BETA
SIMULATION SOLUTIONS

physics on screen

Data Management
Reference Manual

SPDRM
WORKFLOW MANAGER

KOMVOS
SDM CONSOLE

ANSA
PRE PROCESSOR

META
POST PROCESSOR

Table of Contents

1. Introduction

Simulation within a virtual product development environment enables engineers to predict the real-world behavior of the products and improve the design to satisfy a broader range of requirements. Throughout this process, several different teams work with a multitude of software applications which produce a huge amount of data that finally need to be presented in a way that enables the engineers to take informed decisions and improve the product design in a timely manner.

Usually, the author of each file knows everything about it: What it is, which project it relates to, what was the input used to create it, when it was handled, which application can be used to modify it, what was the intention behind the work done with it, etc. However, as each file only describes a fragment of the final product at only a snapshot of the product development time and each user is only a member of one of the various CAE teams -which are also often geographically dispersed, being able to get information on data should not require having direct access to their author.

Data Management in CAE offers a systematic approach for data storage that enables all stakeholders to find the right data easily, identify their dependencies and browse their pedigree. With a Data Management system in place, the CAE users benefit from a common data share point that facilitates collaboration and data reuse, and greatly improves the efficiency of teams and individuals. The systems that deal with the management of CAE data are collectively referred to with the term Simulation Data Management (SDM) systems.

Process Management capabilities can extend the feature list of SDM systems by standardizing and automating frequent, repetitive, complex and time-consuming tasks (e.g. CAD files translation, meshing, run setup, HPC submission, post-processing and report generation etc.). The systems that deal with the management of CAE data together with the processes that produce or consume them are referred to with the term Simulation Process and Data Management (SPDM) systems.

BETA CAE System offers solutions for Simulation Data and Process Management that make up a complete CAE framework with the smooth integration with ANSA and META. This document will focus solely on Data Management aspects and will attempt to become the primary reference for everything related to SDM.

1.1. Core Data Management concepts

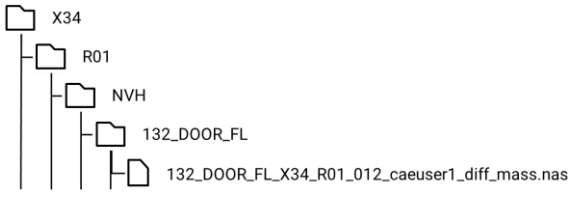
1.1.1. Data objects Vs files

SDM systems store **data objects** rather than files.

- A data object can hold one or more files and also different formats of the same data
- A data object holds metadata: Data about the files (e.g. identity and process-related info, traceability info, system info, etc.)
- A data object is finally identified by a set of metadata and not only by the filename

Let's take for example the Nastran include file of a front left door model used in NVH simulations.

Traditional storage of files without the use of a data management system requires the inclusion of all useful info about the file either in the file name, in the folder structure, or even within the file contents.



An SDM system would manage the data object of the door model that would hold the following information:

Module Id	132_DOOR_FL	Image
Variant	-	image.png 
Project	X34	
Release	R01	
Iteration	012	
Representation	nvh_fe	
File Type	Nastran	
Status	Frozen	
Owner	caeuser1	Part Structure
Creation Date	17-02-2021 12:03:25	hierarchy.xml 
Last Modification Date	18-02-2021 09:28:04	
File	132_DOOR_FL_X34_R01_012.nas	
Comment	Included loudspeaker mass as a lumped mass of 2.85kg	

1.1.2. Types of SDM systems

Now that it's clear that SDM systems deal with data objects rather than plain files, it becomes apparent that data storage needs to facilitate both the storage of the metadata and that of the related files. Metadata is stored in a **database**. The files are stored in physical volumes, usually referred to as **data vaults**.

There are two types of Simulation Data Management systems:

- **File-based** systems: In file-based systems, the files and their metadata are stored in a shared network location that all team members can access. In this case, the database is serverless and self-contained.
- **Server-based** systems: In server-based systems, the data are stored on a server.

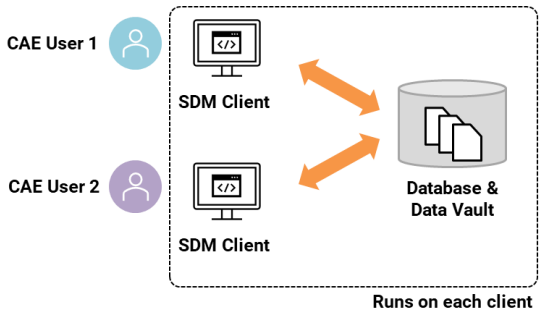
The architecture of file-based systems is fundamentally different from that of server-based systems, as shown in the diagrams below.

File-based systems only require the SDM Client application, which is responsible for data check-in/check-out.

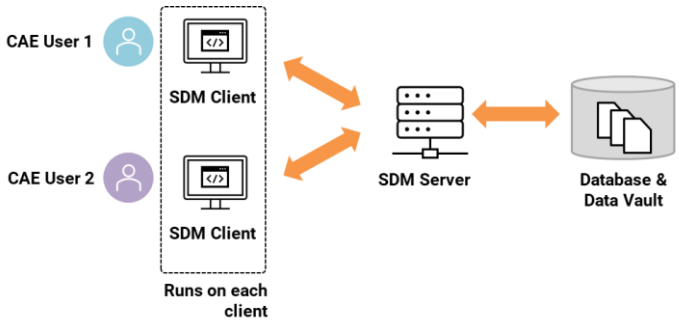
Server-based systems require an additional application, the SDM Server, that is running around the clock and is responsible of data check-in/check-out.

The architecture of the two systems is schematically represented below:

Architecture of a file-based SDM System



Architecture of a server-based SDM System



Depending on application, the recommended type of SDM system may differ. Directions on how to choose the right SDM system for each application are given in paragraph 1.2.

1.1.3. Specifications of SDM systems

SDM systems need to satisfy certain functional and non-functional requirements. Some of the most common functional requirements are:

- Structured storage of files and metadata.
- Version control, on part, subsystem and system level and for all library data.
- Management of model and loadcase variants.
- Management of FE-representations, on part and subsystem level.
- Structured storage of key-results in association with the simulation runs.
- Data traceability.
- Lifecycle Management.

Non-functional requirements include:

- Concurrency and capacity
- Scalability
- Security
- Performance
- Reliability
- Maintainability
- Portability
- Ease of set-up

All functional requirements listed above can be satisfied by both file-based and server-based SDM systems. However, file-based systems cannot satisfy all non-functional requirements. The table below provides some more information on the suitability of file-based and server-based solutions with respect to non-functional requirements of the application:

Requirement	Preferred solution	More info
Concurrency and capacity	Server-based	High concurrency can only be achieved on server-based SDM systems. File-based systems support concurrency but with limited capacity, as performance drops considerably.
Scalability	Server-based	Server-based solutions are a better choice when it comes to scalability in terms of database queries required
Security	Server-based	Data security and user access control is only possible on server-based systems
Performance	Server-based	Server-based solutions are designed to offer good performance even under high load
Volume of data	Server-based	File-based solutions are not suitable for large scale applications.
Maintainability	File-based	Maintenance of server-based solutions is generally fast and easy. However, file-based solutions practically require no maintenance at all.
Portability	File-based	In file-based solutions, the entire database is stored as a single file and thus, it is highly portable.
Ease of set-up	File-based	The database of file-based solutions does not require any preparation for its set-up, as it is embedded in the application.

1.2. BETA's SDM solutions

BETA CAE Systems offers its own SDM solutions but also provides out-of-the-box interfaces for the integration of its products with third party SDM systems and test-result servers.

1.2.1. Proprietary SDM solutions

BETA CAE Systems mainly offers two SDM solutions, the one file-based and the other server-based. The table below summarizes the characteristics of the two solutions:

	File-based DM	Server-based DM
Database	SQLite	MySQL or Oracle
Server	-	SPDRM Server
Client (desktop)	KOMVOS	
Client (embedded in BETA Suite Apps)	ANSA & META	

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

As shown in the table above, the software applications used as Clients are the same for both file-based and server-based solutions. Their main characteristics are described below:

KOMVOS: This is a standalone, desktop application. It is the front-end used for browsing and managing data and processes. Through KOMVOS the users can search for data, review product structures, preview models in the embedded 3D viewer, send data to external applications, export data, review processed results, design and execute processes, get notifications on completed HPC jobs, and many more.

ANSA & META: These are the flagship pre- and post-processor of BETA CAE systems. Both these applications are seamlessly integrated with the SDM systems. This integration, that is available out-of-the-box, enables direct utilization of data and process management functionality right where it's needed: Within the pre- and post-processing tools that produce and consume most of the simulation data.

1.2.2. Interfaces to third party systems

All client applications (KOMVOS and ANSA/META) include out-of-the-box interfaces that enable the use of third-party applications as data back-ends. Right now, two options are available:

Interface to MSC SimManager: For the teams that already make use of MSC's SimManager as an Enterprise SDM system, the built-in interface enables browsing, direct check-in and check-out of data through KOMVOS and ANSA/META.

Interface to Test Results servers according to the ASAM-ODS standard: For direct access to test data like Channels, Photos, Movies, 3D Measurements etc. that are mostly needed for correlation between simulation and test results, the built-in interface enables browsing, direct check-in and checkout of data through KOMVOS and ANSA/META.

1.3. Managed data

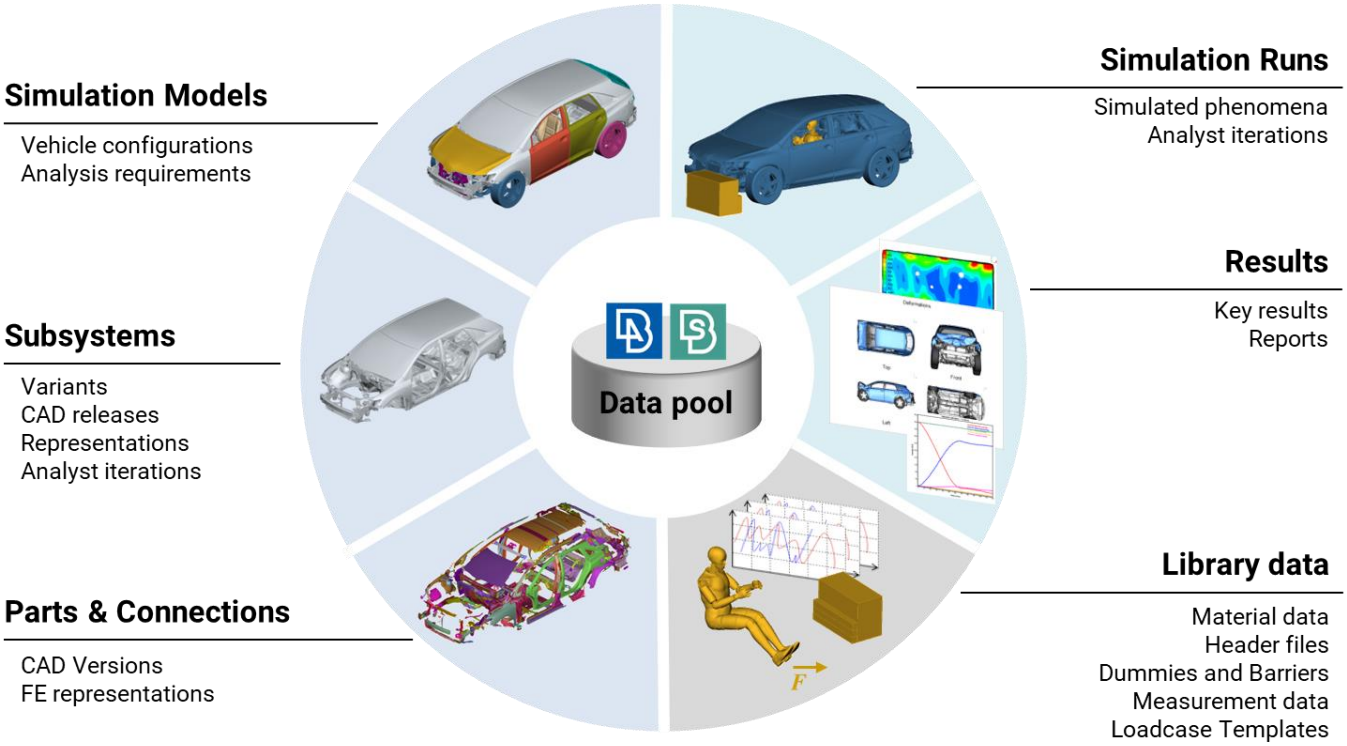
The data managed by BETA's SDM solutions are grouped in two categories:

- **DM Objects**: Data objects as described in paragraph 1.1.1. The supported data objects, as well as their definition in terms of metadata, is part of the SDM system's data model (see paragraph 5.1).
- **Files**: Plain files in the SDM system do not have any definition metadata and are merely identified by their DM path (location in the DM data tree and file name). The only metadata available for files are the System Attributes

From the business perspective, these data can be classified into:

- **Model data**: Parts, Subsystems and Simulation Models
- **Simulation data**: Simulation Runs, key-results and reports
- **Library Items**: Loadcase Templates, header files, control cards, crash dummies, barriers, template loads, etc.
- **ANSA Templates**: Meshing settings, connection templates, custom connector and mass trimming FE-representations, etc.

A more elaborate representation of this classification is given in the image below:



In this classification, all categories are managed by the SDM systems as **DM Objects**. **Plain files** are only used for some Library data.

Other than the management of data through the primary SDM Client, **KOMVOS**, the users can manage these DM Objects directly through ANSA and META. More specifically:

- **Within ANSA**: Create and manage Model data, Simulation data, Library Items and Templates and save them directly to the SDM system. Furthermore, check for updates and include them in the model, identify alternative FE-representations, etc.
- **Within META**: Download Simulation Runs and their raw solver results, produce key-results and reports and store them under the corresponding Simulation Runs, etc.

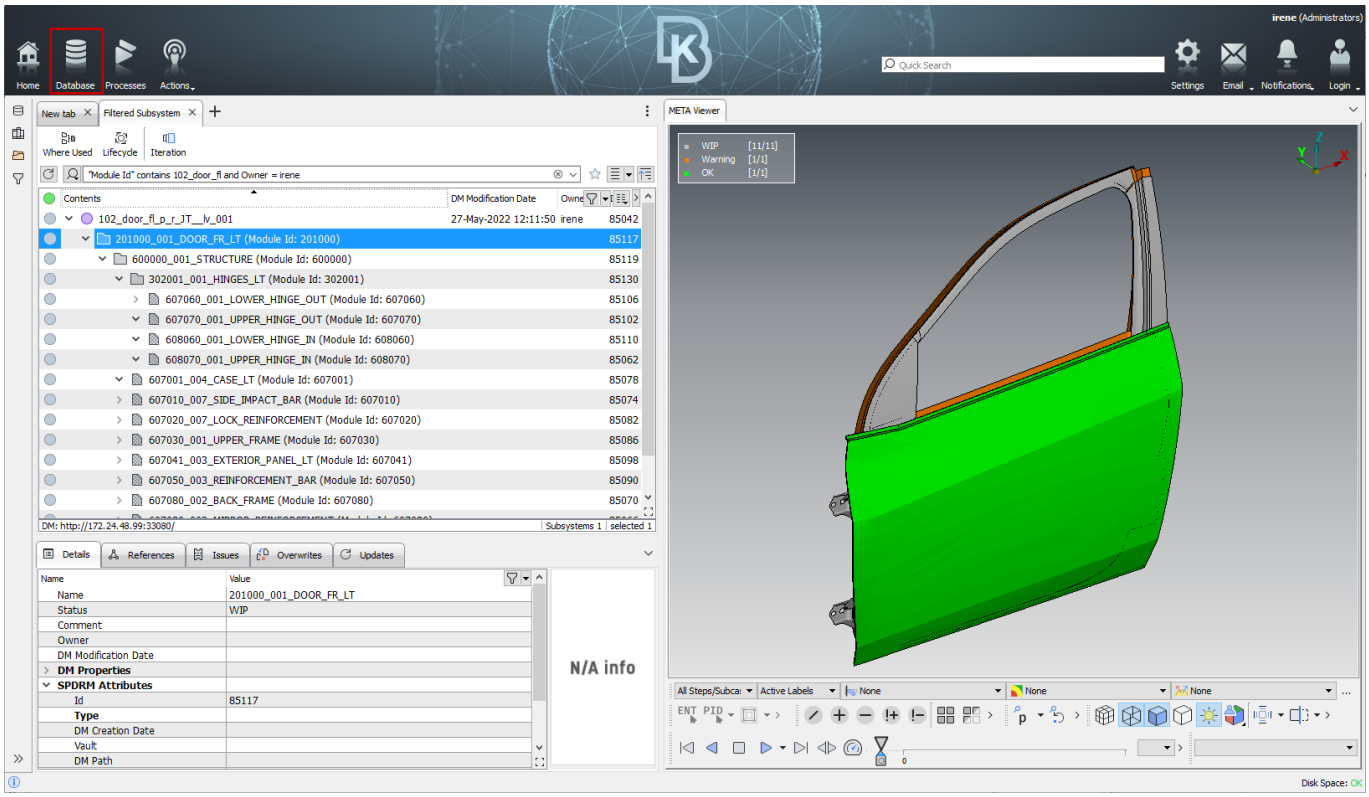
1.4. Basic data browsing tools

BETA's SDM solutions include powerful tools that facilitate data management operations like data check-in and check-out, data search, navigation of data relationships, comparison, editing, etc. These tools are available in KOMVOS but also in ANSA and META. They are built on the same GUI framework and are based upon the same user interface principles, making it extremely easy for a user familiar with one, to get immediately productive with the other.

Chapter 3 of this guide discusses core data management functionality made available through the data browsing tools.

1.4.1. Data Manager in KOMVOS

KOMVOS, the standalone desktop rich client for Simulation Process and Data Management, offers the *Database*, a dedicated workspace for data management. In the Database workspace the users can search for data, review product structures, preview models in the embedded 3D viewer, send data to external applications, export data, review processed results, and many more.

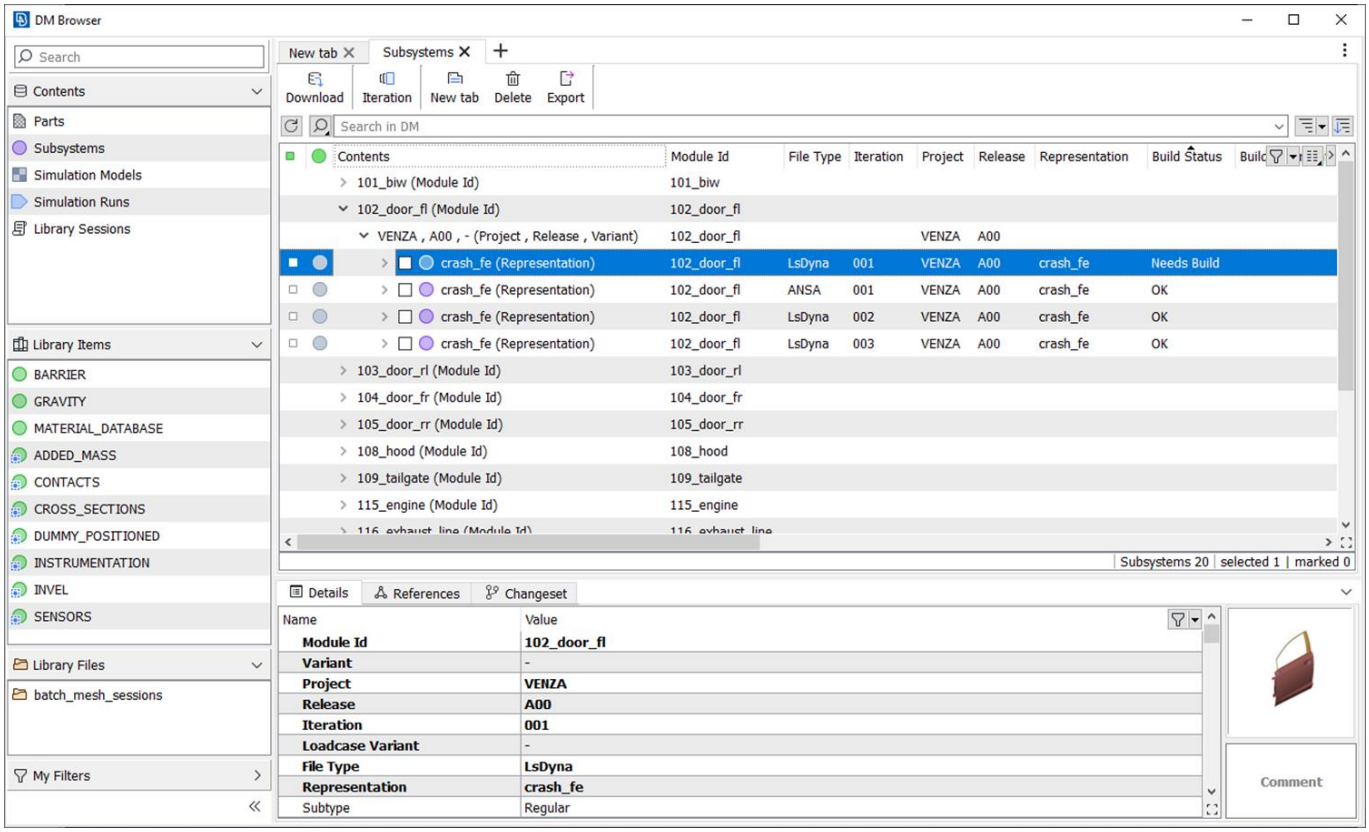


More information on the Database workspace of KOMVOS and the various functionalities offered can be found in the KOMVOS User Guide.

1.4.2. DM Browser in ANSA/META

The *DM Browser* is the workspace for browsing the data management repository through ANSA and META. Through this tool the users can search and find data, load them in the running session or export them in a directory on the disk, explore data relationships, compare different versions/variants of data, review simulation run results etc.

The DM Browser is available in both ANSA and META and it is invoked through **Containers > DM > DM Browser**.



More information on the DM Browser can be found in the ANSA User Guide.

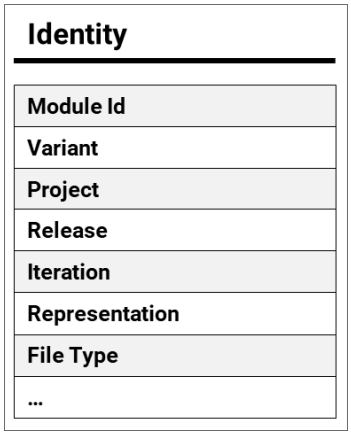
2. Key-features

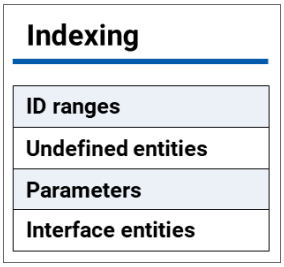
The table below lists the key-features of BETA's SDM Solutions with information on their availability in file-based DM and SPDRM. More information on each topic is given in the respective paragraph of this chapter.

Feature	File-based DM	SPDRM
Metadata	✓	✓
Version control	✓	✓
Lifecycle Management	✓	✓
Traceability	✓	✓
Search	✓	✓
Security	✗	✓
File access through the file system	✓	✓
Data deletion / purging	✗	✓
Data migration	✓	✓
Client-side file caching	✗	✓
Interfaces	✓	✓

2.1. Metadata

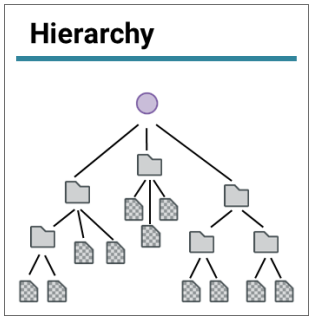
The term metadata refers to data that are used to describe other data. For a Data Management system, metadata consists of information about the data files and is expressed through Attributes. A combination of Attributes and attached files makes up a DM Object. The structure and relationships of DM Objects is described in the *data model*.





Among the attributes, some are mandatory for the definition of the DM Object in the database and they need to be filled-in by the user before data check-in. Some others, like the Owner and the Creation Time, are assigned by the system automatically when the DM Object is created. Indexing attributes reveal information on the file content.

Even the internal tree structure (hierarchy) of a DM Object is a particular type of metadata. For a Subsystem for example, its tree structure consists of groups and parts.



With a data management system in place, it is possible to identify DM Objects at any time based on their metadata. So, metadata is an enabler for well targeted data searches.

2.1.1. Types of metadata

The metadata of DM Objects can be grouped in four categories:

Primary attributes: These metadata are the unique identity of a DM Object in the database. For example, the Module Id, the Version and the Representation of a Part are among the metadata that need to be filled before saving a Part in the data repository. Similarly, the Module Id, the Project, the CAD Release and the Variant of a Subsystem are among the metadata that need to be filled before saving a Subsystem. A DM Object with a particular set of primary attributes is unique in the database.

Secondary attributes: These metadata are used to provide more information for the DM Object and are readily available for a user to fill-in every time a DM Object is about to be added to the data repository. They are primarily used for searching data. For example, the Status of a Subsystem or a Part is a secondary attribute.

Both the Primary and the Secondary attributes are part of the DM Object Type definition, in the SDM System's data model.

Additional attributes: All the attributes that are added by the user through the client application on top of the primary and secondary attributes, fall in this category. Such additional attributes are added by ANSA Save actions or by META "add DM Object" functions, the moment a new DM Object is created. Some examples of additional attributes added automatically by ANSA on Save include the mass, inertia tensor, center of gravity or indexing metadata, like for instance the id ranges of the entities contained in the saved file or the names and locations of interface points.

In ANSA, additional attributes are defined with the aid of User Attributes. Defining a new User Attribute on an ANSA Type that corresponds to a DM Object (e.g. on an ANSAPART), and marking it as "Save in DM: YES", will push this attribute and its current value in DM during save.

System attributes: These include all those attributes that are added by the system's Data Manager, the moment a new DM Object is created. Examples of such attributes are the DM Creation Date, which stands for the time the DM Object was added in the database or its File Size.

In order to identify a specific DM Object in the database, the user should either provide the full set of its primary attributes or its **server id**. The server id is a System Attribute that holds the unique identifying key of a DM Object in the database. Server ids are:

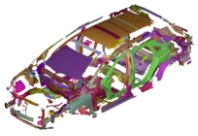
- Integers, for SPDRM back-end
- Integers, for plain file-based DMs
- Alphanumeric strings, for clustered file-based DMs

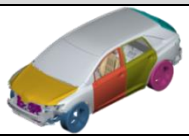
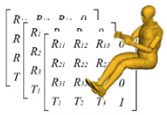
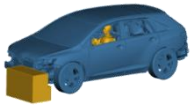
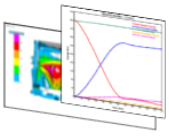
Important note!

The selection of the Secondary and Additional attributes of an SDM system is a crucial step, as it has an impact on the overall performance of data operations. A huge number of unnecessary Additional attributes could lead to unnecessarily big databases and data transactions, which could hamper the efficiency of searches and import/export actions.

2.1.2. Default DM Objects and their Primary Attributes

BETA's SDM solutions come with a default data model which defines the main DM Object types needed for the management of simulation data. The table below describes the key characteristics of these DM Objects. It must be noted that the default data model can be easily modified and adapted to the needs of any engineering team, as described thoroughly in chapter 5 of this document.

Low-level Model Data		
Parts **Groups**		The smallest model unit, equivalent to a CAD part.
Primaries	***Module Id***	*The part number, as defined in the CAD/PDM system. This attribute is also used for the reference of parts in the connectivity of Connection Entities (spotwelds, adhesives, etc.)*
	Version	*The CAD Version*
	Representation	*An identifier that characterizes the content of the part file. Commonly used values include translated_cad, common, 5mm, 8mm, etc.*
	Study Version	*The CAE version of the part. This field is rarely used in typical applications*
	File Type	*The file format of the main representation file associated with the Part DM Object. It can have a value among "ANSA", or any of the supported solver formats, e.g., "Nastran", "LsDyna", etc.*
Subsystems **Subsystem Groups**		A sub-assembly of the model that evolves together with the design and is finally saved as an individual solver file (include file). Groups of Subsystems can also be used to describe nested structures. A Subsystem contains a hierarchy of Groups and Parts, whereas the Subsystem Group contains Subsystems.
Primaries	***Module Id***	*An alphanumeric descriptor, usually a characteristic name describing the subassembly, e.g., 102_fl_door. This attribute is used in the connectivity fields of assembly entities.*
	Variant	*A value representative of the variant of the Subsystem, e.g., the short or the long version of a BiW.*
	Project	*The Project of the model.*
	Release	*The Release of the model, characteristic of the CAD Release / Milestone.*
	Iteration	*The CAE version of the Subsystem. Every new study of a Subsystem is stored as a new Iteration. By default, it follows a 3-digit VERSIONING SCHEME COUNTER scheme (e.g. 001, 002, etc.).*
	Loadcase Variant	*The loadcase-driven variation of the Subsystem. Used for Subsystems affected by the Loadcase, e.g. a seat that may use different node positions for different crash/safety loadcases, may need to use the loadcase code in this field*
	File Type	*The file format of the main representation file associated with the Subsystem DM Object. It can have a value among "ANSA", or any of the supported solver formats, e.g., "Nastran", "LsDyna", etc.*
	Representation	*An identifier that characterizes the content of the Subsystem. By default, it takes value from a set of accepted values, e.g., crash_fe, nvh_fe, dura_fe, lumped_mass, etc.*

Simulation Data		
Simulation Model		The full model assembly that consists of the assembled subsystems. A Simulation Model contains Subsystems and Library Items.
Primaries		
	Discipline	*The Discipline for which the Simulation Model is built. Default accepted values are: crash, nvh, durability, cfd.*
	Model Id	*A characteristic value that describes the content of the Simulation Model (e.g. nvh_assembly, cavity or, trim_body for Discipline NVH and crash_assembly, pedestrian_assembly, for Discipline crash)*
	Model Variant	*The Variant of the Simulation Model. This value represents a particular combination of the variants of the contained Subsystems, e.g., LHD_PHEV_4WD*
	Project	*The Project of the model.*
	Release	*The Release of the model, characteristic of the CAD Release / Milestone.*
	Iteration	*The CAE version of the Model. Every new study of a Model is stored as a new Iteration. By default, it follows a 3-digit VERSIONING SCHEME COUNTER scheme (e.g. 001, 002, etc.).*
	File Type	*The file format of the main representation file associated with the Sim.Model DM Object. It can have a value among "ANSA", or any of the supported solver formats, e.g., "Nastran", "LsDyna", etc.*
	File	*The main file of the Simulation Model.*
Loadcase		The loading scenario of a particular Simulation Model (adapted Loadcase). It contains Subsystems and Library Items. It may be derived from a Loadcase Template library item and therefore, must have identical Primaries with that item (excluding the Sim.Model)
Primaries		
	Loadcase Id	*A value to describe the type of this loadcase scenario, e.g., F1_NCAP, 214_Pole_5th, head-GTR9, Road_Noise, etc.*
	Iteration	*The version of the Loadcase. Every new study of a Loadcase is stored as a new Iteration. By default, it follows a 2-digit VERSIONING SCHEME COUNTER scheme (e.g. 01, 02, etc.)*
	Simulation_Model	*The Simulation Model on which this Loadcase will be applied.*
	File Type	*The file format of the main representation file associated with the Loadcase DM Object. It can have a value among "ANSA", or any of the supported solver formats, e.g., "Nastran", "LsDyna", etc.*
	File	*The main file of the Loadcase.*
Simulation Run		The ready-to-run model, i.e., the main file of the Simulation to be submitted to the solver. It contains one Loadcase, one Simulation Model and one or more Library Items.
Primaries		
	Iteration	*The version of the Run. Every new study of a Run is stored as a new Iteration. By default, it follows a 2-digit VERSIONING SCHEME COUNTER scheme (e.g. 01, 02, etc.).*
	Simulation_Model	*The DM Object of the Simulation Model used in this Run.*
	LoadCase	*The DM Object of the Loadcase used in this Run.*
	File Type	*The file format of the main representation file associated with the Run DM Object. It can have a value among "ANSA", or any of the supported solver formats, e.g., "Nastran", "LsDyna", etc.*
	File	*The main file of the Simulation Run.*
Report		The processed results, produced by the post-processing process.
Primaries		
	Type	*The type of the Report with predefined values, e.g., Image, Video, Curve, MetaProject, etc.*
	Name	*The name of the Report.*
	Simulation_Run	*The DM Object of the Simulation Run that this Report relates to. Only filled if the Report is attached under a Run.*
	LoadCase	*The DM Object of the Loadcase that this Report relates to. Only filled if the Report is attached under a Loadcase.*
	Simulation_Model	*The DM Object of the Simulation Model that this Report relates to. Only filled if the Report is attached under a Model.*
	File	*The main file of the Report (may remain blank for Types that don't require a file, e.g. KeyValue)*

Low-level Library Data		
	Library File	Files like barriers, crash dummies, control cards, headers, material files etc., that can be used in different Simulation Models, Loadcases and Simulation Runs.
Primaries	***Name***	*The name of the Library File*
	Type	*The type of this Library file, among a set of accepted values, e.g., BARRIER, HEADER, MATERIAL_DATABASE, etc.*
	Sub-Type	*Depending on the selected Type, optionally, some sub-type that describe better the file and its use.*
	Version	*The version of the Library File. By default, it follows a 2-digit VERSIONING SCHEME COUNTER scheme (e.g.01, 02, etc.)*
	File	*The actual file.*
	Loadcase File	Files like positioned dummies, added mass for curb weight, cross sections, etc., that are already adapted for use with a particular version/variant of the model and/or the Loadcase. Comparing to Library Files, their identity also holds characteristics of the model.
Primaries	***Name***	*The name of the Loadcase File*
	Type	*The type of this Loadcase file, among a set of accepted values, e.g., ADDED_MASS, CONTACTS, INVEL, etc.*
	Sub-Type	*Depending on the selected Type, optionally, some sub-type that describe better the file and its use.*
	Project	*The Project of the model this Loadcase File is suitable for.*
	Release	*The Release of the model this Loadcase File is suitable for.*
	Model Variant	*The Variant of the model this Loadcase File is suitable for.*
	Version	*The version of the Loadcase File. By default, it follows a 2-digit VERSIONING SCHEME COUNTER scheme (e.g.01, 02, etc.)*
	File	*The actual file.*
	Loadcase Header	Holds a Loadcase Assistant entity that is used as a generator of Nastran, Abaqus and Permas headers.
Primaries	***Name***	*The name of the Loadcase Header*
	Project	*The Project of the model this Loadcase File is suitable for.*
	Release	*The Release of the model this Loadcase File is suitable for.*
	Model Variant	*The Variant of the model this Loadcase File is suitable for.*
	Loadcase Variant	*The loadcase variation.*
	Iteration	*The version of the Loadcase Header. By default, it follows a 2-digit VERSIONING SCHEME COUNTER scheme (e.g.01, 02, etc.)*
	File	*The actual file.*
	Session	Meta-post sessions for producing post-processing results.
Primaries	***Name***	*The name of the Session*
	Loadcase	*The DM Object of the Loadcase that this Session relates to. Only filled if the Report is attached under a Loadcase.*
	Simulation_Run	*The DM Object of the Simulation Run that this Session relates to. Only filled if the Report is attached under a Run.*
	Simulation_Model	*The DM Object of the Simulation Model that this Session relates to. Only filled if the Report is attached under a Model.*
	File	*The actual file.*
	Display style	A DM Object that contains all display settings relative to a META 2D window, e.g., plots layout, plot types, plots ranges, curves line styles, fonts and colors, etc.
Primaries	***Name***	*The name of the Display style.*
	Discipline	*The Discipline for which the item is used. Default accepted values are: crash, nvh, durability, cfd.*
	Version	*The version of the Display Style. By default, it follows a 2-digit VERSIONING SCHEME COUNTER scheme (e.g.01, 02, etc.)*
	File	*The actual file.*

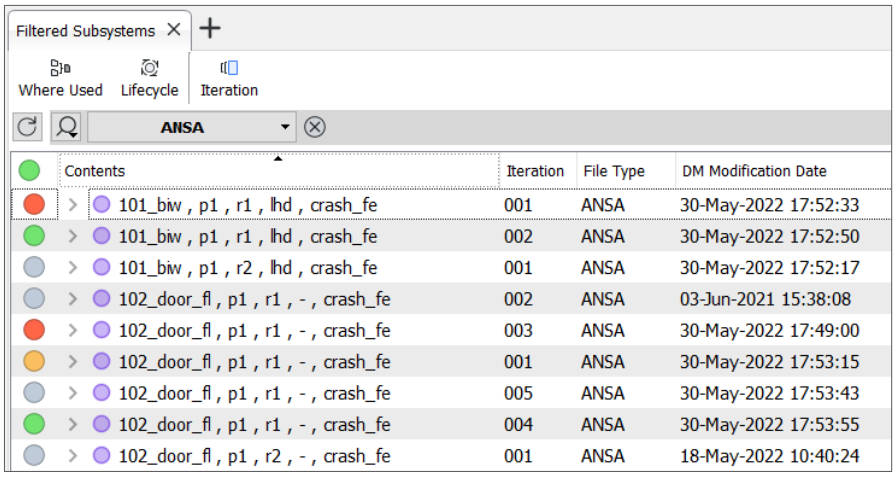
Higher-level Library Data		
	Loadcase Template	The un-adapted loading scenario saved as Library Item, to be used in future Simulation Runs. It consists of several library files and optionally subsystems. When it is adapted to a particular Simulation Model it is transformed into a Loadcase item and should therefore have identical Primaries with that DM Object (excluding the Simulation_Model)
Primaries	***Name***	*The name of the template*
	Loadcase_Id	*A value to describe the type of loadcase that this template will be used for.*
	Iteration	*The version of this template. By default, it follows a 2-digit VERSIONING SCHEME COUNTER scheme (e.g.01, 02, etc.)*
	File Type	*The file format of the main representation file associated with the Loadcase Template DM Object. It can have a value among "ANSA", or any of the supported solver formats, e.g., "Nastran", "LsDyna", etc.*
	File	*The actual file.*
	Target Points	Holds a Run Generator entity that is used to generate multiple runs based on Target Point entities. It is mainly used for pedestrian protection and occupant interior safety loadcases.
Primaries	***Loadcase Id***	*A value to describe the type of loadcase that this Library Item will be used for (e.g. head-GTR9, head-euncap, upper leg, lower leg, etc.)*
	Project	*The Project of the model this Target Points item is suitable for.*
	Release	*The Release of the model this Target Points item is suitable for.*
	Model Variant	*The Variant of the model this Target Points item is suitable for.*
	Iteration	*The version of the Target Points item. By default, it follows a 2-digit VERSIONING SCHEME COUNTER scheme (e.g.01, 02, etc.)*
	File	*The actual file.*
	Simulation Configuration Table	Templates that describe the composition of compound Model Browser Containers. They can be of three types: Simulation Model, Loadcase, or Simulation Run Configuration Tables and they hold information on the construction of different model variants out of subsystem variants and the combination of Model Variants with Loadcases for each development phase.
Primaries	***Name***	*The name of the template*
	Project	*The Project of the model the template corresponds to.*
	Release	*The Release of the model the template corresponds to.*
	Version	*The version of the template. By default, it follows a 2-digit VERSIONING SCHEME COUNTER scheme (e.g.01, 02, etc.)*
	File	*The actual file.*
Feature Items		
	Fastener **Stamp**	Feature items of the model that are handled by the Feature Manager tool of ANSA and can be saved in DM with different mesh representations. Their properties are auto-filled by ANSA during their recognition and/or during their saving in DM.
Primaries	***Module Id***	*A unique alphanumeric id.*
	Version	*The version of the item relatively to the CAD Version of the Part.*
	Study Version	*The CAE version of the item.*
	File Type	*The file format of the file associated with the item. Default value is "ANSA".*
	Representation	*An identifier that characterizes the content of the item with respect of the applied mesh*

Default Items		
	Predictor	A Predictor is a trained Machine Learning Algorithm created for each of the Responses of a Simulation Model. The task of a Predictor is to predict the Response values taking as input the Design Variable values that we provide.
Primaries	***Output Measure***	*The name of the ANSA or META Response that corresponds to the selected Predictor.*
	Group Name	*The name of a specific group of Predictors.*
	Iteration	*The version of the Predictor.*
	Sub-Type	*The type of Machine Learning algorithms used (DV based predictor or DV based predictor Kriging).*
	File	*The actual file.*
	Optimization Study	The Optimization Study uses an optimization algorithm to find the optimum values of the Design Variables that respect the constraints introduced by the Responses.
Primaries	***Name***	*The name of the Optimization Study.*
	Iteration	*The version of the Optimization Study*
	Algorithm	*The optimization algorithm used (IPOPT or Simulated Annealing).*
	Method	*The method that will be used to extract the Responses values in order to provide them to the optimization algorithm (RSM or DIRECT)*
	DOE Study	A collection of Simulation Run versions produced by different values of Design Variables.
Primaries	***Optimization Task Name***	*The name of the Optimization Task used to create the DOE Study.*
	Iteration	*The version of the DOE Study.*
	File	*The actual file.*
	Modular Environment Profile	A collector for all settings that affect the behavior of Model Containers in ANSA and their save/load parameters. It also contains information for the composition of Model Containers, their preferred file format and their Build Process.
Primaries	***Discipline***	*The Discipline for which the profile is used. Default accepted values are: crash, nvh, durability, cfd.*
	Target Solver	*The solver for which the contained settings are defined.*
	Version	*The version of the profile.*
	File	*The actual file.*
	Build Action	One build action defined in the ANSA Modular Environment settings for one type of Model Containers that fulfills specific criteria. It can be used as one step of one or more Build Processes.
Primaries	***Entity Type***	*The entity type for which the Build Action is applied, e.g., Parts, Subsystems, Simulation_Models, etc.*
	Type	*The type of the action to be executed, one of the: Built_In, Script, Check_Template.*
	Name	*The name of the action.*
	Iteration	*The version of the action.*
	File	*The actual file.*
	Build Process	The build process defined in the ANSA Modular Environment settings for one type of Model Browser Containers that fulfills specific criteria. It consists of several Build Actions.
Primaries	***Entity Type***	*The entity type for which the Build Process is applied, e.g. Parts, Subsystems, Simulation_Models, etc.*
	Name	*The name of the process.*
	Iteration	*The version of the process*
	File	*The actual file.*
	Build Setup	The complete build setup of a type of Model Browser Containers, defined in the ANSA Modular Environment settings. It consists of several Build Processes.
Primaries	***Entity Type***	*The entity type for which the Build Setup is applied, e.g. Parts, Subsystems, Simulation_Models, etc.*
	Discipline	*The Discipline for which the setup is used. Default accepted values are crash, nvh, durability, cfd.*
	Target Solver	*The solver for which the setup is defined.*
	Iteration	*The version of the setup.*
	File	*The actual file.*

2.1.3. DM Object maturity

The DM Object maturity is reflected in the attribute named *Status*. This attribute, with default accepted values **WIP** (Work in Progress), **OK**, **Warning**, **Error**, is the attribute that reveals the maturity state of a DM Object to the DM users. The definition of this attribute can be customized in terms of the different values it may get. For example, an SDM system could be configured to use the scheme **Draft**, **Frozen**, **Valid**, **Error** to represent different phases of the object lifecycle.

The *Status* of the DM Objects is displayed in the data browsing tool with a color icon, as shown below.



This is the default color mapping for the default accepted values:

Status values and color mapping customization are discussed in paragraph 5.1.5.1.

The Status, as the default DM Object maturity indicator, is the base attribute for the definition and set-up of the Lifecycle Management functionality described in paragraph 2.3.

2.2. Version control and conflicts resolution

Every time a new DM Object is about to be imported to the system, a check is performed for conflicts of the incoming item with existing DM Objects. Essentially, with this check the system tries to find an existing DM Object with the same primary attributes with the incoming. In case such a DM Object is found, it is considered a conflict that can be resolved in one of the following ways:

- **Version spin-up**: This conflict resolution method makes use of the versioning attributes of the DM Object, as these are defined in the DM Schema and automatically increments the version of the incoming item to the first available value
- **Overwrite**: This conflict resolution method pushes the file and other Secondary and Additional attributes of the incoming object on the existing DM Object in DM. Overwriting is only possible at early phases of the Lifecycle of an object. SPDRM can keep track of all *snapshots* of the file of objects being overwritten and can potentially restore any of them if required. There's no such possibility in file-based DM, where overwritten data are permanently lost.

2.2.1. Version spin-up

Version spin-up is facilitated by the versioning attributes that can follow one of the following versioning schemes:

- the VERSIONING SCHEME COUNTER and
- the STUDY VERSION.

The spin-up of a VERSIONING SCHEME COUNTER attribute is implemented with incrementation of the current value by one, until the first available is found.

The spin-up of a STUDY VERSION scheme follows a particular logic so that only by seeing the version value, one can unfold the history of the DM Object.

In the default DM Schema, the Part object makes use of the STUDY VERSION scheme in its primary attribute *Study Version*. All the other DM Object types make use of VERSIONING SCHEME COUNTER attributes.

The moment the SDM resolves a conflict with spin-up, a *history* link is generated between the new DM Object and the existing one, facilitating the tracking of the object's lifecycle. The history of a DM Object is graphically represented in the Lifecycle Graph View.

2.2.1.1. Versioning scheme counter

This is the most frequently used versioning scheme, although historically it was the second option offered by BETA's SDM solutions. Usually it appears as a 2- or 3-digit counter (its formatting is configurable through the DM Schema, as described in paragraph 5.1.1.3).

This scheme is mainly used because of its simplicity. Its only drawback is that it's not possible to tell which was the parent version of an item only by seeing the version number. However, that's not really a problem with BETA's SDM solutions, as the origin of an object can always be retrieved through a history link.

2.2.1.2. Study Version scheme

This versioning scheme produces version values like 1.1, 1.3.1.2, etc. Historically, it was the first option offered by BETA's SDM solutions as the versioning scheme used by the Study Version attribute of Parts.

The advantage of this versioning scheme is that only by seeing the version value, one can unfold the history of the DM Object. However, the drawback of this scheme is that it may lead to a very long version text. Considering that the version usually participates in the filename generation rule, the use of this scheme may also lead to very long include file names.

The logic behind the version assignment in case of Parts, whose Study Version attribute uses this versioning scheme by default, is described below:

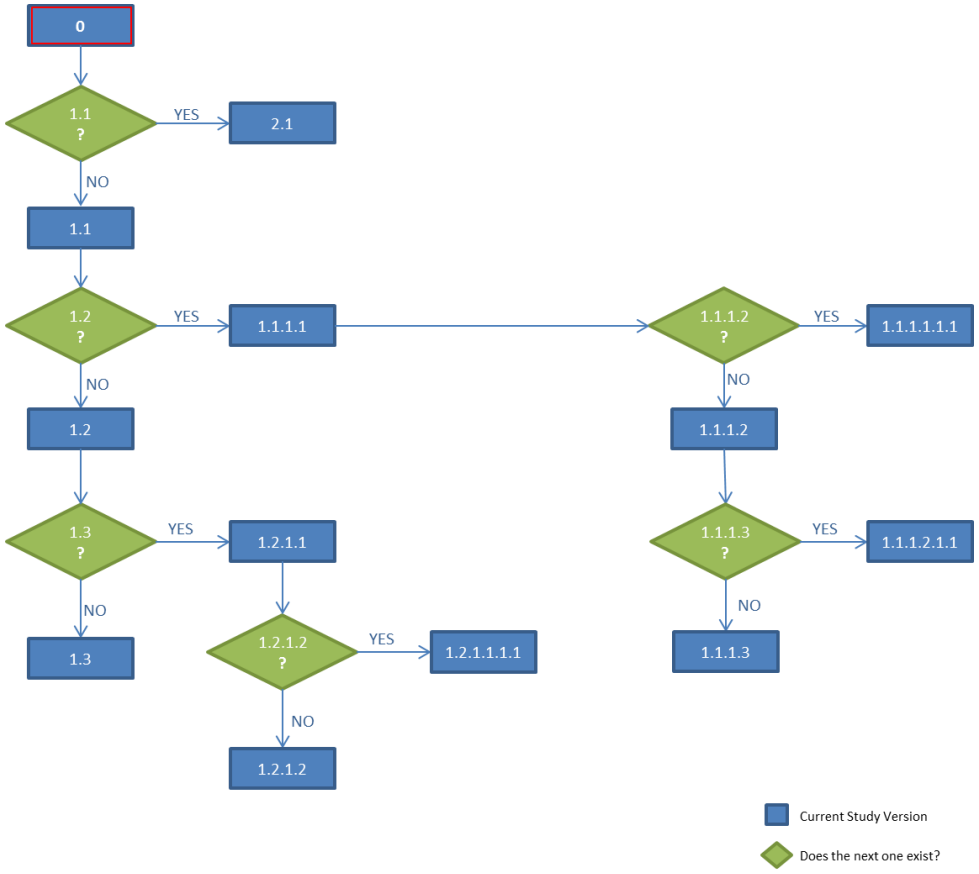
Every Study Version has a unique number. The first file that is saved in the repository gets the zero (0) Study Version. From that point, any future Study Version cannot get a value with odd number of digits. So, the next Study Versions will be *1.1, 1.2, 1.3, 1.1.1.1, 1.2.1.1, 1.2.1.2, 1.3.1.2.1.1* etc. By default, the first value after *0* is the *1.1*. Each new Study Version is given a new value by increasing the rightmost digit by one. Thus, the *1.2* comes after the *1.1* and the *1.3* comes after the *1.2*.

In case the intended Study Version already exists, a new branch is initiated by appending to the current study version the digits *1.1*. For example, if Study Versions *1.2* and *1.3* exist already and the user attempts to create a new one based on the *1.2*, then the new Study Version that will be created will be the *1.2.1.1*.

All main branches, e.g., *1.1, 2.1, 3.1*, etc. are created exclusively from Study Version *0*. For example, if the user would like to create the new main branch *2.1*, this must be created from the Study Version *0*, provided that branch *1.1* have been already saved in DM.

This method makes sure that the history of a file will always be traceable.

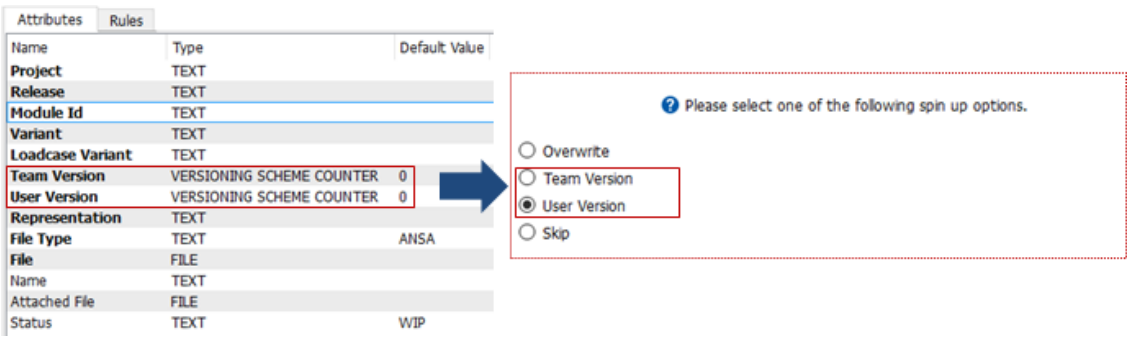
The flowchart below, demonstrates how the new Study Versions are created.



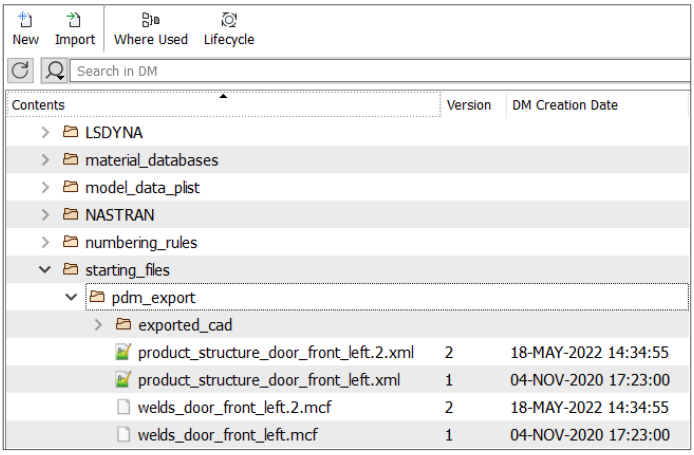
2.2.2. Multiple level Versioning

BETA's SDM solutions support the use of multiple versioning attributes on a DM Object, to accommodate versioning on multiple levels. This can be achieved with proper customization of the data model. In such case, the data manager assumes a default *nesting* between the different versions defined, based on the order they appear in the DM Schema. When a spin-up of a high level version attribute is requested, subsequent version attributes are initialized. In such cases, in the Conflicts Resolution page of the data import wizard, all different version attributes are shown, in the order defined.

In the example below, two different versions are defined for the Subsystem: "Team Version" and "User Version".



2.2.3. Versioning of files



Plain files versioning is possible when working with SPDRM back-end. In this case, the conflict is detected by the system based on the unique path and name of the file. To resolve the conflict, a spin-up of the file version is performed by incrementing the value by one. Additionally, the version value is appended to the file name:

```
<file_name>.<version>.<extension>
```

2.3. Lifecycle Management

The Lifecycle Management of CAE data is supported for both SPDRM and file-based DMs. The Lifecycle Business Rules, which are a set of rules that govern the evolution a DM object, can be specified for all the available data types. These rules are enforced while using a DM repository which has Lifecycle Management activated.

Among others, through the Lifecycle Business Rules it is possible to control:

- Accepted transitions of property/attribute values
- Conflict resolution policy and versioning scheme
- Maturity of DM object based on the maturity of its references and contents

Apart from defining a system of constraints that govern the evolution of a DM object during its lifecycle, a key feature of the lifecycle management mechanism is the tracking of the complete evolution of an object, from the moment it is first created until it is eventually consumed by higher level entities, in order to offer a solid understanding of the dependencies between different DM Objects.

The Lifecycle Management mechanism is Status based. The Status field of a DM Object reflects the item's maturity and it is the fundamental characteristic for the evaluation of the majority of lifecycle imposed constraints.

2.3.1. Activation of Lifecycle Management

Lifecycle Management functionality is enabled with the appropriate configuration of the SDM system:

SPDRM

The Lifecycle Management is activated by adding the following entry in `taxis.conf`:

```
<entry key="LBREnabled">true</entry>
```

Apart from this, the JSON file named `lbr_rules.json`, that contains the lifecycle business rules, should be present in the SPDRM server configuration folder.

File based DM

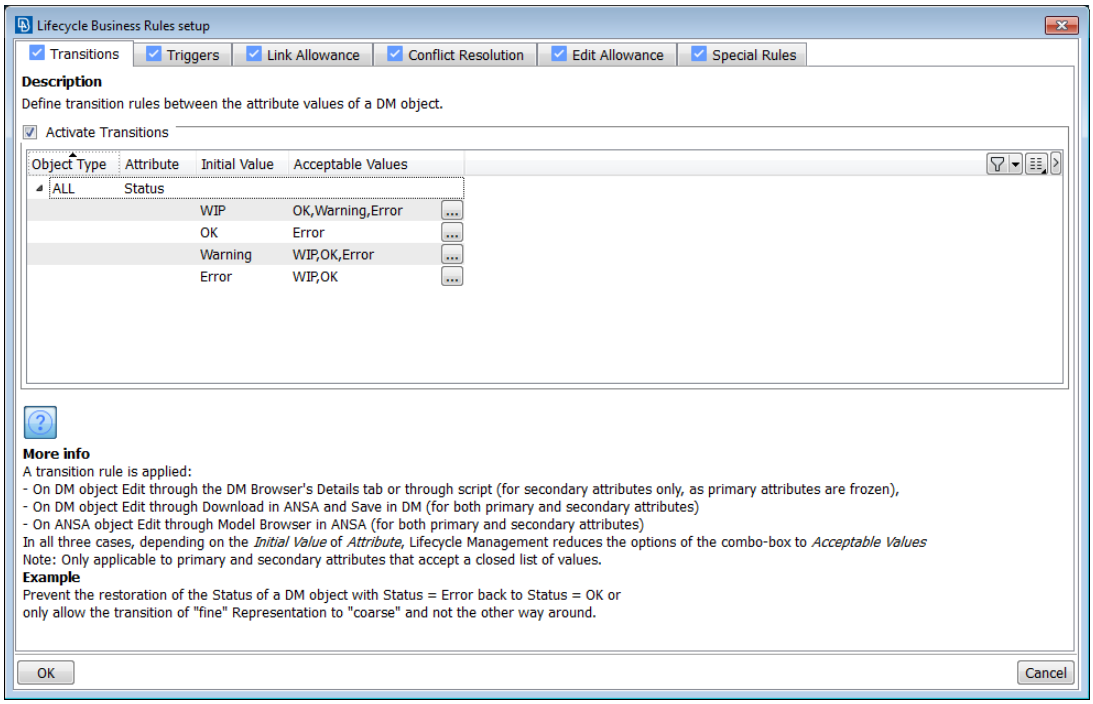
The Lifecycle Management is activated by adding the `lbr_rules.json` file directly in the DM root folder.

2.3.2. Configuration of Lifecyle Business Rules

The Lifecycle Business Rules configuration file `lbr_rules.json` is created through a Python script (`lbr_wizard.pyb`) that is available in the standard ANSA scripts library, in the installation directory. The script provides a dialog for the user to define the different types of Lifecycle Business Rules. Upon confirmation, the `lbr_rules.json` is created. In case of file-based DM the file is saved automatically in the DM folder. In case of SPDRM back-end, a file manager pops-up for the user to select the destination folder, so that the SPDRM administrator moves it in the SPDRM configuration folder at a second step.

The types of supported Lifecycle Business Rules are the following:

- Transitions
- Triggers
- Link Allowance
- Conflict Resolution Rules
- Edit Allowance
- Special Rules



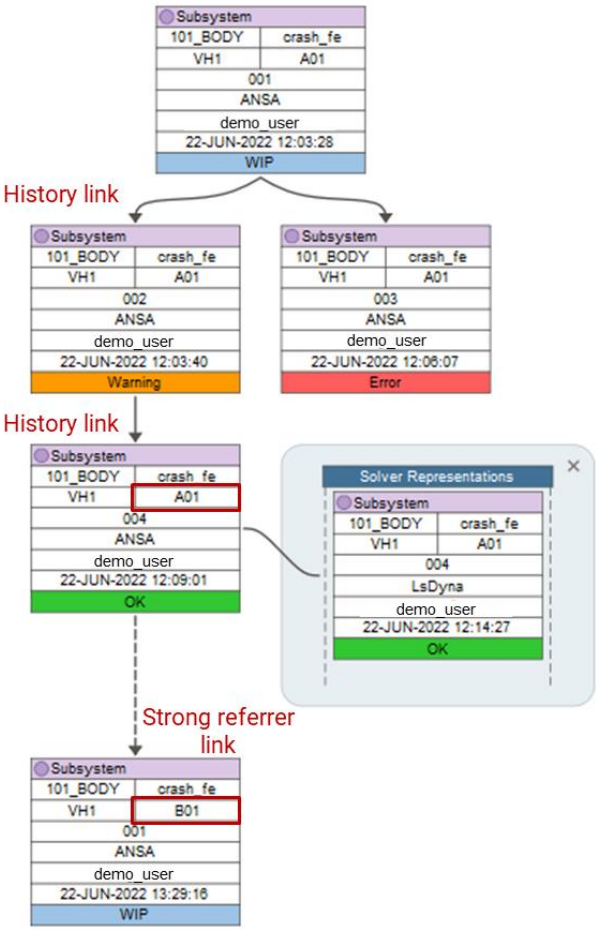
Each rule type can be activated by the respective check box in each tab. If a type is not activated, the Lifecycle Management mechanism will not impose any constraints of the specific type.

2.3.3. Tracking the evolution of a DM object

The Lifecycle Management mechanism can track the complete evolution of a DM Object. Tracking the different versions of a DM object is possible, even without Lifecycle Management, through the history links that are automatically created during conflict resolution in case of spin-up.

The Lifecycle Management mechanism offers an extra capability since it creates a link (strong referrer/strong referee) when an existing DM object is used as the starting point to create a new DM object with different properties. Such links can be displayed in the Lifecycle Graph similar to the history links.

In the lifecycle graph displayed, a Subsystem with release A01 has several iterations (001-004) and a solver representation (LsDyna). The iteration 004 of release A01 was used as the basis for the generation of release B01. To track this relationship in the evolution of the Subsystem, a strong referrer/referee link between the two items is automatically created upon saving release B01.



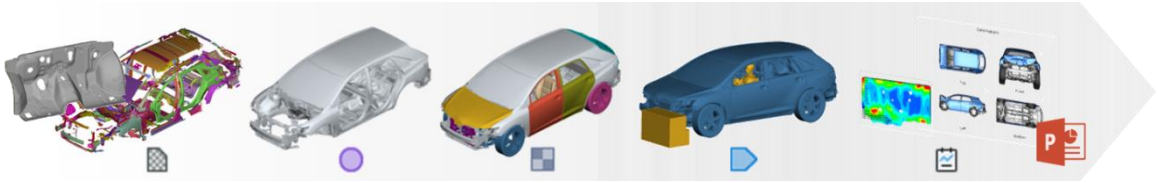
2.4. Traceability

BETA's SDM solutions come with functionality that offers a gateway to traceability, facilitating the efficient preparation and management of simulation data. As the model evolves and its various components are created and stored as data objects, the SDM system captures their relations using links (i.e. references between data objects).

There are three main categories of reference links traced by BETA's SDM solutions:

- **Where Used**: these hold the relations between data objects and their contents and are created automatically by the system. For example, selecting a Subsystem, one can view all its direct "where used" relations, e.g. the Simulation Models that contain it, or even its inferred relations, e.g. the related Simulation Runs.
- **Lifecycle**: these hold relations between data objects during the course of their lifecycle and are created automatically by the system. For example, the user could view all the different versions of a Subsystem or the solver representations that have been created based on an ANSA Subsystem.
- **Other Links**: these hold various types of relations that are created automatically by the system and do not fall in any of the above link types. In addition, links created by the user using BETA's scripting API are also categorized as *Other Links*. For example, one could view here the relation between a DM Object and the Modular Environment Profile used when it was saved, the relation between an Optimization Study with one or more Predictors that were used to create it, the compatibility/incompatibility relation between a solver File Type Connecting Subsystem with one or more Regular Subsystems, etc.

The links created between data objects enable users to track the evolution of the model and associated data during model build. Quite commonly, users in the CAE world need to answer questions such as "*Which was the CAD version of a Part used in my simulation?*". With BETA's SDM solutions, traceability of CAE data is maximized. Through the *Where Used* relationships for example, it is possible to start from a specific Part and identify the Subsystem that uses it, the Simulation Model that contains the Subsystem and eventually the Simulation Runs that were created using the Simulation Model.



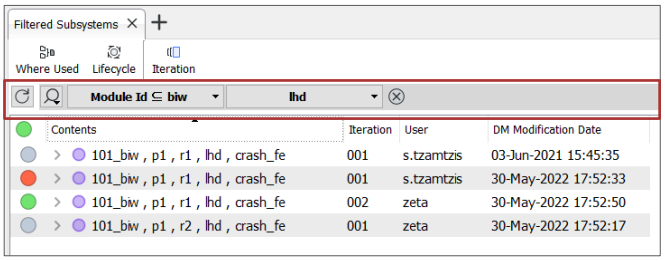
It is also possible to traverse the relationships in the reverse direction, starting from the results of a simulation, it is possible to identify the Simulation Model that was used, the respective Subsystems and eventually the exact version of a specific Part used in one of the Subsystems.

2.5. Search

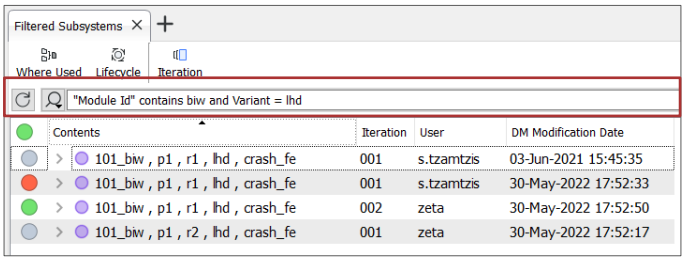
2.5.1. Query language-based searches

BETA's SDM solutions offer a powerful mechanism for searching and identifying data. In the heart of this search mechanism lies a powerful query language, the BETA QL, which is common for all the products of the BETA Suite. In the data management context, searches are primarily performed based on the metadata of the DM Objects. However, BETA QL also supports the creation of relationship-based queries, which enable the search of DM Objects based on the metadata of other DM Objects that use them. For example, the Parts contained in a Subsystem, can be identified based on the *Name* of this Subsystem, or even going one level higher, based on the *Discipline* attribute of the Simulation Model that uses this Subsystem.

BETA QL supports two modes for the definition of queries: The *Basic* mode, which is an assisted mode to define simple queries and the *Advanced* mode, which enables the writing of more advanced expressions in the query field that supports suggestions and auto-complete.



Basic mode



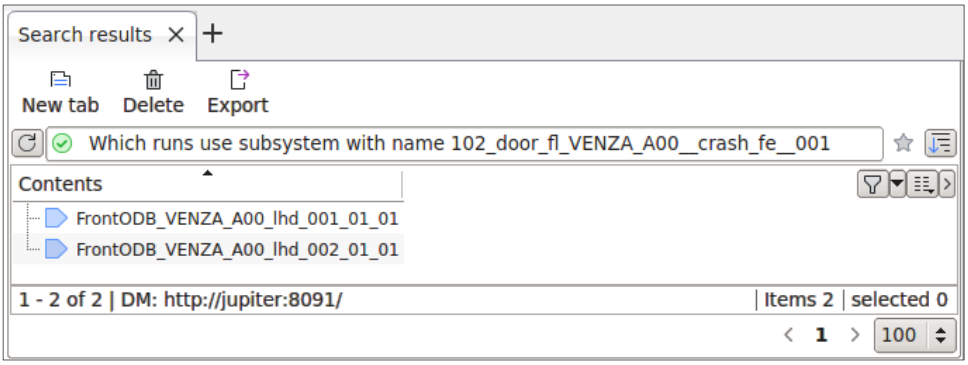
Advanced mode

BETA QL searches can be saved for quick access.

For more information on search possibilities, please refer to paragraph 3.5.

2.5.2. Free text searches

The SPDRM-based solution offers a free text search, complementary to the BETA QL-based searches. This search mode enables the description of queries in natural language.



For more information on search possibilities, please refer to paragraph 3.5.

2.6. Security and access control

As described in the specifications of SDM systems in Chapter 1 of this guide, data security and user access control are only possible on server-based systems[1]. In SPDRM, BETA's server-based SDM solution, these are achieved through role-based user management and the use of Access Control Lists (ACLs).

SPDRM supports the definition and management of different Roles (Groups), each of which contains different users with explicit privileges regarding the system resources (data and tools). Some key benefits of SPDRM's role-based user management are:

- Roles support a parent-child relationship, which can be used for the propagation of privileges.
- A user can be assigned more than one role. This makes it possible for the same user to have different access rights when he/she logs into the system with a different role.

Privileges on data are set with the aid of Access Control Lists (ACLs), which are lists of permissions associated with the various data objects. ACLs essentially control the levels of privilege each user will have, depending on their role. For example, if a user's role doesn't have "View" privileges on a Subsystem, it won't be possible to search and find it through the SDM client. Similarly, without "Modify" privileges, it won't be possible to edit its file or modify its attributes and without "Delete" privileges it won't be possible to delete it or overwrite it.

An additional solution for data security is the use of data encryption. In general, BETA's SDM solutions do not offer any data encryption capabilities. An exception to this is the capability to encrypt script files managed by SPDRM, BETA's server-based SDM solution. This applies both to scripts stored in the library of SPDRM, which can be encrypted during export, as well as scripts used by SPDRM process definitions. Nevertheless, already encrypted data files can be managed through BETA's SDM clients in a similar fashion to their management in the file system. Therefore, encrypted data files can be imported from the SDM client, stored in the database and exported as needed.

2.7. File access through the file system

Direct access, without an SDM client, to files associated with DM objects is possible both in case of SPDRM and file-based DMs. In case of file-based DMs, the data repository is a file structure directly accessible to any user that has appropriate permissions in the file system. In case of SPDRM, depending on the configuration of the environment, a file structure can be generated and maintained real-time by the SPDRM server. In both cases, the file structure provides direct access to the files of the SDM repository to external users and applications.

For example, during submission of a Run to the solver, the HPC needs access to various data stored in the SDM system. Instead of exporting the data from the SDM system, it is possible to use directly the files in the file structure. Additionally, when SPDRM is used as SDM system, the direct access to the exported file structure is important in case of system maintenance and downtime.

The configuration of the file structure (i.e. format and structure levels), in case of file based DM, is done through the dm_structure.xml as described in paragraph 6.2. In case of SPDRM, the activation and the configuration of the file structure is done through the *Data > DM structure export setup* option as described in SPDRM Administrator's guide paragraph 4.3.

The exported file structure in SPDRM is read-only, so it is not possible for the end-user to make modifications to any file. Contrary, in the case of file-based DMs, the end-user typically has full permissions to the file structure, which is the actual data repository. It is therefore crucial not to make modifications to the contents of a file-based DM directly

[1] *Using BETA's file-based SDM solution, it's possible to have a minimum level of data security that offers protection from accidental data deletion. Refer to paragraph 6.7 of this guide for a more detailed description.*

through the file system, even if this is possible. Extreme caution is required when such operations are made, since there is no data restoration possibility offered by file-based DM.

When SPDRM is used for Modular Run Management through ANSA, whether an exported file structure is available or not, affects the way include references are written within the solver files of compound containers (i.e Simulation Model, Loadcase, Run). By default, the exported file structure is not active and such references are relative paths (i.e. just the file name is written, assuming that all referenced files reside in the same directory with the main file). If the exported file structure is activated these references are absolute, to enable the direct utilization of the solver file from its exported location.

Exported file structure	References in compound container's solver file
ON	INCLUDE 'D:/spdrm_vaults/vault1/DM/Entities/Subsystems/101_BODY/ALL/VH1/A01/001/nvh_fe/-/Nastran/repr/101_BODY_VH1_A01_ALL_nvh_fe__001.nas' INCLUDE 'D:/spdrm_vaults/vault1/DM/Entities/Subsystems/102_DOOR_FL/ALL/VH1/A01/001/nvh_fe/-/Nastran/repr/102_DOOR_FL_VH1_A01_ALL_nvh_fe__001.nas' INCLUDE 'D:/spdrm_vaults/vault1/DM/Entities/Subsystems/900_CONNECTIONS/ALL/VH1/A01/001/nvh_fe/-/Nastran/repr/900_CONNECTIONS_VH1_A01_ALL_nvh_fe__001.nas'
OFF	INCLUDE '101_BODY_VH1_A01_ALL_nvh_fe__001.nas' INCLUDE '102_DOOR_FL_VH1_A01_ALL_nvh_fe__001.nas' INCLUDE '900_CONNECTIONS_VH1_A01_ALL_nvh_fe__001.nas'

In case of file-based DM, these references are by default absolute. However, it is possible to enforce the use of relative paths also in case of file-based DM, through the '*support_solver_relative_paths*' option in the DM settings section of `dm_structure.xml` (see paragraph 5.1.4).

2.8. Data deletion and purging

Data deletion and purging is an integral feature of any effective SDM system. SPDRM, BETA's server-based SDM solution, employs a deletion mechanism that facilitates the deletion of unwanted data and the cleanup of the vaults from old and/or temporary data.

The deletion of any data object through the SDM client is governed by a set of rules based on its existing dependencies, ensuring that data needed for the definition of other data objects are protected from deletion. Although any user with the respective privileges can potentially delete a data object through the SDM client, provided of course that the deletion rules are satisfied, the management of deleted data is handled only by the system administrator. SPDRM comes with data management features that enable system administrators to:

- Define a data retention policy, in order to prevent accidental data loss and facilitate data restoration
- Purge deleted data, in order to clean up the data vault, while simultaneously maintain the metadata

For more information on BETA's data deletion and purging mechanism, please refer to the SPDRM Administrators Guide.

2.9. Data migration

Data migration is the process of transferring data from one SDM system to another. For example, transferring data from a production to a development environment is a common data migration process.

In case of SPDRM, the data migration is a two-step process that can be executed by the system administrator, through the SPDRM administrator console. Initially the **DM Export** function is executed on the source environment, and then the exported package is imported to the target environment through the **DM Import** function. For more information on **DM Export** and **DM Import** functions please refer to paragraph 4.2.1 of SPDRM Administrator's Guide.

Data migration to a different repository is also possible in case of file-based DM through the **Send to other DM** function in ANSA DM Browser. This function allows to save a selection of DM objects that exist in one repository to another. For more information on **Send to other DM** function please refer to paragraph 6.4.

Note that both in case of SPDRM and file-based DM, a common data model between the source and target repositories is a prerequisite for data migration.

2.10. Client-side file caching

The SDM client applications need to ensure that DM operations are executed with maximum performance. To facilitate this requirement, a file caching service is employed within KOMVOS, ANSA and META.

Files downloaded from DMs are cached in a managed, persistent store. This way, subsequent usages are very efficient in terms of performance, as the files are recalled from the cache store and are not downloaded again. The file caching service is enabled by default.

The footprint of the cache store is supervised and automatically controlled according to the values set in the client settings, under the settings group "Cache Management". The available settings are:

cache.service_enabled
This setting controls whether the File Cacher service is enabled or not. Default: true
cache.store_size_limit
The maximum cache store footprint that can be sustainably supported. Default: 2.00 GBytes
cache.overcommitment_allowed
The fraction by which the Store Size Limit may be transiently surpassed. For example, if set to 0.5 (default), the File Cacher mechanism will allow data to be stored until 150% of the store size limit is reached. At the same time, it will take actions in order to get the cache store footprint to what has been configured as Store Size Limit
cache.reserved_disk_space
Do not add data in the cache store if the free space in the disk where the cache store resides would fall below this threshold. Default: 200.00 MBytes

The File Caching functionality is also available through the Python scripting language, through the classes `utils.FileCacher` and `utils.FileCacherWatchdog`.

2.11. Interfaces

BETA's SDM solutions provide Application Programming Interfaces (APIs) that enable their connection with other systems and the exchange of information with external applications.

