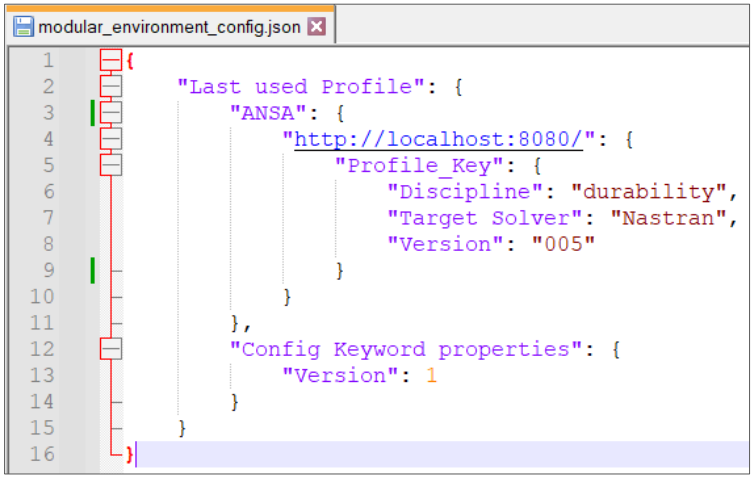
Select to spin up the Version of the MEP if any conflicts with existing DM Item exist.



Modular Environment Profiles

Now that the MEP is saved in the DM, a new DM Object has been created and can be viewed in the *Data Manager*. This DM Object must be coupled with the ANSA registered application that will be used along the process. The coupling is done through a configuration file, following the next steps.



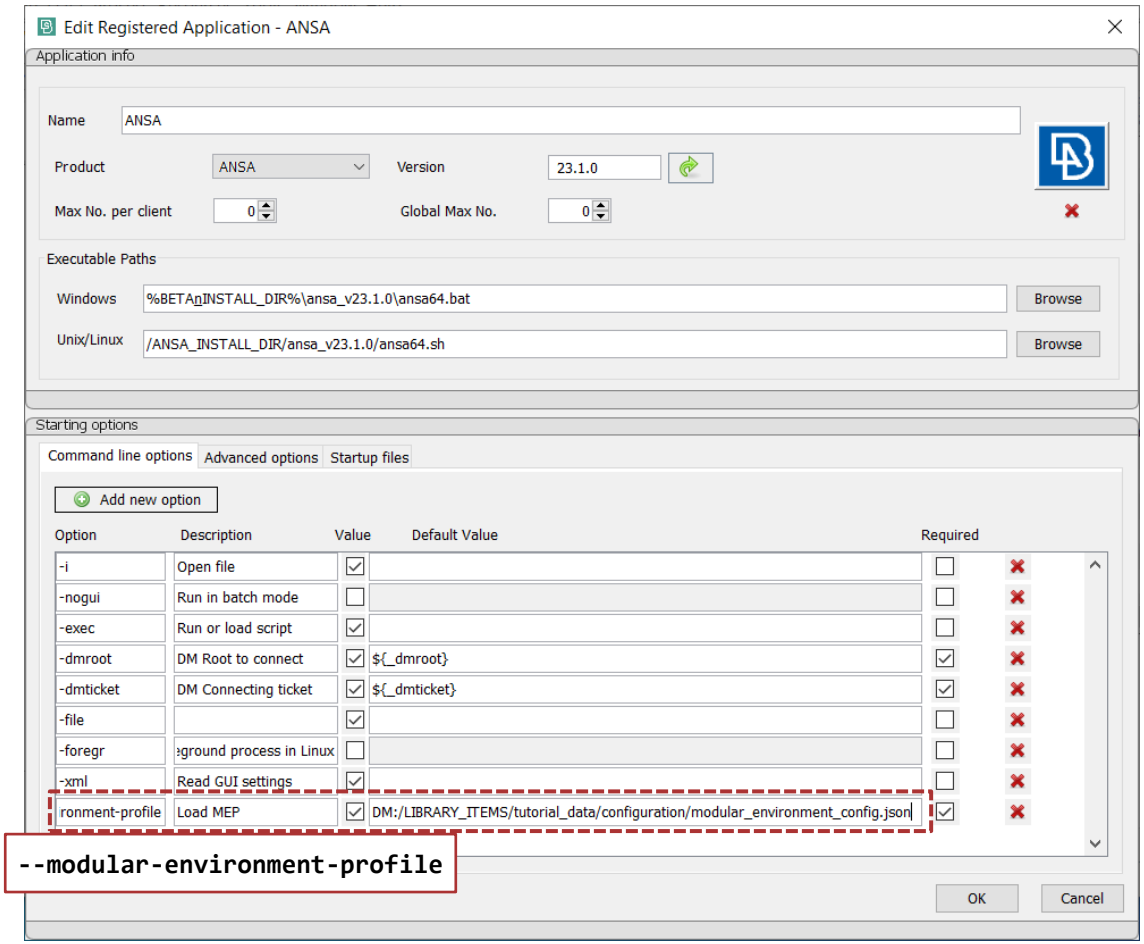


4 Open with a text editor the file `..meps/modular_environment_config.json`.

This will be the file that will instruct ANSA to read the correct set of settings. Edit the file so as the written **Profile_Key** to match the one of the MEP DM Object, as this was saved in SPDRM. It is the **Version** property that should be corrected. Save the file.

Modular Environment Profiles

Next the ANSA registered application should be edited so as to read this file.



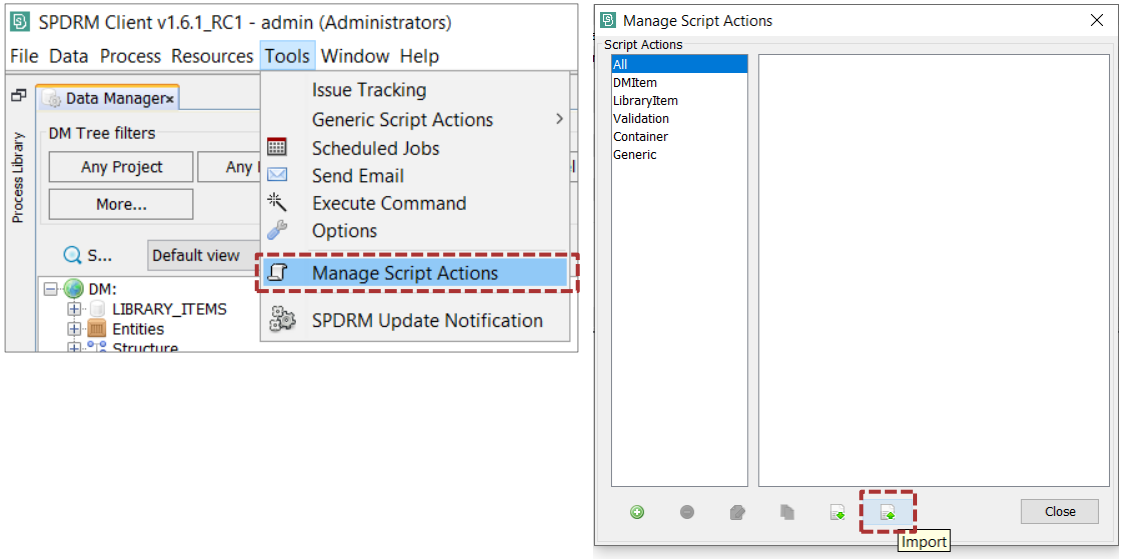
5 Edit the ANSA registered application as described previously.

Add one new option as shown in the image, pointing to the previously imported .json file.

Script Actions

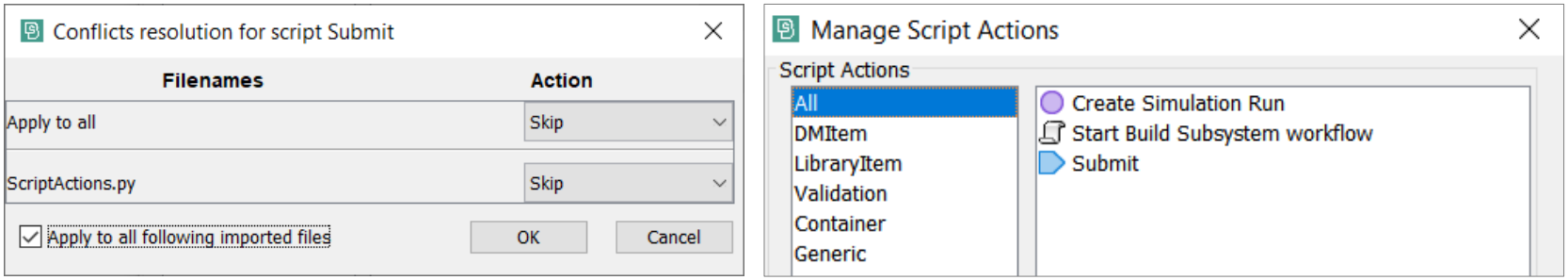
Next we will import the script actions. These are python scripts executed from the main menu of the SPDRM client or on DM Items, so as to perform custom tasks.

1 Go to **Tools > Manage Script Actions**. In the *Manage Script Actions* window press the **Import** button.



2 Select to import the **…/ScriptActions.zip** file. Skip to any conflicts from imported script files.

The script actions are populated in the window.



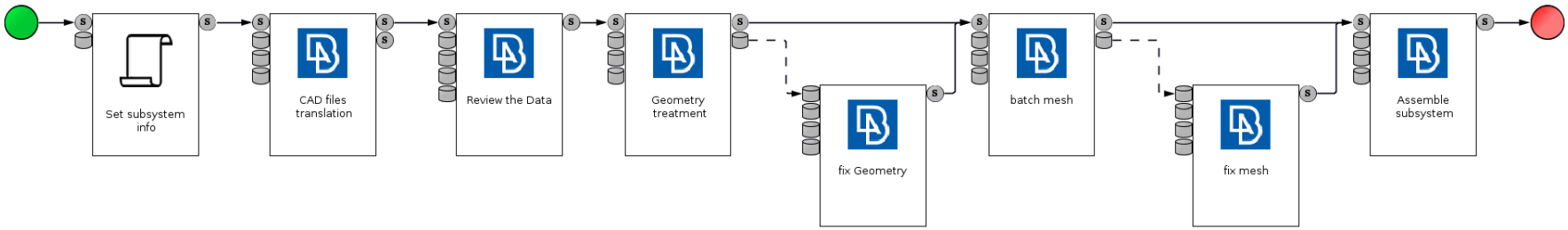
Overview of the processes

Build Subsystem

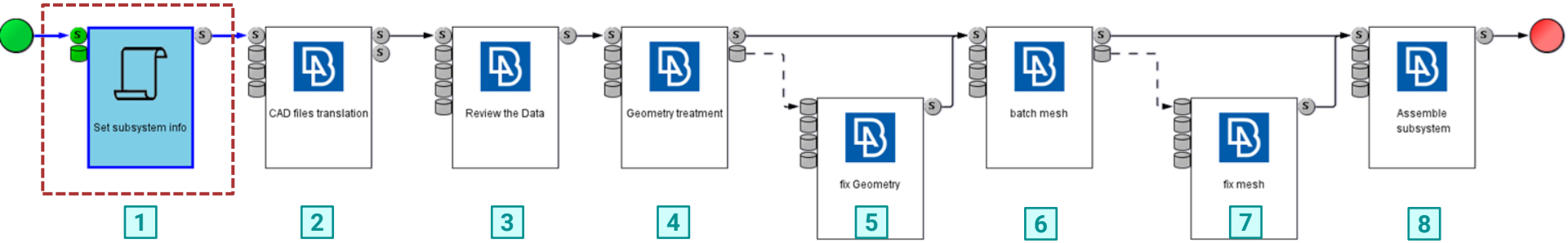
BETA
SIMULATION SOLUTIONS

Build Subsystem

This is a workflow that guides an end-user through the build of a subsystem. From the definition of the primary properties of the subsystem, to the CAD translation of the respective parts, the treatment of possible geometrical errors and the meshing of the translated parts, up to the final assembly and save of the subsystem to the SPDRM repository. Such a workflow should manage all the data that are needed as well as the resources that will be used.

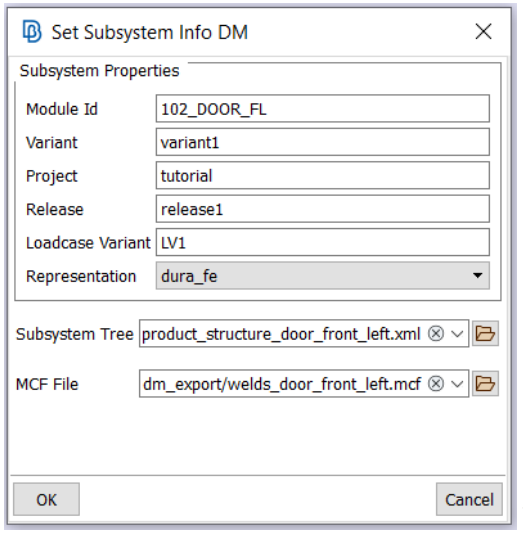


Build Subsystem



Node 1: Set subsystem info

This is an automatically executed Script Node, i.e. it will be executed once the workflow is instantiated by the user.
Its purpose is first to produce a custom interface in which the user will input all the information as well as files, that are needed for the built of the subsystem, and then to deliver all this information to the rest workflow.

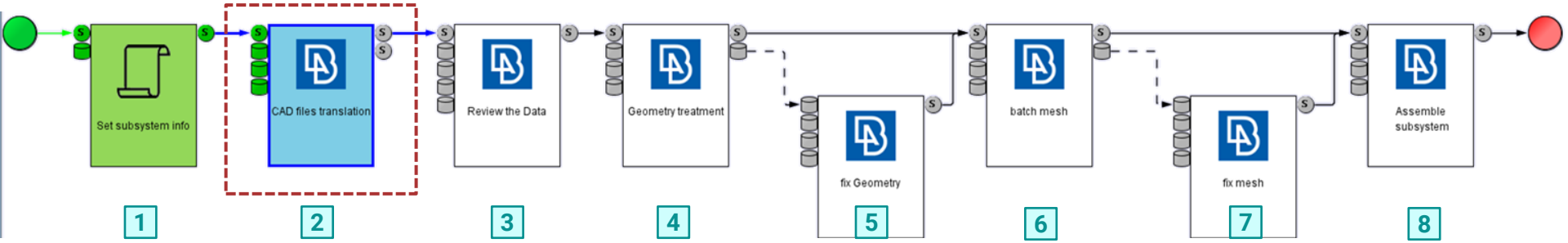


Input data (Input Slots):
- a 'start' signal, in the form of a string variable
- the python script file (process_script) that will produce the custom interface

Output data (Output Slot):
- the path of the model definition file of the subsystem (given by the user in the custom interface), in the form of a string variable (subsystem_tree)

Output data (Variables of the workflow):
- the path of the connections file of the subsystem (given by the user in the custom interface)
- all the properties of the subsystem as they will be defined by the user via the custom interface. The input values in the custom interface will update the runtime values of the respective variables.

Build Subsystem



Node 2: CAD files translation

This is an automatically executed Application ANSA Node. Its purpose is to launch ANSA with specific settings. ANSA will perform the translation of the CAD files of the parts of the subsystem and will save them into the DM. ANSA will be launched in no-gui mode, it will run a specific python script and then it will terminate.

Input data (Input Slots):

- the path of the model definition file of the subsystem, in the form of a string variable (subsystem_tree)
- the python script file that will be executed by ANSA(BuildSubsystemScripts.py)
- the python script file that will be executed by SPDRM before ANSA is launched and after it has quit (Build_Subsystem_PrePostRun.py)
- the python script used by the build actions of the loaded ANSA Modular Environment Profile

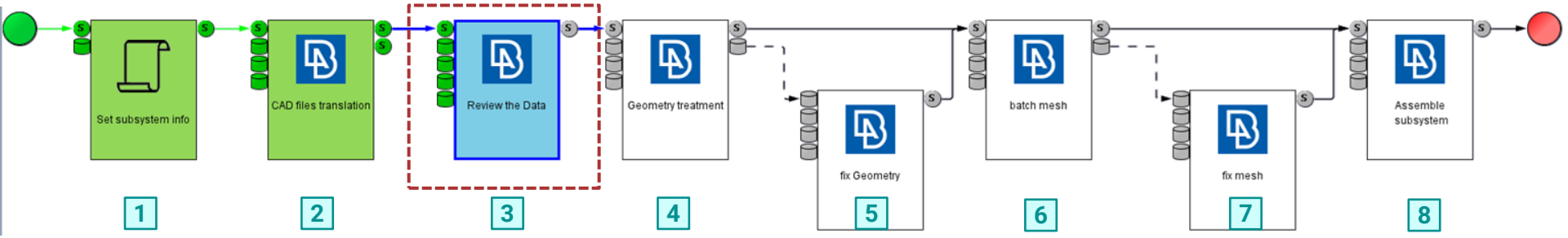
Output data (Output Slot):

- a signal to continue to the next node, in the form of a string variable
- a string variable which is updated upon termination of ANSA with the information of the outcome of the job, i.e. success or failure (0 or 1, respectively).

Output data (Variable of the node):

- the name of a .txt file which contains the values of the subsystem's properties and it is updated by ANSA

Build Subsystem



Node 3: Review the Data

This is an Application ANSA Node. Its purpose is to launch ANSA with specific settings and provide also a User Script button for deleting the parts which are not required for the particular analysis. The script prompts the user to delete not needed parts. Once deletion is completed it proceeds to save the parts and the updated subsystem in DM.

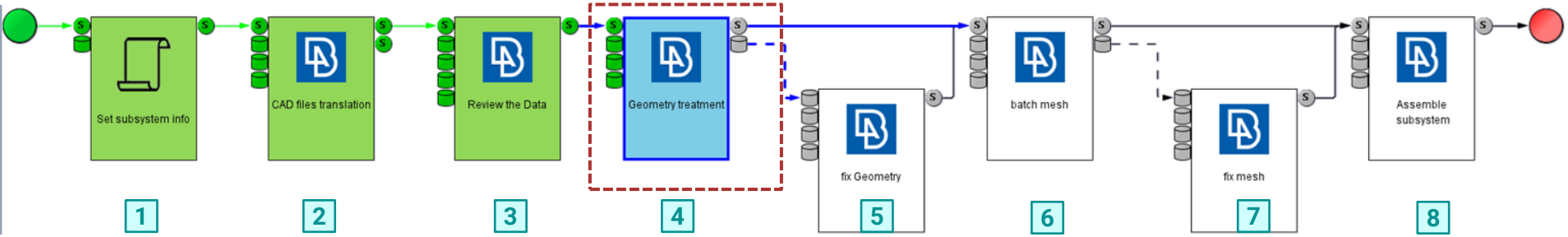
Input data (Input Slots):

- a string value signal passed from the previous node (input_1)
- the python script file that will be executed by ANSA (BuildSubsystemScripts.py)
- the python script file that will be executed by SPDRM before ANSA is launched and after it has quit (Build_Subsystem_PrePostRun.py)
- the python script used by the build actions of the loaded ANSA Modular Environment Profile
- the GUI settings file (ANSA.xml) to customize ANSA UI

Output data (Output Slot):

- a signal to continue to the next node, in the form of a string variable

Build Subsystem



Node 4: Geometry treatment

This is an automatically executed Application ANSA Node. Its purpose is to launch ANSA in no-gui mode with specific settings. ANSA will execute for all the parts of the subsystem the Build process of the Model Browser which contains specific script build actions. These actions perform corrections and improvements of the translated parts' geometry. Finally, the common representation of the parts that were successfully built is saved in DM, as well as the updated subsystem. The parts that will fail to obtain an error free geometry will have to be treated manually by the user in another node.

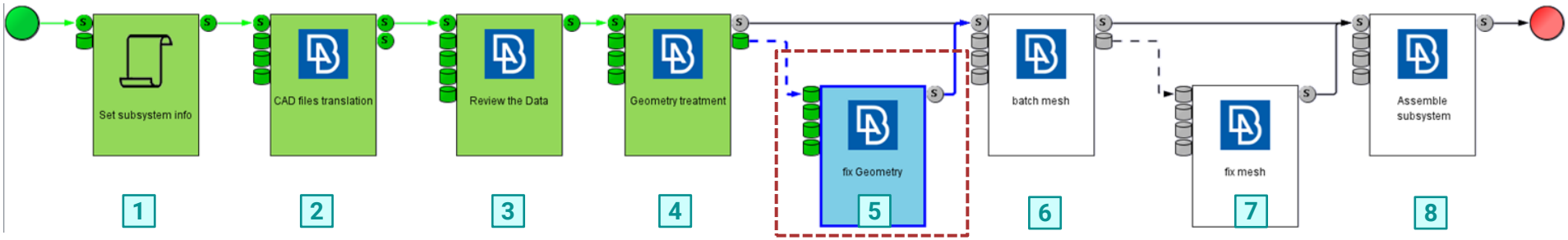
Input data (Input Slots):

- a string value signal passed from the previous node (start_subsystem_built_in)
- the python script file that will be executed by ANSA(BuildSubsystemScripts.py)
- the python script file that will be executed by SPDRM before ANSA is launched and after it has quit (Build_Subsystem_PrePostRun.py)
- the python script used by the build actions of the loaded ANSA Modular Environment Profile.

Output data (Output Slot):

- a signal to continue to the next node, in the form of a string variable
- a folder with files, one file per part, for those parts that have failed the automatic geometry healing actions, if any.

Build Subsystem



Node 5: fix Geometry

This is an Application ANSA Node which will be executed only in the case that failed parts exist from the previous node. Its purpose is to launch ANSA with specific settings and provide instructions which will guide the user through several steps regarding the correction and improvement of the translated parts inside ANSA. The final job in ANSA is to save the common representation of the parts into the DM, as well as the updated subsystem.

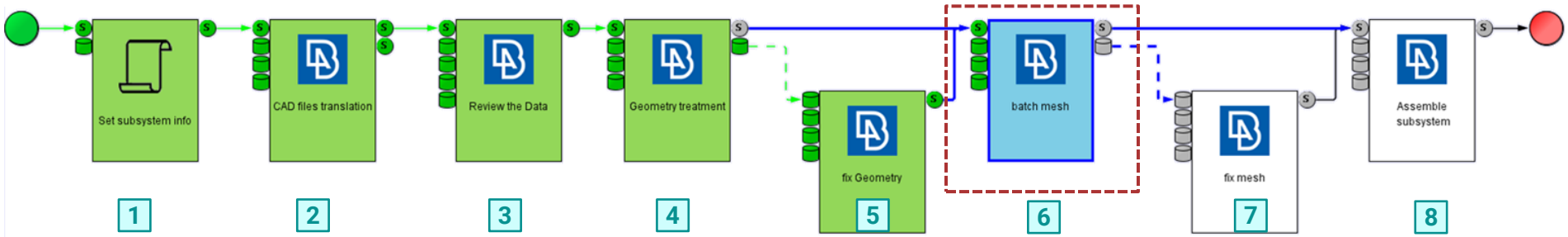
Input data (Input Slots):

- the folder with the files that will be treated manually
- the script file that will be executed by ANSA (BuildSubsystemScripts.py)
- the python script file that will be executed by SPDRM before ANSA is launched and after it has quit (Build_Subsystem_PrePostRun.py)
- the python script used by the build actions of the loaded ANSA Modular Environment Profile.

Output data (Output Slot):

- a signal to continue to the next node, in the form of a string variable

Build Subsystem



Node 6: batch mesh

This is an automatically executed Application ANSA Node. Its purpose is to launch ANSA in nogui mode with specific settings so as to perform the batch-meshing of the parts of the subsystem and save them into the DM. The parts that will not fulfill the meshing quality criteria will have to be treated manually by the user in another node.

Input data (Input Slots):

- the signal to start, in the form of a string variable
- the script file that will be executed by ANSA(BuildSubsystemScripts.py)
- the python script file that will be executed by SPDRM before ANSA is launched and after it has quit (Build_Subsystem_PrePostRun.py)
- the python script used by the build actions of the loaded ANSA Modular Environment Profile.

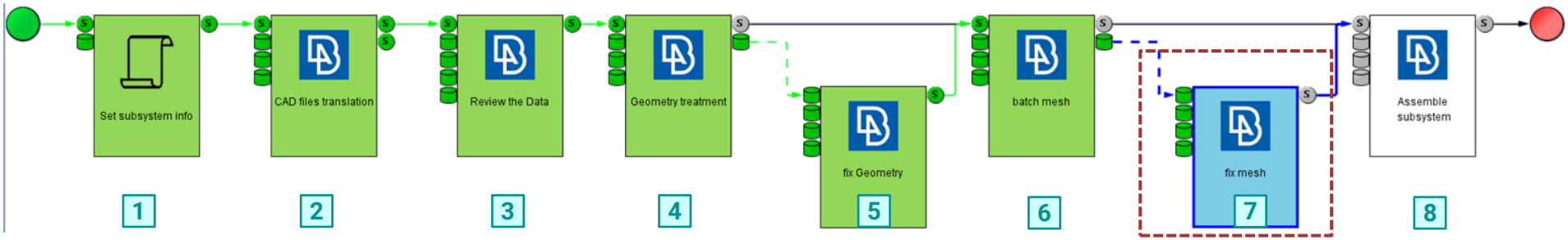
Output data (Output Slot):

- a signal to continue to the next node in case all parts were meshed successfully, in the form of a string variable
- a folder with files, one file per part, for those parts that will need manual improvements.

Output data (Variables of the node):

- the name of a .txt file which contains the outcome of the batch-meshing procedure in the form of “0” or “1” value, for success or failure respectively. It is updated by ANSA.

Build Subsystem



Node 7: fix mesh

This is an Application ANSA Node which will be executed only in the case that parts which need manual mesh improvements exist. Its purpose is to launch ANSA with these parts loaded and let the user improve and then save them into the DM.

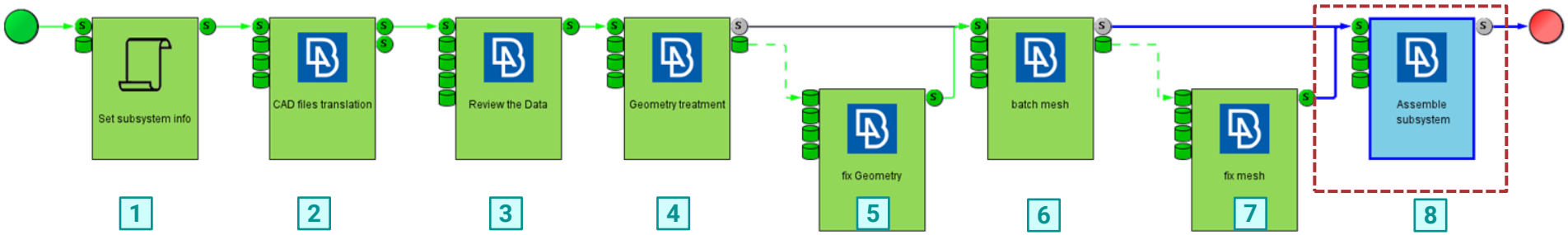
Input data (Input Slots):

- the folder with the files that will be treated manually.
- the script file that will be executed by ANSA(BuildSubsystemScripts.py)
- the python script file that will be executed by SPDRM before ANSA is launched and after it has quit (Build_Subsystem_PrePostRun.py)
- the python script used by the build actions of the loaded ANSA Modular Environment Profile.

Output data (Output Slot):

- a signal to continue to the next node, in the form of a string variable

Build Subsystem



Node 8: Assemble subsystem

This is an Application ANSA Node. Its purpose is to launch ANSA with specific settings and provide also a set of build actions for the subsystem in the Model Browser. These will guide the user through several steps for the creation of LC_Points, Assembly, and checking of the final subsystem. Finally, the subsystem is saved in the DM.