2.11.1. SPDRM's Interfaces

2.11.1.1. REST API

SPDRM offers a rich library of REST endpoints that allow a user to interact with the SDM back-end. The library includes endpoints that provide access to the following features:

- Authentication
- Server configuration
- Data Management
- Process Management

- BETA Apps Launcher (BAL) Management

The documentation of the REST API can be found in the SPDRM installation under:

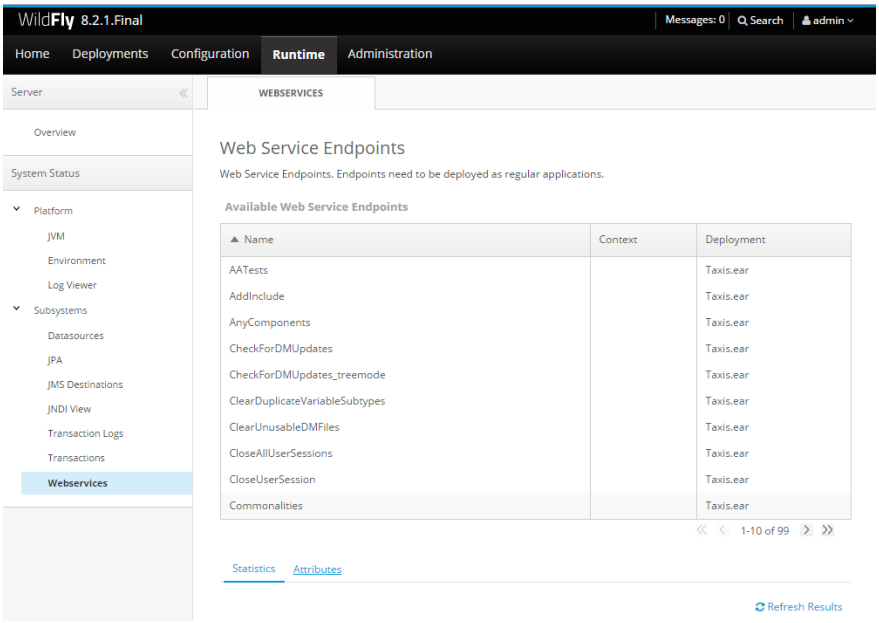
```
docs/assets/users_guides/spdrm_rest_api/rest-api-documentation.html
```

Note! The use of the REST API requires the special feature `SPDRM_EXTERNAL_CONNECTIONS` in the license file

2.11.1.2. SOAP API

SPDRM also offers a SOAP API that enables external applications to primarily access its Authentication and Data Management functionality. Data management through ANSA, META and KOMVOS with SPDRM back-end is based on the SOAP API.

The list of all available Web Services is available in SPDRM Server's web-page. However, this API has no documentation and is practically not usable from 3rd party applications developers.



More information can be found in the SPDRM Administrator's Guide.

2.11.1.3. Python API

SPDRM offers a complete Python API that is accessible through KOMVOS, ANSA and META and enables the interaction of the user with the SDM back-end directly through the client application. Through this API, a user can manage authentication topics and access data, process and issue management functionality.

The same API is available through all client applications, making it possible to use a function created for KOMVOS within ANSA or META and vice versa.

2.11.2. File-based DM's Interfaces

2.11.2.1. REST API

The **DM Microservice** is the REST API offered for a file-based DM. This is essentially a protocol translator: it connects to a file-based DM backend using the appropriate protocol (i.e., SQLite file access) and exports to the user a REST protocol. A single instance of the DM Microservice handles a single DM backend. However, it can have multiple,

independent users performing operations towards this DM. Authorization/authentication is exposed over the REST API, but it is delegated to the file-based DM backend.

The DM Microservice is not included in the BETA Suite's package and can be provided upon request.

2.11.2.2. Python API

File-based DM offers a complete Python API that is accessible through KOMVOS, ANSA and META and enables the interaction of the user with the SDM back-end directly through the client application. Through this API, a user can connect to different DMs and access all data management functionality.

The same API is available through all client applications, making it possible to use a function created for KOMVOS within ANSA or META and vice versa. Furthermore, the same API is used for SPDRM back-end too.

3. Core functionality

Data navigation

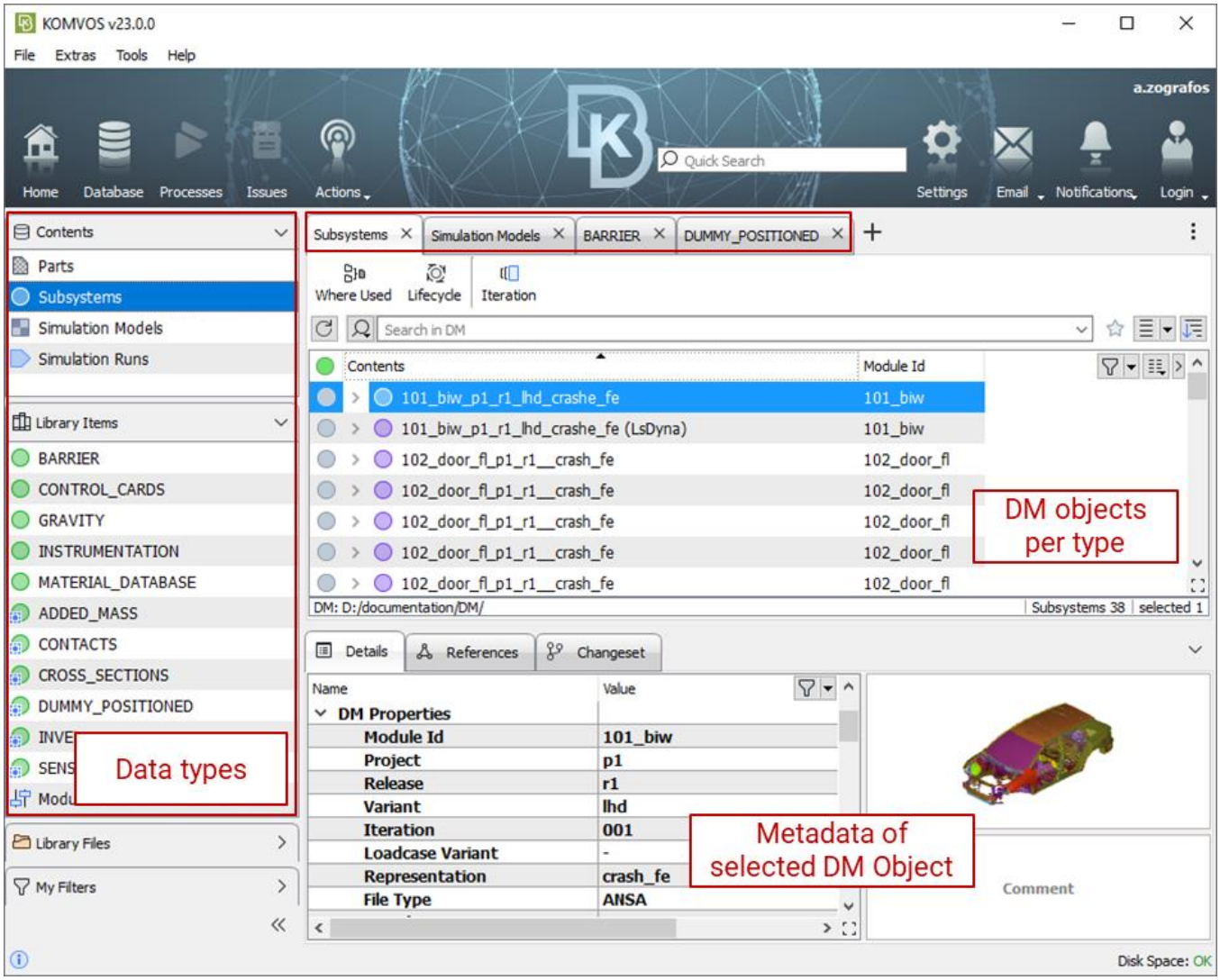
The contents of the SDM repository can be browsed through the DM Browser in ANSA/META and through the Database workspace in KOMVOS. These tools provide several functionalities such as search, edit, export, explore relationships between DM Objects, compare different versions/variants, review simulation run results, etc.

The window is split vertically in two main panels:

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

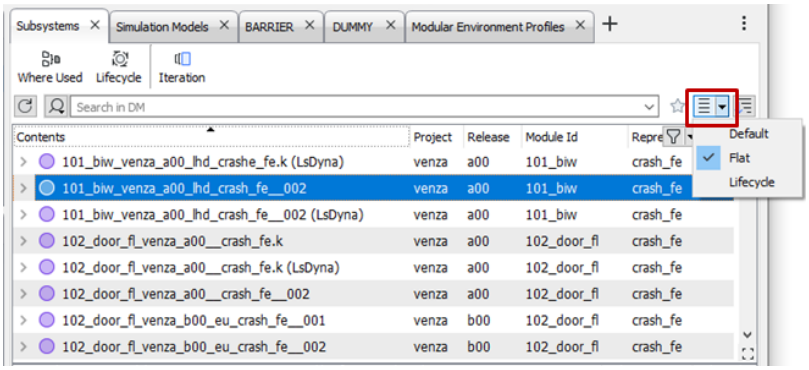
- **Contents**: Main model data types (e.g. Parts, Subsystems, Simulation Models, Simulation Runs)
- **Library items**: Auxiliary data types (e.g. Materials, Dummies)
- **Library files**: Single file library items (e.g. scripts, batch mesh sessions)

The right panel displays in different tabs the actual DM Objects per type. Double-click on a data type on the left opens a new tab in the right panel. Each tab displays the main list of DM Objects.

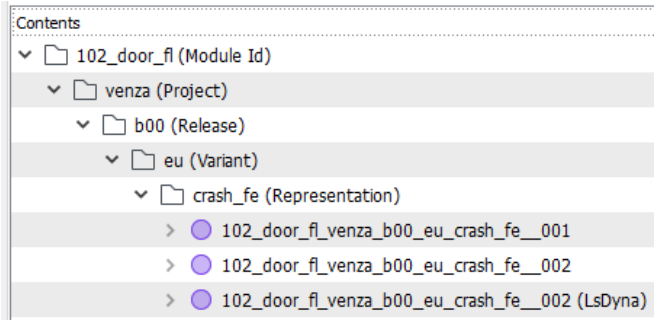


3.1.1. Data Views

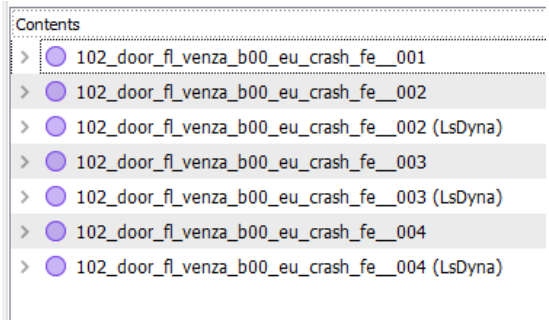
The structure and layout of the information displayed in the browser is organized in different “views”. Each data type (e.g. Subsystems, Simulation Models, Simulation Runs, etc.) has its own views that are accessible through the **View Modes** button.



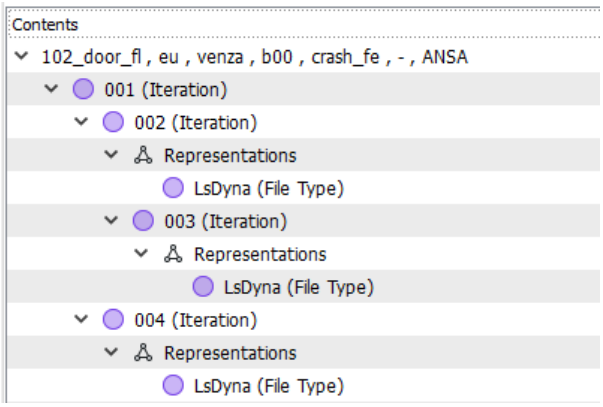
Depending on the data type, several alternative views are available:



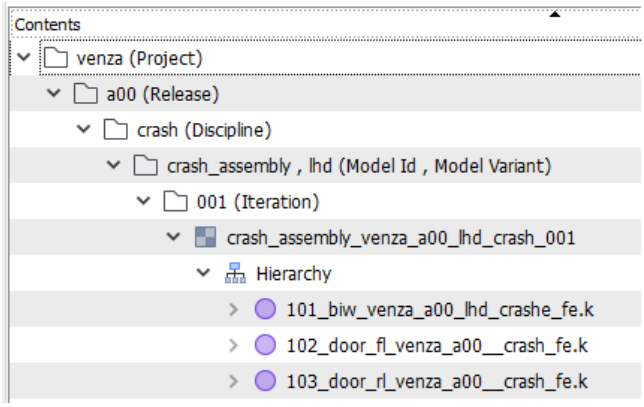
Default: DM Objects are presented in a tree view, where each grouping level corresponds to one or more attributes of the DM Object contained.
This view is available for all data types.



Flat: DM Objects are presented in a flat list.
This view is available for all data types.



Lifecycle: DM Objects are grouped based on history links. DM Objects that are created as iterations or solver file representations are grouped together.

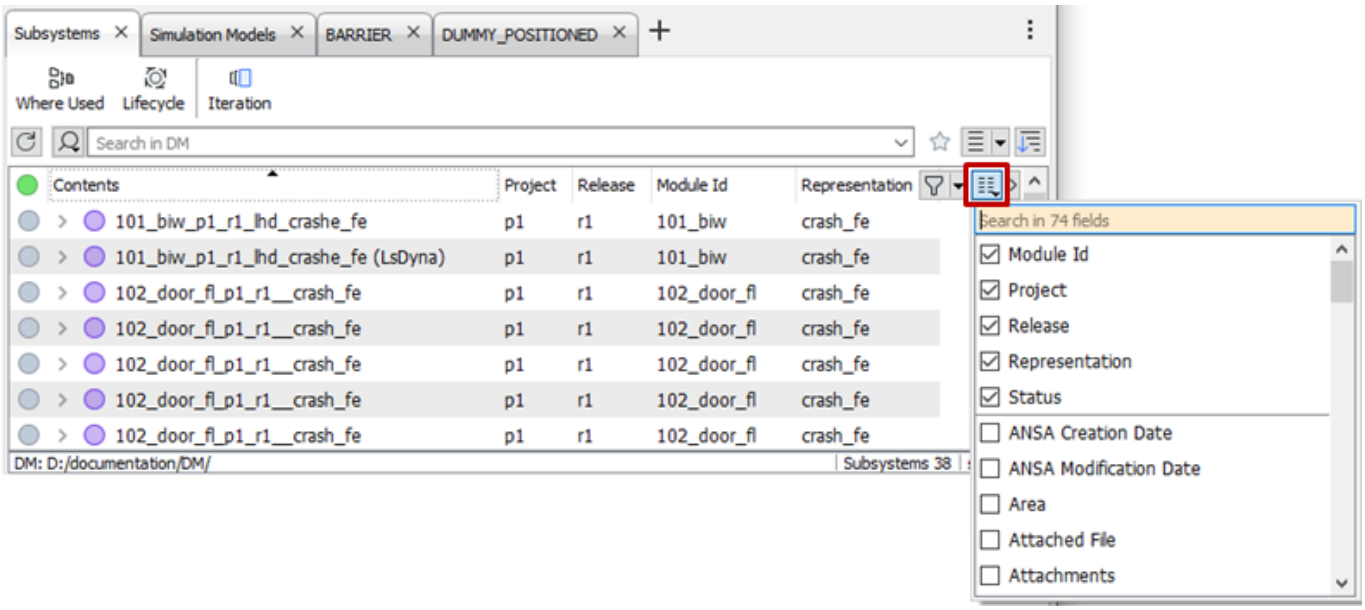


Hierarchy: DM Objects are presented in a tree view similar to that of the **Default view**. However, the DM object itself can be further expanded to reveal its contents. This view is only available for Simulation Models and Simulation Runs. In case of Subsystems, the part contents appear in the **Default** and **Flat** view modes.

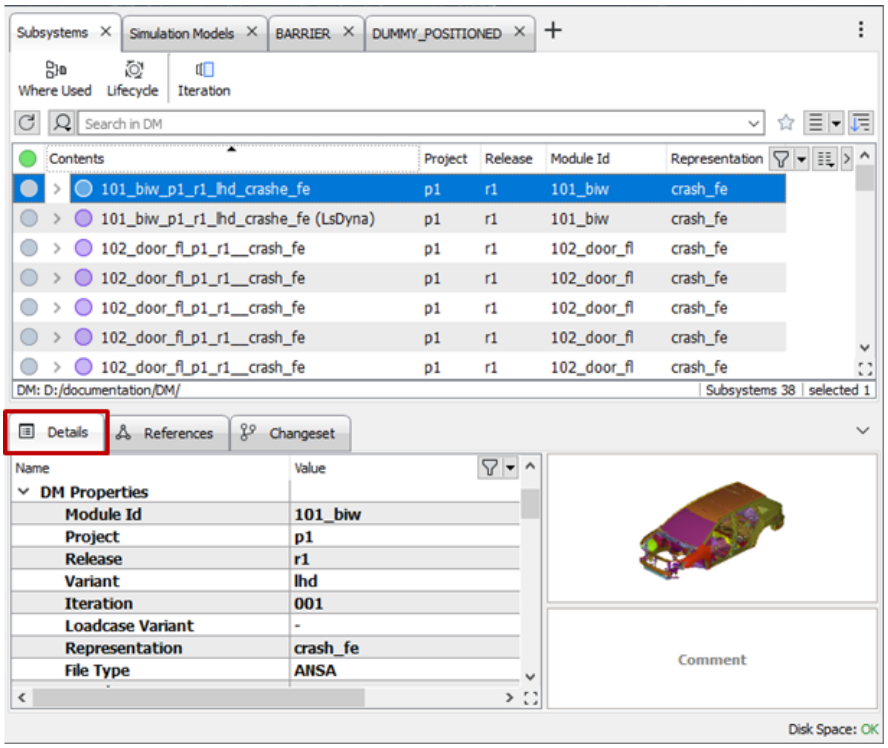
3.1.2. DM Object information

Within the main list, the metadata of the listed items can be seen either in columns or in the Details tab at the bottom.

Usually, columns are added for those attributes that need to be visible at all times in order for the user to make quick comparisons of their values for different items. The default visible columns can be controlled centrally through the views configuration, as described in paragraph 5.2.1. The user can add or remove columns through the Add/Remove Columns button, as shown below:

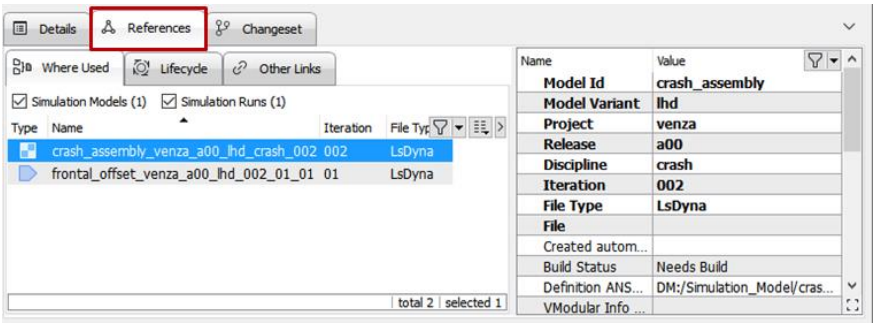


Note that the visible columns info is stored in the GUI configuration file (KOMVOS.xml, ANSA.xml or META.xml in the user's settings folder)

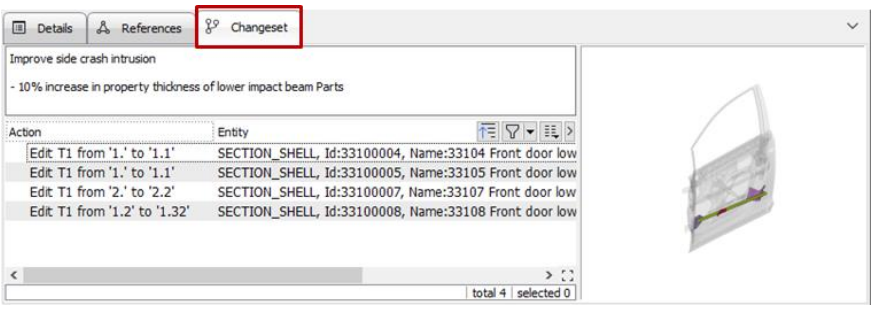


Under the main list there is the info tabs area which is used to provide more information on the selected item. In the **Details** tab, the primary, secondary and additional attributes of the selected item are listed.

The visible attributes and their structure are configurable centrally, as described in paragraph 5.2.1.



The **References** tab lists the DM objects related to the selected items. The types of relationships are grouped in three categories (i.e. *Where Used, Lifecycle, Other Links*) which are displayed in separate sub-tabs. Refer to paragraph 3.8 for more details.



Finally the **Changeset** tab lists all the recorded modifications of the selected Part/Subsystem that represent the difference from its predecessor. For more details, refer to paragraph 3.9.

3.2. Import data

Data can be added to the SDM system in two ways:

- By importing existing files
- By saving new DM Objects from ANSA/META

In both cases, the procedure is similar. The main difference lies in the fact that during Import, the file pre-exists and it's the user's responsibility to describe its identity to the system, whereas in the case of save, the file is created at the time by ANSA or META, which is also responsible to describe its identity to the system.

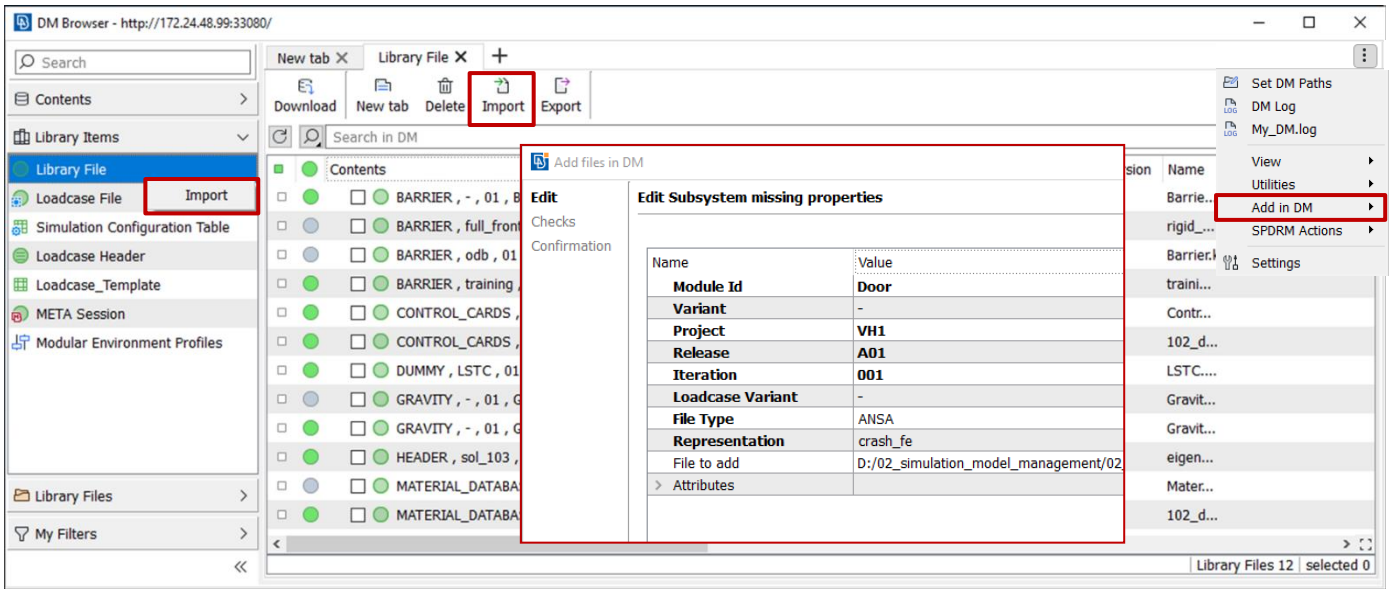
Once the file, the identity and any other metadata are provided, the proceedure that is followed is common and is outlined in the steps below:

1. Conflict detection and resolution (paragraph 3.2.1)
2. Indexing (paragraph 3.2.2)
3. Addition of the DM Header - where applicable (paragraph 3.2.3)
4. Upload to the SDM system

The following is a brief description of the Import and Save procedures. More information and step-by-step guidance can be found in the ANSA and KOMVOS User's Guides, in Chapters 30 and 4 respectively.

Import existing files

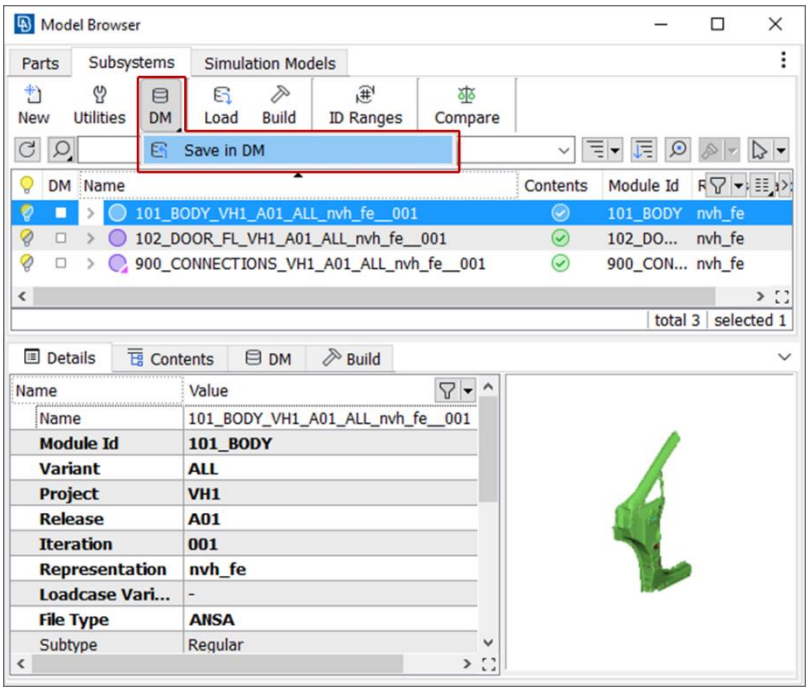
Any file on the file system can be used as a representation file of a new DM object. This functionality is generally referred to as **Import** or **New** in KOMVOS and **Add in DM** or **Import** in ANSA/META.



Existing files can be also imported with script, through dedicated functions in the `dm` module of the Python API.

Save new DM Objects from ANSA

Through the ANSA user interface, model and loadcase data can be pushed to the SDM back-end through the Model Browser and more specifically, with the **Save in DM** function.



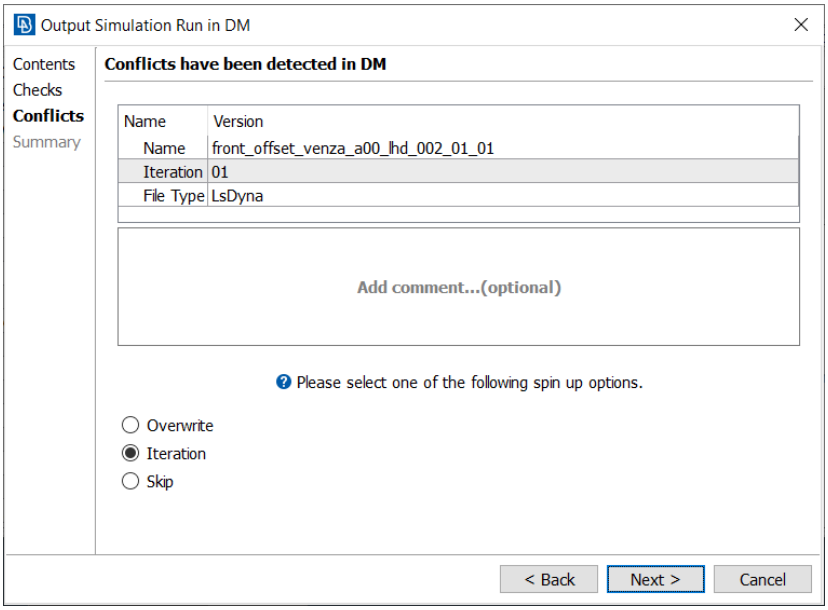
Data can be also saved to the SDM back-end with script, through dedicated functions in the `dm` module of the Python API.

Save new DM Objects from META

Through META, it is possible to save reports in the SDM back-end using session commands. Apart from this, existing reports and session files can be added in DM, through the respective context menu option of Simulation Model and Runs in the data manager. More information on these options can be found in paragraphs 4.1 and 4.5.

3.2.1. Conflict detection and resolution

During the import process, a check for potential conflicts within the SDM system is performed. Essentially, a conflict is detected if there is a DM Object with the same primary attributes already in SDM system. In case a conflict is identified, it is reported in the *Conflicts* page of the Save/Add in DM wizard and the user is prompted to take action.



To resolve the conflict the user can:

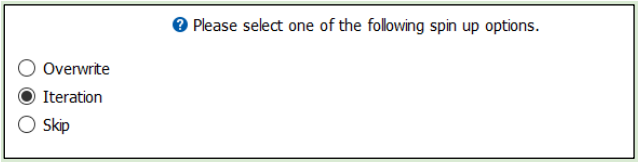
- Save the DM Object as a new version, i.e. spin-up the version
- Overwrite the existing DM Object
- Halt the process

More information on each option is given in the paragraphs below.

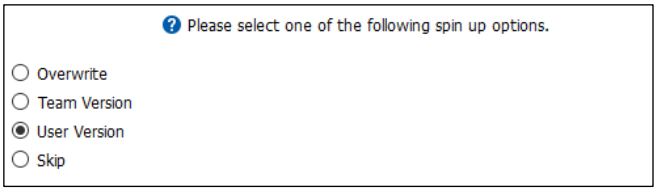
3.2.1.1. Saving a new version

Version spin-up is given as an option when the DM Object type concerned has some primary attribute of versioning type defined in the DM Schema (data model). In the default DM Schema, there's a single version attribute for each DM Object type named *Iteration*. Therefore, that's the only spin-up option given in the wizard.

However, as described in paragraph 2.2.2, with customization of the DM Schema it is possible to define more than one versioning attributes for a DM Object type. In this case, the Save/Add in DM wizard will be adapted automatically, showing all different version attributes in the order defined.

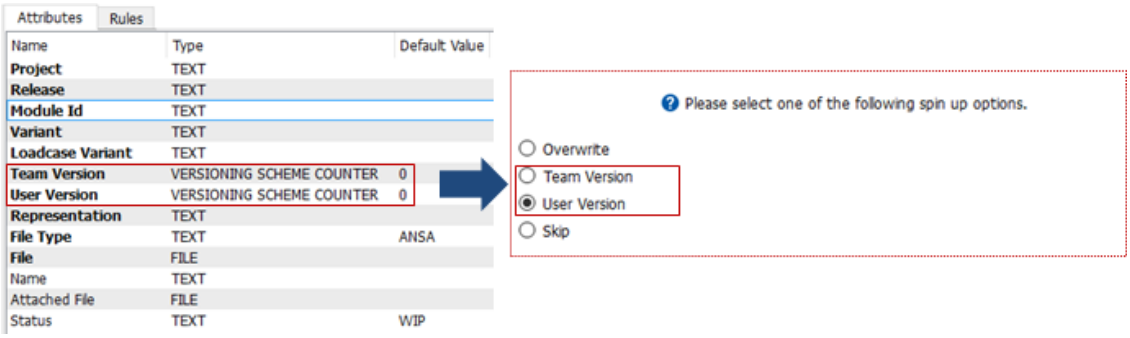


Conflict resolution with one version attribute

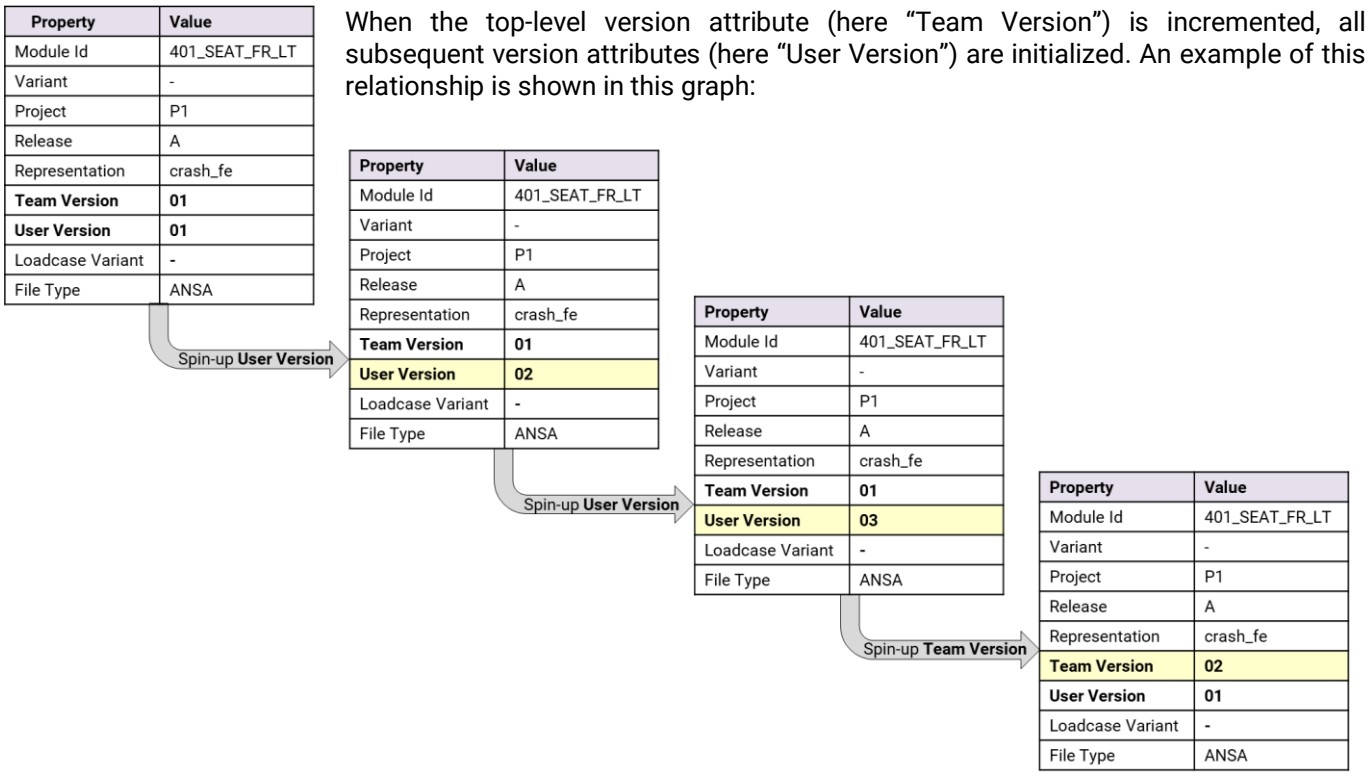


Conflict resolution with two version attributes

In the example below, two different versions are defined for the Subsystem: "Team Version" and "User Version".



When the top-level version attribute (here “Team Version”) is incremented, all subsequent version attributes (here “User Version”) are initialized. An example of this relationship is shown in this graph:



3.2.2. Indexing

When a user adds some data in the SDM system, either with **Save in DM**, where the file is generated on the spot by ANSA, or with **Import** or **Add in DM**, where an existing file is copied to the SDM back-end as a Subsystem or Library Item, there’s a background process which scans the file in order to extract useful information on its contents and add it as metadata to the generated DM object. This operation takes always place irrespective of the SDM back-end (file-based or SPDRM).

When the import operation takes place through ANSA, the background process runs in the same ANSA. However, when the import operation is done in KOMVOS, the background process runs in a dedicated ANSA worker that will be launched automatically if ANSA is found in the installation or specified in Settings > BETA Apps.

The indexing metadata are stored as Additional Attributes of the DM object and can provide valuable information about each DM object directly, eliminating the need for loading the representation file in ANSA.

There are two categories of indexing metadata, as shown in the tables below.

Category 1: Definition Attributes			
Attribute Name	**Description**	**Save in DM**	**Add in DM**
Nodes / Elements / Sets / Properties / Materials / Define / Function / Rest Min / Max Id	The id ranges of the entities of the file, categorized in the 8 types that can be used for id offsetting in LS-DYNA	✓	✓
ID Ranges Excluded Entities	The ids of nodes that are excluded from the calculation of the node id range. These are nodes marked by interface points that request particular node id that lies outside the id range of their base module.	✓	✗
Undefined Entities	The name and id of all undefined entities found in the file, except for Parameters, that are stored in a separate attribute	✓	✓
Undefined Parameters	The name of parameters that are used in the file but are not defined in the file	✓	✓
Defined Parameters	The name and value of parameters and parameter expressions that are defined in the file	✓	✓
Unsupported Numbering	A Boolean indication of whether the numbering of entities in the file will lead to automatic renumbering when reading in ANSA	✗	✓
Unsupported Keywords	A Boolean indication of whether the file contains unsupported keywords	✗	✓

Category 2: Interface Attributes			
Attribute Name	**Description**	**Save in DM**	**Add in DM**
Interface Representation > A-Points [1]	This set of attributes describes the interface points to be used for assembly.	✓	✓
Interface Representation > LC-Points[1]	This set of attributes describes the interface points to be used for loadcase setup.	✓	✓
Interface Representation > Interface Sets[1]	The Name, Id and Label of sets marked as Interface Sets	✓	✓
Interface Representation > Properties[1]	The Name and Id of properties marked as Interface Properties	✓	✓
Interface Representation > BC Sets[1] (NASTRAN only)	The type and Id of B.C. SETs	✓	✓
Interface Representation File	An ANSA file containing all interface entities of a Subsystem or Library Item. Interface Representation Files are used during Smart Assembly and Load-case set-up, in order to provide “rich” information about the Interfaces to the Assembly or Load-case set-up methodologies. They contain all types of interface entities plus B.C. SETs information.	✓	✓[2]

[1] Note that the interface representation attributes are zipped in order to reduce the size of database records, i.e. a single additional attribute packs information of 20 interface entities at a time (zipped attributes are named EncodedInterface_0001_0020, etc.)
[2] In case the file is an LS-DYNA keyword file containing unsupported numbering of sets, the creation of the Interface Representation file is skipped.

3.2.3. DM Header

The DM header is a specially formatted block of comment text that is prepended at the top of the Main Representation keyword file of a DM Object. It describes the metadata of the Model Browser Container, mainly Primary attributes but also some Secondary and optionally Additional Attributes. By default, the DM header includes the primary attributes.

Example: DM header of a Subsystem

```
$============================DM INFO BEGIN============================
$ANSA_DM_INFO_PRE_FORMAT;
$
$Entity Type       : Subsystem
$Module Id         : 101_BODY
$Variant           : NSM
$Project           : VH1
$Release           : A01
$Iteration         : 001
$Loadcase Variant  : -
$File Type         : Nastran
$Representation    : nvh_fe
$Name              : 101_BODY_VH1_A01_NSM_001
$User              : username
$Special Type      : Regular
$Solver Version    : MSC Nastran
$END_ANSA_DM_INFO_PRE_FORMAT;
$============================DM INFO END============================
$
```

Through the DM header, the user can understand the identity of an object when text editing its file. Furthermore, the DM Header is used by ANSA when reading a keyword file in order to regenerate the corresponding Model Browser Containers. In case a compound Model Browser Container was saved as a monolithic keyword file by ANSA, several DM headers are written in the file, at the beginning of each block of keywords that corresponds to a Model Browser Container. Therefore, even in this case, it is still possible to regenerate the Model Browser Containers by reading a monolithic solver keyword file.

3.2.4. Pre-import checks and validation

With BETA's SDM solutions it is possible to associate to the main import operation certain pre-import actions that can take care of checks and validations. The table below summarizes the supported pre-import actions:

Action	KOMVOS	ANSA	META
Lifecycle Management	✓	✓	✓
Validation Script Actions (SPDRM only)	✓	-	-
Build Process	-	✓	-
Pre-/post-save Script Functions	-	✓	-
Overriding default "Save" by Script	-	✓	-

3.2.4.1. Lifecycle Management

Through the Lifecycle Management mechanism, a set of rules that govern the evolution a DM Object can be specified for all the available data types. These rules are enforced while using an SDM repository which has Lifecycle Management activated. Within this context, it is possible to conditionally block import, depending on the metadata of the item to be imported and the related DM objects that are available in the repository.

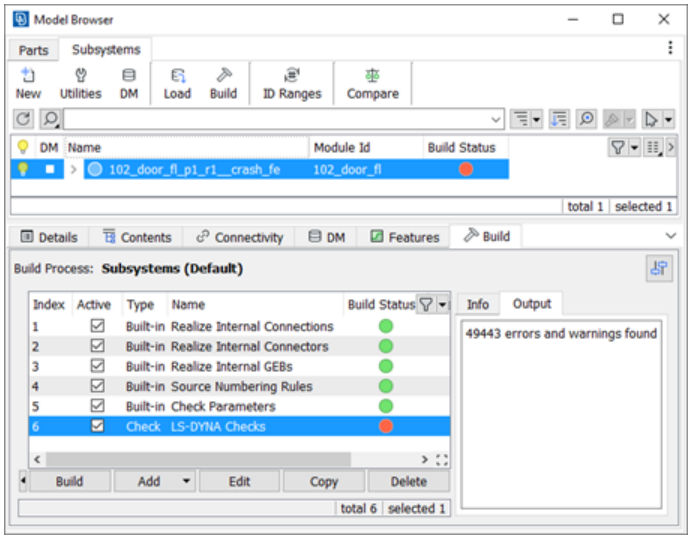
3.2.4.2. Validation Script Actions

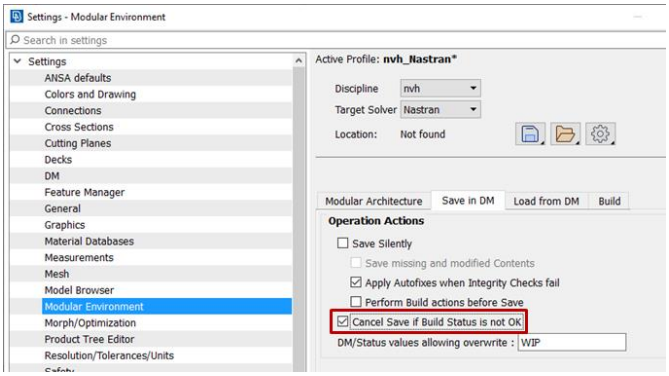
In case that the SDM system is SPDRM, a user defined validation action can be defined per data type. This action controls whether the creation of a new DM object, or the modification of an existing DM object, is allowed based on the metadata and the representation file.

3.2.4.3. Build Process

In the BETA Suite's Modular Run Environment, each data type has a default process that must be followed to verify that an item is ready to be saved in the SDM system. The Build process consists of several built-in individual steps, the Build actions, which can be extended and enhanced with the aid of scripting, by adding one or more custom actions. Build actions differ between the various data types (i.e. there are different actions for Subsystems, Library Items, Simulation Models, etc.), while at the same time it is possible to have different Build Actions for each individual item. A library of different Build processes can be stored in the SDM system, per discipline and target solver.

Optionally, saving an item in the SDM repository can be prevented if its Build process is not successfully completed.





3.2.4.4. Pre/Post-Save script actions

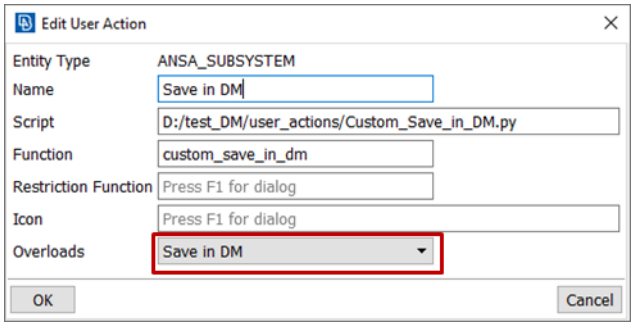
The **PrePostSaveRepresentation()** function can be optionally triggered upon saving a Model Browser container in DM to ensure that some user checks defined in a script function are always executed during **Save in DM**. To activate the execution of this function, the following settings must be configured in the *ANSA.defaults*:

PrePostSaveRepresentationScriptFilePath
This is the path to the Python script that contains the function to be used

PrePostSaveRepresentationScriptFunctionName
The name of the Script function found within *PrePostSaveRepresentationScriptFilePath*. The function can be called by ANSA three times when saving in DM: Before, during and after the save operation. It accepts 3 arguments: • entities: The DM objects that are being saved • deck: The current deck • Info_dict: A dictionary that contains information about the state at which the script function is called: *{"PrePostSaveRepresentationFunctionCallState", state}* The possible values of *state* are: ◦ 0 : Before save representation ◦ 1: During save representation ◦ 2: After save representation

3.2.4.5. Custom Save function

A custom user action can be specified to override the built-in **Save in DM**. In this way, it is possible both to ensure that a user-defined action is executed when **Save in DM** is triggered and to block save under certain conditions.



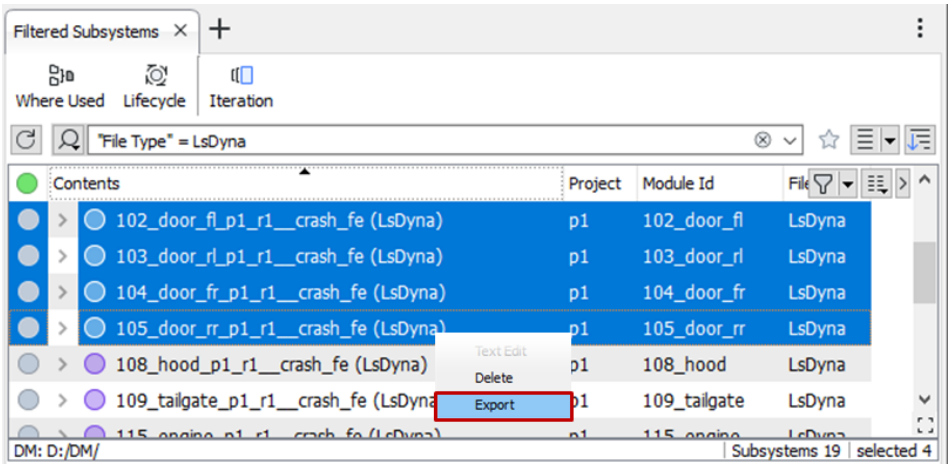
3.3. Export data

All the DM objects can be exported from the SDM system using the **Export** option in the context menu. This functionality is available in KOMVOS and in the DM Browser of ANSA/META.

On Export, the user is prompted to select the directory where the selected DM Objects will be exported.

The Export function is needed to share data with users that do not have access to the SDM system or to transfer data to a different file system, for example in case of HPC submission.

For SPDRM-based systems and Python scripts in particular, the option **Encrypt & Export** is available that encrypts the scripts with BETA's encryption algorithm and produces pyb files.



Note that there's no need to export data only to edit their representation files, as the direct **Edit** function (see paragraph 3.4) should be used for this purpose.

3.3.1. Exported files

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

For compound DM objects, i.e. DM objects that contain other DM objects, if the exported item is tagged in a way that shows it requires the files of its contents for its proper definition, its export will trigger the export of its contents too. Characteristic examples of compound DM objects are the Simulation Models, the Loadcases and the Simulation Runs:

- A Simulation Model consists of Subsystems and Library Items
- A Loadcase consists of Library Items and Subsystems
- A Simulation Run consists of a Simulation Model and a Loadcase

The tag that a compound DM object requires the files of its contents for its proper definition is an Additional Attribute named "*Save Option*". When this attribute has the value "*File with References*" then the contents will also be exported.

Note that this functionality is applied recursively: Exporting a Simulation Run may trigger the export of a Simulation Model and a Loadcase which, in turn, may trigger the export of their contents.

Hierarchy of Compound DM object	"Save Option"	Exported items
DM Name body_VH1_A01__nvh_001 101_BODY_VH1_A01_ALL_nvh_fe__001 102_DOOR_FL_VH1_A01_ALL_nvh_fe__001 900_CONNECTIONS_VH1_A01_ALL_nvh_fe__001	Monolithic file	body_VH1_A01__nvh_001.nas
	File with References	body_VH1_A01__nvh_001.nas 900_CONNECTIONS_VH1_A01_ALL_nvh_fe__001.nas 102_DOOR_FL_VH1_A01_ALL_nvh_fe__001.nas 101_BODY_VH1_A01_ALL_nvh_fe__001.nas

3.3.2. Handling of absolute paths within keyword files

In case of file-based DM, the solver representation files of compound containers may contain references to absolute paths. Such representation files can be accessed and utilized directly through the DM repository without export. However, they are not portable since they cannot be used in a different file system. To tackle this limitation, if the selected DM objects contain solver files with absolute paths in their include references, these paths are automatically converted to relative paths upon export, and the files referenced are all placed in the same directory.

Absolute paths in the original representation file of the Simulation Model

```
INCLUDE 'D:/test_DM/Subsystems/101_BODY/ALL/VH1/A01/001/nvh_fe/-/
        Nastran/repr/101_BODY_VH1_A01_ALL_nvh_fe__001.nas'
INCLUDE 'D:/test_DM/Subsystems/102_DOOR_FL/ALL/VH1/A01/001/nvh_fe/-/
        Nastran/repr/102_DOOR_FL_VH1_A01_ALL_nvh_fe__001.nas'
INCLUDE 'D:/test_DM/Subsystems/900_CONNECTIONS/ALL/VH1/A01/001/nvh_fe/-/
        Nastran/repr/900_CONNECTIONS_VH1_A01_ALL_nvh_fe__001.nas'
```

Relative paths in the modified representation file of the Simulation Model after Export

```
INCLUDE '101_BODY_VH1_A01_ALL_nvh_fe__001.nas'
INCLUDE '102_DOOR_FL_VH1_A01_ALL_nvh_fe__001.nas'
INCLUDE '900_CONNECTIONS_VH1_A01_ALL_nvh_fe__001.nas'
```

In case of SPDRM or file-based DM with relative paths (‘*support_solver_relative_paths: yes*’) the referenced paths inside the repository are relative and this conversion is not required.

3.3.3. The use of the xml metadata file during input

Through the data manager it is possible to modify the metadata of a DM object without editing its representation file. Such modifications, which are not directly reflected to the representation file, are communicated to ANSA through an xml file which is written upon DM object export.

Upon reading in ANSA the representation file of a DM object, either in form of a solver file or an ANSA database, ANSA searches for an xml file with the same name as the file to read. In case that such file is detected, ANSA reads the metadata available in the xml file and updates the attributes of the respective item within the ANSA database. In this way, any modification made on the attributes of a DM object through the data manager, are reflected to the respective item inside ANSA.

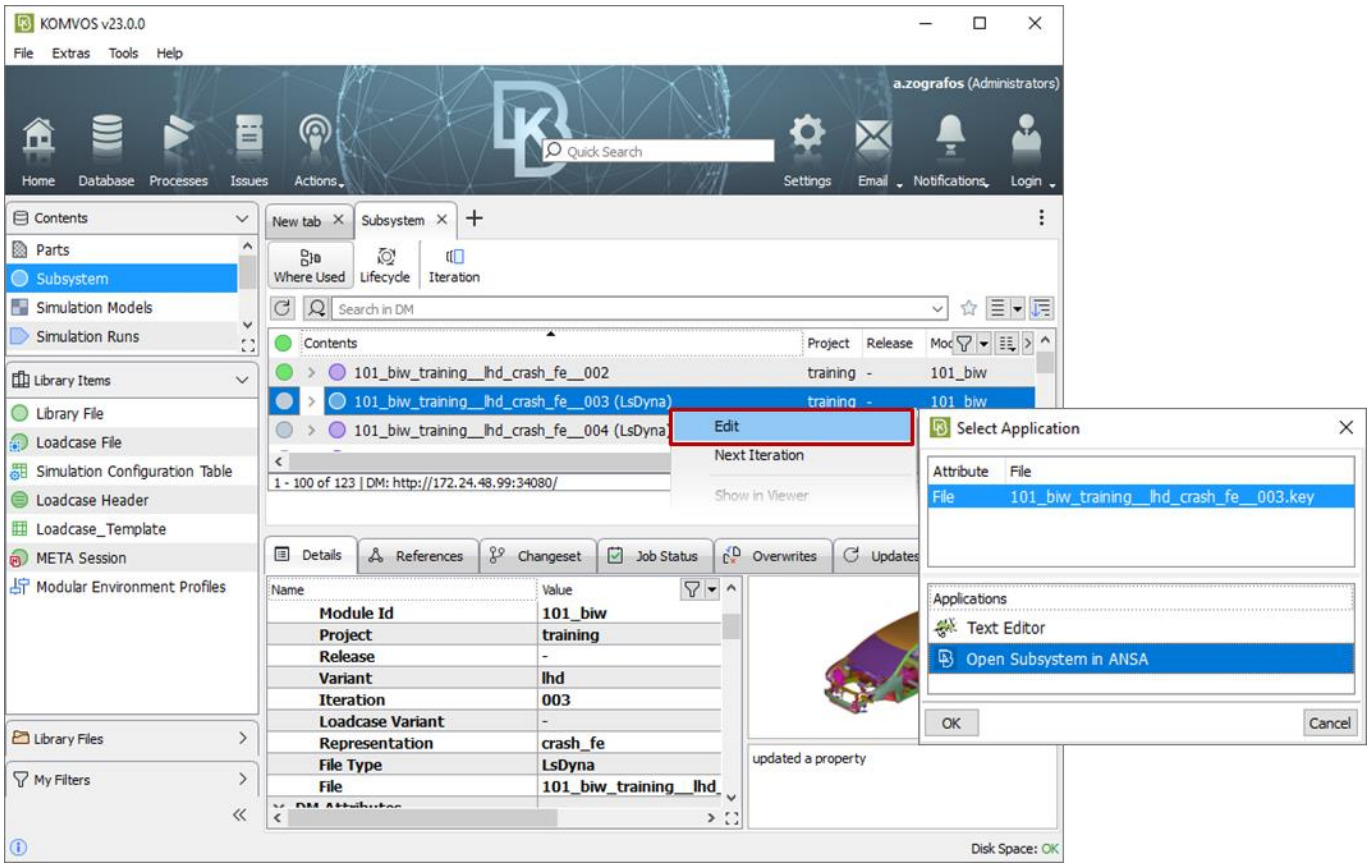
Note that the xml file is not exported in case of single file library items.

3.4. Edit and View data

Through the data manager a DM object can be opened in an editor for view or edit purposes. Upon edit, the selected DM object is downloaded in a temporary folder and its representation file is opened in the editor. Different capabilities are available depending on the SDM back-end.

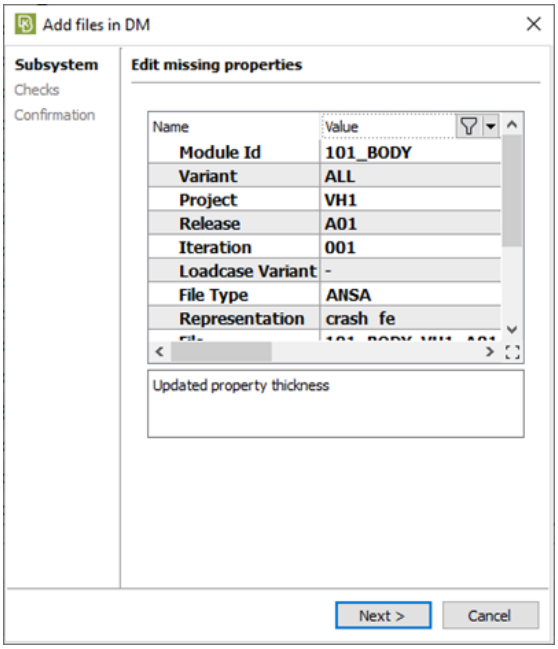
3.4.1. Editing data in SPDRM back-end

When SPDRM is used as the SDM back-end, a DM object can be selected in the data manager and be edited in the preferred editor or in the system default application.



The available applications that can be used to edit a DM object (mimetypes), are defined per data type and file extension by the SPDRM system administrator. In case no such mapping exists, the system default application for the respective file type is used. For more information please refer to SPDRM Administrator's guide paragraph 6.2.

If the user modifies the file that was opened, saves the changes and quits the application, SPDRM detects that the item's representation file was modified and the *Add files in DM* wizard opens, in order to decide the versioning policy for the edited DM item.



Optionally, it is possible to demand a user comment upon update by marking the "Comment" attribute of the respective data type in the *dm_structure_TBM.xml* with the following key:

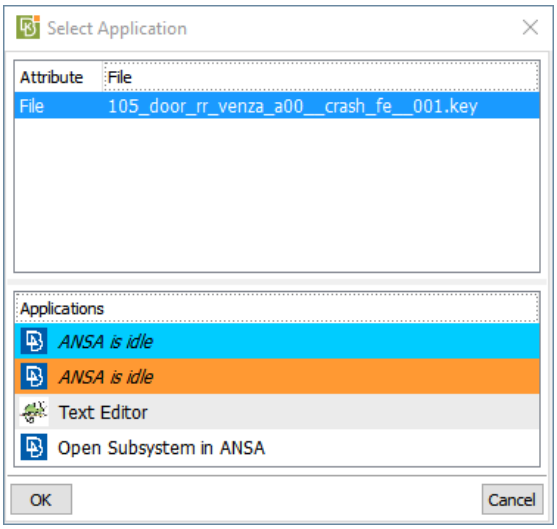
```
commitComment="YES"
```

3.4.1.1. Reusing ANSA/META sessions

There's a built-in mechanism that keeps track of every ANSA/META session that was launched for the editing of DM objects through KOMVOS, facilitating the reusability of those sessions when editing another DM item, and eliminating potential waiting time during the applications' launch.

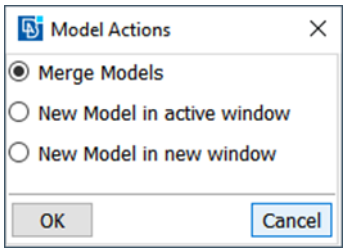
When editing a DM Item and an open ANSA/META session already exists, the auto-reusability mechanism takes effect and the *Select Application* window also includes existing ANSA/META sessions that can be selected as editors.

The color indication in the *Select Application* window is identical to the color indication that each ANSA/META session has in the toolbar and helps the user detect the session in which the item will open, so that proper actions for the existing session content is requested.



If the user selects to reuse an existing ANSA session, the *Model Actions* window pops-up to select the desired merge options.

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



3.4.1.2. Viewing multiple items

Through KOMVOS it is possible to select multiple items and open them in the editor for view purposes. Opening multiple items is only available for mimetype applications which have been configured accordingly (i.e. the "Allow Multi Edit" flag is active). For more information, please refer to paragraph 6.2 of SPDRM Administrator's guide.

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

3.4.2. Editing data in file-based DM

In case of file-based SDM back-end, it is possible to open and view the representation file of a DM object in an editor, but it is not possible to make any modification to the representation file. In case of solver files, this is possible through the **Text Edit** context menu option, whereas for ANSA files the **Open in ANSA** option can be used.

Reusing an ANSA/META session or viewing multiple items is not supported in case of file-based SDM systems.

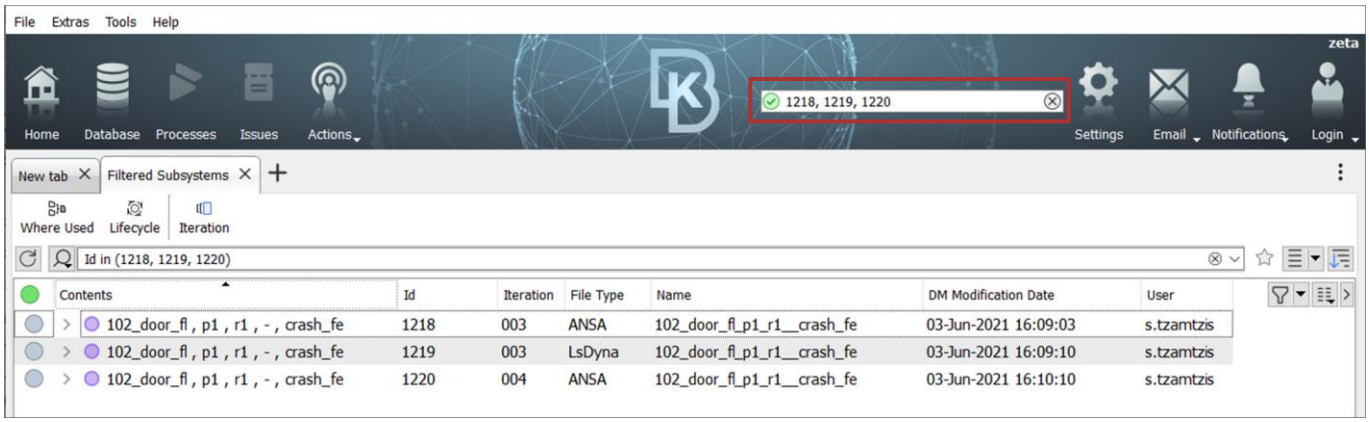
3.5. Data search and data focus

BETA's SDM solutions offer effective and user-friendly functionality to enable the identification and focus on data of interest. Several possibilities are there, all of them useful in different cases:

- **Quick search**: Search based on id or Name or Module Id (paragraph 3.5.1)
- **BETA QL search**: Search based on metadata or data relationships, in BETA Query Language (paragraph 3.5.2)
- **Free text search**: Search based on metadata or relationships, in natural language (paragraph 3.5.3)
- **Data tree focus**: Focus on particular branches of simulation data (paragraph 3.5.4)

3.5.1. Quick Search

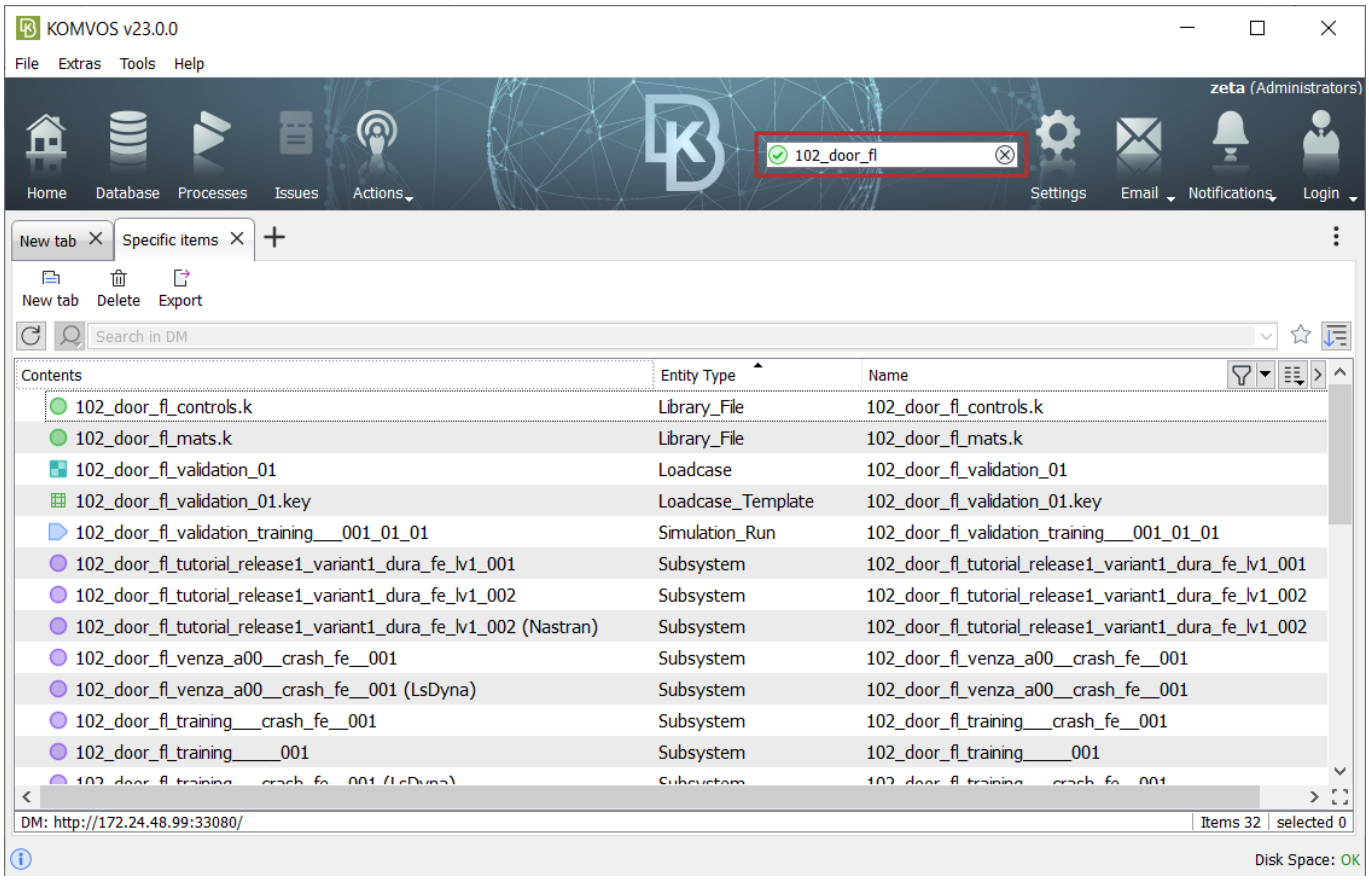
All DM Objects have a unique identifier in the DM, their Id. It is possible to quickly search for a single or multiple DM Objects by providing their Id.



Note that in file-based DMs the plain Library Files, contrary to the DM Objects, do not own an Id. However, with SPDRM back-end all entries of the DM Repository hold a unique, searchable Id.

The same search tool can be used for quick searches based on the Name or Module Id of DM items.

The search results are shown in a new tab.

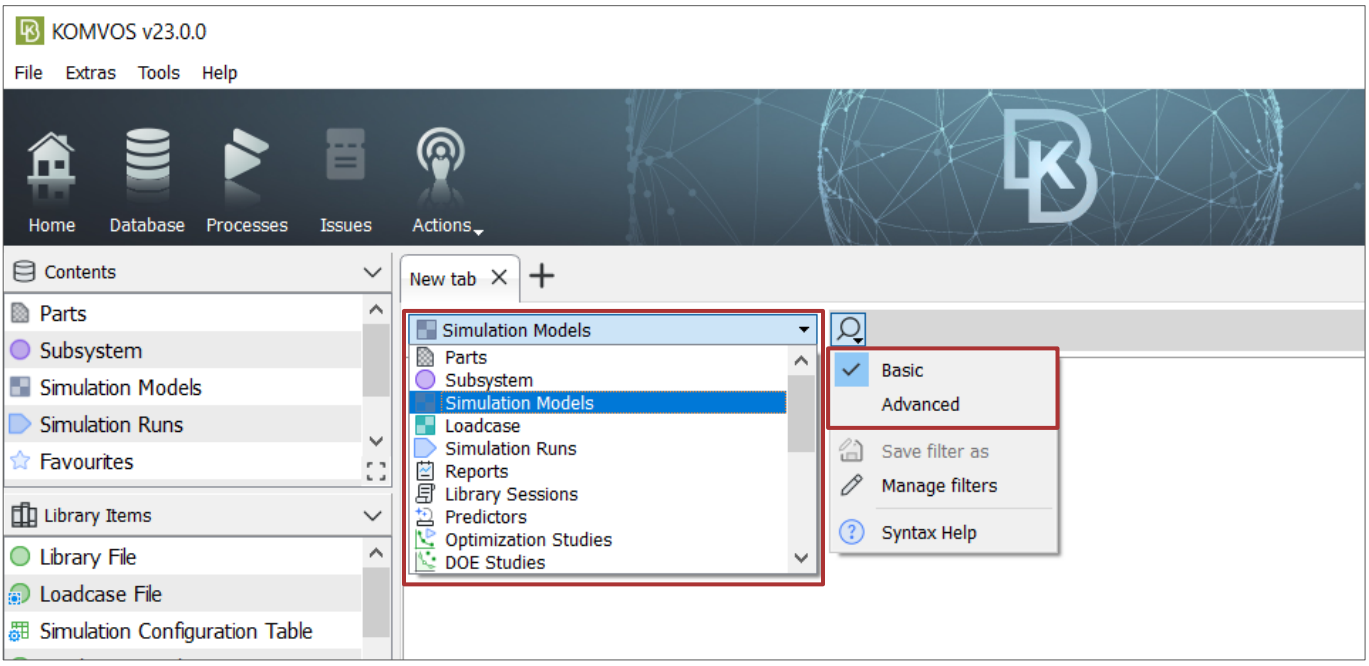


3.5.2. BETA QL search

Searching the database based on the metadata of the DM Objects is achieved with the aid of BETA QL, a powerful query language that is common for all the products of the BETA Suite. BETA QL search supports two modes for the definition of queries in the **Search in DM** field: **Basic** and **Advanced**.

The Search in DM field is scoped to a particular DM Object type. Therefore, in the default tab labeled “New tab”, the user first needs to define the DM Object type to search for. All other tabs are already type-specific, so the definition of type is not needed.

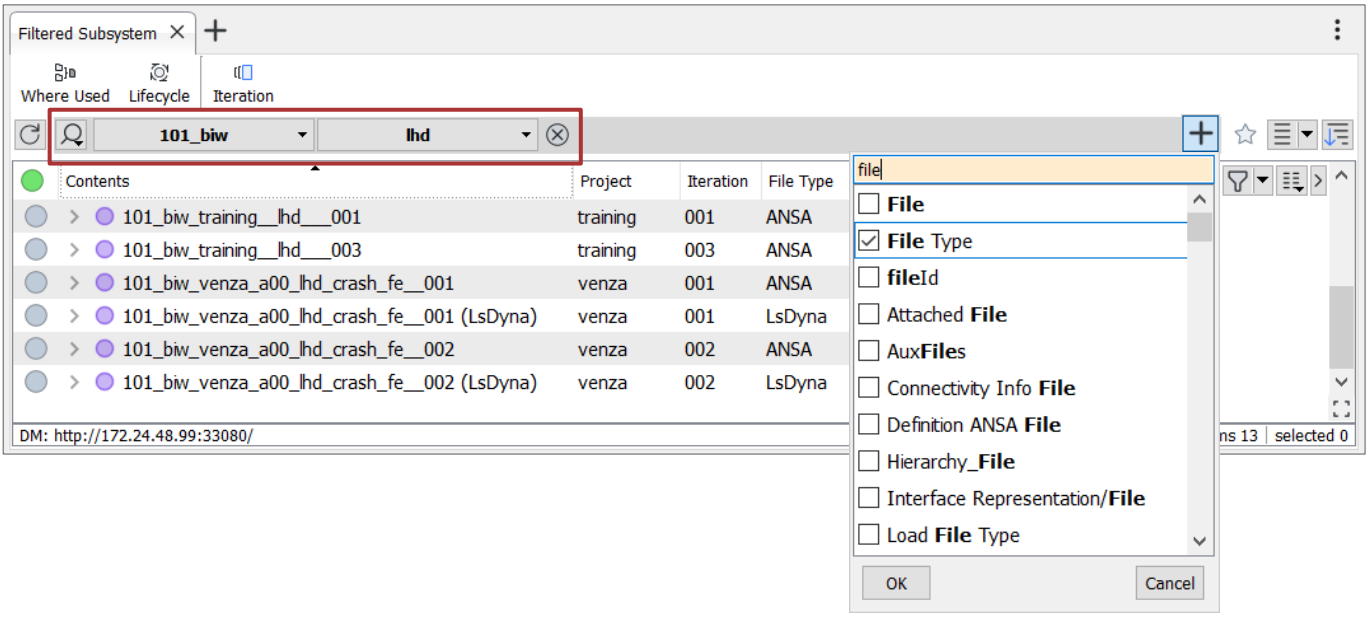
3.5.2.1. Attributes-based queries



The query mode can be selected by pressing the **Search** button  on the left of the query definition area, activating the respective option.

In the Basic mode, a query expression is built as a sequence of forms that the user fills in (e.g., on Module Id, on Variant, on Project), selecting from a list of valid values in each form.

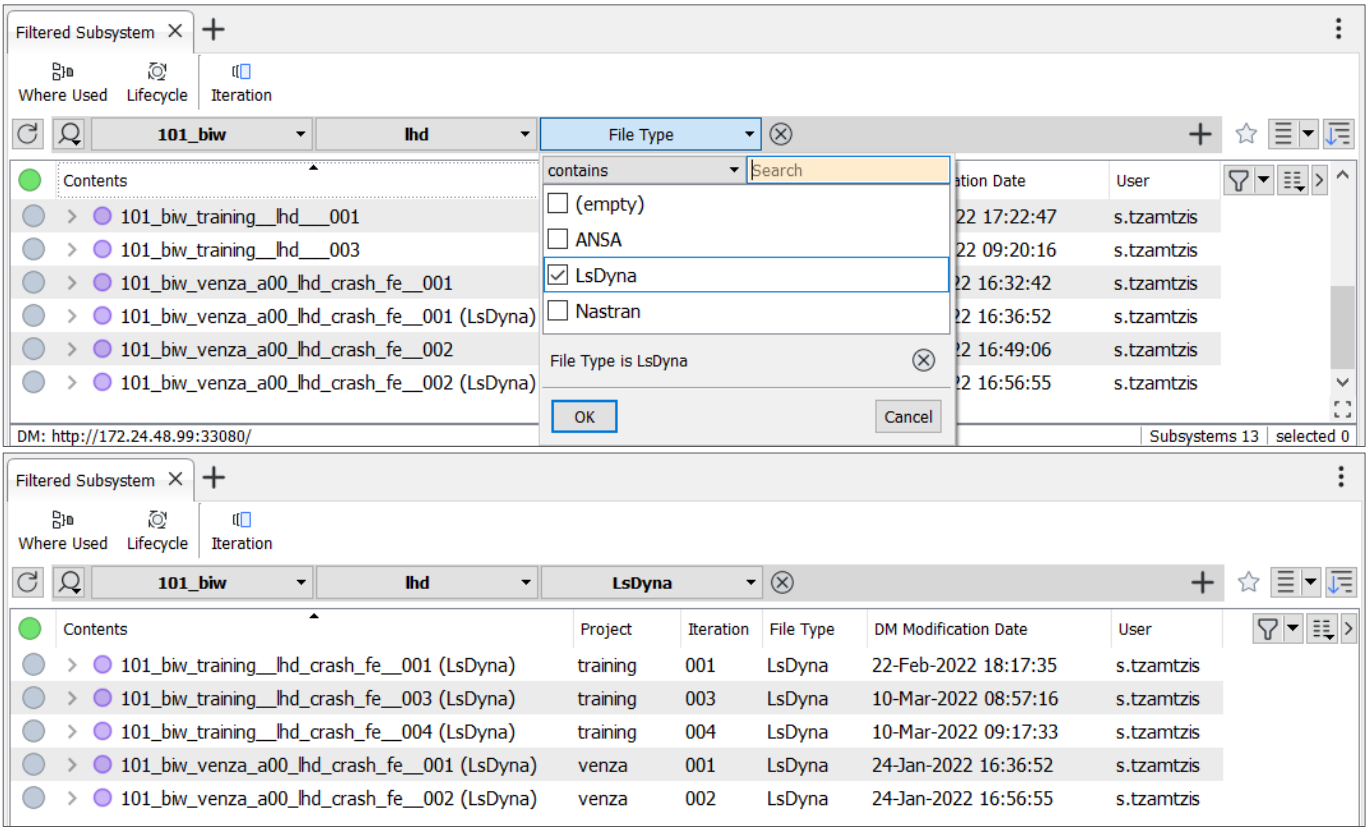
A new form can be added by pressing the **Add** button that lists all available properties/attributes of the DM Object Type. One or more items can be checked. The user can also start typing in the search field to narrow down the listed attributes and easily find the one to add.



By pressing the **OK** button, the selected form is added in the **Search in DM** field.

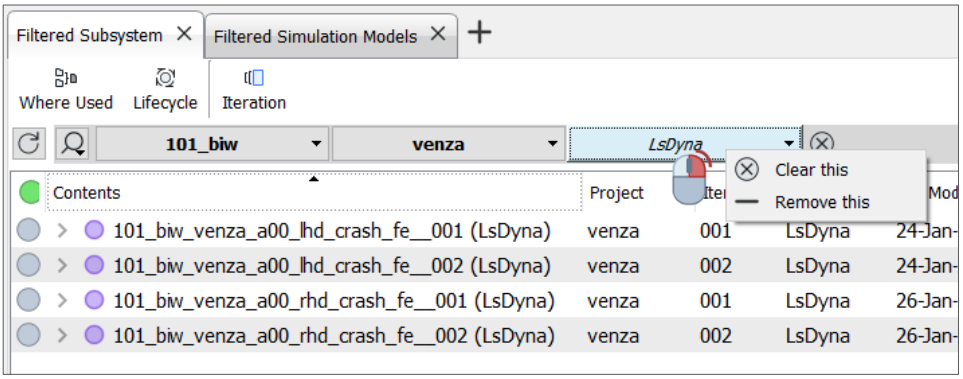
Then, the user can query for an exact match by selecting among the suggested values listed in the pop-up menu of each form or type some text in the Search field and adjust the operator accordingly in order to request a “contains”, “starts with”, etc. match.

When the **OK** button is pressed the selected value will form the search rule and will be combined with the rest, if existing. The DM Objects that fulfill the condition will be listed.

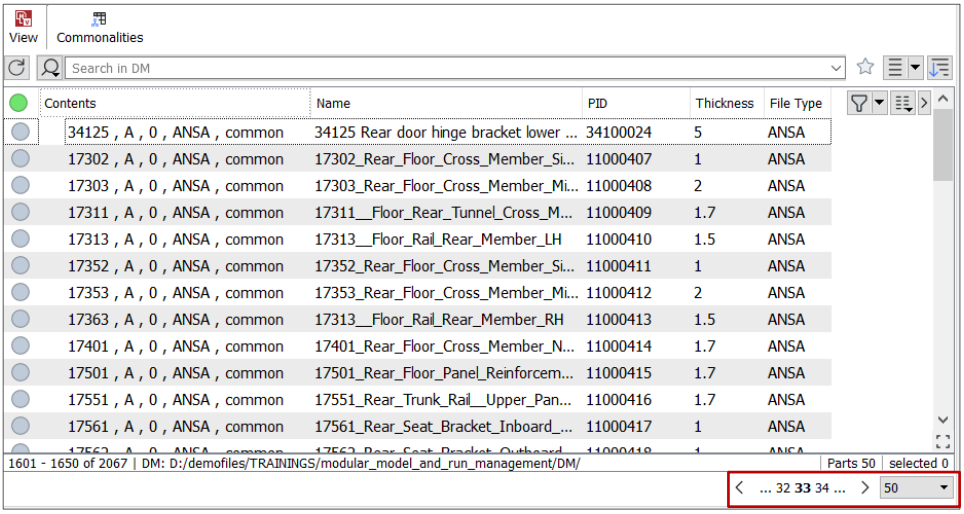


The results of an applied query, are always displayed in a *tab* labelled “*Filtered <DM Object type>*” (e.g., *Filtered Simulation Models*).

The results can be further narrowed down by filling-in more forms. The resulting items are the ones that fulfill all the conditions assuming all queries are combined with an AND.



To clear an applied criterion or to remove it completely, one can use the **Clear this** or **Remove this** option of its context menu.

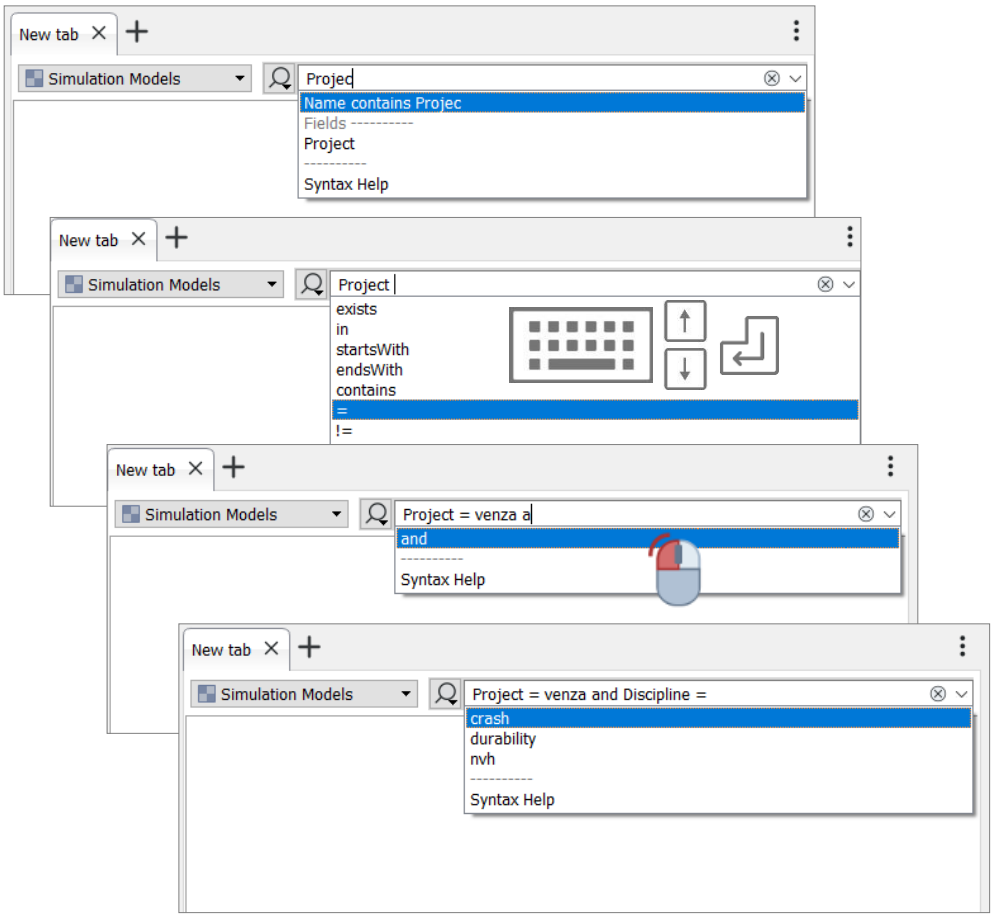


According to the number of the query results, the browser may create multiple pages of results. The user can navigate through the results using the arrows or the page numbers at the bottom right of the list.

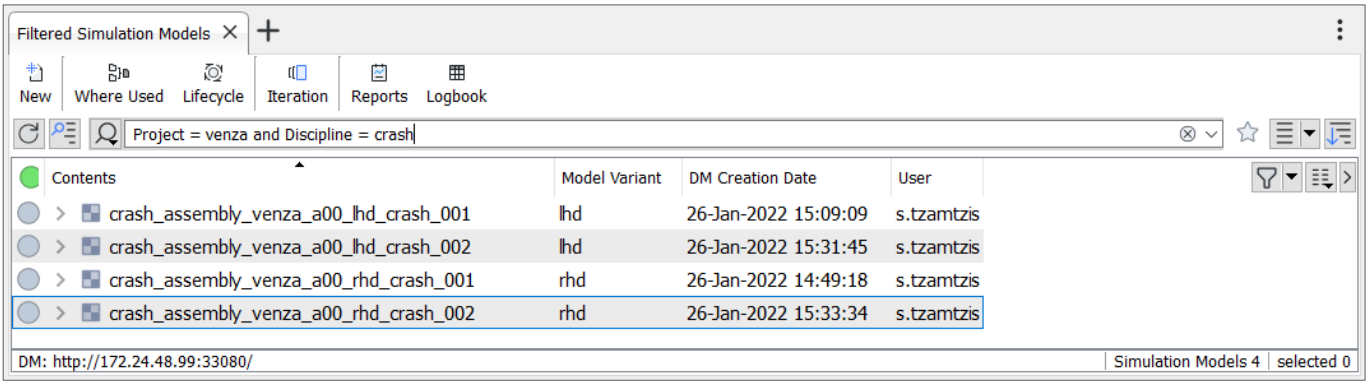
The page size is controllable. For server-based SDM back-ends, the max page size is limited to 250 entries.

In the **Advanced** mode, BETA QL expressions are built as a combination of basic queries consisting of field names (attributes), operators, values, and keywords. For example, to identify all the "door" subsystems created today by user "abc12345", the following expression can be used:

```
"Module Id" contains door AND Created>=startOfDay() AND User=u123456
```

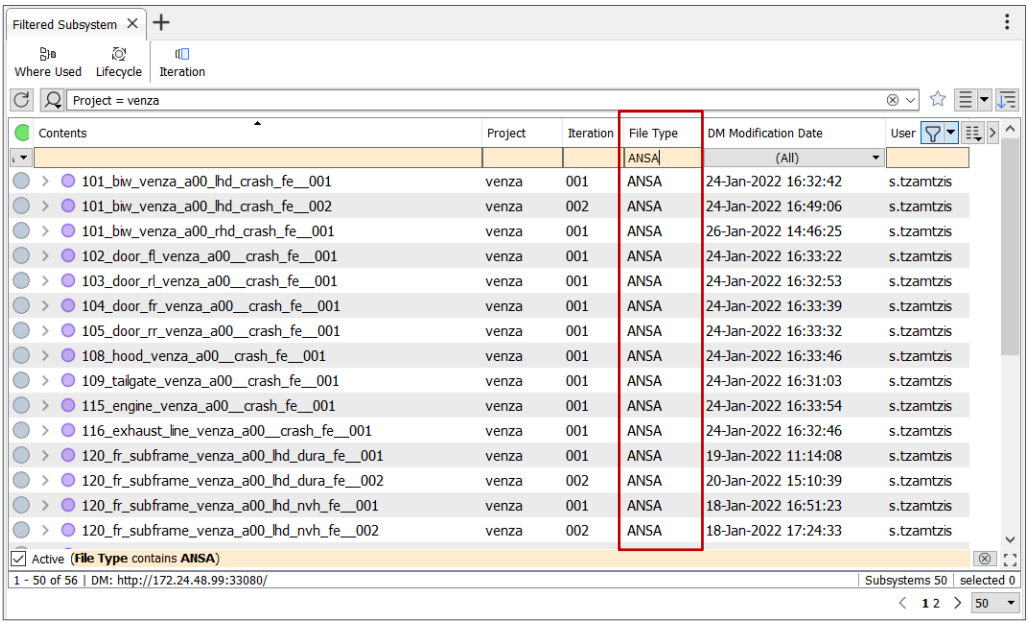


During the typing of the expression and at each step on the way, ANSA offers suggestions that can be navigated with the aid of the arrow keys, and selected with Enter or left mouse click.



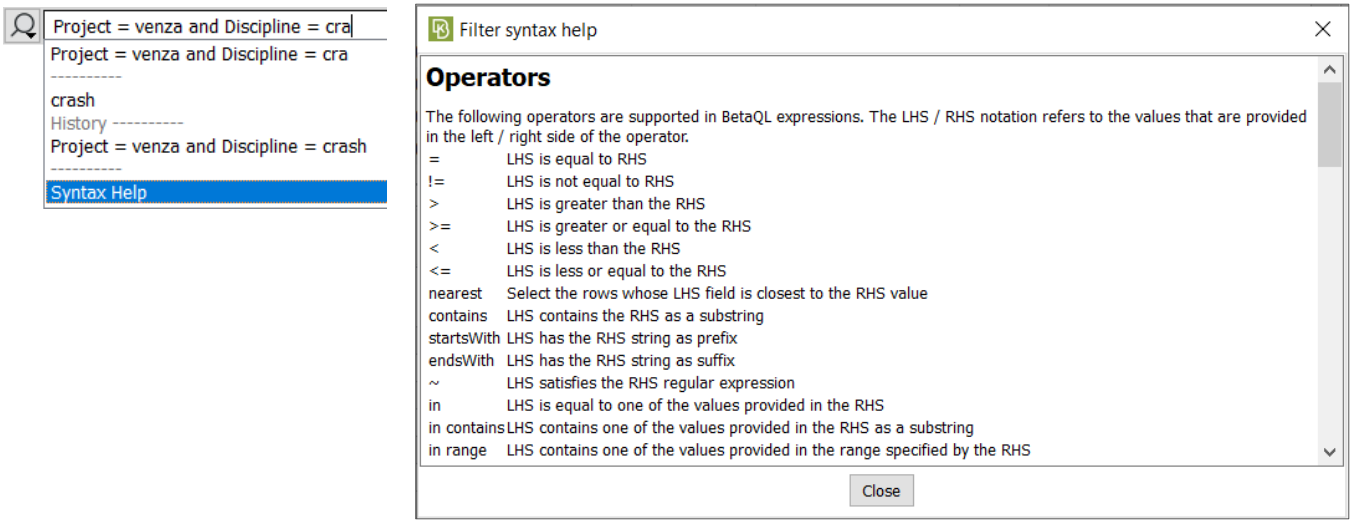
Additionally, the user can paste a value, previously copied to clipboard, directly in the Search in DM field. ANSA automatically tries to comprehend the property being searched and makes suggestions accordingly.

The *column filter* can be used inside the list, to further narrow down the results. It can be activated by pressing the respective button  , similarly to all lists in BETA suite.



It should be noted that in cases of multiple pages of results, this filter is applied only on the active page.

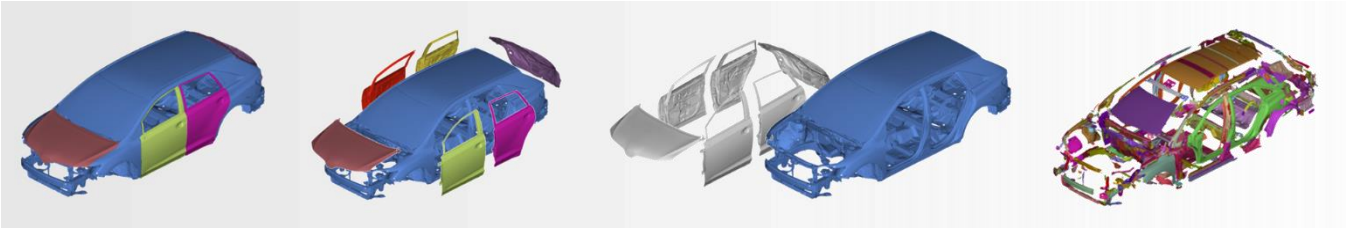
NOTE: While typing a query, the *Syntax Help* option can launch a window that contains several useful tips related to the BETA QL syntax.



3.5.2.2. Relationship-based queries

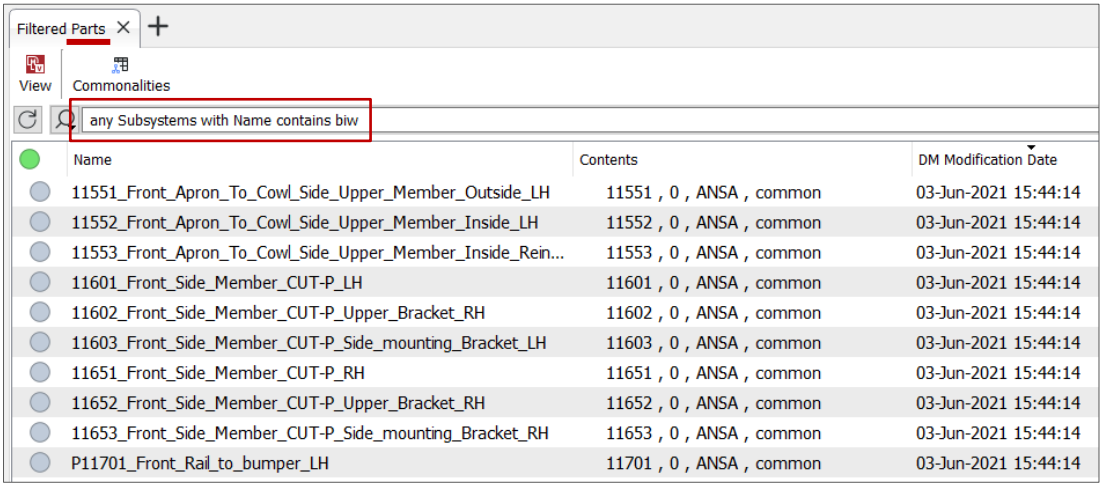
The BETA QL language also supports dedicated syntax for performing relationship-based queries in the DM. The user can search for DM Objects of a specific type, based on metadata of DM Objects that use them.

For example, one can search for all the Parts that are used by any Subsystem with a specific *Name*. This type of search can travel recursively to higher levels of containers, e.g., search for all the Parts that are used by any Simulation Model with a specific *Project*.



This is achieved with the following expression of the BETA QL language:

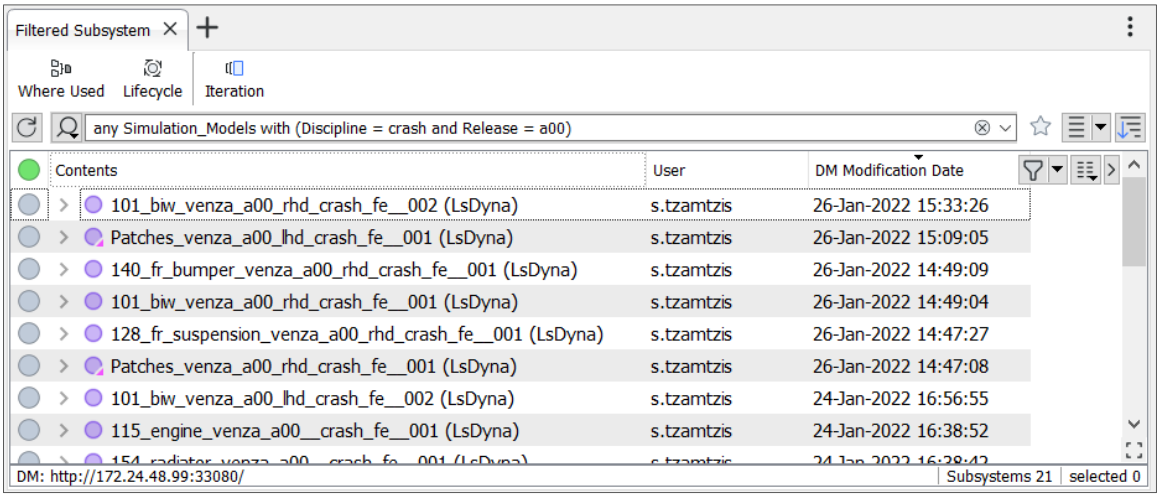
```
any <DM_Object_type> with <condition>
```



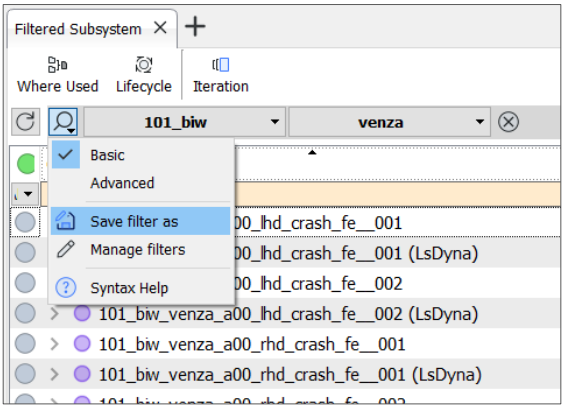
Essentially, the user can query for all objects of a specific type that are used by any other object whose properties/attributes fulfill certain criteria.

For example, one can find all Subsystems used by all the Simulation Models of ***crash*** Discipline and ***a00*** Release. The equivalent BETA QL search must be performed in the Subsystems container and the expression would be:

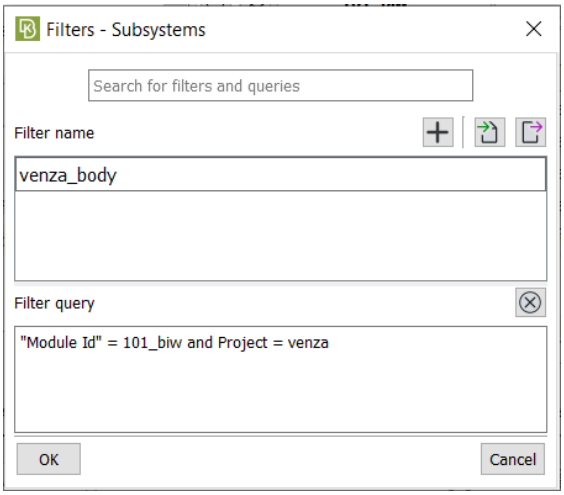
```
any Simulation_Models with (Discipline = crash and Release = a00)
```



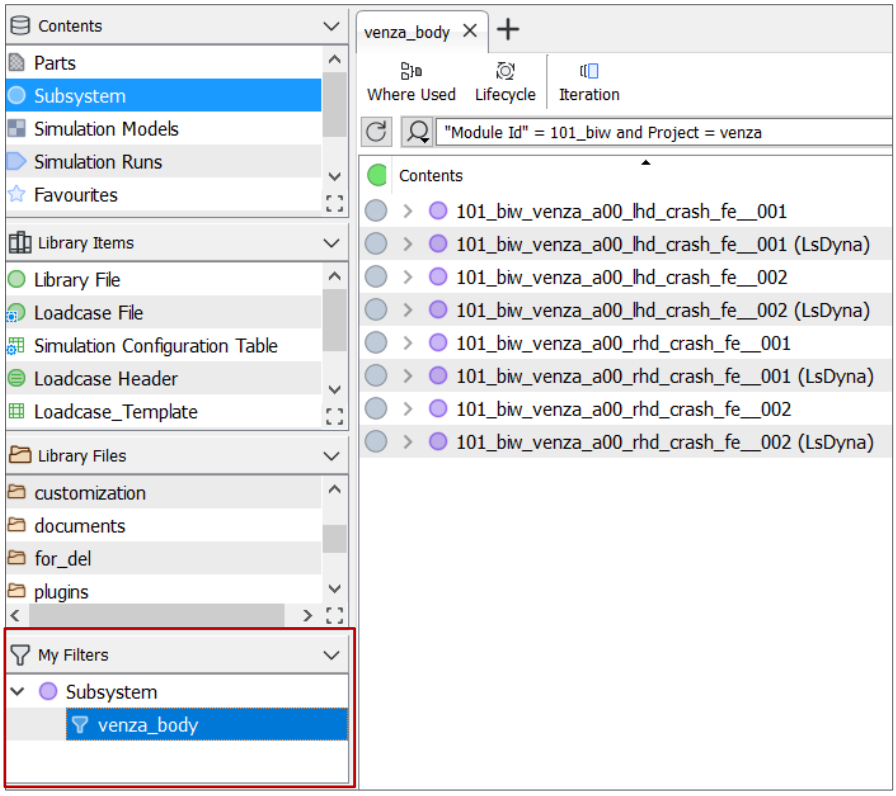
3.5.2.3. Store searches for future use



An applied search can be saved for future use through the **Save filter as** option in the **Search** menu  .



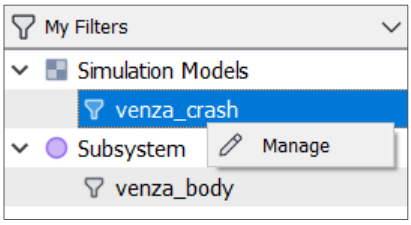
The *Filters* window appears prompting the user to provide a name for this search. When **OK** is pressed, the saved search is added in the *My Filters* section, at the bottom of the Contents Index pane on the left of the browser.

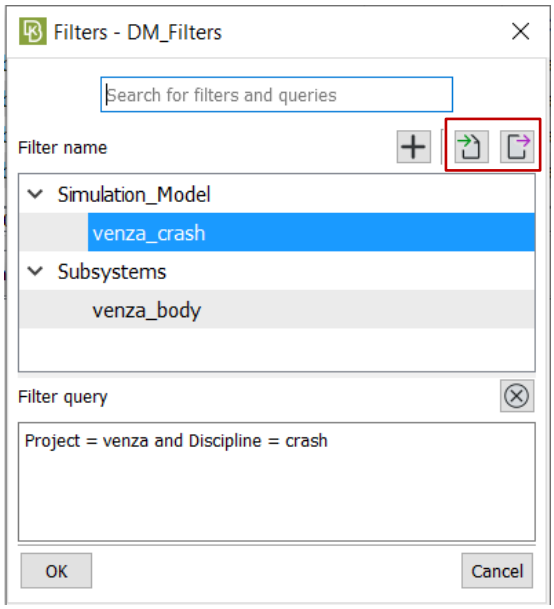


In the *My Filters* sections, saved searches are grouped per DM Object type.

With double click on a saved filter, this is applied on the database, and its results are opened in a new tab of the browser, named after the filter name.

Saved searches can be exported and imported, thus shared among user teams. This functionality is accessed through the *Filters* window. To activate it, select an individual filter in the *My Filters* area and from its context menu press the option **Manage**.



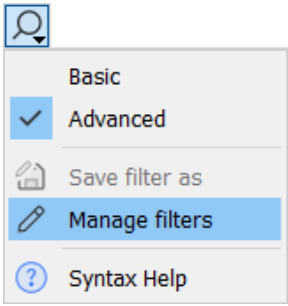


The *Filters* window appears listing all the saved searches per object type. When a filter is selected from the list, the respective query expression appears in the lower part of the window.

Making use of the button **Export**, the filters of the window can be exported in the format of an .xml file.

Such an .xml file can be imported in another environment using the button **Import**.

Note that the same window can be also invoked through the **Manage filters** option in the **Search** menu.



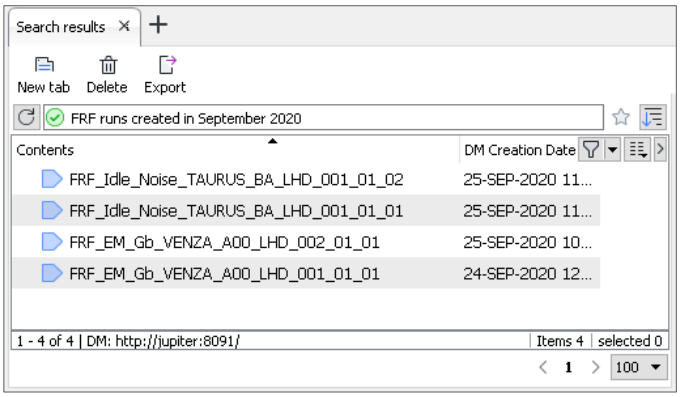
The DM Filters are also saved in the .xml settings file that contains all the GUI settings of the program, inside the user's home folder:

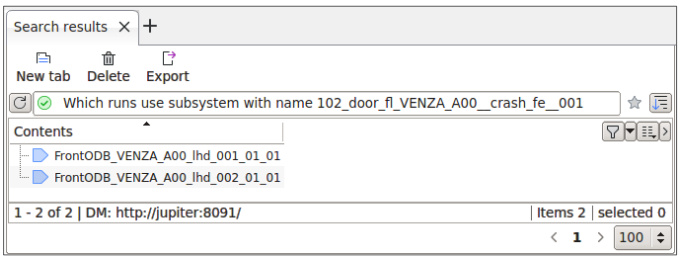
```
~/.BETA/KOMVOS/version_<version_number>/KOMVOS.xml
~/.BETA/ANSA/version_<version_number>/ANSA.xml
```

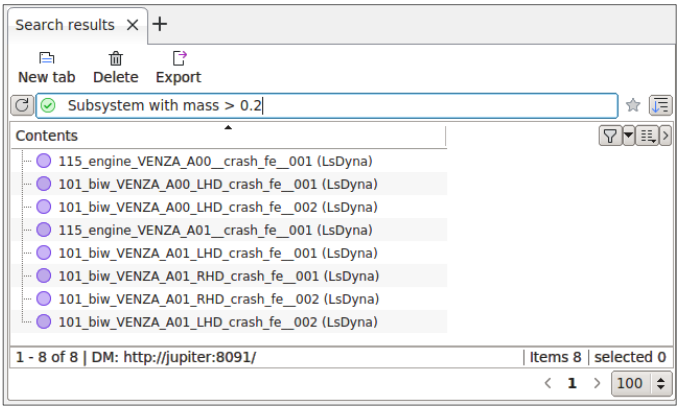
The saving is done automatically upon quit for KOMVOS. For ANSA/META the GUI settings are saved explicitly (unless the auto-save GUI settings option is enabled, in which case they will be automatically saved upon Quit.)

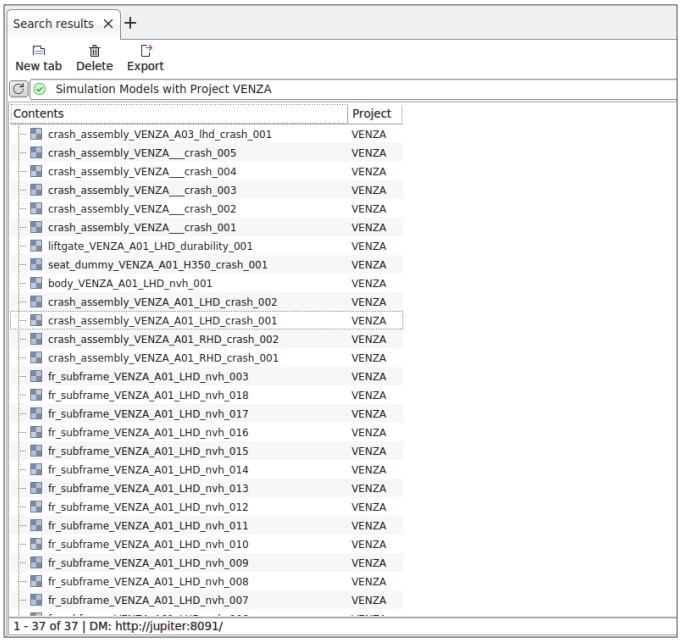
3.5.3. Free-text search

The SPDRM-based solution offers a free text search, complementary to the BETA QL-based searches. This search mode enables the description of queries in natural language letting the user to type usual verbal expression in the search tool of the Data Browser. The capability is offered through a specialized microservice available after a certain configuration of KOMVOS coupled with an SPDRM back-end. This configuration is based on an NLP (Natural Language Processing) model. Through this model the user's simple, usual language expressions are converted to BETA QL queries in the background. The Data Browser will then list all the identified data, in a way similar to what was described in the previous paragraphs. Some examples of such searches are given below.

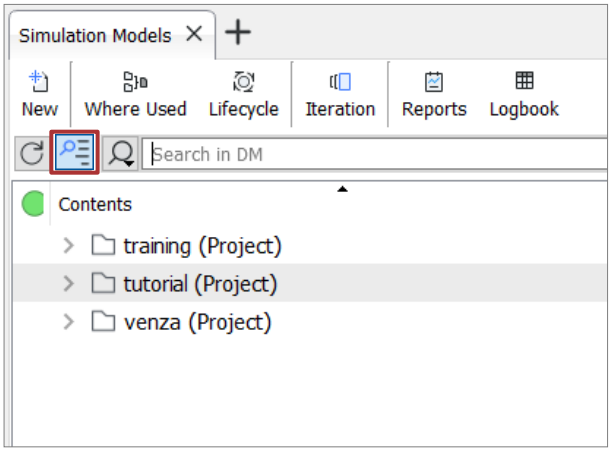






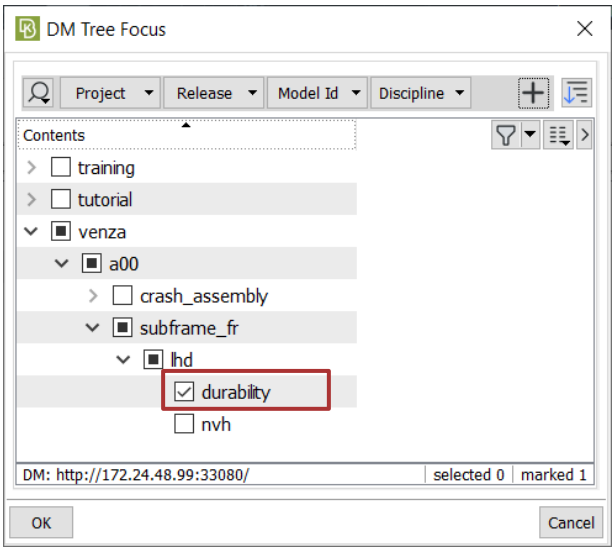


3.5.4. Data Tree Focus

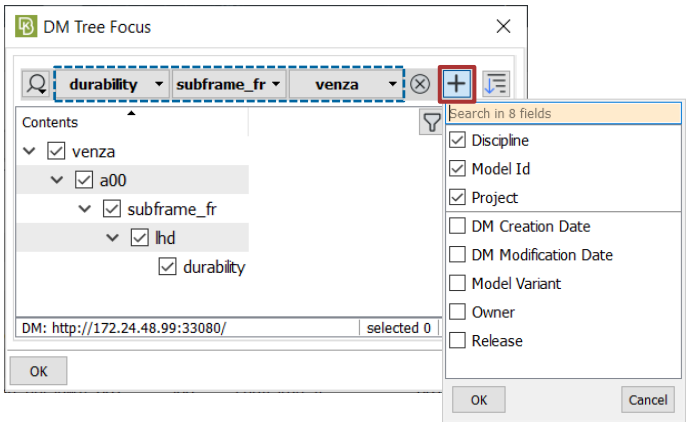


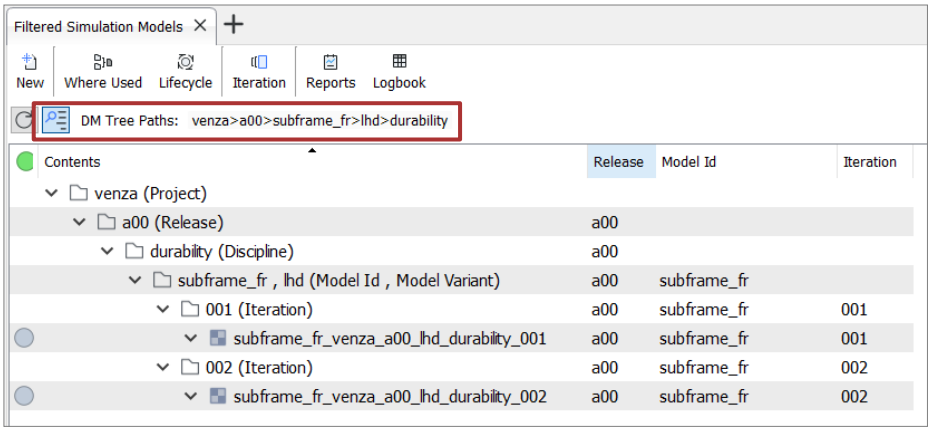
For simulation data like Simulation Models, Loadcases, Simulation Runs and their results, it is necessary to be able to isolate and focus only on parts of the data structure that represent the current working projects.

The **DM Tree Focus** does exactly that: It focuses data browsing on particular branches of the Data tree.



The *DM Tree Focus* window presents in a tree-structure all the attributes of existing simulation data. The user can browse through the tree and activate the branches of interest. This selection is further facilitated by the filters that can be applied at the top filter area, in order to isolate only specific branches.





Finally, when **OK** is pressed, the view is adapted accordingly.

This information is stored in the user settings, so every time the program is launched, the data tree paths selected are reapplied. Note that the Data Tree Focus functionality is only available in lists of simulation data, i.e. Simulation Models and Simulation Runs.

3.6. Viewer

A viewer is embedded to all data browsing tools of BETA's SDM solutions, for the visualization of the selected data objects. Depending on the type of the object selected, this viewer serves as a 3D model Viewer or as a Results Viewer.

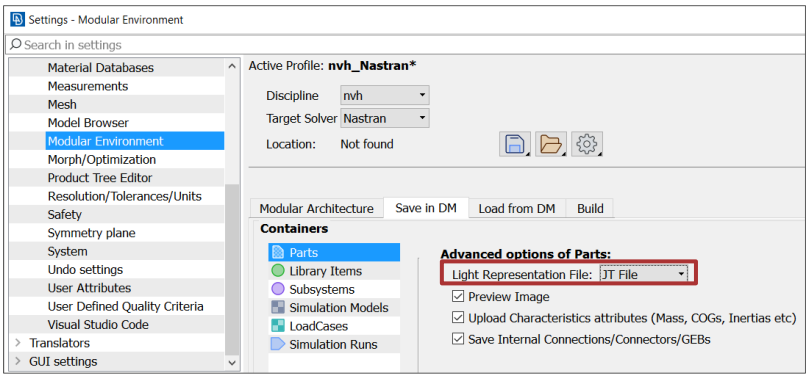
3.6.1. 3D model Viewer

The 3D model viewer can display:

- lightweight jt representations of Parts and Subsystems
- solver keyword files associated with Simulation Runs

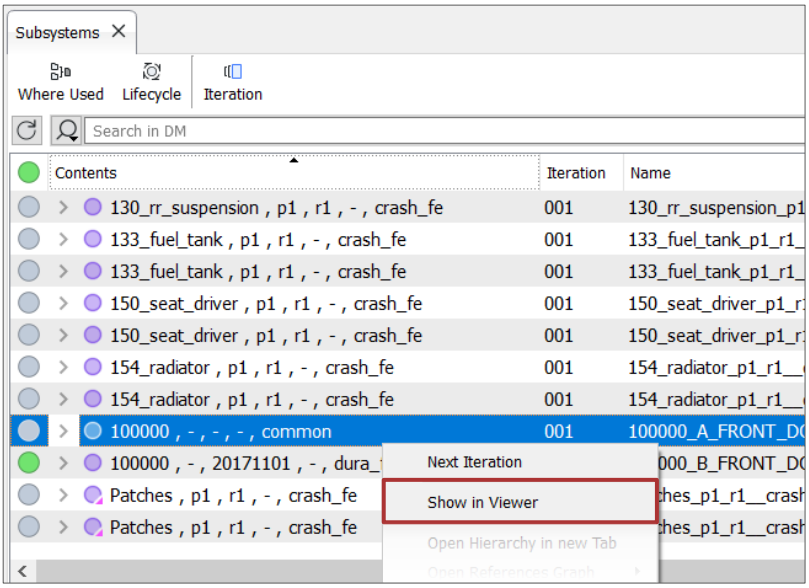
3.6.1.1. JT files

There are two alternatives for importing jt files to the system: With the import of product structures exported from PDM systems, where the jt files are usually the primary representation files of parts, or with Save from ANSA, where jt files can be requested as a light representation, complementary to the primary ANSA or solver representation file.



Within ANSA, the upload of JT Light Representation files may be requested separately for parts and subsystems through the Modular Environment Profile settings.

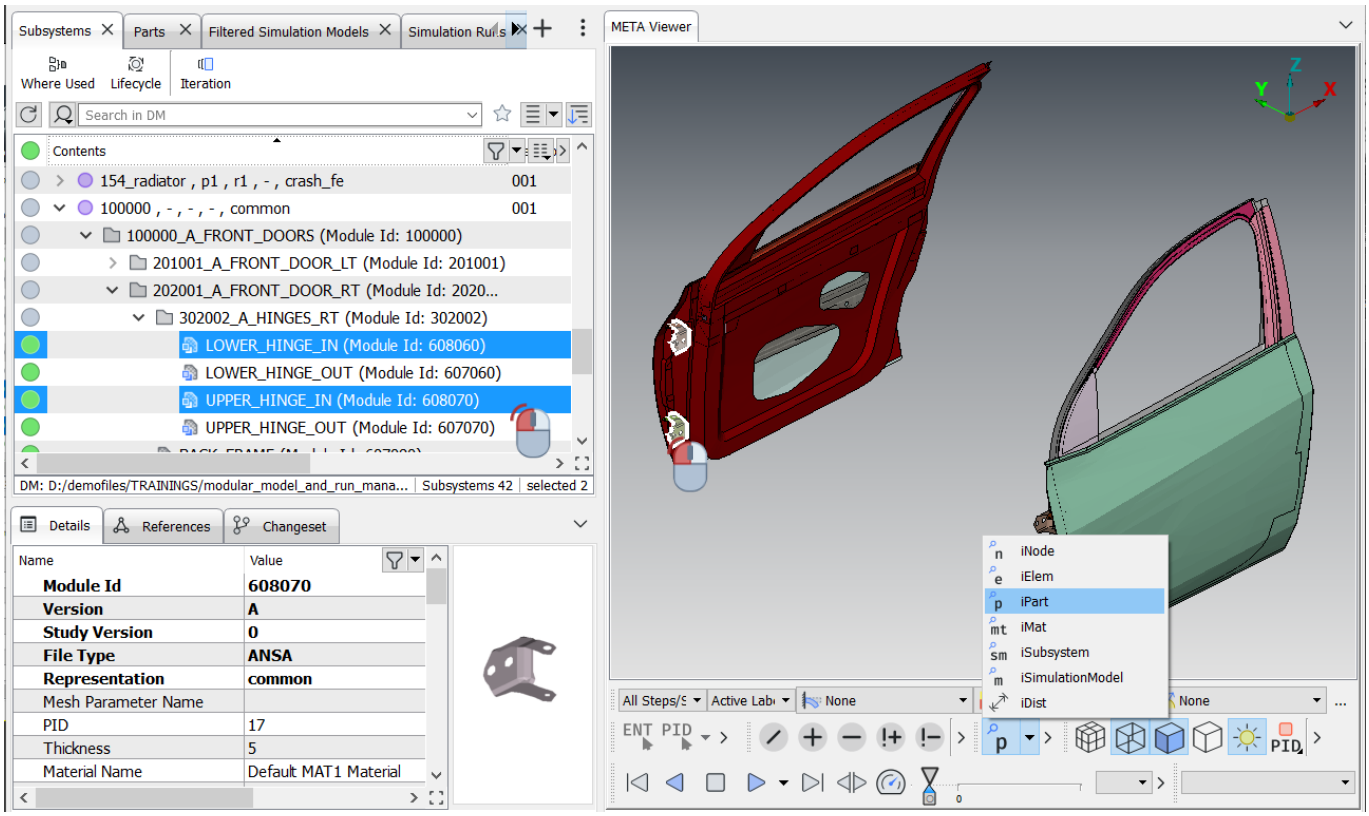
Note that through ANSA it is also possible to save existing jt files as the primary representation of parts, if the absolute path of the jt file is defined in the File attribute of parts in the Model Browser. In such case, File Type JT should be set for the parts.

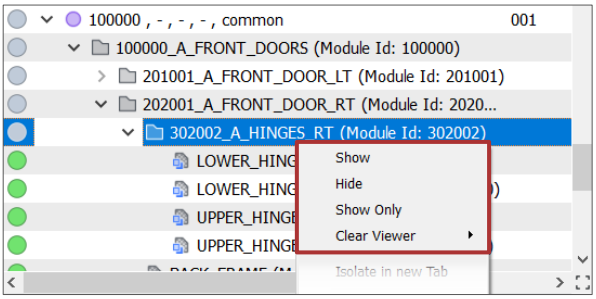


The 3D model viewer is activated through the context menu option **Show in Viewer** of DM Objects. Provided that a file of suitable type is associated with the selected listed item, the embedded META Viewer opens in the right pane of the data browsing tool, displaying the selected DM Object.

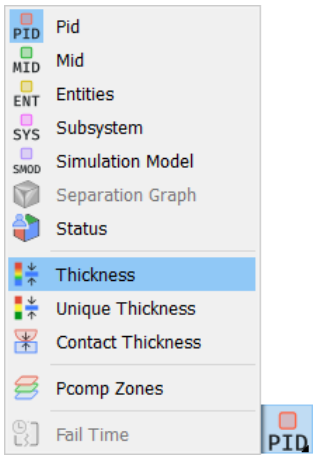
Note that when **Show in Viewer** is requested on a Subsystem that has no monolithic jt file associated with it but has jt files associated with its contained parts, it is the parts that will be loaded in the Viewer instead. In case the Subsystem has its own jt file, it will prevail over those of its parts, rendering drawing and picking per part impossible.

All basic actions for model review are supported in the 3D model viewer. Rotate, pan, zoom and focus functions are available and interoperability with the list view is supported (select items in the list and highlight or isolate in the viewer or select items in the viewer and identify in the list). Getting in identification mode in the viewer (iPart), enables selections in the viewer to be reflected to the list and vice versa.

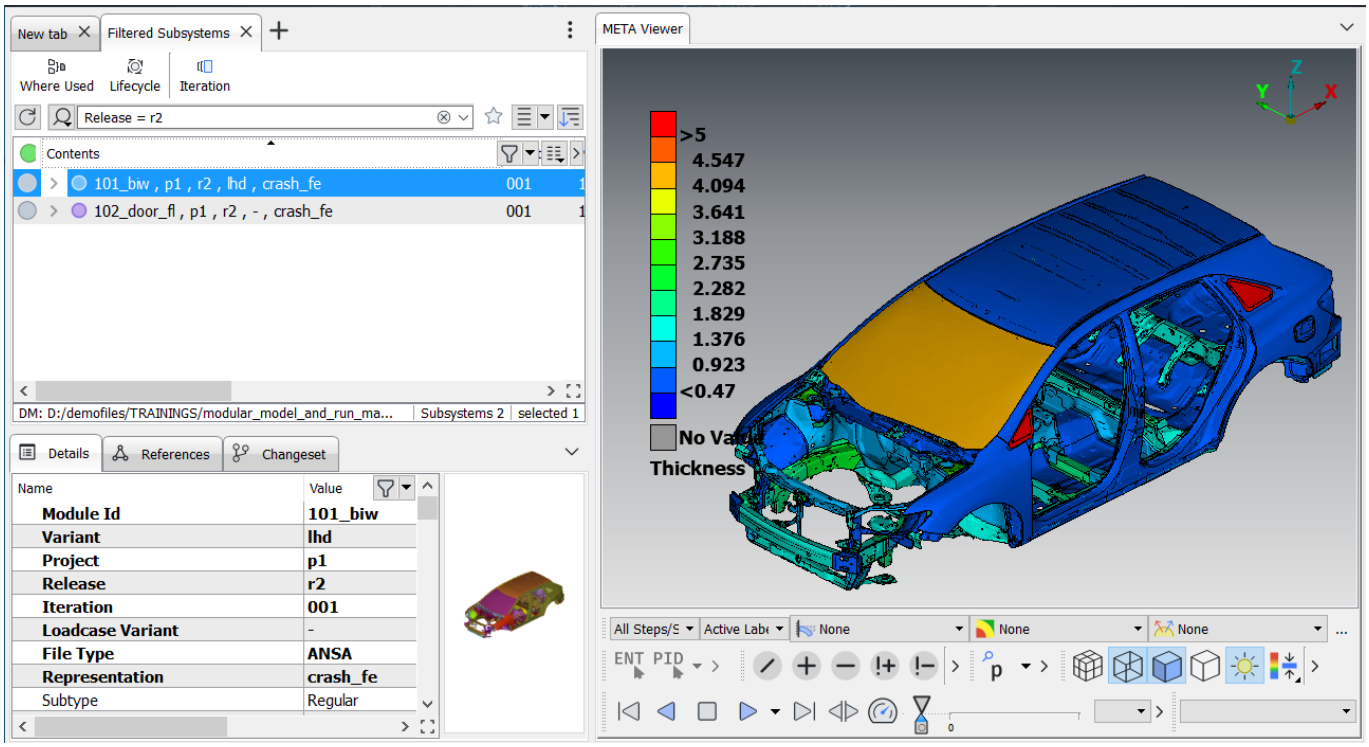




Once the viewer is opened, respective focus functions are available through the context menu of selected items in the list.



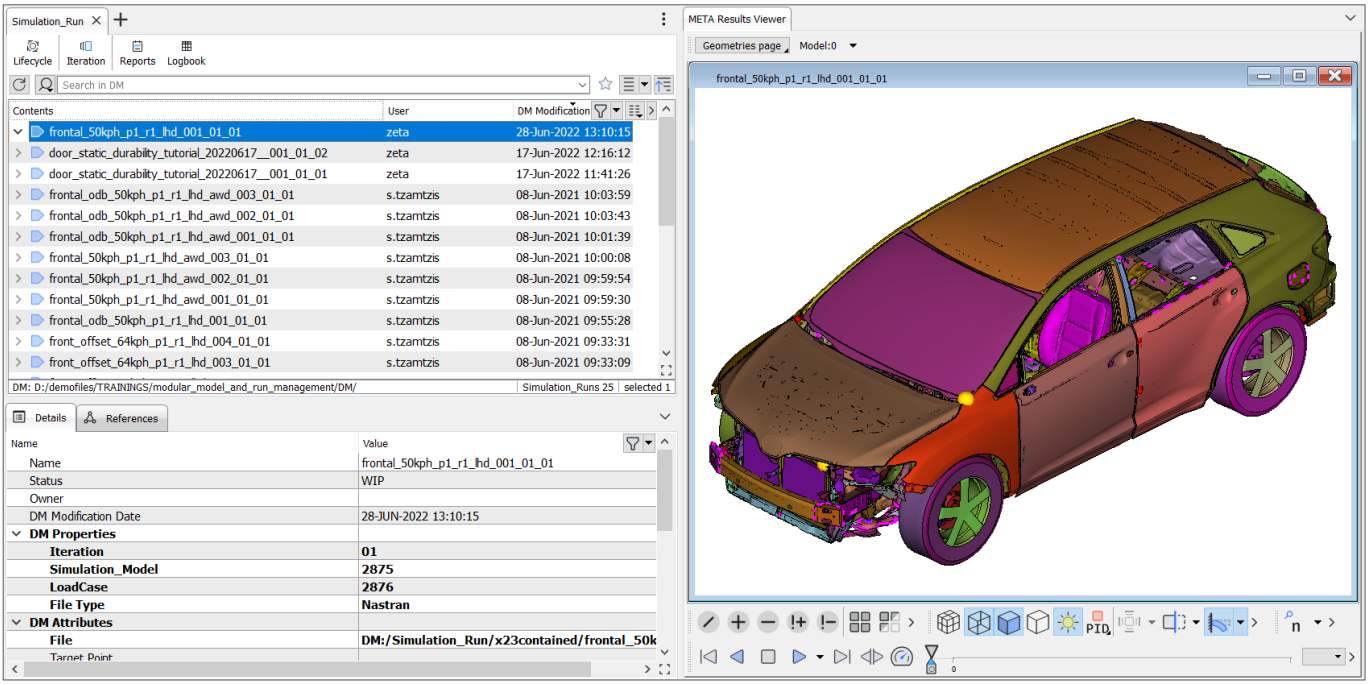
In the 3D model viewer, the model can be colored based on the value of different attributes. By default, when the viewer is opened the model is loaded and colored in PID mode. Pressing the respective button, the user has various other options to perform a visual inspection according to other attributes.



Furthermore, custom visibility modes can be added in the viewer according to needs. This customization is performed through the `dm_views.xml` file. For more details on this topic, please refer to chapter 5 of this document.

3.6.1.2. Solver keyword files

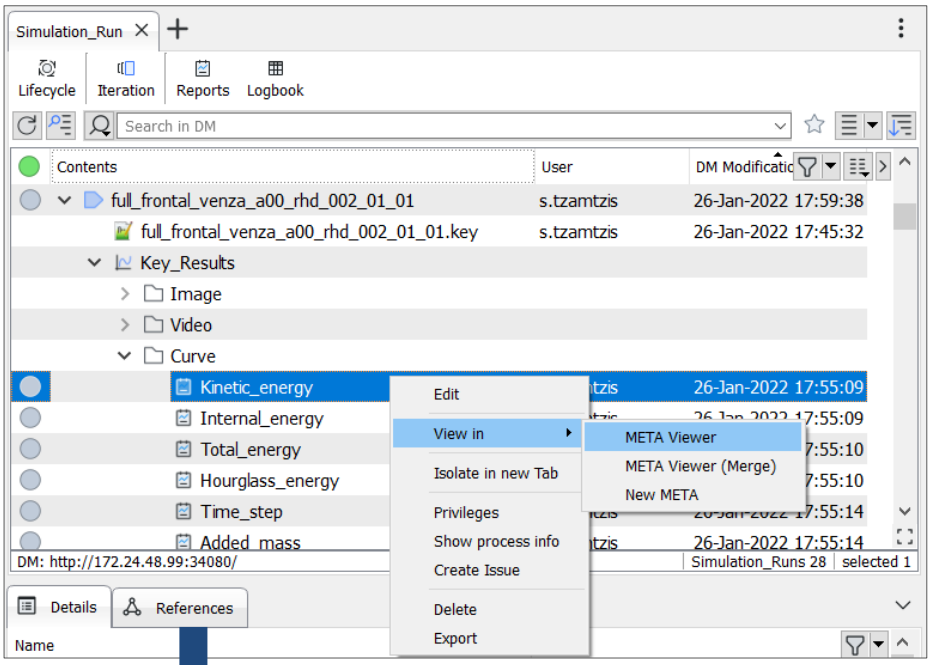
Simulation Runs saved in DM are associated with a solver keyword file according to their File Type. These keyword files can be visualized in the Viewer. With double click on a Simulation Run listed in the data browser, the embedded viewer is launched and the respective keyword file is displayed.



3.6.2. Results Viewer

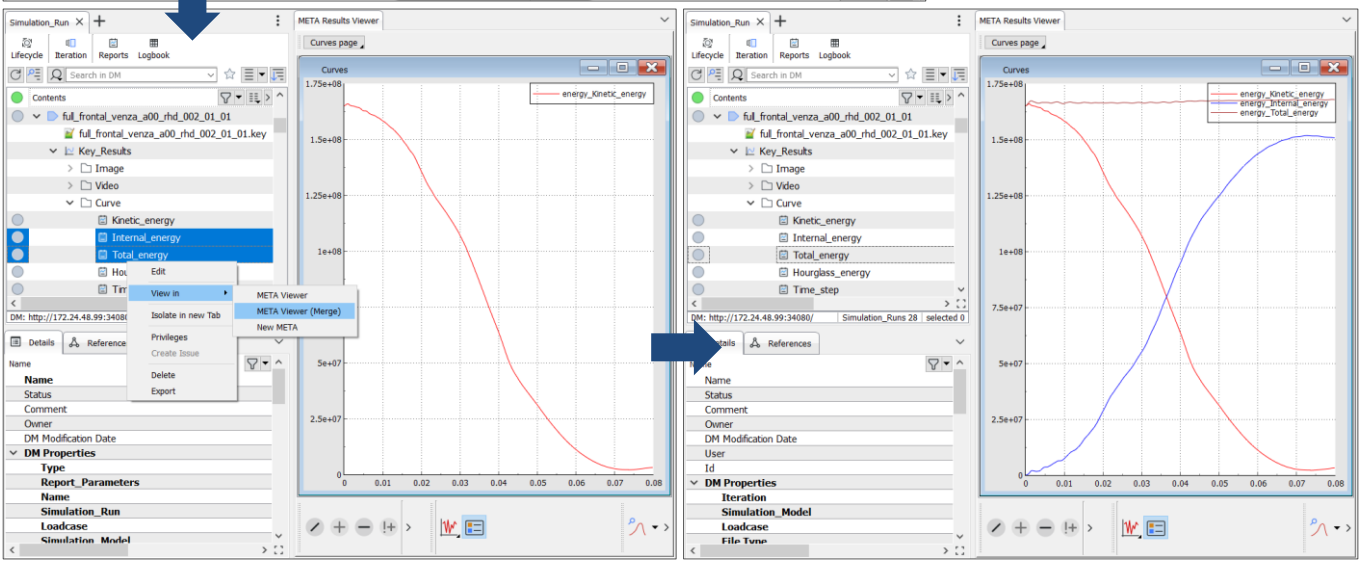
The Results Viewer offers the visualization of processed results of all types (values, images, 2D plots, META DBs, spreadsheets, presentations, videos, etc.) with integrated automatic “overlay” functionality for the effortless comparison of results produced by different Simulation versions.

The Reports of each Simulation Run can be accessed directly under the run in the main list. From this list the user can easily view them in the embedded viewer by activating the option **View in > META Viewer** of their context menu, or simply with double-click.



The viewer is then launched on the right, and the selected result is loaded and displayed. Based on the type of the selected result, further operations may be available in the Viewer.

In case the viewer is already open displaying previously selected reports, the user can merge the selected with the option **View in > META Viewer (Merge)**. Depending on the type of reports, the viewer will pick a suitable displaying mode for them.



For more information about viewing of reports, please refer to paragraph 4.2.2. of this document.

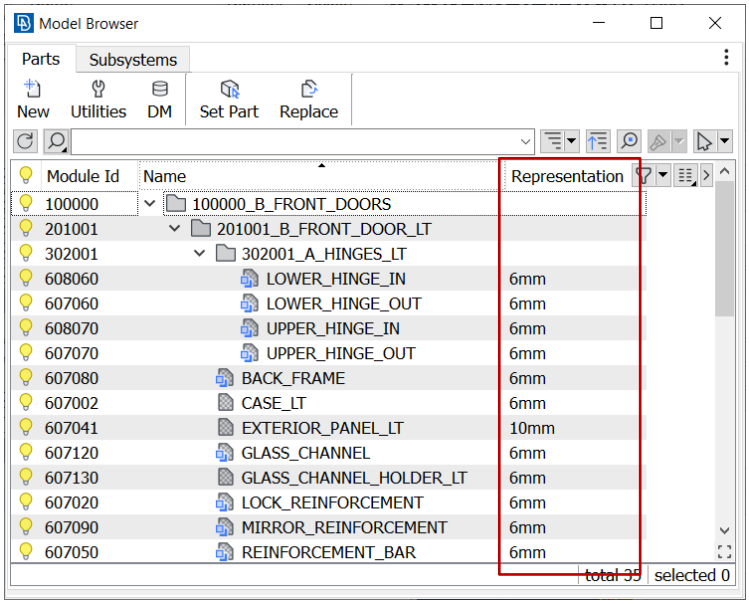
3.6.3. Viewer configuration

The default configuration of the viewer is implemented upon its launch by the embedding tool. When the embedding tool is the DM Browser in ANSA/META, a minimum set of configurations is hard-coded in ANSA/META. When the embedding tool is KOMVOS, the default configuration is applied by reading the `viewer.xml` settings file that lies in KOMVOS' application home folder. By default, this is the *config* folder of the KOMVOS installation directory. Alternatively, it can be the directory indicated by the **SDM_CONSOLE_HOME** environmental variable.

Any changes in the view settings applied by the user during the usage of the tool, are saved in the user's settings home folder (.BETA/META/…). The save is performed automatically upon quit in the case of KOMVOS. In the case of ANSA/META, the saving of the GUI settings is done explicitly by the user through the program's Settings.

3.7. Representations

The base DM Objects are stored in the SDM system with various representations. These representations are interchangeable modules that can be loaded from the DM to the pre-processor at any time and get integrated in the full assembly, deriving various discipline-specific models. With BETA's SDM solutions, storage of alternative representations is an easy task as Representation is, by default, a primary attribute for Parts and Subsystems, allowing a certain object to be stored in the database in different representation flavors. Additionally, Lifecycle Business Rules (see paragraph 2.3) can be defined to control the representation creation process (i.e. which representations can be derived from a given representation, etc.). However, when it comes to representation management the SDM system only offers the foundation, as the core functionality is handled by the pre-processor.



ANSA offers complete solutions for the management of representations on Part and Subsystem level.

In the Model Browser, the representation is kept in the **Representation** field. This field can be edited freely and can hold alphanumeric strings. ANSA supports a list of built-in representations that fall in two categories:

- Special representations, i.e. representations that signify some particular state of the container
- Derived representations, i.e. that have a built-in creation process.

The table below lists the reserved representations for Parts and Subsystems:

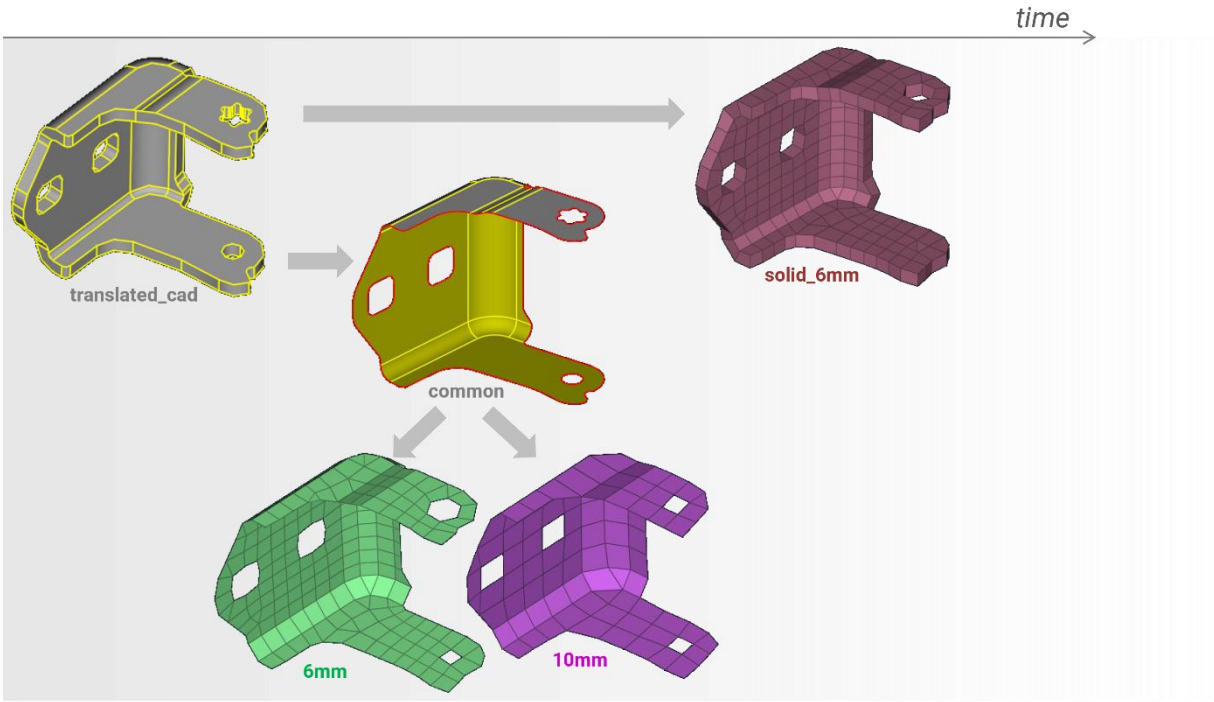
Parts	Subsystems
translated_cad	Lumped mass
common	
Trim	
Lumped mass	
SPC	
Don't Use	

3.7.1. Special built-in representations

The Part entity has two fundamental reserved representations, the *translated_cad* and the *common*. These are given automatically by the system and are used with specific handling in standard ANSA processes like the CAD files translation and the batch meshing.

As the Part evolves during the model build phase, several new representations of it may be stored in the data repository. If one would like to look at its typical lifecycle, a part is first stored in DM as a *translated_cad* representation, which is the direct result of conversion of the CAD file to an ANSA file. This representation name is automatically assigned to the parts generated by the automated **CAD to ANSA** process. (for more details on this topic please refer to the ANSA User's Guide paragraph "Reading product structure from PDM systems").

Based on this representation, the model building teams will generate and store its *common* representation. The *common* representation is the one used by the meshing tools as the basis for the creation of any discipline-specific mesh representation and as such, it must meet specific requirements dictated by the FE-modelling specifications.



3.7.2. Derived built-in representations

ANSA provides functionality for the direct conversion of detailed Part/Subsystem representations to reduced ones, with the minimum input requirements. To generate these representations the user should access the **DM>Change Representation** functionality within the *Model Browser*.

The table below lists all the derived built-in representations. For details about the steps to be followed for their creation, please refer to paragraph “Built-in reduced representations” in Chapter 30 of ANSA User’s Guide.

Representation name	Applicable on	Description
Lumped mass	Parts/ Subsystems	Replace a part, group or subsystem by a lumped mass, i.e. a point mass connected with the rest of the model with either a rigid or an interpolation element. Mass and inertia properties of the point mass, as well as its position are calculated automatically by the characteristics and CoG of the substituted entities.
Trim	Parts	Replace a part or group by equivalent mass, distributed on the nearby parts .
Don’t use	Parts	Exclude selected parts or groups from the model.
SPC	Parts	Exclude selected parts or groups and enforce a zero-displacement constraint to the locations where they were attached to the rest of the model.

3.8. Exploring data relations

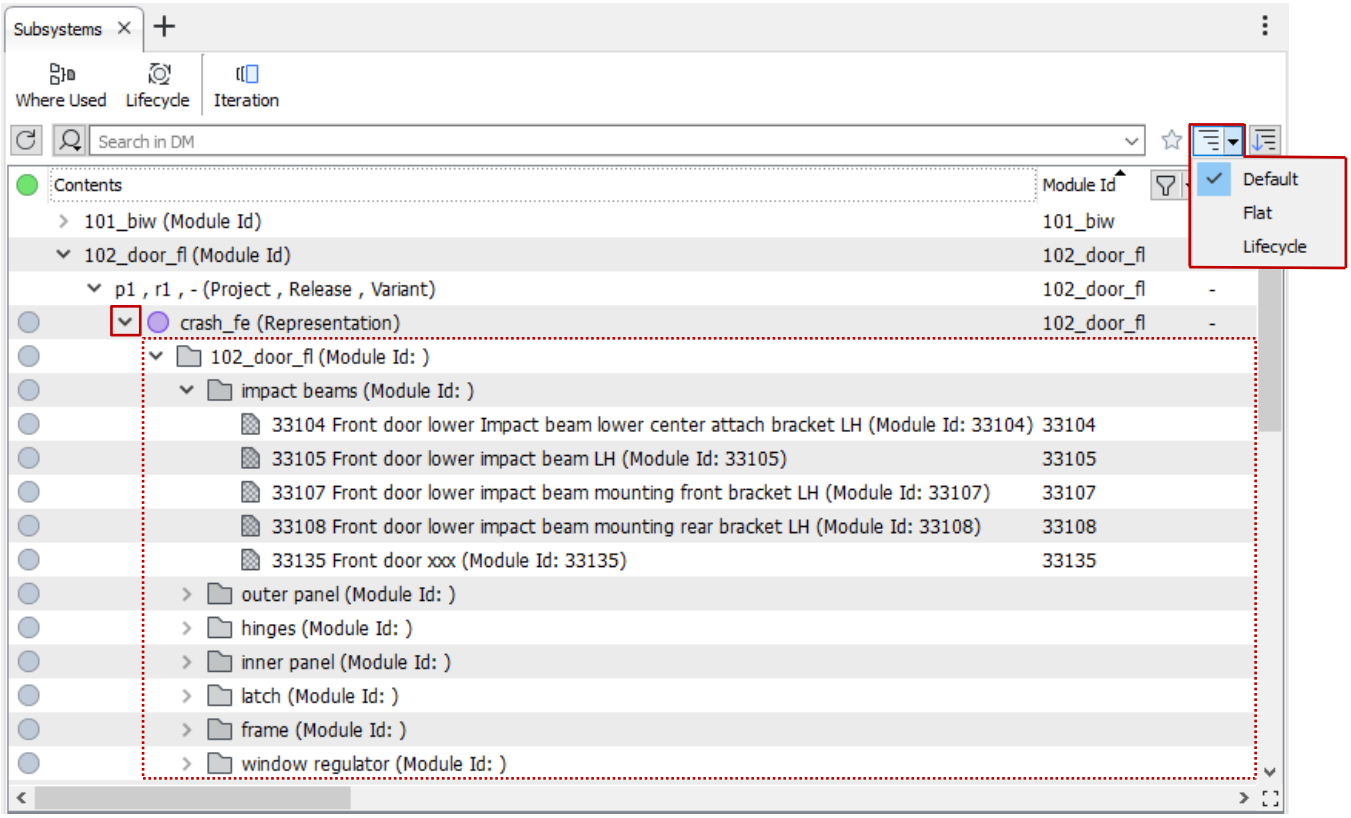
Data traceability is one of the key features of BETA's SDM solutions, which is facilitated with the use of links (i.e. references between data objects) that capture the relations between the various data objects stored in the database. The SDM client offers tools that allow the user to explore data relations in different ways:

- The contents and hierarchy of a data object are displayed in dedicated views in the database browser.
- The references of a data object are listed in the **References** info tab and are also presented graphically in the **References Graphs**.

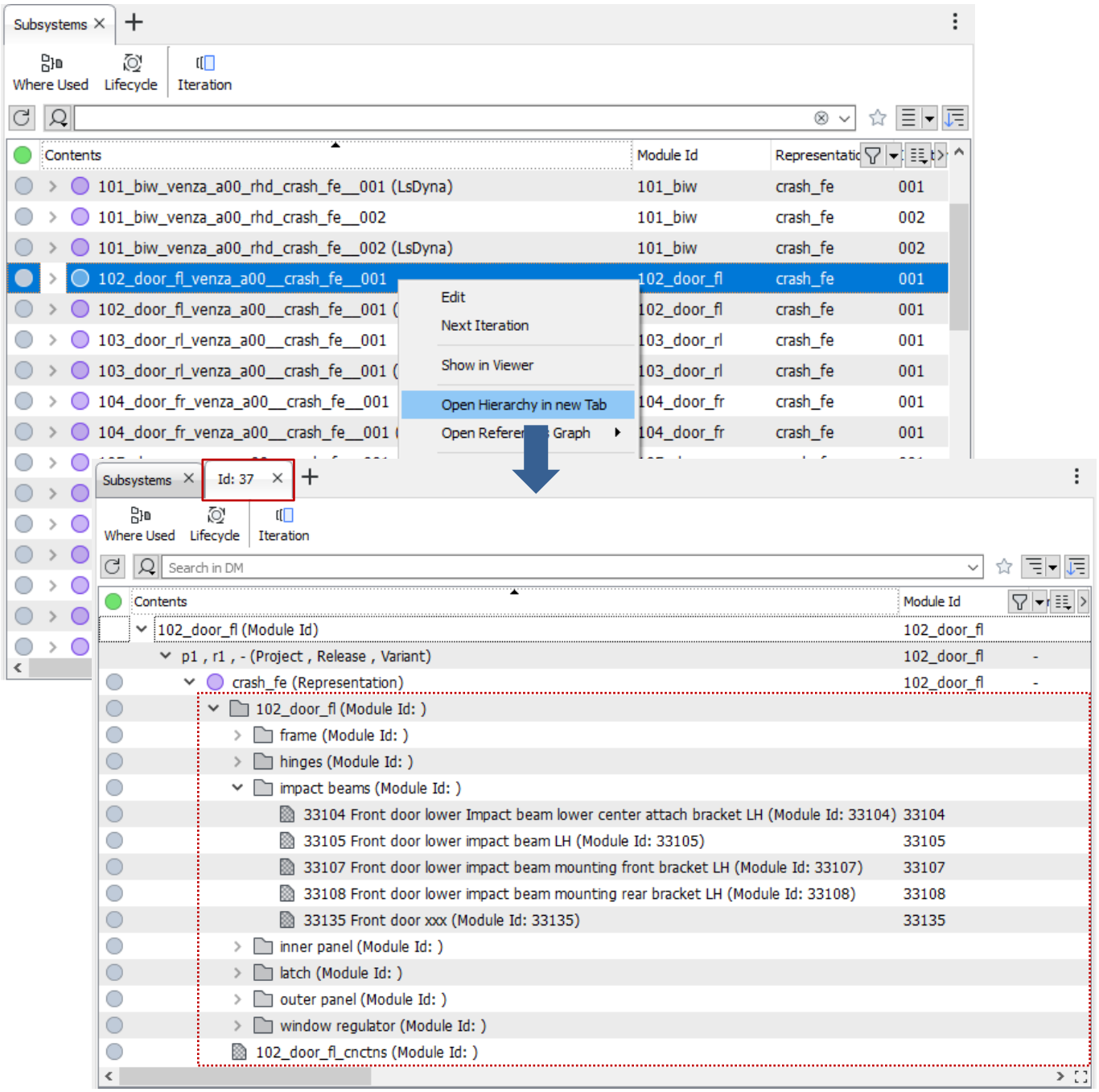
3.8.1. Contents and hierarchy

The database browser of the SDM client can display the Parts hierarchy of Subsystems and the contents of compound Model Containers (i.e. Simulation Models, Loadcases and Simulation Runs) in the views labeled *Default* and *Hierarchy* respectively. Switching the view of Subsystems into *Default* or the view of any other object type into *Hierarchy*, expanding the list view item of the respective object will list its contents in place.