Input data (Input Slots):

- a signal to start, in the form of a string variable
- the script file that will be executed by ANSA(BuildSubsystemScripts.py)
- the python script file that will be executed by SPDRM before ANSA is launched and after it has quit (Build_Subsystem_PrePostRun.py)
- the python script used by the build actions of the loaded ANSA Modular Environment Profile.

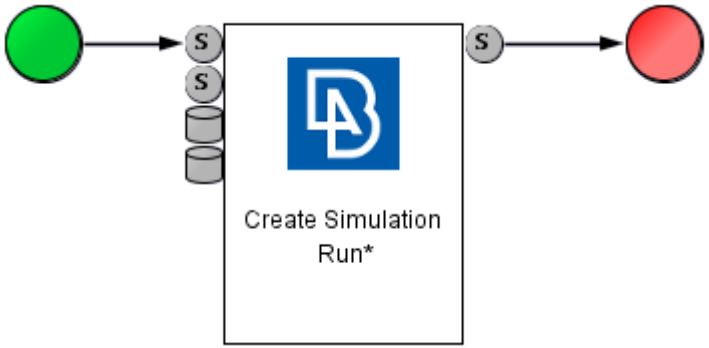
Output data (Output Slot):

- a signal to end the process, in the form of a string variable

Overview of the processes

Create Run

Create Run Workflow



This is a workflow to create a ready-to-run FE-model by creating and saving a Simulation Run based on the subsystem built in the previous workflow of “Build Subsystem” and a predefined Loadcase Header library item.

This workflow consists of a single ANSA Node which launches ANSA, loads the subsystem and then prompts the user to select one of the existing in DM Loadcase Headers. Once the user makes the selection, the Loadcase Header library item is downloaded in the current ANSA session and a script proceeds to the creation of the needed Simulation Model, Loadcase, and Simulation Run. The built-in Build process of the Simulation Run is executed in the Model Browser, and finally the Simulation Run is saved in the DM.

Input data (Input Slots):

- a signal to start, in the form of a string variable
- the handle_id of the selected subsystem, in the form of a string variable
- a python script file that will be executed by ANSA (CreateSimulationRun.py)
- the python script file that will be executed by SPDRM before ANSA is launched and after it has quit.(Create_Run_PrePostRun.py)

Output data (Output Slot):

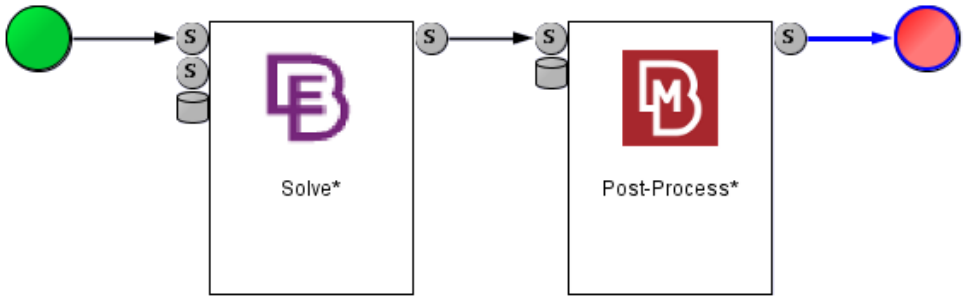
- a signal to continue to the next End node, in the form of a string variable

BETA SIMULATION SOLUTIONS

Overview of the processes

Submit Run

Submit Run



This is a workflow for the solving and post-processing of the Simulation Run that was produced by the workflow “Create Run”. The Epilysis solver and the Metapost Post-processor are used in order to accomplish these tasks.

Solve Node

Input data (Input Slots):
- a signal to start, in the form of a string variable
- the handle_id of the selected Simulation Run, in the form of a string variable
- the python script file that will be executed by SPDRM before ANSA is launched and after it has quit
- (Submit_Run_PrePostRun.py)

Output data (Output Slot):
- a signal to continue to the next Post-Process node, in the form of a string variable

Post-Process Node

Input data (Input Slots):
- a signal to start, in the form of a string variable
- a python script file that will be executed by SPDRM as pre and post run.(Submit_Run_PrePostRun.py)

Output data (Output Slot):
- a signal to continue to the End node, in the form of a string variable

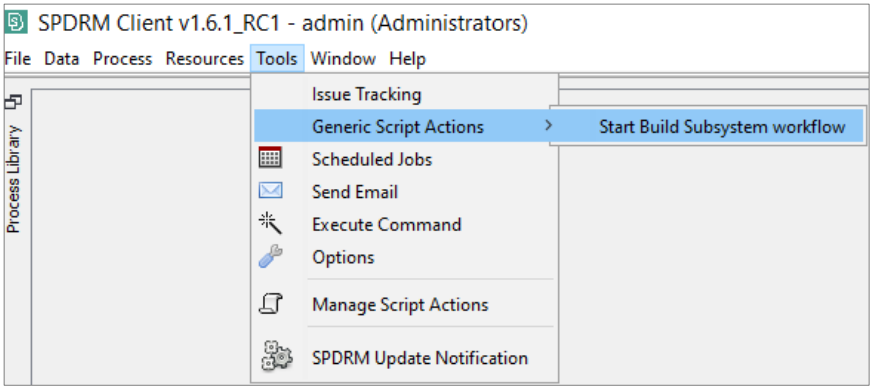
Execution of the processes

Build Subsystem

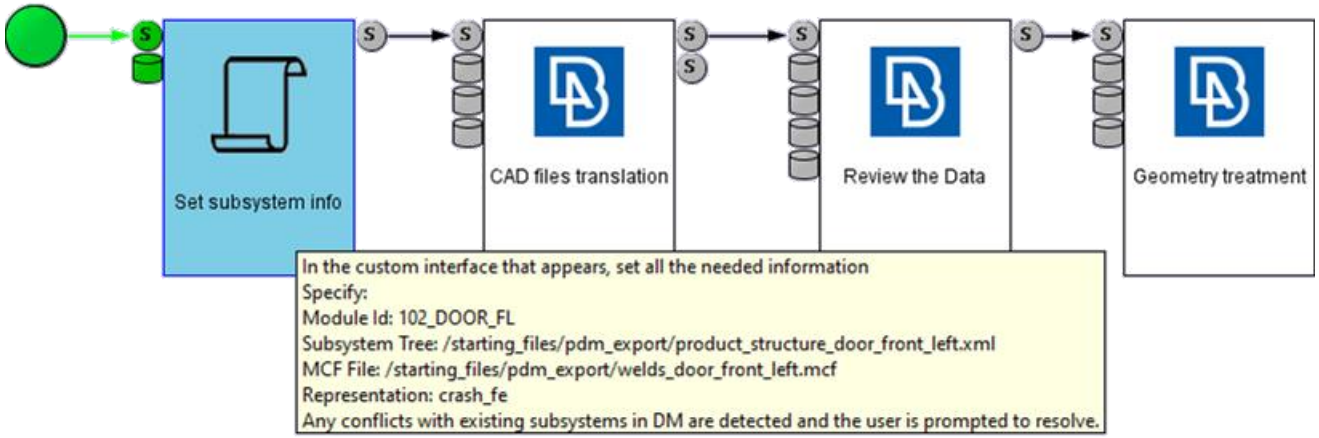
Process Execution - Build Subsystem

This section is a detailed guide throughout the execution of the *Build Subsystem* workflow.

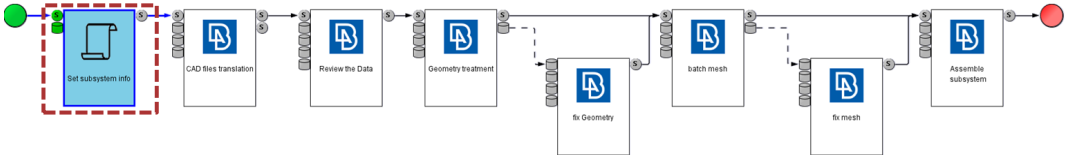
1 In order to initiate a new instance of the workflow go to **Tools>Generic Script Actions>Start Build Subsystem Workflow**.



2 The process is instantiated in a new window and the execution of the first node ("Set subsystem info") starts automatically. On hover of the mouse on the nodes of the process, a useful tooltip with a description of the steps to be followed is displayed.



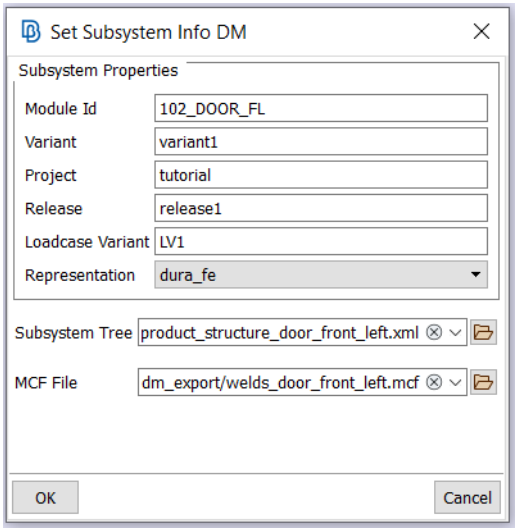
Build Subsystem – Set subsystem info



1 A custom GUI is launched upon execution of node which prompts the user to select the properties of the subsystem. Fill in the values as shown in the image and press OK.

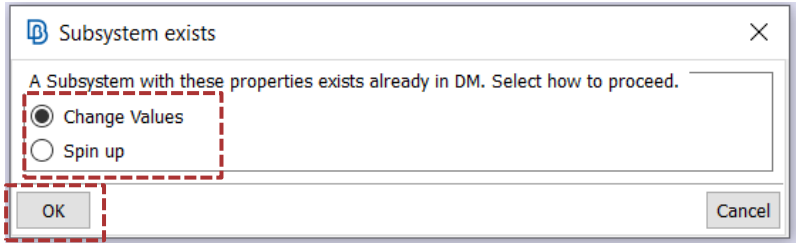
The files to be used are:

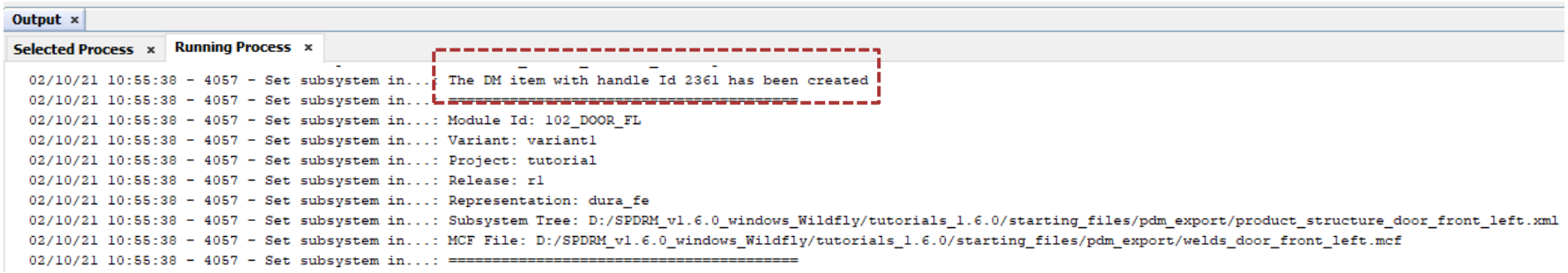
Subsystem Tree:
../starting_files/pdm_export/product_structure_door_front_left.xml
MCF File:
../starting_files/pdm_export/welds_door_front_left.mcf



2 In case a subsystem with the same properties exists already in DM, a message informs the user accordingly. The user should either return to the previous GUI so as to change properties, or proceed to create a new iteration of the existing subsystem.

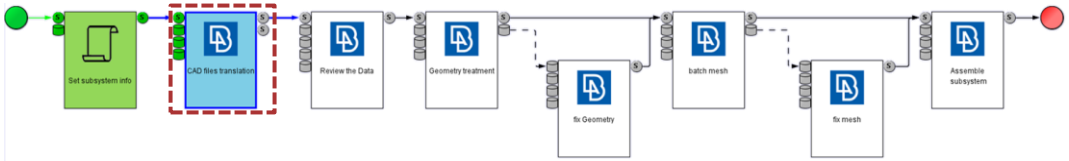
Upon confirmation a subsystem is created as stated in the output window and the process proceeds to the next node.





BETA SIMULATION SOLUTIONS

Build Subsystem – CAD files translation



1 ANSA is launched in no-gui mode automatically and all the CAD files are translated.

2 The representation of “translated_cad” is saved for all the parts in DM.

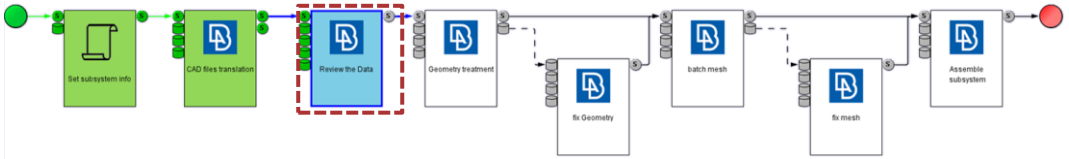
Output

Selected Process × Running Process ×

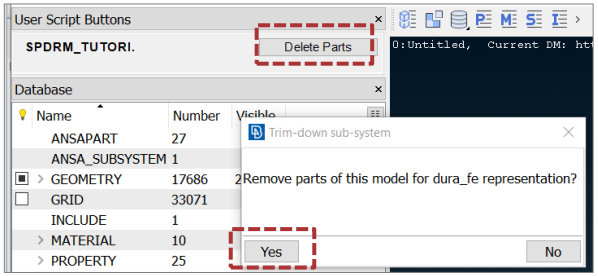
```
02/10/21 10:57:24 - 4059 - CAD files transl...: 100% [##################################################]
02/10/21 10:57:27 - 4059 - CAD files transl...: 100% [##################################################]
02/10/21 10:57:28 - 4059 - CAD files transl...: Updating Part with Name '2000367_004_DOOR_PANEL_UPPER_REINFORCFetching DM Item ... DMObject  with server id  2361
02/10/21 10:57:28 - 4059 - CAD files transl...: *************  Downloading DM Item  ***************
02/10/21 10:57:28 - 4059 - CAD files transl...: Processing part 2000363_001_DOOR_PANEL_METAL_BRACKET
02/10/21 10:57:28 - 4059 - CAD files transl...: Processing part 607090_002_MIRROR_REINFORCEMENT
02/10/21 10:57:28 - 4059 - CAD files transl...: Processing part 2000364_006_DOOR_PANEL_UPPER_FOAM
02/10/21 10:57:28 - 4059 - CAD files transl...: Processing part 608060_001_LOWER_HINGE_IN
02/10/21 10:57:28 - 4059 - CAD files transl...: Processing part 607020_007_LOCK_REINFORCEMENT
02/10/21 10:57:28 - 4059 - CAD files transl...: Processing part 607010_007_SIDE_IMPACT_BAR
02/10/21 10:57:28 - 4059 - CAD files transl...: Processing part 2000372_003_DOOR_BUTTON_BRACKET
02/10/21 10:57:28 - 4059 - CAD files transl...: Processing part 607050_003_REINFORCEMENT_BAR
02/10/21 10:57:28 - 4059 - CAD files transl...: Processing part 607041_003_EXTERIOR_PANEL_LT
02/10/21 10:57:28 - 4059 - CAD files transl...: Processing part 2000366_001_DETAILED_DOOR_PANEL
02/10/21 10:57:28 - 4059 - CAD files transl...: Processing part 2000367_004_DOOR_PANEL_UPPER_REINFORCEMENT
02/10/21 10:57:28 - 4059 - CAD files transl...: Processing part 607060_001_LOWER_HINGE_OUT
02/10/21 10:57:28 - 4059 - CAD files transl...: Processing part 607080_002_BACK_FRAME
02/10/21 10:57:28 - 4059 - CAD files transl...: Processing part 2000365_003_DOOR_PANEL_PAPER_HOLDER
02/10/21 10:57:28 - 4059 - CAD files transl...: Processing part 2000375_003_DOOR_ARMREST_SHELL
02/10/21 10:57:28 - 4059 - CAD files transl...: Processing part 607001_004_CASE_LT
02/10/21 10:57:28 - 4059 - CAD files transl...: Processing part 608070_001_UPPER_HINGE_IN
02/10/21 10:57:28 - 4059 - CAD files transl...: Processing part 2000374_003_DOOR_ARMREST
02/10/21 10:57:28 - 4059 - CAD files transl...: Processing part 607030_001_UPPER_FRAME
02/10/21 10:57:28 - 4059 - CAD files transl...: Processing part 607101_006_REINFORCEMENT_BELT_LT
02/10/21 10:57:28 - 4059 - CAD files transl...: Processing part 607070_001_UPPER_HINGE_OUT
02/10/21 10:57:28 - 4059 - CAD files transl...: Processing part 2000379_001_DOOR_BRACKET
02/10/21 10:57:28 - 4059 - CAD files transl...: ***************************************
02/10/21 10:57:29 - 4059 - CAD files transl...: Updating Part with Name '2000363_001_DOOR_PANEL_METAL_BRACKET' and Module Id '2000363' (22/22).
02/10/21 10:57:29 - 4059 - CAD files transl...:
02/10/21 10:57:29 - 4059 - CAD files transl...: File creation info :
02/10/21 10:57:29 - 4059 - CAD files transl...: HOST : saos.localdomain
02/10/21 10:57:29 - 4059 - CAD files transl...: USER : zeta
02/10/21 10:57:29 - 4059 - CAD files transl...: DATE :27-SEP-2017  16:18:19
02/10/21 10:57:29 - 4059 - CAD files transl...:
02/10/21 10:57:29 - 4059 - CAD files transl...: LAST EDIT :
02/10/21 10:57:29 - 4059 - CAD files transl...: HOST : BETAINDIA-L-035
02/10/21 10:57:29 - 4059 - CAD files transl...: USER : prabhat.s
02/10/21 10:57:29 - 4059 - CAD files transl...: DATE :20-SEP-2021  13:41:43
02/10/21 10:57:29 - 4059 - CAD files transl...:
02/10/21 10:57:29 - 4059 - CAD files transl...: START: CAD to ANSA: 02-OCT-2021 10:57:24
02/10/21 10:57:29 - 4059 - CAD files transl...: ---------------------------------------
02/10/21 10:57:29 - 4059 - CAD files transl...: START: CAD to ANSA
02/10/21 10:57:29 - 4059 - CAD files transl...: END: CAD to ANSA: 02-OCT-2021 10:57:24
02/10/21 10:57:29 - 4059 - CAD files transl...: Report directory: C:/Users/prabhat.s/CAD_to_ANSA_logs/20211002_105724/
02/10/21 10:57:29 - 4059 - CAD files transl...: Total Parts: 26
02/10/21 10:57:29 - 4059 - CAD files transl...: No treatment: 4
02/10/21 10:57:29 - 4059 - CAD files transl...: Skip: 22
02/10/21 10:57:29 - 4059 - CAD files transl...: END: CAD to ANSA
02/10/21 10:57:29 - 4059 - CAD files transl...:
02/10/21 10:57:29 - 4059 - CAD files transl...: ==================Save Subsystems in DM==================
02/10/21 10:57:29 - 4059 - CAD files transl...: Definition attributes of '102_DOOR_FL_tutorial_rl_variant1_dura_fe_LV1_001' are filled.
02/10/21 10:57:29 - 4059 - CAD files transl...: Save Subsystems in DM: Successfully completed
02/10/21 10:57:29 - 4059 - CAD files transl...: ================End of Save Subsystems in DM================
02/10/21 10:57:29 - 4059 - CAD files transl...:  ANSA Quit.
###Executing Post-Run Script
02/10/21 10:57:29 - 4059 - CAD files transl...: D:\SPDRM_v1.6.0_windows_Wildfly\vaults\vault1\NodeExec\297c2d55-7287-41d8-babb-cb14ec2decd7>echo off
02/10/21 10:57:31 - 4059 - CAD files transl...: Reading subsystem_tree_attributes.txt file...
```

BETA SIMULATION SOLUTIONS

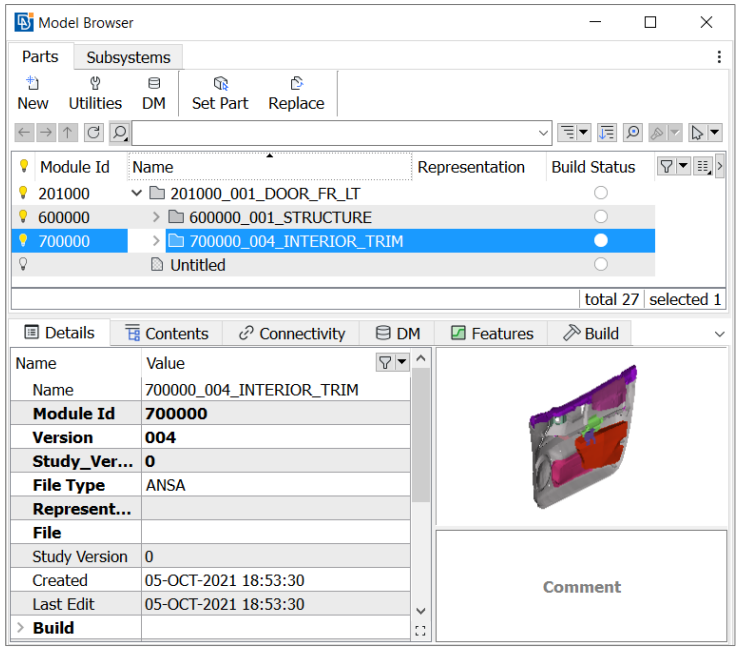
Build Subsystem – Review the Data

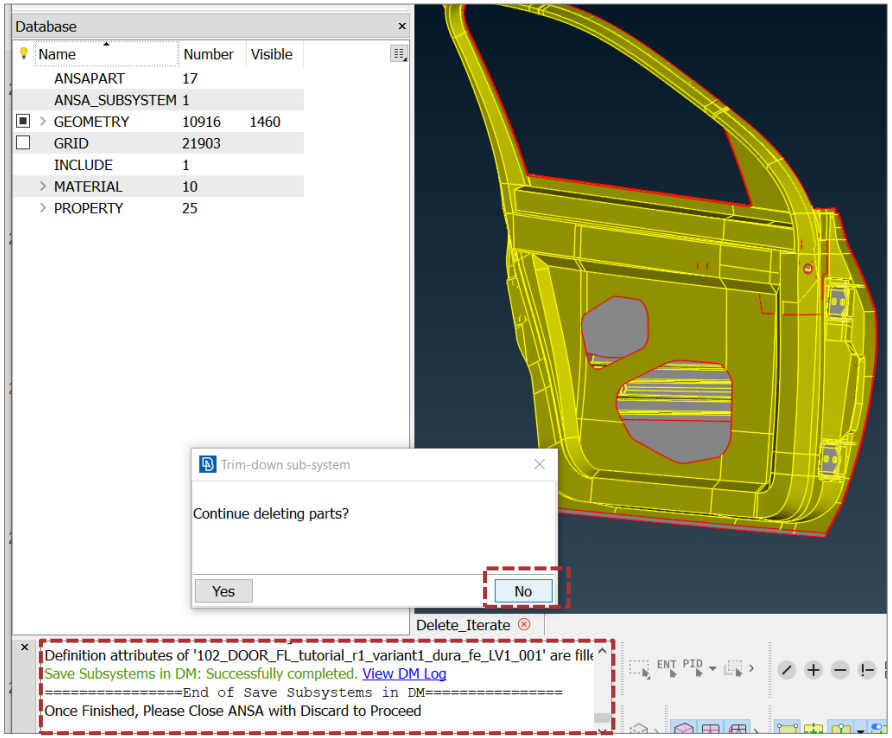


1 Upon execution of the node, ANSA is launched and executes a predefined script which allows the user to optionally delete parts that are not needed for the particular representation. The same script is also available through the **Delete Parts** button, in case the user exits the functionality.



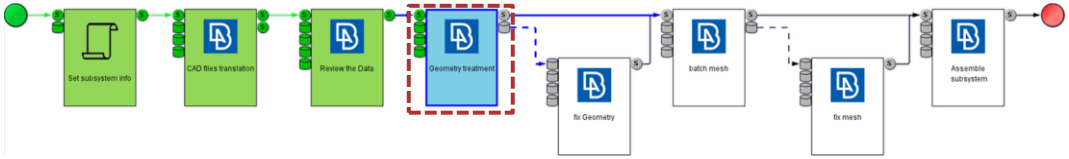
2 Open the Model Browser to assist you. Select the group INTERIOR_TRIM and isolate it in the graphics area. Close the Model Browser, select the parts and confirm with middle mouse button. Upon confirmation ANSA deletes the selected parts. The user can continue to delete parts in case the selection was not complete. Once finished the user should close ANSA with the Discard option.





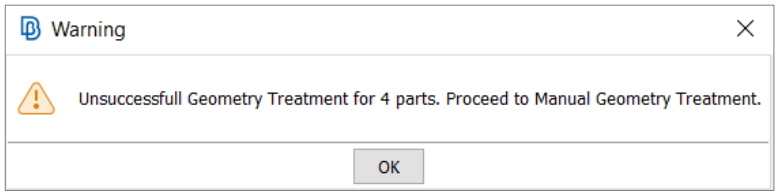
3 Finally, the subsystem is automatically saved in DM.

Build Subsystem – Geometry treatment



1. ANSA is launched in no-gui mode and geometry treatment is performed on all the parts.

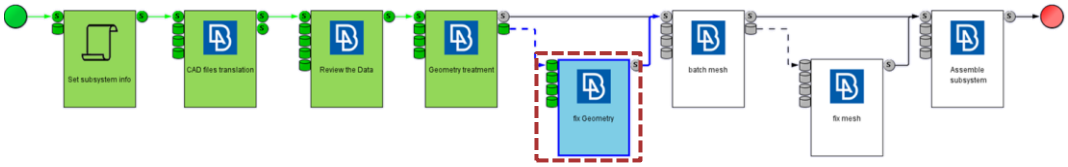
2. In case of unsuccessful treatment of parts the user is informed through a warning message and the list of files output slot is populated to be solved by the fix Geometry node.