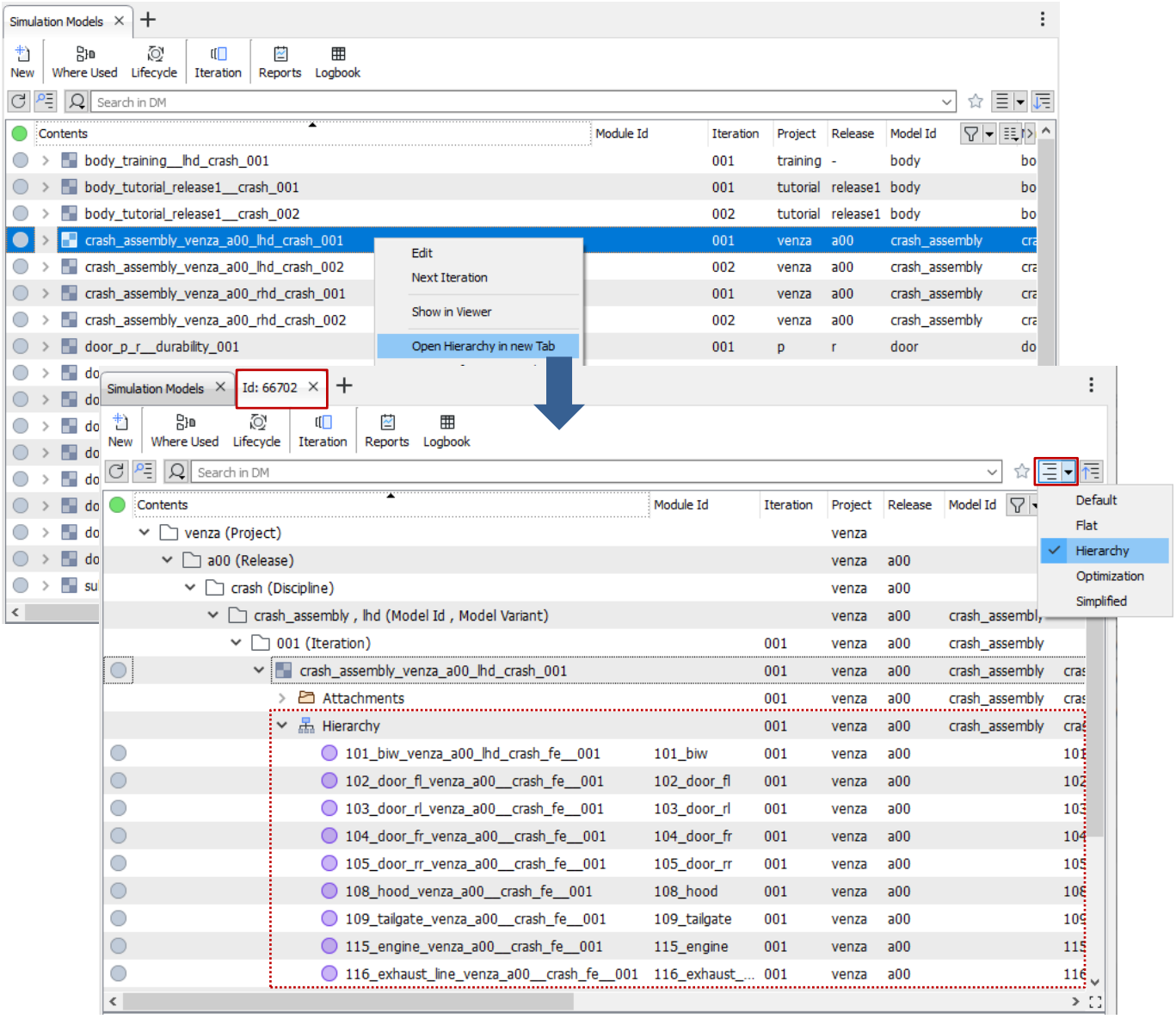
The parts hierarchy can be accessed by expanding a Subsystem container in the *Subsystems* tab, in which case the structure of the Parts/Groups contained in the selected Subsystem will be listed in place.



For those cases where the parts hierarchy is not available in the current view (e.g. while reviewing Subsystems in the *Flat* view), the user may select the context menu option **Open Hierarchy in new Tab**, in order to open the part structure of the selected Subsystems in a new tab. In this case, the new tab contains only the selected Subsystems and uses the view that shows the parts hierarchy, i.e. the *Default* view for Subsystems.



Similar to the Parts structure of Subsystems, the contents of compound model containers can be also viewed in the SDM client. Selecting for example a Simulation Model in the *Simulation Models* tab and using the same context menu option **Open Hierarchy in new Tab**, the contents of the Simulation Model will be displayed in a new database browser tab using the *Hierarchy* view.



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

3.8.2. References

All data relations can be explored through the **References** tab of the database browser. Selecting any model container in the main list of the database browser, all DM objects related to the selected item are listed in three different sub-tabs, depending on the reference type:

- **Where Used**: these hold the relations between data objects and their contents and are created automatically by the system.
- **Lifecycle**: these hold relations between data objects during the course of their lifecycle and are created automatically by the system.

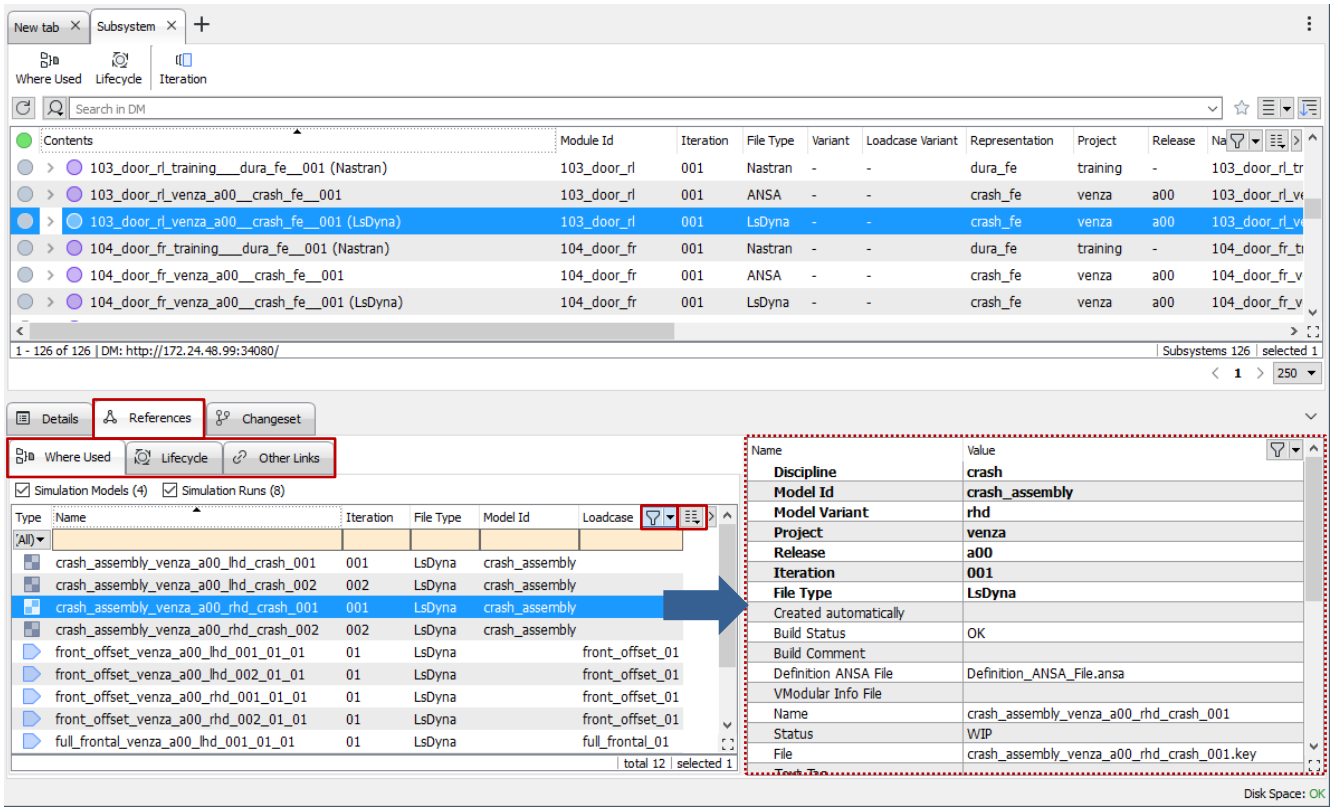
- **Other Links**: these hold various types of relations that are created automatically by the system and do not fall in any of the above link types. In addition, links created by the user using BETA's scripting API are also categorized as *Other Links*.

Except for the "Where Used" reference links, all other references have a specific link type. The table below offers an overview of all supported types of links, the label by which thay are displayed in the data browser and a brief description for each.

Reference Type: Lifecycle		
Link Type	**Displayed as**	**Description**
history	previous/next version	Generic link between a pair of two same type DM objects that represent the previous and the next version of an item (i.e. only version attributes will differ between the two DM Objects). It is created automatically when saving through ANSA or importing through the SDM client, and only when a conflict is detected and it is auto-resolved with version spin-up. It is considered a weak link.
repr_derivation	ansa/solver representation	ANSA specific link between an ANSA File Type subsystem and the solver subsystem that was created from it. It is created automatically when saving through ANSA and is considered a weak link.
variant	multi-variant(150%)/ single-variant(100%)	ANSA specific link between a multi-variant subsystem containing all configurations, and one or more subsystems saved while one configuration was active. It is created automatically when saving through ANSA and is considered a weak link.
adaptation	adaptation/adapter	ANSA specific links between an adapted and the un-adapted Subsystem, or a Loadcase and the un-adapted Loadcase Template. It is created automatically during when saving through ANSA. It is a strong link, thus the un-adapted items cannot be deleted.
changeset	changeset	ANSA specific links between a changeset object and an ANSA Subsystem or Part, which has been saved as the descendant of another. The changeset object holds a summary of the changes that took place while creating the new version of a subsystem/part. It is created automatically when saving through ANSA and is considered a weak link.
creation	creation/creator	Generic links between a Report and the Session that was used to generate it. It is created automatically during the creation of the Report with the execution of the Session. It is a strong link, thus the Session cannot be deleted.
Reference Type: Other Links		
Link Type	**Displayed as**	**Description**
strong	strong reference/ referee	Generic link between two DM Objects of the same type, created through the Lifecycle Business Rules or using scripting (DMObject class and method connect).
weak	weak reference/ referree	Generic link between two DM Objects of same or different type, created using scripting (DMObject class and method connect).
modular_ environment_ profile	created using profile/ profile's product	ANSA specific link between a DM Object and the Modular Environment Profile used when it was saved. It is created automatically when saving through ANSA. It is a strong link, thus the MEP cannot be deleted.
compatible connectivity	connecting (compatible)	ANSA specific link between a solver File Type Connecting Subsystem and one or more Regular Subsystems that is compatible with (i.e. can - be used "as is" with these Subsystems, with no need to re-apply its Connections). It is created automatically when saving the Connecting Subsystem through ANSA and it is a weak link.

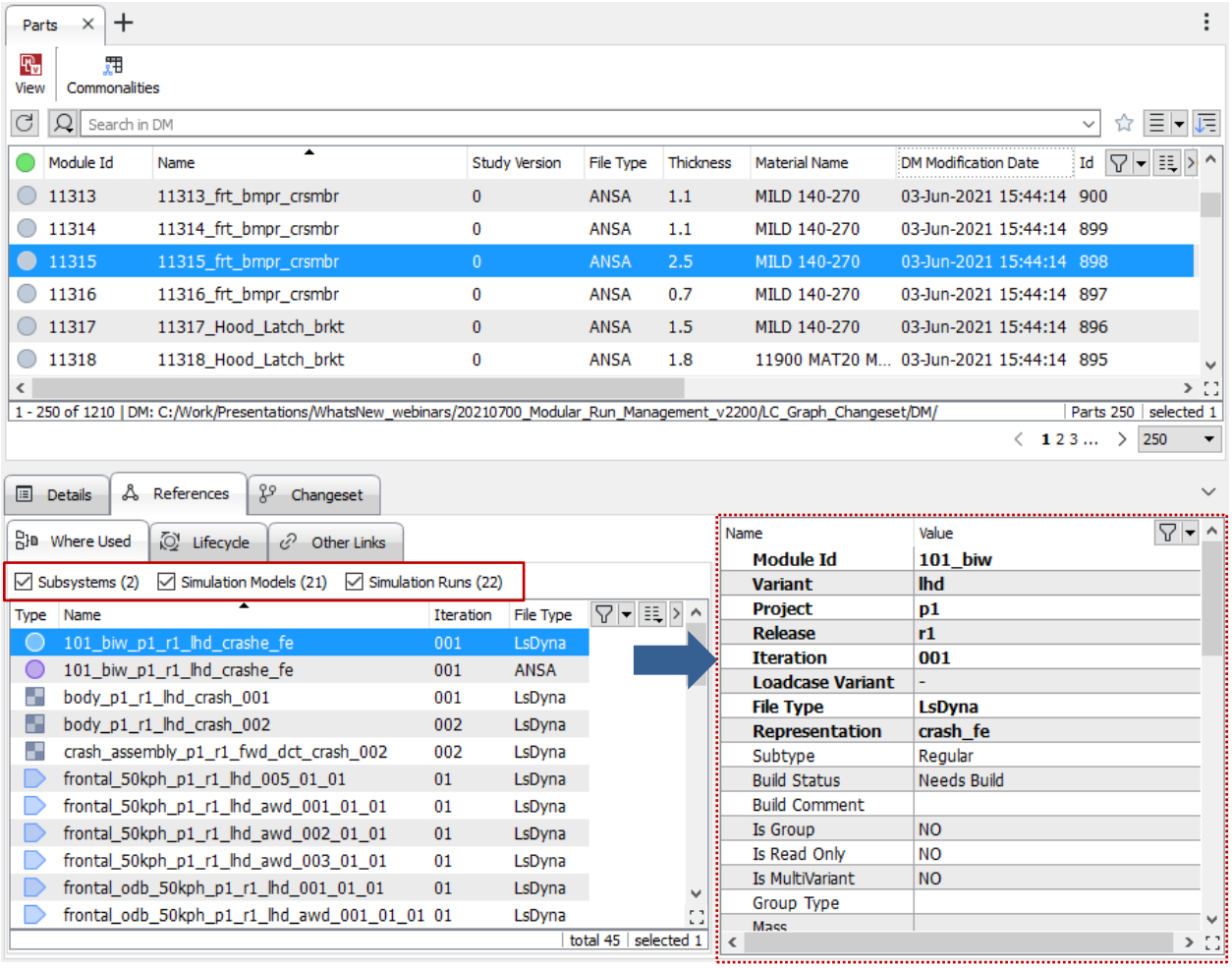
incompatible connectivity	connecting (incompatible)	ANSA specific link between a solver File Type Connecting Subsystem and one or more Regular Subsystems that it is not compatible with (i.e. cannot be used "as is" with these Subsystems). It is created automatically when saving the Connecting Subsystem through ANSA and it is a weak link.
multiple fe representation	alternative representation/ nominal	ANSA specific link between two or more Simulation Models that make use of alternative FE-representations of the same Subsystems. It is created automatically when saving through ANSA and it is a weak link.
method	optimization method/ optimization study	ANSA specific link between an Optimization Study and one or more Predictors that were used to create it. It is created automatically during “Optimization Tool>New Optimization Study” and it is a weak link.
optimization	optimization run/ optimization study	ANSA specific link between one or more Simulation Runs and the Optimization Study that created them. It is created automatically during “Optimization Tool>Start Optimization” and it is a weak link.
training	training input/ training product	Generic link between one or more Simulation Runs that were created by one Predictor. It is created through scripting with the DMObject class and the method connect. It is a weak link.

In each list of references in the **References** tab, the user can add more attributes of the listed items as columns, in order to have direct access to the desired information, as well as filter the listed items. Moreover, selecting any of the listed references, its detailed attributes are displayed on the right.

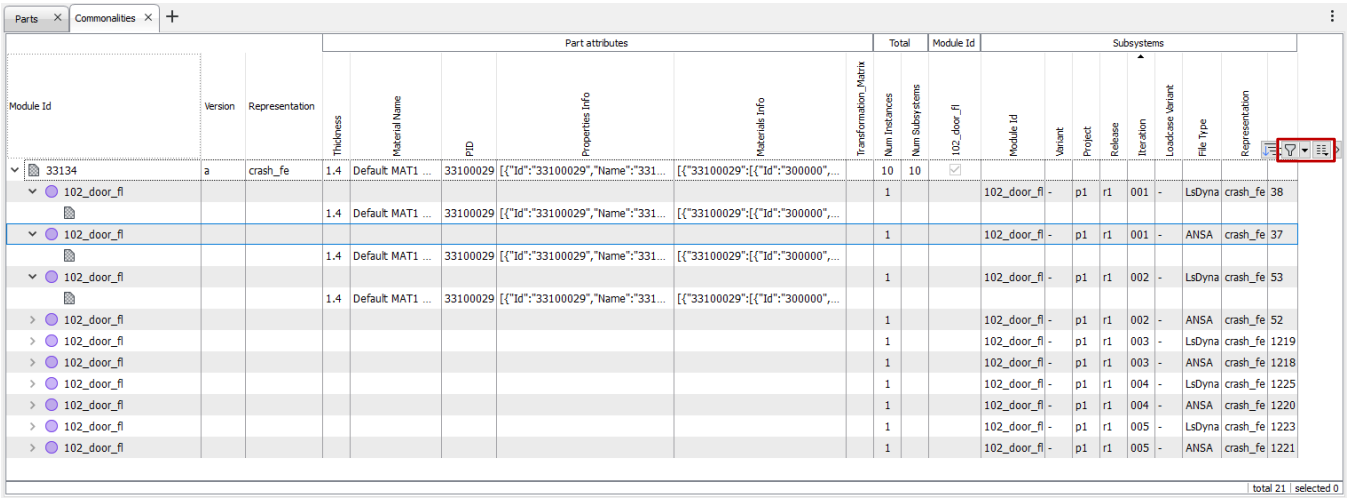


3.8.2.1. ‘Where Used’ relations

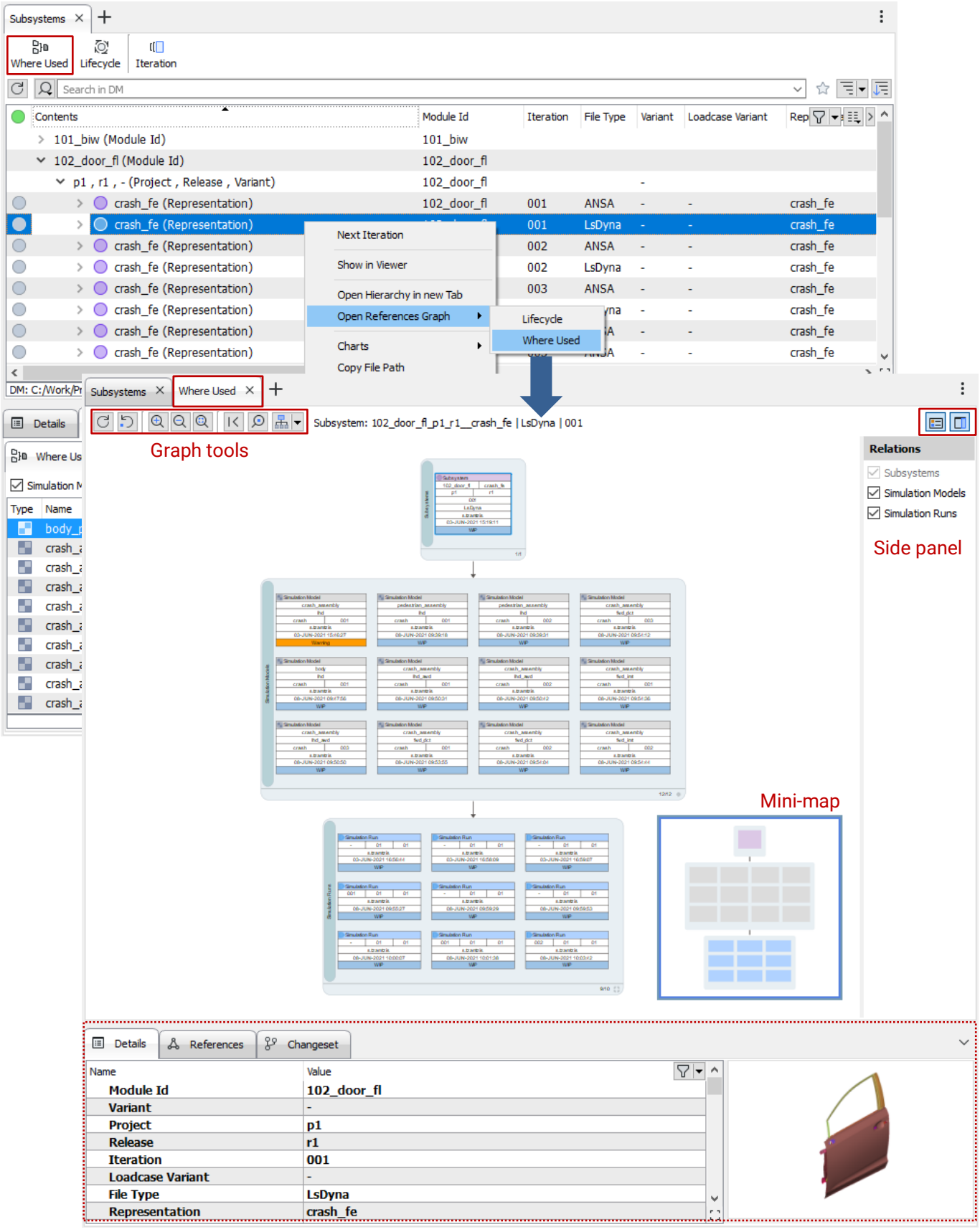
The **Where Used** tab lists the data objects that make use of the item selected in the main tab of the database browser. The relations are listed recursively bottom-up; for example, selecting a Part in the database browser, the **Where Used** tab will list the Subsystems that contain the Part, the Simulation Models that contain those Subsystems and the Simulation Runs that contain the respective Simulation Models. The user can control which types of data objects will be listed, through the respective check boxes.



Specifically for Parts, an alternative way to identify the Subsystems where a Part is used in, is the **Commonalities** function, that can be accessed either from the toolbar of the **Parts** tab in **KOMVOS** or using the respective context menu option. A new tab opens in the database browser, listing all the different Subsystems where the selected Part is used in. The table in the **Commonalities** tab displays some key attributes of the Part, as well as the primary attributes of the Subsystems that contain the Part. The user can control the attributes that are displayed as columns in the table and filter the listed items .

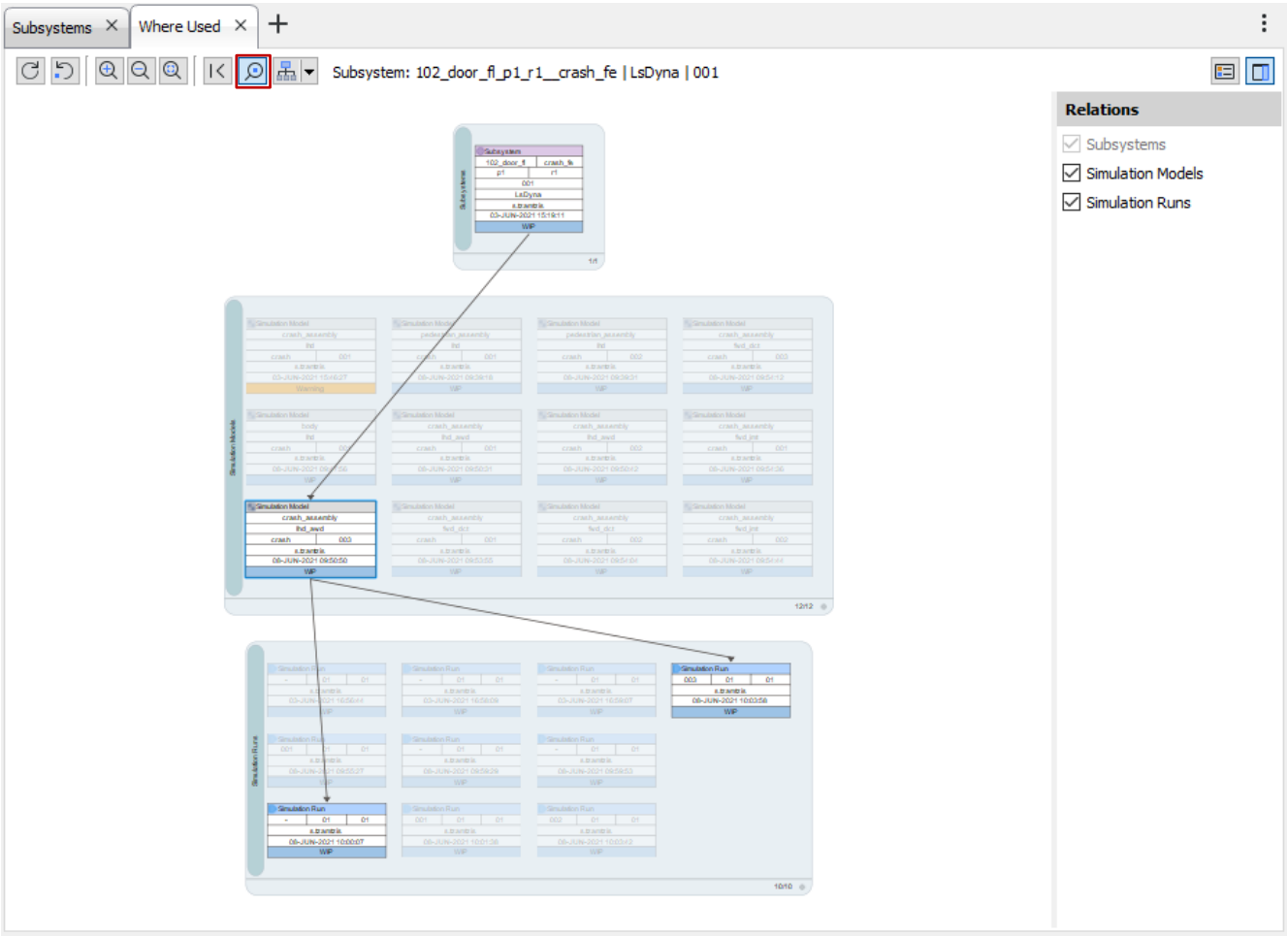


A graphical representation is also offered by BETA's SDM clients for the display of "*Where Used*" relations. This can be accessed by selecting any model container in the database browser and using the **Where Used** button in the toolbar or the **Open References Graph > Where Used** context menu option.



A new tab opens in the database browser with a graphical representation of the *"Where Used"* relations of the selected DM object, grouped per DM object type. Each node in the graph displays metadata for a different DM object. When working with a server-based SPDRM backend, the content of the nodes' tables can be customized to display the desired metadata. Please refer to paragraph 5.2.2 of this document for more information.

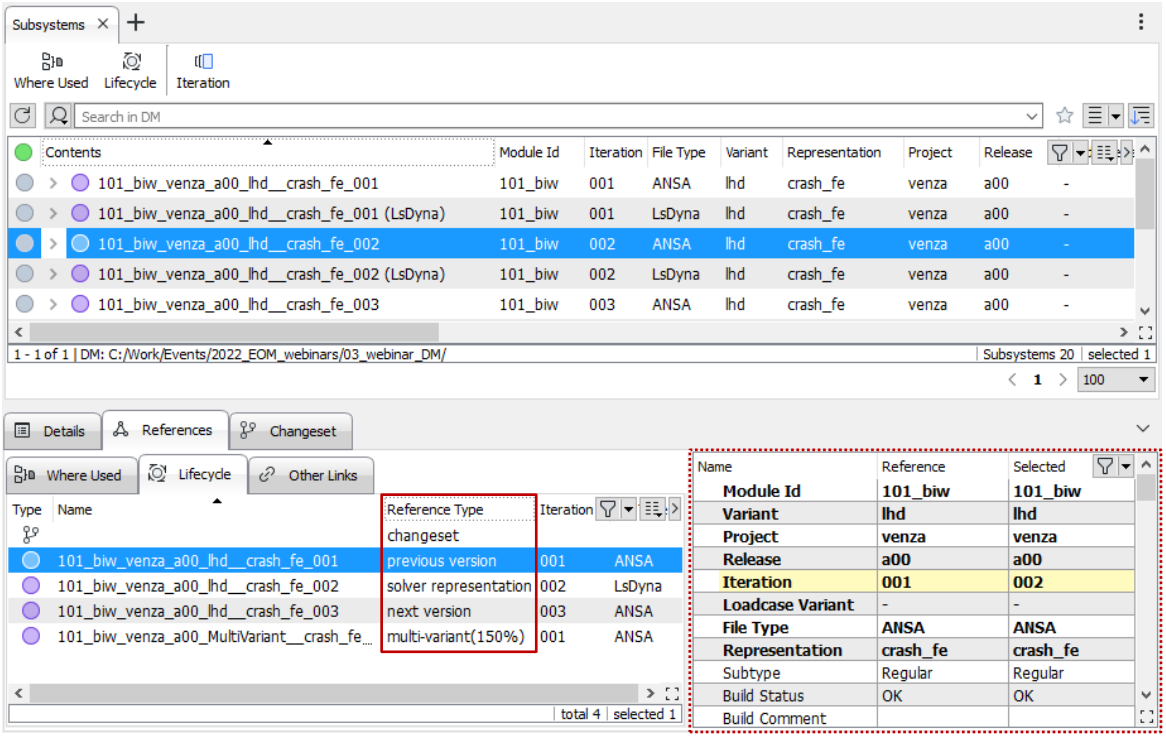
The user has access to various graph tools, a side-panel to control the visibility of the displayed relations and a mini-map for enhanced navigation in the graph. The visibility of the mini-map and the side panel can be toggled using the respective buttons . Selecting any of the DM objects displayed in the graph, the user can access their metadata from the bottom tabs (e.g. Details, References, etc.). Using the **Isolate Relations** button in the graph tools, the user can isolate the relations between selected DM objects in the graph. For example, the relations between a Subsystem, a Simulation Model and the Simulation Runs that contain them can be isolated, by selecting in the Subsystem's graph the respective Simulation Model.



3.8.2.2. Lifecycle relations

The **Lifecycle** tab lists any data objects that are related with the item selected in the main tab of the database browser with links captured during the course of the object's lifecycle. These may include previous or next versions, alternative representations or variants, links with changeset objects, etc., as described in detail in the respective table in paragraph 3.8.2.

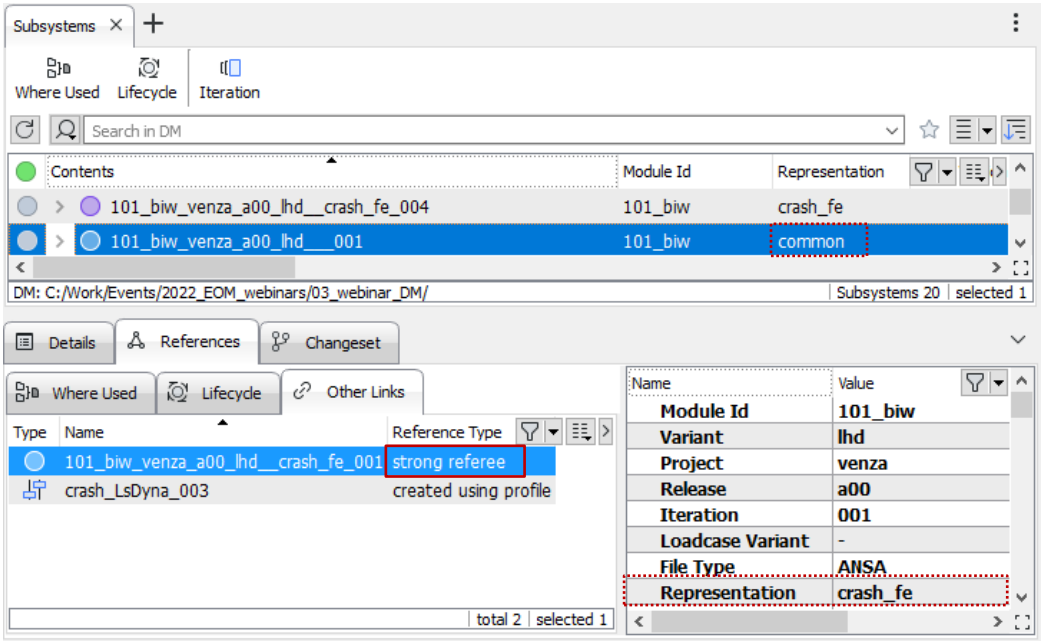
Selecting a Subsystem in the database browser, the user can access its existing lifecycle relations in the **Lifecycle** tab. The *Reference Type* column displays the type of lifecycle link between the selected Subsystem and its references. Selecting any of the listed references, its metadata are listed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



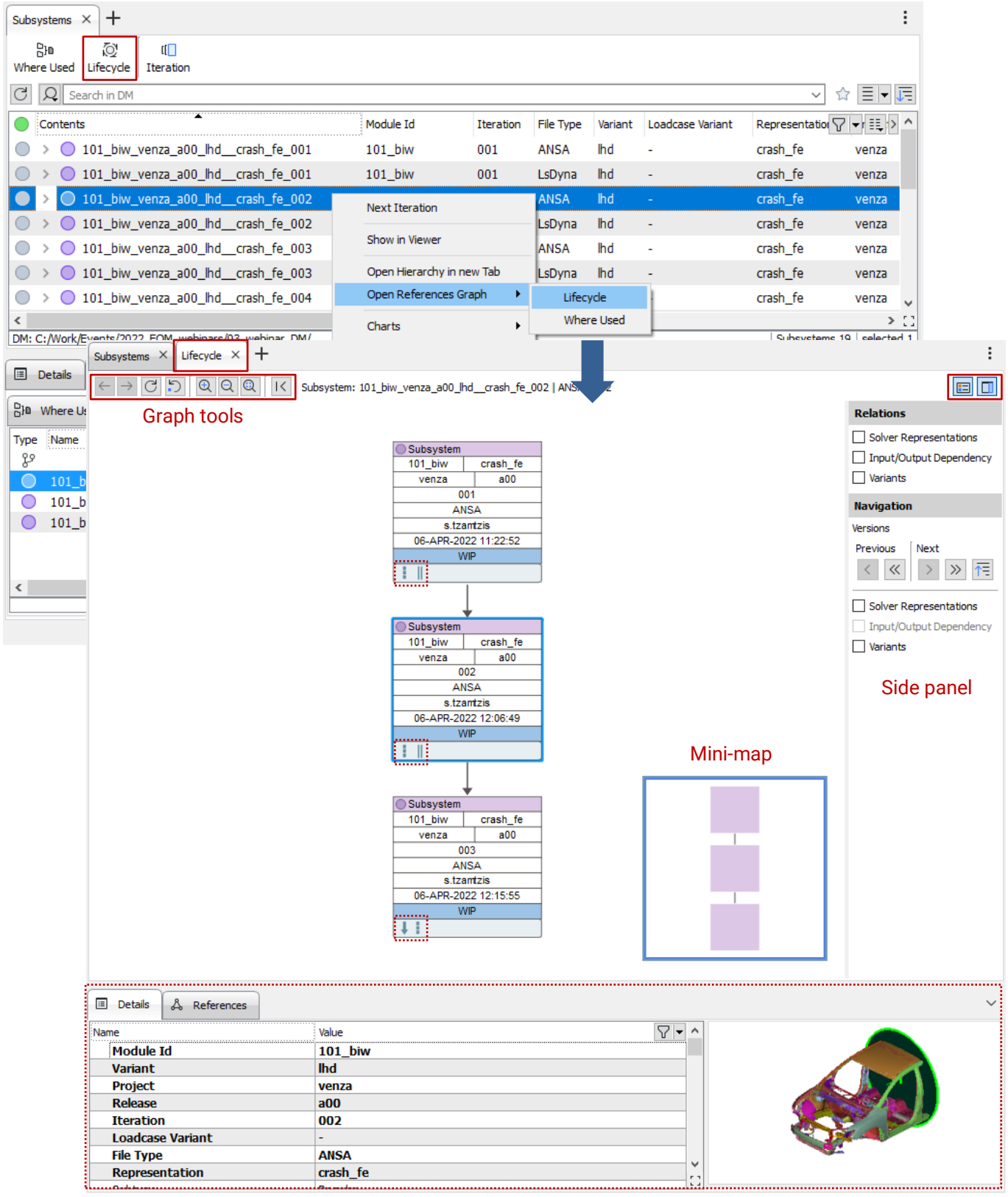
In addition to the lifecycle references supported by the system out-of-the-box, BETA's SDM solutions support the creation of references that capture the evolution of a data object in two other ways:

- Through the setup of Lifecycle Business Rules in the SDM environment. Once configured, it is possible for the system to automatically generate links between same type DM objects that represent different stages in the lifecycle of a data object (e.g. a reference between the *common* and the *crash* representation of a Subsystem). For a detailed description, please refer to paragraph 2.3.
- Using BETA's python scripting API.

Lifecycle references created in any of the above ways have *Reference Type=strong referrer/referee* when created between DM objects of the same type, and *Reference Type=weak referrer/referee* when created between DM objects of different type. In the SDM client, these references are listed in the **Other Links** tab of the database browser.



A graphical representation is also offered by BETA's SDM clients for the display of lifecycle relations. This can be accessed by selecting any model container in the database browser and using the **Lifecycle** button in the toolbar or the **Open References Graph > Lifecycle** context menu option.



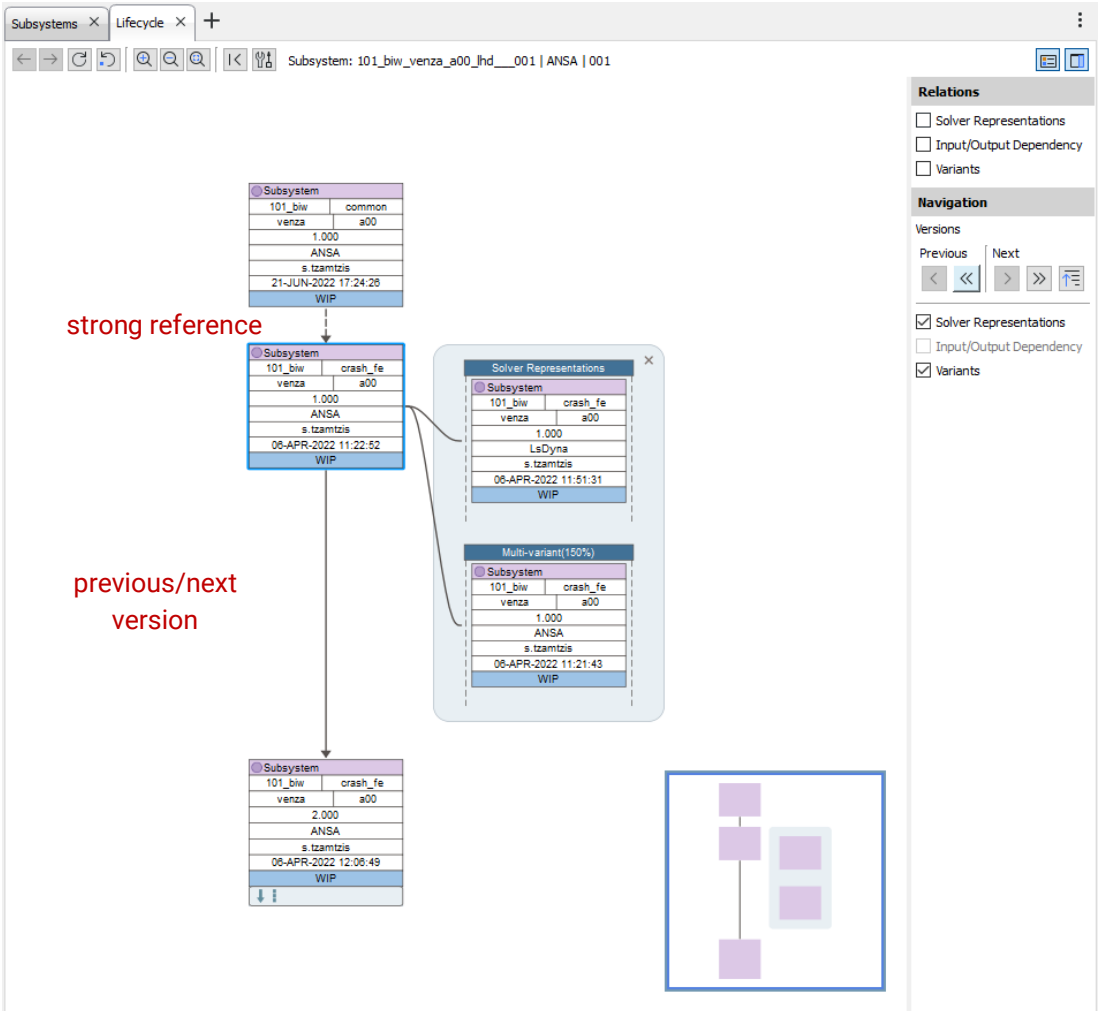
A new tab opens in the database browser with a graphical representation of the lifecycle relations of the selected DM object. The user has access to various graph tools, a side-panel to control the visibility of the displayed relations and to navigate through the lifecycle relations, and a mini-map for enhanced navigation in the graph. The visibility of the mini-map and the side panel can be toggled using the respective buttons . Selecting any of the DM objects displayed in the graph, the user can access their metadata from the bottom tabs (e.g. Details, References, etc.).

The side panel of the *Lifecycle* graph contains two sections:

- **Relations**: this section allows the user to control the visibility of lifecycle relations per type, using the respective checkboxes. The actions applied in this section affect the visibility of all relations listed in the graph as a whole.
- **Navigation**: this section appears only when the user selects a DM object in the graph. It offers navigation through the previous/next versions of the selected DM object (i.e. any previous/next versions that exist but are not yet listed in the graph will be displayed), as well as visibility control per type for the lifecycle relations. The actions applied in this section affect the visibility of the lifecycle relations of the selected DM object only and not the graph as a whole.

The visibility of lifecycle relations in the graph can also be controlled through the characteristic buttons on the graph nodes:

- Previous or Next versions of a given DM object can be displayed using the and buttons respectively.
- Solver representations of an ANSA Subsystem can be displayed using the button.
- Variant relations of an ANSA Subsystem can be displayed using the button.



The *Lifecycle* graph displays previous/next versions and strong references of a DM object in the nodes of the main branch of the lifecycle pedigree tree. They can be distinguished through the connecting arrows, which are dashed for strong references and continuous for previous/next versions. Other lifecycle relations such as solver representations, variant relations input/output dependencies (i.e. weak references) are displayed in auxiliary nodes for each DM object.

3.8.3. Data Alerts

One of the key features of SPDRM, BETA's server-based SDM solution, is that it offers an alert mechanism, which, in case of errors, can raise alerts for the affected objects and propagate them to all their dependencies, offering the means to complete error impact analysis.

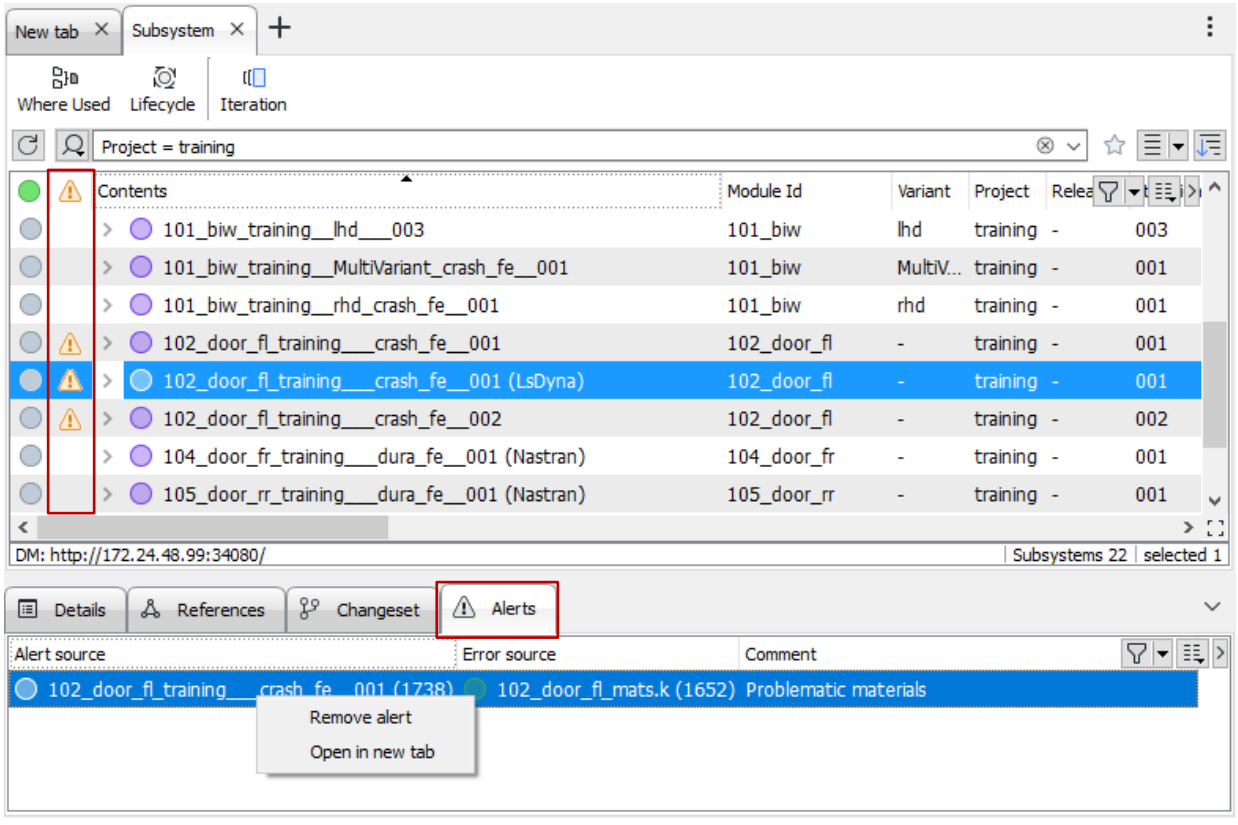
The generation of alerts is supported through two different mechanisms:

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

Detailed descriptions of the SPDRM alert mechanism and the Issue Tracking module can be found in the SPDRM administration guide and the KOMVOS User's Guide respectively.

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

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



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

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

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

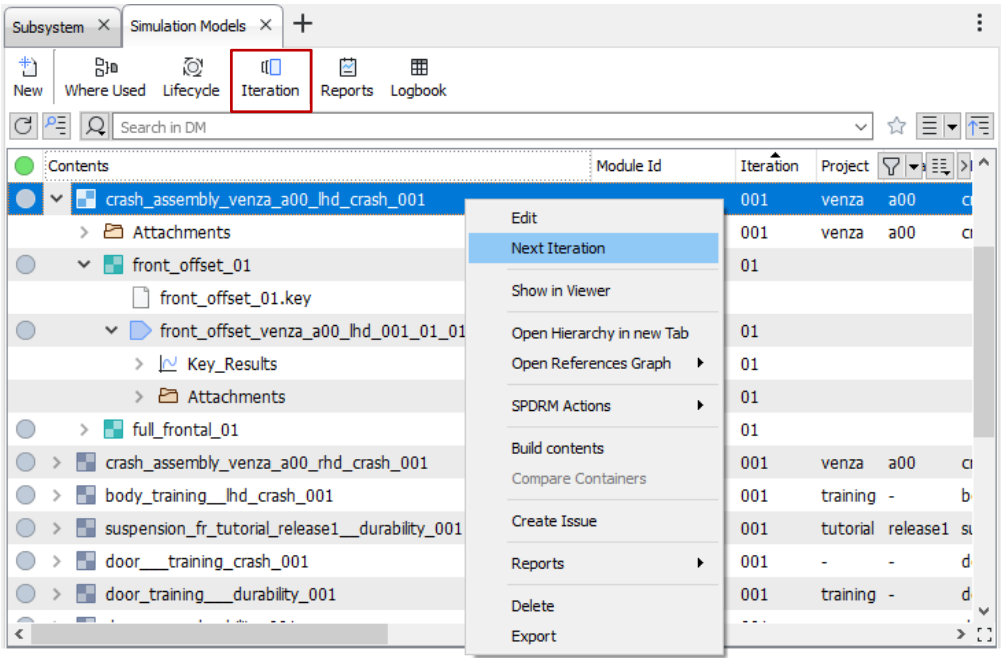
3.9. Creating new iterations

In BETA's Modular Environment, the creation of new simulation loops in order to improve the behavior of the model is referred to with the term **Next Iteration**. A detailed description of the recommended workflow for the creation of new simulation iterations is given in Chapter 8 of the Modular Model and Run Management User's Guide.

The following sections will provide an overview of how to access the core functionality that facilitates the creation and management of new simulation iterations.

3.9.1. Next iteration

The **Next Iteration** action is the starting point for the creation of new simulation iterations. It can be accessed either from the respective context menu of a DM object or from the **Iteration** button in the database browser toolbar.



Depending on the selected DM object, the **Next Iteration** will perform the following actions:

- For Subsystems, the ANSA representation file will be loaded in a new ANSA session [2]. If the selected Subsystem is a solver file, the ANSA representation file is traced through the lifecycle references. So although the selected subsystem has a solver file type, the process will load in ANSA the definition file of that Subsystem, which has File Type = ANSA [3].
- For compound model containers (i.e. Simulation Models, Loadcases and Simulation Runs), the ANSA definition file will be loaded in a new ANSA session. A detailed description of the ANSA definition file is given in Chapter 3 of the Modular Model and Run Management User's Guide.

If the next iteration process is performed on Subsystem level, the downloaded Subsystem will be modified in the pre-processor, it's *Build* process will be executed to verify that it is ready to be saved in DM and eventually the Subsystem will be saved in DM as a new Iteration.

On the other hand, if the next iteration process is performed on a compound model container, updates to its contents need to be identified in the DM tab of the Model Browser. Once the compound container is updated with the right contents, it's *Build* process will be executed to verify that it is ready to be saved in DM and eventually the compound container will be saved in DM as a new Iteration.

3.9.2. Changesets

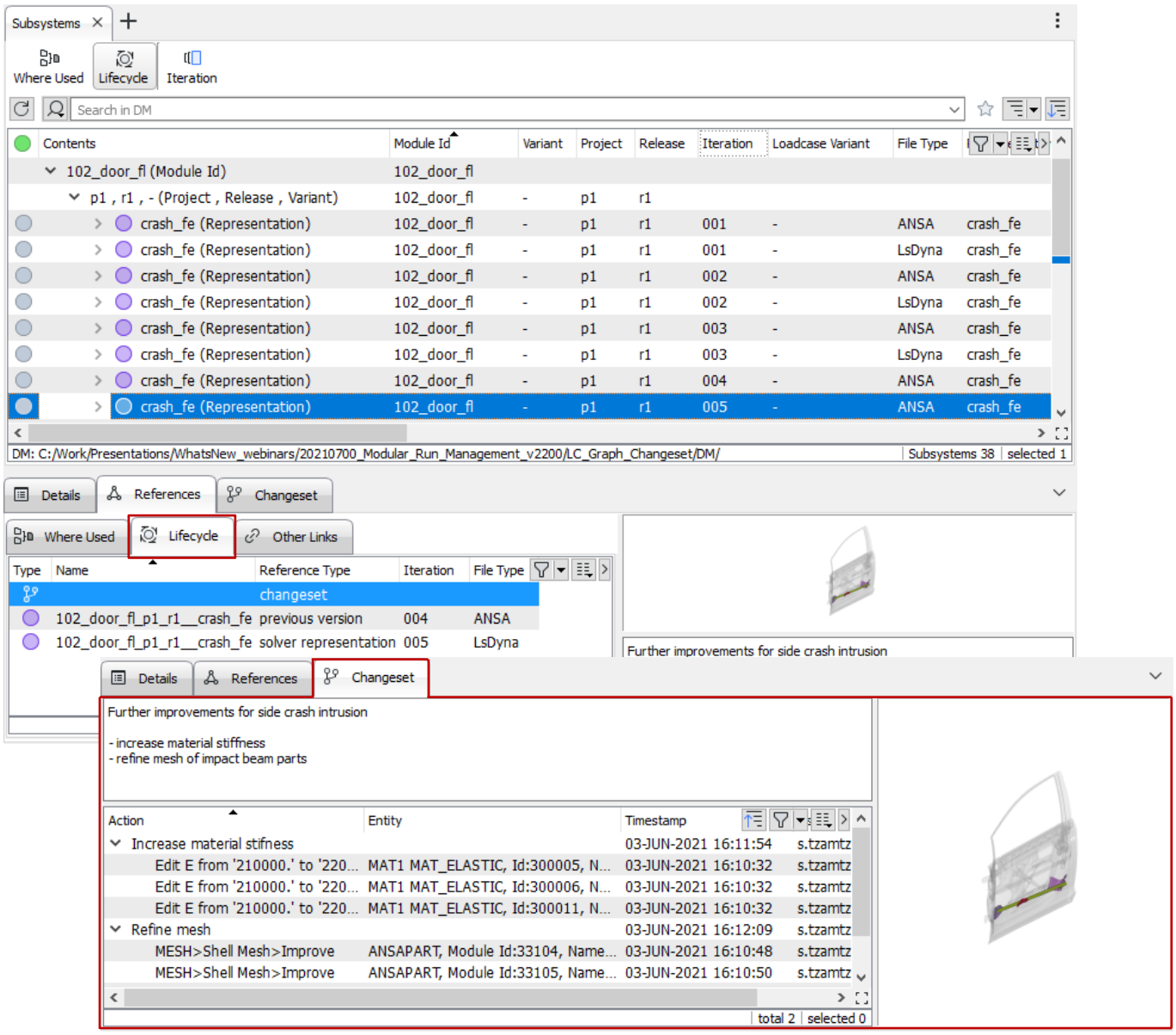
BETA's Modular Environment offers Changeset management functionality that can record the modifications that take place on Part and Subsystem level and report them as a changeset, which represents the delta between two successive versions of a Part or Subsystem. A detailed description is available in Chapter 3 of the Modular Model and Run Management User's Guide.

For any Part or Subsystem that is downloaded from DM, modified in ANSA and then saved back to DM as a new version, its changeset is saved along as a separate DM object. In the SDM client, changesets are only shown as metadata of the Part or Subsystem DM Objects they relate to, either in the **Lifecycle** tab or in the dedicated **Chanageset** tab. These can be accessed either from the database browser lists or from the Lifecycle graphs of DM objects.

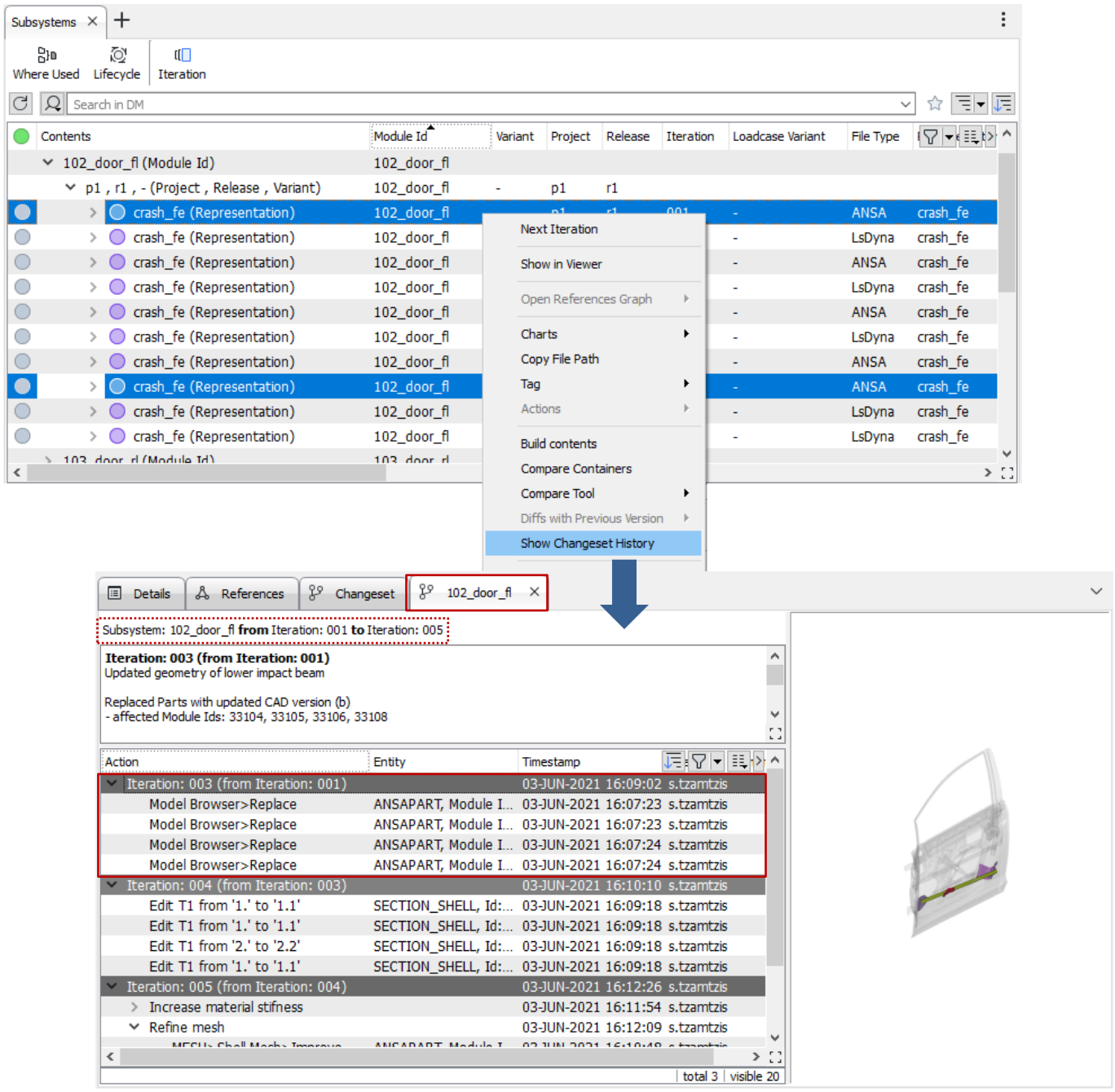
The changeset consists of a screenshot that highlights the modified areas and a *Summary* to describe the engineering intention behind the modifications. A list of all the actions that took place since the opening of the file from DM are listed in the *Changeset* tab. The *Action* column displays detailed information for the recorded actions. Information related to the modified entity (e.g. ANSA keyword, Entity Id, Part Module Id, Name) is displayed in the *Entity* column. Moreover, the changeset contains information related to when the modification took place (*Timestamp*), by whom (*User*) and an optional user *Comment*.

[2] The **Next Iteration** action will always discard the existing ANSA session and download the ANSA definition file in a new session.

[3] If for some reason the solver file type Subsystem does not have a link to an ANSA subsystem, a Definition File Error will be shown and the Next Iteration process will be halted.



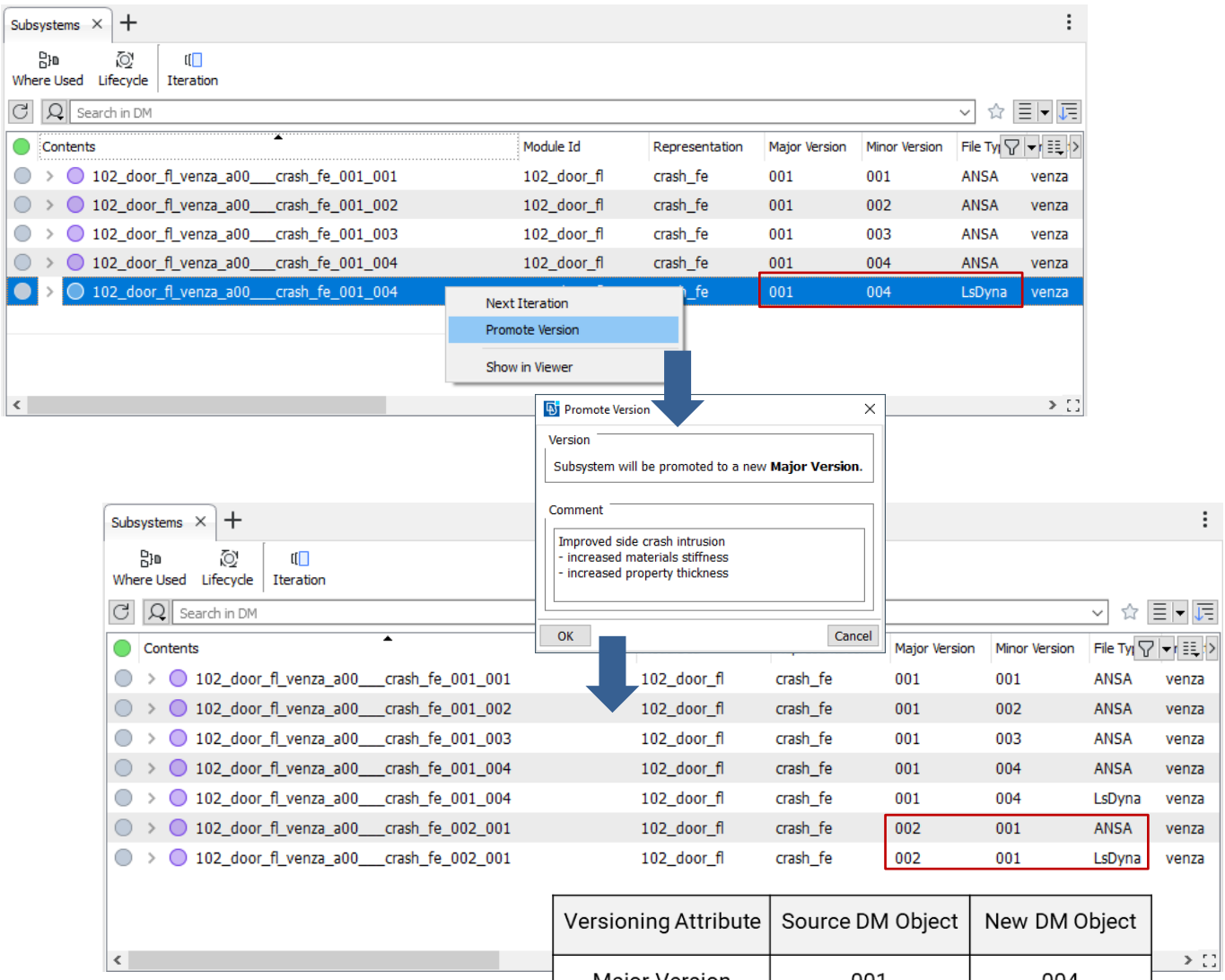
The complete changeset history between two versions of a Part or Subsystem saved in DM can also be viewed, selecting the desired versions and using the context menu option **Show Changeset history**. A new bottom tab in the SDM client, labeled after the Module Id of the selected entity, will display all the changesets between the selected versions. A label at the top of the tab will display information for the selected versions; the *Summary* field will display all changeset summaries clearly separated, while the *Actions* are grouped per changeset object.



3.9.3. Promote

A common need when working within an SDM environment that supports multiple versioning attributes (e.g. Minor Version and Major Version), is to be able and “promote” a minor version to a new major version to be used as the basis for new simulations, once the analysis reaches a desired level of maturity. To respond to this need, BETA’s SDM solutions offer a context menu option labelled **Promote Version**, which is applicable to base modules (Subsystems or Library Items) that have more than one versioning attributes in their DM properties.

By selecting in the database browser a DM object whose minor version is greater than the initial minor version value, **Promote Version** will trigger the creation of a new DM object that will contain the same file, but with the next available major version and initialized minor version.



Versioning Attribute	Source DM Object	New DM Object
Major Version	001	004
Minor Version	002	001

During this process, all metadata will be updated as required (e.g. the versions referenced in the DM header, in case of a solver keyword file or the versions in the ANSA file) and a proper lifecycle history link will be generated between the source and the new DM object.

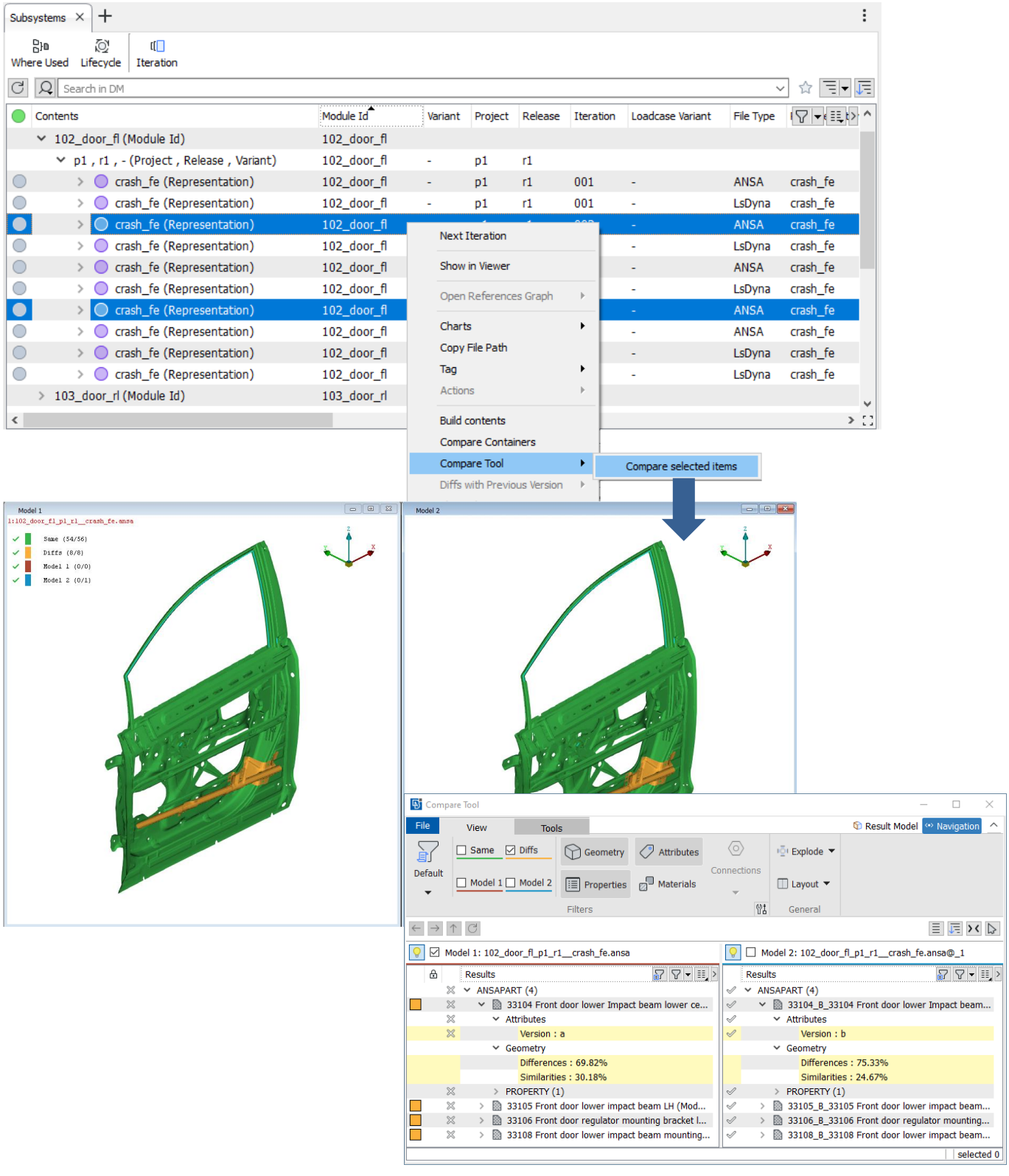
Finally, in case a solver file of a Subsystem is selected that also had a lifecycle link with an ANSA file, both file formats will be promoted to the new version, to maintain the data traceability.

3.10. Comparing data

Working within an SDM environment, where a large number of model variants and their contents are stored, including several simulation iterations, it is essential to be able and compare various DM objects in order to understand their differences and the impact that they may have in the simulation activity. BETA's SDM solutions offer powerful functionality that enable users to compare both base modules and compound model containers, either in terms of their geometry and engineering attributes, or in terms of their metadata and contents.

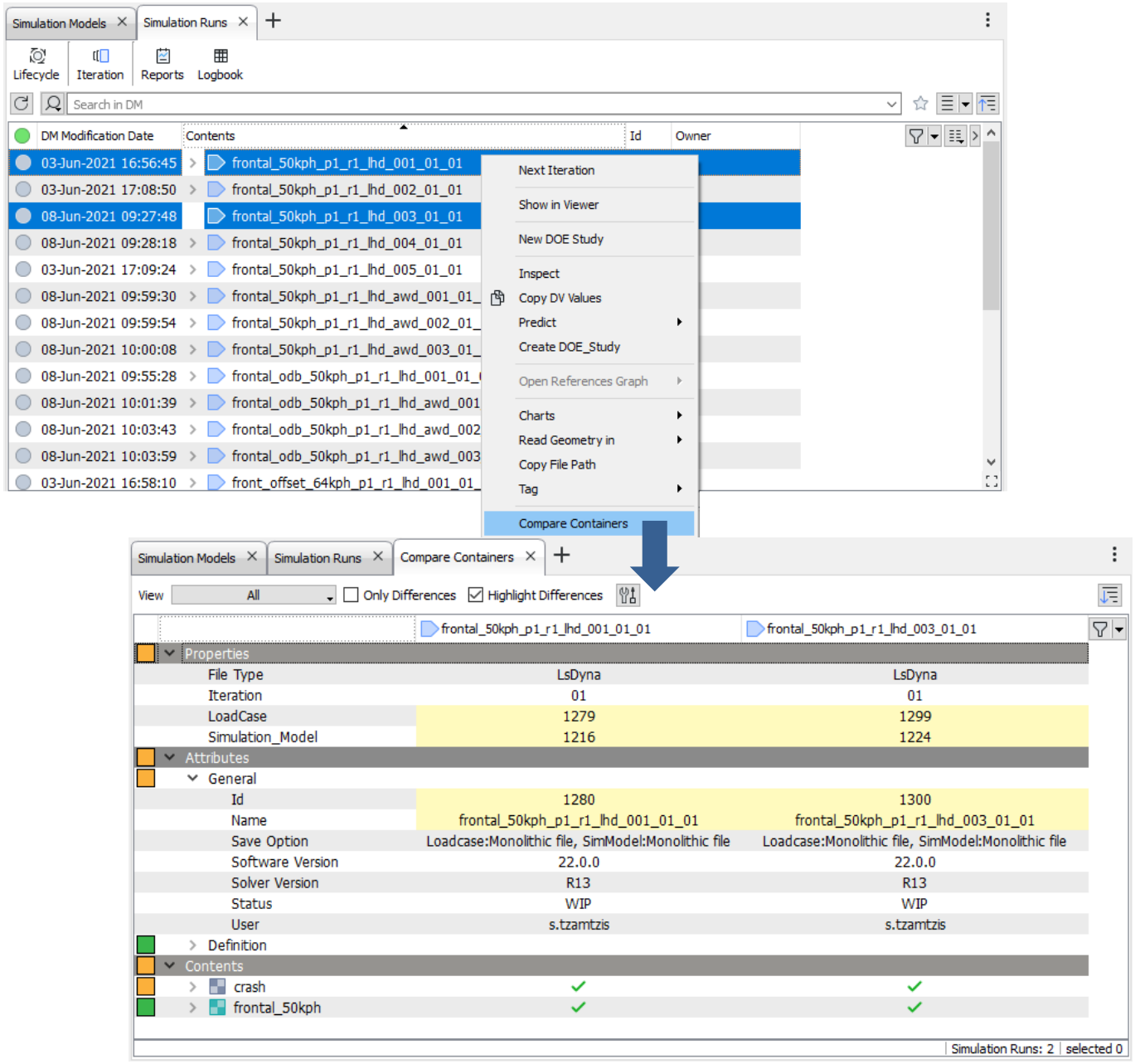
3.10.1. Comparing Base Modules

The comparison of base modules can be triggered from within the SDM client, using built-in functionality that launches the ANSA Compare Tool. The user can select the two base modules of interest (e.g. two different versions of a Subsystem) and use the context menu option **Compare Tool > Compare selected items**, in order to load the two selected modules in ANSA and launch a detailed comparison using the Compare Tool. A detailed description of the Compare Tool is available in Chapter 19 of the ANSA User's Guide.



3.10.2. Comparing Compound Containers

The attributes and contents of Subsystems, Simulation Models, Loadcases and Simulation Runs can be compared in order to find any differences directly in the SDM Client. This can be achieved through the **Compare Containers** tool available in the context menu.



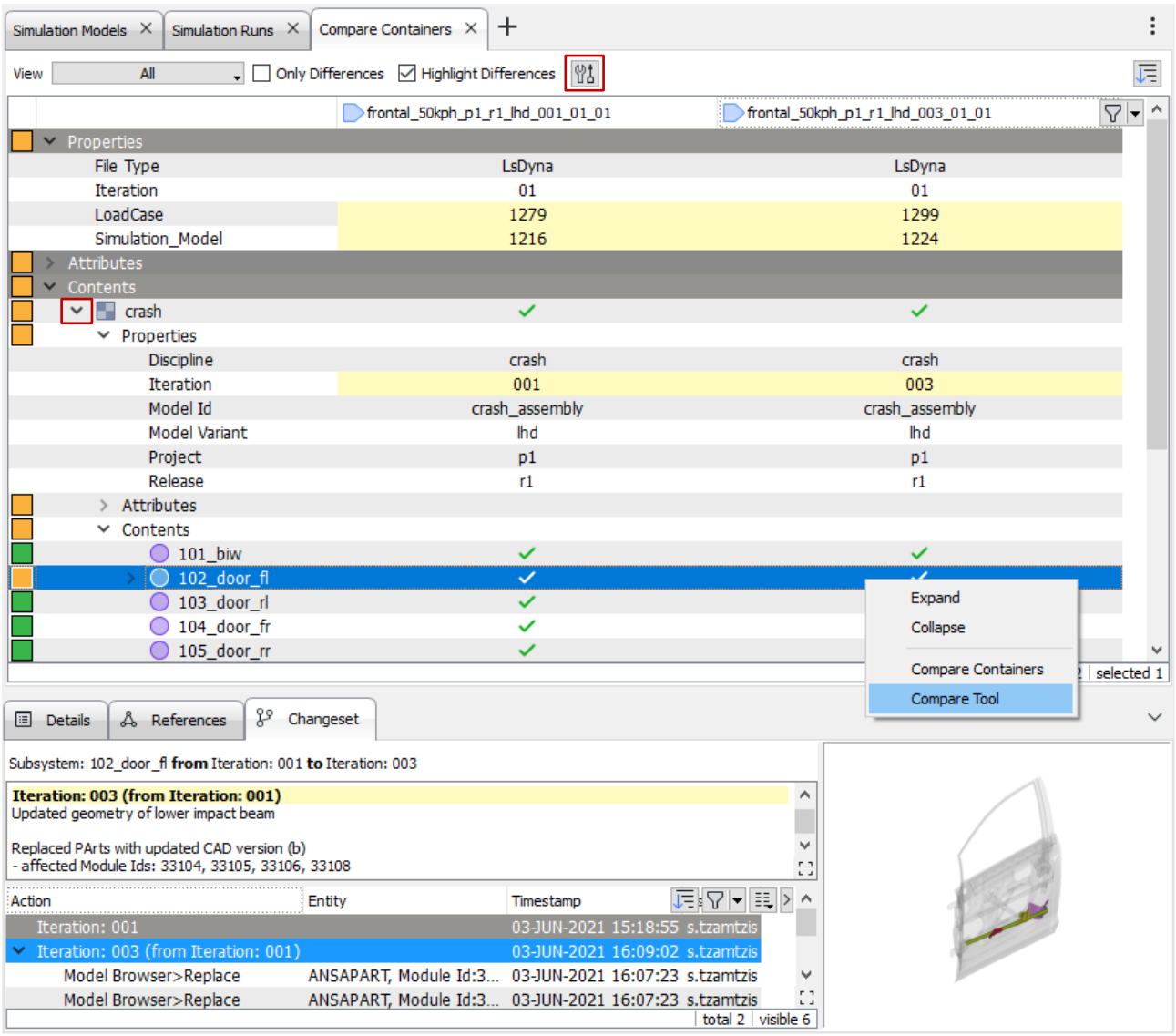
A new tab opens in the SDM Client labelled *Compare Containers*. In this tab the selected objects are displayed into separate columns. Their properties, attributes and contents are listed and any differences are highlighted. A diffing indicator is used to distinguish the listed items:

- Same in both Models
- Different between the two Models
- Entity was added
- Entity was removed

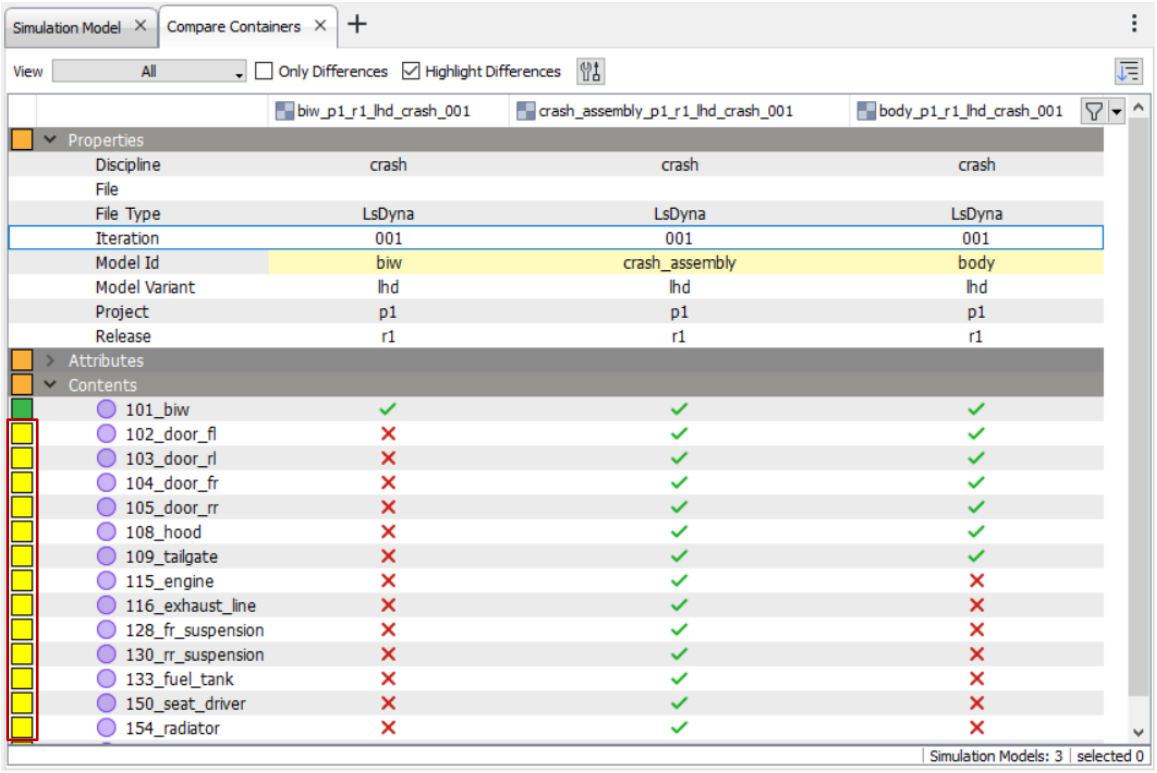
The comparison can proceed to a deeper level, into the contents of the selected containers, by expanding the respective listed entities. The matching of the contents for the comparison is done by using as Matching Key the

first property of the respective DM object type (first according to the order properties are defined in the data model). For example, running Compare Containers on two Simulation Models, the Subsystems Matching Key should be the Module Id. In case of customization of the data model that has led to some unrelated property being defined first, the comparison matching key can be explicitly defined per DM object type through a group of settings defined in the SDM client (i.e. in the *Compare Containers Settings* group in the ANSA defaults or in the *Compare Containers* group in the *Actions* tab of the settings in KOMVOS).

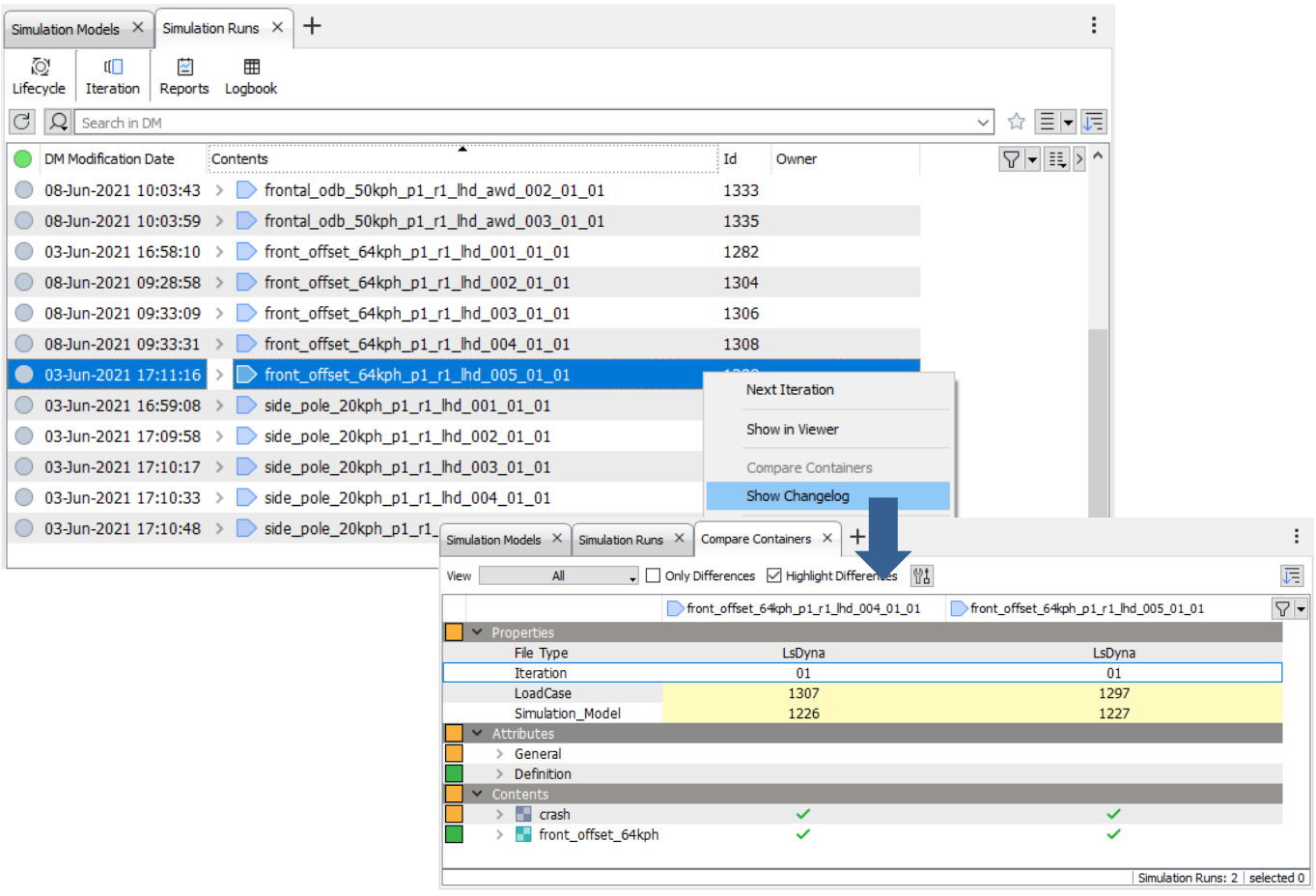
Once the comparison has reached the Subsystem level, it is possible to select any two listed items with differences and in the *Changeset* tab, review their complete Changeset History. Moreover, it is possible to perform a geometric comparison of the two subsystems in ANSA, invoking the ANSA Compare Tool. Finally, a "*Settings*" button allows the user to determine which attributes will be compared. Entire categories of attributes can be excluded from the comparison process in a single shot.



In cases where more than two containers are compared and some of the contents do not participate in all of them, they will be marked as yellow. This indication is used in order to distinguish them from contents that do participate in all containers but have differences (Orange color).



Another option that BETA's SDM Clients offer for the comparison of compound model containers, is the possibility to view the changelog of a new version in respect to its parent. This is achieved using the **Show Changelog** context menu option, which will launch a new *Compare Containers* tab, listing the selected DM object and its parent, for a direct comparison of their properties, attributes and contents.



4. Management of simulation and test results

Management of simulation results within an SDM system is usually associated with the following key requirements:

- **Results storage**: An SDM system must facilitate the structured yet compact storage of results, always in association with the simulation run, loadcase and model they relate to
- **Results review and comparison**: An SDM system must offer tools to review processed results in context, ideally without the need to open an external application, and enable the comparison in of:
    - Simulation Vs Simulation: To compare different versions of a simulation

When access to test results is available, the following additional requirements emerge:

- **Test results review**: The data browsing tools of the SDM system must enable the review of test results in context, ideally without the need to open an external application
- **Test results comparison:**
    - Simulation Vs Test: To correlate simulation results with physical tests, or even
    - Test Vs Test: To compare results of different tests

To meet these requirements, BETA's SDM solutions include dedicated tools for the management of simulation and test results. Through the data browsing tools it is possible to:

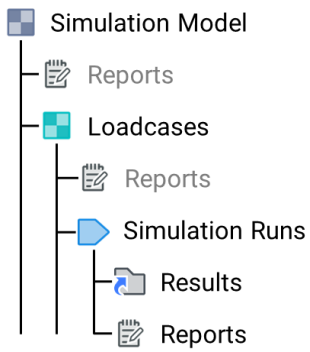
- Attach results under simulation objects
- Review them in the embedded viewer
- Browse the results of ASAM-ODS-compliant test result servers and review them in the viewer
- Compare results of different simulations and
- Compare results of simulation with test.

Furthermore, with the integration of META in the data browsing tools and its direct interface with the SDM systems, additional capabilities are given, as it is possible to:

- Open raw, solver results directly in the post-processor
- Compress results and save them under a simulation object
- Record post-processing sessions "live" and save them in the SDM Library
- Run post-processing sessions "on the spot" and save key-results directly to the SDM system
- Open processed simulation results and test results in the post-processor

The paragraphs below describe the various tools available in BETA's SDM solutions for the management of simulation and test results.

4.1. Simulation Results storage



In the default data model of BETA's SDM solutions, the processed results, which are referred to with the generic term **Reports**, are always associated with a simulation object.

Apart from their most usual association with Simulation Runs, Reports can be also associated to Loadcases, facilitating the storage of reports that accumulate results coming from different Simulation Runs, like for example in the case of pedestrian protection simulations, or even to Simulation Models, facilitating the storage of summarized reports.

Reports may be of any of the following types:

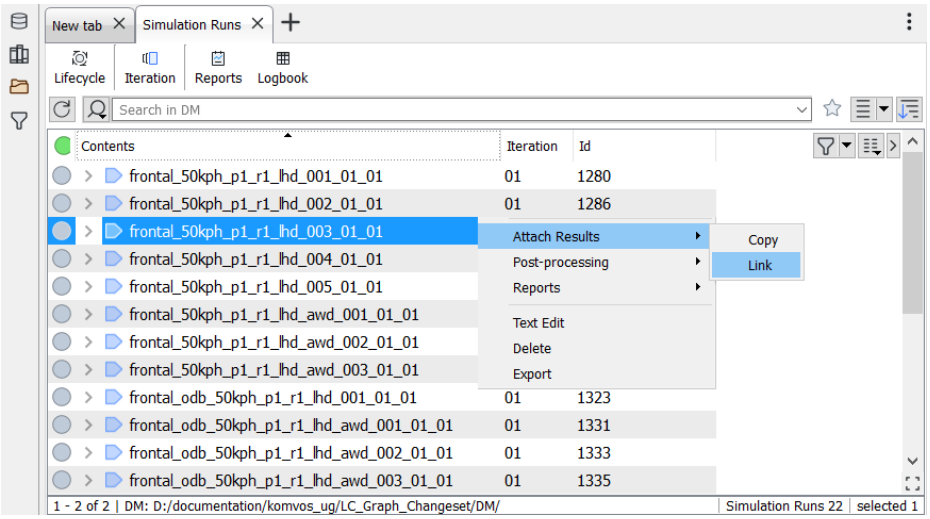
- Images
- Annotations
- Videos
- Key values
- Presentations
- Meta Projects
- Spreadsheets
- Curves
- Tables
- PDF
- HTML
- 3dModel

Depending on the size of the raw results, the following alternatives are offered for the storage of results in the SDM system:

- Upload to the SDM system the processed results (key-results) as Report objects and only keep a link to the raw results, in their original, temporary storage (recommended for raw results of big file size)
- Upload to the SDM system the processed results (key-results) as Report objects, and also upload the raw results compressed, again as Report objects. Result compression can be achieved with META or with 3rd party tools (e.g. FEMZIP)
- Upload to the SDM system the processed results (key-results) as Report objects, and also upload the raw results in an attached directory on the Simulation Run (only recommended for raw results of relatively small file sizes)

4.1.1. Adding solver results under a simulation object

The first step after the submitted job completion is the association of the raw results of the solver with the Simulation Run object they relate to. Raw solver results can be either copied or linked under the Simulation Run DM object through the *Attach results>Copy* and *Attach results>Link* functions of the context menu (due to the big size of raw result files it is recommended to use the "Link" option).



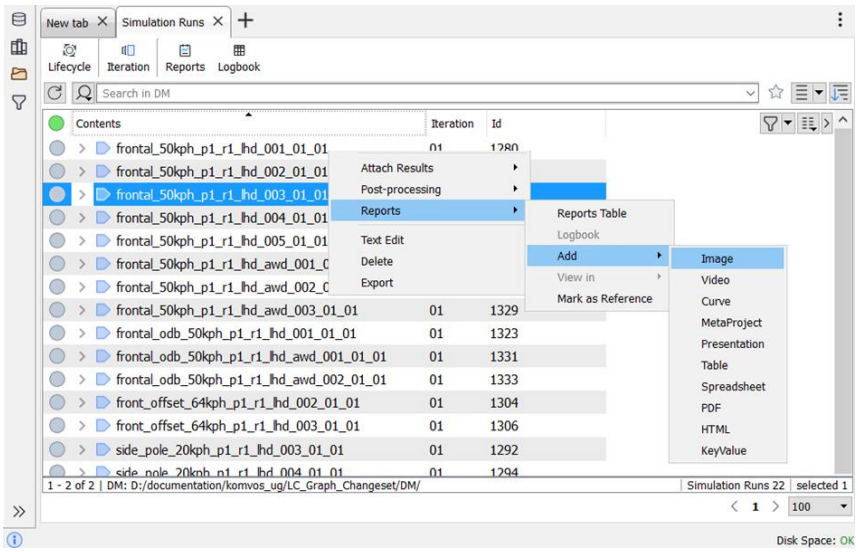
As soon as the copy or link operation finishes, a new folder named **Results** is displayed under the Simulation Run, containing either the result files or a link to the results' containing folder. In any case, this folder will be used later for the automatic reading of geometry in META.

4.1.2. Adding existing Reports under a simulation object

Processed results created with META, scripts or 3rd party post-processors can be easily attached under existing Simulation Runs either manually, through the data browser, or automatically, with the aid of a Python script in KOMVOS, ANSA or META.

4.1.2.1. Through the data browser of the SDM Client

If key-results like images, curves, etc. have been produced by an existing post-processing session and are available as files in the file system, they can be associated to the Simulation Run manually through the Simulation Run context menu option **Add file** for the DM Browser in ANSA & META and **Reports>Add** for the database workspace of



KOMVOS. Upon selecting the type of file to add, a file manager pops-up, prompting the user to select the file to import as a Report item.

4.1.2.2. Through Python

Python script is another option offered by the ANSA, META and KOMVOS for automating the handling of results. All operations related to Report creation and overall management are handled through the `dm` module and more specifically, through the Python class **`DMObject`.** The class methods below are all that is required for the management of Reports:

DMObject methods	Description
`add_new`	Adds a new DM Object in DM
`get_attribute_values`	Gets attributes from a DM Object
`set_attribute_values`	Sets attributes to a DM Object

Adding any DM object must be done always according to its definition in the DM schema. A small snippet for adding a video under a Simulation Run, based on the default data model, is the following in case of ANSA API:

Code snippet

```
try:
    import ansa
except ModuleNotFoundError:
    try:
        import sdm
    except ModuleNotFoundError:
        import meta
        dm_object = meta.dm.DMObject #Script executed from META
    else:
        dm_object = sdm.dm.DMObject #Script executed from KOMVOS
else:
    dm_object = ansa.dm.DMObject #Script executed from ANSA

new_item_props = dict()
new_item_props ["Type"] = "Video" #Type of video as found in dm schema
new_item_props ["Name"] = "my_video"
new_item_props["Report_Parameters"] = ""
new_item_props["File"] = "D:/Temp/crash_video.avi"
new_item_props["Simulation_Run"] = "1197" #id of the Simulation Run
new_object = dm_object(names_values=new_item_props, type="Report")
new_object.add_new()
```

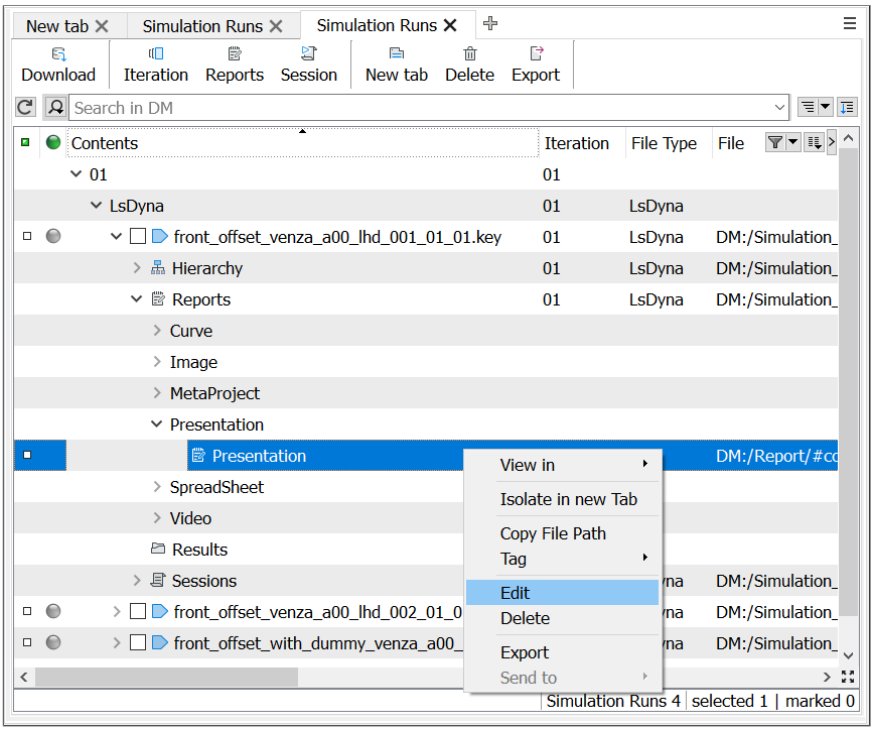
Keep in mind that the construction of the `DMObject` creates an object in memory that represents the dm item, but it doesn't add it in DM. The object method `add_new` will take over to save it in the SDM.

4.1.3. Adding Reports while post-processing with META

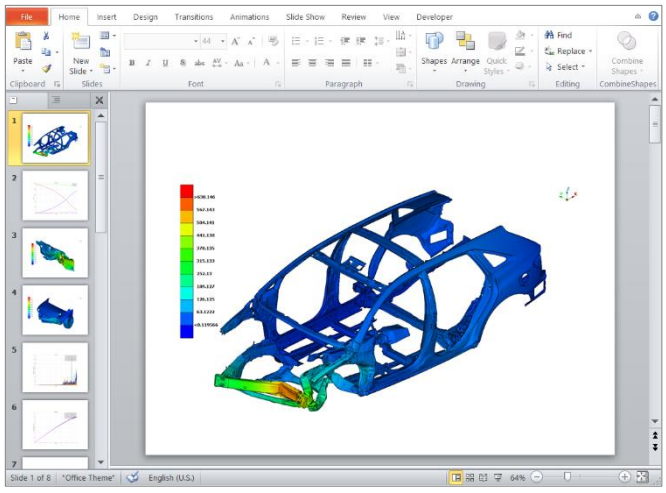
When the post-processor is META, reports can be added directly to the SDM database during the course of post-processing, with the aid of dedicated session commands. This capability is described in paragraph 4.5.1.

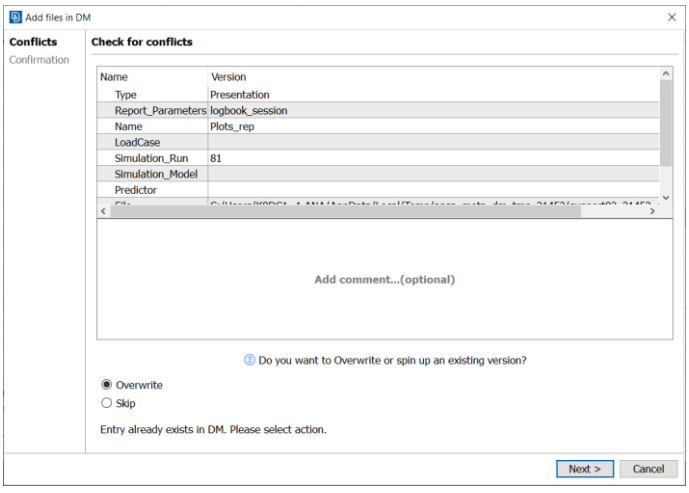
4.1.4. Editing Reports

It is possible to edit the file of a Report, launching the default system application according to file type. In case modifications are made, the Report file in DM can be updated.



The editing can be done either from the list of Simulation Runs and their attached Reports, or through the Reports Table of the DM Browser.





When the **Edit** option is selected, according to the type of the selected Report, the default system application is launched with the file open for editing.

When the changes are completed and the user saves the file, upon quitting the external application, ANSA detects that the file has been updated and prompts the user to either **Overwrite** or **Skip** the saving of the file.

4.1.5. Adding Reports without existing simulation objects

It is possible to store simulation Reports to an SDM system without having the actual pre-processing objects in this system. In this case empty such Simulation Model, Loadcase and Simulation Run objects need to be created, so that Reports can be later searched according to the metadata of the simulation objects they belong to.
To achieve this, a code snippet as the one below can be included in the post-processing session / script that generates the Simulation reports.

Code snippet

```
    # Get or create if it does not exist the Simulation Model. The necessary keyvalues for
the names_values depend on the data model.
    _names_values_sim_model = {}
    _names_values_sim_model['Project'] = 'MyProject'
    _names_values_sim_model['Release'] = 'MyRelease'
    _names_values_sim_model['Model'] = 'MyModel'
    _names_values_sim_model['Main Version'] = '001'
    _names_values_sim_model['Minor Version'] = '002'
    _names_values_sim_model['File Type'] = 'Nastran'

    _sim_model = dm.DMObject(names_values = _names_values_sim_model,
type='Simulation_Model')
    if not _sim_model.server_id:
        _sim_model.add_new()
    if not _sim_model.server_id:
        print('Failed to create and store Simulation model!')
        return None,None
    print('Sim model:', _sim_model.server_id)

    # Get or create if it does not exist the LoadCase. The necessary keyvalues for the
names_values depend on the data model.
    _names_values_loadcase = {}
    _names_values_loadcase['Type'] = 'SomeType'
    _names_values_loadcase['LoadCase Label'] = 'SomeLoadCase'
    _names_values_loadcase['File Type'] = 'Nastran'
    _names_values_loadcase['Simulation_Model'] = str(_sim_model.server_id)
    _loadcase = dm.DMObject(names_values = _names_values_loadcase, type='LoadCase')
    if not _loadcase.server_id:
        _loadcase.add_new()
    if not _loadcase.server_id:
        print('Failed to create and store LoadCase!')
        return None,None
    print('Loadcase:',_loadcase.server_id)

    # Get or create if it does not exist the Simulation Run. The necessary keyvalues for the
names_values depend on the data model.
    _names_values_run = {}
    _names_values_run['File Type'] = 'Nastran'
    _names_values_run['Simulation_Model'] = str(_sim_model.server_id)
    _names_values_run['LoadCase'] = str(_loadcase.server_id)
    _names_values_run['Main_File'] = ''

    _sim_run = dm.DMObject(names_values = _names_values_run, type='Simulation_Run')
    if _sim_run.server_id:
        print('Simulation Run with same properties exists! No new Simulation Run stored!')
    else:
        _sim_run.add_new()
    if not _sim_run.server_id:
        print('Failed to create and store LoadCase!')
    print('Simulation Run server id', _sim_run.server_id)

    #Set the Simulation Model or the Simulation Run as the object under which the Simulation
Reports will be stored
    utils.MetaCommand('dm setcurid {}'.format(sim_model.server_id))
    utils.MetaCommand('dm setcurid {}'.format(sim_run.server_id))
```

4.2. Simulation Results review

The SDM clients include several tools for the review of simulation results. These tools can manage and visualize **Report** DM Objects, i.e. processed results and also **Test Results** (e.g. Channels, photos, movies, 3D geometries, etc.)

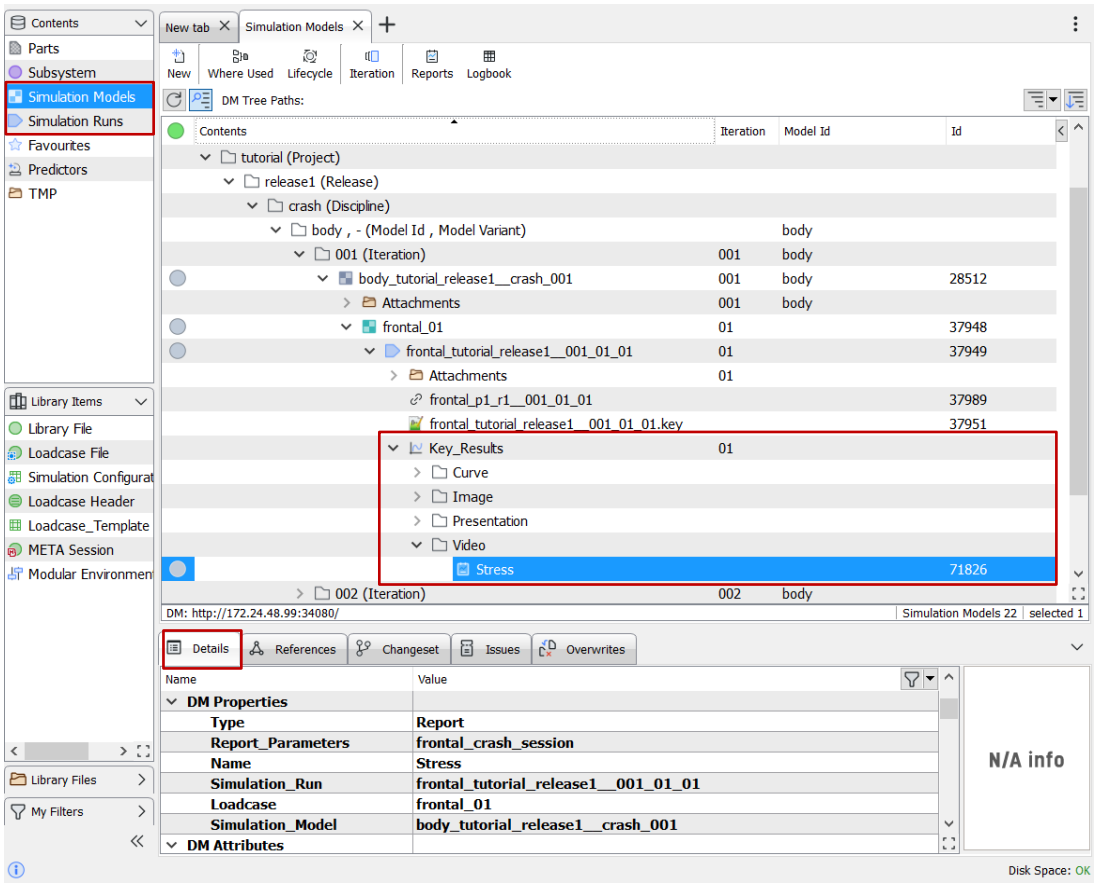
The main tools for the review of Reports are:

- **Reports Table**: A concise view for the listing of all Reports of selected simulations in a per-type fashion
- **Embedded META Viewer:** Used for the visualization of selected Reports and Test Results.

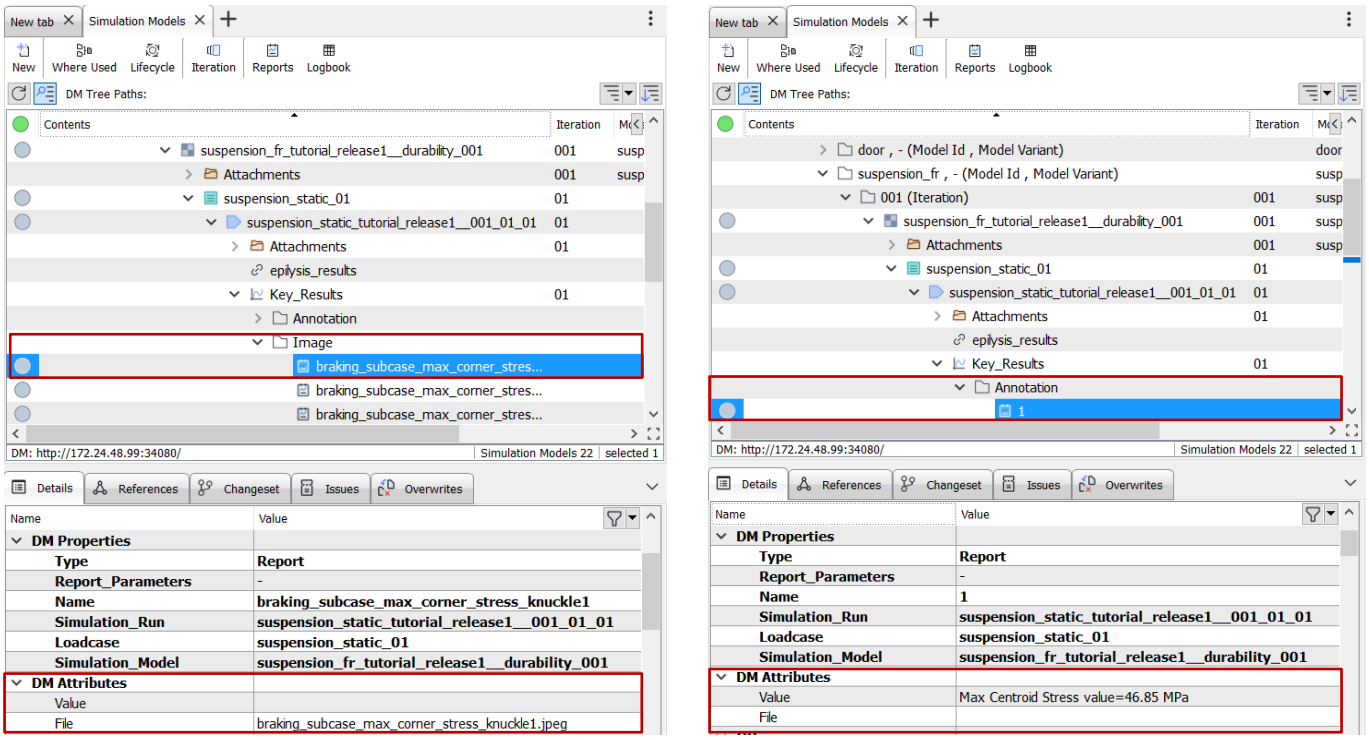
Note that that the results reviewing tools do not substitute the post-processor. The post-processor is still needed for the extraction of key-results from the raw results of the solver.

4.2.1. Presentation of Reports in the main lists

Reports in the SDM clients are attached directly under the objects they are associated with and are displayed in the Simulation Models and Simulation Runs tabs, while in the Defaut View. In the Simulation Models tab, expanding a Simulation Model will show the Loadcases it was used with, and expanding further will show the Simulation Runs produced by this Simulation Model - Loadcase combination. Finally, expanding the Simulation Run will reveal the Reports attached to the latter, as shown in the image below. By selecting a Report, the metadata of the object are displayed in the Details tab.

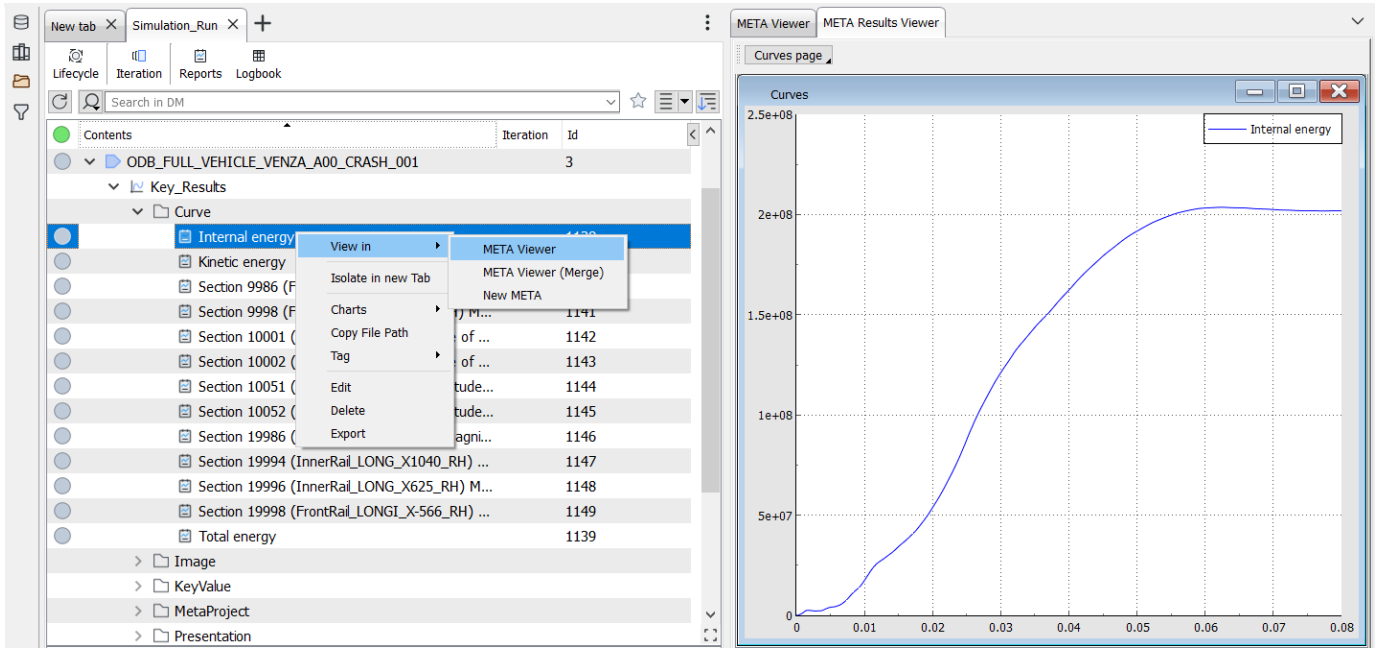


For a selected Report, in the DM Attributes category of the Details tab, the name of the attached file is displayed and in case of Annotations or KeyValues, the value is stored in the Value field.

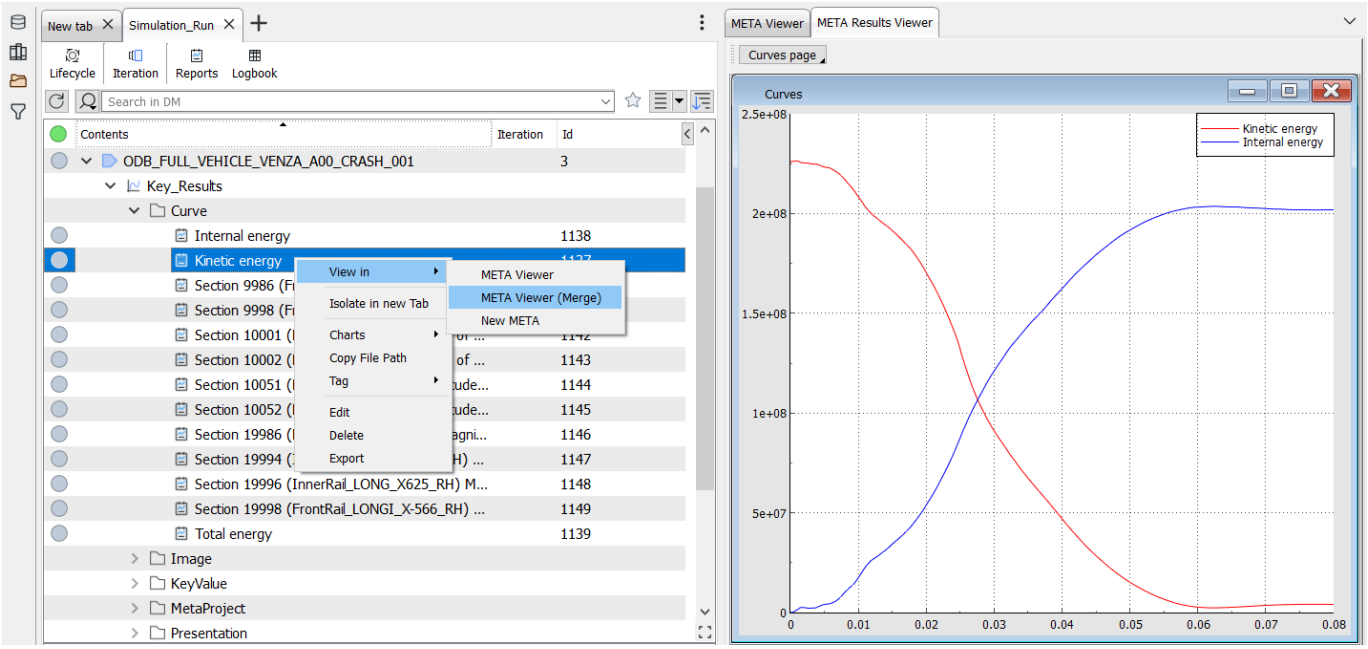


4.2.2. Viewing Reports in the embedded META Viewer

Reports can be view in the embedded META Viewer by selecting a listed item and triggering through the context menu, the action **View in > META Viewer**. The report is displayed in the tab with the label META Results Viewer.

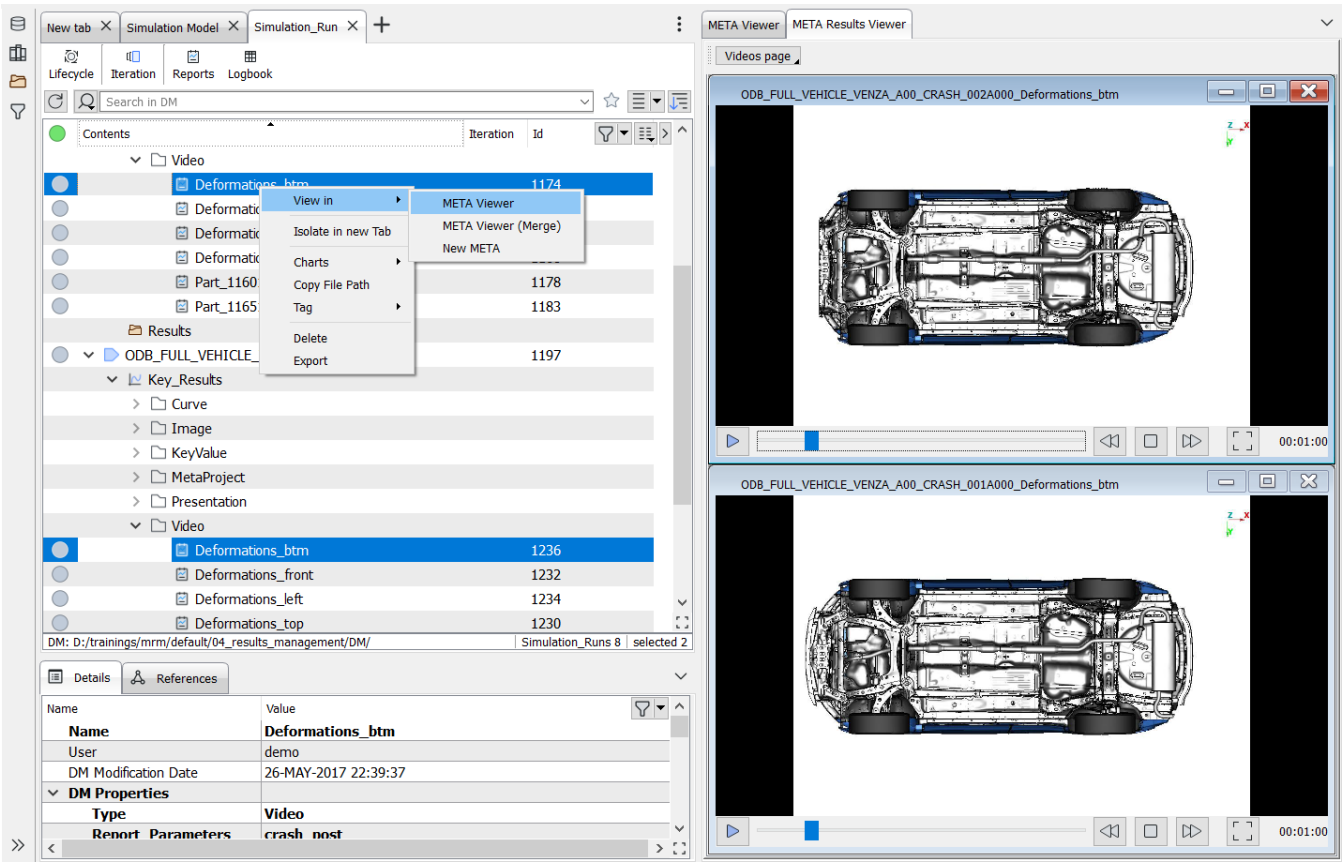


Reports of the same type can be added in the existing META Viewer with the option **View in > META Viewer (Merge)**. For example, Reports of Curve type can be added in the same graph.

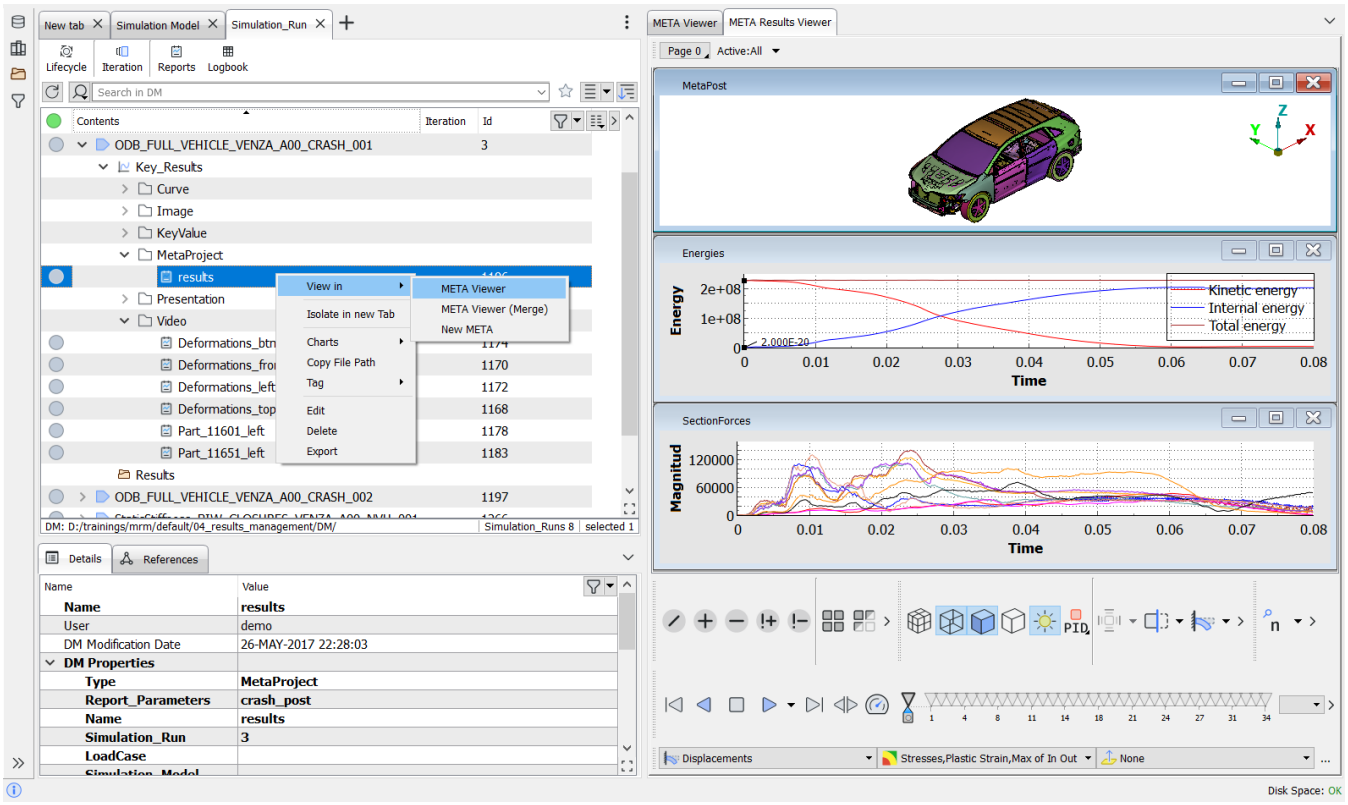


By selecting the option **New META**, META is initialized and the selected Report is displayed.

When sending to the viewer a multi-selection of presentations or videos, they are presented in a tiled layout and their navigation is also synchronized.

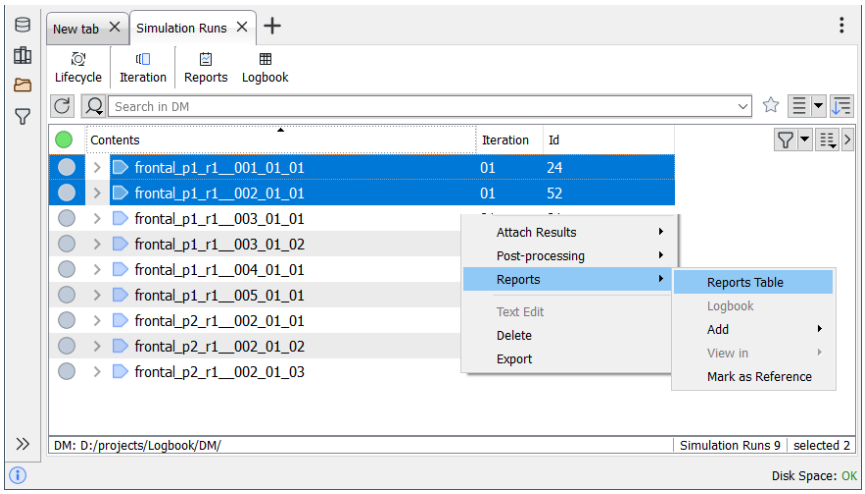


In case of the META project, the viewer will be split in as many sections as necessary in order to show the entire content.

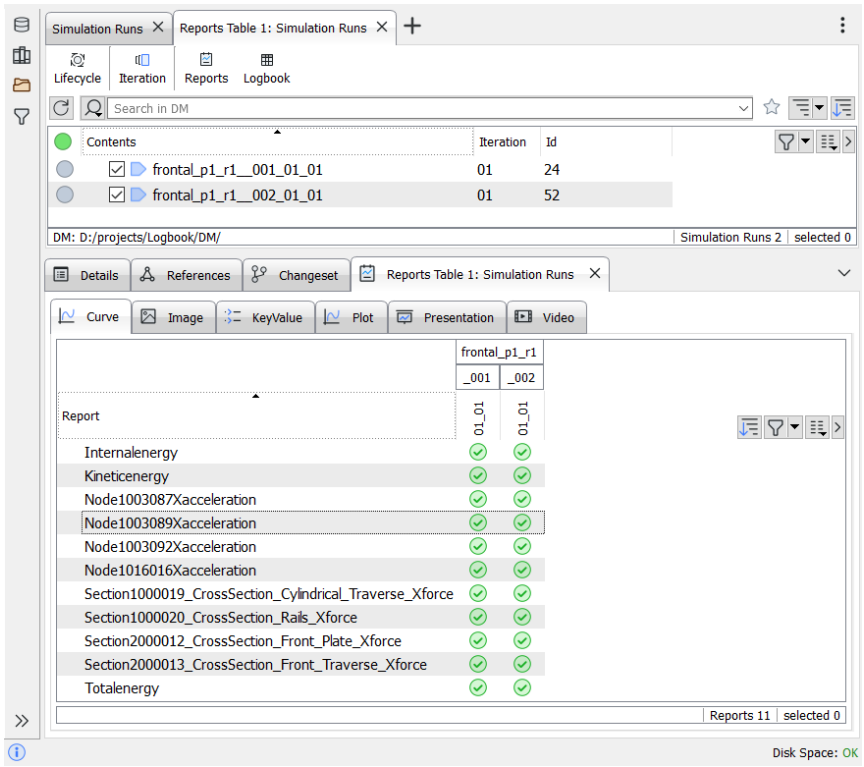


4.2.3. Presentation of Reports in the Reports Table

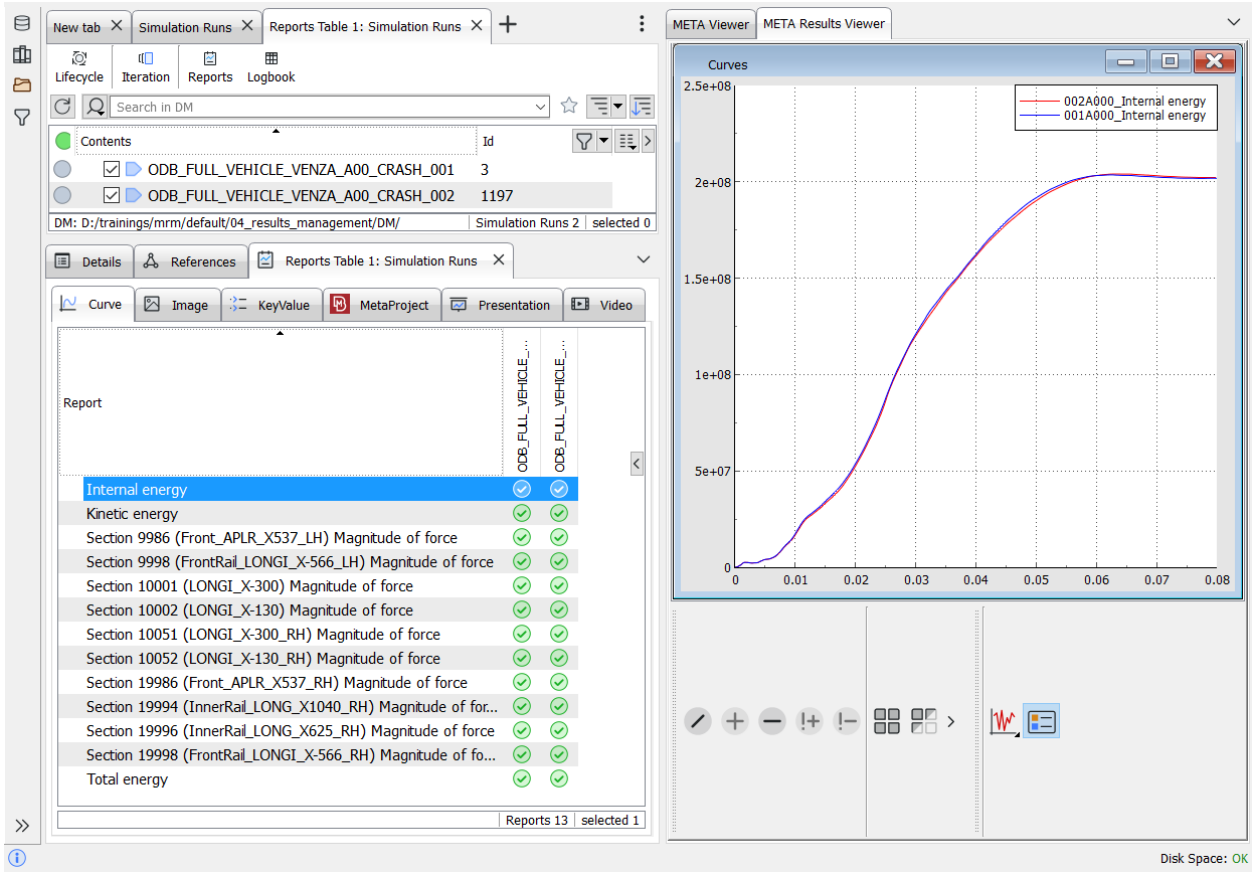
The main implement for viewing the reports of a Simulation Run is the **Reports Table**, a special for the presentation of Reports in a concise manner. This tool can be accessed through the Simulation Models, Loadcases and Simulation Runs context menu option **Reports > Reports Table**.



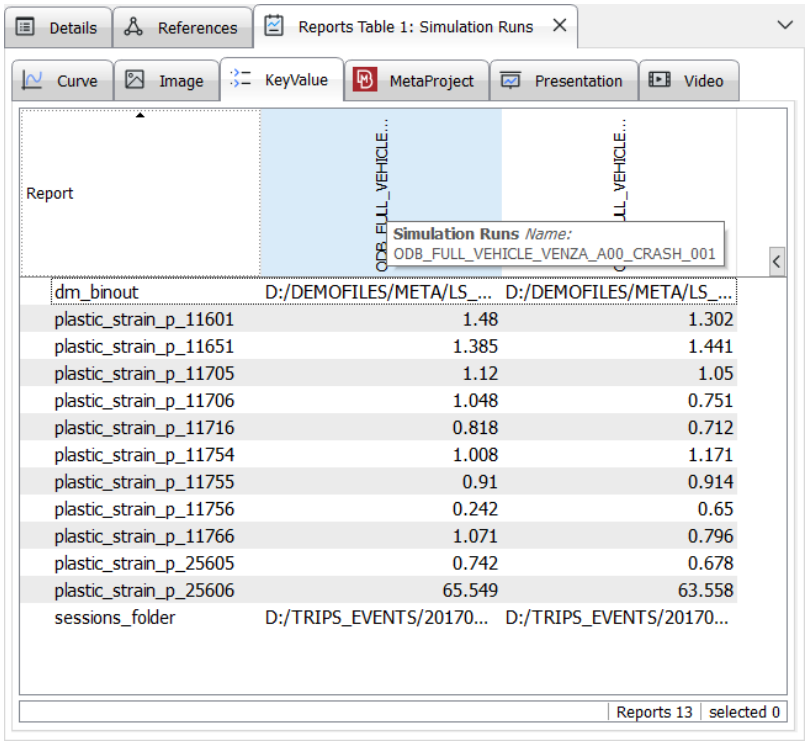
The function opens a new tab in the bottom section of the window, where all different Report types are organized in sub-tabs. Each tab shows a list of all available Reports of this type.



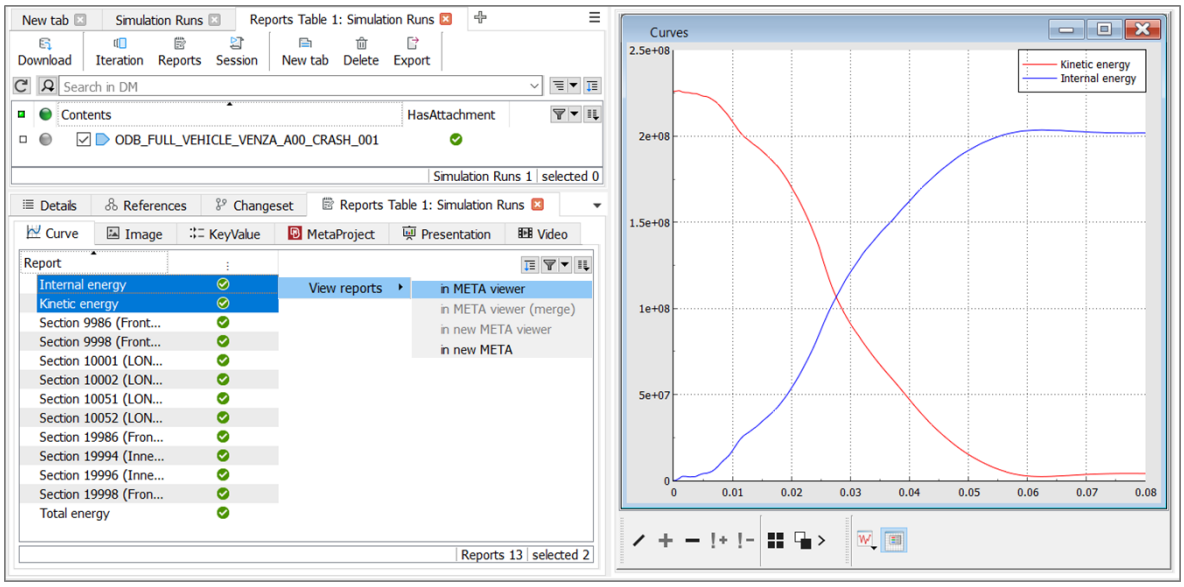
In order to view any of the Reports graphically, double click on a report to open it in the embedded META Viewer in the right section of the main window.



Results like Key values or Annotations, which only concern numeric values or text, display their values directly inside the **Reports Table**.

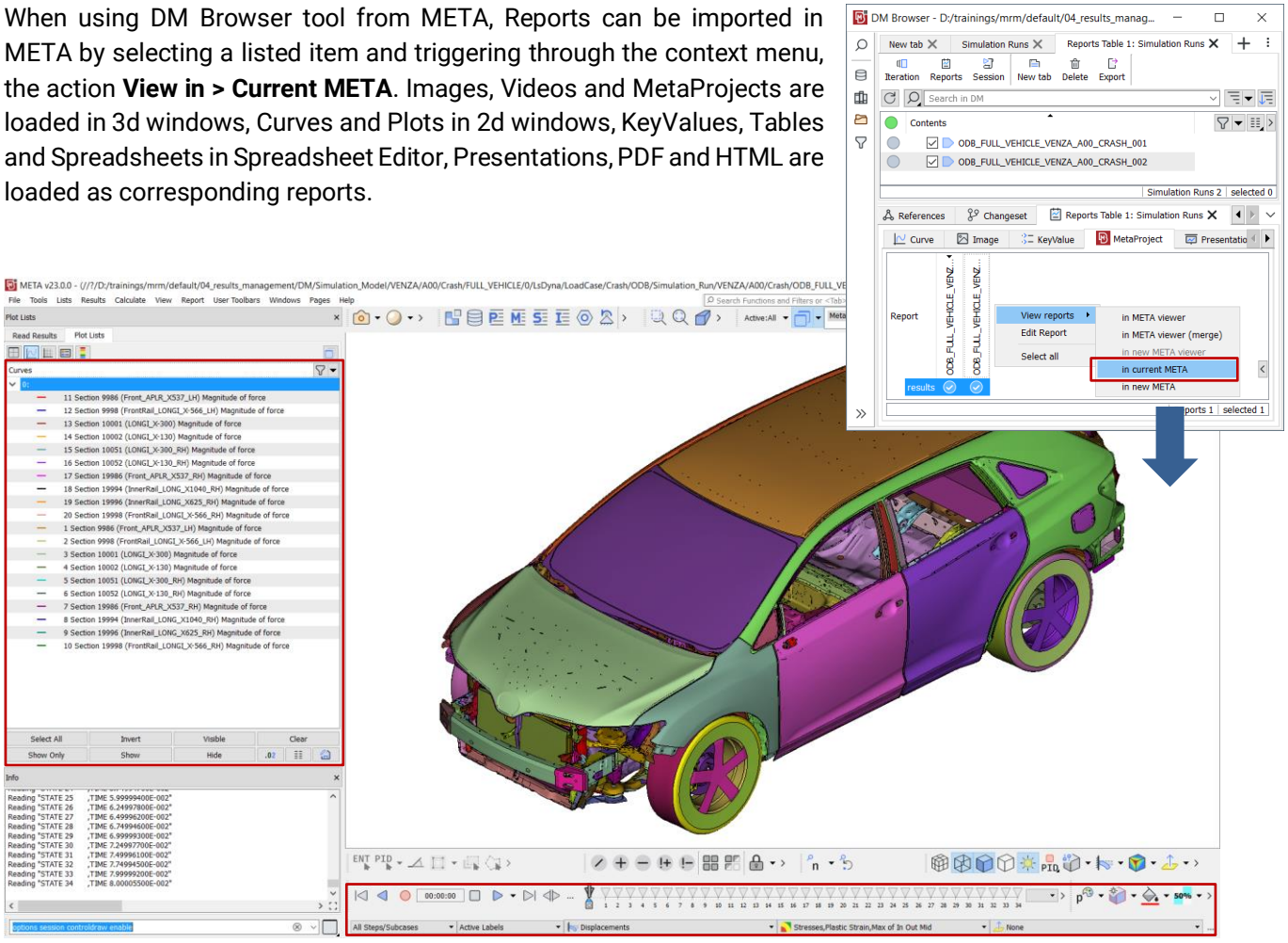


Multi-selection is also available for the same type of results. As an example, two or more selected curves can be sent to META Viewer through the function **View reports>In META viewer** of the context menu. The Viewer will superimpose the curves, offering a comprehensive view to the analyst.



4.2.4. Viewing Reports in META

When using DM Browser tool from META, Reports can be imported in META by selecting a listed item and triggering through the context menu, the action **View in > Current META**. Images, Videos and MetaProjects are loaded in 3d windows, Curves and Plots in 2d windows, KeyValues, Tables and Spreadsheets in Spreadsheet Editor, Presentations, PDF and HTML are loaded as corresponding reports.



It is also possible to load a Simulation Report using META session commands as described in paragraph 4.5.1.

4.3. Simulation Results comparison

A very important aspect of the results assessment process is the efficient comparison of results produced by different Simulation Run versions. In the SDM Client applications, the same tools used for simulation results review are also used for comparison of simulation results. These tools can manage and visualize **Report** DM Objects, i.e. processed results.

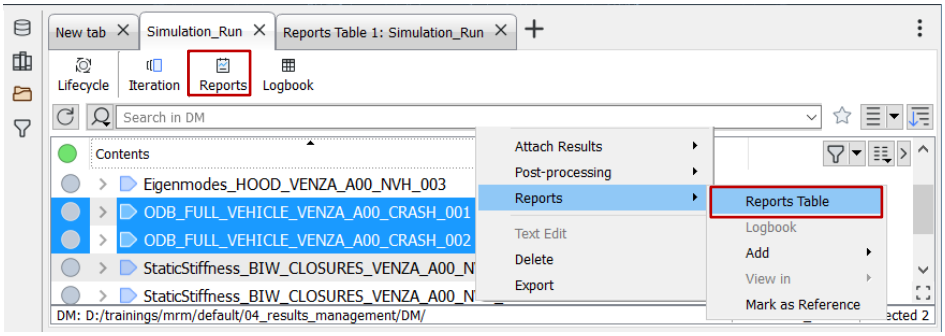
The main tools for the comparison of Reports are:

- **Reports Table**: Used to review reports of different simulations side by side
- **Embedded META Viewer:** Used for the overlaid presentation of Reports coming from different simulations
- **Results Logbook**: Used for the tabulated presentation of key-values from different simulations, with custom highlighting of cells based on conditional formatting

4.3.1. Comparing results in the Reports Table

The Reports Table is a dedicated tab for the efficient review of reports. The Reports Table tab is divided in two main sections: the main list and the below section. In the below section, for every available Report type, a dedicated tab is created. All Report type tabs are grouped under the Reports Table tab in the below section.

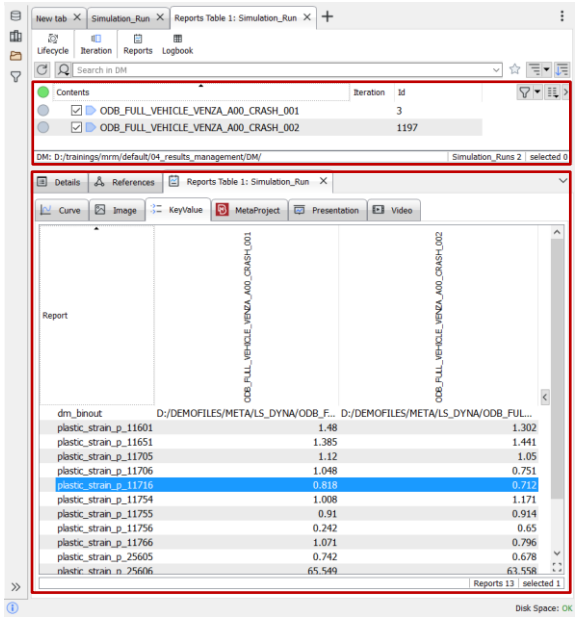
To trigger the Reports Table action, first the desired Simulation Runs are selected and either from the context menu the **Reports > Reports Table** option can be selected or the Reports button in the top toolbar can be pressed.

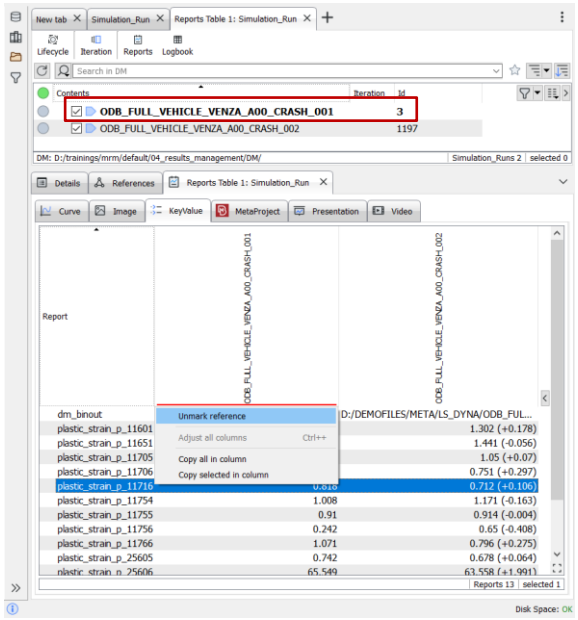


At every Report type tab, the Simulation Runs' names are displayed vertically in the columns and in each row, the Report Name and the value for each Simulation Run is shown.

The side-by-side display of the values, specifically for Report types of Key Values and Annotations, allows the easy and fast comparison between the various Simulation Runs.

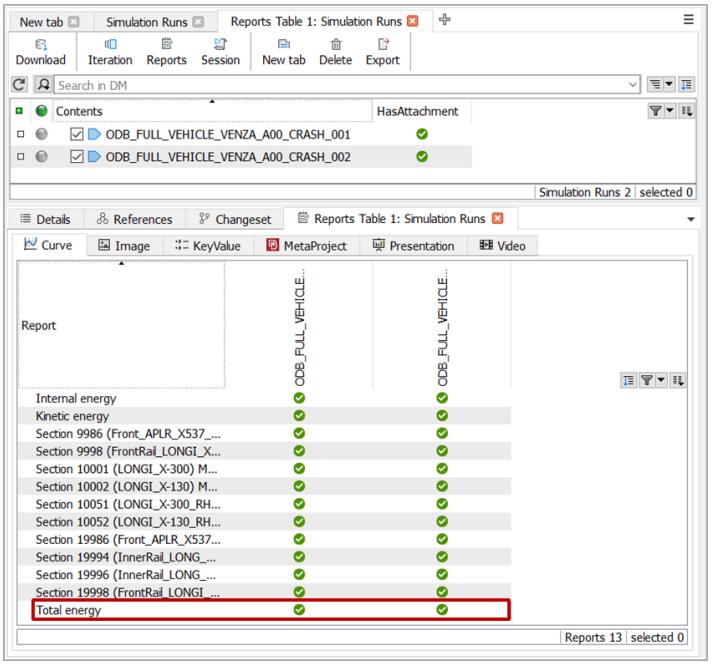
Within the Reports Table View it is possible to mark a Simulation Run as a reference, so that Key Values of all other runs are compared against it (a percentage difference is also provided). In order to mark a Run as reference, select the option **Mark Reference** available in the context menu of Simulation Runs in the main list and in the context menu of the column header that corresponds to the run of interest in the bottom tab.





In case of more than one selected Simulation Runs, the Reports Table lists all available Reports per type and per Simulation Run. Different Simulation Runs are displayed in different columns.

A first level comparison of available Reports between Simulation Runs is done inside the **Reports Table**. Inside this table, the analyst can identify at a glance which reports are available for which Simulation Runs.

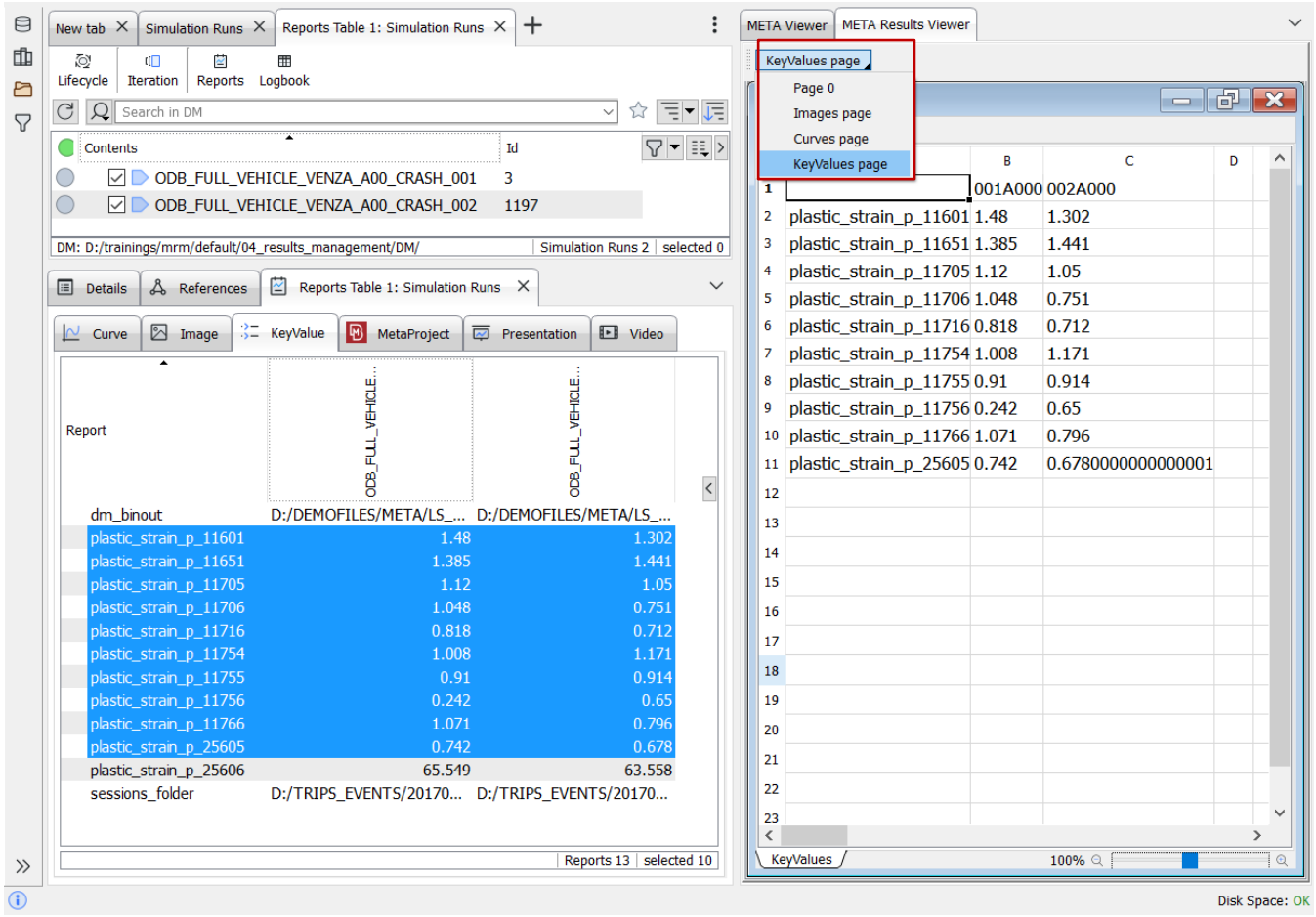


4.3.2. Comparing results in the embedded META Viewer

The data browsing tools in the SDM clients make sure that when two or more versions of simulations are about to be compared, their Reports will be overlayed automatically in the embedded META Viewer. In this way, the analyst does not need to spend time finding the proper Reports to compare and can focus on result evaluation.

Using the **Reports Table** as a starting point, once a report is available for more than one Simulation Run, a more detailed comparison between the reports is possible inside the META Viewer by double clicking on any report of the table. Depending on the type of report, the viewer will pick a suitable comparison mode, e.g. curves will be shown in the same plot or, in case of key values, they will be shown one next to other in a table view.

Having used the **View reports > in META Viewer** option, a dedicated page per Report type (e.g. Curves, KeyValues etc) is created, while the previously Report type pages are kept by the META Viewer. By clicking on the **<Report_type> page**, a drop-down menu is opened and the user can easily swift to a previously opened Report type page.



4.3.3. Results Logbook

The quick and efficient review of the key results of different simulations is of great importance for the analyst. In order to assess a simulation, key values of the results are usually collected and compared against threshold values. This task can be quite time consuming and is usually carried out with some script or automation in the spreadsheet editor.

To simplify this task, the SDM clients offer the **Results Logbook** that is available in the **DM Browser** of ANSA and META and in the **Database Workspace** of KOMVOS. Using the Results Logbook, processed simulation results are presented in a cumulative tabular view. The Logbook is based on an Excel Spreadsheet which is used as a Template and is the key for advanced yet user-friendly customization. More than one Templates can be used in a DM, to satisfy the needs of different teams or even different Loadcases. Furthermore, different Templates can be defined for Simulation Models, Loadcases and Simulation Runs. More often the Logbook includes Reports of type “Key Value”. However, other Report types such as Images or Videos can also be included. Such Reports can be displayed in the embedded META viewer.