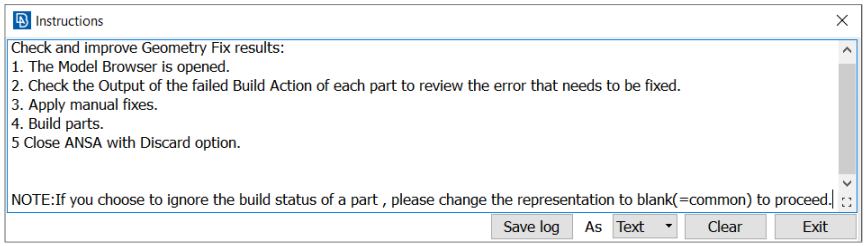

```
04/10/21 13:16:36 - 4568 - Geometry treatme...: Processing part 607001_004_CASE_LT
04/10/21 13:16:36 - 4568 - Geometry treatme...: Processing part 607050_003_REINFORCEMENT_BAR
04/10/21 13:16:36 - 4568 - Geometry treatme...: Processing part 608060_001_LOWER_HINGE_IN
04/10/21 13:16:36 - 4568 - Geometry treatme...: Processing part 607020_007_LOCK_REINFORCEMENT
04/10/21 13:16:36 - 4568 - Geometry treatme...: Starting BuildProcess on topmost  201000_001_DOOR_FR_LT
04/10/21 13:16:36 - 4568 - Geometry treatme...: ================================================================
04/10/21 13:16:53 - 4568 - Geometry treatme...: Saved file geom_failed_parts/608060_001_LOWER_HINGE_IN.ansa
04/10/21 13:16:53 - 4568 - Geometry treatme...: Saved file geom_failed_parts/608070_001_UPPER_HINGE_IN.ansa
04/10/21 13:16:53 - 4568 - Geometry treatme...: Saved file geom_failed_parts/607060_001_LOWER_HINGE_OUT.ansa
04/10/21 13:16:53 - 4568 - Geometry treatme...: Saved file geom_failed_parts/607070_001_UPPER_HINGE_OUT.ansa
04/10/21 13:16:53 - 4568 - Geometry treatme...: ================================================================
04/10/21 13:16:53 - 4568 - Geometry treatme...: Build Actions Completed
04/10/21 13:16:54 - 4568 - Geometry treatme...: Representation 'common' for Part '607001_004_CASE_LT' with Module Id '607001' has been saved in DM successfully.
04/10/21 13:16:54 - 4568 - Geometry treatme...: Build ANSAPART status OK
04/10/21 13:16:54 - 4568 - Geometry treatme...: Build ANSAGROUP: 700000_004_INTERIOR_TRIM
04/10/21 13:16:54 - 4568 - Geometry treatme...: Build ANSAGROUP status OK
04/10/21 13:16:54 - 4568 - Geometry treatme...: Please Wait. Saving Database
04/10/21 13:16:54 - 4568 - Geometry treatme...: Save Completed
04/10/21 13:16:54 - 4568 - Geometry treatme...: Please Wait. Saving Database
04/10/21 13:16:54 - 4568 - Geometry treatme...: Save Completed
04/10/21 13:16:54 - 4568 - Geometry treatme...: Please Wait. Saving Database
04/10/21 13:16:54 - 4568 - Geometry treatme...: Save Completed
04/10/21 13:16:54 - 4568 - Geometry treatme...: Please Wait. Saving Database
04/10/21 13:16:54 - 4568 - Geometry treatme...: Save Completed
04/10/21 13:16:54 - 4568 - Geometry treatme...:  ANSA Quit.
###Executing Post-Run Script
04/10/21 13:16:55 - 4568 - Geometry treatme...: D:\SPDRM_v1.6.0_windows_Wildfly\vaults\vault1\NodeExec\23500cbf-2031-40be-bca9-0daee5c5ae4c>echo off
04/10/21 13:16:56 - 4568 - Geometry treatme...: Unsuccessfull Geometry Treatment for 4 parts. Proceed to Manual Geometry Treatment.
```

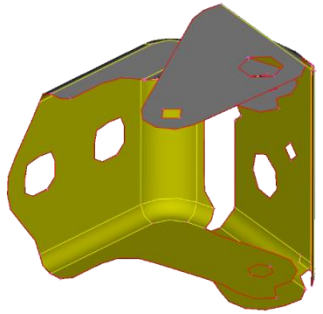
BETA SIMULATION SOLUTIONS

Build Subsystem – fix Geometry

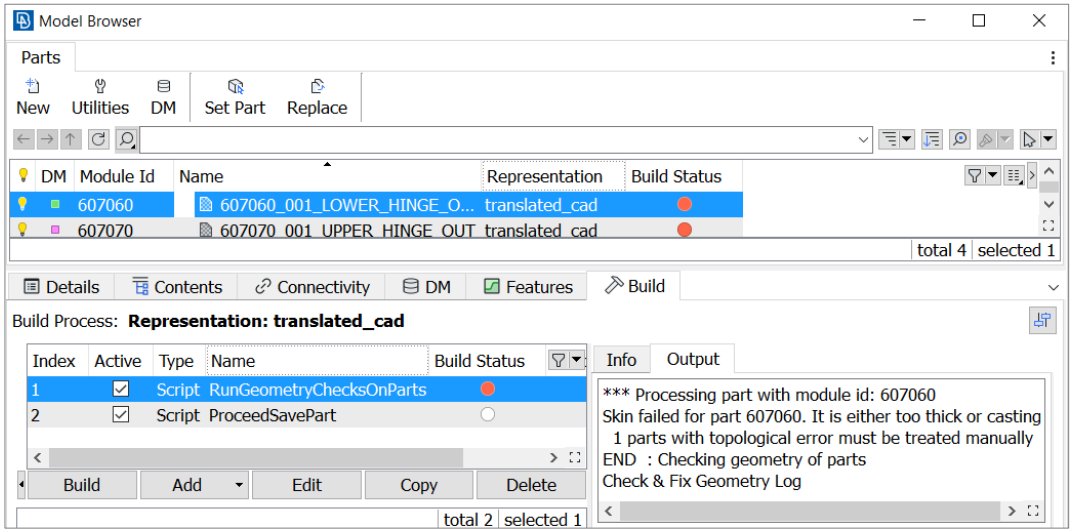


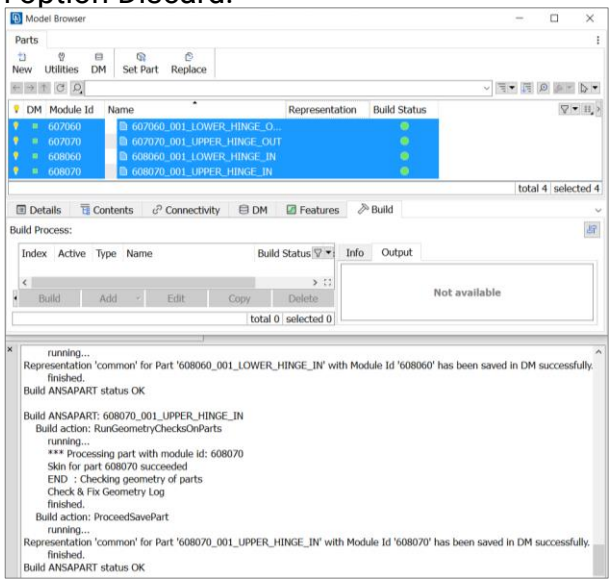
1 ANSA is launched upon execution of node with the Instructions window open. The parts that failed the geometry treatment in previous node are loaded.





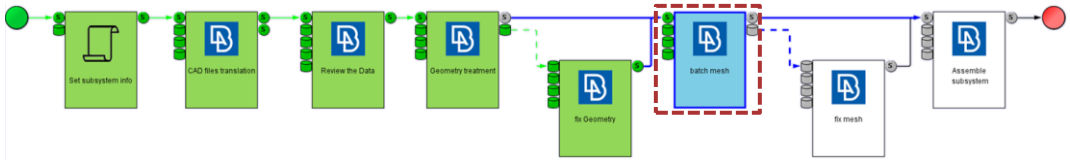
2 The Model Browser is open with the Parts and Build tabs active. The pre-defined Build Process for the parts that has been read through the Modular Environment Profile is failed and thus the Build Status of parts appear as Error. The user could check the details in the *Output* tab of each build action and proceed to remove the errors for each part. Alternatively, open the file **../starting_files/fixed_parts_translated_cad.ansa** and Build again all the parts, using the **Build** option from their context menu. The parts must obtain a green Build Status. When finished, quit ANSA with option Discard.





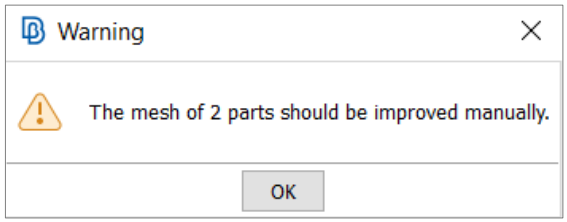
BETA SIMULATION SOLUTIONS

Build Subsystem – batch mesh



1 ANSA is launched in no-gui mode and batch-mesh is performed on all the parts based on different meshing scenarios.

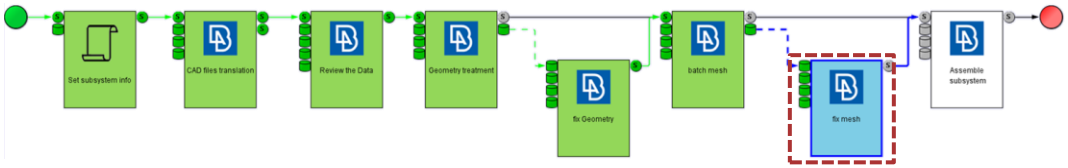
2 In case of unsuccessful meshing of parts a prompt appears and the list of files output slot is populated to be solved by the fix mesh node.



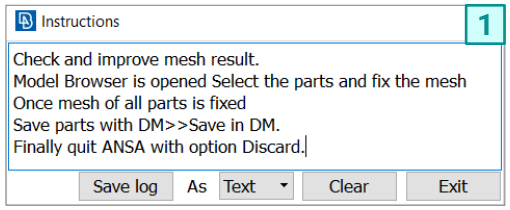

```
Saved file batch_mesh_failed_parts/607070_001_UPPER_HINGE_OUT.ansa
Saved file batch_mesh_failed_parts/607001_004_CASE_LT.ansa
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------

START: Creating durability representation for parts


  11 parts with no unmeshed macros and quality violations should be saved as "durability" in DM
  11 parts were saved successfully as "durability" in DM
  2 parts with topological or attribute error were saved in "batch_mesh_failed_parts"


END  : Creating durability representation for parts
Batch Mesh log file: C:/Users/prabhat.s/.BETA/ANSA/version_22.1.0/ANSA_BatchMesh.log
Running Meshing Scenario durability
 All macros meshed .
 All macros meshed .
 All macros meshed .
Meshing scenario: durability finished - Execution time :  39 sec
Batch Mesh log file: C:/Users/prabhat.s/.BETA/ANSA/version_22.1.0/ANSA_BatchMesh.log
Running Meshing Scenario durability
 All macros meshed .
 All macros meshed .
 All macros meshed .
Meshing scenario: durability finished - Execution time :  39 sec
Report saved in D:/SPDRM_v1.6.0_windows_Wildfly/vaults/vault1/NodeExec/7e07c0f8-e7f7-40a1-8dle-efc683lad5ef/bm_statistics.txt.
Representation 'durability' for Part '608060_001_LOWER_HINGE_IN' with Module Id '608060' has been saved in DM successfully.
Representation 'durability' for Part '608070_001_UPPER_HINGE_IN' with Module Id '608070' has been saved in DM successfully.
Representation 'durability' for Part '607060_001_LOWER_HINGE_OUT' with Module Id '607060' has been saved in DM successfully.
Please Wait. Saving Database
Save Completed
Representation 'durability' for Part '607041_003_EXTERIOR_PANEL_LT' with Module Id '607041' has been saved in DM successfully.
Representation 'durability' for Part '607101_006_REINFORCEMENT_BELT_LT' with Module Id '607101' has been saved in DM successfully.
Representation 'durability' for Part '607050_003_REINFORCEMENT_BAR' with Module Id '607050' has been saved in DM successfully.
Representation 'durability' for Part '607080_002_BACK_FRAME' with Module Id '607080' has been saved in DM successfully.
Representation 'durability' for Part '607030_001_UPPER_FRAME' with Module Id '607030' has been saved in DM successfully.
Representation 'durability' for Part '607010_007_SIDE_IMPACT_BAR' with Module Id '607010' has been saved in DM successfully.
Representation 'durability' for Part '607020_007_LOCK_REINFORCEMENT' with Module Id '607020' has been saved in DM successfully.
Representation 'durability' for Part '607090_002_MIRROR_REINFORCEMENT' with Module Id '607090' has been saved in DM successfully.
Please Wait. Saving Database
Save Completed
 ANSA Quit.
```

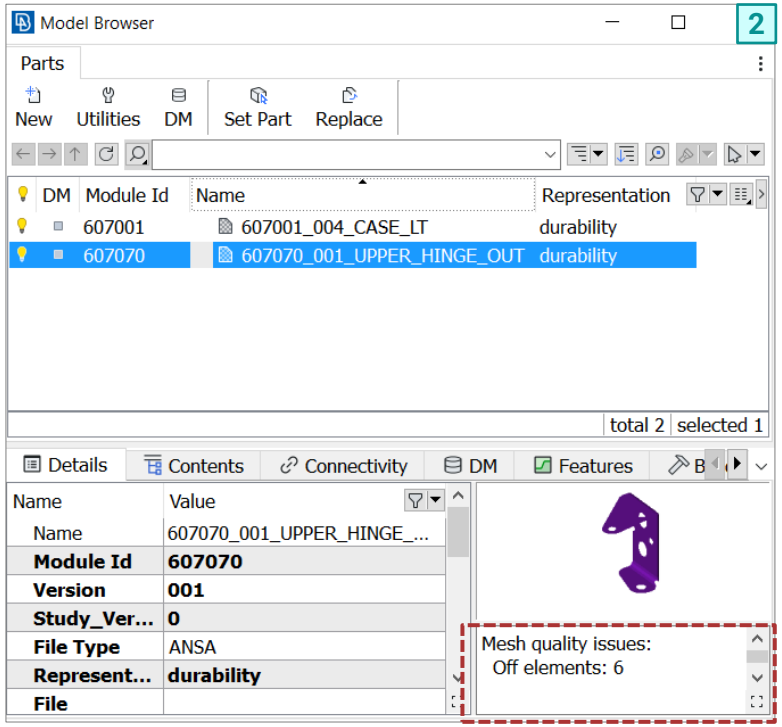
BETA SIMULATION SOLUTIONS

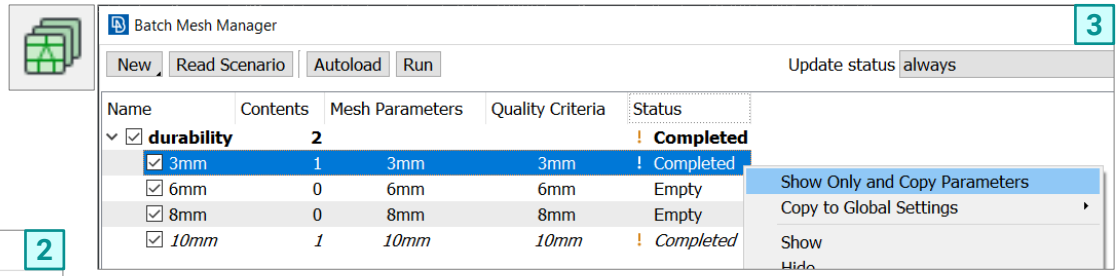
Build Subsystem – fix mesh

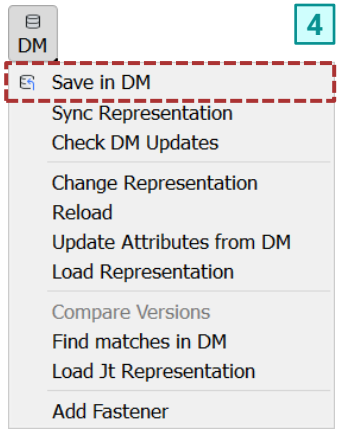


1. ANSA is launched upon execution of node with the Instructions window open and the subsystem loaded.
2. In the comment area of the Model Browser a useful note has been recorded referring to the problems of each part.
3. Copy parameters from the batch-mesh manager and proceed to corrections.
4. When finished save the durability representation of the parts through **DM>Save in DM** option.

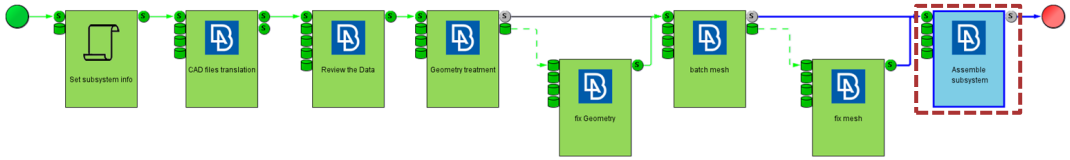




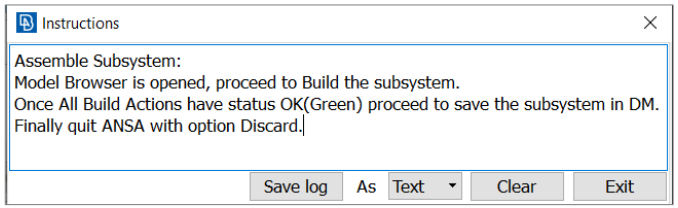


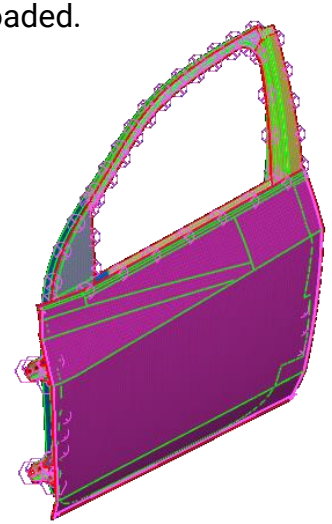


Build Subsystem – Assemble Subsystem

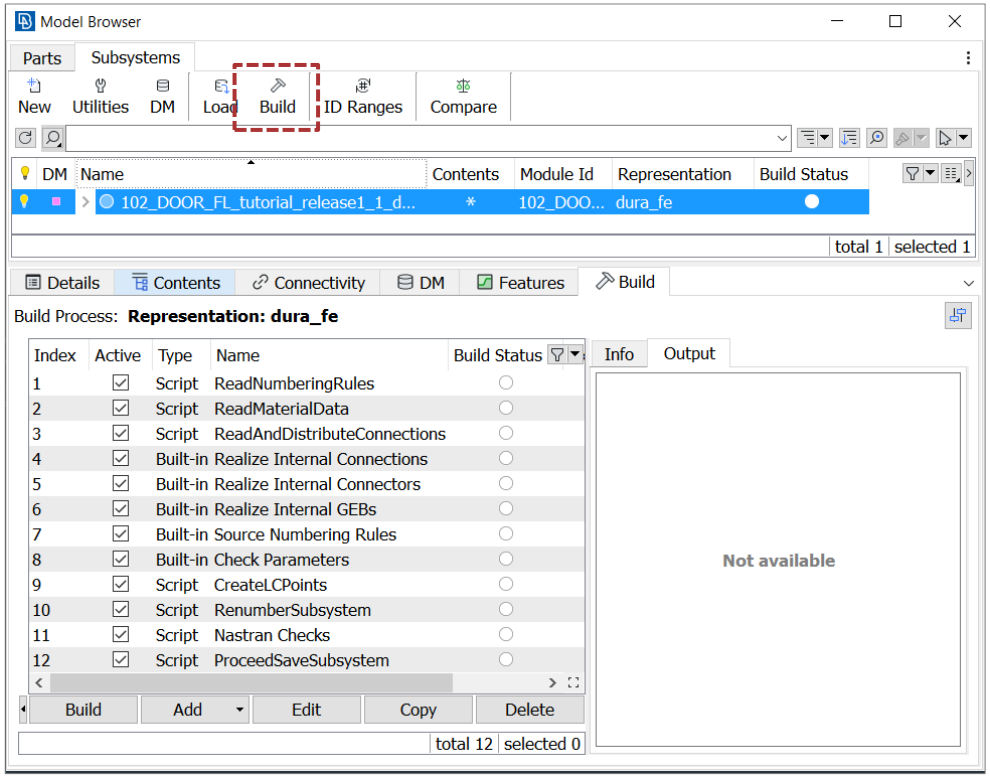


1 ANSA is launched upon execution of node with the Instructions window open and the door subsystem loaded.

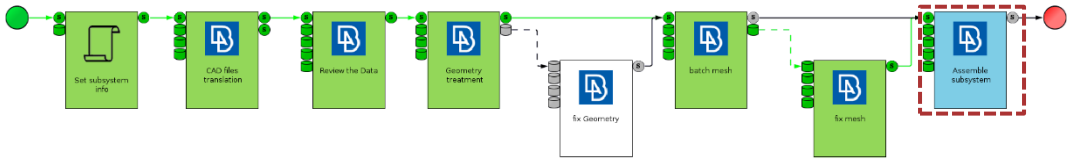




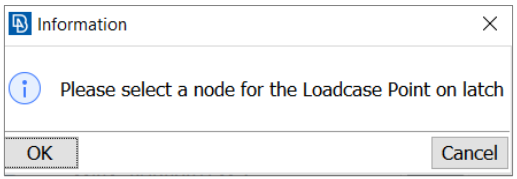
2 The Model Browser is open with the Subsystems and Build tabs active. A pre-defined Build Process for the subsystem has been read through the Modular Environment Profile. Press the **Build** button.

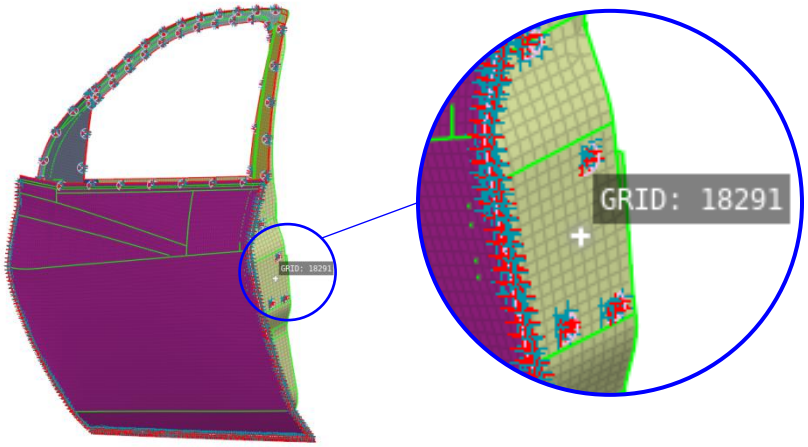


Build Subsystem - Assemble subsystem

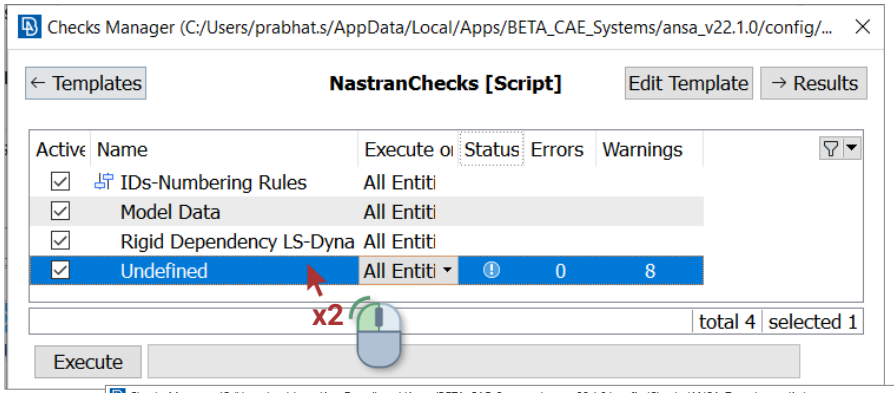


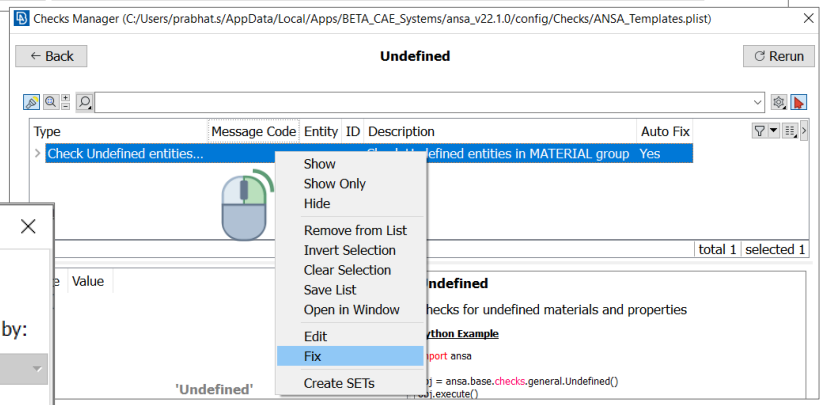
3 The Build Actions are executed sequentially one after another. Once the **CreateLCPoints** action is executed, the user is prompted to select a node of the door subsystem for the creation of an LC_POINT. This LC_POINT is to be used for the application of forces during the Simulation Run set-up at a later stage. Press **OK** and proceed to select a node near the latch of the door (grid id might differ). Confirm the selection with middle mouse button.

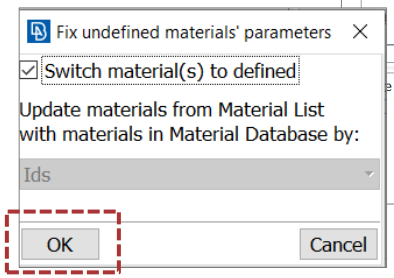




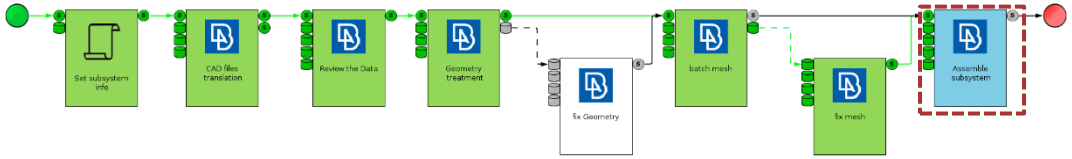
4 ANSA proceeds to the execution of rest actions. In the **Nastran Checks** build action undefined materials are detected. Double mouse click on the check to view the errors and proceed to corrections through the option **Fix** of the context menu.



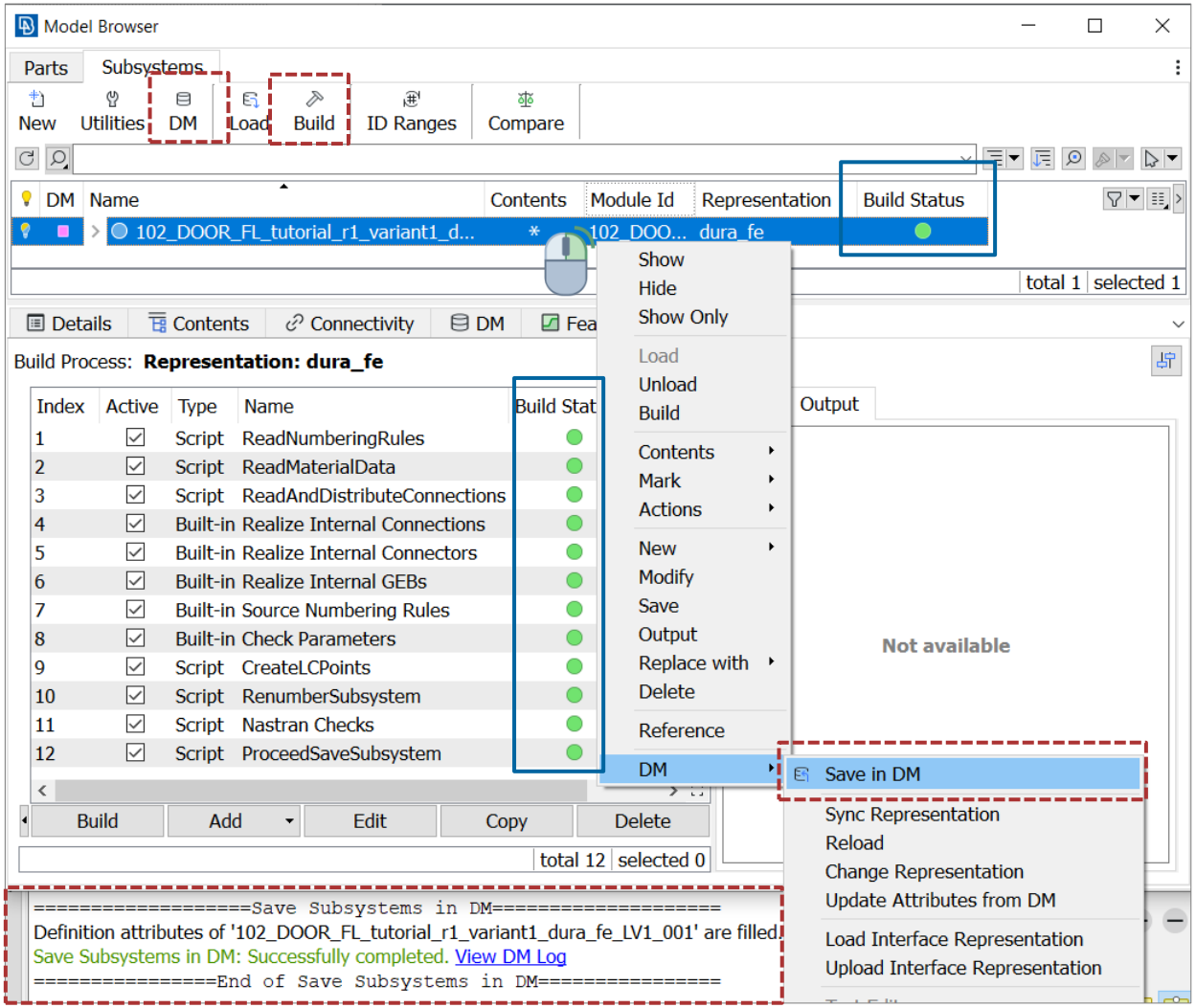




Build Subsystem - Assemble subsystem



5 Press again the **Build** button for the subsystem. The Build Actions are completed and the Subsystem acquires a green Build Status. Save the Subsystem in DM and quit ANSA with **Discard** option.



BETA SIMULATION SOLUTIONS

Execution of the processes

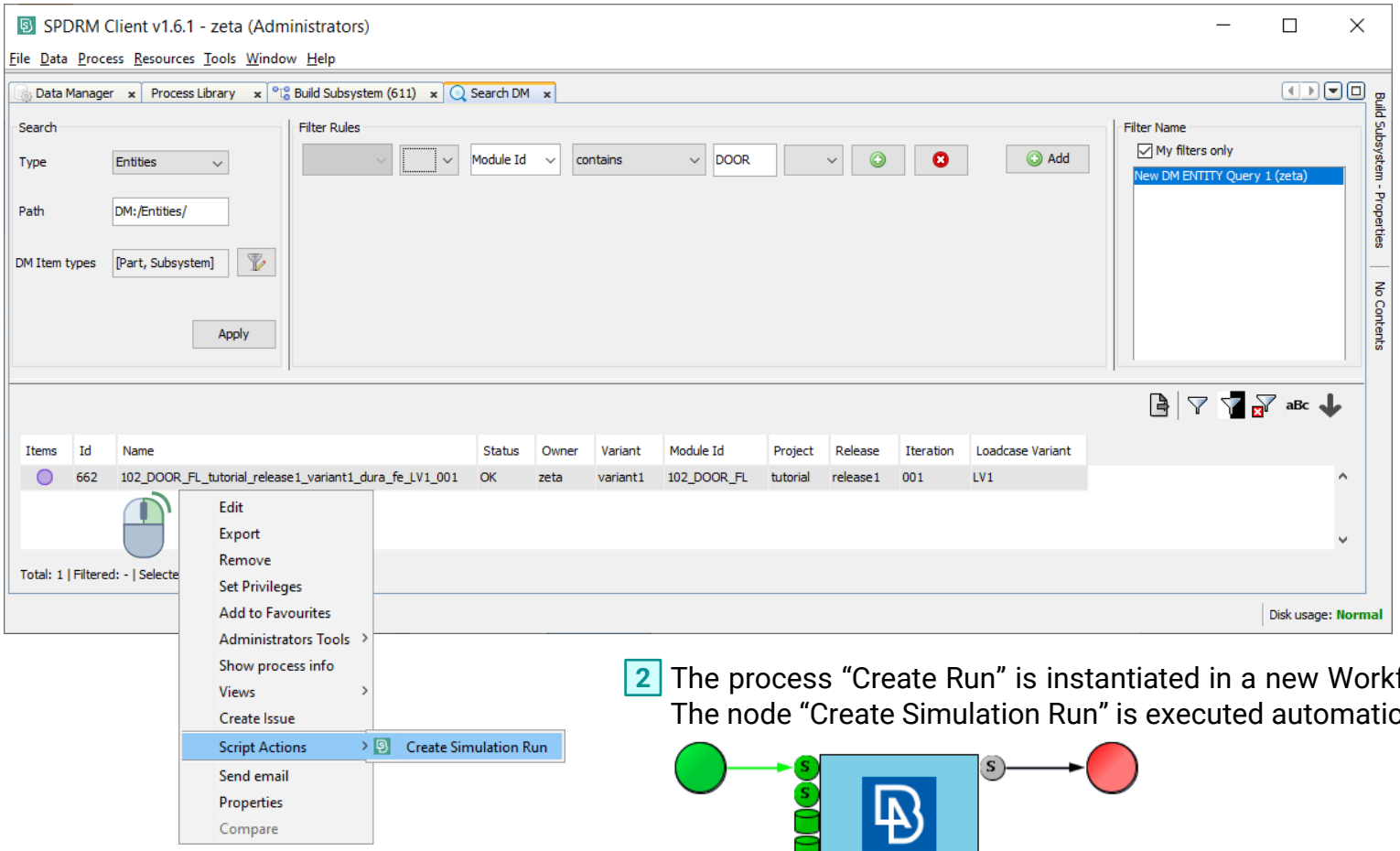
Create Run

BETA
SIMULATION SOLUTIONS

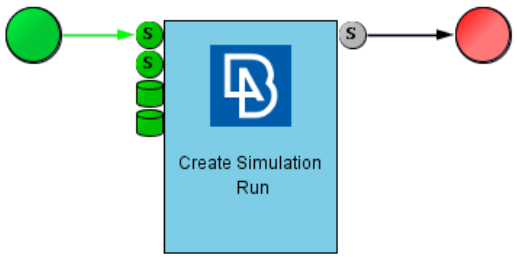
Process Execution – Create Run

This section is a detailed guide throughout the execution of the Create Run workflow.

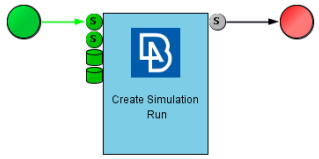
1 This workflow is executed on the subsystem that is produced by the Build Subsystem. Go to the **Search > in DM** window and find the previously created subsystem using proper queries. From its context window activate the option **Script Actions > Create Simulation Run**.



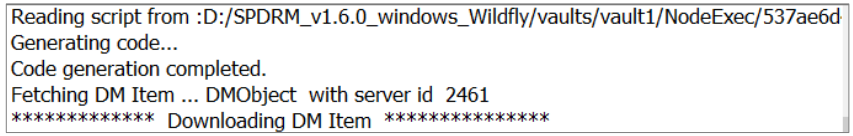
2 The process “Create Run” is instantiated in a new Workflow scene window. The node “Create Simulation Run” is executed automatically.



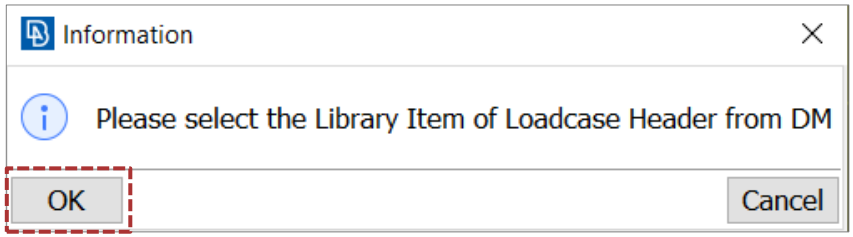
Process Execution – Create Run



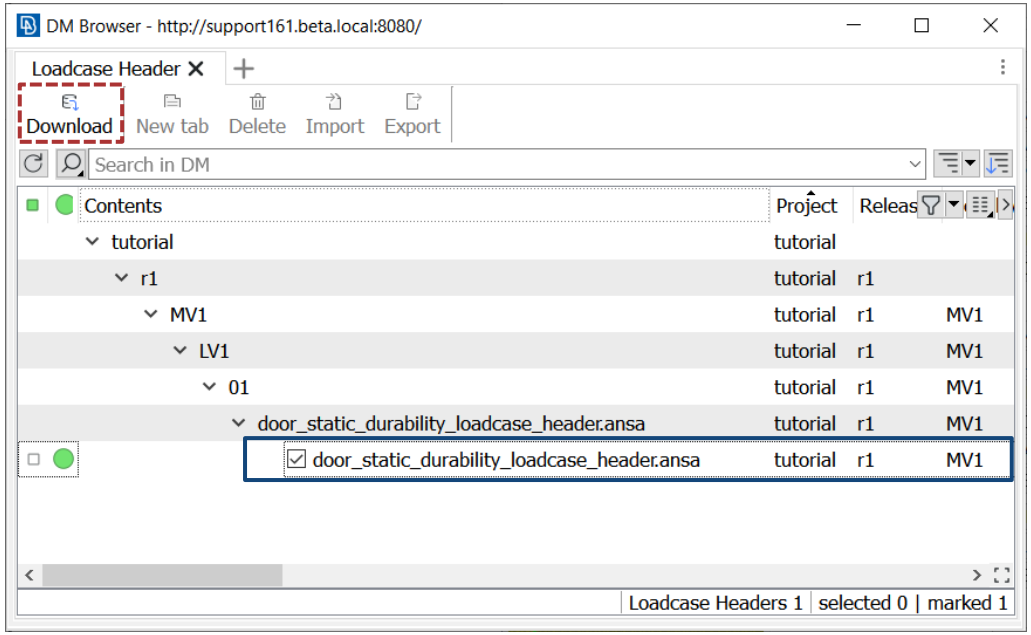
1 ANSA is launched with the selected subsystem loaded.



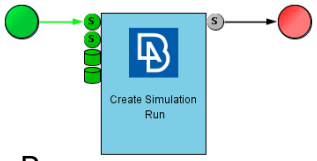
2 The user is prompted to select the Loadcase Header library item. Press **OK** to proceed.



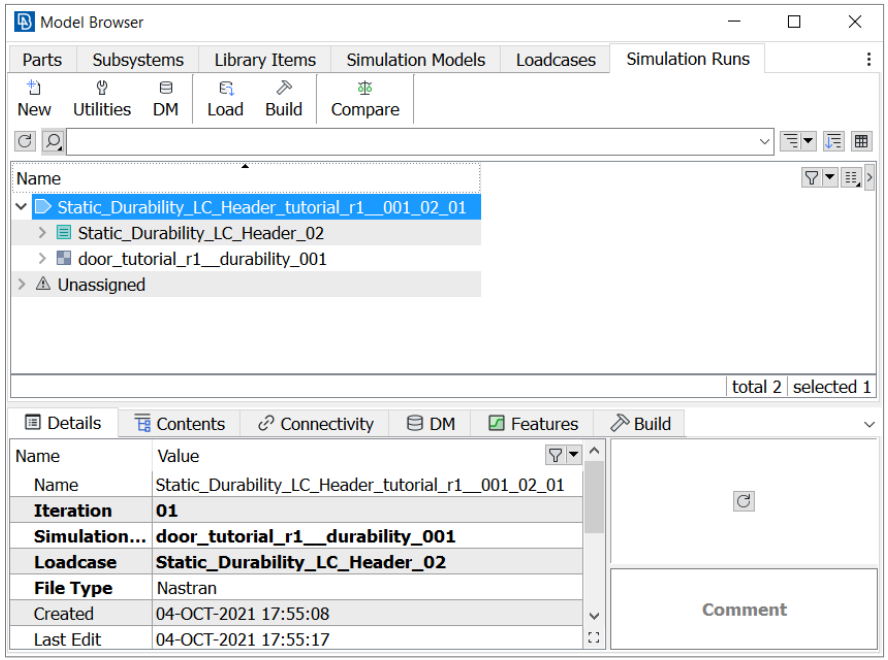
3 The DM Browser opens. Select the Static Durability Loadcase Header and press the **Download** button.



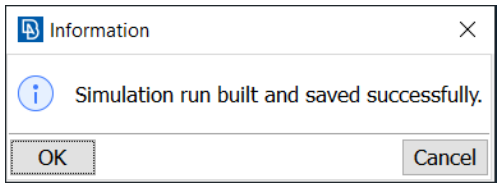
Process Execution - Create Run

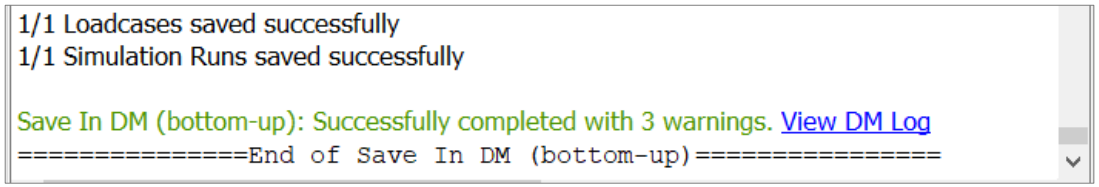


5 The script execution proceeds with the creation of a Simulation Model, a Loadcase and a Simulation Run

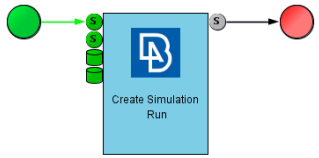


6 Upon completion a window informs the user that the Simulation Run has been built and saved in DM.





Process Execution - Create Run



In the Data Manager of SPDRM, the newly created DM Items are attached in the Structure.

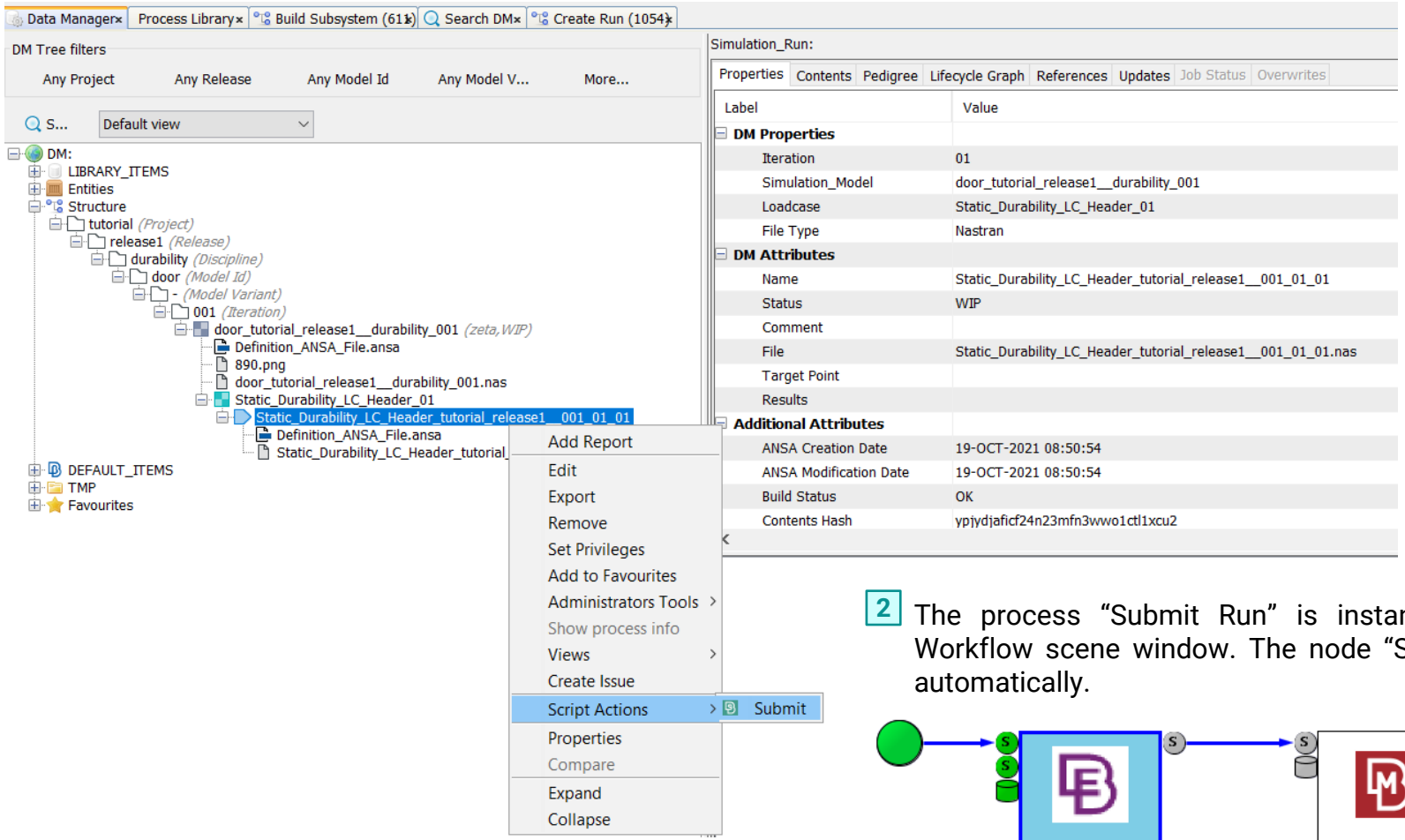
Execution of the processes

Submit Run

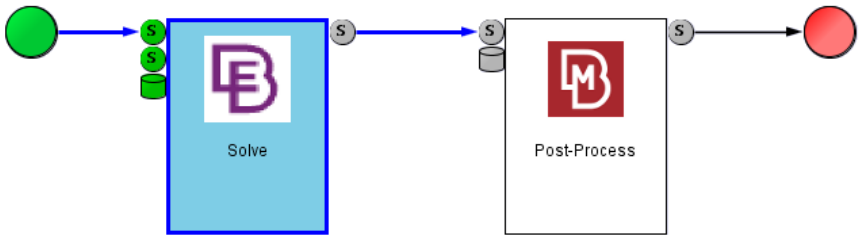
Process Execution – Submit Run

This section is a detailed guide throughout the execution of the Submit Run workflow.

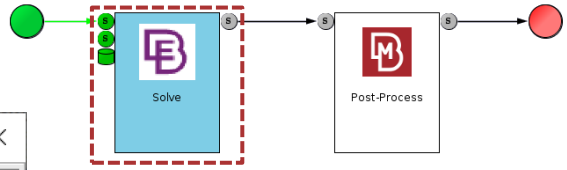
1 This workflow is executed on the Simulation Run that is produced by the Create Run process. Go to the **Data Manager** and select the previously created Simulation Run. From its context window activate the option **Script Actions > Submit**.



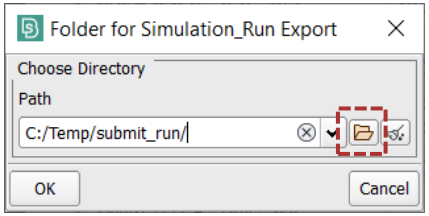
2 The process “Submit Run” is instantiated in a new Workflow scene window. The node “Solve” is executed automatically.



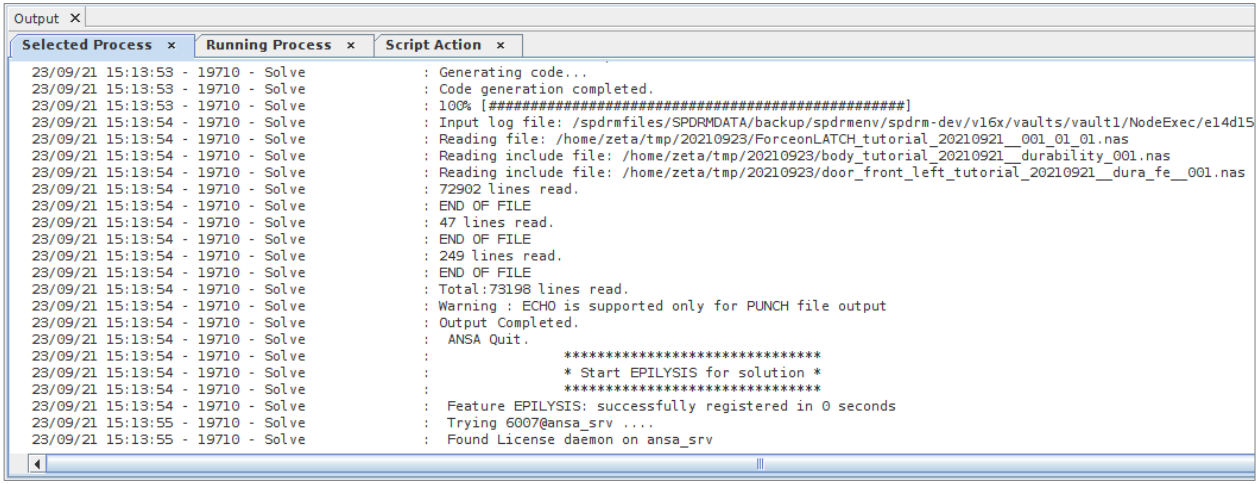
Process Execution – Submit Run



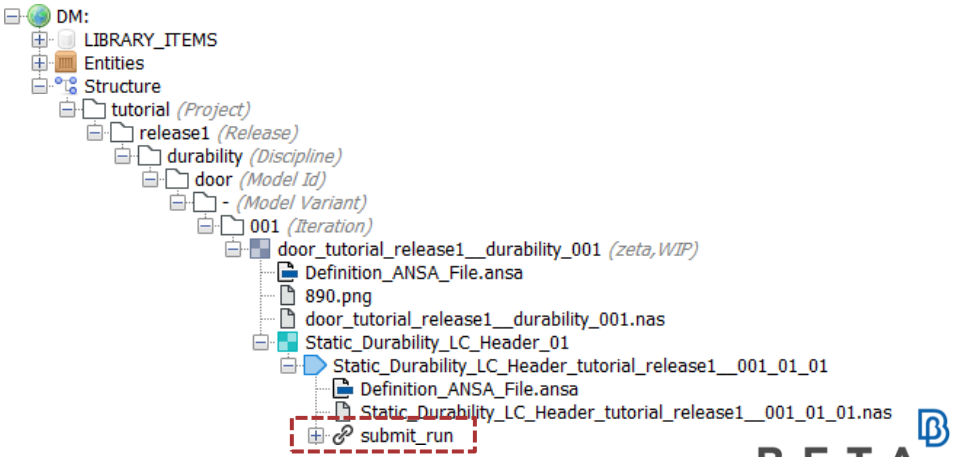
1 Once executed the Solve node launches a window for the user to select a directory where the Simulation Run will be exported and the solving will be performed. Select a directory and press **OK** to proceed.



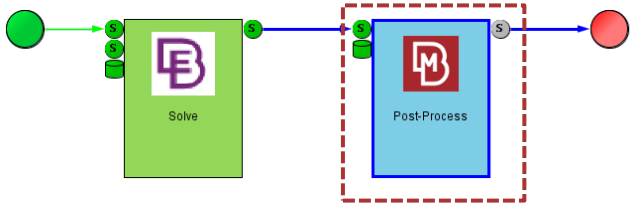
2 Epilysis is launched with input the selected Simulation Run. The solving starts and all related messages are printed in the Output window of SPDRM.



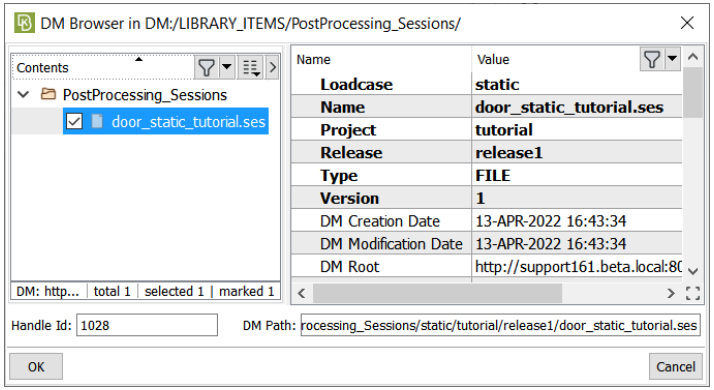
3 Epilysis exports the results files in the selected directory. Once finished, the post-run script of the node, attaches this directory to the Simulation Run DM Item as a Linked Folder.

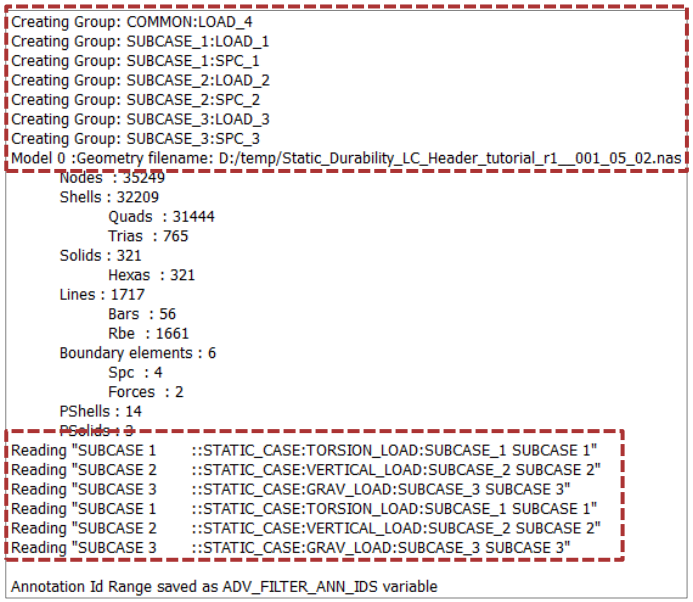


Process Execution – Submit Run



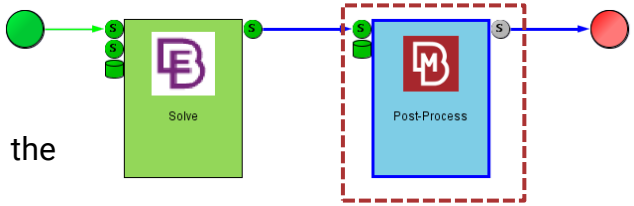
1 Once the Solve node finishes the Post-Process node becomes ready and is executed automatically. First, the user is prompted to select the META session to be executed. Select the one imported at the start (door_static_tutorial.ses). Then, META is launched in no-gui mode for the post processing of the results.





2 The session file that was imported as a PostProcessing_Session library item, is now called and executed in the background and creates several Reports (Curves, Images, KeyValues, Plots, etc).

Process Execution – Submit Run



All the generated Reports are now listed per type under the Simulation Run. Access the Data Manager to see the Reports.