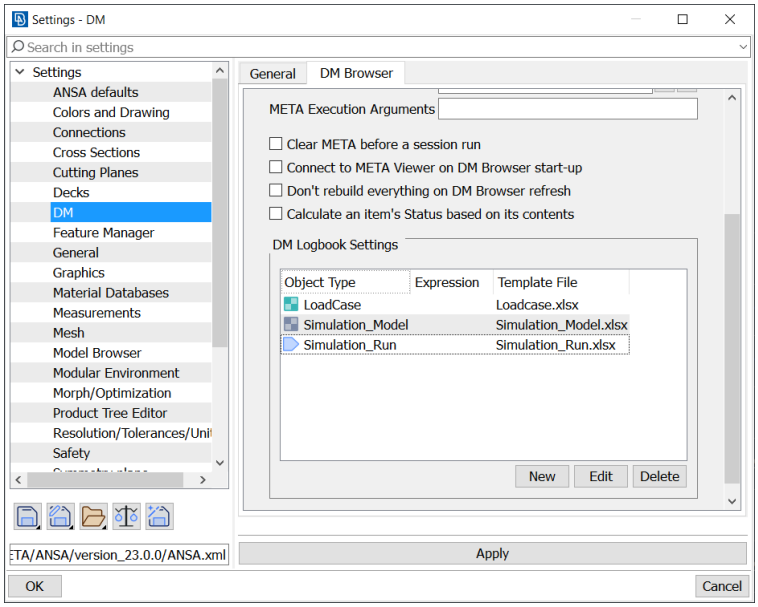
The usage of the Results Logbook leads to a quick overview of results, making it easy to compare the different simulation versions and identify the best simulation model versions, saving the analyst the time spent in the manual creation of logbooks or in the development and maintenance of complex scripts.

4.3.3.1. Configuration

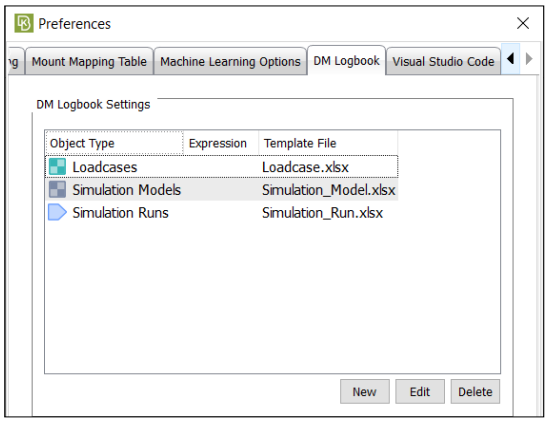
First, the Excel file that will be used as a Template for the Logbook must be defined. The definition of the Template file in ANSA is done through the *Settings > DM > DM Browser > DM Logbook Settings*, in META through the *Settings > Global Settings > DM > DM Logbook Settings*, where as in KOMVOS through the *DM Logbook* tab of the *Preferences window*, accessed through the *Extras > Settings*.

It is possible to use different Templates for different Object Types. The Object Types that can contain Reports are:

- Loadcase
- Simulation Model
- Simulation Run

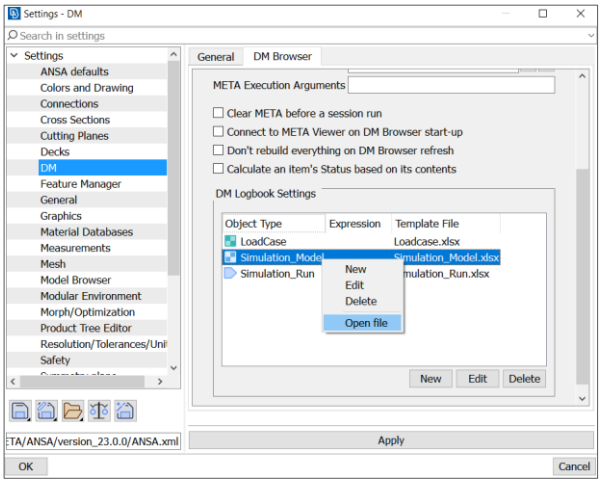


Logbook configuration in ANSA

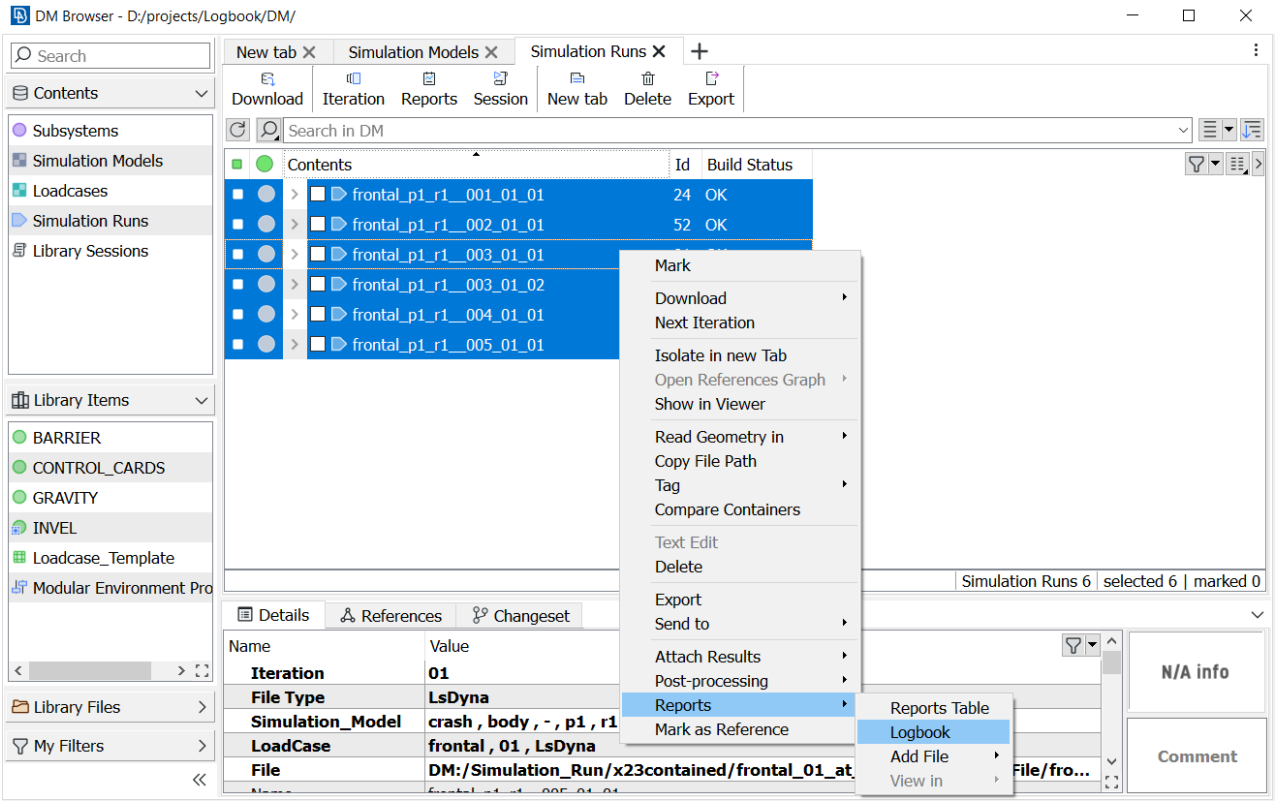


Logbook configuration in KOMVOS

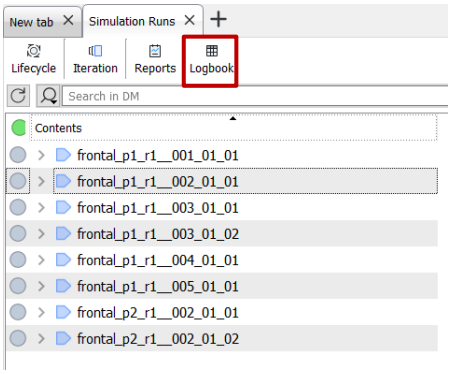
With the **Edit** option, the template that is set for a specific Object Type can be changed. From the context menu, by selecting the **Open file** option, the Template file is opened by the system's application and can be reviewed and/or edited.



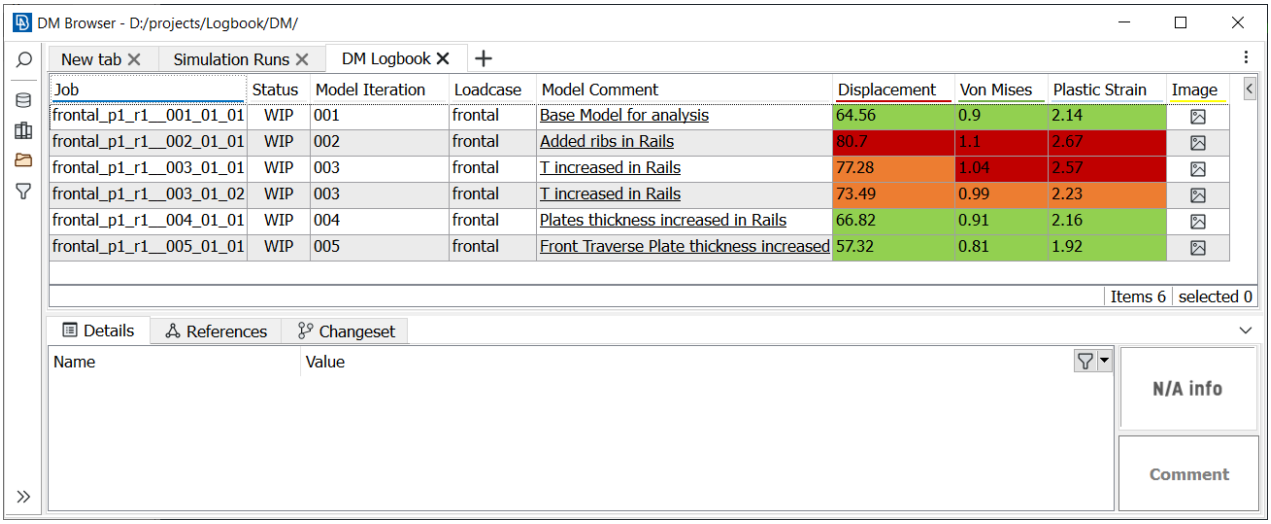
Having set the Template files for the specific Object Types, the **Reports > Logbook** action can be triggered from the context menu of the selected DM Objects in the respective data browsing tool.



In KOMVOS the *Logbook* can also be viewed by pressing the respective button in the toolbar.

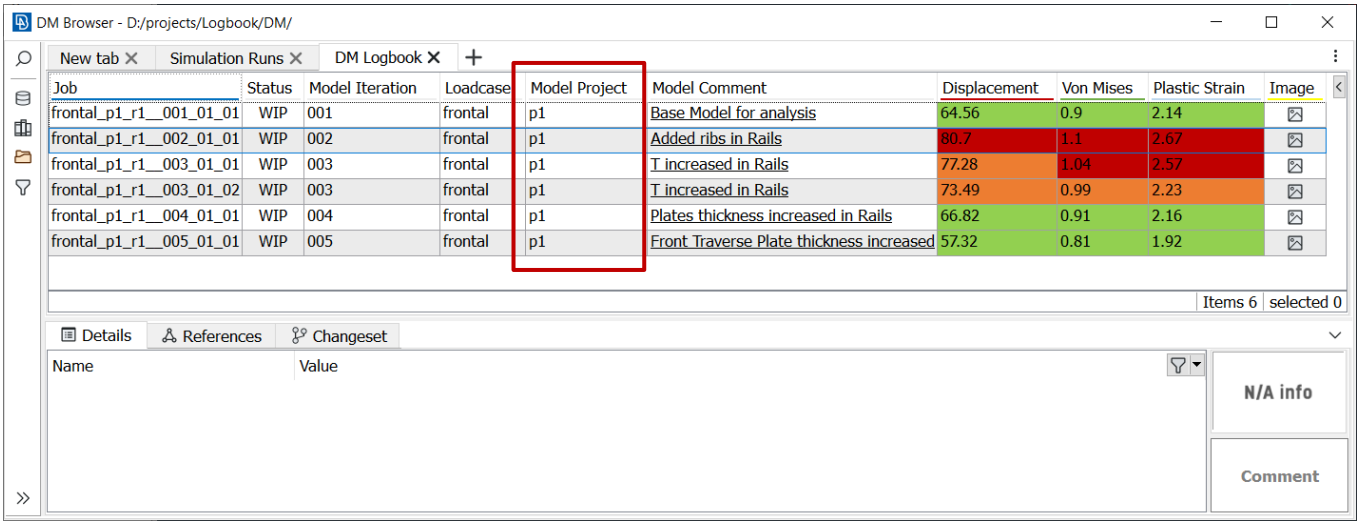


The *Logbook* is displayed in a new tab at the data browsing tool, from which it was executed.



The mechanism of the DM Logbook is always aware of the status of the Excel template; if changes are applied in the template, the **Logbook** action can be triggered anew, resulting in an updated view of Results.

For example, the column with the project of the Model is added in the Excel Template file. The Template file is saved, and the action is executed again.



4.3.3.2. Template set-up

Using an Excel file as Template, it is easy to set up the Logbook while exploiting features of this widely used software. To set up the Excel file, some rules need to be followed. These rules can be separated in two main categories:

a) Syntax rules
b) Formatting rules

A. Syntax

A1. Column headers

The first row in the Excel sheet is the source for the creation of the column headers in the Logbook. To add a column in the Logbook, the name of the header must be typed enclosed between # characters (e.g., **#Iteration#**) in the corresponding cell of the first row in the Excel sheet.

Notes:

If the # characters are removed from a column header cell, the whole column will be “hidden” and will not be displayed in the Logbook.

A2. Data rows

The cells of the main data rows are filled with the aid of expressions that are defined in the second row of the template. These expressions follow a BETA QL-like syntax and can be used to retrieve attribute values of the primary object type of the Logbook or attribute values of related objects (e.g. for a Logbook defined for Simulation Runs, the primary object type is Simulation Run). Expressions must be enclosed between **$** characters and the attribute names must be typed precisely as they appear in DM (e.g., **Id**).

The expressions used to retrieve attribute values differ depending on the DMObject type for which the Template file is registered. Assuming a Logbook Template defined for a **Simulation Run**, Simulation Run attributes like *Iteration* are retrieved as follows:

```
$Iteration$
```

Attributes of the run contents i.e., of the Simulation Model or Loadcase like *Model Variant* or *Loadcase Id*, are retrieved with the following syntax:

```
$Simulation_Model->"Model Variant"$
```

```
$LoadCase->"Loadcase Id"$
```

To retrieve information of the Reports contained in the Simulation Run, the syntax depends on the type of Report. For Reports of Type “Key Value”:

```
$Report->Value with Type = KeyValue and Name contains <keyword_in_name>$
```

For all other Report Types:

```
$Report with Type = <report_type> and Name contains <keyword_in_name>$
```

Assuming a Logbook Template defined for a **Simulation Model**, Simulation Model attributes like *Model Variant* are retrieved as follows:

```
$Model Variant$
```

Attributes of the runs using this Simulation Model, like Run Name, are retrieved with the following syntax:

```
$Simulation_Run->Name with <constraints>$
```

<constraints> stands for the syntax to define the desired filters. For example, to get the Simulation Runs that use this Simulation Model and have the string “LATERAL” in their name, the following expression is used:

```
$Simulation_Run->Name with Name contains LATERAL$
```

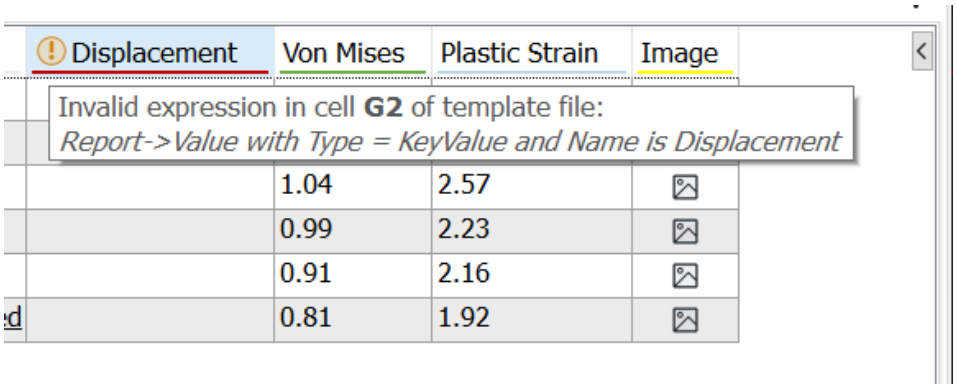
Notes:

1. No blanks should be left between the attribute names and the arrow symbol.
2. To set fixed values, the **$** character must be omitted from the expression.
3. If for the expression set in the Template file, there is no value in the DMObject, the corresponding cell in the Logbook will be empty.

A3. Invalid expressions

In case an invalid expression is inserted in the Excel file, an orange exclamation mark is displayed in the column header with the invalid cell. If the cursor is placed on the header, a window pops up, indicating that there is an error in the expression for the attribute value.

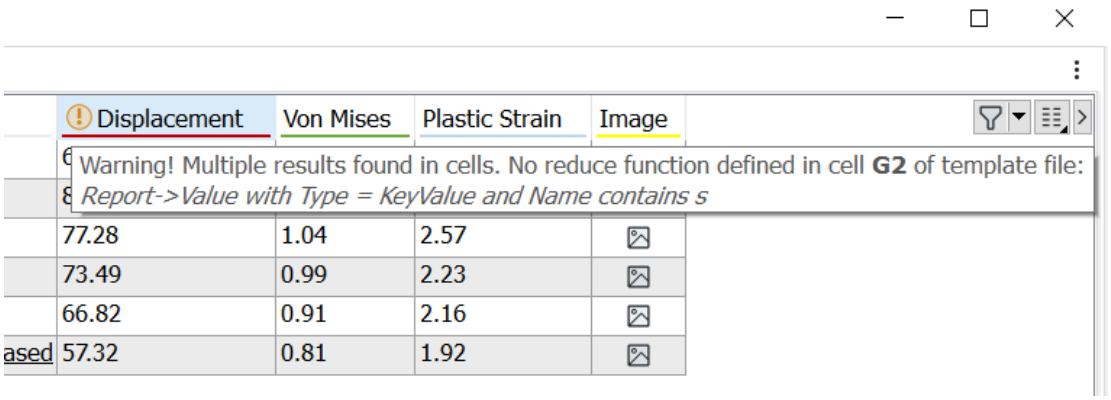
Invalid expression in cell referring_cell of template file.



A4. Multiple results, reduce functions

In cases where an expression returns more than one results, the orange exclamation mark is displayed once again in the column header, but with a different message this time:

Multiple results found. No reduce function defined in cell referring_cell of template file.



To get one value of the multiple results, there are some reduction functions that can be used complementary to the basic expression. These are:

- MIN
- MAX
- FIRST
- LAST

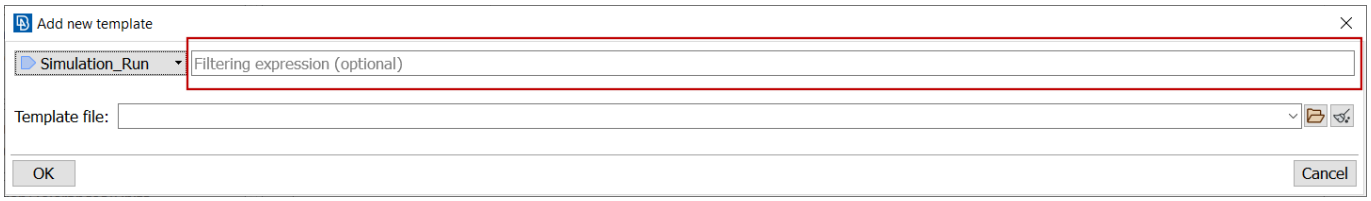
Let's assume a case where the basic expression returns more than one KeyValue Reports. In this case, the expression becomes:

```
$<reduction_function> Report->Value with Type = KeyValue and Name contains <keyword>$
```

A5. Multiple templates for same DM Object type

It is possible to define and use different templates for the same DM Object type (e.g. Simulation Run), to cover the reporting needs of different Loadcases, Disciplines or even Releases and Projects.

In the DM Logbook Settings field, *Settings > DM > DM Browser > DM Logbook Settings*, an expression can be used to define the criteria to search for a specific DM Item, which will make use of the template.



The expression syntax to define the criteria for the search of a DM Item varies depending on the level of the target Property. If the desired Property is of the DM Object type that is selected, the expression is defined as follows:

```
"<Property>" <operator> <Value>
```

If the desired Property is a property of a contained DM Object (e.g. Project of a Simulation Model), the expression is of the following syntax:

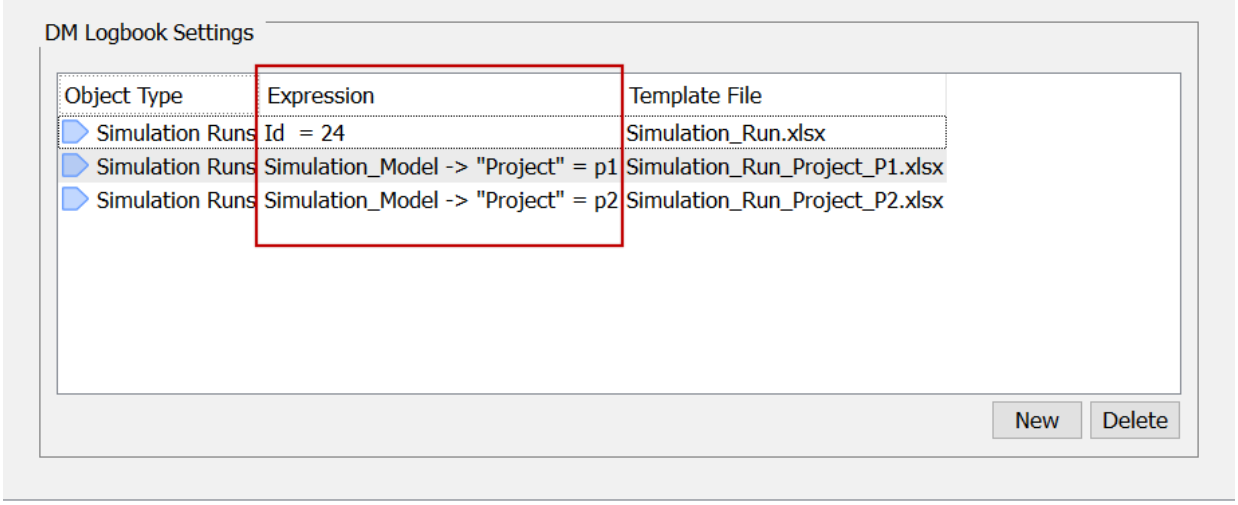
```
<DM Object Type> -> "<Property>" <operator> <Value>
```

The operator "->" is used since the search is applied at a second level.

Let's assume a scenario, for which, different Logbook templates are needed for DM Objects with type Simulation Run, depending on the Project Value.

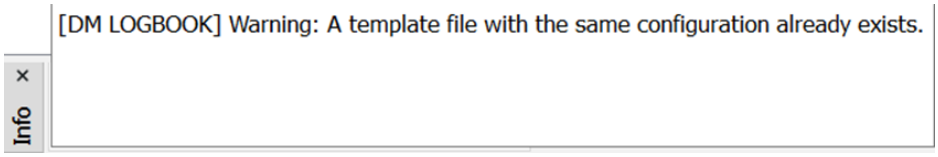
The expression syntax is the following:

```
Simulation_Model -> "Project" = <project_name>
```



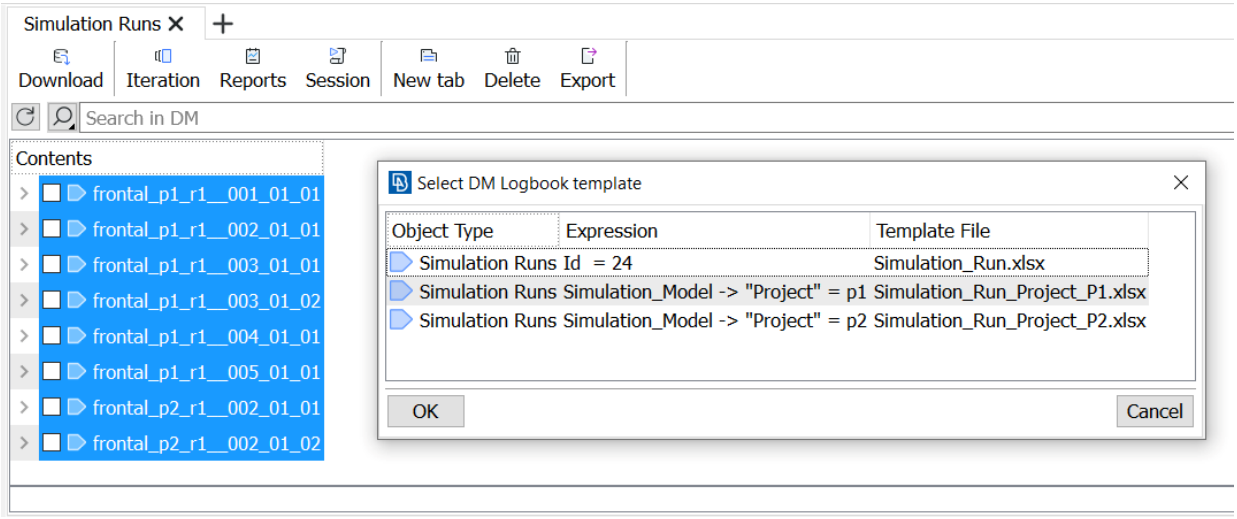
Note: The syntax for the search is based on BETAQL expressions.

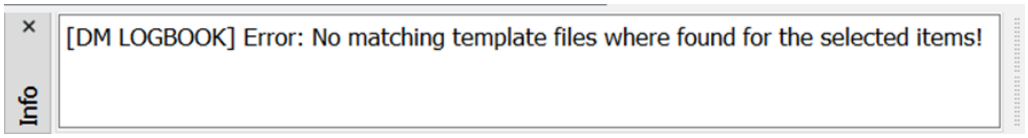
In case that there is already a configuration set, with the same template selected and the same expression, a relative message is printed in the Info Window.



After the definition of the Logbook templates, the moment the Logbook view is requested for a selection of DM Items, the matching templates are searched based on the expressions set and:

- In case a single template is defined, the Logbook opens directly at a new tab
- In case more than one suitable templates are identified, a window pops-up asking the user which template to use
- In case no suitable template is found, a relevant message is printed in the Info window



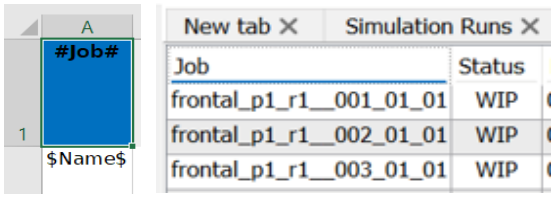


B. Formatting

Certain formatting applied in the template Excel file is passed to the Logbook. The main formatting options possible are described in this paragraph.

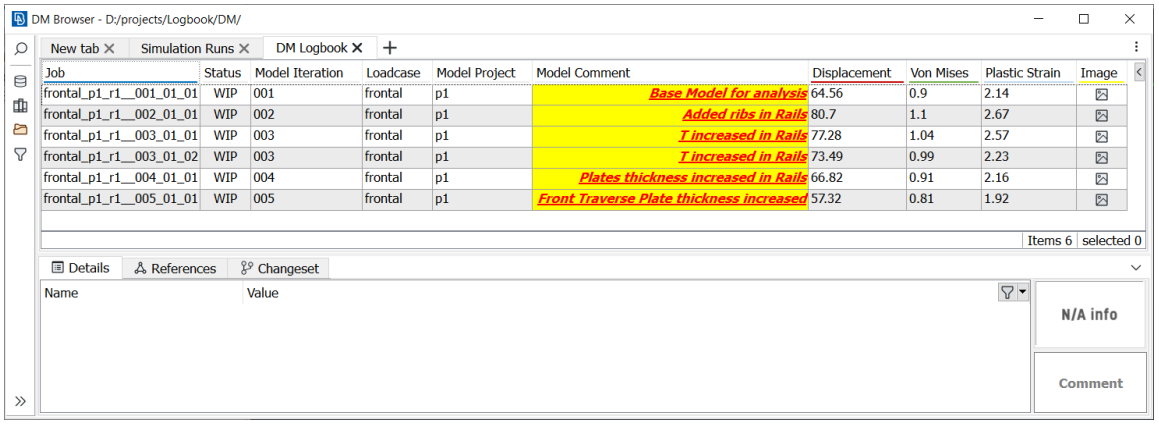
B1. Header formatting

If the header cell is filled with a color, this specific color is displayed as a line below the column header in the Logbook.



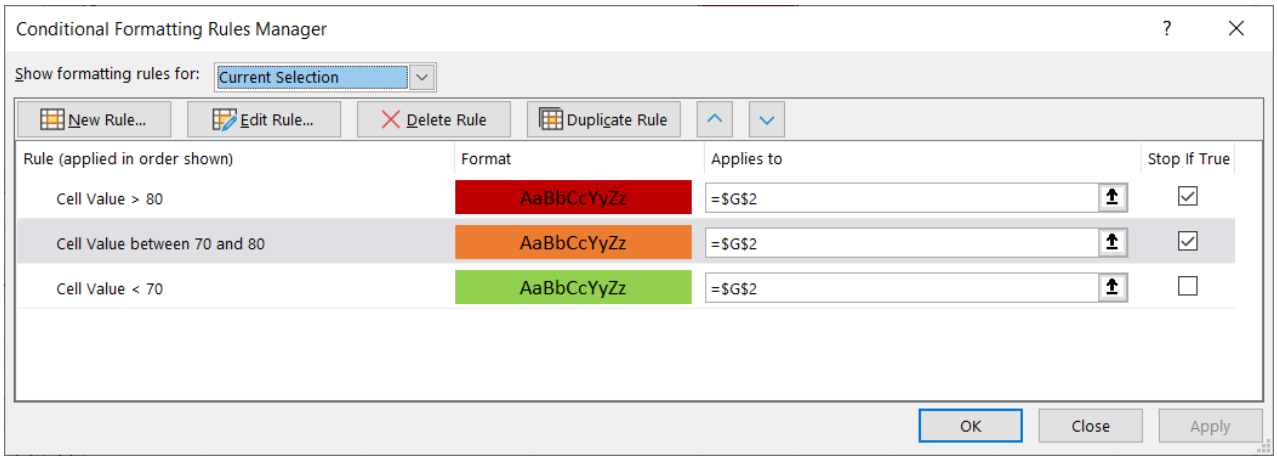
B2. Text formatting

The text formatting is applicable only for the cells of the data rows, not for the cells of the header row. Horizontal alignment, font style (e.g., bold, italics, underline), text and cell color are all passed to the Logbook.



B3. Conditional formatting

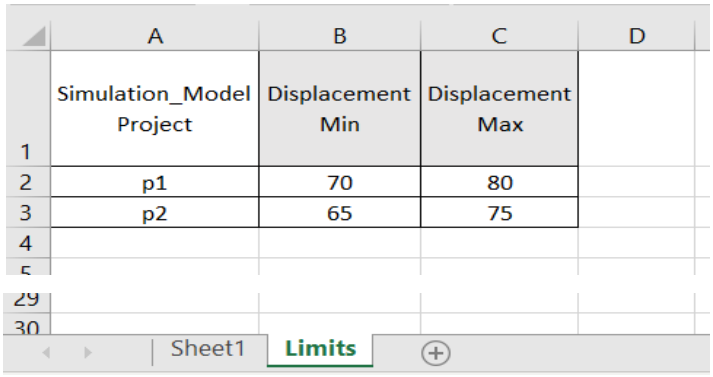
A key feature of the Logbook is the possibility to use the conditional formatting functionality of spreadsheets. With conditional formatting, the values retrieved for each cell of the data rows are compared against threshold values and are formatted accordingly, allowing good overview of the results and quick focus on the information of interest.



Conditional formatting includes the capability of defining threshold values through reference to other Excel Spreadsheets. This functionality enables the creation of multiple combinations of thresholds and makes it easier for the user to handle these scenarios.

Example

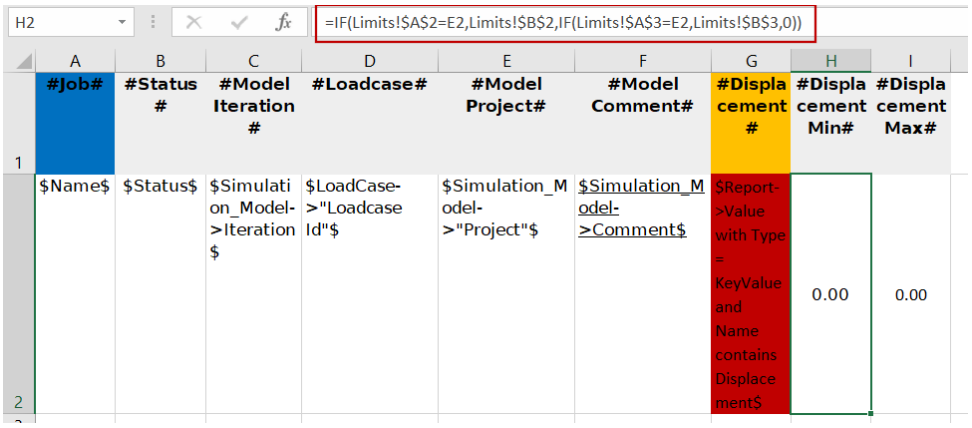
Different threshold values are needed for the Displacement values of Simulation Runs that use Simulation Models of different Projects. A second Spreadsheet with the threshold values is set.



	A	B	C	D
1	Simulation_Model Project	Displacement Min	Displacement Max	
2	p1	70	80	
3	p2	65	75	
4				

Sheet1 | Limits

At the main Spreadsheet, a check is made for the Project value and the equivalent minimum and maximum values are set.



H2 =IF(Limits!$A$2=E2,Limits!$B$2,IF(Limits!$A$3=E2,Limits!$B$3,0))

	A	B	C	D	E	F	G	H	I
1	#Job#	#Status #	#Model Iteration #	#Loadcase#	#Model Project#	#Model Comment#	#Displacement #	#Displacement Min#	#Displacement Max#
2	$Name$	$Status$	$Simulation_Model->Iteration$	$LoadCase->"Loadcase Id"$	$Simulation_Model->"Project"$	$Simulation_Model->Comment$	$Report->Value with Type = KeyValue and Name contains Displacement$	0.00	0.00

Note:

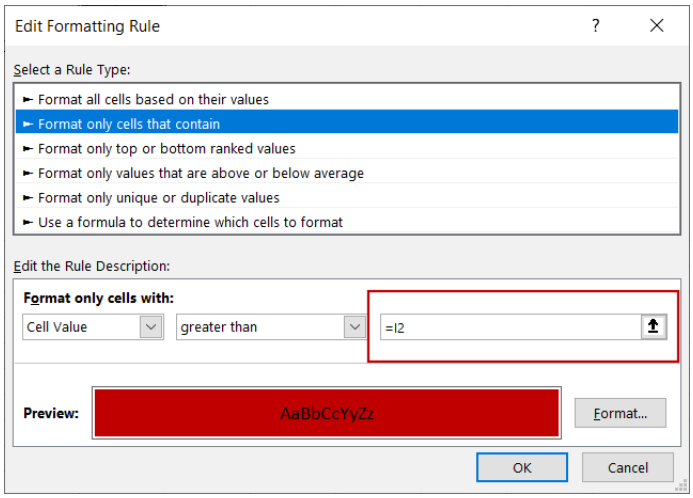
In the check cells the syntax of the referencing should be first of the referencing cell and then the cell of the main sheet

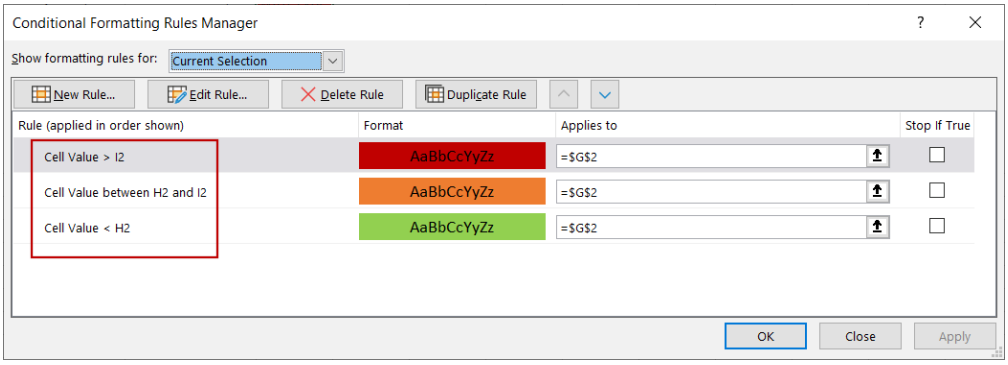
```
<threshold_sheet>!$<column>$<row>=<column><row>
```

In the example above, the syntax that is used is:

```
Limits!$A$2 = E2
```

The Displacement Min and Max values are used in the conditional formatting. When setting the limits in the Conditional Formatting Rules Manager, the “$” symbol should NOT be used in the Cell Value.





Activating the DM Logbook after having set the threshold values in the conditional formatting Project depended, a different mapping of the results is achieved.

Simulation Runs | DM Logbook

Job	Status	Model Iteration	Loadcase	Model Project	Model Comment	Displacement	Displacement Min	Displacement Max
frontal_p1_r1__003_01_02	WIP	003	frontal	p1	T increased in Rails	73.49	70	80
frontal_p1_r1__004_01_01	WIP	004	frontal	p1	Plates thickness increased in Rails	66.82	70	80
frontal_p1_r1__005_01_01	WIP	005	frontal	p1	Front Traverse Plate thickness increased	57.32	70	80
frontal_p2_r1__002_01_01	WIP	002	frontal	p2	Added ribs in Rails	77.58	65	75
frontal_p2_r1__002_01_02	WIP	002	frontal	p2	Added ribs in Rails	69.32	65	75

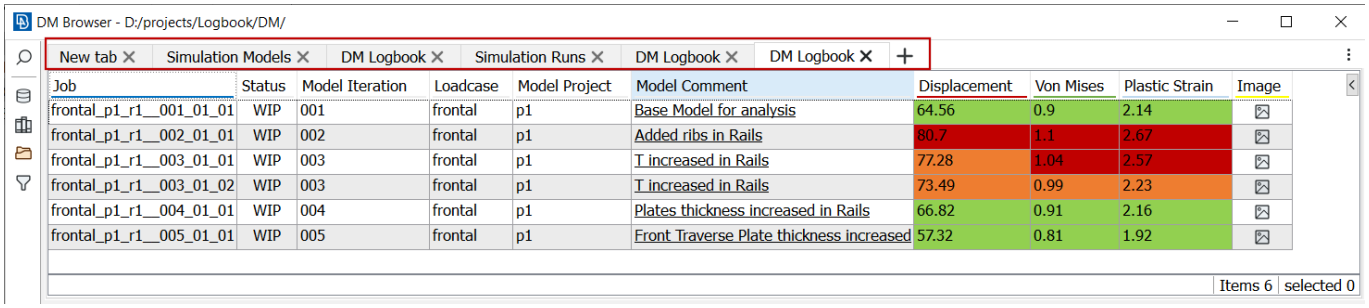
4.3.3.3. Working with the Logbook

The Logbook is more than just a table with values and colors. Numerous possibilities are there while working with the Logbook in the DM Browser, allowing the analyst to have a complete overview of the results in a concise manner.

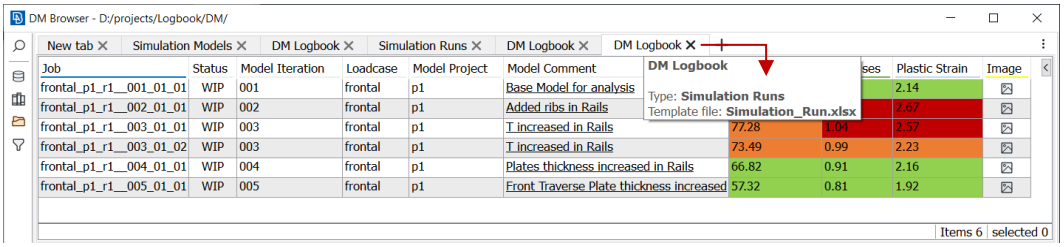
Multiple Logbook tabs

It is possible to open more than one Logbooks simultaneously in different tabs in the DM Browser. After applying a change in the Template file and the Logbook action is executed again, a new tab is appended in the DM Browser and the previous Logbook is not deleted. In this way, for the same or even different DM Object Type, the analysts can have multiple Logbooks depending on their needs.

An example is that for the same DM Object type and betaQL expression used, the analyst can have a Logbook for detailing report of results and another with fewer key results aimed at comparing different models.



A useful tooltip is popped up when hovering over the Tab of each DM Logbook with information about the template that is being used and the DM Object Type that is used upon.

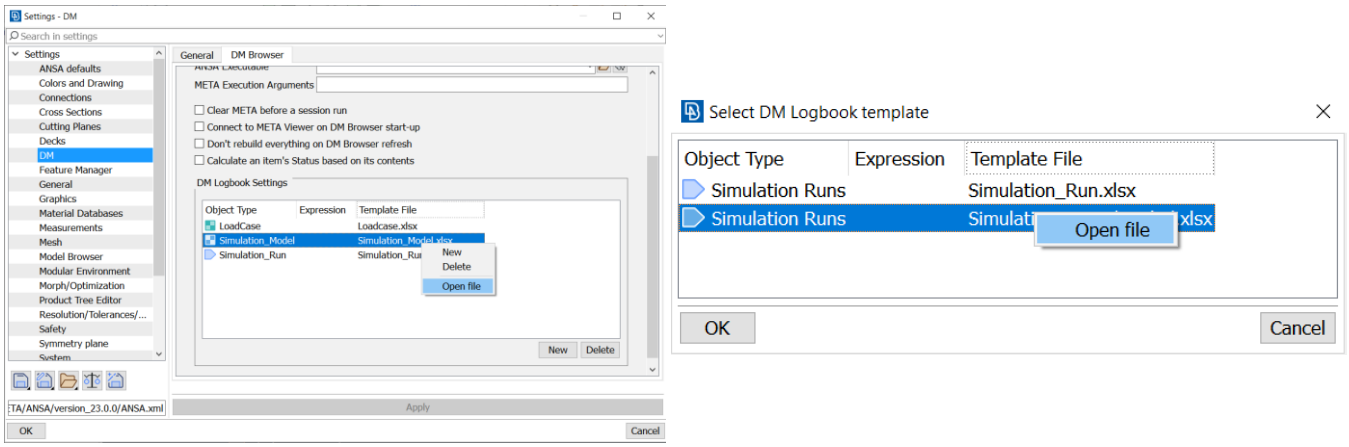


Sorting order

The objects listed in the Logbook can be sorted by any visible column, just by clicking on the column header, just as in any list in ANSA, META or KOMVOS.

Templates Editing

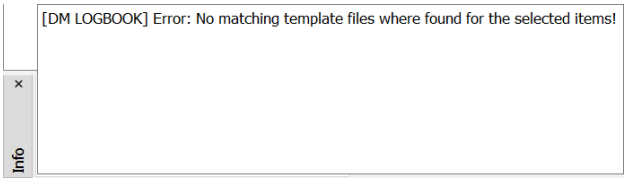
In the lists of the configured template files, either in the *Settings > DM > DM Browser > DM Logbook Settings* or in the Template file selection window in case of multiple matches, it is possible to Edit the Template file using the system default editor (e.g. Excel) by trigger the *Open file* option of the context menu.



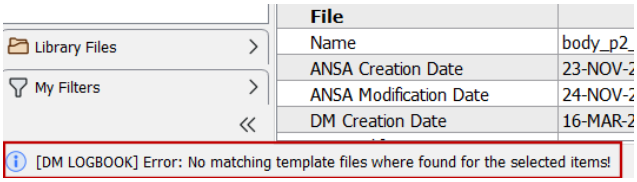
Missing Template

If no template file has been configured for the selected DM Objects, on **Reports > Logbook** a relevant error message is printed in the info window for ANSA & META and in the Info feed for KOMVOS.

ANSA & META

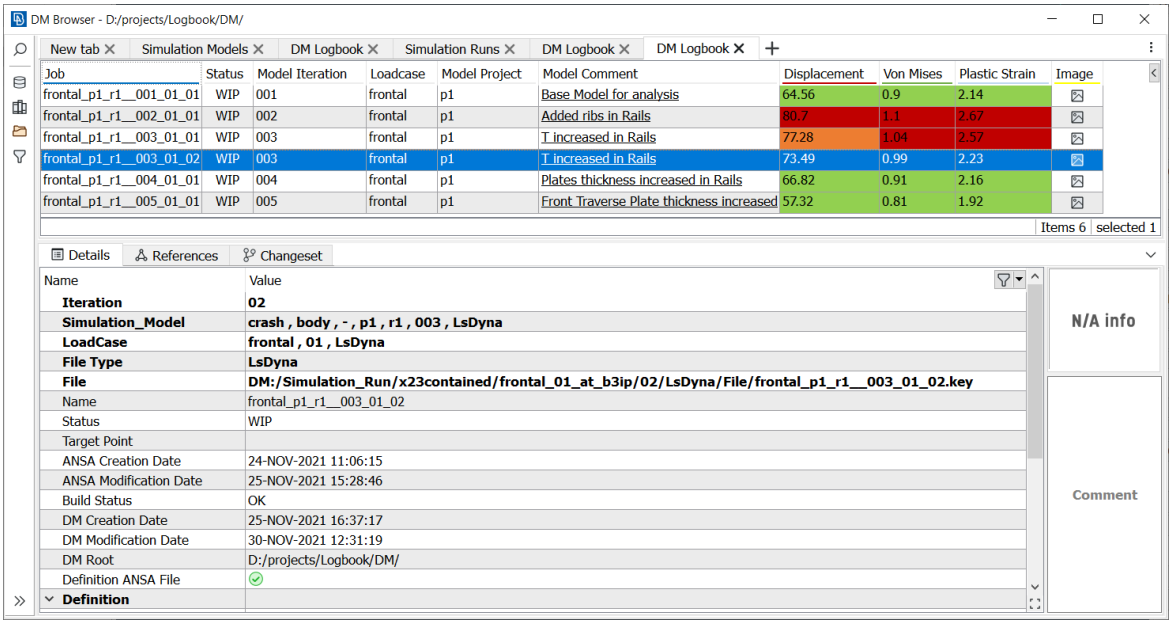


KOMVOS



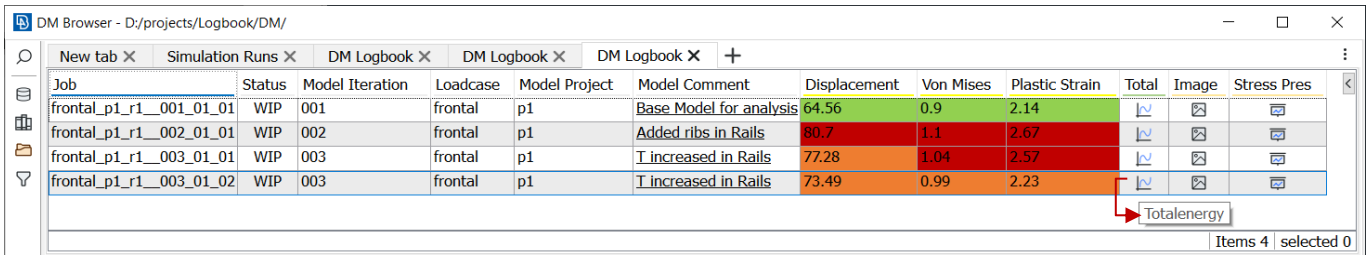
DMObject Details

The DM information for the DM Object is still available while in the Logbook. If a row is selected, the metadata of the respective DM Object (Details, References, etc.) can be viewed in the bottom tabs of the DM Browser.

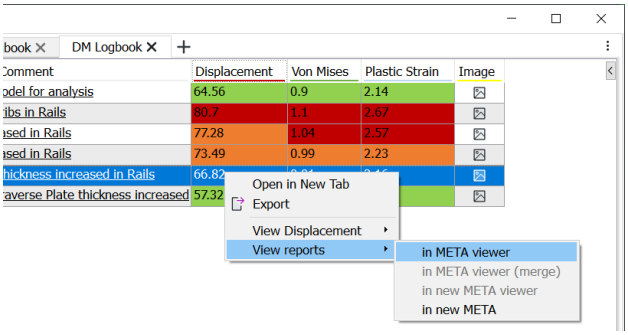


View of Reports

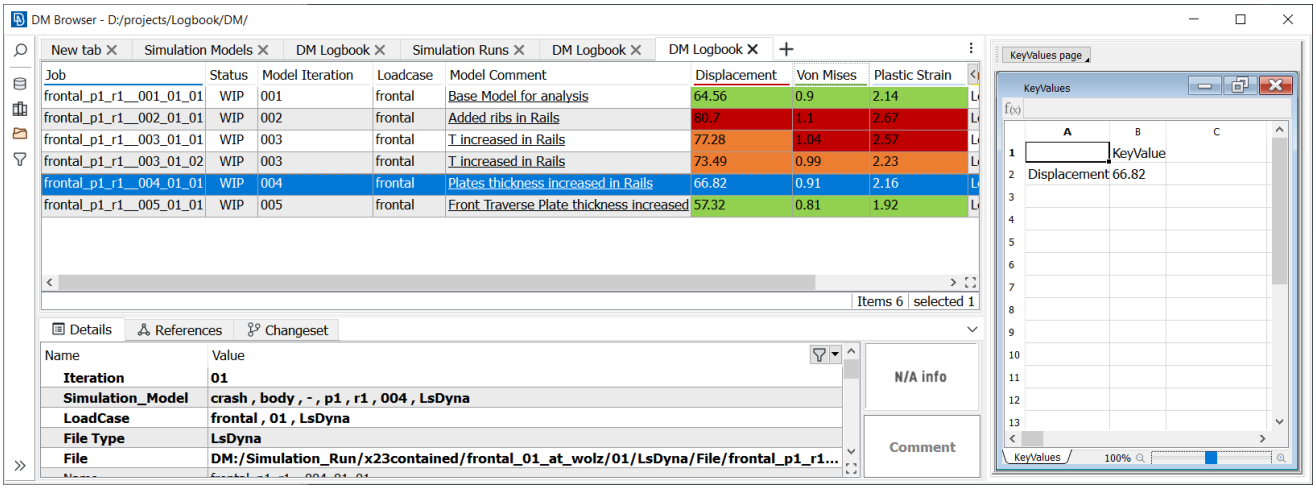
For “Key Values” Reports, the value is displayed at the respective cell. For non “Key Values” Reports, a matching icon with the type of the Report is displayed. By hovering over the icon, a tooltip with the Name of the Report is shown.



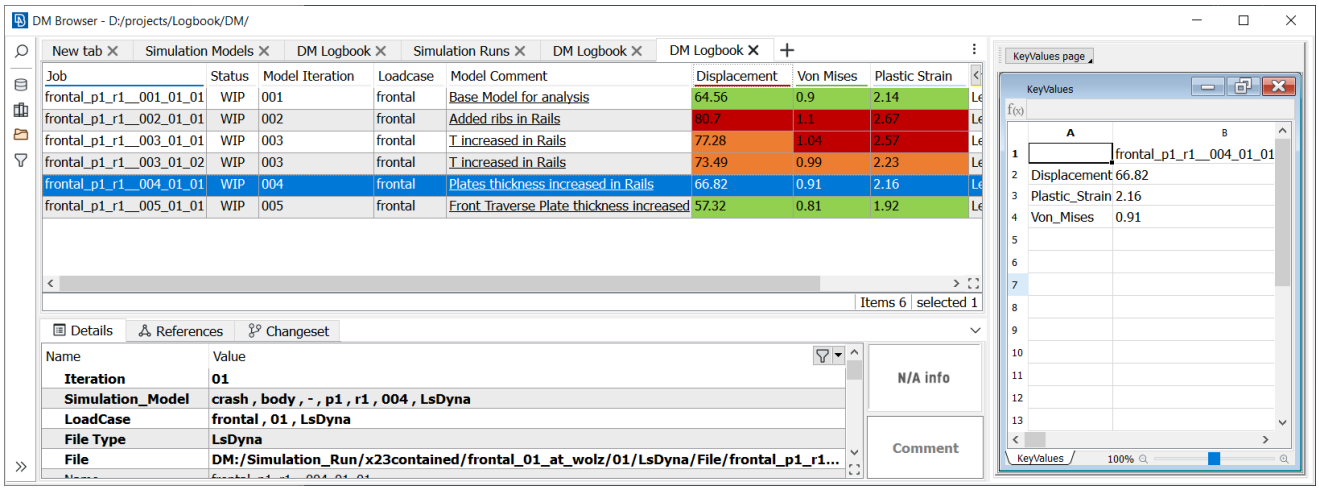
By selecting a row and opening the context menu on any of the cells that correspond to some Reports, the “Key Values” can be viewed in the META Viewer or in META. Two options are available: a) to view only the selected key value or b) all the report values of the selected DM Object.



a) View only the selected key value

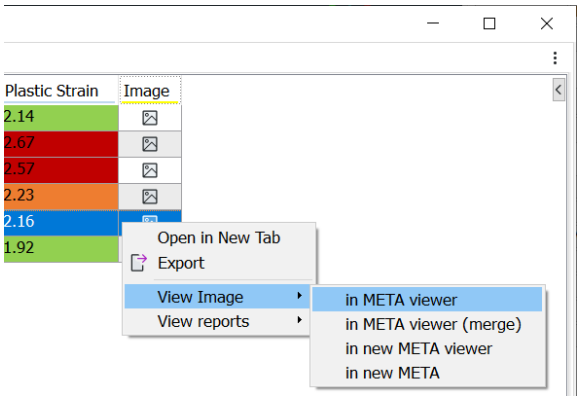


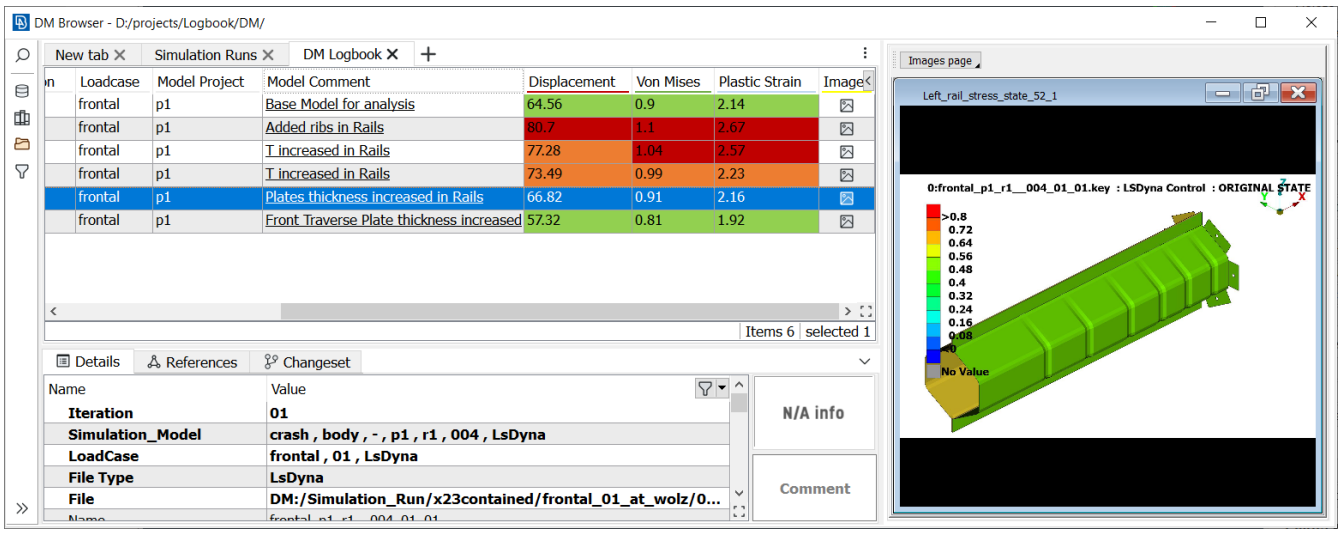
b) View all the report values of the selected DM Object



Report types other than Key Value (e.g., Images, Annotations), are displayed as plain text in the respective cells of the *Logbook*.

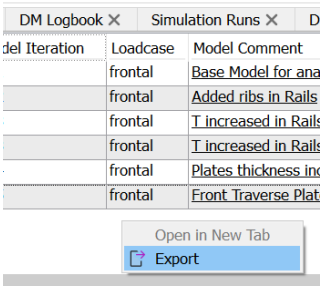
To view these types of Reports, the desired Reports can be selected directly from the respective cell in the *Logbook* and be opened in the viewer with the aid of the context menu. For example, to view a Report of type Image, the **View Image > in META Viewer** is available through the context menu.





Export Logbook data

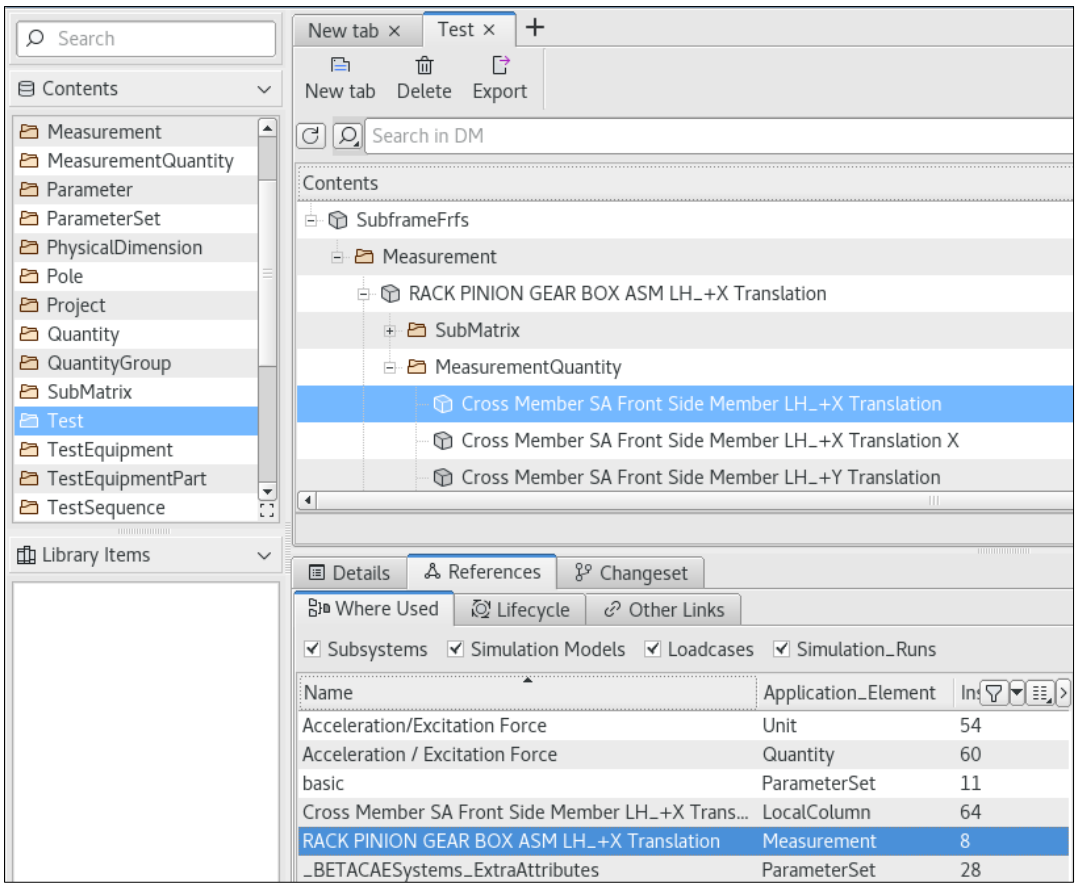
The results displayed in the Logbook can be exported in an Excel file which allows the user to share information with external users or further process the data. By opening the context menu inside the Logbook tab and selecting the option **Export**, the File Manager pops up and prompts the user to define the file name of the Excel file to be exported.



	C	D	E	F	G	H	I	J	K
1	Model Iteration	Loadcase	Model Project	Model Comment	Displacement	Von Mises	Plastic Strain	Image	
2	001	frontal	p1	Base Model for	64.56	0.9	2.14	Left_rail_str	
3	002	frontal	p1	Added ribs in	80.70	1.1	2.67	Left_rail_str	
4	003	frontal	p1	T increased in	77.28	1.04	2.57	Left_rail_str	
5	003	frontal	p1	T increased in	73.49	0.99	2.23	Left_rail_str	
6	004	frontal	p1	Plates	66.82	0.91	2.16	Left_rail_str	
7	005	frontal	p1	Front Traverse	57.32	0.81	1.92	Left_rail_str	
8									
9									
10									
11									
12									
13									

4.4. Test Results review

The SDM clients can connect to Test Results servers according to the ASAM-ODS standard in order for the simulation and test engineers to review Test Results like Channels, Photos, Movies, 3D Measurements, etc.



The key capabilities offered, include:

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

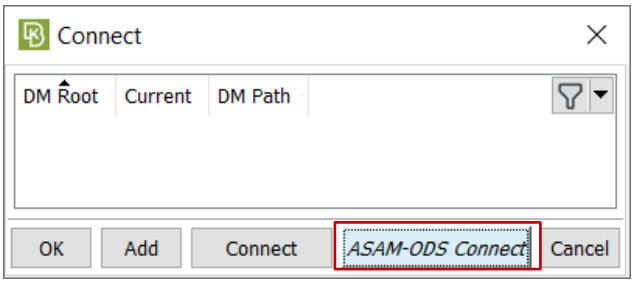
4.4.1. Connecting to a Test Results server

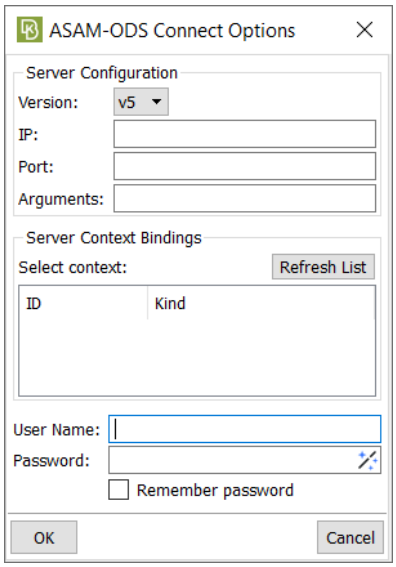
The data manager supports ASAM ODS v5 and v6 servers. KOMVOS, META and ANSA can connect to ASAM ODS servers in a way similar to how connections are established with other SDM back-ends.

Through KOMVOS, the **Login > Add/Connect** option can be selected through the main toolbar or the **Extras > Connect** option of the top menu-bar.

In ANSA and META the **Set DM Paths** option of the DM menu on the Lists toolbar can be used.

In the **Connect** window that pops-up, pressing the **ASAM-ODS Connect** button will open the **ASAM-ODS Connect Options** window.





In the **ASAM-ODS Connect Options**, the Version, IP, Port, any necessary Arguments, User Name and Password fields should be filled and finally the OK button should be pressed to add the selected server in the **Connect** Window.

When the connection to the server is established successfully, the server path appears in the Connect window and the user must confirm and close the window by pressing OK

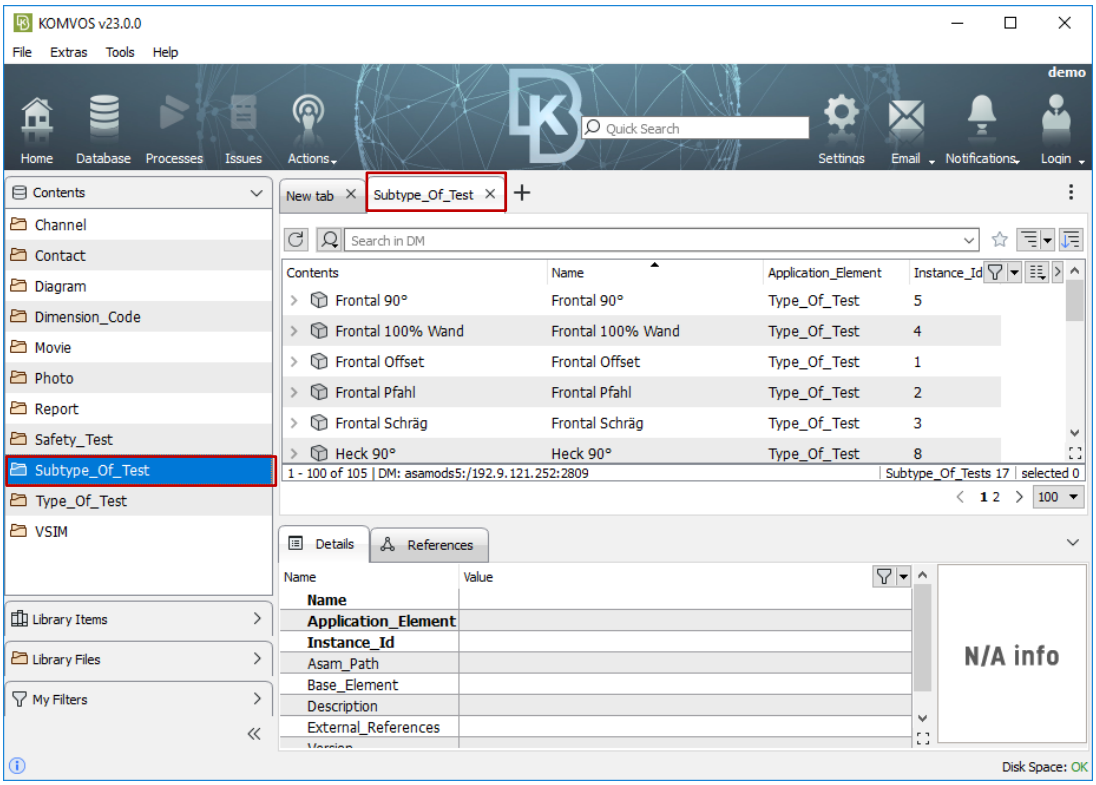
If the connection to the server fails, a relative message is displayed in the Info Feed

4.4.2. Navigation

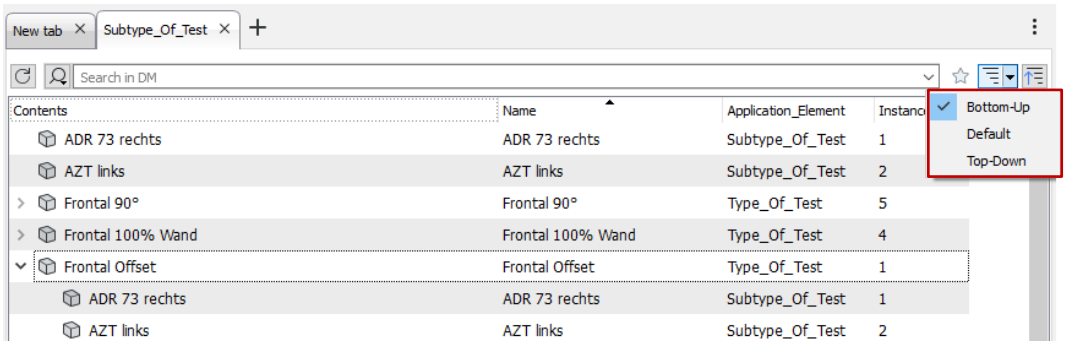
4.4.2.1. Viewing Application Instances

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

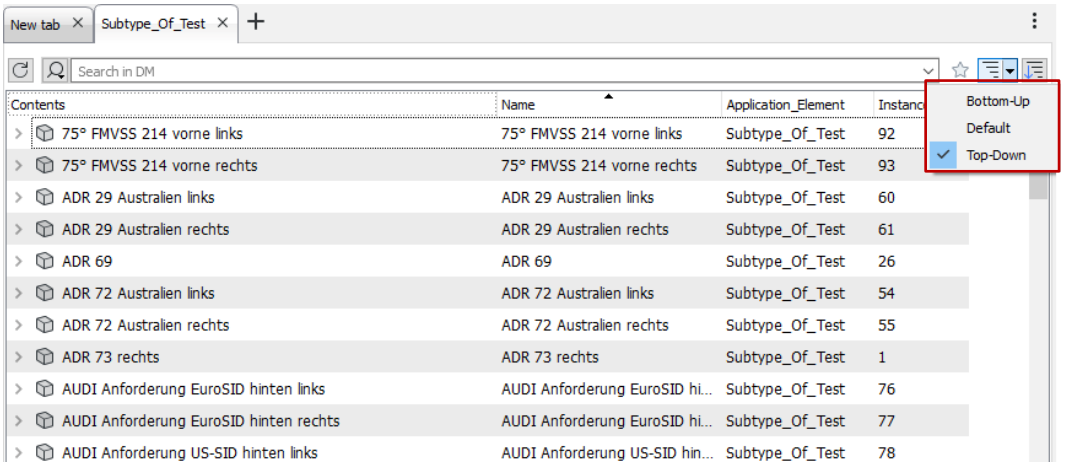
With double click on a specific application element, a new tab is created. The tab contains all the application instances of this Application Element.



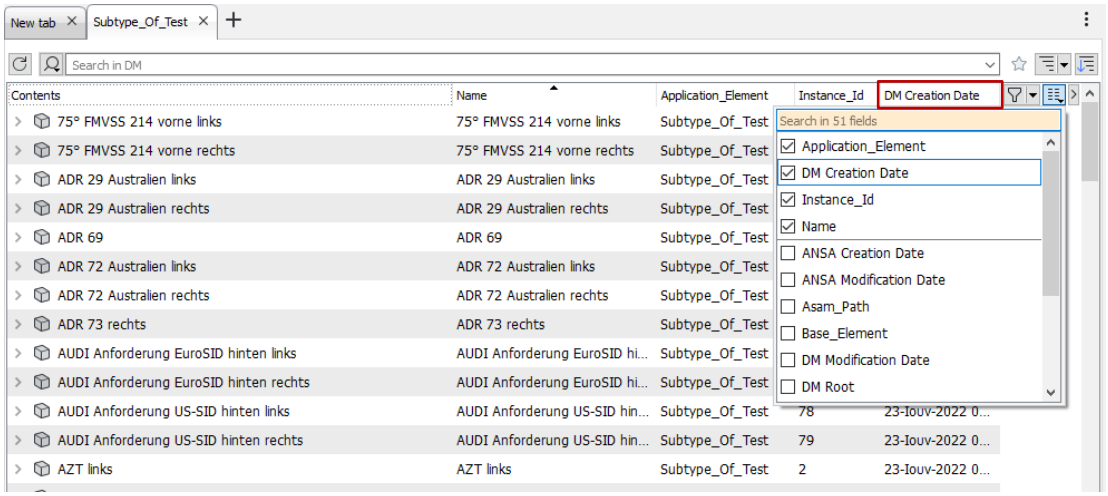
A tree mode view can be also used through the *Change View* button. There is the option to select *Bottom-Up* tree view, where all the high hierarchy related application instances (parent elements) are depicted as well.



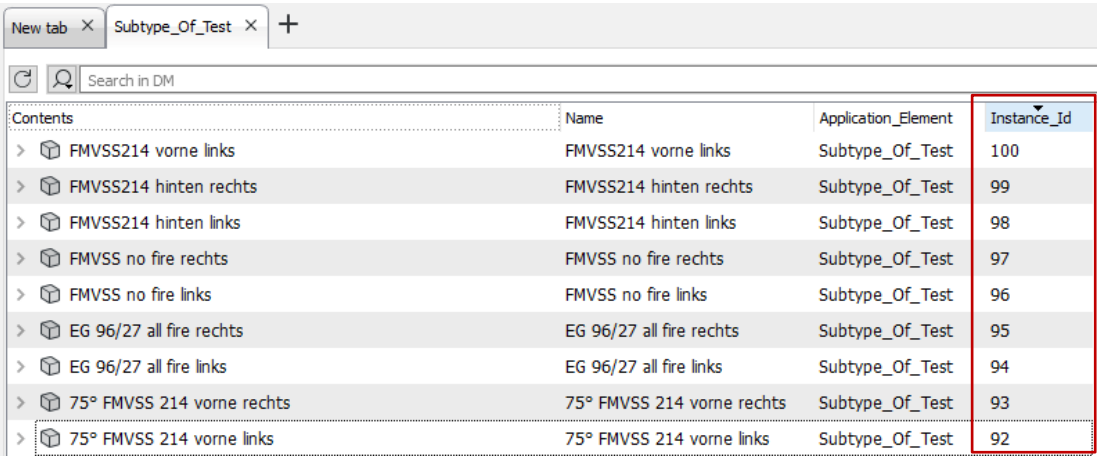
By selecting the *Top-Down* option, all the low hierarchy related instances (children elements) are depicted in the tree view as follows:



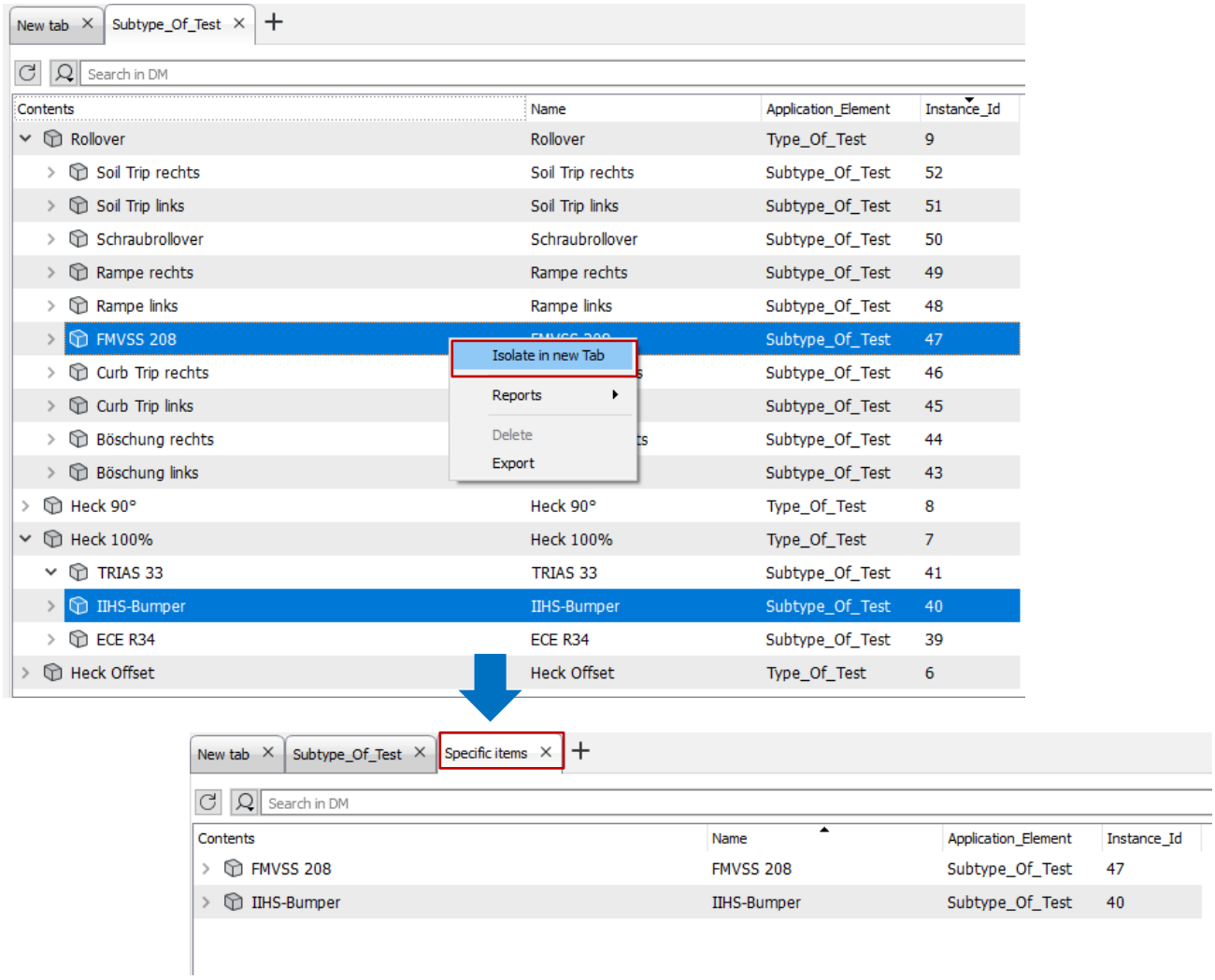
Information about the name, application element type and instance id can be found in the respective columns *Name*, *Application_Element* and *Instance_Id* which are shown by default. By pressing *the Show/Hide Columns* button, existing columns can be hidden, or extra columns can be added. Enable/Disable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



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



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



4.4.2.2. Filtering to fetch Application Instances

It is possible to apply filtering in order to fetch instances of application elements. Two types of filters are available. The *Basic Filter*, where complex queries can be defined in a GUI assisted mode, and the *Advanced mode*, which allows the definition of even more advanced queries, using the BETA Query Language (BETA QL).