File Data Process Resources Tools Window Help

Data Manager | Submit Run (1058)

Project | Release | Model Id | Model Variant

Search | Default view

- release1 *(Release)*
  - durability *(Discipline)*
    - door *(Model Id)*
      - - *(Model Variant)*
        - 001 *(Iteration)*
          - door_tutorial_release1__durability_001 *(zeta,WIP)*
            - door_tutorial_release1__durability_001.nas
            - 1471.png
            - Definition_ANSA_File.ansa
            - door_tutorial_release1__durability_001.alc_aux
            - door_static_durability_loadcase_header_01
              - door_static_durability_loadcase_header_tutorial_release1__001_01_01
                - Definition_ANSA_File.ansa
                - door_static_durability_loadcase_header_tutorial_release1__001_01_01.nas
                - 20220413
                - Key_Results
                  - Curve *(Type)*
                    - Model_0_Max
                    - Model_0_Min
                    - Node_2005142TotalNodeData_Displacements_Translational
                    - Node_2005142XNodeData_Displacements_Translational
                    - Node_2005142YNodeData_Displacements_Translational
                    - Node_2005142ZNodeData_Displacements_Translational
                    - Quad4_2003426_Stresses_VonMises_MaxofTopBottom_Centroid
                    - Quad4_2003436_Stresses_VonMises_MaxofTopBottom_Centroid
                    - Quad4_2003449_Stresses_VonMises_MaxofTopBottom_Centroid
                    - Quad4_2003848_Stresses_VonMises_MaxofTopBottom_Centroid
                    - Quad4_2003864_Stresses_VonMises_MaxofTopBottom_Centroid
                  - Image *(Type)*
                    - Dtot_subcase_3
                    - Dx_subcase_1
                    - Dz_subcase_2
                  - KeyValue *(Type)*
                    - max_stress_subcase_1
                    - max_stress_subcase_2
                    - max_stress_subcase_3
                    - max_tot_disp
                    - max_x_disp
                    - max_z_disp
                  - Plot *(Type)*
                    - Deformation_2d
                    - Scalar_2d
                    - Scalar_Statistics_2d
                  - Presentation *(Type)*
                    - Post_processing_report
                  - Video *(Type)*
                    - hindges_max_stress_values_per_subcase

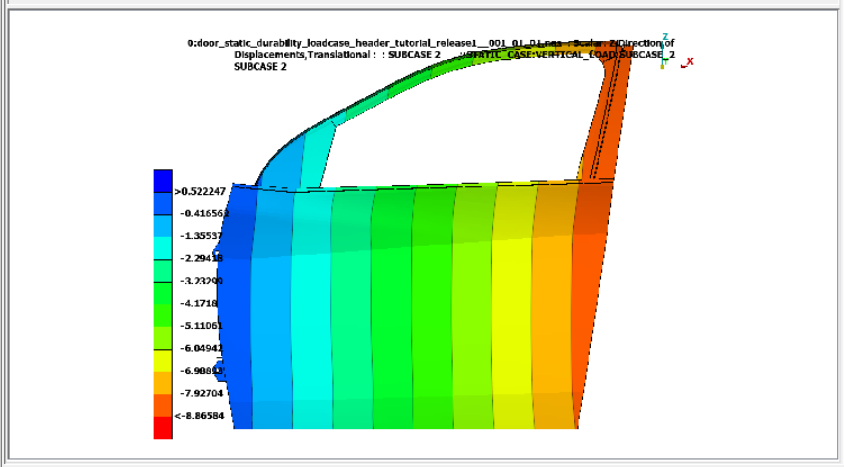
Handle Id: 1495

DM Path: :atic_durability_loadcase_header_01/01/Nastran/door_static_durability_loadcase_header_tutorial_release1__001

Report: Dz_subcase_2

Properties | Contents | Pedigree | Lifecycle Graph | References | Updates | Job Status | Overwrites

Label	Value
DM Properties	
Type	Image
Report_Parameters	-
Name	Dz_subcase_2
Loadcase	door_static_durability_loadcase_header_01
Simulation_Run	door_static_durability_loadcase_header_tutorial_release1__001_01_01
Simulation_Model	door_tutorial_release1__durability_001
DM Attributes	
Value	
Status	WIP
Comment	
File	Dz_subcase_2.png
Additional Attributes	
Software Version	22.1.1
User	zeta
SPDRM Attributes	
Id	1495

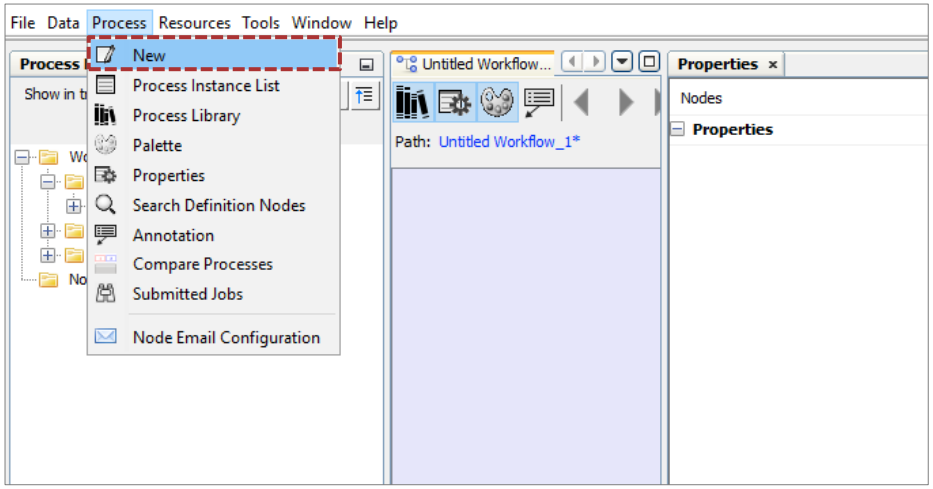


BETA SIMULATION SOLUTIONS

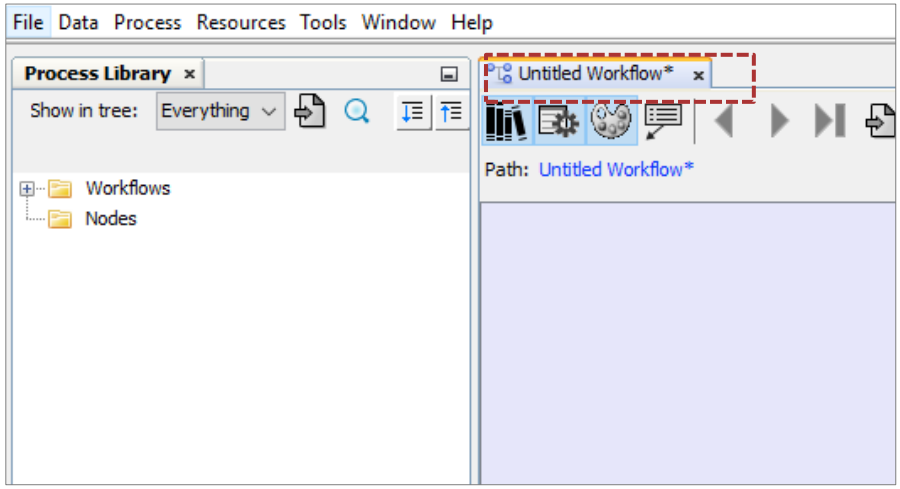
Design the process Build Subsystem

Creation of the first nodes

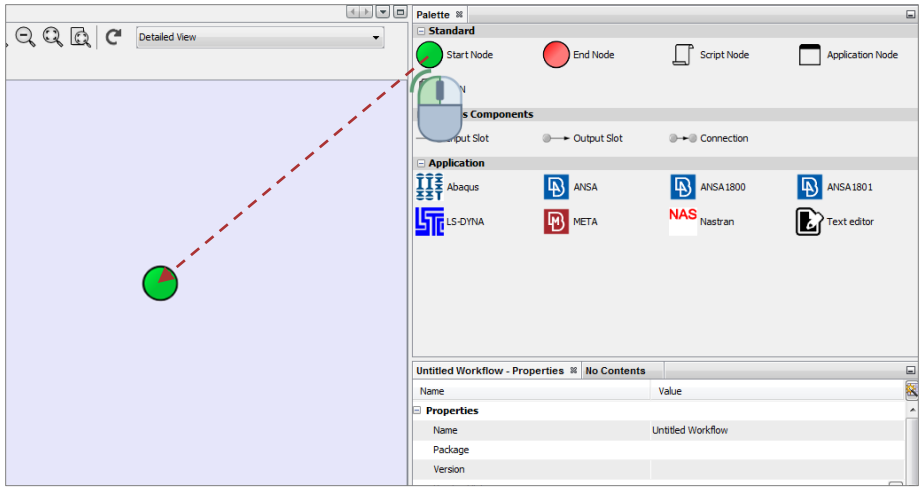
1 Go to Process>New



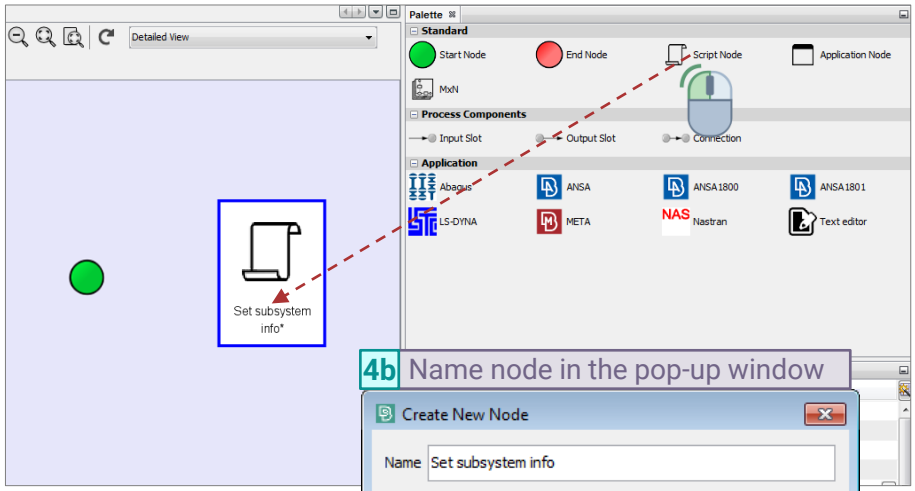
2 A new “Untitled Workflow” tab opens



3 Drag & drop Start Node from Palette to the Workflow scene



4 Drag & drop Script Node from Palette to the Workflow scene

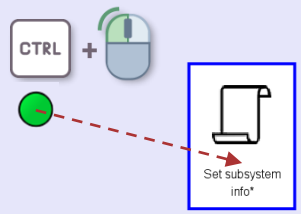


4b Name node in the pop-up window

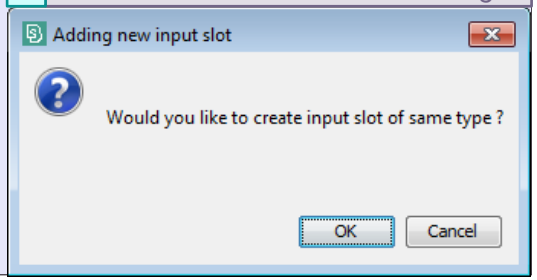
BETA SIMULATION SOLUTIONS

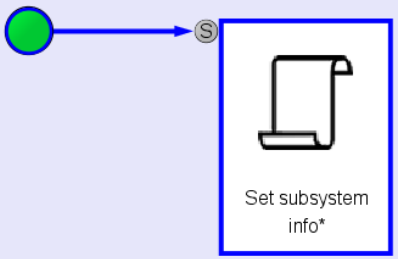
Connect the nodes

1a To connect the nodes, move mouse over the Start Node. Pressing Ctrl key and left mouse button, drag mouse towards the Script Node.

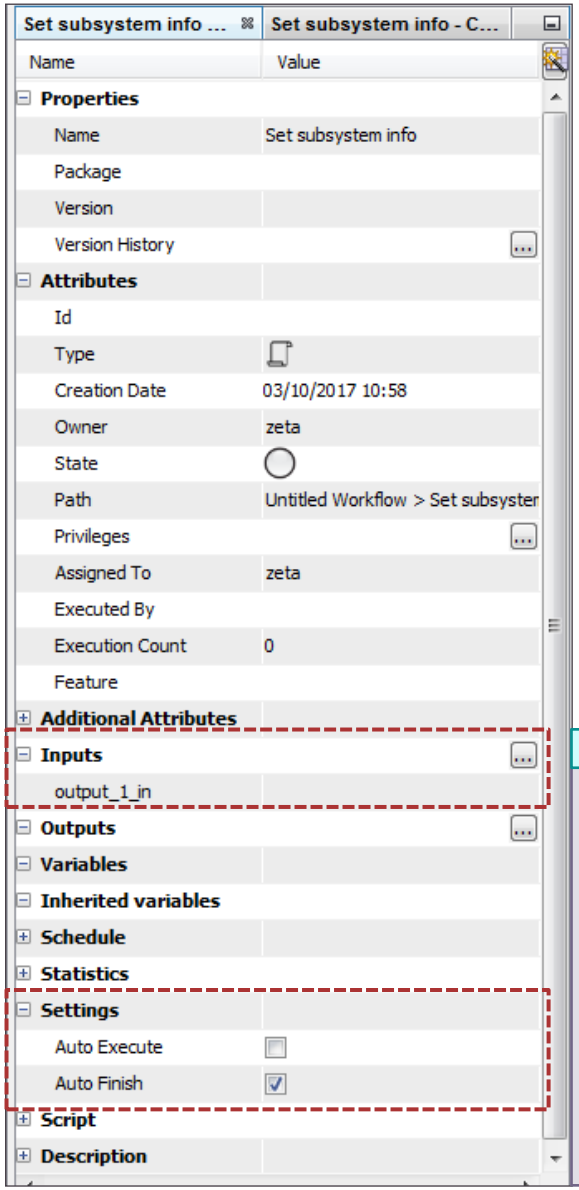


1b Press **OK** in the confirmation dialog





2 In order to connect the two nodes, SPDRM created a new **Input Slot** of type String for the "Set subsystem info" node.



3 Activate options **Auto Execute** and **Auto Finish**.
These options will make the node to start automatically once its input is available and finish automatically once its job is terminated.
Both options affect the behavior of the node during the execution of the workflow.

Once created, a node can be edited through the Properties window, accessed through **Main menu Process > Properties**, or the respective toggle button in the toolbar.

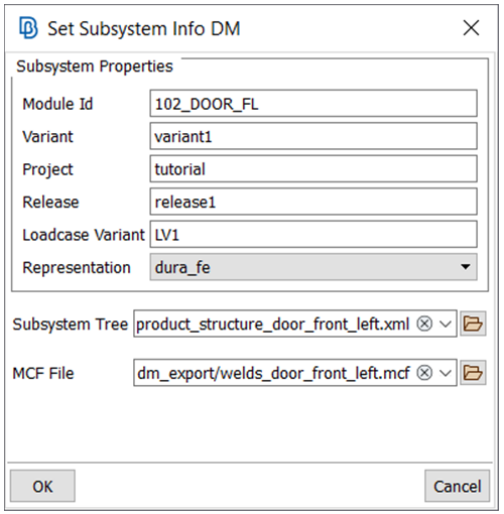
BETA SIMULATION SOLUTIONS

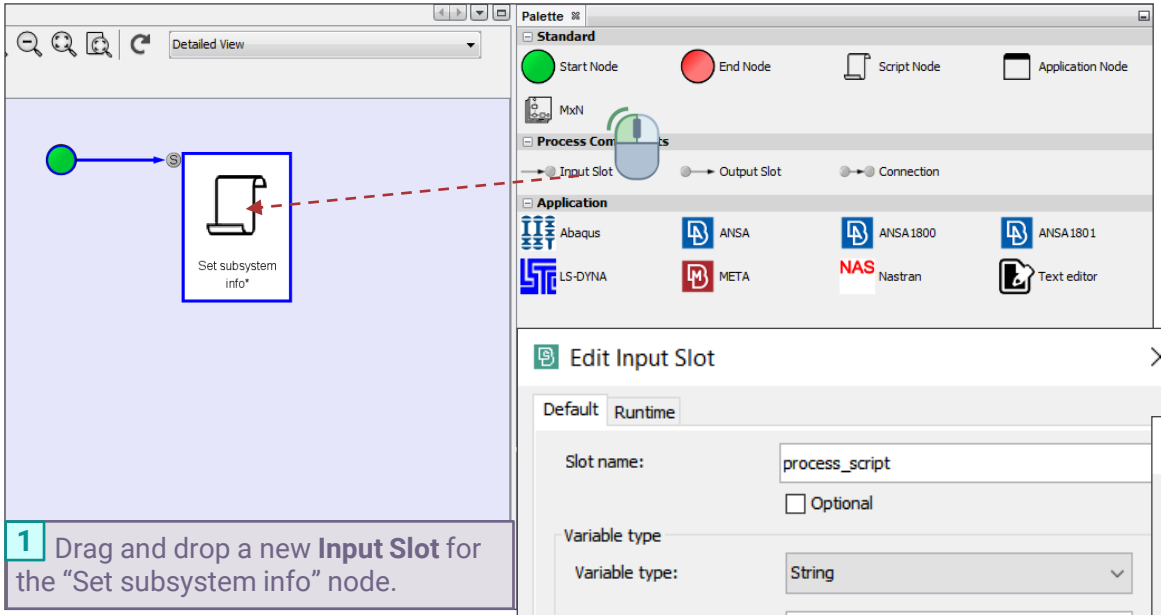
“Set subsystem info”: Create Input Slot

When the “Set subsystem info” is executed it launches a custom window, that allows the user to input all the needed information and files regarding the subsystem to be built. For this, the node will run a special script, currently located under DM.

`DM:/LIBRARY_ITEMS/tutorial_data/spdrm_scripts/process_scripts/Build_Subsystem_PrePostRun.py`

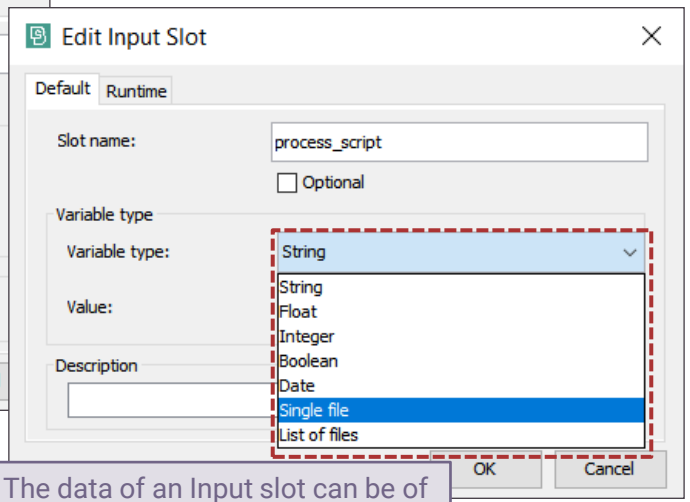
This script will be provided to the node via an **Input Slot**.





1 Drag and drop a new **Input Slot** for the “Set subsystem info” node.

2 Name the slot

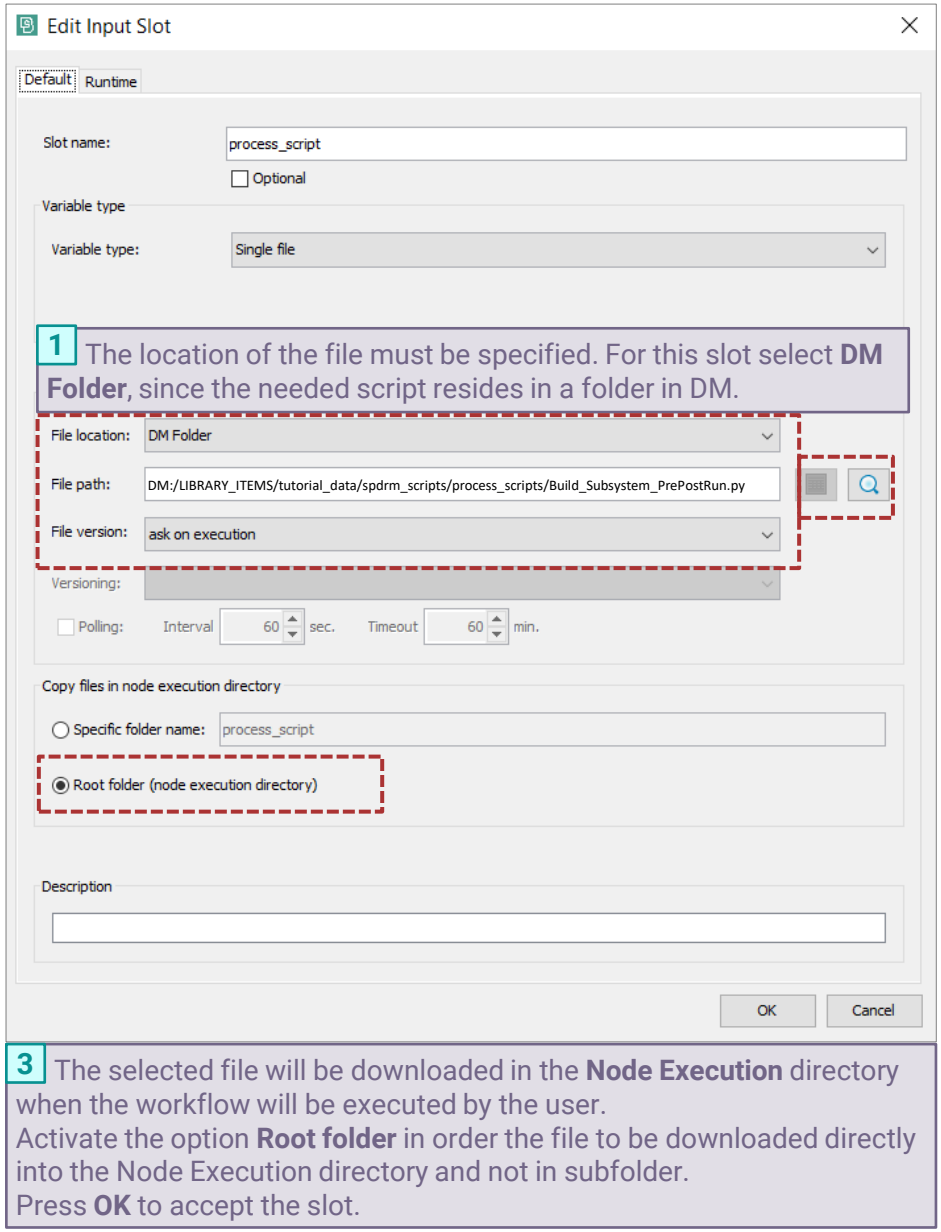


3 The data of an Input slot can be of any type from the available ones in the **Variable type** drop-down menu. For this slot, select type **Single file**.

New **Input/Output Slots** can be created either by editing the node from the **Properties** window, or by selecting and dragging the respective item from the **Palette** window.

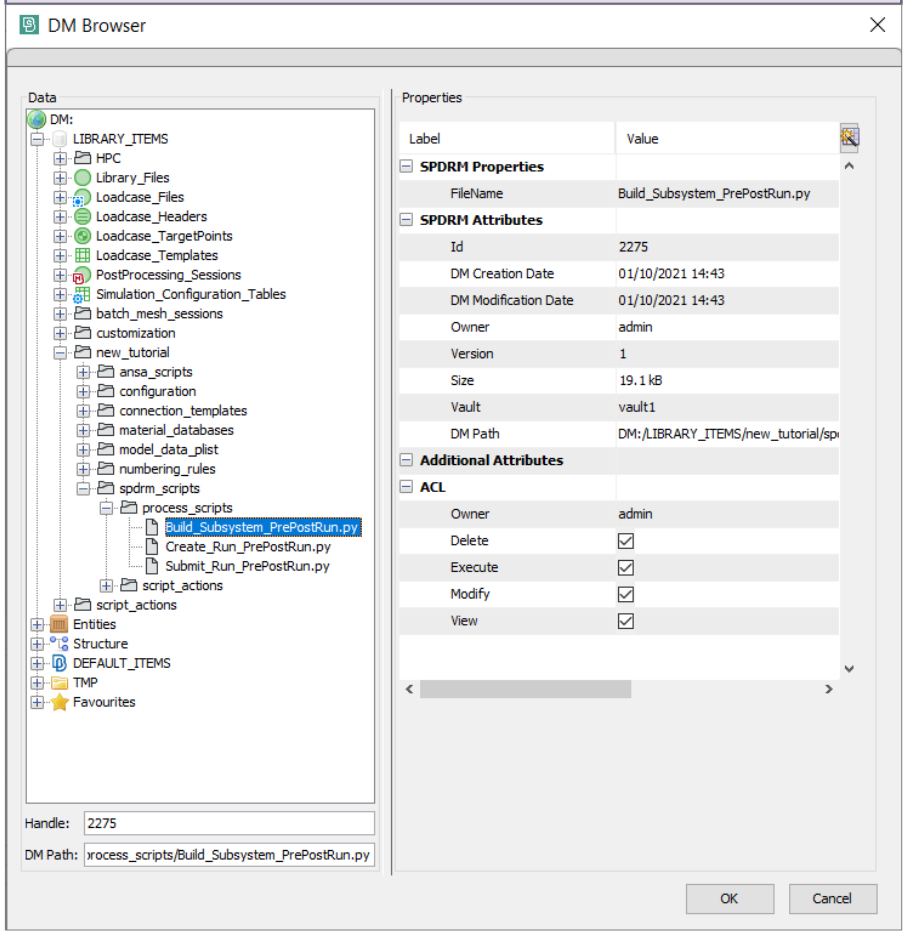
BETA SIMULATION SOLUTIONS

“Set subsystem info”: Create Input Slot

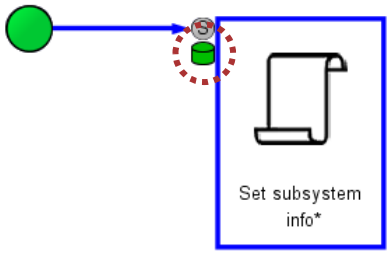


1 The location of the file must be specified. For this slot select **DM Folder**, since the needed script resides in a folder in DM.

2 Press the **Browse** button to navigate into the folders of the DM. Select the script “Build_Subsystem_PrePostRun.py” and press **OK**.

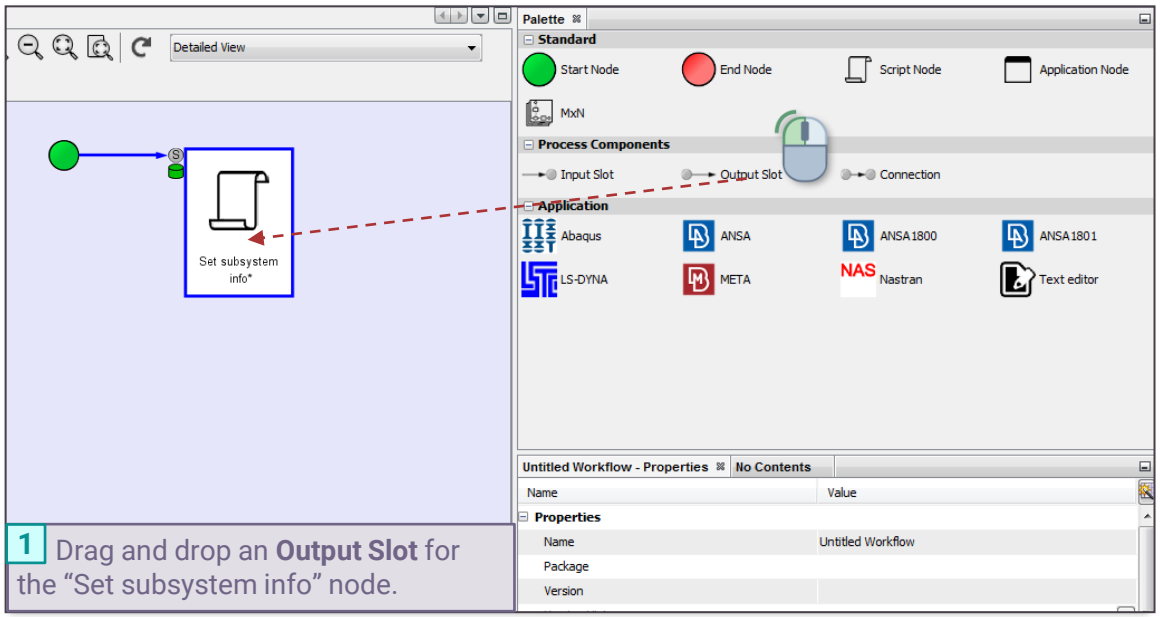


3 The selected file will be downloaded in the **Node Execution** directory when the workflow will be executed by the user.
Activate the option **Root folder** in order the file to be downloaded directly into the Node Execution directory and not in subfolder.
Press **OK** to accept the slot.

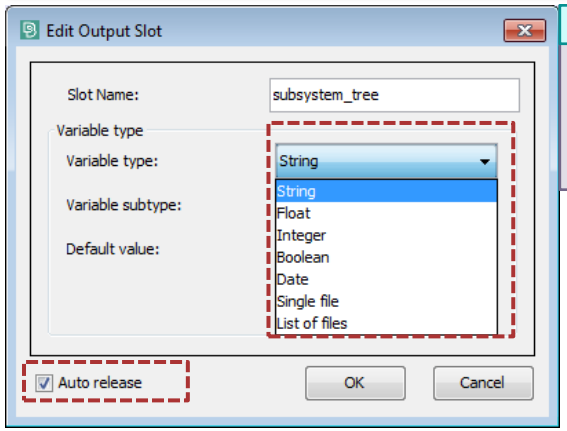


“Set subsystem info”: Create Output Slot

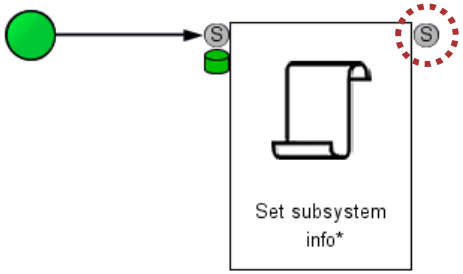
The “Set subsystem info” node needs also an **Output Slot**. This will be the connector to the rest nodes of the workflow and also it will deliver the information of the file location selected by the user during the execution.



1 Drag and drop an **Output Slot** for the “Set subsystem info” node.



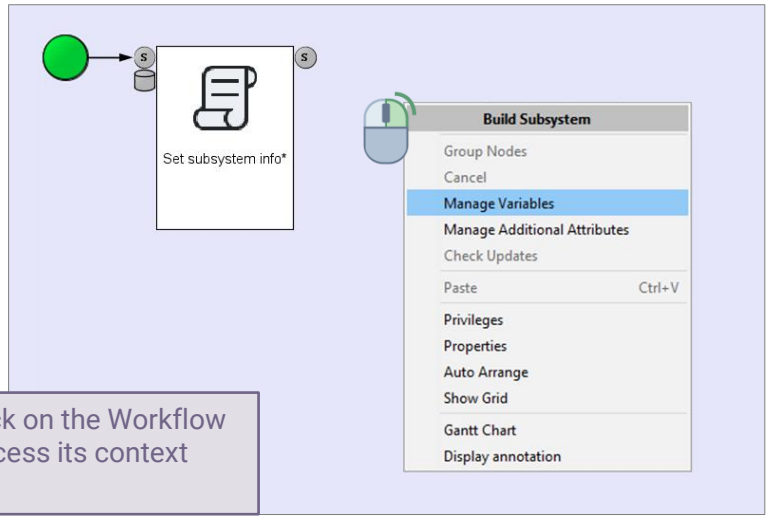
2 Name the slot.
The data of an **Output slot** can be of any type from the available ones in the **Variable type** drop-down menu. For this slot, select type **String**.
Press **OK** to accept the slot.



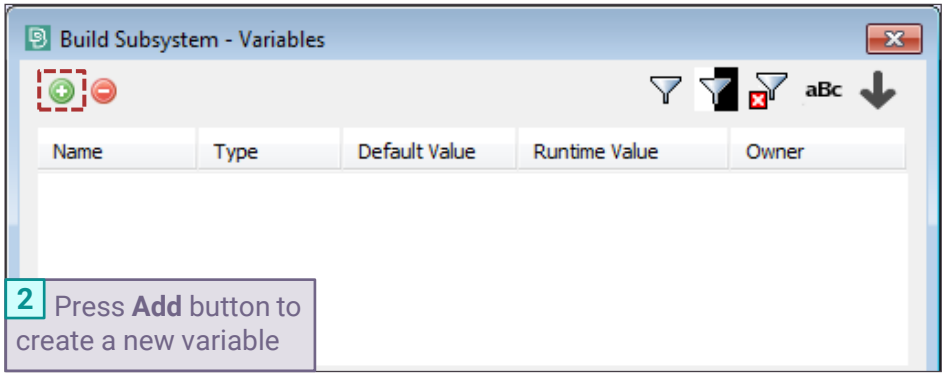
Activate **Auto release** option of **Output Slot** so as during the execution of the workflow the data will be released as soon as the node is finished, without any user action.

Add Variables to the workflow

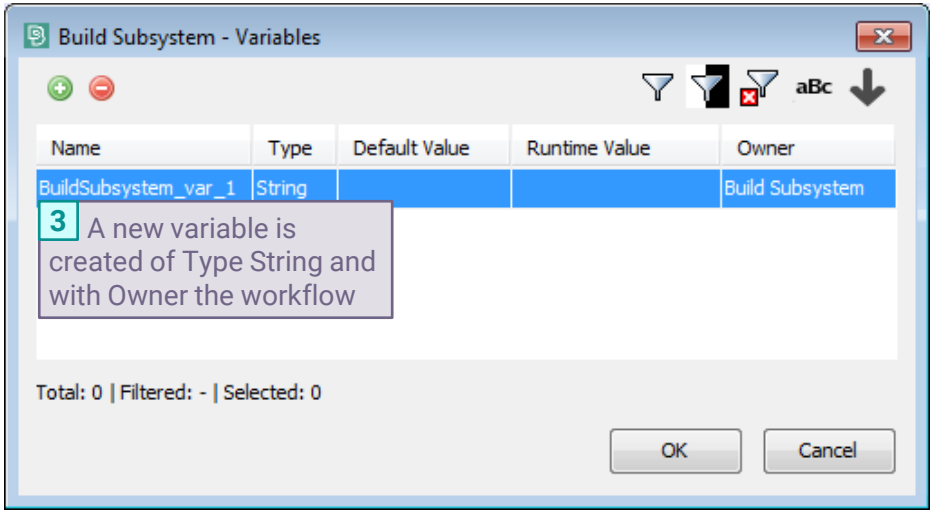
Another way of communicating data throughout a workflow, apart from Input/Output Slots, is the process and nodes Variables. The variables are defined during the design of the workflow, and they take values during its execution. Each node of a workflow can have its own variables but also inherit the variables of the parent workflow. Thus, the value that a variable of the workflow acquires, is known to all of its nodes.



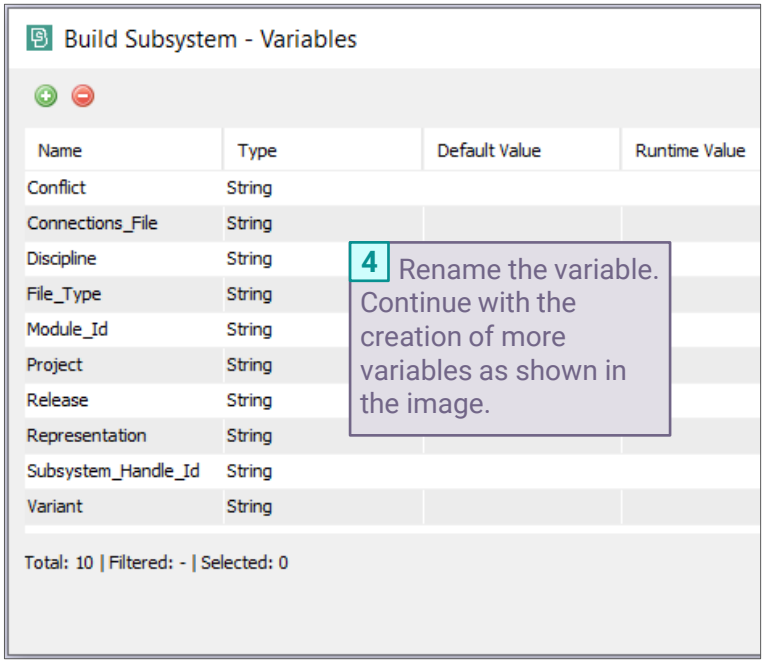
1 Right-click on the Workflow scene to access its context menu



2 Press **Add** button to create a new variable



3 A new variable is created of Type String and with Owner the workflow

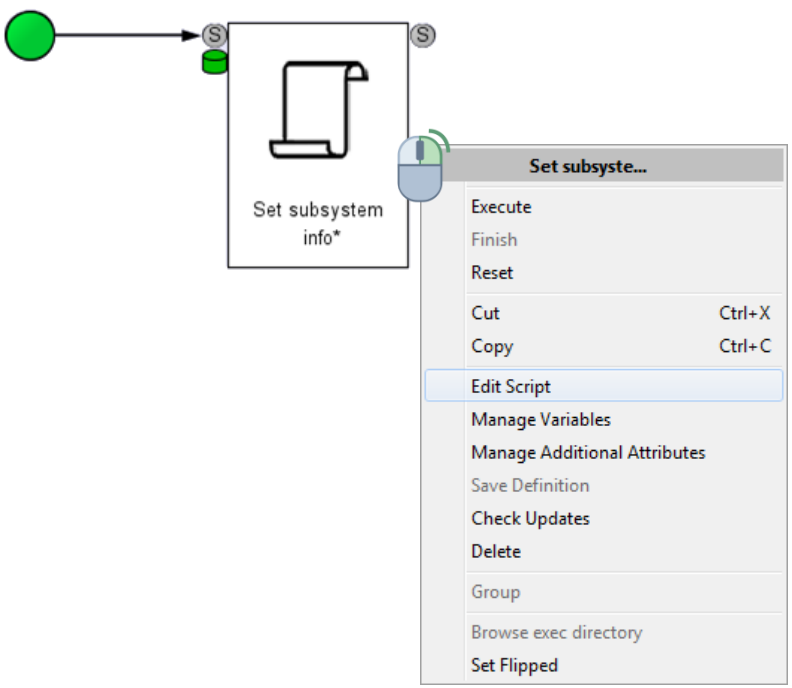


4 Rename the variable. Continue with the creation of more variables as shown in the image.

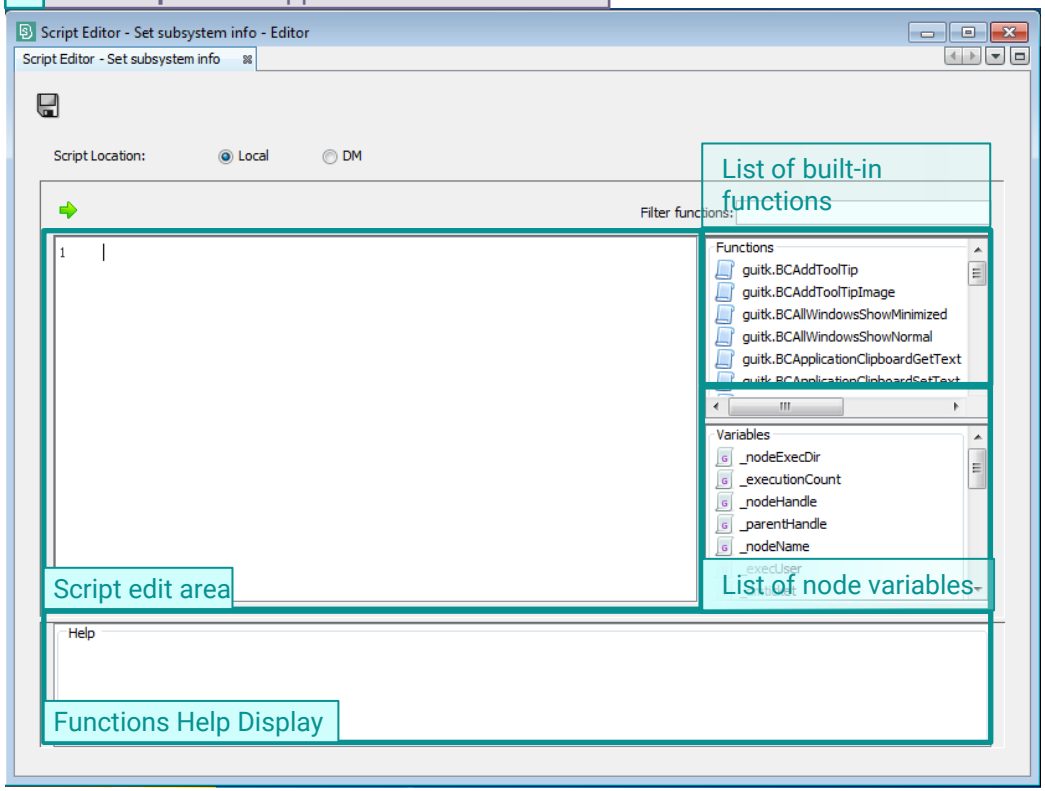
“Set subsystem info”: Set-up the script

During the execution of this node, the script that is set-up in the Script Editor is executed. This, in turn, imports and executes the script of the Node Execution directory (Build_Subsystem_PrePostRun.py) which is made available through the input slot (process_script). The function SetSubsystemInfo() that is executed, is the one that creates and launches the custom GUI.

1 Select **Edit Script** from the context menu of the node

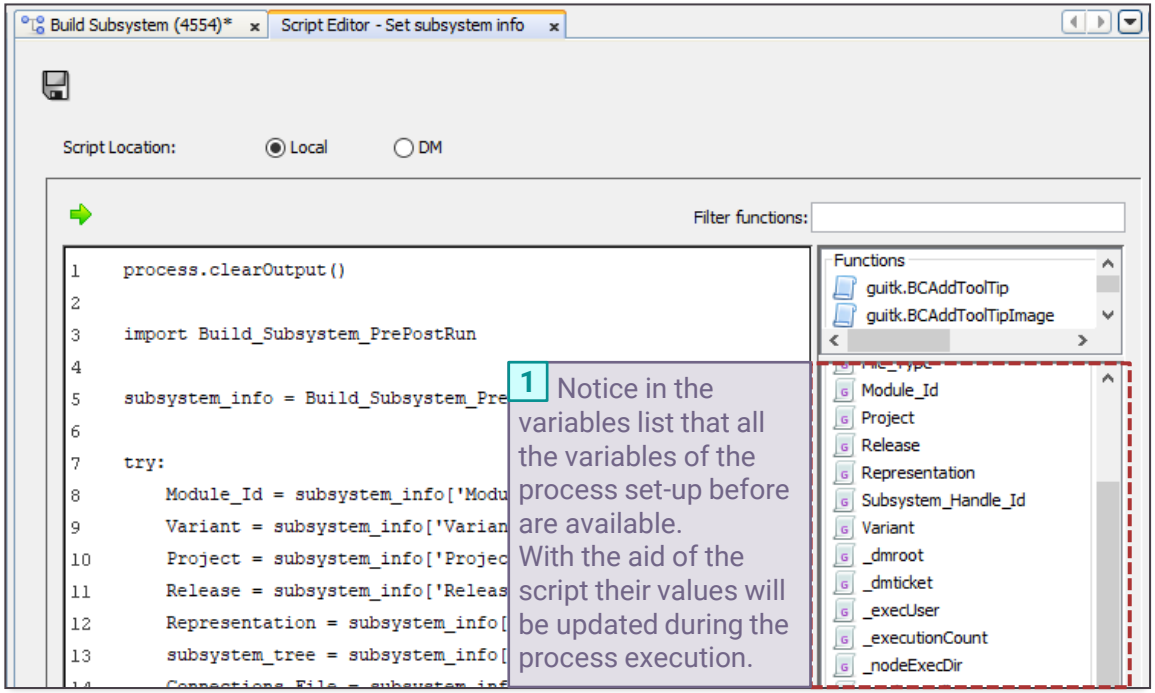


2 The **Script Editor** appears



“Set subsystem info”: Set-up the script

All the variables of the node and process are listed in the Variables list. Their values can be used and updated by the script during the node execution according to the user input in the custom interface.



1 Notice in the variables list that all the variables of the process set-up before are available.
With the aid of the script their values will be updated during the process execution.

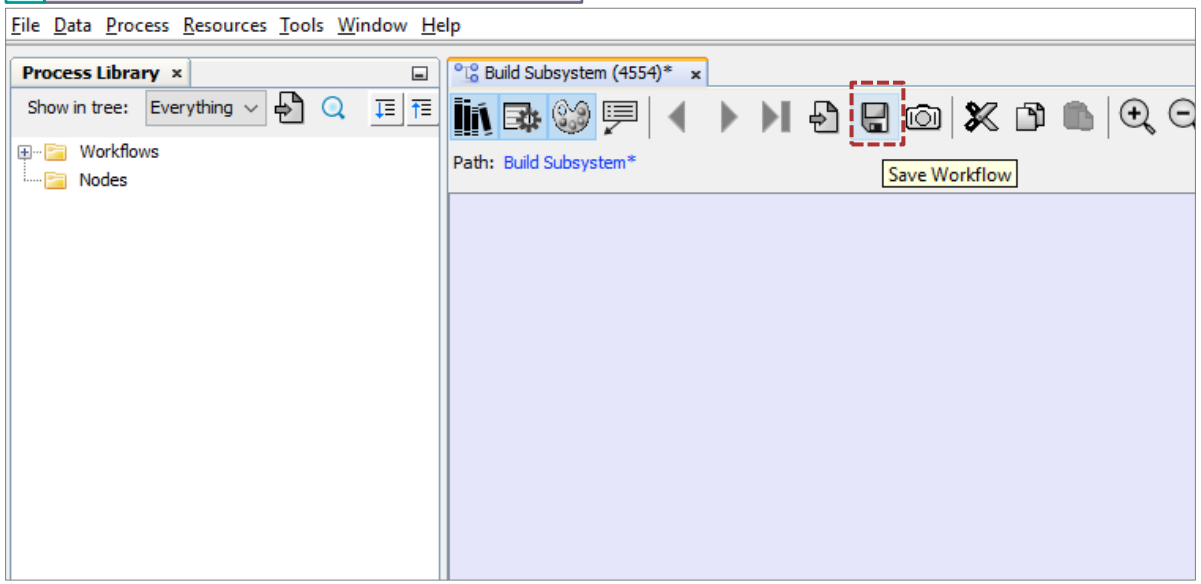
2 The python script that will be called, once executed will return a dictionary with keys the variables’ names and values their updated values as these were input by the user in the custom interface.
The final code that must be written in the script editor can be copied from the text below. Type the code in the script edit area and press the **Save** button.

```
process.clearOutput()
import Build_Subsystem_PrePostRun
subsystem_info = Build_Subsystem_PrePostRun.SetSubsystemInfo()
try:
    Module_Id = subsystem_info['Module_Id']
    Variant = subsystem_info['Variant']
    Project = subsystem_info['Project']
    Release = subsystem_info['Release']
    Representation = subsystem_info['Representation']
    subsystem_tree = subsystem_info['Hierarchy_Tree']
    Connections_File = subsystem_info['MCF_File']
    Conflict = subsystem_info['conflict']
    Subsystem_Handle_Id = subsystem_info['Handle_id']
except:
    scriptCommands.abort('Process canceled by user')
```

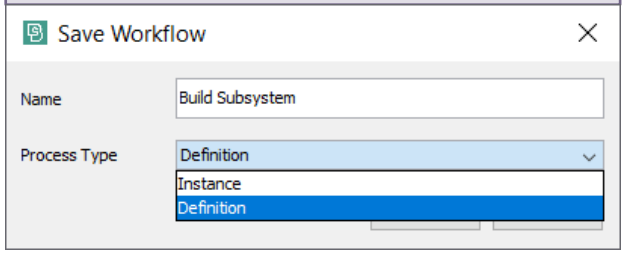
BETA SIMULATION SOLUTIONS

Save the workflow

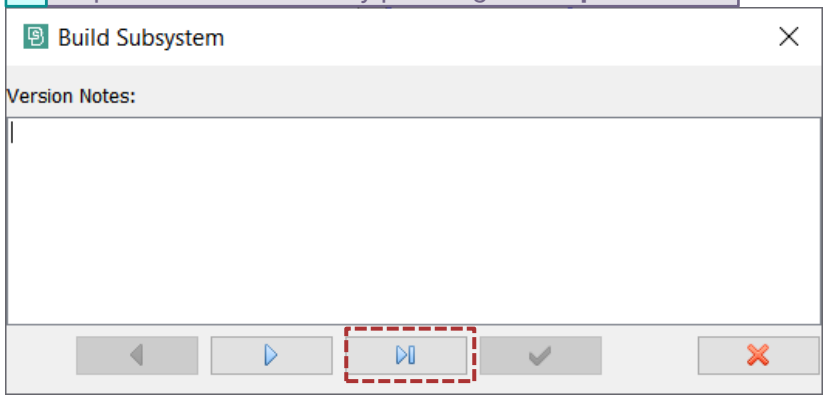
1 Press the **Save** button from the toolbar

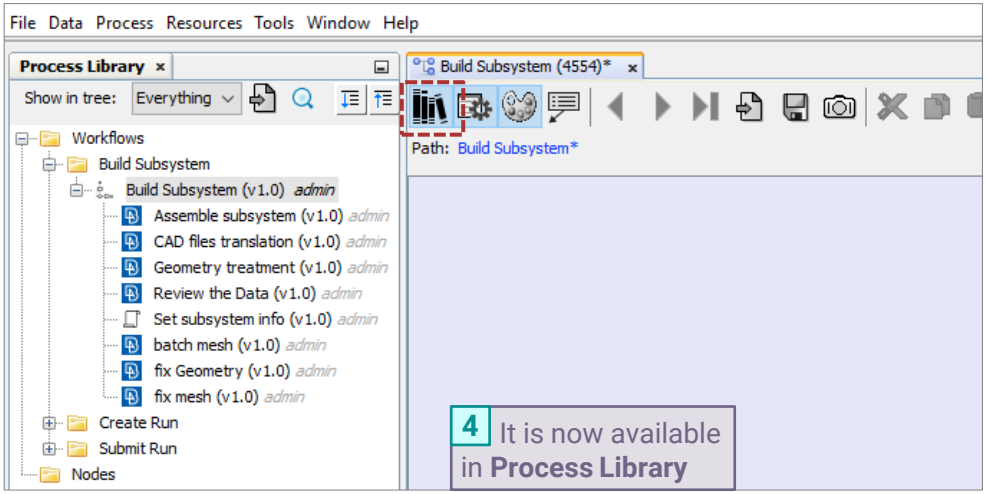


2 Name the workflow. A workflow can be saved as **Instance** or as **Definition**.
Select **Definition** and press **OK**.



3 Skip the Version Notes by pressing the **Skip all** button





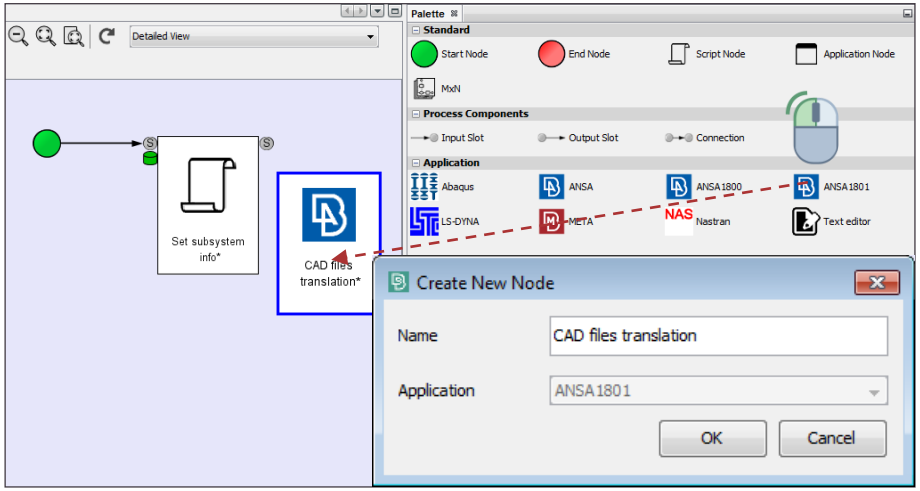
4 It is now available in **Process Library**

A workflow saved as **Definition** is available in **the Process Library** and accessible to other users as well. It subjects to versioning and it is editable. It can be exported to file of *.json or *.ser format.

BETA SIMULATION SOLUTIONS

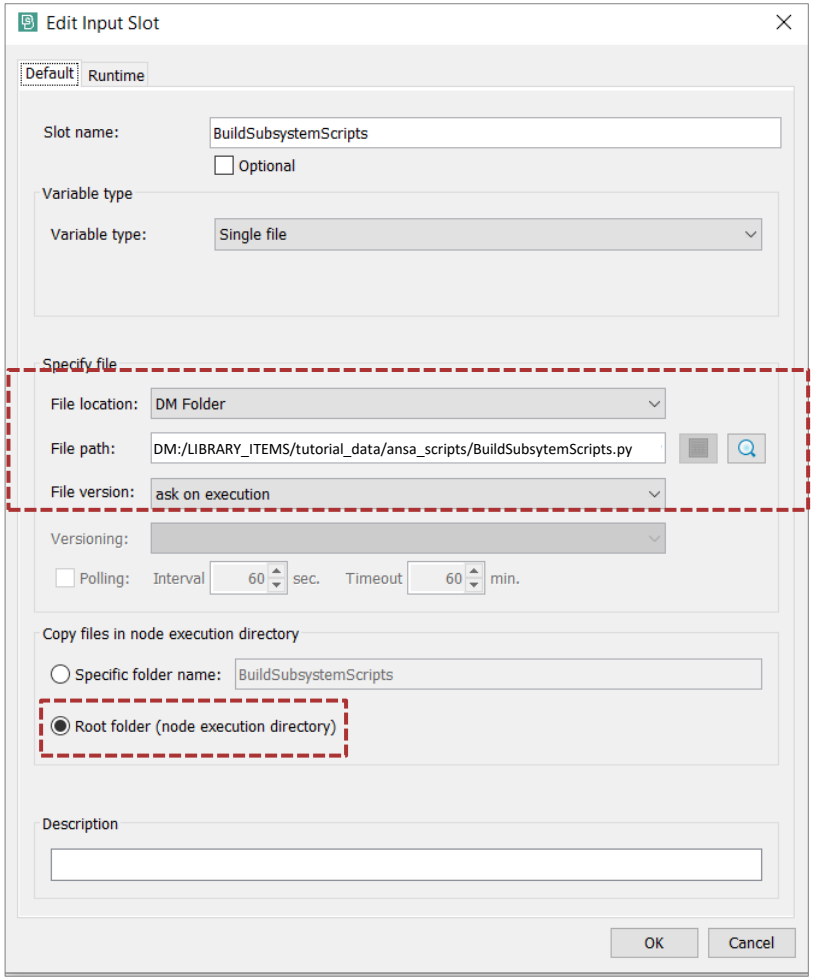
Creation of “CAD files translation” node

1 Create an **Application Node** ANSA and connect it with Ctrl key pressed and drag-drop operation from the output slot of the “Set subsystem info node to the “CAD translation” node.



When connecting two nodes with drag & drop operation from the output slot of first node to the second node, the automatically created input slot acquires the name of the connected output slot with the suffix: `_in`

2 Create one more **Input Slot** of type **Single File**. This will input the python script that ANSA will execute once launched. The script resides in the DM, so define as File location **DM Folder** and navigate in DM to select the file “BuildSubsystemScripts.py”.
Activate option **Root folder**.