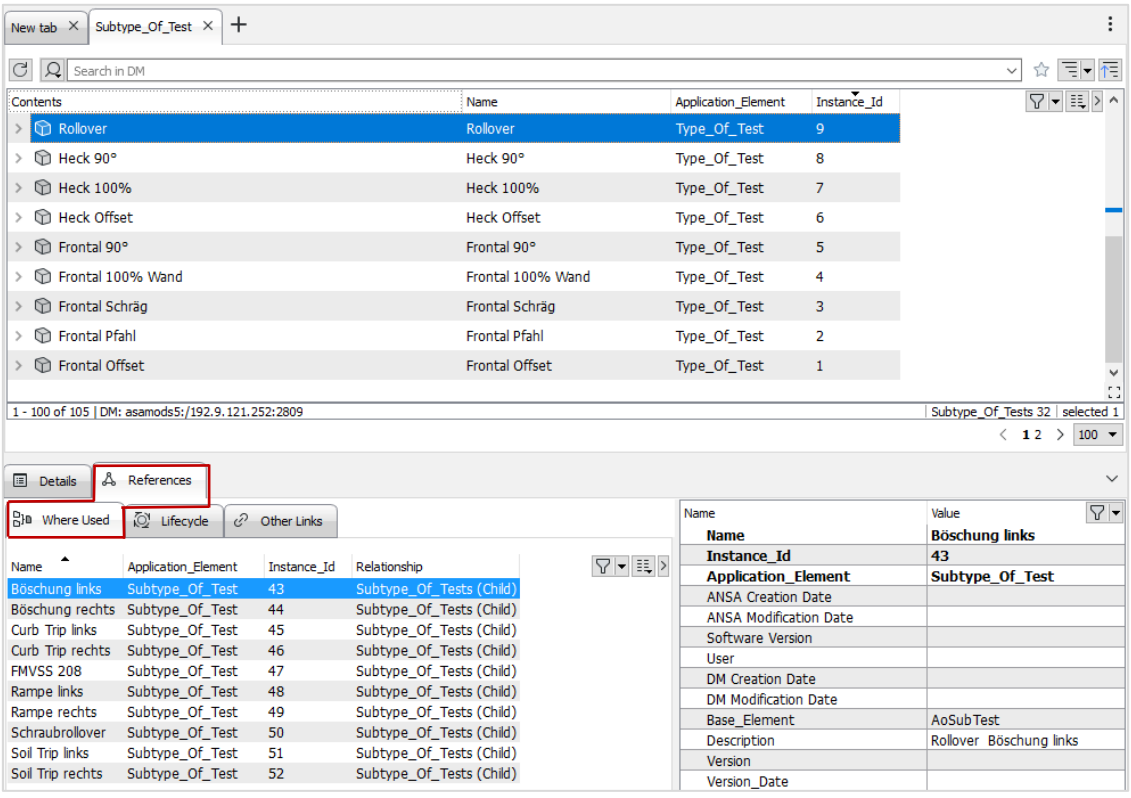
In the Basic mode, the available application elements can be added through the respective button, as filtering forms. When pressed, the drop-down list is filled with the available values for the specific element. By activating the checkbox of the desired values and pressing OK, the query expression is modified with the specified values.

In the Advanced mode, BETA QL can be used, providing a dynamic typing of filtering expressions. More information on the filtering can be found in the chapter 3.5.2 of this document.

4.4.2.3. Viewing attributes/relations of Application Instances

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

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

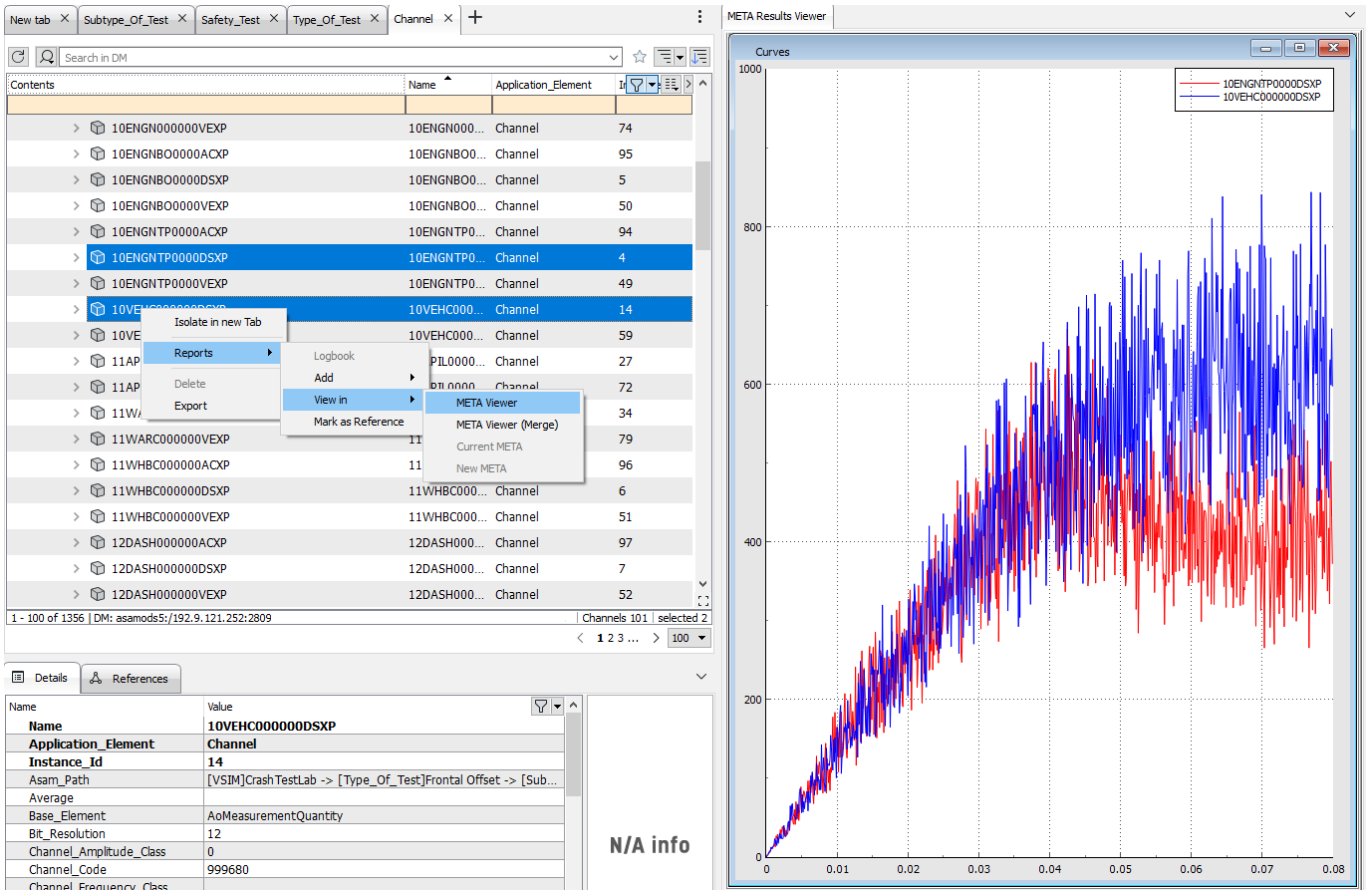


4.4.3. Viewing Application Instances in the embedded META Viewer

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

selection is also available for the same type of results. As an example, two or more selected curves can be sent to META Viewer. The Viewer will superimpose the curves, offering a comprehensive view to the analyst.

After sending some curves to the META Viewer, the **META Viewer (Merge)** option gives the possibility to append new curve data to the existing plot.



4.4.4. Viewing Application Instances in META

In a similar fashion, data from servers according to the ASAM-ODS standard can be visualized viewed in META. When using the DM Browser tool from META, Application Instances like Channels/2D Measurements, 3D Measurements, Photos and Movies can be imported in META by selecting the listed item and triggering through the context menu, the action **View in > Current META**. 3D Measurements, Photos and Movies are loaded in 3d windows, Channels/2D Measurements in 2d windows.

It is also possible to load an Application Instance using META session commands as described in paragraph 4.5.1.

4.5. Integration with META

META's integration with the SDM system enables simulation and test engineers to interact with the data storage effortlessly, hiding the complexity related to the storage and retrieval of data, so that they can focus more on their analysis tasks.

First, as mentioned in previous paragraphs, the built-in interfaces with SDM systems and Test Results servers enable the browsing of Simulation and Test Results directly through the META DM Browser.

Second, the same interfaces enable simulation engineers to upload data to the SDM system directly during post-processing, without the need to first write key-results like images, videos, 2d plots, etc. in temporary folders and then upload them to the SDM back-end at a second step. Saving data from META to the SDM back-end is direct (see paragraph 4.5.1).

Similarly, during the loading of data from the SDM back-end or from a Test Results server, the engineers do not need to log in to the SDM system or Test Results server client application, find the data they need to use and export them in order to import them in META at a second step. Loading data in the running META is direct (see paragraph 4.5.2).

Last, META's integration with the SDM system enables its direct launch from within the data browsing tools in KOMVOS or ANSA in order to record a session that will be stored in the SDM library or to produce data through a live post-processing session that will be uploaded to the SDM system automatically on META quit (see paragraph 4.5.3).

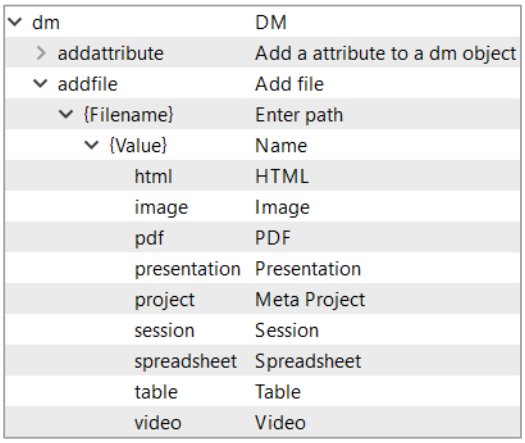
4.5.1. Adding reports through session commands

META supports a series of commands to communicate with DM in order to save various reports. These commands are found in the Commands list (Tools>Commands) under the category ***dm***. The most important commands are:

- `setcurid`
- `addfile`
- `addmetaobject`

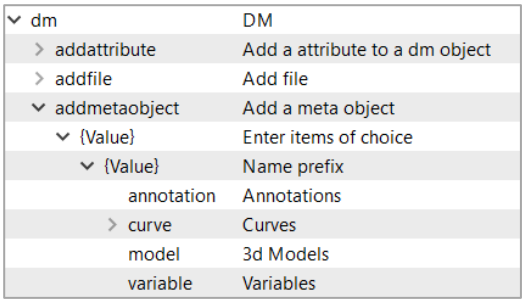
Name	Value
Simulation_Model	**VENZA , A00 , Crash , FULL_VEHICLE , 0 , LsDyna**
LoadCase	**Crash , ODB**
Name	**ODB_FULL_VEHICLE_VENZA_A00_CRASH_002**
ANSA Creation Date	26-MAY-2017 22:33:09
ANSA Modification Date	26-MAY-2017 22:33:09
User	demo
Id	1197
Software Version	
Status	WIP
Build Status	
DM Root	C:/Users/myusername/Desktop/ANSA DM TRAINING/...
Target Point	
DM Creation Date	26-MAY-2017 22:33:09
DM Modification Date	26-MAY-2017 22:33:09
Discipline	Crash
Main_File	DM:/Simulation_Model/VENZA/A00/Crash/FULL_VEHICL...
Project	VENZA

The **`setcurid`** is the command that informs META for the DM Object id of the Simulation Run (or Loadcase, or Simulation Model) under which the Reports will be stored.



dm	DM
addattribute	Add a attribute to a dm object
addfile	Add file
{Filename}	Enter path
{Value}	Name
html	HTML
image	Image
pdf	PDF
presentation	Presentation
project	Meta Project
session	Session
spreadsheet	Spreadsheet
table	Table
video	Video

The **`addfile`** is used to store images, presentations, projects, spreadsheets, tables and videos. The way these data are stored is indirect, which means that first they must be saved on the disk and then they must be saved in DM



dm	DM
addattribute	Add a attribute to a dm object
addfile	Add file
addmetaobject	Add a meta object
{Value}	Enter items of choice
{Value}	Name prefix
annotation	Annotations
curve	Curves
model	3d Models
variable	Variables

The **`addmetaobject`** is used to store curves, models and variables. These data are saved directly in DM.

Example 1: Store data through the addfile session command

A presentation is usually a characteristic example of a Report that needs to be saved in DM. Reports of this type are saved in DM in two steps: First they are saved in the file system and then attached to the Simulation Run through the addfile function. In the image below the syntax of this command is shown.

```
dm addfile "D:/Work/results_temp/my_presentation.pptx" my_presentation presentation
```

Example 2: Store data through the addmetaobject session command.

Annotation text is a typical example of a Report item that is saved directly in DM. The function to use is the addmetaobject which requires the id of the annotation.

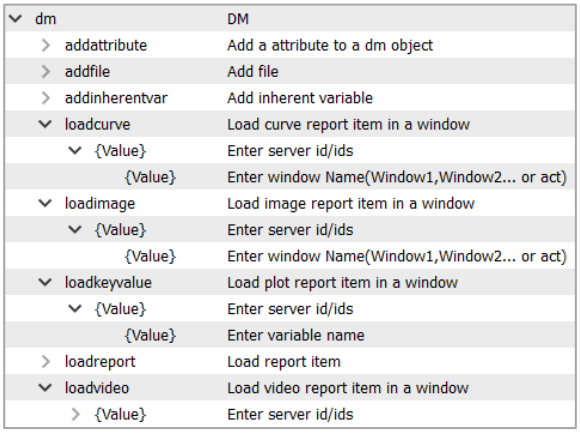
```
dm addmetaobject 1 ymax annotation
```

4.5.2. Loading Simulation Reports and Test Results in META through session commands

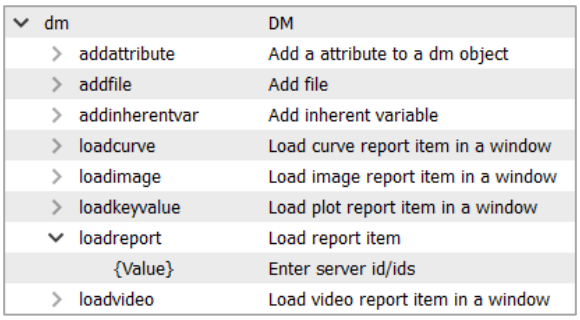
META supports a series of commands to communicate with DM and load simulation results or tests results Application Instances (from ASAM-ODS compliant servers). The commands are the same for both cases (simulation and tests results). These commands are found in the Commands list (Tools>Commands) under the category ***dm***. The most important commands are:

- `loadcurve`
- `loadimage`
- `loadkeyvalue`

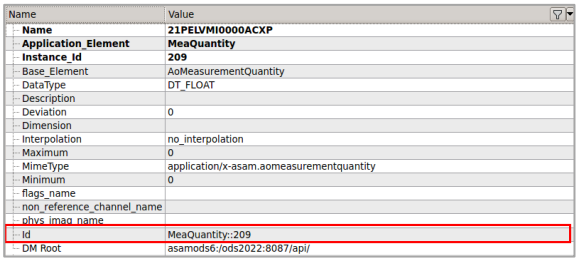
- `loadreport`
- `loadvideo`



The **`loadcurve/loadimage/loadvideo`** are used to load simulation Reports of type Curve, Image and Video to a given window.
The same commands can be used to load test Application Instances like Channels/Measurements, Photos and Movies.



The **`loadreport`** is used to load simulation Reports of any type. It is mostly used to load Reports types that are not loaded in a window like KeyValues, Presentations, Tables, Spreadsheet, etc.



`Id` is the Object id of the simulation Report or the test Application Instance item to be loaded. A series of comma separated ids can be given to be loaded together.

Example 1: Load Simulation Report curves in a META 2d window

To load some Reports of type Curve with server ids **1834,1835,1989** in window **Window1** in META the following command is needed:

```
dm loadcurve 1834,1835,1989 "Window1"
```

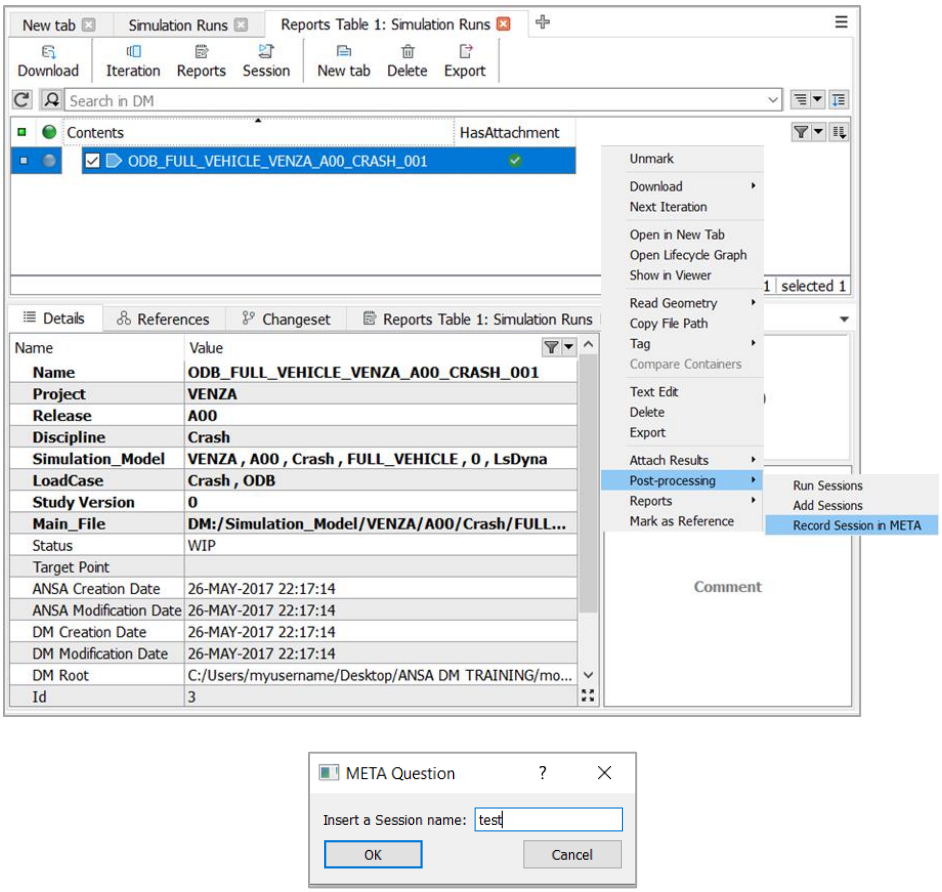
Example 2: Load an ASAM-ODS MeaQuantity in a META 2d window.

To load some Application Instances of Application Element Channel with server ids **MeaQuantity::465, MeaQuantity::2435, MeaQuantity::2476** from an ASAM-ODS server in window **Window1** in META the following command is needed:

```
dm loadcurve MeaQuantity:465,MeaQuantity:2435,MeaQuantity:2476 "Window1"
```

4.5.3. Recording new session files

The recording of a session file may be initiated through the **Post-processing>Record session in META** of the context menu of a Simulation Run. This function asks the session name and then launches META having the selected Simulation Run as the active one.

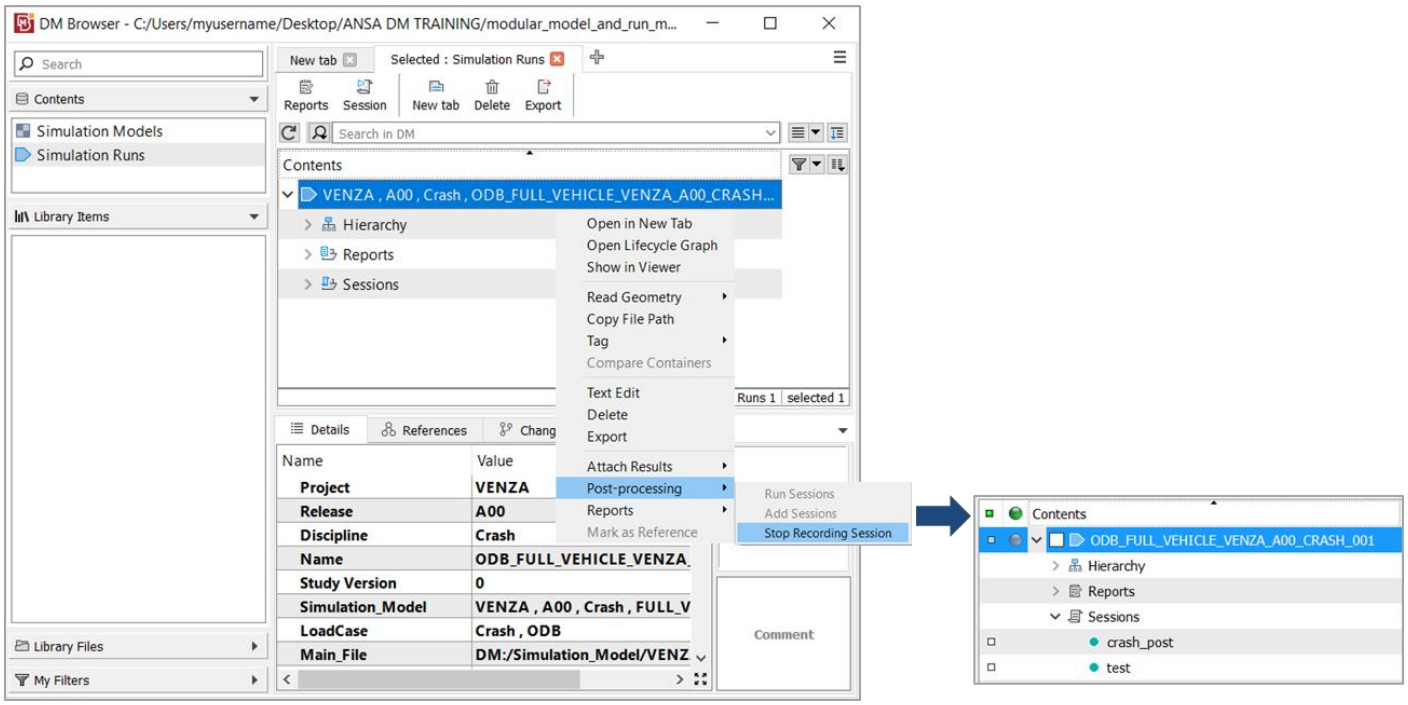


Alternatively, the recording of the session file can be performed in META. In that case, the command “dm setcurid” must be used to specify the id of the Simulation Run, where the results will be attached to. The id of the Simulation Run is the “Id” attribute written in the Details tab of the Simulation Run.

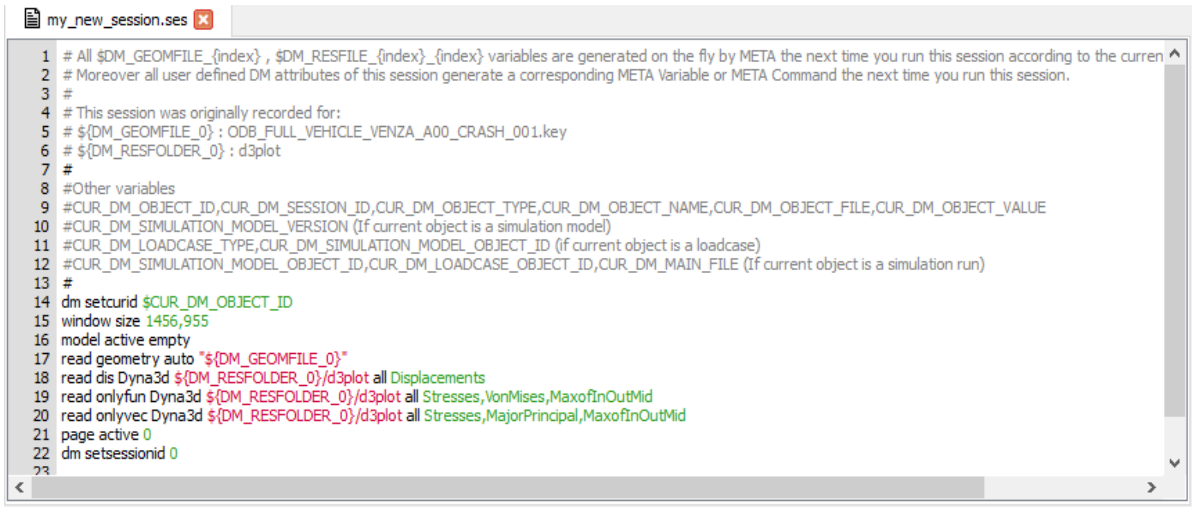
Provided that the raw results have been added in DM, see paragraph 4.1.1, the geometry of the model can be loaded in META through the *Read geometry>in current META* function of the context menu of the Simulation Run. Assuming again that the import of geometry was made through DM, the results file can also be identified automatically through the linked dm folder.

At any time, the recording of the session can stop through the *Session>Stop recording* session of the context menu of the Simulation Run.

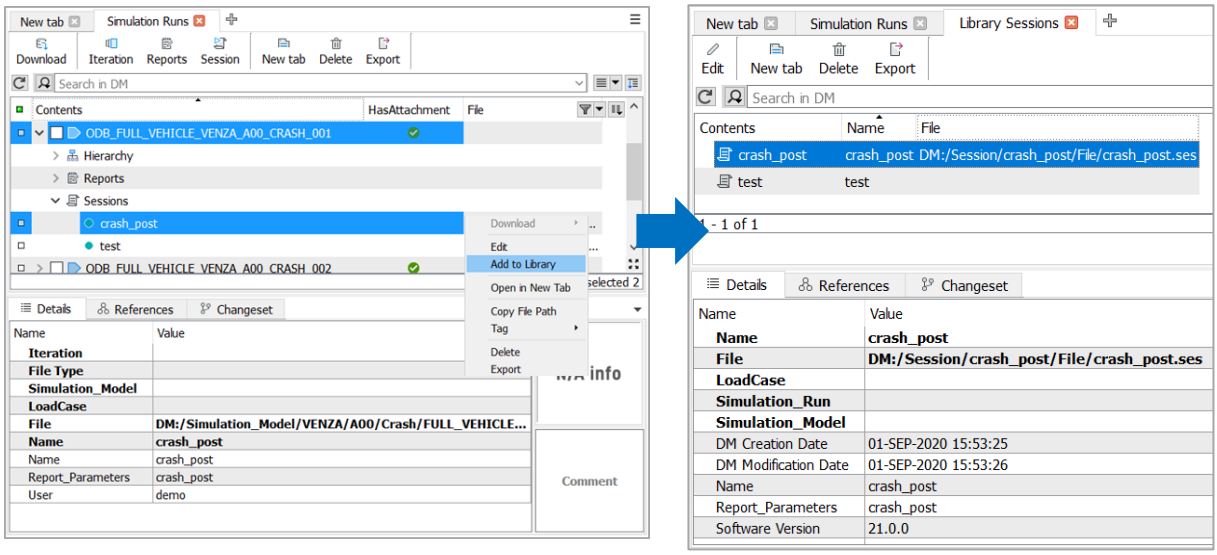
The new session is automatically saved under the “Sessions” container of the Simulation Run.



The recorded saved session is parametrized in terms of the id of the Simulation Run, the geometry and the results in order to be reused in the future. In the context menu of the session pick the *Edit* function to view the written commands

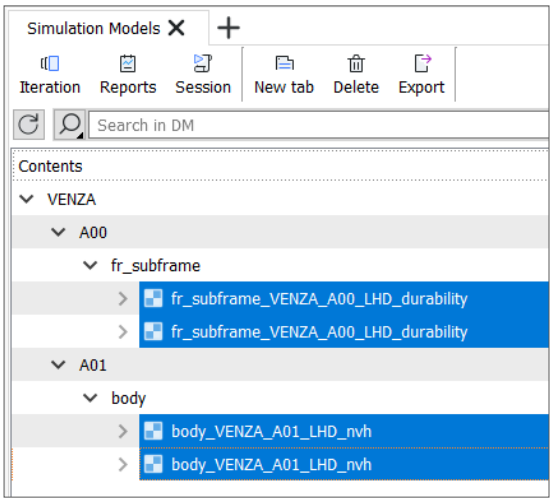


A run-specific session can be added in the library for general use using the function *Add session to library* of the context menu of a session

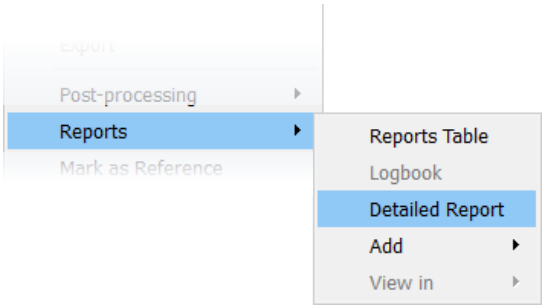


4.6. Creating Detailed Reports

It is possible to set-up a customized META DB-based presentation of results attached under the Runs of different Simulation Models in the ANSA/META DM Browser with the aid of a custom META script.

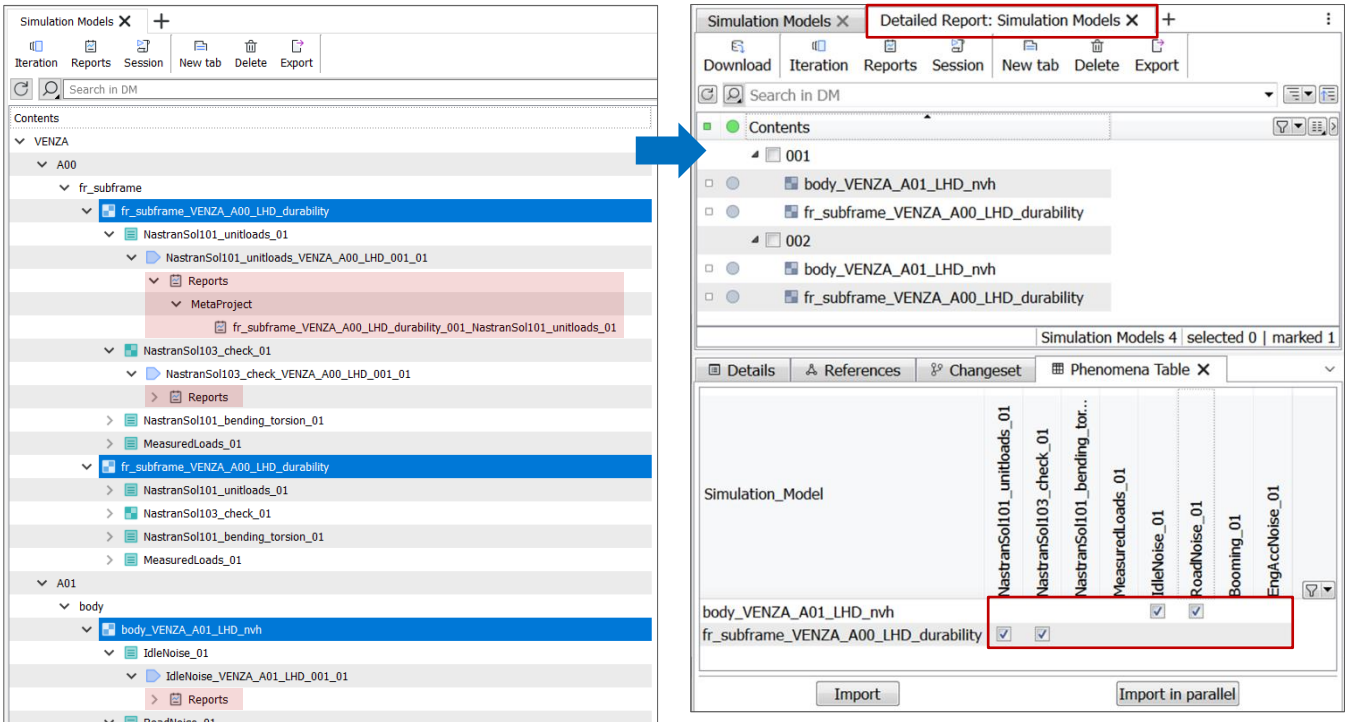


After proper configuration, in the DM Browser the user can select one or multiple Simulation Models and from their context menu activate the option **Reports > Detailed report**.

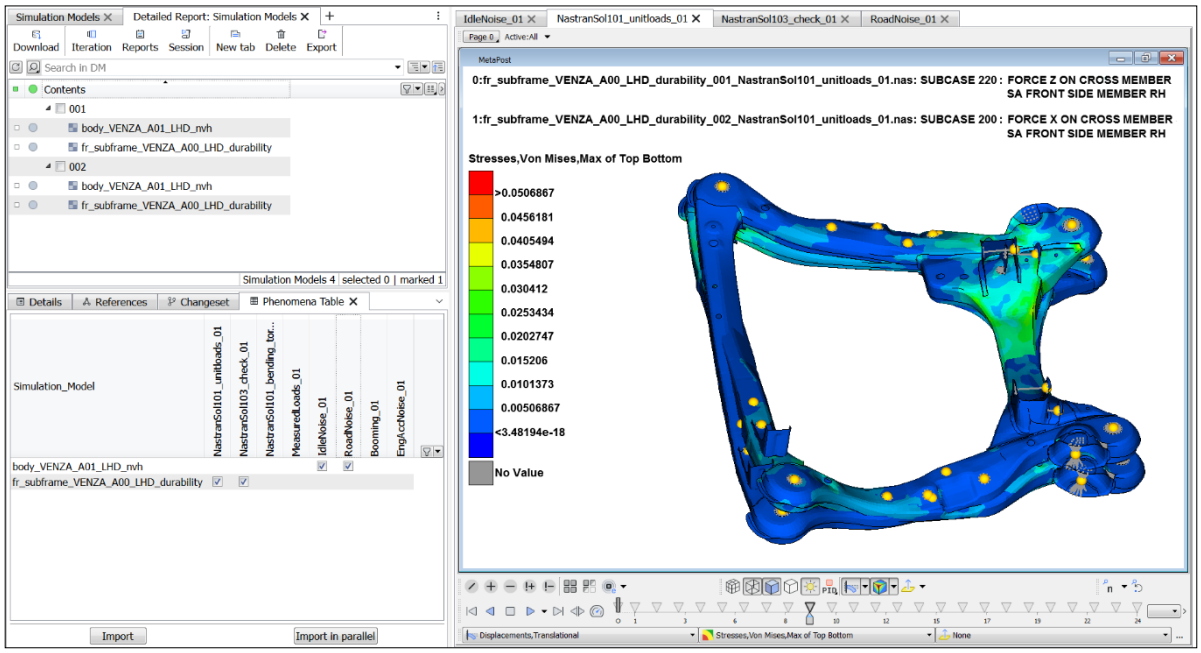


This action opens the new top tab named *Detailed Report: Simulation Models*, that lists the selected Simulation Models. At the same time, the new bottom tab named *Phenomena Table* is created. This tab contains a table with the selected Simulation Models as rows and their contained Loacases as columns.

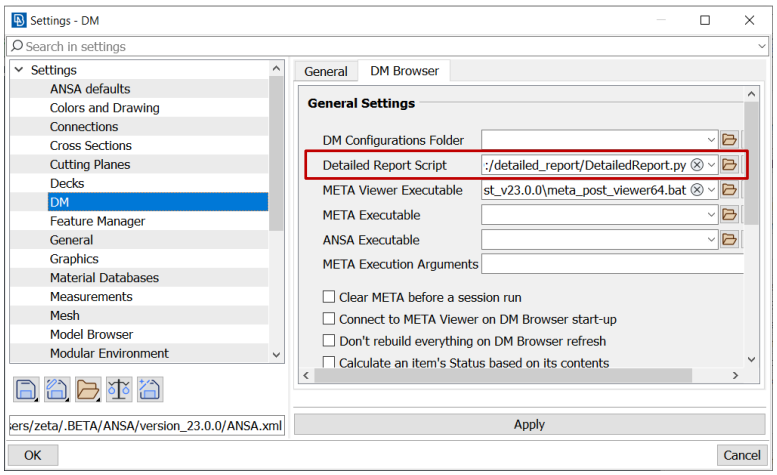
The Simulation Models - Loadcase combinations for which a Report of type MetaProject or metadb is available under their Runs, can participate in the Detailed Report. Thus, for these items a check box is displayed in the corresponding cell of the table.



By selecting one or more checkboxes and pressing the button **Import**, the META Results Viewer opens with one tab for each activated Loadcase.



The button **Import in parallel** will open each viewer in a different processing thread.



The script is defined in **Tools > Settings > DM > DM Browser > Detailed Report Script**.

The DM Browser expects to find within the script a function with name `create_detailed_report` with the following prototype:

```
create_detailed_report(server_ids)
```

The argument passed by the DM Browser to the script is a list with the Simulation Run server ids.

The script runs in each META Viewer in order to adjust the presentation of each corresponding META DB.

5. Customization

Customization is a key requirement for an SDM system, as it enables its adaptation to the organizational needs of each engineering team and eases its adoption as an everyday tool by all simulation stakeholders. There are two main areas for customization of an SDM system: The data model, which determines the structure of data and their relationships, and the data views, which control how data will be presented in the various tools in the SDM Client applications.

Both the file- and the SPDRM-based SDM solutions of BETA are highly customizable in terms of both the data model and the graphical user interface. Paragraphs 5.1 and 5.2 below describe the customization possibilities in these two areas.

5.1. Data model

The data model holds all the information regarding the structure and relationships of DM Objects. In BETA's SDM solutions, the part of the data model that describes which are the supported data objects and the relationships between them, facilitates the general requirements of Modular Model Build, is hard-coded and cannot be modified. However, there's a fully customizable layer that enables the adaptation of the hard-coded data model to the organizational needs and the terminology used in different engineering teams. In this guide, the term data model is used interchangeably with the term DM Schema to refer to this configurable layer.

BETA's SDM solutions come with a default data model pre-configured to cover the data organization needs of engineering teams in the automotive industry. Thus, the terms used to refer to Library Data, Loadcases, etc. resonate better to automotive simulation engineers. However, as it will be thoroughly explained in the paragraphs below, the default data model can be easily modified and adapted to the needs of any engineering team.

The table below describes the key characteristics of the default data model:

Low-level Model Data	
Parts / Groups	-
Subsystems	contain Parts
Subsystem Groups	contain Subsystems
Simulation Data	
Simulation Model	contains Subsystems and Library Items
Loadcase	contains Subsystems and Library Items
Simulation Run	contains one Loadcase, one Simulation Model and one or more Library Items
Report	**-**
Low-level Library Data	
Library File	
Loadcase File	
Loadcase Header	
Display style	
Session	
Higher-level Library Data	
Loadcase Template	contains Subsystems and Library Items
Target Points	contains Subsystems and Library Items
Simulation Configuration Table	contains Subsystems, Library Items, Simulation Models, Loadcases, Runs

Geometric Feature		
Fastener		
Stamp		
Default Items		
Changeset	Optimization Study	DOE Study
Predictor	Modular Environment Profile	Build Action/Process/Setup

With the customization of the data model, organizations can:

- Define custom attributes of the DM entities
- Add new types of Library Data
- Restrict the accepted values of attributes by defining pre-defined lists of values
- Define rules for the auto-generation of DM Object Names and filenames file-based DMs

The customization of the data model when working with file-based DMs, is done with a file named `dm_structure.xml`. This file is expected to be found within the DM root folder. When a new DM root folder is created (**DM>Set DM Paths>Add** in ANSA/META or **Login>Add** in KOMVOS), this file, containing the default configuration, is copied from the *config* folder to the new DM folder.

For SPDRM-based DMs, the file responsible for the customization of the data model is named `dm_structure_TBM.xml`. This file resides in the SPDRM Installation directory under:

`/server/Wildfly/standalone/configuration/``dm_structure_TBM.xml`.

The DM administrator can configure the data model xml file using one of the following alternatives:

- The *DM Schema Editor*, a GUI tool within ANSA/META or KOMVOS
- A text editor, for manual editing of the xml file

The *DM Schema Editor* offers a user-friendly interface for the quick and easy customization of the data model (*see paragraph 5.1.1*). There are certain advanced customization capabilities that are not supported by the *DM Schema Editor* and therefore, the xml file generated by the tool may require some manual text editing *(see paragraph 5.1.2)*.

5.1.1. DM Schema Editor

The DM Schema Editor is used as an XML writer and Data Model editor and is available within ANSA/META or KOMVOS. This tool aims to help the DM administrator or the project leader to customize the DM environment so that the names of attributes and the overall description of data object fit well to the needs of each CAE team.

5.1.1.1. Customization capabilities

Through the DM Schema Editor, it is possible to:

- Modify attribute names, e.g. the CAE Release may be called just "Release" by some team, "Milestone" by another or even "Main Event" by some.
- Modify the accepted values of attributes, e.g. accepted values for Module Ids, Projects, Representations.
- Define whether it should be possible to leave an attribute blank
- Define the default value of attributes
- Define the generation rules that should be used to compose the Name of DM Objects based on the values of other attributes
- Define new Library Item types

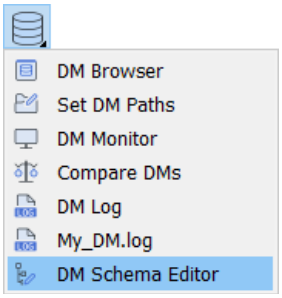
! Important note:

Modifications that must be avoided, because they may disrupt the smooth operation of the system, are:

- Renaming and changing the type of the hard-coded attributes: Module Id, Representation, Status, File Type, Status, Name

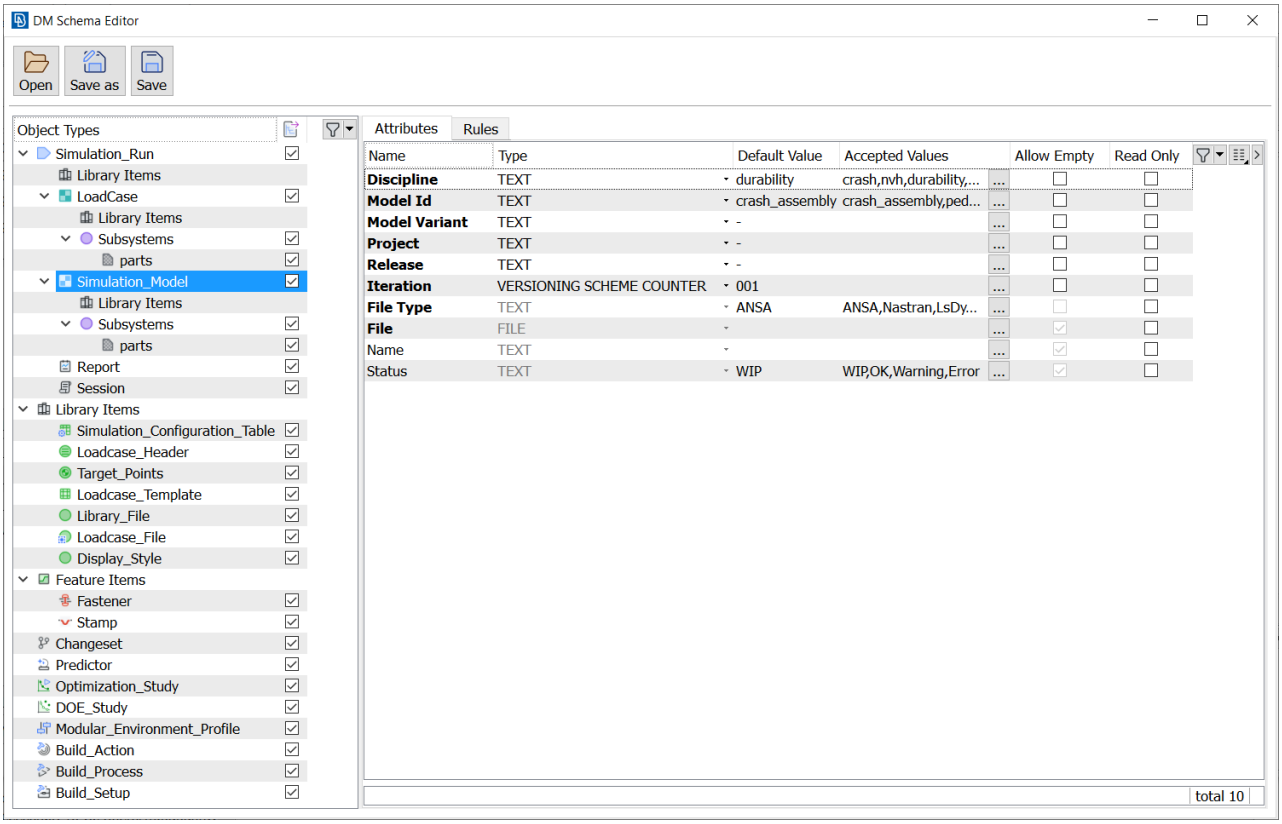
- Defining attributes with the reserved names Version, Study Version and assigning them any type other than the default i.e. TEXT and STUDY VERSION respectively
- For Simulation Models, Loadcases, Simulation Runs and any other type that has a primary attribute of type FILE, this is defined by default in the last position of the list of primary attributes in the `dm_structure` xml file. No other new primary attribute must be manually added after the FILE attribute.

5.1.1.2. Graphical User Interface

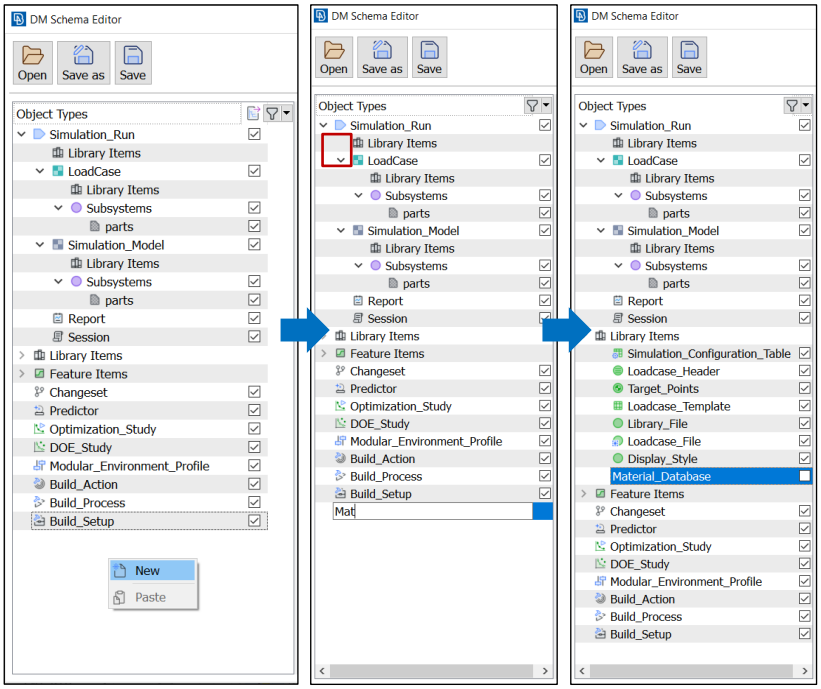


In ANSA, the *DM Schema Editor* is accessible through **Lists>DM>DM Schema Editor**, or through the respective option in the DM toolbar. To access DM Schema Editor through the DM Browser of ANSA/META the option **Utilities>DM Schema Editor** of the main menu can be used. In KOMVOS, the tool is accessed through **Tools>DM Schema Editor**.

The left pane lists the available default DM Object Types. The checkbox next to the DM Object Type indicates whether this item will be written out during save. By selecting a DM Object Type on the left, its attributes appear in the right pane of the window. The properties of the object (i.e. identifying keys, also referred to as primary attributes) appear in bold font style. For each of the existing attributes the user can customize several characteristics, such as its type, its default value, the closed list of accepted values if it won't be a free-text field, etc. This customization can be done with direct in-list editing or through the attribute's edit card that pops-up with double click.



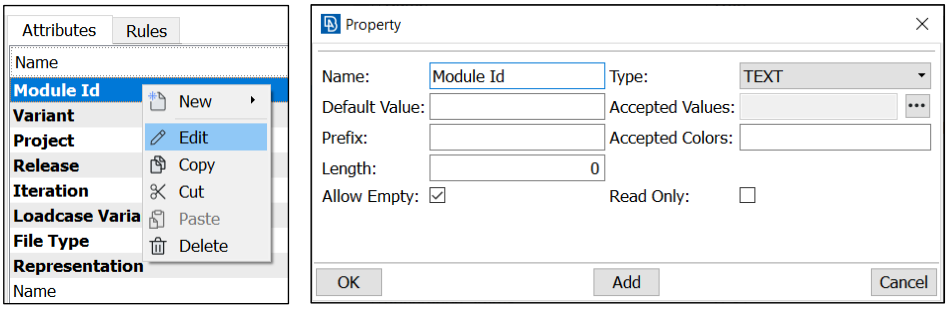
To define a new Library Item, right click anywhere in the left pane of the *DM Schema Editor* and select the option **New** of the context menu. Type the name of the new item and press Enter.



Once the new item is created, it can be configured as described in the paragraphs above.

Note that the **Output** column controls with the aid of the checkboxes which DM Object Types will be written in the data model xml file during save. New created items are unchecked by default.

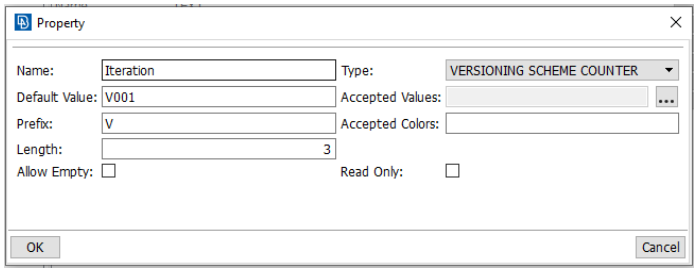
5.1.1.3. Adding properties/attributes



Double click or select the option **Edit** from the context menu of an attribute to edit its characteristics. The *Property/Attribute* window appears. Here one can configure the property/attribute according to needs.

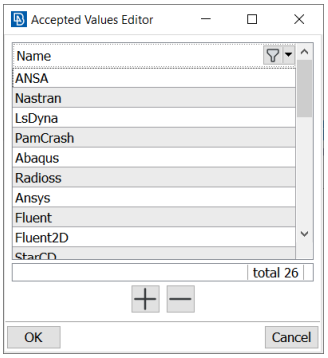
A property/attribute of a DM Object can be of various types. The supported types that are available in the **Type** field, are shown in the table below:

TEXT	Accepts any kind of text
BOOLEAN	Accepts True/False values
INTEGER	Accepts integer values
DOUBLE	Accepts double values
FILE	Used to attach a file to the DM Object
LINK FILE	Used to link a file to the DM Object
DIRECTORY	Used to attach a directory to the DM Object
LINK DIRECTORY	Used to link a directory to the DM Object
STUDY VERSION	Values following a hardcoded versioning scheme
TIME STAMP	Value filled with the current timestamp
VERSIONING SCHEME COUNTER	Values following a counter versioning scheme of custom number of digits



The **Default Value** is used to assign a value to the attribute of a newly created DM Object. If accepted values are defined, the default value must necessarily be one of them. The **Prefix** field is a fixed string that will be prepended before auto-filled attribute values. For example, in order to request the version to appear as V001 instead of 001, one can set "V" as Prefix. In the Length field, the length of the prefix is set.

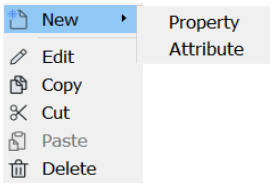
The **Allow Empty** checkbox can be used to allow a DM Object to be defined with this attribute empty. The **Read Only** checkbox is used to control if the attribute value should be editable or not.



In the **Accepted Values** field the accepted values of the attribute are set. When defining or editing such an object through the UI, the user is prompted to select one of the accepted values from a combo box.

Accepted values can be added to a property/attribute with the aid of the *Accepted Values Editor*, accessed through the respective button ... . Press the Add/Remove buttons to configure the accepted values accordingly. When finished, press the **OK** button to close the window.

Any errors occurring while working with the *DM Schema Editor* will be displayed with a pop-up message.

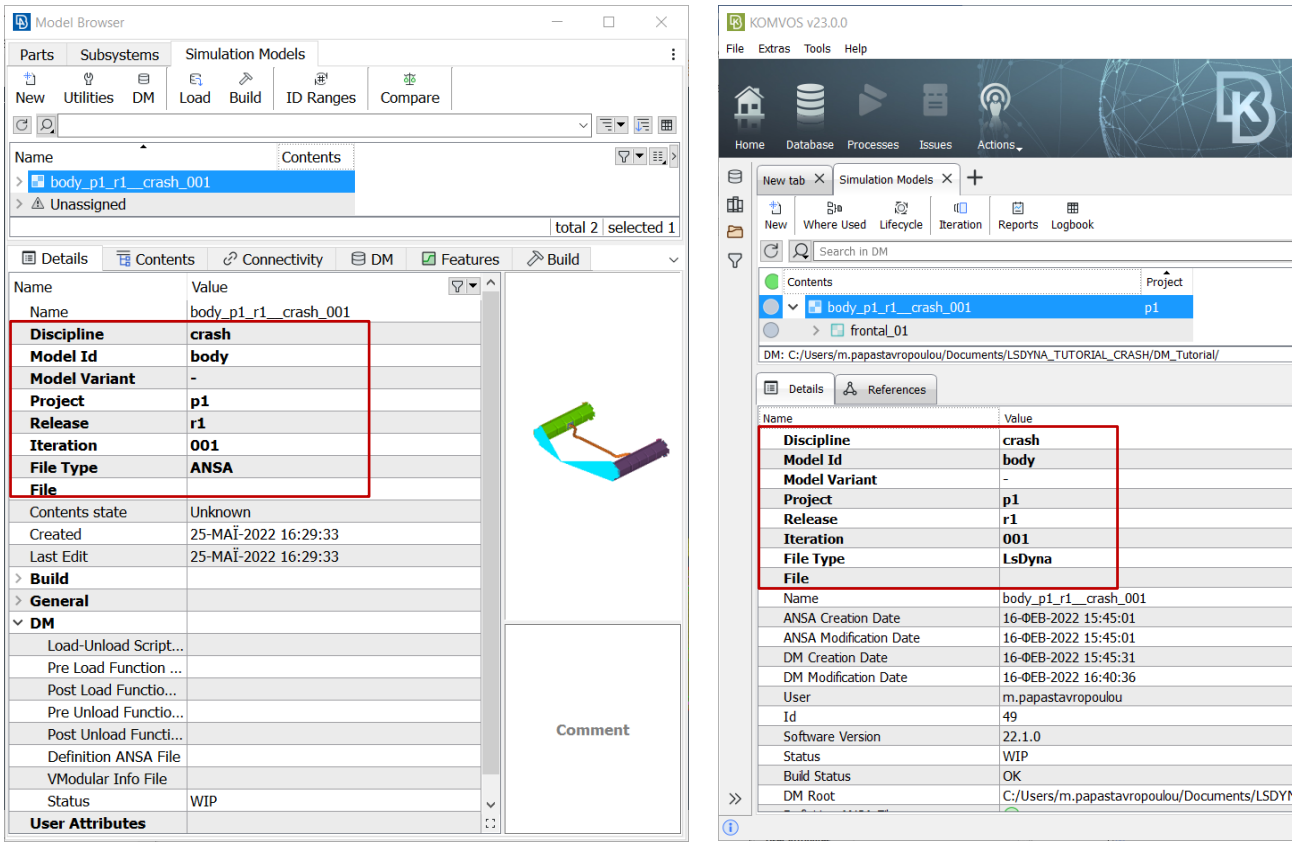


A new property/attribute can be created by selecting **New>Property/Attribute** option in the context menu.

Drag and drop of attributes within the list can change their order. This way frequently used attributes e.g. *Module Id,* that are prioritized by the user, can be displayed first.

Note that no two Properties/Attributes can have the same Name. The *DM Schema Editor* will prevent this.

Both primary and secondary attributes are displayed in the Details Tab in the Data Workspace of KOMVOS and in the Model Browser/DM Browser in ANSA/META. By default, the primary attributes are displayed first and in bold font style and the secondary attributes follow (to configure the Details Tab according to preference, see the related options in *paragraph 5.2.1.4*). In the Model Browser of ANSA/META in particular, the secondary attributes are displayed inside the *DM* category, except for the *Name* attribute which is shown at the very top at all cases.

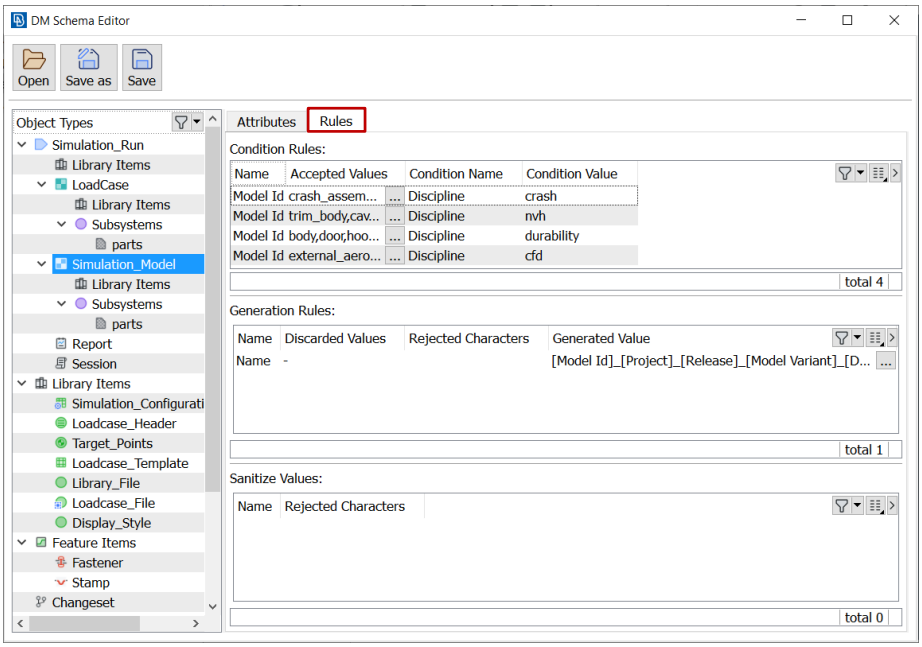


! Important note:

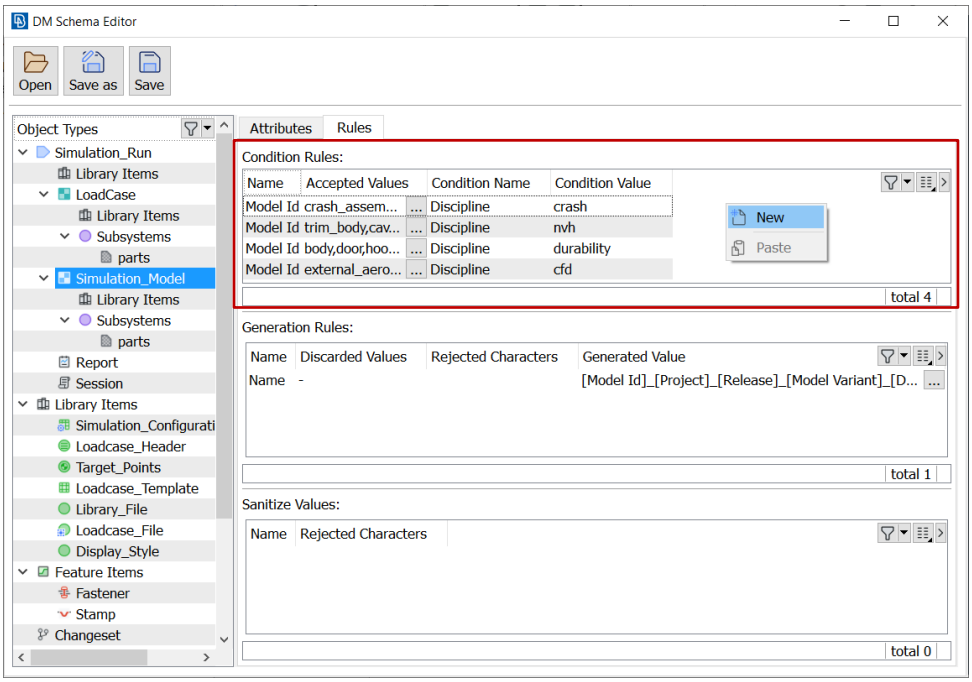
Once the DM has been populated with DM objects, the Properties of their DM Object Types cannot be modified anymore as this would corrupt the DM. This means that no new properties can be added, no existing properties can be modified and that the characteristics of the existing properties cannot be modified (except for the accepted values that can be updated as required). Such modifications may only be possible for environments with SPDRM back-end and should only be attempted after consulting BETA CAE Systems support team.

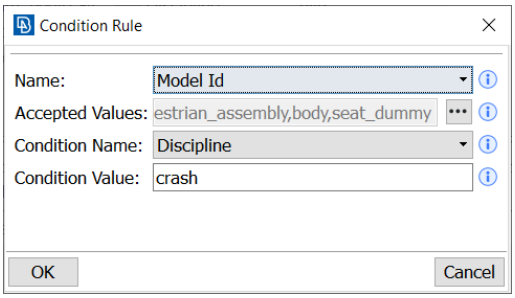
5.1.1.4. Adding Condition Rules

Rules to restrict the accepted values of a property/attribute, as well as to auto-generate its value based on some condition, can be defined in the *Rules* tab of the *DM Schema Editor* window.



In the *Condition Rules* area one can define the accepted values of a property with respect to the values of another property.



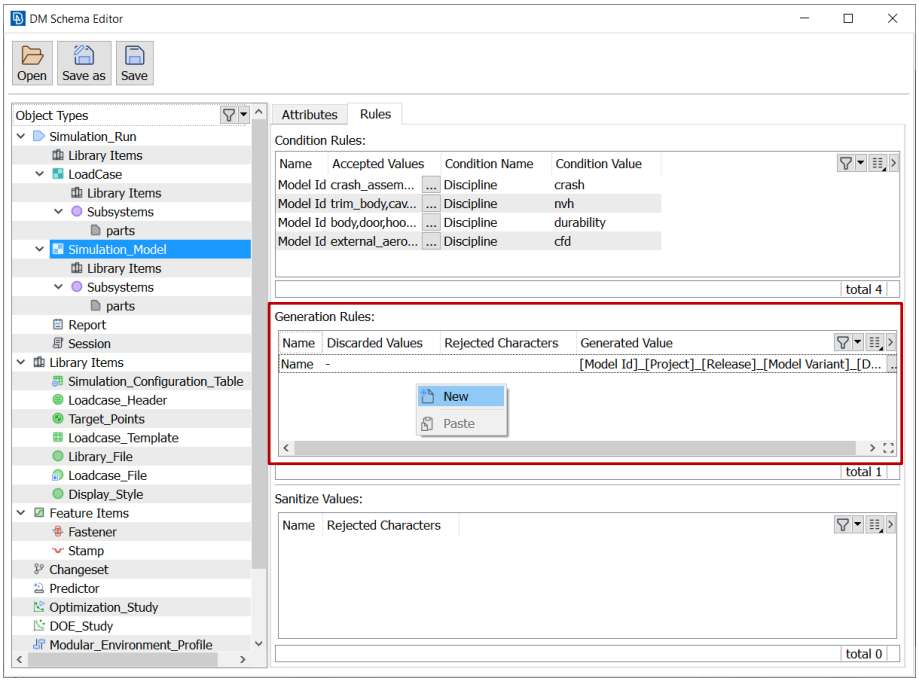


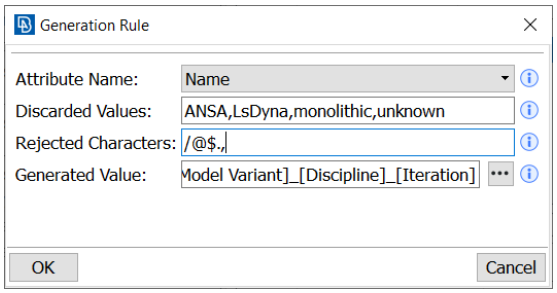
Taking as an example the *Model Id* property of the Simulation Model in the default data model, the rule shown on the left defines the accepted values of this property based on the value of the property *Discipline*.

If the property/attribute **Condition Name** has value equal to **Condition Value**, then the property/attribute **Name** must only accept values among **Accepted Values.**

5.1.1.5. Adding Generation Rules

In the *Generation Rules* area one can define the rule according to which the SDM system will auto-generate the value of a property/attribute based on the values of other properties/attributes. Taking as an example the *Name* attribute in the default data model, the rule shown below composes the value of this attribute based on the values of the attributes written in the generated value field.

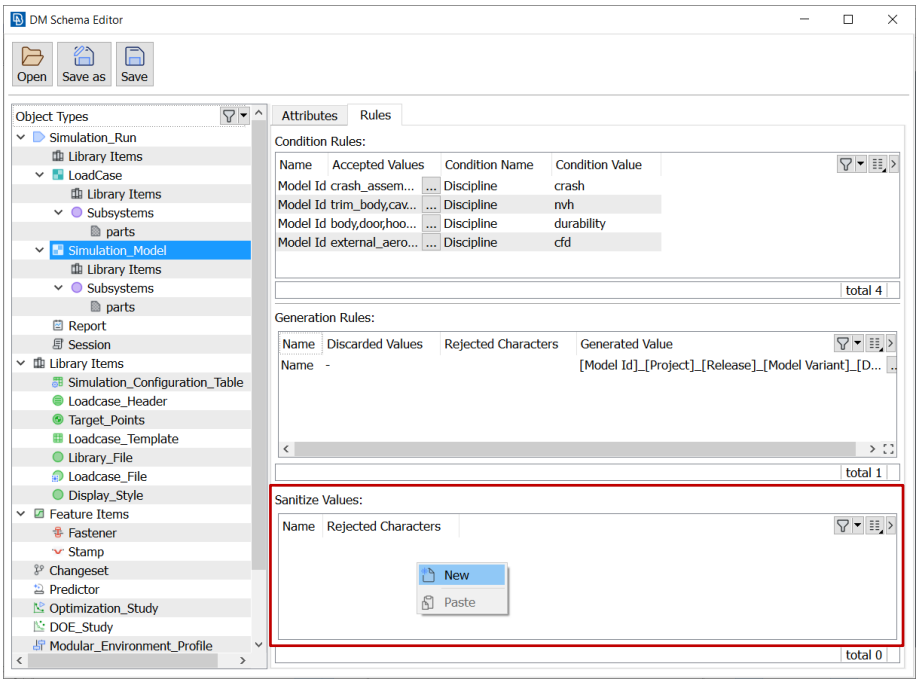


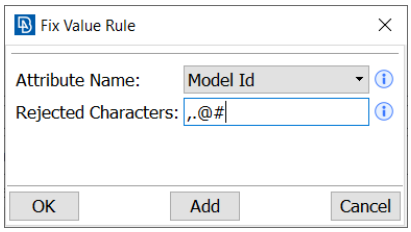


The **Attribute Name** field holds the attribute that will be affected by the rule. In the **Generated Value** field, the user can type the expression based on which the value of the affected property will be composed. In the **Discarded Values** field, the user can optionally type any values of the attributes used in the rule that are not accepted and should be discarded from the generated value as a whole. In the **Rejected Characters** field, special characters that should not be allowed anywhere in the generated value can be typed. They are also discarded from the generated string.

5.1.1.6. Adding Sanitize Values

In the *Sanitize Values* area one can define for each property/attribute of the specific DM Object Type, a list of characters that are considered non-accepted and should be removed from the value.

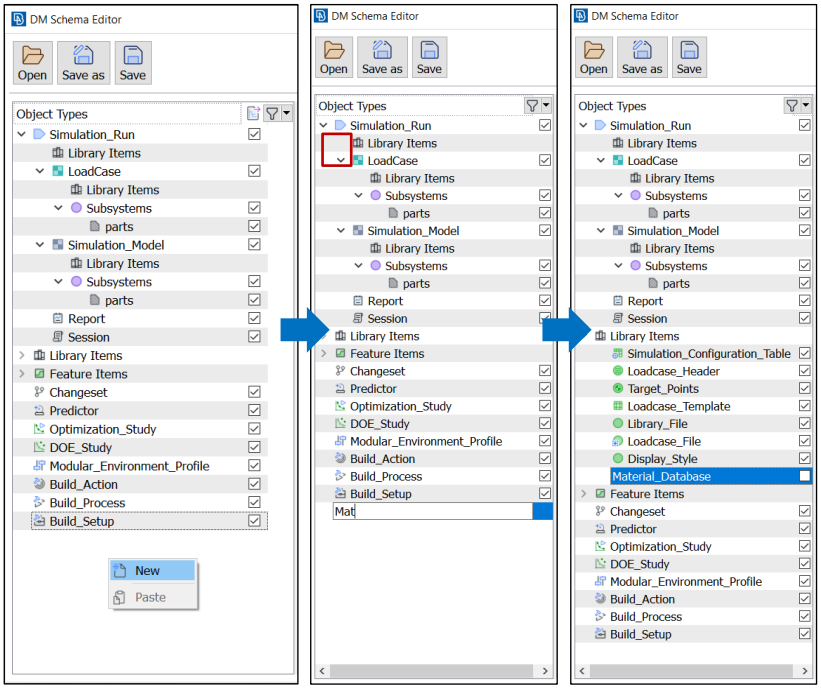




The property/attribute concerned is defined in the **Attribute Name** field by selecting among a list of all properties/attributes of the specific DM Object Type. In the **Rejected Characters** field, a list of all characters to be removed and replaced with blank value, can be defined. Note that these characters should not be comma separated. The comma could be among the rejected characters.

5.1.1.7. Adding new Library Items

To define a new Library Item, right click anywhere in the left pane of the *DM Schema Editor* and select the option **New** of the context menu. Type the name of the new item and press Enter.



Once the new item is created, it can be configured as described in the paragraphs above.

Note that the **Output** column controls with the aid of the checkboxes which DM Object Types will be written in the data model xml file during save. New created items are unchecked by default.

5.1.1.8. Saving the DM Schema

The file created with this tool is the data model xml file that is described in detail in *paragraph 5.1.2* and can be saved using the **Save** or **Save as** buttons. The file is ready-to-use for file-based DMs but requires certain manual modifications for SPDRM. More information on the required modifications is given in *paragraph 5.1.3*.

For file-based DMs, the *Save* button will update the `dm_structure.xml` file of the current DM.

For SPDRM back-end, the *Save* button is inactive, as the update of the data model takes place server-side by the system administrator.

5.1.2. The data model xml file

The data model xml describes the characteristics of the different DM Object Types. Within the xml file, each DM Object Type is described in a block like the one shown below:

File structure	Description
⊟ Simulation_Model	DM Object Type.
⊟ Properties	Properties section.
⊞ Property	One property. One such node is required for each property.
⊟ Attributes	Attributes section.
⊞ Attribute	One attribute. One such node is required for each attribute.
⊟ Rules	Rules section (condition, generation, etc.)
⊞ Rule	One rule. One such node is needed for each rule.

The data model xml used by SPDRM has some differences comparing to that of a file-based DM. The top-level differences are summarized in the table below.

File-based DM (`dm_structure.xml`)	SPDRM (`dm_structure_TBM.xml`)
⊟ XMLData	⊟ XMLData
⊞ **p**art**s**	⊞ **P**art
⊞ Subsystem**s**	⊞ Subsystem
⊞ Simulation_Model	⊞ Simulation_Model
⊞ Load**C**ase	⊞ Loadcase
⊞ Simulation_Run	⊞ Simulation_Run
⊞ Report	⊞ Report
⊞ Session	⊟ LIBRARY_ITEMS
⊞ **Changeset**	⊞ LIBRARY_ITEM: META_Session
⊞ Predictor	⊟ DEFAULT_ITEMS
⊞ Optimization_Study	⊞ DEFAULT_ITEM: Predictor
⊞ Modular Environment Profile	⊞ DEFAULT_ITEM: Optimization_Study
⊞ Build Action	⊞ DEFAULT_ITEM: Modular_Environment_Profile
⊞ Build Process	⊞ DEFAULT_ITEM: Build Action
⊞ Build Setup	⊞ DEFAULT_ITEM: Build Process
⊞ DOE Study	⊞ DEFAULT_ITEM: Build Setup
⊞ LIBRARY ITEMS	⊞ DEFAULT_ITEM: DOE Study
⊞ FEATURE ITEMS	⊞ FEATURE_ITEMS
⊞ **DM Settings**	⊞ **Aliases**

In this paragraph we will only refer to the xml top elements that are common in both cases. The *DM Settings* node, which is file-based DM-specific, as well as the *Aliases* node, which is SPDRM-specific, are described in *paragraphs 5.1.4 and 5.1.3.4* respectively.