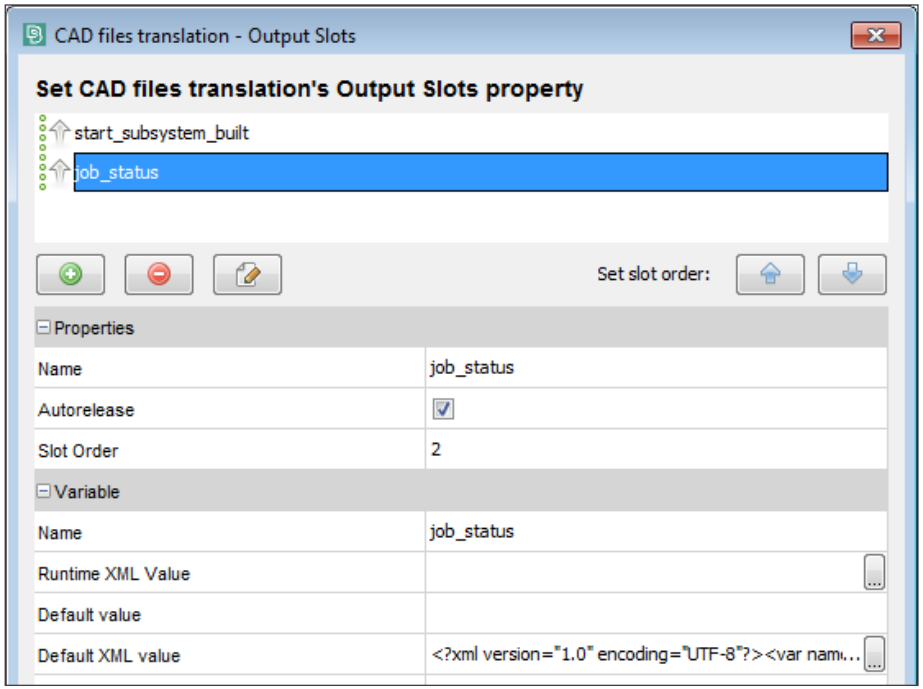
Creation of “CAD files translation” node

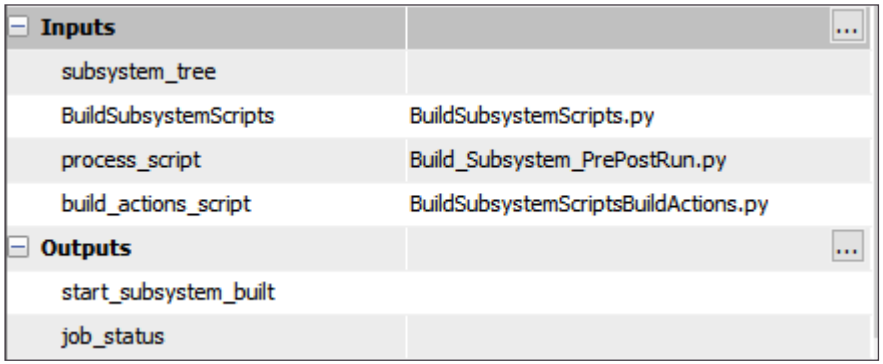
1. Connect the node with the “Set subsystem info” node, as you did in slide 53, so as the adequate input slot to be created automatically.
2. Create two more input slots of type Single file. These will provide to the node the pre and post run scripts that will be executed before and after the execution of the ANSA application, and also the script for the ANSA build actions:

   DM:/LIBRARY_ITEMS/tutorial_data/spdrm_scripts/process_scriptsBuild_Subsystem_PrePostRun.py

3. DM:/LIBRARY_ITEMS/tutorial_data/ansa_scripts/BuildSubsytemScriptsBuildActions.py

   Create also two **Output Slots** of type **String**

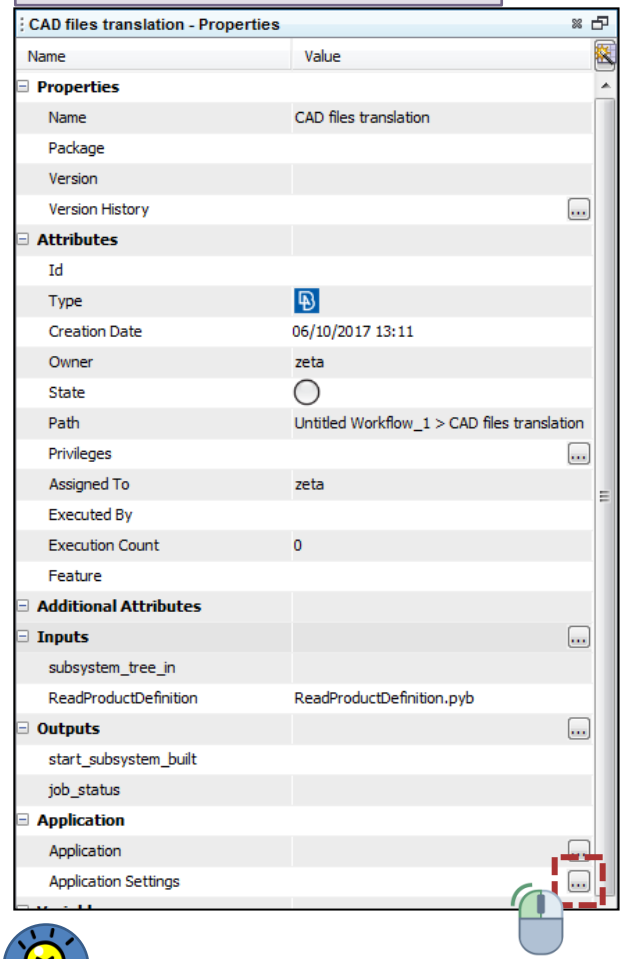




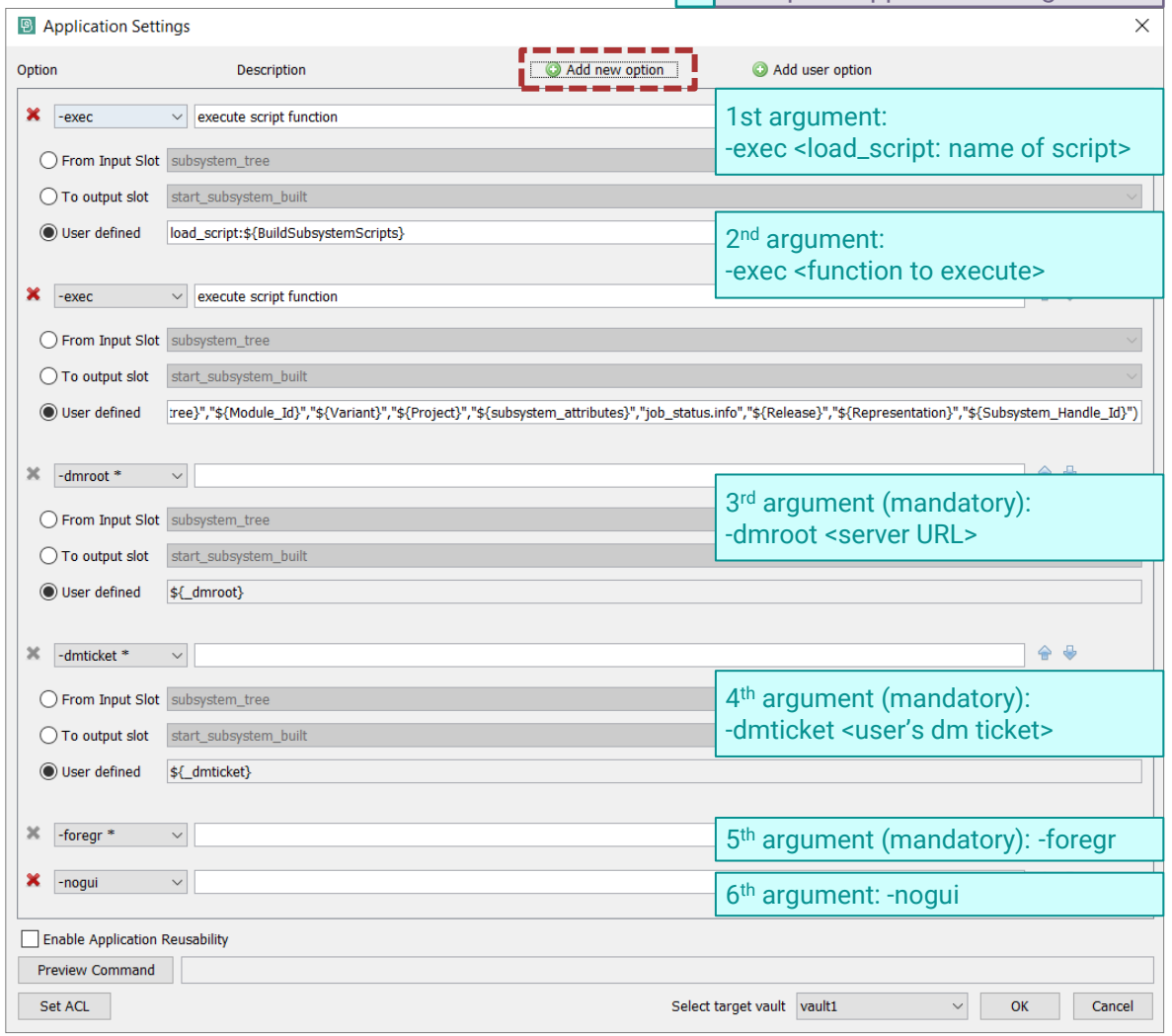
“CAD files translation”: Set up the application settings

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



2 Set-up the application’s arguments



1st argument:
-exec <load_script: name of script>

2nd argument:
-exec <function to execute>

3rd argument (mandatory):
-dmroot <server URL>

4th argument (mandatory):
-dmticket <user’s dm ticket>

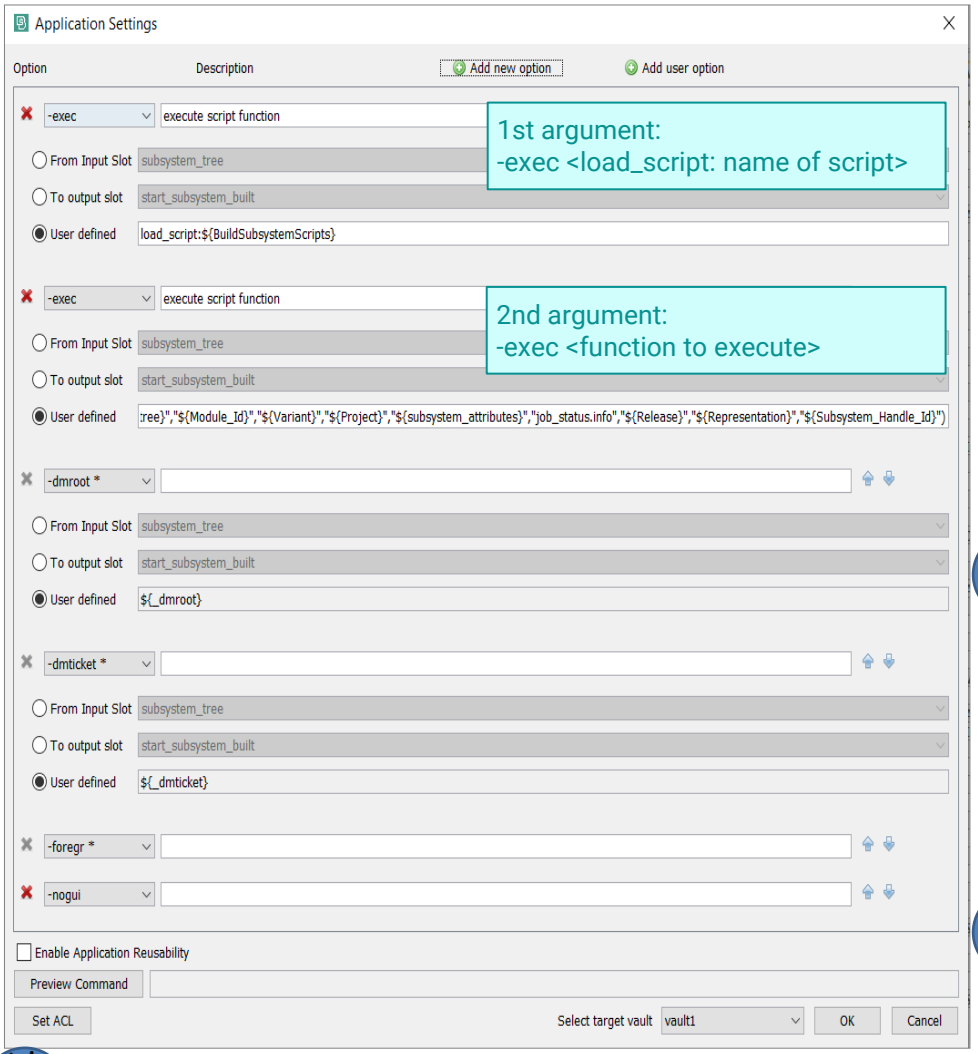
5th argument (mandatory): -foregr

6th argument: -nogui

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.

BETA SIMULATION SOLUTIONS

“CAD files translation”: Set up the application settings



During the execution of the node, the *${}* values will be filled by the runtime values of the respective variables.

For example:

${BuildSubsytemScripts} will be replaced by the name of the script file that will be downloaded to the Node Execution directory via the **Input Slot** with the name “ReadProductDefinition”.

Respectively the second argument will be filled with the runtime values of the corresponding variables. These will be the arguments of the script function `ReadProductDefinition()`.

NOTE that no blank spaces are allowed within the arguments.
See below the exact text of these two arguments.

```
load_script:${BuildSubsytemScripts}
```

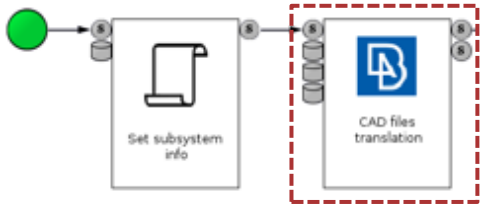
```
ReadProductDefinition("${subsystem_tree}","${Module_Id}","${Variant}","${Project}","${subsystem_attributes}","job_status.info","${Release}","${Representation}","${Subsystem_Handle_Id}","${profile_version}")
```

Pressing the **Preview Command** button the user can have a preview of the command that is going to be executed upon execution of the node.

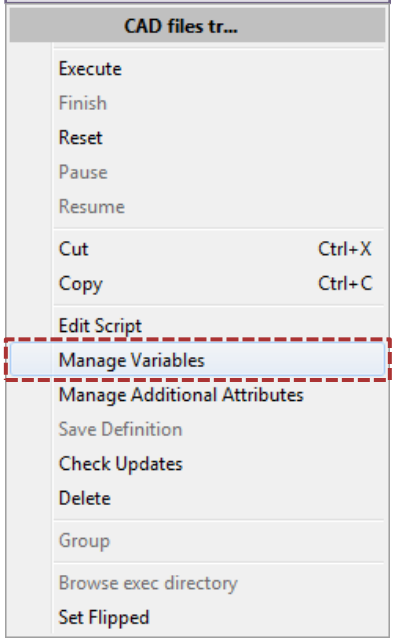
Throughout the workflow for all ANSA executions there is one unique ANSA script that is used (BuildSubsystemScripts.py) This is used for all ANSA nodes and just different functions are called.

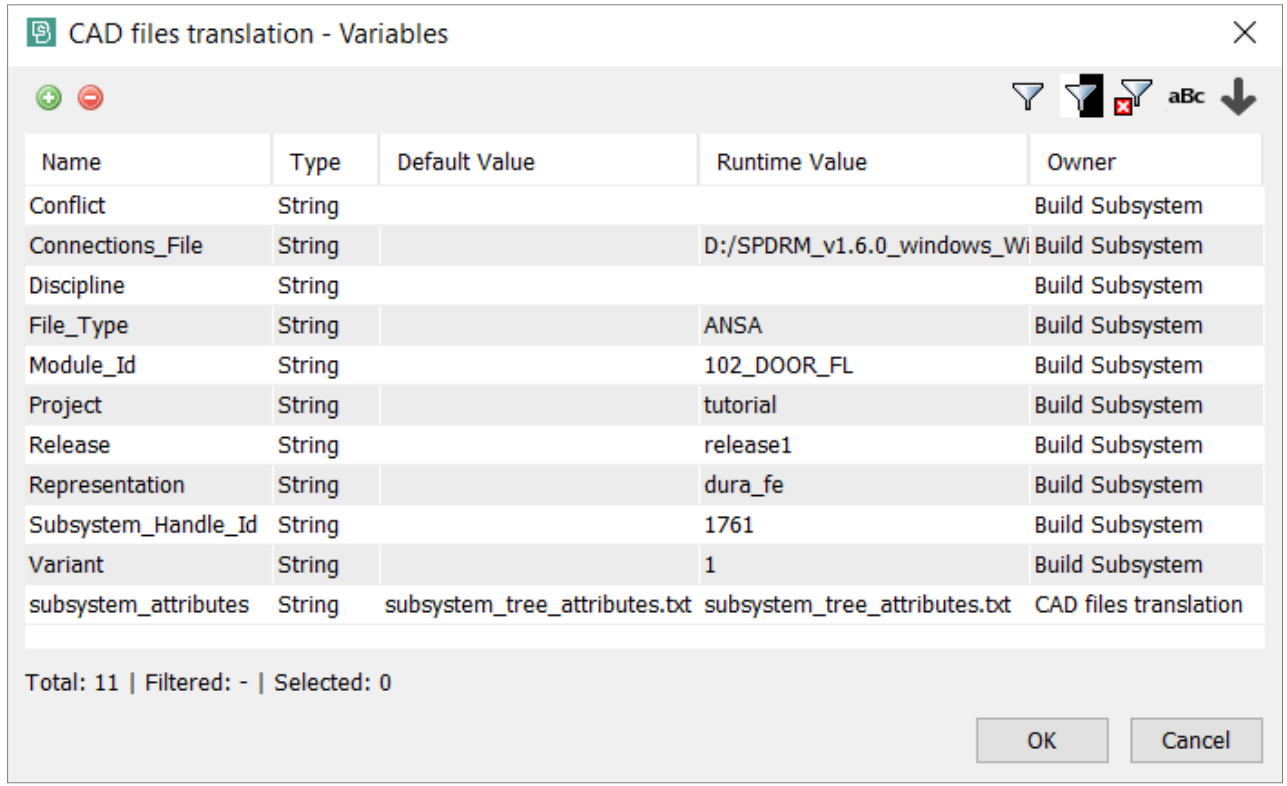
“CAD files translation”: add node variable

Apart from the workflow’s variables, the user can add variables to nodes separately, in order to be used during the node’s execution. In “CAD files translation” node we will add a variable that will host the name of a .txt file which will be generated by ANSA and will be read by SPDRM upon the termination of ANSA. This file will contain the updated values of the subsystem’s properties.



1 Right-click on the node and select “Manage Variables” from its context menu.

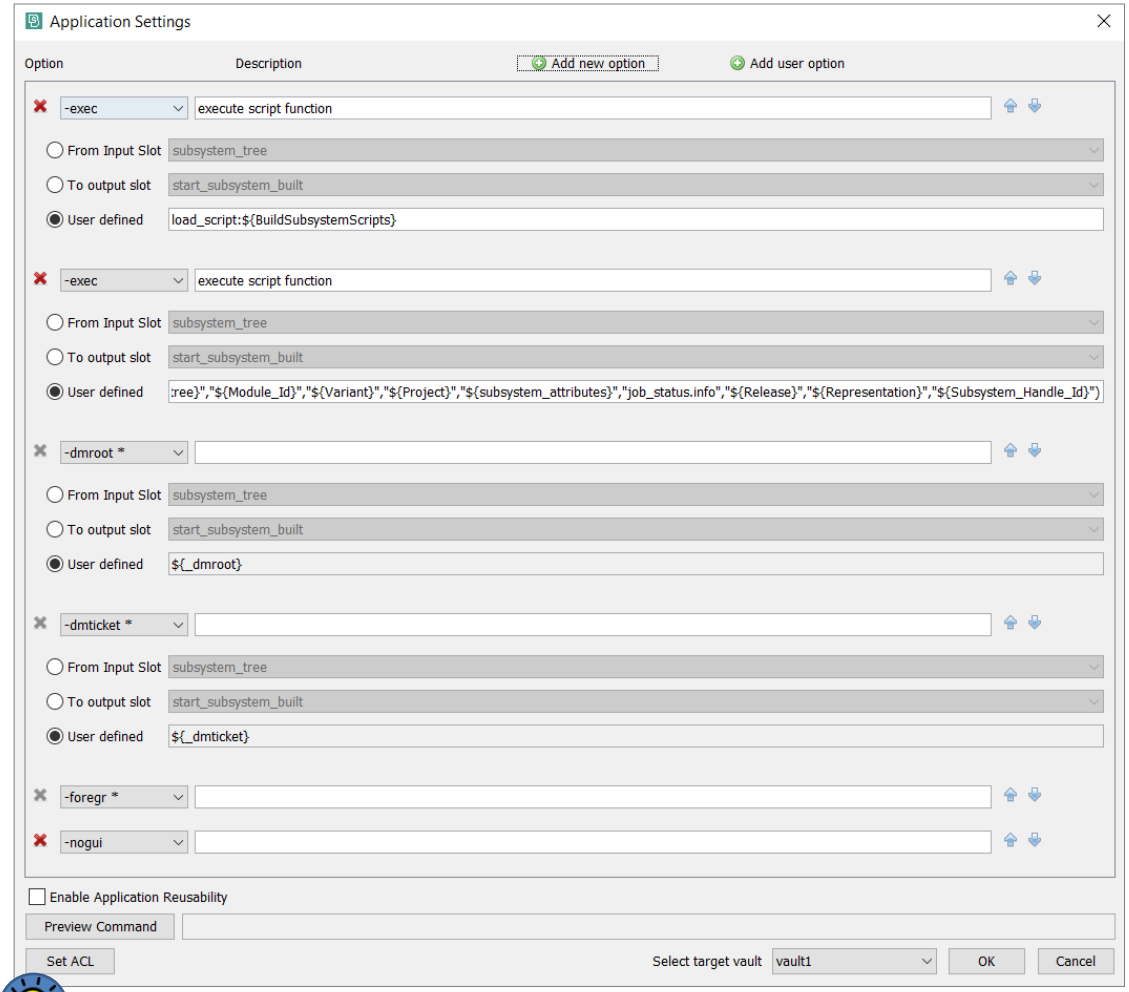




2 The Variables window appears, listing all previously defined workflow variables. Create a new variable as shown above.

“CAD files translation”: Set-up the script

Application Nodes can execute scripts similarly to the script nodes. The difference is that they can execute one script right before the launch of the application, and one right after the termination of the application. The script edit area of the Script Editor of an application node is separated in three tabs: **Pre-run**, **Main** and **Post-run** script.



1 The Pre-run script that will be used here is for the creation of a file called “job_status.info” which is used to store the outcome of the process in the form of 0=success, 1=failure.

```
process.clearOutput()
import Build_Subsystem_PrePostRun

try:
                profile_version =
Build_Subsystem_PrePostRun.CADfilesTranslation__preRun(Repr
esentation, _nodeExecDir)
except NameError as e:
                scriptCommands.abort(str(e))
```

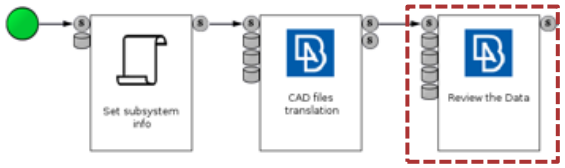
2 The Post-run script that will be used here is for the reading of the job_status.info file and update of the output slot according to success or failure value.
Also, it will read the file that is created and updated by ANSA (its name was set us a node variable previously) and update the workflow’s variables accordingly.

```
import Build_Subsystem_PrePostRun
try:
                subs_attrs =
Build_Subsystem_PrePostRun.CADfilesTranslation__postRun(_no
deExecDir, subsystem_attributes)
except NameError as e:
                scriptCommands.abort(str(e))
start_subsystem_built = 'start'
Module_Id = subs_attrs['Module Id']
Release = subs_attrs['Release']
Representation = subs_attrs['Representation']
Variant = subs_attrs['Variant']
File_Type = subs_attrs['File Type']
Subsystem_Handle_Id = subs_attrs['Handle Id']
```

The same SPDRM script is used throughout the process for the pre and post run scripts of the nodes, calling each time the needed function (Build_Subsystem_PrePostRun).

Creation of “Review the Data” node

In the same way as before, create one more application ANSA node as shown below.



Review the Data - Input Slots

Set Review the Data's Input Slots property

- input_1
- BuildSubsystemScripts
- process_script
- gui_settings
- build_actions_script

Input Slots:
i. type **String**
ii. type **Single File** in **DM Folder**, pointing to script “BuildSubsystemScripts.py”
iii. type **Single File** in **DM Folder**, pointing to script “Build_Subsystem_PrePostRun.py”
iv. type **Single File** in **DM Folder**, pointing to ANSA.xml gui settings file
v. type **Single File** in **DM Folder**, pointing to script “BuildSubsystemScriptsBuildActions.py”

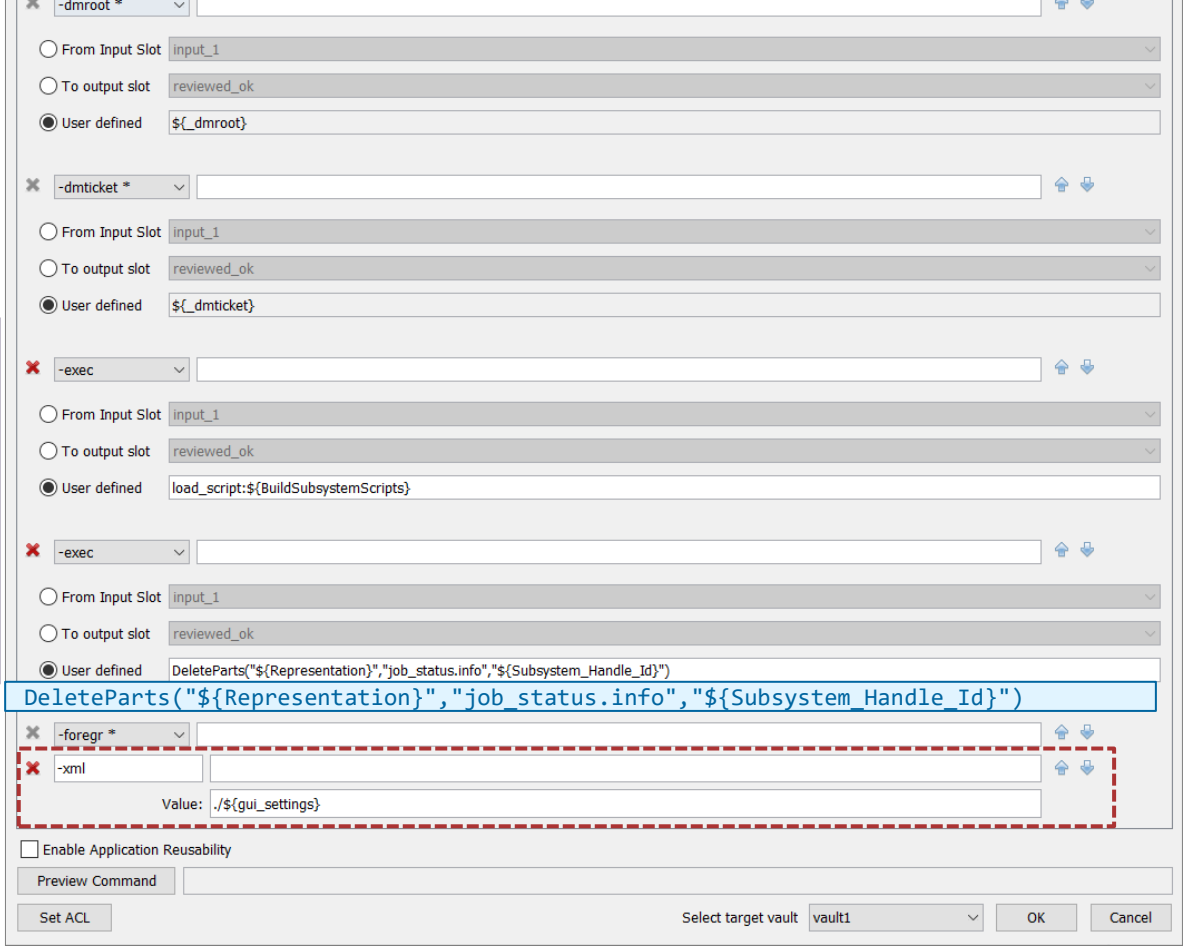
Review the Data - Output Slots

Set Review the Data's Output Slots property

- reviewed_ok

Output Slots:
i. type **String** holding the status of review

NOTE that the -xml option is not available among the options of ANSA application. For such cases the **Add user option** can be used.



Pre post run scripts Review the Data

Review the Data pre-run script

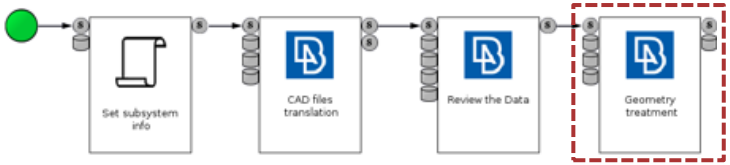
```
process.clearOutput()

import Build_Subsystem_PrePostRun

Build_Subsystem_PrePostRun.HandleJobStatus__preRun(_nodeExecDir)
```

Review the Data post-run script

```
import Build_Subsystem_PrePostRun

try:
                Build_Subsystem_PrePostRun.HandleJobStatus__postRun(_nodeExecDir)
except NameError as e:
                scriptCommands.abort(str(e))

reviewed_ok='ok'
```

Creation of “Geometry treatment” node

In the same way as before, create one more application ANSA node as shown below.



For the Pre and Post Run Scripts refer to the next slide

Geometry treatment - Input Slots

Set Geometry treatment's Input Slots property

- start_subsystem_built_in
- BuildSubsystemScripts
- process_script
- build_actions_script

Input Slots:

i. type **String**

ii. type **Single File** in **DM Folder**, pointing to script “BuildSubsystemScripts.py”

iii. type **Single File** in **DM Folder**, pointing to script “Build_Subsystem_PrePostRun.py”

iv. type **Single File** in **DM Folder**, pointing to script “BuildSubsystemScriptsBuildActions.py”

Geometry treatment - Output Slots

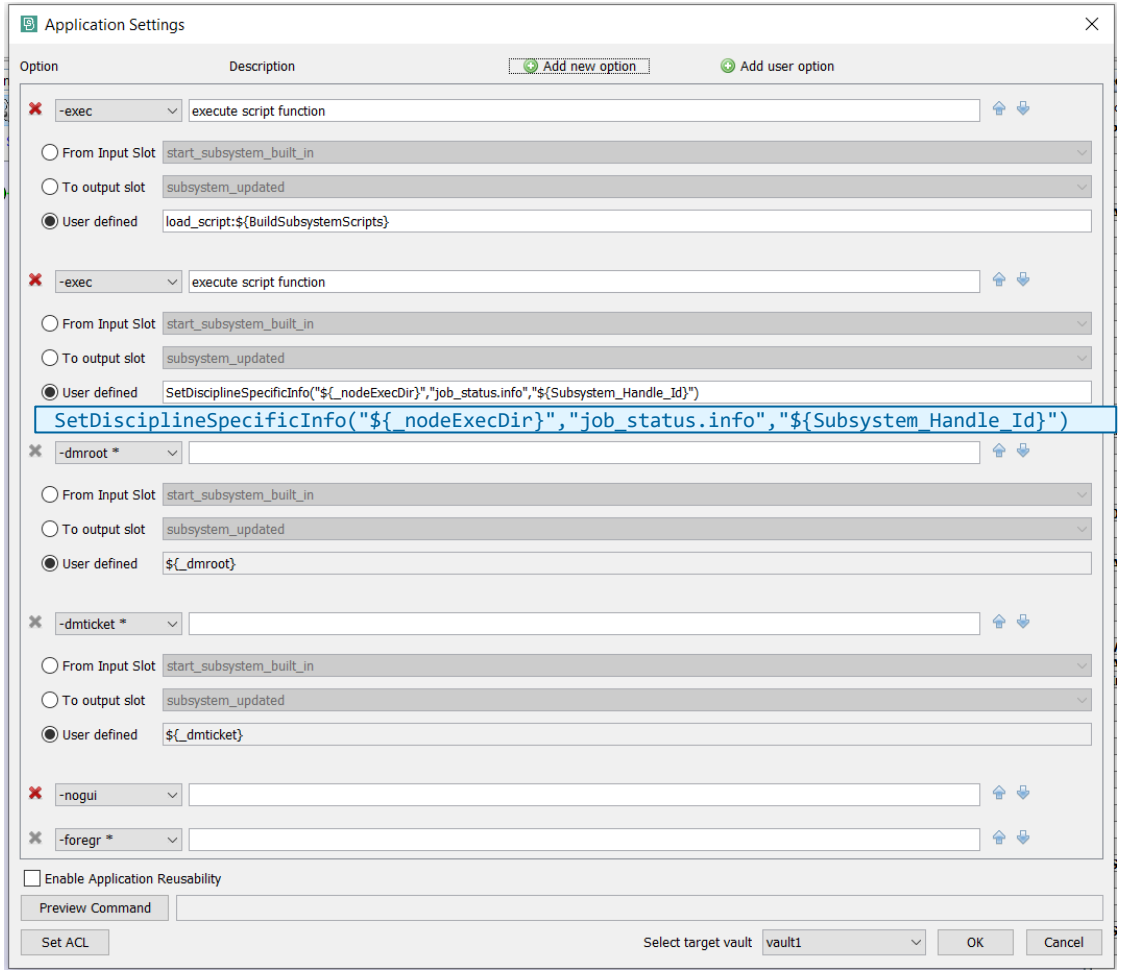
Set Geometry treatment's Output Slots property

- subsystem_updated
- geom_failed_parts

Output Slots:

i. type **String**

ii.type **List of Files** (with default settings)



Pre post run scripts Geometry Treatment

Geometry treatment pre-run script

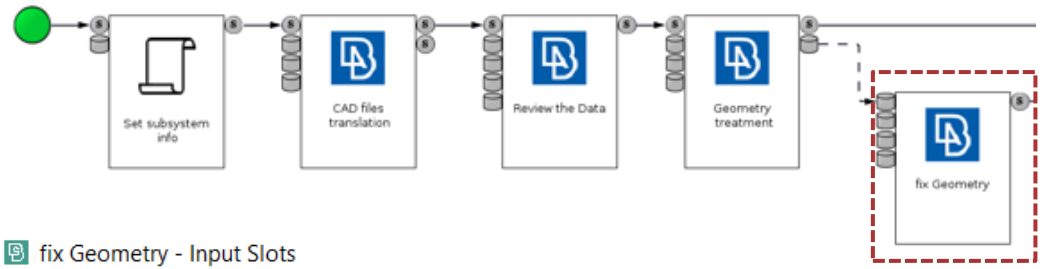
```
process.clearOutput()

import Build_Subsystem_PrePostRun

try:
                Build_Subsystem_PrePostRun.HandleJobStatus__preRun(_nodeExecDir)
except NameError as e:
                scriptCommands.abort(str(e))
```

Geometry treatment post-run script

```
import Build_Subsystem_PrePostRun

try:
                subsystem_updated = Build_Subsystem_PrePostRun.GeometryTreatment__postRun(_nodeExecDir)
except NameError as e:
                scriptCommands.abort(str(e))
```


BETA
SIMULATION SOLUTIONS

Creation of fix Geometry node



Don't forget that with drag & drop operation you can connect nodes by creating the needed slot automatically. This is the case for any type of slot, for **List of files** also.

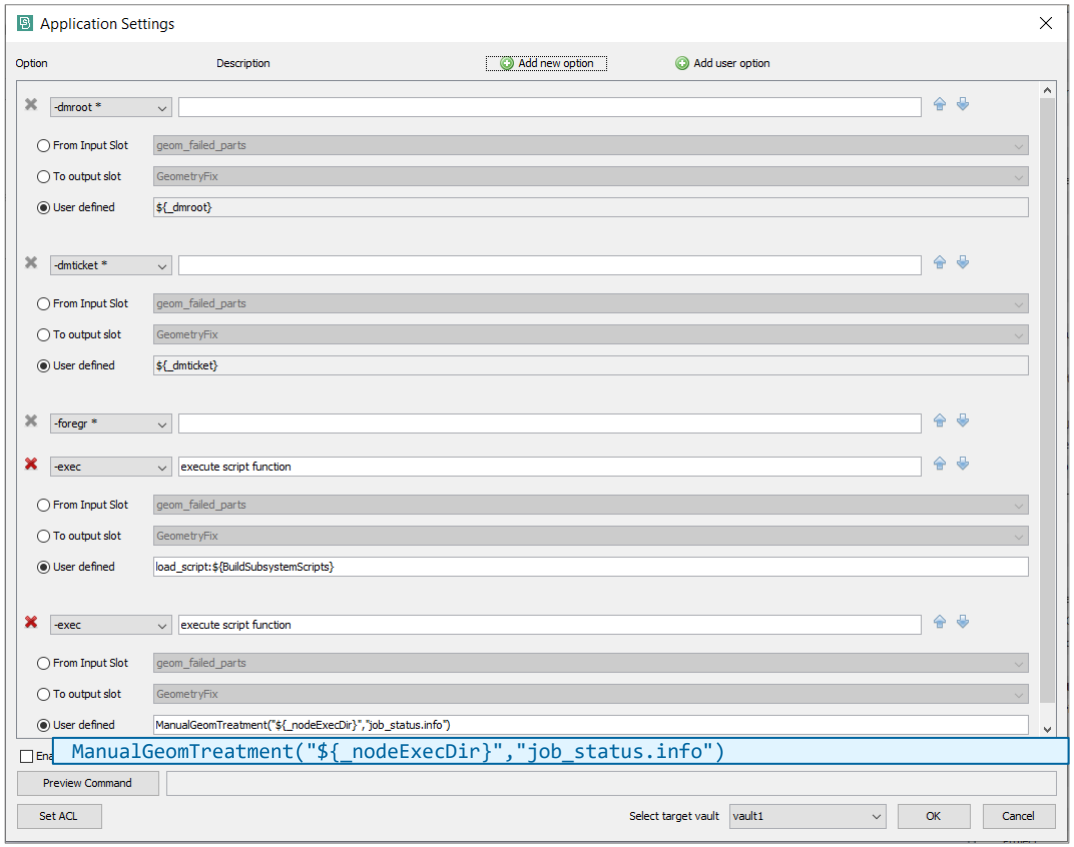
fix Geometry - Input Slots

Set fix Geometry's Input Slots property

- geom_failed_parts
- BuildSubsystemScripts
- process_script
- build_actions_script

Input Slots:

i. "geometry_failed_parts": type **List of Files** from the previous Geometry Treatment Node
ii. type **Single File** in **DM Folder**, pointing to script "BuildSubsystemScripts.py"
iii. type **Single File** in **DM Folder**, pointing to script "Build_Subsystem_PrePostRun.py"
iv. type **Single File** in **DM Folder**, pointing to script "BuildSubsystemScriptsBuildActions.py"



fix Geometry - Output Slots

Set fix Geometry's Output Slots property

- GeometryFix

Output Slots:

i. "GeometryFix": type **String holding the status of** fix Geometry Node

For the Pre and Post Run Scripts refer to the next slide

Pre post run scripts fix Geometry

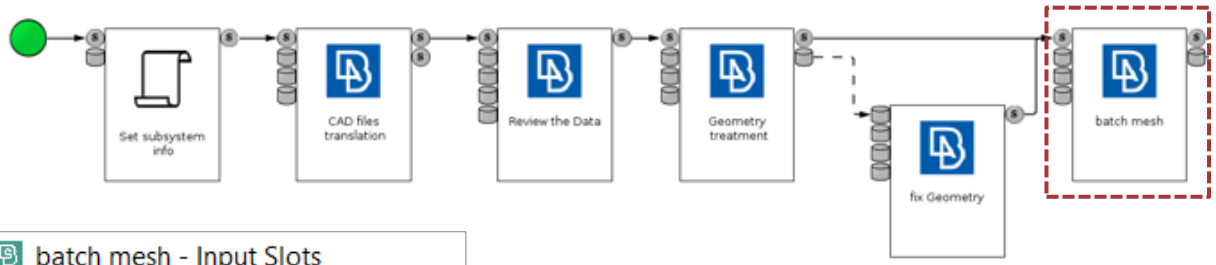
fix Geometry pre-run script

```
process.clearOutput()

import Build_Subsystem_PrePostRun

Build_Subsystem_PrePostRun.HandleJobStatus__preRun(_nodeExecDir)
```

fix Geometry post-run script

```
import Build_Subsystem_PrePostRun

try:
                subs_attrs = Build_Subsystem_PrePostRun.HandleJobStatus__postRun(_nodeExecDir)
except NameError as e:
                scriptCommands.abort(str(e))

GeometryFix = 'ok'
```

Creation of batch mesh node



For the Pre and Post Run Scripts refer to the next slide

batch mesh - Input Slots

Set batch mesh's Input Slots property

- geometry_quality_ok
- BuildSubsystemScripts
- process_script
- build_actions_script

Input Slots:

i. "geometry_quality_ok": type **String**
ii. type **Single File** in **DM Folder**, pointing to script "BuildSubsystemScripts.py"
iii. type **Single File** in **DM Folder**, pointing to script "Build_Subsystem_PrePostRun.py"
iv. type **Single File** in **DM Folder**, pointing to script "BuildSubsystemScriptsBuildActions.py"

Variables:

i. "batch_mesh_output_txt_file": type **String**, default value: "**batch_mesh_output.txt**"

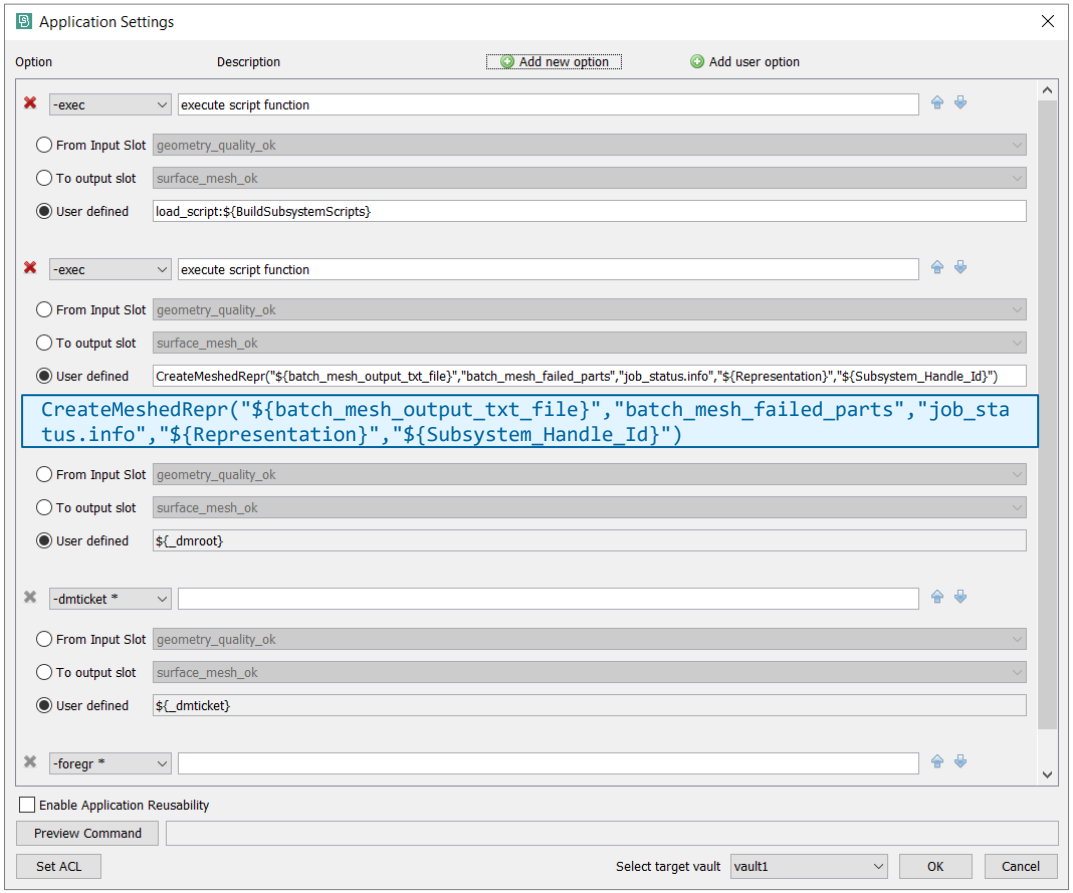
batch mesh - Output Slots

Set batch mesh's Output Slots property

- surface_mesh_ok
- batch_mesh_failed_parts

Output Slots:

i. "surface_mesh_ok": type **String**
ii. "batch_mesh_failed_parts": type **List of Files**



BETA SIMULATION SOLUTIONS

Pre post run scripts Batch Mesh

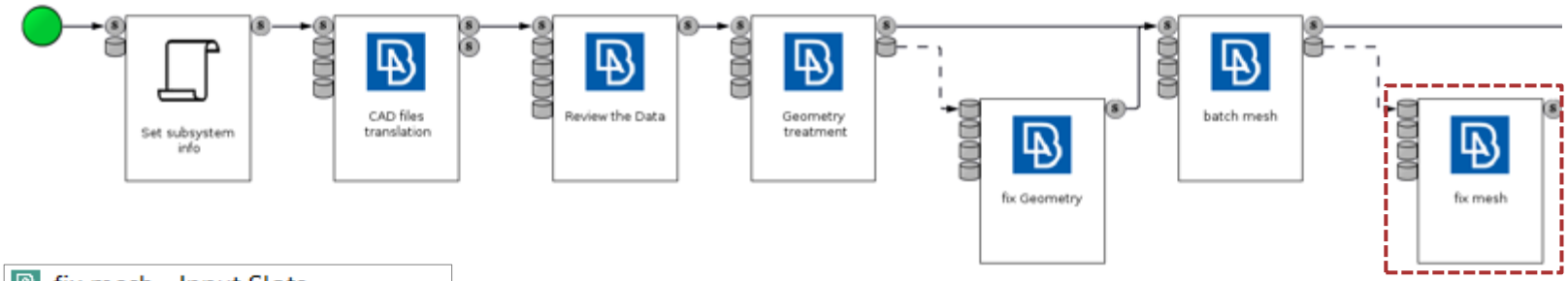
"batch mesh" pre-run script

```
process.clearOutput()

import Build_Subsystem_PrePostRun

Build_Subsystem_PrePostRun.HandleJobStatus__preRun(_nodeExecDir)
```

"batch mesh" post-run script

```
import Build_Subsystem_PrePostRun

try:
                surface_mesh_ok = Build_Subsystem_PrePostRun.BatchMesh__postRun(_nodeExecDir, batch_mesh_output_txt_file)
except NameError as e:
                scriptCommands.abort(str(e))
```

Creation of fix mesh



fix mesh - Input Slots

Set fix mesh's Input Slots property

- batch_mesh_failed_parts
- BuildSubsystemScripts
- process_script
- build_actions_script

Input Slots:

i. batch_mesh_failed_parts: type **List of Files**, as shown in image on the right (create it automatically by connecting the nodes)

ii. type **Single File** in **DM Folder**, pointing to script "BuildSubsystemScripts.py"

iii. type **Single File** in **DM Folder**, pointing to script "Build_Subsystem_PrePostRun.py"

iv. type **Single File** in **DM Folder**, pointing to script "BuildSubsystemScriptsBuildActions.py"

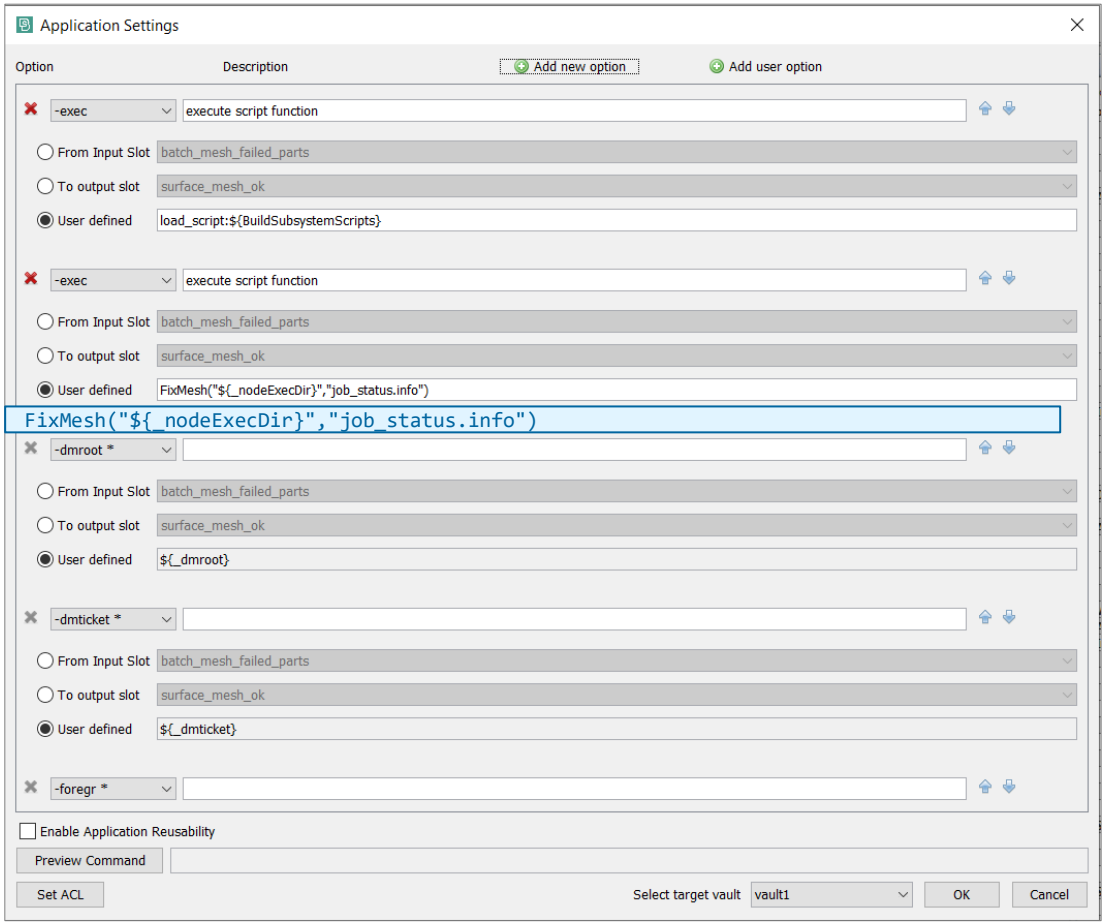
fix mesh - Output Slots

Set fix mesh's Output Slots property

- surface_mesh_ok

Output Slots:

i. "surface_mesh_ok": type **String**



Pre post run scripts fix mesh

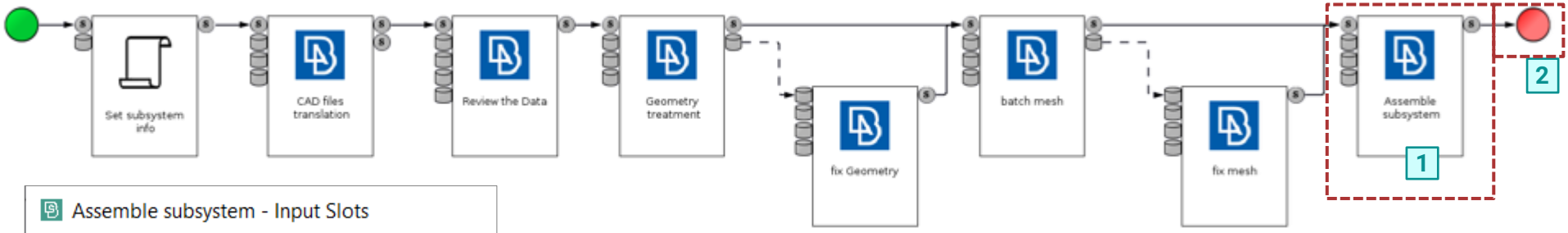
"fix mesh" pre-run script

```
process.clearOutput()

import Build_Subsystem_PrePostRun

Build_Subsystem_PrePostRun.HandleJobStatus__preRun(_nodeExecDir)
```

"fix mesh" post-run script

```
import Build_Subsystem_PrePostRun

try:
                surface_mesh_ok = Build_Subsystem_PrePostRun.BatchMesh__postRun(_nodeExecDir, batch_mesh_output_txt_file)
except NameError as e:
                scriptCommands.abort(str(e))
```

Creation of “Assemble subsystem” and End nodes



Assemble subsystem - Input Slots

Set Assemble subsystem's Input Slots property

- input_1
- BuildSubsystemScripts
- process_script
- build_actions_script

Input Slots:
i. “input_1”: type **String**
ii. “BuildSubsystemScripts” : type **Single File** in **DM Folder**, pointing to script BuildSubsystemScripts.py
iii. “process_script”: type **Single File** in **DM Folder**, pointing to script Build_Subsystem_PrePostRun.py
iv. “build_actions_script”: type **Single File** in **DM Folder**, pointing to script BuildSubsystemScriptsBuildActions.py

Assemble subsystem - Output Slots

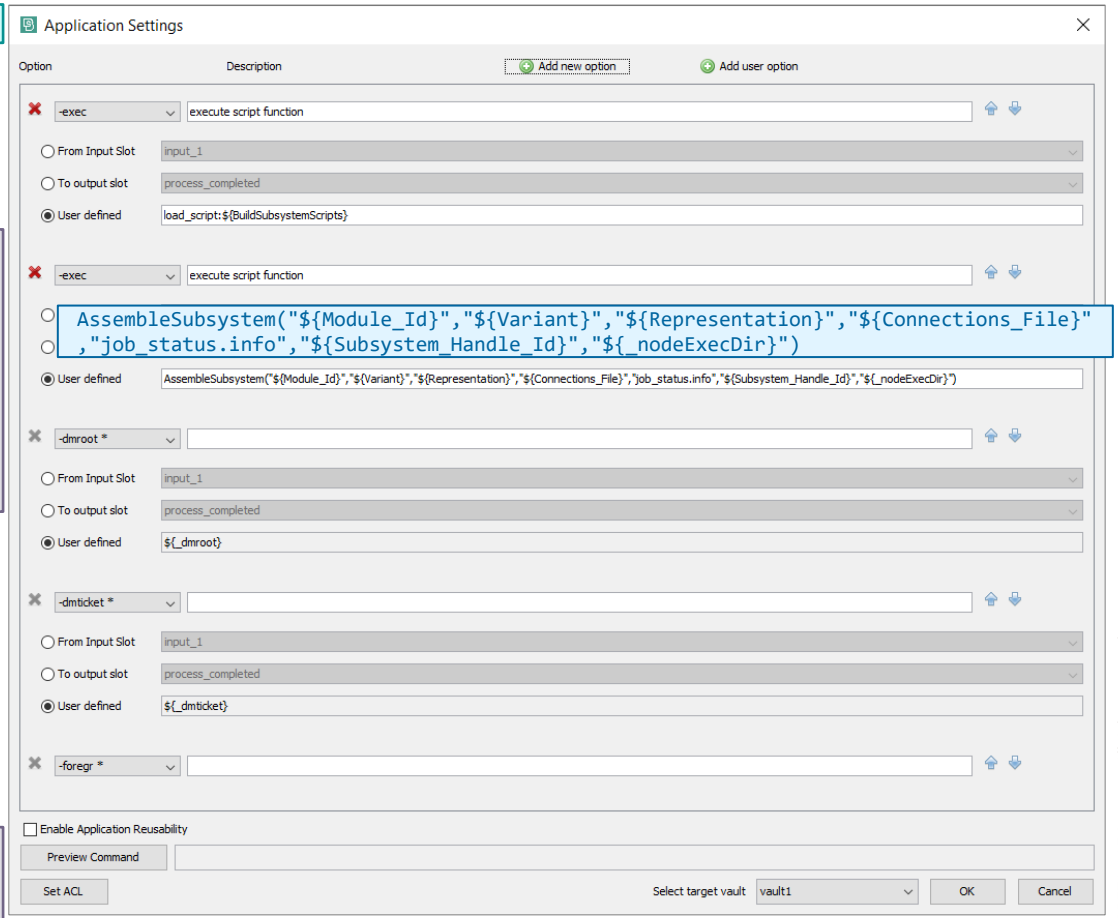
Set Assemble subsystem's Output Slots property

- process_completed

Output Slots:
i. “process_completed”: type **String**

2 Drag an End Node from the Palette window to the Workflow scene and connect it with the Assemble subsystem node.
Save the process as **Definition** to create a new version.

1



Pre and Post run scripts of “Assemble subsystem”

Assemble subsystem pre-run script

```
process.clearOutput()

import Build_Subsystem_PrePostRun

try:
                Build_Subsystem_PrePostRun.AssembleSubsystem__preRun(_nodeExecDir, Representation)
except NameError as e:
                scriptCommands.abort(str(e))
```

Assemble subsystem post-run script

```
import Build_Subsystem_PrePostRun

try:
                subs_attrs = Build_Subsystem_PrePostRun.HandleJobStatus__postRun(_nodeExecDir)
except NameError as e:
                scriptCommands.abort(str(e))

process_completed = 'ok'
```