5.1.2.1. DM Object Types general info

The core definition of a DM Object Type element in the `dm_structure.xml` is shown below:

File structure

- <DM Object Type Name>
  - Properties
    - Property
  - Attributes
    - Attribute
  - Rules
    - Rule

Each element that corresponds to a DM Object Type e.g. *Subsystem* can contain three main sub-elements: The Properties, *Attributes* and *Rules*. The element **Properties** holds the primary attributes that are used as identifying keys of the DM Object. The element **Attributes** holds the secondary attributes of the DM Object. The **Rules** element can hold *Condition* and *Generation Rules*.

The definition of a Subsystem element is shown below.

```
<Subsystems>
    <Properties>
        <Property name="Module Id" type="TEXT" allow_null="NO" read_only="NO"/>
        <Property name="Variant" type="TEXT" default_value="-" allow_null="NO" read_only="NO"/>
        <Property name="Project" type="TEXT" default_value="-" allow_null="NO" read_only="NO"/>
        <Property name="Release" type="TEXT" default_value="-" allow_null="NO" read_only="NO"/>
        <Property name="Iteration" type="VERSIONING SCHEME COUNTER" default_value="001" allow_null="NO" read_only="NO" format="%03d"/>
        <Property name="Loadcase Variant" type="TEXT" default_value="-" allow_null="NO" read_only="NO"/>
        <Property name="File Type" type="TEXT" default_value="ANSA" accepted_values="ANSA,Nastran,LsDyna,PamCrash,Abaqus" allow_null="NO" read_only="NO"/>
        <Property name="Representation" type="TEXT" accepted_values="crash_fe,nvh_fe,dura_fe,lumped_mass,'',Regular" allow_null="YES" read_only="NO"/>
    </Properties>
    <Attributes>
        <Attribute name="Name" type="TEXT"/>
        <Attribute name="Status" type="TEXT" default_value="WIP" accepted_values="WIP,OK,Warning,Error"/>
    </Attributes>
    <Rules>
        <Rule name="Name" discarded_chars="-" rejected_characters="[!@/?" generated_value="[Module Id]_[Project]_[Release]_[Variant]"/>
        <Rule name="Variant" accepted_values="v1" condition_name="Project" condition_value="p1"/>
        <Rule name="Module Id" rejected_characters="@!.,?"/>
    </Rules>
</Subsystems>
```

A **Property** element defines a primary attribute for the DM Object Type with the use of the following xml attributes:

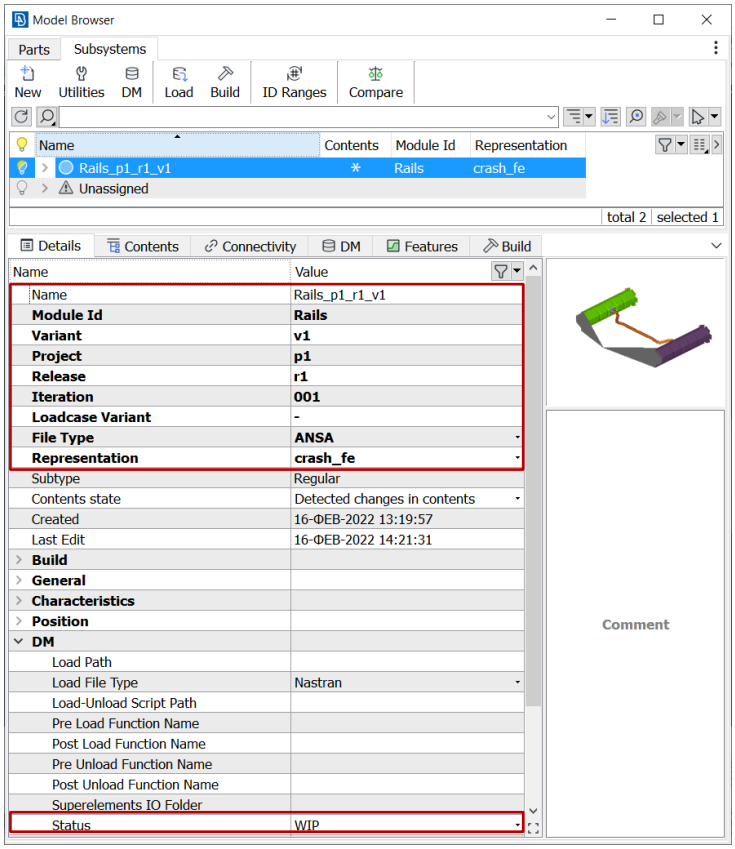
- **name**: The name of the attribute as it will appear in the UI
- **type**: The type of the attribute (see table with accepted attribute types below)
- **default_value**: The value the attribute should have upon object creation and before any user modification
- **accepted_values**: The accepted values of the attribute. In the UI, to set the value of an attribute with accepted values the user is prompted to select among the values of a combo box.
- **allow_null**: This setting controls whether an object can be defined with this attribute empty (values: YES/NO)
- **read_only**: controls if the attribute value is editable or not (values: YES/NO)

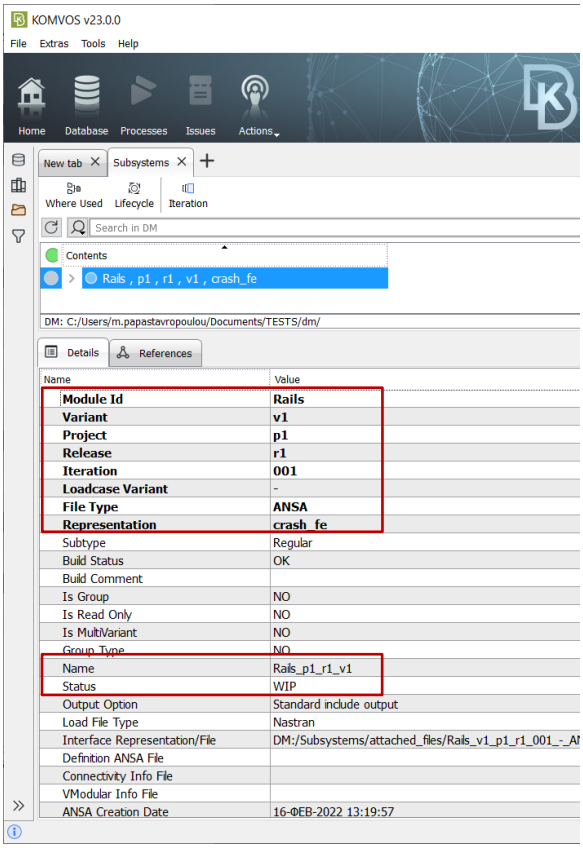
The **Attribute** element defines a secondary attribute for the DM Object Type with the use of the same xml attributes.

A property/attribute of a DM Object can be of various types. The supported types are shown in the table below:

TEXT	Accepts any kind of text
BOOLEAN	Accepts True/False values
INTEGER	Accepts integer values
DOUBLE	Accepts double values
FILE	Used to attach a file to the DM Object
LINK FILE	Used to link a file to the DM Object
DIRECTORY	Used to attach a directory to the DM Object
LINK DIRECTORY	Used to link a directory to the DM Object
STUDY VERSION	Values following a hardcoded versioning scheme
TIME STAMP	Value filled with the current timestamp
VERSIONING SCHEME COUNTER	Values following a counter versioning scheme of custom number of digits

Both primary and secondary attributes are displayed in the Details Tab in the Data Workspace of KOMVOS and in the Model Browser/DM Browser in ANSA/META. By default, the primary attributes are displayed first and in bold font style and the secondary attributes follow (to configure the Details Tab according to preference, please refer to *paragraph 5.2.1.4*). In the Model Browser of ANSA/META in particular, the secondary attributes are displayed inside the *DM* category, except for the *Name* attribute which is shown at the very top at all cases.





There are three types of rules supported, the **Condition Rules**, the **Generation Rules** and the **Sanitize Values**. All rules are introduced with the **Rule** element.

With a *Condition Rule* one can define the accepted values of a property/attribute with respect to the values of another property/attribute. The **Rule** element adds a new *Condition Rule* with the use of the following xml attributes:

- **name**: the name of the property that will be affected by the rule
- **accepted_values**: the values the affected property will take when the condition property takes the condition value
- **condition_name**: the name of the property that will control the value of the affected property
- **condition_value**: the value of the Condition property defined in *condition_name*

In file-based DM, in order to denote that a property/attribute should get its accepted values from a *Condition Rule,* no *accepted_values* keyword must be present. Note that if this property/attribute needs to have a default_value, it must be among the accepted values defined by the rule.

With a *Generation Rule* one can define the rule according to which the SDM system will auto-generate the value of a property/attribute based on the values of other properties/attributes. The **Rule** element adds a new *Generation Rule* with the use of the following xml attributes:

- **name:** the name of the property/attribute that will be affected by the rule
- **discarded_chars**: the user can optionally type any values of the attributes used in the rule that are not accepted and should be discarded from the generated value as a whole
- **generated_value**: the expression based on which the value of the affected property will be composed
- **rejected_characters**: special characters that should not be allowed anywhere in the generated value can be typed.

With a *Sanitize Value* one can define the rule according to which the SDM System will define for each property/attribute of the specific DM Object Type, a list of characters that are considered non-accepted and should be removed from the value and replaced by a blank value.

- **name:** the name of the property that will be affected by the rule
- **rejected_characters:** special characters that should not be allowed anywhere in the property value.

Restrictions in the customization of DM Object Types

There are certain restrictions in the customization of the data model in order to avoid making changes that might disrupt the smooth operation of the system. When the editing of the data model is done through the DM Schema Editor, the compliance with these restrictions is checked by the tool. However, when the data model is edited manually, they need to be taken care of by the user. These are:

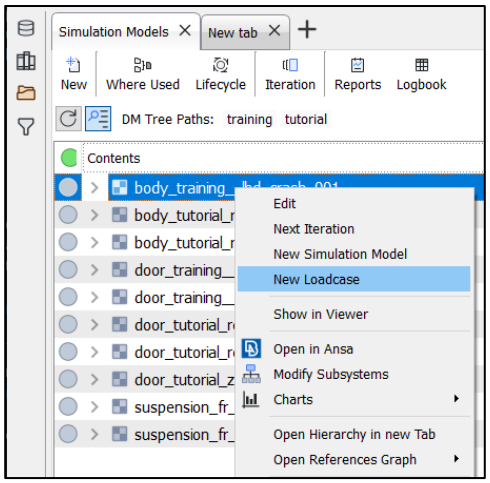
- *Module Id, Representation, File Type, Status, Name* are special attributes that are referred to by several tools in ANSA/META and KOMVOS. Therefore, renaming, deleting and changing their type is not permitted.
- *Version* and *Study Version* are default attributes with attribute types TEXT and STUDY VERSION respectively. Using them with any other type is not permitted.
- For Simulation Models, Loadcases, Simulation Runs and any other DM Object Type that by default has a primary attribute of type FILE defined last, no other new primary attribute should be added after the FILE attribute

5.1.2.2. Simulation Data-specific info

The core definition of a **Simulation Data** DM Object Type element in the `dm_structure.xml` is shown below:

File structure

```
<Simulation Data Type>
    Properties
        Property
    Attributes
        Attribute
    Containing_Items
        Containing_Item
    Rules
        Rule
```



Comparing to the general description of DM Object Types (see *paragraph 5.1.2.1*), Simulation Data types (i.e. Simulation Models, Loadcases, Simulation Runs and Reports) have one more element, the **Containing_Items**.

DM Object Types defined as Containing_Items of a DM Object, are those that will be found in the context menu of DM items of this type, under the option “New” in the data browsing tools.

The relationships that are permitted among Simulation Data types are:

- A Simulation Model contains Loadcases and may also contain Reports and Sessions
- A Loadcase contains Simulation Runs, and may also contain Reports and Sessions
- A Simulation Run contains Reports and may also contain Sessions

```
<Simulation_Model>
    <Properties>
    <Attributes>
    <Containing_Items>
        <Containing_Item type="LoadCase"/>
        <Containing_Item type="Report"/>
        <Containing_Item type="Session"/>
    </Containing_Items>
    <Rules>
</Simulation_Model>
```

5.1.2.3. Library Data-specific info

Through the `dm_structure.xml` it is possible to define new DM Object Types under the category of Library Items. The types of Library Items that are supported in the default data model, are shown below:

```
<LIBRARY_ITEMS>
    <LIBRARYITEM type="Simulation_Configuration_Table">
    <LIBRARYITEM type="Loadcase_Header">
    <LIBRARYITEM type="Target_Points">
    <LIBRARYITEM type="Loadcase_Template">
    <LIBRARYITEM type="Library_File">
    <LIBRARYITEM type="Loadcase_File">
    <LIBRARYITEM type="Display_Style">
</LIBRARY_ITEMS>
```

As an example, the definition of a **Library Item** to hold crash dummy models, is shown below:

File structure

- XMLData
  - LIBRARY_ITEMS
    - LIBRARYITEM
      - Properties
        - Property
      - Attributes
        - Attribute

Sample

```
<XMLData>
    <LIBRARY_ITEMS>
        <LIBRARYITEM type="Dummy" >
            <Properties>
                <Property name="Maker" type="TEXT" default_value="" accepted_values="FTSS,LSTC" allow_null="NO" />
                <Property name="Type" type="TEXT" default_value="" accepted_values=",H350,H395,H305_female,SID-II
                <Property name="Maker Version" type="TEXT" default_value="" allow_null="NO" />
                <Property name="File" type="FILE" allow_null="NO" />
            </Properties>
            <Attributes>
                <Attribute name="Encrypted" type="TEXT" default_value="" accepted_values="NO,YES," allow_null="NO
                <Attribute name="Comment" type="TEXT" default_value=""/>
            </Attributes>
        </LIBRARYITEM>
    </LIBRARY_ITEMS>
</XMLData>
```

The **type** attribute of LIBRARYITEM element defines the name of the new Library item type. For the definition of the *Properties* and *Attributes* elements see *paragraph 5.1.2.1*.

5.1.2.4. DM Object Type Aliases

The Data Model supports the definition of aliases that can define an alternative name for the DM Object Types. This is achieved with the aid of aliases defined in the `dm_structure.xml` file. Using such aliases, labels like Subsystems, Simulation_Model, etc. that are widely used in the *Data Workspace* of KOMVOS and in the *Model Browser* or *DM Browser* of ANSA/META, can be renamed to adapt to the custom terminology used by each engineering team of an organization.

The definition of a DM Object Type **Alias**, is shown below:

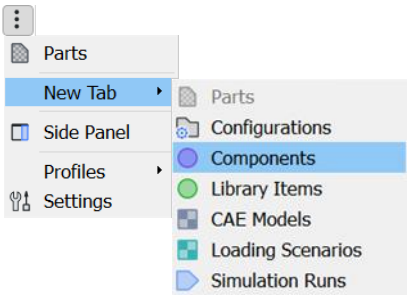
File structure

- XMLData
  - <Custom DM Object Type Name>
    - Properties
    - Attributes
    - Containing Items
    - Rules

Sample

```
<XMLData>
    <CAE_Model aliasOf="Simulation_Model">
        <Properties>
        <Attributes>
        <Containing_Items>
            <Containing_Item type="Loading_Scenario"/>
            <Containing_Item type="Report"/>
            <Containing_Item type="Session"/>
        </Containing_Items>
        <Rules>
    </CAE_Model>
    <Loading_Scenario aliasOf="LoadCase">
    <Component aliasOf="Subsystems">
</XMLData>
```

! **NOTE:** that the *Containing_Items* element must also be updated to refer to the aliased names as shown in the picture above.



Defining such aliases in the data model, all type references in the Model Browser and DM Browser of ANSA/META and in the Data workspace of KOMVOS will be updated accordingly.

5.1.3. SPDRM-specific topics

As previously mentioned, there are certain differences between the data model configuration file of SPDRM and that of file-based DM. To customize SPDRM's data model the user needs to edit the default `dm_structure_TBM.xml` file. However, for cases where the user does not feel comfortable making vast changes in the xml file through the text editor, it is possible to use the *DM Schema Editor* to do the core modifications of the data model, export an xml file formatted for file-based DM, and then convert it to a structure compatible with SPDRM.

5.1.3.1. General conversion guidelines

General guidelines for the conversion of the `dm_structure.xml` to the `dm_structure_TBM.xml` are presented below:

A. The following renaming of Object Types must take place:
   a. **<parts>** must be switched to <**Part>**
   b. **<Subsystems>** must be switched to <**Subsystem>**
   c. **<LoadCase>** must be switched to <**Loadcase>**

B. For each DM Object, SPDRM constructs a path that is stored in the database and must be unique for each object. This path is generated with concatenation of selected object *Properties*. The **dmpath_position** keyword (old '*position*' keyword is still supported), added in the *Properties* element, indicates the properties that must be used for the path generation and their order within this path.

C. For the Simulation Model, Loadcase, Simulation Run and Report object types, the attached file is defined as an *Attribute* in SPDRM, whereas in file-based DM is defined as a *Property*. (see *File* attribute below)

D. To denote that a property/attribute should get its accepted values from a *Condition Rule* the *accepted_values* keyword must be added in the property/attribute element with an empty value, **accepted_values=""**. This is necessary only for SPDRM (see *Team* property below). Note that if this property/attribute needs to have a default_value, it must be among the accepted values defined by the rule.

E. When an empty value is specified as an **accepted value** of an attribute, for file-based DM single quotes are used to denote the empty value, whereas for SPDRM just an empty space is used. (see *Discipline* property accepted values below)

F. In SPDRM, a regular expression validator exists (*regexValid* keyword) in the `taxis.conf` file, that is used to validate values inserted as properties and attributes by the user (The comment field does not need to pass this validation)

G. The **version_check** keyword is used for properties/attributes of type VERSION_SCHEME or VERSION_SCHEME_COUNTER. The accepted values are "YES" and "NO" (default). When that attribute is used for a DM Item or a Rich Library Item and a user attempt to edit that item, a warning message appears if the selected item is not the latest version.

H. For the definition of Generation Rules with SPDRM back-end, the **Rejected Characters** should be removed as it will be neglected.

I. For the Subsystem object type, *the* representation file (*File*) is defined as a *Property* in SPDRM. For file-based DM, this property is not required.

File-based DM

```
<Simulation_Model>
    <Properties>
        <Property name="Model Id" read_only="NO" type="TEXT" default_value="" allow_null="NO"/>
        <Property name="Model Variant" read_only="NO" type="TEXT" default_value="-" allow_null="NO"/>
        <Property name="Project" read_only="NO" type="TEXT" default_value="" allow_null="NO"/>
        <Property name="Release" read_only="NO" type="TEXT" accepted_values="" allow_null="NO"/>
        <Property name="Discipline" read_only="NO" type="TEXT" accepted_values="C,N,'',S"/>
        <Property name="Iteration" read_only="NO" type="VERSIONING_SCHEME_COUNTER" format="
        <Property name="File Type" type="TEXT" default_value="ANSA" accepted_values="ANSA,Na
        <Property name="File" type="ATTACHED_FILE" default_value=""/>
        <Property name="Team" type="TEXT" />
    </Properties>
    <Attributes>
        <Attribute name="Name" type="TEXT"/>
        <Attribute name="Status" type="TEXT" default_val
    </Attributes>
    <Rules>
        <Rule name="Name" discarded_chars="-" rejected_characters="[$^.?" generated_value="[Model Id]_[Project]_[Release]_[Model Va
        <Rule name="Team" condition_name="Discipline" condition_value="CRASH" accepted_values="FRONT,SIDE,REAR,PEDESTRIAN" />
    </Rules>
</Simulation_Model>
```

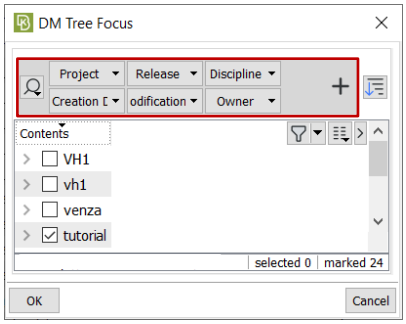
SPDRM

```
<Simulation_Model>
    <Properties>
        <Property name="Model Id" dmpath_position="4" read_only="NO" type="TEXT" default_value="" allow_null="NO"/>
        <Property name="Model Variant" dmpath_position="5" read_only="NO" type="TEXT" default_value="-" allow_null="NO"/>
        <Property name="Project"  dmpath_position="1" dmfilter="YES" dmfilter_position="1" read_only="NO" type="TEXT" default_value="" allow_null="N
        <Property name="Release" dmpath_position="2" dmfilter="YES" dmfilter_position="2" read_only="NO" type="TEXT" accepted_values="" allow_null="
        <Property name="Discipline" dmpath_position="3" dmfilter="YES" dmfilter_position="3"  read_only="NO" type="TEXT" accepted_values="C,N,,S"/>
        <Property name="Iteration" dmpath_position="6" read_only="NO" type="VERSIONING_SCHEME_COUNTER" version_check="YES" format="%03d" d
        <Property name="File Type" dmpath_position="7" type="TEXT" default_value="ANSA" accepted_values="ANSA,Nas
        <Property name="Team" dmpath_position="8" type="TEXT" accepted_values="" />
    </Properties>
    <Attributes>
        <Attribute name="Name" type="TEXT"/>
        <Attribute name="Status" type="TEXT" default_value="WIP" accept
        <Attribute name="File" type="ATTACHED_FILE" default_value=""/>
    </Attributes>
    <Rules>
        <Rule name="Name" discarded_chars="-" generated_value="[Model Id]_[Project]_[Release]_[Model Variant]_[Discipline]_[Iteration]"/>
        <Rule name="Team" condition_name="
        ine" condition_value="CRASH" accepted_values="FRONT,SIDE,REAR,PEDESTRIAN" />
    </Rules>
</Simulation_Model>
```

B C D E G H

5.1.3.2. DM Tree Focus Configuration

It is possible to control which quick filters will be available in the Data Tree Focus of the Data workspace in KOMVOS, for Simulation Model and Simulation Run DM Objects using the **dmfilter=YES** keyword. To define the order of appearance of these filters, the **dmfilter_position** key is used (optional). The *DM Creation Date*, *DM Configuration Date* and *Owner* attributes are also added in the filter area by default.




```
<Simulation_Model>
 <Properties>
   <Property name="Model Id" read_only="NO" type="TEXT" default_value="" allow_
   <Property name="Model Variant" read_only="NO" type="TEXT" default_value="-"
   <Property name="Project" dmfilter="YES" dmfilter_position="1" read_only="NO"
   <Property name="Release" dmfilter="YES" dmfilter_position="2" read_only="NO"
   <Property name="Discipline" dmfilter="YES" dmfilter_position="3"  read_only=
   <Property name="Iteration" read_only="NO" type="VERSIONING_SCHEME_COUNTER" f
   <Property name="File Type" type="TEXT" default_value="ANSA" accepted_values=
 </Properties>
...
</Simulation_Model>
```

5.1.3.3. Library Items

In SPDRM, the session files are considered Library Items of type **META_Session** and must be added inside the LIBRARY_ITEMS element.

```
<LIBRARY_ITEMS>
  <LIBRARYITEM type="Simulation_Configuration_Table">
  <LIBRARYITEM type="META_Session">
  <LIBRARYITEM type="Library_File">
  <LIBRARYITEM type="Loadcase_File">
  <LIBRARYITEM type="Loadcase_Header">
  <LIBRARYITEM type="Target_Points">
  <LIBRARYITEM type="Loadcase_Template">
</LIBRARY_ITEMS>
```

New types of Library Items can also be defined in the `dm_structure_TBM.xml`. However, the attached file must be defined within the **Name** property and not through a standalone **File** property as it is defined in `dm_structure.xml`.

```
<LIBRARY_ITEMS>
 <LIBRARYITEM type="Material" container_name="Materials">
  <Properties>
    <Property name="Type" dmpath_position="1" type="TEXT" accepted_values="Steel,Aluminium" allow_null="NO"/>
    <Property name="Revision" dmpath_position="2" type="VERSIONING_SCHEME_COUNTER" format="%02d" default_value="01"/>
    <Property name="Name" type="ATTACHED_FILE" default_value="" allow_null="NO"/>
  </Properties>
 </LIBRARYITEM>
<LIBRARY_ITEMS>
```

5.1.3.4. Accepted Values Aliases

It is possible to create a list of aliases to be used in the definition of the accepted values of properties and attributes. In this way, when the same property/attribute is used by different object types, we can ensure that the accepted values will be the same and when modifications need to be made, this can be made centrally.

```
<Simulation_Model>
 <Properties>
   <Property name="Model Id" dmpath_position="4" read_only="NO" type="TEXT" accepted_values ="$_Model_Id_" allow_null="NO"/>
   <Property name="Model Variant" dmpath_position="5" read_only="NO" type="TEXT" default_value="-" allow_null="NO"/>
   <Property name="Project"  dmpath_position="1" read_only="NO" type="TEXT" accepted_values ="$_Project_" allow_null="NO"/>
   <Property name="Release" dmpath_position="2" read_only="NO" type="TEXT" accepted_values="$_Release_" allow_null="NO"/>
   <Property name="Discipline" dmpath_position="3" read_only="NO" type="TEXT" accepted_values="$_Discipline_"/>
   <Property name="Iteration" dmpath_position="6" read_only="NO" type="VERSIONING_SCHEME_COUNTER" format="%03d" default_valu
   <Property name="File Type" dmpath_position="7" type="TEXT" default_value="ANSA" accepted_values="$_File_Type_" />
</Properties>
 …
</Simulation_Model>
<Aliases>
  <Alias name="_Project_" value="proj1,proj2,proj3" />
  <Alias name="_Release_" value="rel1,rel2,rel3" />
  <Alias name="_Model_Id_" value="crash_assembly,trim body,full vehicle,biw" />
  <Alias name="_Discipline_" value="crash,strength,nvh" />
  <Alias name="_Status_" value="WIP,OK,Warning,Error" />
  <Alias name="_File_Type_" value="ANSA,Nastran,LsDyna,Abaqus" />
</Aliases>
```

5.1.4. File-based DM-specific topics

There are settings that affect file-based DMs and are considered custom per DM. For this reason, they are configured through the **DM_Settings** element, of the `dm_structure.xml` file, in order to affect the behavior of the Data Manager in ANSA, META or KOMVOS only when connected to this specific file-based DM. These settings are configured as shown below:

```
<DM_Settings>
    <DM_Setting name="Software Version" value="22.1.0"/>
    <DM_Setting name="adaptation_key_calculation_method" value="v1"/>
    <DM_Setting name="avoid_special_chars" value="YES"/>
    <DM_Setting name="avoid_special_chars_everywhere" value="NO"/>
    <DM_Setting name="display_types_instead_of_object_types" value="YES"/>
    <DM_Setting name="flatten_contained_items" value="YES"/>
    <DM_Setting name="intermodular_connectivity_links" value="NO"/>
    <DM_Setting name="library_dm" value="NO"/>
    <DM_Setting name="support_solver_relative_paths" value="NO"/>
    <DM_Setting name="target_point_property_name" value="Target Point"/>
</DM_Settings>
```

Software Version: Keeps the software version of ANSA or KOMVOS that was used to create a DM for the first time.

adaptation_key_calculation_method: This setting should not be modified by users. It should only be filled by ANSA for file-based DMs.

avoid_special_chars [**YES**/NO]**:** Requests the avoidance of all special characters in folder names when some file creation operation takes place in DM. Affects the way DM paths are created for all DM Object Types except from: Parts / Subsystems / Includes / Configurations. Do not change these values in DMs that already contain DM Objects.

avoid_special_chars_everywhere [YES/**NO**]: When set to YES, it requests the avoidance of all special characters in any folder names and filenames that can appear within a DM. Affects the way DM paths are created for all DM Object Types except for Parts and Subsystems. These values should not be changed in DMs that already contain DM Objects.

! NOTE: Special characters are those that are not alphanumeric ([A - Za - z0 - 9]), dash ([' - ']), dot ([' . ']), or underscore ([' _ ']).

flatten_contained_items [**YES**/NO]: When set to YES, it requests the avoidance of the nesting of children DM Objects below their parent and thus shortens the maximum path length appearing in DM. Affects the way DM paths are created for all DM Object Types except for Parts and Subsystems.

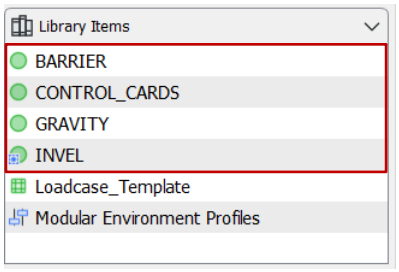
intermodular_connectivity_links [YES/**NO**]**:** When set to YES, it enables the functionality of Intermodular Connections.

library_dm [YES/**NO**]**:** Denotes that this DM is used as a library DM (read only member) of a Cluster DM. With this setting active, when this DM is used as a primary, plain DM (e.g. in order to be populated with library data by the team leader), trying to delete or overwrite any object, raises a warning, as this could affect the integrity of the Cluster DM.

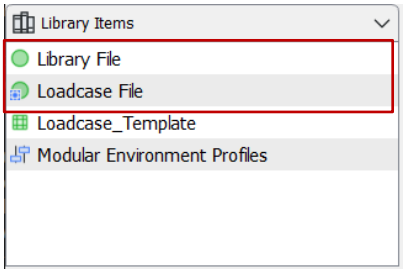
support_solver_relative_paths [YES/**NO**]**:** Controls whether include keyword references will be written as absolute paths (NO) or as relative paths.

target_point_property_name [Target Point]**:** Defines the name of the Simulation Run Property, used as Target Points identifier for the Pedestrian implementation model assembly. By default, a secondary attribute named "Target Point" is used.

display_types_instead_of_object_types [**YES**/NO]: Controls how Library Items will be displayed in the Contents section of the data browsing tools. If a Library Item DM Type has a property named **Type**, setting this option to YES will display one category for each different Type property of the Library Item. Taking the example of Library File and Loadcase File of the default dm_structure.xml, the impact of this setting is shown in the screenshots below:



display_types_instead_of_object_types = YES

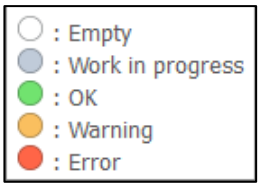


display_types_instead_of_object_types = NO

5.1.5. Special attributes

5.1.5.1. The “Status” attribute

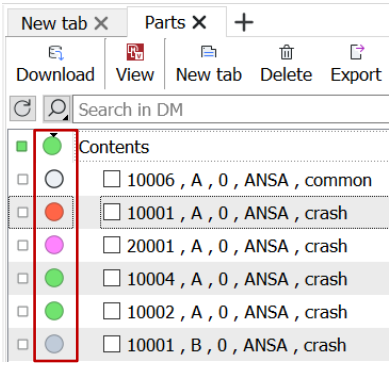
The *Status* attribute is the attribute that holds information on the state of a DM Object in terms of maturity or suitability for use. By default, it has four accepted values: **WIP** (Work in progress), **OK**, **Error**, **Warning**, that are matched with one default color each, as shown in the picture on the right. This color is used in the Status column in the DM Objects lists and as background color in the cell that shows the Status of DM Objects in the Graph Views.



It is possible to request a different number of accepted values for status control and also modify the mapping of values with colors. This is done by adding the **colors_for_accepted_values** keyword in the definition of the *Status* attribute in the `dm_structure.xml`, for the DM Object Types of interest. Taking as an example the *parts* definition below, the colors that are added, as shown in the picture below, are matched one by one with the corresponding accepted value e.g. gray for WIP, green for OK, etc.

```
<parts>
    <Properties>
        <Property name="Module Id" type="TEXT" allow_null="NO" read_only="NO"/>
        <Property name="Version" type="TEXT" allow_null="NO" read_only="NO"/>
        <Property name="Study Version" type="STUDY VERSION" default_value="0" allo
        <Property name="File Type" type="TEXT" default_value="ANSA" accepted_value
        <Property name="Representation" type="TEXT" allow_null="YES" read_only="NO
    </Properties>
    <Attributes>
        <Attribute name="Name" type="TEXT"/>
        <Attribute name="Status" type="TEXT" default_value="WIP"
        accepted_values="WIP,OK,Warning,Error,In Progress,Empty"
        colors_for_accepted_values="gray,green,orange,red,magenta,white"/>
    </Attributes>
</parts>
```

Note that the colors should be among those included in the image library of ANSA, META and KOMVOS, as shown below, in order for the DM Browsing tools to use the right icon automatically:



In any case, the icon used for the different Status values in the Data Browsing lists can also be configured in the `dm_views.xml`, as described in paragraph 5.2.1.6.

Finally, note that *Status* is the key attribute used for Lifecycle Management as described in paragraph *2.3*.

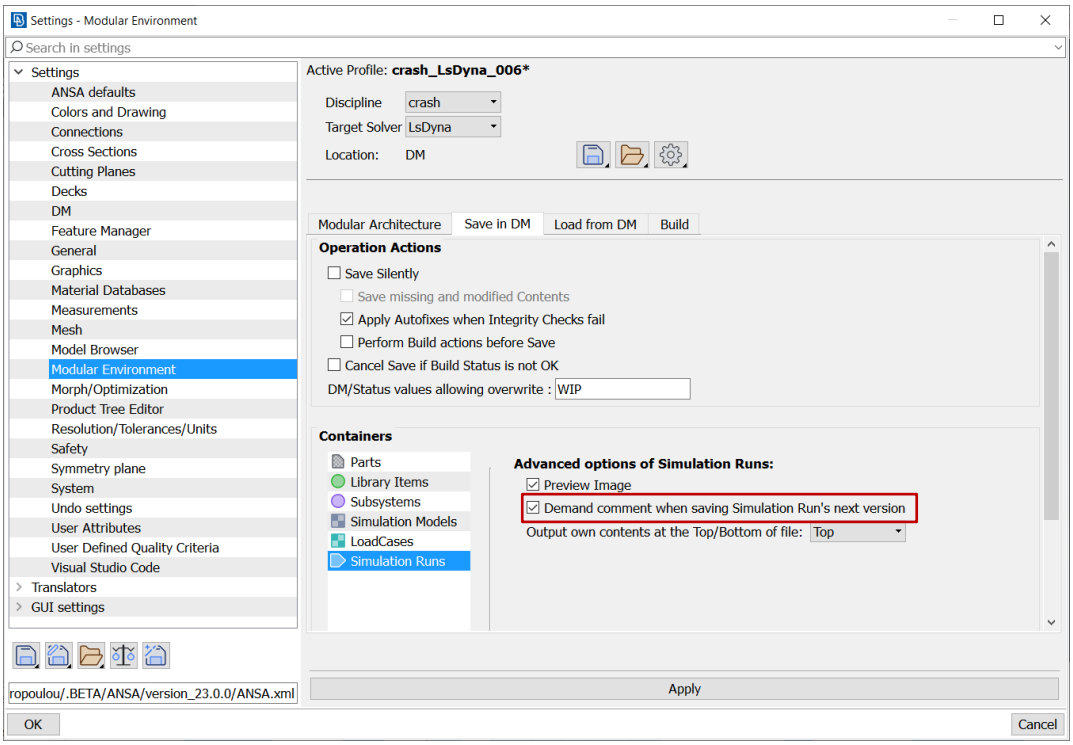
5.1.5.2. The “Comment” attribute

A Comment is a multi-line text description of a DM Object. In many cases, Comments are the only means to understand the engineering intention of the DM Object creator. Therefore, many engineering teams request the addition of Comments to be mandatory. Handling of Comments differs slightly for file-based and SPDRM DMs.

File-based DM

In file-based DMs, Comment does not need to be defined as an attribute of all DM Object Types in the dm_structure.xml, as ANSA handles it as such by default.

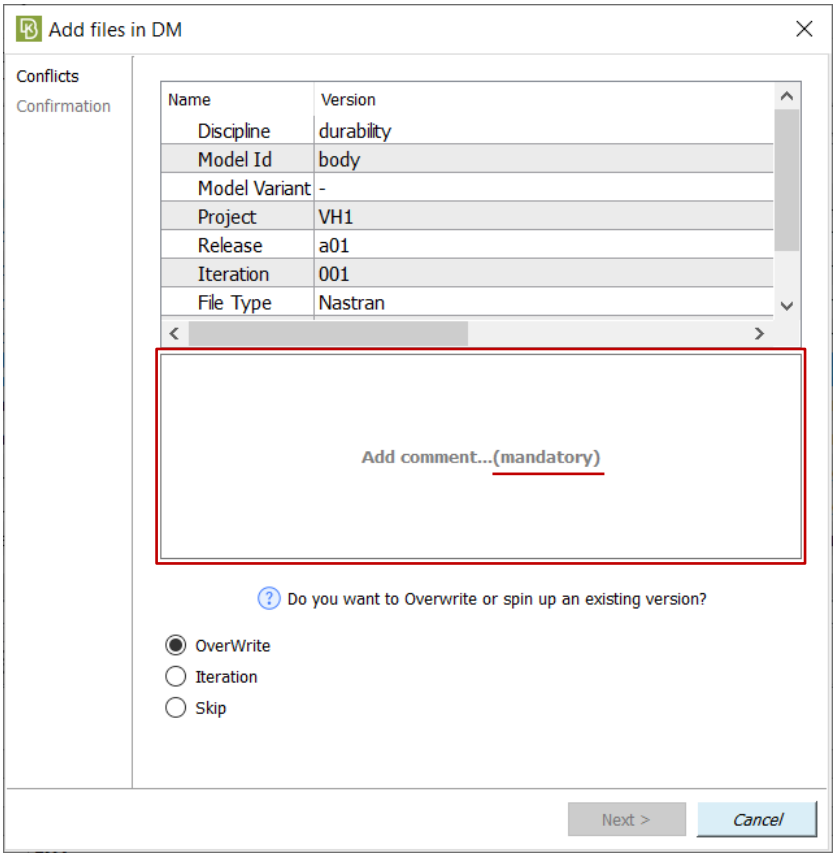
It is possible to request the addition of a comment every time a new version of a DM Object is about to be created through the Model Browser or the DM Browser in ANSA, using the Modular Environment setting “**Demand comment when saving** *<Model Browser Container Name>* **next version**” in ANSA, found under *Settings > Modular Environment > Save in DM > Containers*.



SPDRM back-end

For SPDRM back-end, the Comment needs to be defined as an attribute of all DM Object Types in the data model xml file. The special xml attribute `commitComment` can be used with the value `YES` to denote that the addition of a comment is mandatory when creating a new version of a DM Object or when overwriting an existing item.

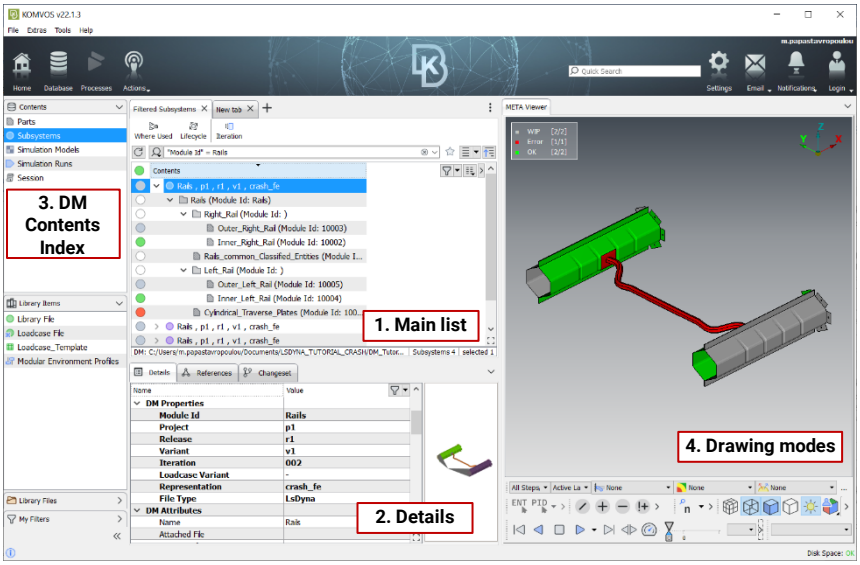
```
<Attributes>
  <Attribute name="Name" type="TEXT" generated_value="YES" />
  <Attribute name="Status" type="TEXT" default_value="WIP" accepted_values="$_Status_" />
  <Attribute name="Attached File" type="ATTACHED_FILE" />
  <Attribute name="Comment" type="TEXT" multirow="YES" commitComment="YES"/>
</Attributes>
```



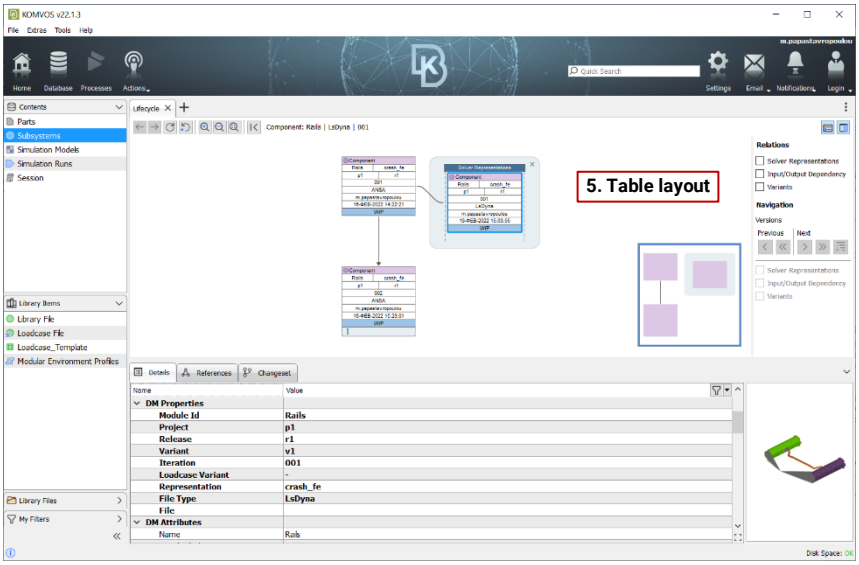
5.2. Graphical User Interface

The Graphical User Interface of the Data Browsing tools of ANSA/META and KOMVOS is highly customizable, allowing its adaptation to the needs of different teams and workflows. Present customization options can be used in order to configure:

1. The main lists used for data browsing
2. The bottom tab with the Details of the selected item
3. The labels and icons used in the DM contents index pane on the left
4. The drawing modes of the embedded 3D model Viewer
5. The layout of the table used to represent a DM Object in the Graph Views (SPDRM back-end only)



Items 1 through 4 are customized with the aid of a configuration xml file named `dm_views.xml`.



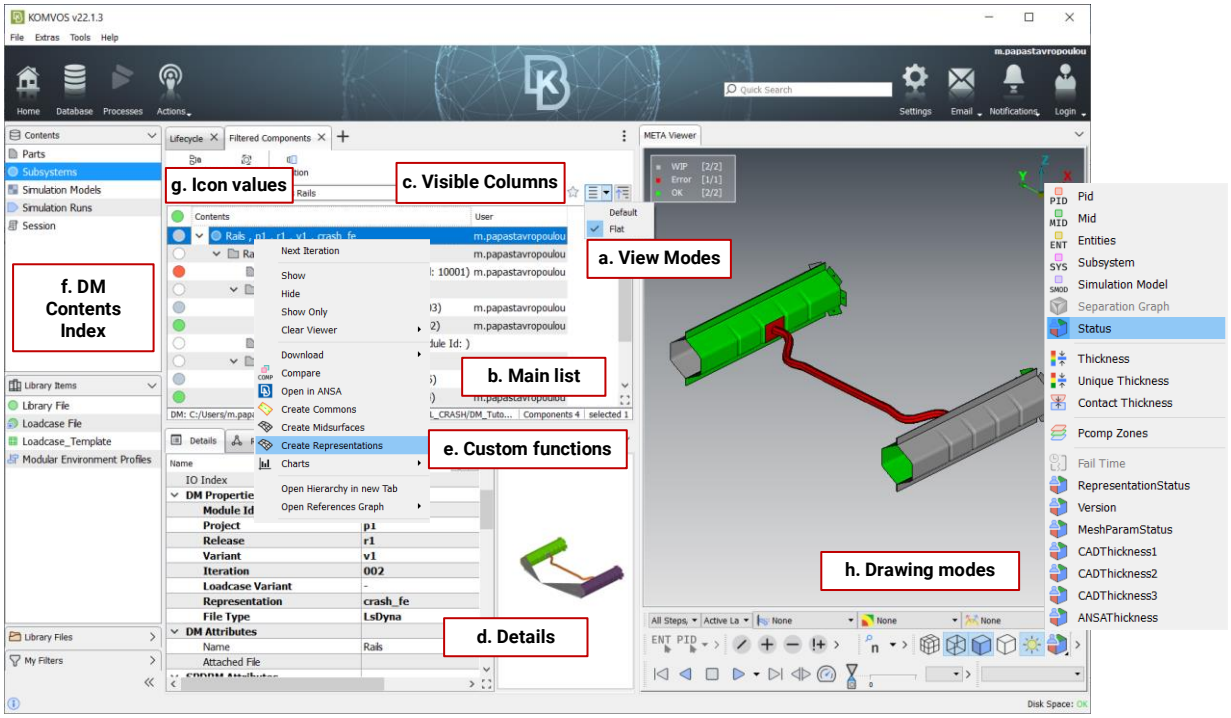
Item 5 is customized SPDRM server-side, with the aid of a configuration xml file named `data_views.xml`.

Information on the customization possibilities is given in the paragraphs below.

5.2.1. GUI customization with the dm_views.xml file

The GUI characteristics that can be configured through the `dm_views.xml` are:

a. The alternative Views available for each DM Object Type in the **View Modes** menu
b. The structure of the main list in each view, i.e. if it will be a flat list or a tree list. If it will be a tree list, how many and which grouping levels will be displayed
c. The default visible columns in the main list in each view
d. The attributes that will be displayed in the *Details* tab, their order and structure.
e. Custom functions that will be available in the context menu of DM Objects and will execute Python scripts
f. The DM Object Type labels and icons that will be displayed in the DM contents index pane of the Data Browsing tools
g. Whether the value of an object will be displayed in the respective column as text or with an icon (only applicable for attributes with a pre-defined list of accepted values)
h. The drawing modes available in the embedded Viewer, for the coloring of different items based on the values of certain attributes



5.2.1.1. Location and reading of the dm_views.xml

ANSA, META and KOMVOS can all read the same `dm_views.xml` in order to ensure identical presentation of information from all data browsing tools (DM Browser in ANSA/META and Data Workspace in KOMVOS).

Configuration in ANSA/META

The default layout of the *DM Browser* is hard-coded. A sample `dm_views.xml` that corresponds to the hard-coded views of the *DM Browser* can be found in the *`config`* directory in the installation of ANSA.

For ANSA/META to read any custom `dm_views.xml` file, the absolute path of the directory that contains it must be set in the **DM Configurations folder** setting found under *Tools > Settings > DM > DM Browser* tab. Any custom icons or scripts possibly referenced in the `dm_views.xml` should be placed within the same directory.

The DM Configuration folder can be configured during start-up by setting the variable `DM_configurations_folder` of the ANSA.defaults (or META.defaults). The `dm_views.xml` file is read during ANSA/META start-up. If any changes are performed to the file while the ANSA or META session is running, the user

must reload the file for the changes to take effect. This is possible through the **Utilities > Reload dm_views.xml** option located at the **Main Menu** button ⋮ of the *DM Browser*.

Configuration in KOMVOS

The default layout of the *Data Workspace* in KOMVOS is the one read from the `dm_views.xml` file located in the directory indicated by the **SDM_CONSOLE_HOME** environmental variable (by default, this is the *`config`* folder of the KOMVOS installation directory). This variable can either be set as an environmental variable in the system or can be defined in the start script of KOMVOS.

The `dm_views.xml` file is read during KOMVOS start-up. If any changes are performed to the file while the KOMVOS session is running, the user must reload the file for the changes to take effect. This is possible through the main menu-bar, **Extras > Reload dm_views.xml** option.

5.2.1.2. Structure of the `dm_views.xml` file

The main structure of the xml file is shown below:

File structure	
⊟	XMLData
	⊞ alternative_type_names
	⊞ view
	⊞ Gui_settings
	⊞ Right_click_actions
	⊞ value_image_maps
	⊞ General_settings
	⊞ Custom_attributes_drawing
	⊞ attribute_value_nick_names

The **alternative_type_names** element occurs once and is used to control the labels that will be used for each DM Object Type in the *Contents Index* list (paragraph 5.2.1.7)

The **view** element controls the structure and layout of the main list and occurs as many times as the defined views. So, to customize the default views of Subsystems (Default, Flat, Lifecycle), Simulation Models (Default, Flat, Hierarchy) and Simulation Runs (Flat, Tree) one would need 3 + 3 + 2 = 8 view elements (paragraph 5.2.1.3).

The **GUI_settings** element occurs once and is used to control the content and layout of the *Details* tab as well as the default visible columns in the main list (paragraph 5.2.1.4).

The **Right_click_actions** element occurs once and is used to control the custom actions that will appear in the context menu of a listed item (paragraph 5.2.1.5).

The **value_image_maps** element occurs once and is used to create a custom column with an image for each of the accepted values of an attribute (paragraph 5.2.1.6).

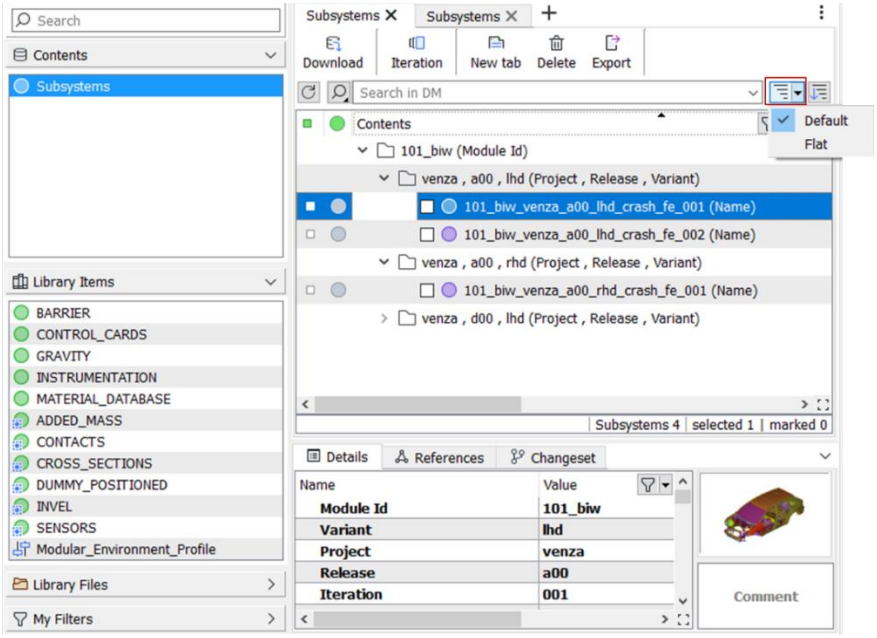
The **General_settings** element occurs once and is used to control settings of general purposes (paragraph 5.2.1.7).

The **Custom_attributes_drawing** element occurs once and is used to create a new drawing mode (Model coloring) in the **Styles** toolbar of the META Viewer of KOMVOS (paragraph 5.2.1.9).

The **attribute_value_nick_names** element occurs once and is used to map an attribute's value to a different label in the list of DM Browser of ANSA/META or Database Workspace of KOMVOS (paragraph 5.2.1.10).

5.2.1.3. Defining a View in the main list

The structure and layout of the information displayed in the Data Browsing tools of ANSA/META and KOMVOS is organized in different “views”. Each DM Object Type (e.g. Subsystems, Simulation Models, Simulation Runs, etc.) has its own views that are accessible through the **View Modes** button as shown in the image below.



The default available views per DM Object Type are listed in the table below:

DM Object Type	Default	Flat	Tree / Hierarchy	Simplified	Lifecycle	Per Sim.Model	Per Loadcase	Optimization	Groups
parts	✓	✓							
Subsystems	✓	✓			✓				
Library Items	✓	✓							
Simulation Model	✓	✓	✓	✓ (1)				✓	
Loadcase Template	✓	✓	✓						
Simulation Run	✓	✓	✓			✓ (1)	✓ (1)		
Predictors	✓	✓							✓

(1) Available in KOMVOS, only when connected to SPDRM

Some more information on the main views is given below:

- **Default**: DM Objects are presented in a tree view, where each grouping level corresponds to one or more properties / attributes of the DM Object contained. This view is available for all DM Object Types.
- **Flat**: DM Objects are presented in a flat list. This view is available for all DM Object Types.
- **Tree** or **Hierarchy**: DM Objects are presented in a tree view like that of the Default view. However, the DM Object itself can be further expanded to reveal its hierarchy. The hierarchy of a Subsystem is its parts structure. For a Simulation Model, it's the list of Subsystems and Library Items it contains. For a Simulation Run, its Simulation Model and Loadcase. This view is available for Simulation Models, Simulation Runs and Loadcase Templates.
- **Lifecycle:** DM Objects are presented in a tree view, where the relationships between the different Subsystem iterations can be seen. For each Subsystem iteration, its child in the lifecycle is shown as a child item in the tree view.

The definition of a **view** is shown below:

File structure	Sample
XMLData view view_name entry_type sdm_system tab_settings_index node	<view> <view_name>Default</view_name> <entry_type>Subsystems</entry_type> <sdm_system>AnsaDM</sdm_system> <tab_settings_index>Subsystems_Default </tab_settings_index> <node> </view>

The **view_name** key is required and defines the name of the view that will be displayed in the **View Modes** button of the *DM Browser*. The built-in view names are *Default, Flat, Tree* and *Hierarchy*. These views can be customized through the xml but cannot be removed.

The **entry_type** key is required and defines the DM Object Type for which the view is defined. The type should be given with the exact same spelling as the DM Object Types are defined in the DM Schema. E.g., Subsystems, parts, LoadCase, Library_File, etc.

The **sdm_system** key is required and defines the DM back-end that this view is applicable to. The accepted values are AnsaDM, SPDRM and SimManager for file-based DM, SPDRM and MSC SimManager respectively. This setting makes it possible for the same dm_views.xml to hold instructions for the customization of the views for more than one back-end types.

The **tab_settings_index** key is optional and enables the control of the default visible columns and the structure and content of the *Details* tab. The value of this key is the name of a *Tab* node defined in the **GUI_settings** main node.

Finally, the **node** element is required and it defines the structure and layout of the main list. A node element corresponds to a row in the list. Therefore, to define a tree structure like the *Default* view used for Subsystems, nested node elements are required to represent each different grouping level.

There are different options for the tuning of a node according to its type. The node type is defined in the **view_item** key. There are 2 main node types:

Subfolder: A node of this type represents a subfolder for data grouping and essentially triggers the creation of a tree list view.

Data_File: A node of this type represents the actual DM Object.

Thus, a tree list view is essentially constructed by several nested Subfolder nodes that finally contain a Data_File node while a flat list only consists of a Data_File node.

Configuring a Subfolder node

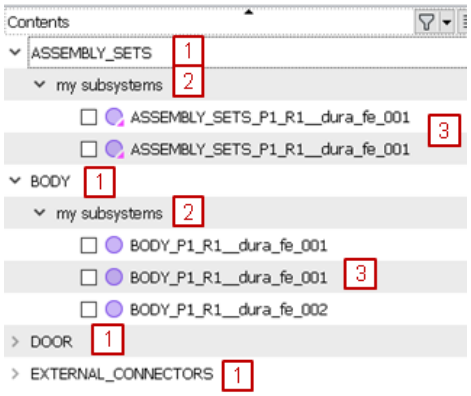
Subfolders are used for data grouping. There are 2 sub-types of Subfolder nodes. The Subfolder sub-type is defined in the **value_type** key:

Attribute_Label: The label of a subfolder of this type is an attribute of the DM Object to be displayed.

Fixed_Label: The label of a subfolder of this type is a fixed text.

The example below shows a tree list view for Subsystems with 2 grouping levels. The top level uses the *Module Id* attribute and the second level is the fixed label “my subsystems”.

```
<view>
    <view_name>Default</view_name>
    <entry_type>Subsystems</entry_type>
    <sdm_system>AnsaDM</sdm_system>
    <tab_settings_index>Subsystems_Default
    </tab_settings_index>
    <node>
        <value_type>Attribute_Label</value_type>
    1   <attribute_field>Module Id</attribute_field>
        <view_item>Subfolder</view_item>
        <node>
            <value_type>Fixed_Label</value_type>
        2   <fixed_field>my subsystems</fixed_field>
            <view_item>Subfolder</view_item>
            <node>
                <view_item>Data_File</view_item>
            3   <value_type>Attribute_Label</value_type>
                <attribute_field>Name</attribute_field>
            </node>
        </node>
    </node>
</view>
```



The available settings for the tuning of **Subfolder** nodes are shown below:

File structure

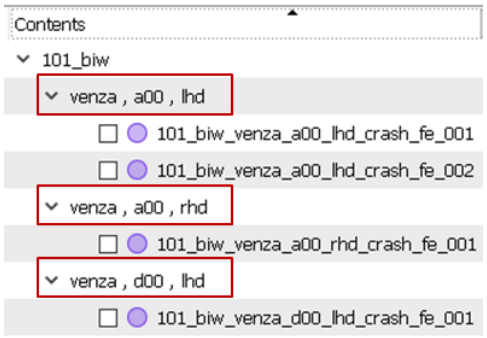
⊟ XMLData
⊟ View
⊟ Node
view_item = Subfolder
value_type
attribute_field
fixed_field
image
use_detailed_label
actions_index
⊞ node

The **value_type** is required and defines the sub-type of the Subfolder node. It can be one of `Attribute_Label` or `Fixed_Label`.

For Subfolder nodes of sub-type `Attribute_Label`, the **attribute_field** key is required, that defines the attribute whose value should be used for grouping. More than one attribute_field keys can be defined, in order to consolidate more attributes in one grouping level.

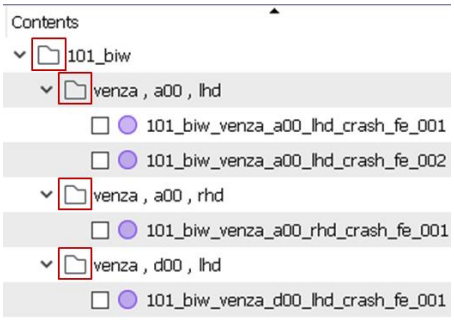
For Subfolder nodes of sub-type `Fixed_Label`, the **fixed_field** key is required, that defines the fixed text to be used.

```
<view>
    <view_name>Default</view_name>
    <entry_type>Subsystems</entry_type>
    <sdm_system>AnsaDM</sdm_system>
    <tab_settings_index>Subsystems_Default</tab_settings_index>
    <node>
        <value_type>Attribute_Label</value_type>
        <attribute_field>Module Id</attribute_field>
        <view_item>Subfolder</view_item>
        <node>
            <value_type>Attribute_Label</value_type>
            <attribute_field>Project</attribute_field>
            <attribute_field>Release</attribute_field>
            <attribute_field>Variant</attribute_field>
            <view_item>Subfolder</view_item>
            <node>
                <view_item>Data_File</view_item>
                <value_type>Attribute_Label</value_type>
                <attribute_field>Name</attribute_field>
            </node>
        </node>
    </node>
</view>
```



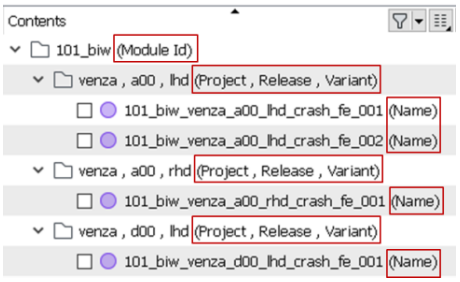
The **image** key is optional and can be used to display an icon next to the group label. It can either be the name of one of the embedded icons of the BETA Suite or the file name of a custom icon that should be placed within the DM Configurations folder**.**

```
<view>
    <view_name>Default</view_name>
    <entry_type>Subsystems</entry_type>
    <sdm_system>AnsaDM</sdm_system>
    <tab_settings_index>Subsystems_Default</tab_settings_index>
    <node>
        <value_type>Attribute_Label</value_type>
        <attribute_field>Module Id</attribute_field>
        <view_item>Subfolder</view_item>
        <image>mb_group_empty_small.svg</image>
        <node>
            <value_type>Attribute_Label</value_type>
            <attribute_field>Project</attribute_field>
            <attribute_field>Release</attribute_field>
            <attribute_field>Variant</attribute_field>
            <view_item>Subfolder</view_item>
            <image>mb_group_empty_small.svg</image>
            <node>
                <view_item>Data_File</view_item>
                <value_type>Attribute_Label</value_type>
                <attribute_field>Name</attribute_field>
            </node>
        </node>
    </node>
</view>
```



The **use_detailed_label** key is optional and triggers the display of the Attribute Name used for grouping next to each grouping level. To activate it, use it with the value **true.**

```
<view>
    <view_name>Default</view_name>
    <entry_type>Subsystems</entry_type>
    <sdm_system>AnsaDM</sdm_system>
    <tab_settings_index>Subsystems_Default</tab_settings_index>
    <node>
        <value_type>Attribute_Label</value_type>
        <attribute_field>Module Id</attribute_field>
        <view_item>Subfolder</view_item>
        <image>mb_group_empty_small.svg</image>
        <use_detailed_label>true</use_detailed_label>
        <node>
            <value_type>Attribute_Label</value_type>
            <attribute_field>Project</attribute_field>
            <attribute_field>Release</attribute_field>
            <attribute_field>Variant</attribute_field>
            <view_item>Subfolder</view_item>
            <image>mb_group_empty_small.svg</image>
            <use_detailed_label>true</use_detailed_label>
            <node>
                <view_item>Data_File</view_item>
                <value_type>Attribute_Label</value_type>
                <attribute_field>Name</attribute_field>
                <use_detailed_label>true</use_detailed_label>
            </node>
        </node>
    </node>
</view>
```



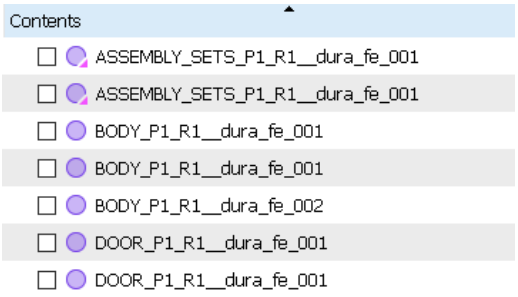
The **actions_index** key is optional and enables the addition of custom actions in the context menu of the Subfolder item. Custom actions are defined in the **Right_click_actions** element.

Configuring a Data_File node

Data_File nodes represent the actual DM Object being listed. In a view for Subsystems, the Data_File node will represent the actual Subsystem whereas in a view for parts, the Data_File node will represent the part. The corresponding list item is checkable, allowing it to be marked and downloaded.

The example below shows the configuration for the flat view of Subsystems.

```
<view>
    <view_name>Flat</view_name>
    <entry_type>Subsystems</entry_type>
    <sdm_system>AnsaDM</sdm_system>
    <tab_settings_index>Subsystems_Default</tab_settings_index>
    <node>
        <view_item>Data_File</view_item>
        <value_type>Attribute_Label</value_type>
        <attribute_field>Name</attribute_field>
    </node>
</view>
```



The available settings for the tuning of **Data_File** nodes are shown below:

File structure

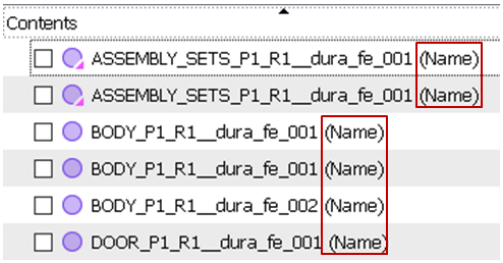
```
XMLData
    view
        node
            view_item = Data_File
            value_type
            attribute_field
            image
            use_detailed_label
            show_in_parenthesis
            actions_index
            node
```

The **value_type** key is required and determines the label of the Data_File list item. It can only get the value `Attribute_Label`, to indicate that the label of the list item will be the value of an attribute of the DM Object. The attribute that will be used is defined in the **attribute_field** key.

The **image** key is optional and can be used in order to display an icon different than the default icon of the DM Object Type. It can either be the name of one of the embedded icons of the BETA Suite or the file name of a custom icon that should be placed within the DM Configurations folder**.**

The **use_detailed_label** key is optional and triggers the display of the Attribute Name used for labeling the list item. To activate it, use it with the value **true.**

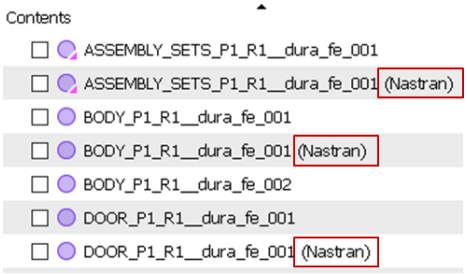
```
<view>
    <view_name>Flat</view_name>
    <entry_type>Subsystems</entry_type>
    <sdm_system>AnsaDM</sdm_system>
    <tab_settings_index>Subsystems_Default</tab_settin
    <node>
        <view_item>Data_File</view_item>
        <value_type>Attribute_Label</value_type>
        <attribute_field>Name</attribute_field>
        <use_detailed_label>true</use_detailed_label>
    </node>
</view>
```



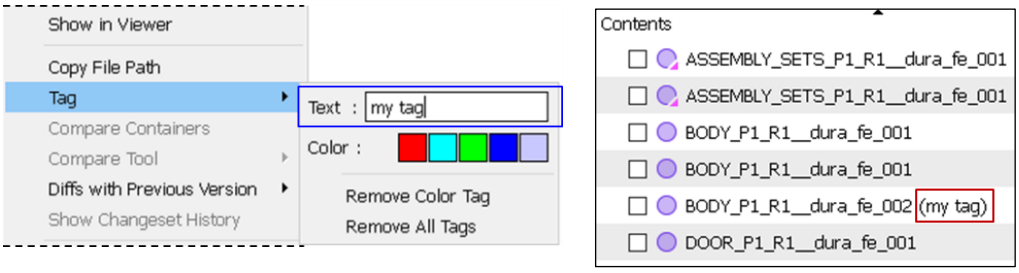
The **show_in_parenthesis** key is optional and triggers the display of another attribute of the DM Object within the parenthesis next to the primary label of the listed item. In the example below, the File Type attribute is displayed.

Note that this setting cannot be combined with the **use_detailed_label** setting.

```
<view>
    <view_name>Flat</view_name>
    <entry_type>Subsystems</entry_type>
    <sdm_system>AnsaDM</sdm_system>
    <tab_settings_index>Subsystems_Default</tab_settings_index>
    <node>
        <view_item>Data_File</view_item>
        <value_type>Attribute_Label</value_type>
        <attribute_field>Name</attribute_field>
        <show_in_parenthesis>File Type</show_in_parenthesis>
    </node>
</view>
```



The **show in parenthesis** key also accepts the built-in value `Text Tag` that will display the text of any tag added through the context menu option **Tag**.



Finally, the **actions_index** key is optional and enables the addition of custom actions in the context menu of the listed item. Custom actions are defined in the **Right_click_actions** element.

The view of a Data_File node can be further tuned to reveal more characteristics of a DM Object. For example, a Subsystem could be expanded to display its Parts Structure (hierarchy) or a Simulation Run could be expanded to display its contained Reports. The display of such characteristics of the Data_File nodes can be enabled with the addition of nodes of specific types within the Data_File nodes.

There are 5 specific node types that can be added under Data_File nodes (remember that the type of a node is declared in its **view_item** key).

1. Hierarchy: A node of this type triggers the display of the hierarchy of the listed DM Object. As hierarchy:

- Subsystems show their Parts Structure.
- Subsystem Groups show their Subsystem and Parts Structure.
- Simulation Models and Loadcase_Templates show their contained Subsystems and Library Items
- Simulation Runs show their contained Simulation Model and LoadCase

2. Attached File: A node of this type triggers the display of files attached to the DM Object as child items.

3. Attached Folder: A node of this type triggers the display of files/folders attached to the DM Object as child items.

4. Containing_Item: A node of this type triggers the display of the contained items of a DM Object as child items. The contained items of DM Objects are those listed in the Containing Items tab of the *DM Schema Editor*. In the default data model (dm_structure.xml) such relationships are defined among simulation data. More specifically:

- A Simulation Model contains Loadcases and may also contain Reports and Sessions
- A Loadcase contains Simulation Runs, and may also contain Reports and Sessions
- A Simulation Run contains Reports and may also contain Sessions

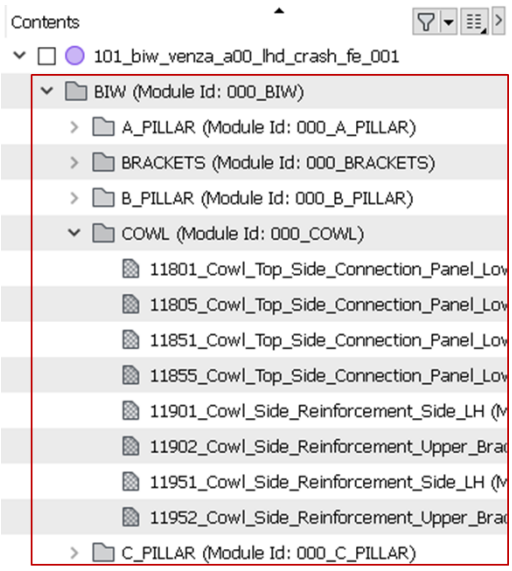
Thus, a node of this type will enable the display of the Reports under a Simulation Run or the display of the adapted Loadcases and Runs under a Simulation Model.

5. Containing_Data_File: This node type is very similar to the **Containing_Item** type above. The only difference is that the listed item will be checkable, allowing it to be marked and downloaded.

Data_File node > Hierarchy sub-node

The example below shows the set-up required for the display of hierarchy under Subsystems. Notice that although the Subsystems are still presented in a flat view, they can now be further expanded in order to reveal their Parts Structure.

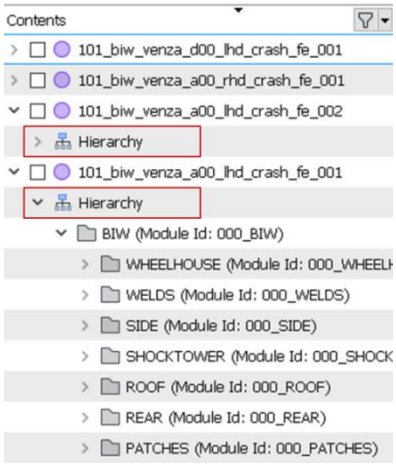
```
<view>
    <view_name>Flat</view_name>
    <entry_type>Subsystems</entry_type>
    <sdm_system>AnsaDM</sdm_system>
    <tab_settings_index>Subsystems_Default</tab_settings_index>
    <node>
        <view_item>Data_File</view_item>
        <value_type>Attribute_Label</value_type>
        <attribute_field>Name</attribute_field>
        <node>
            <view_item>Hierarchy</view_item>
            <value_type>Hierarchy_Label</value_type>
        </node>
    </node>
</view>
```



! **NOTE:** Nodes of **view_type** `Hierarchy` must necessarily have **value_type** `Hierarchy_Label`.

This view can be further improved by adding an additional grouping level to hold the hierarchy:

```
<view>
    <view_name>Flat</view_name>
    <entry_type>Subsystems</entry_type>
    <sdm_system>AnsaDM</sdm_system>
    <tab_settings_index>Subsystems_Default</tab_settings_index>
    <node>
        <view_item>Data_File</view_item>
        <value_type>Attribute_Label</value_type>
        <attribute_field>Name</attribute_field>
        <node>
            <view_item>Subfolder</view_item>
            <value_type>Fixed_Label</value_type>
            <fixed_field>Hierarchy</fixed_field>
            <image>hierarchy_small.svg</image>
            <node>
                <view_item>Hierarchy</view_item>
                <value_type>Hierarchy_Label</value_type>
            </node>
        </node>
    </node>
</view>
```



The available settings for the tuning of **Hierarchy** nodes are shown below:

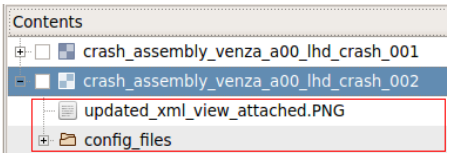
File structure

XMLData
view
node
view_item = Data_File
…
node
view_item = Hierarchy
value_type = Hierarchy_Label
entry_type
preview_image_exists
actions_index
tab_settings_index

Data_File node > Attached_File/Folder sub-node

The example below shows the set-up required for the display of an attached file/folder under Simulation Models. Notice that although the Simulation Models are still presented in a Flat view, they can now be further expanded to reveal their attachments.

```
<view>
    <view_name>Flat</view_name>
    <entry_type>Simulation_Model</entry_type>
    <sdm_system>AnsaDM</sdm_system>
    <tab_settings_index>Simulation_Model_Default</tab_s
    <node>
        <value_type>Attribute_Label</value_type>
        <attribute_field>Name</attribute_field>
        <view_item>Data_File</view_item>
        <image>mb_sim_model_full_small.svg</image>
        <node>
            <view_item>Attached_Folder</view_item>
            <value_type>Attached_Folder</value_type>
            <image>folder_small.svg</image>
        </node>
        <node>
            <view_item>Attached_File</view_item>
            <value_type>Attached_File</value_type>
            <image>mb_file_small.svg</image>
        </node>
    </node>
</view>
```



The **value_type** is required and defines the type of the attachment. It can be one of `Atached_File` or `Attached_Folder`.

There are 2 node types (remember that the type of a node is declared in its **view_item** key). The available settings for the tuning of Attached File/Folder nodes are shown below:

File structure	
XMLData view node view_item = Data_File … node value_type = Attached File view_item = Attached File image actions_index	**view_item = Attached_File**: will trigger the display of the Attached Files only, plain, directly under the DM Object. Any attached folders will not be displayed.
File structure	**view_item = Attached_Folder**: will trigger the display of both Attached Files and Attached Folders, each within a folder that has as folder name the name of the attribute being used respectively.
XMLData view node view_item = Data_File … node value_type = Attached Folder view_item = Attached Folder image actions_index	

! **NOTE:** It is recommended to use both nodes described above because it displays both attached folders and attached files with the plus that the attached files are displayed once, as plain files, directly under the object.

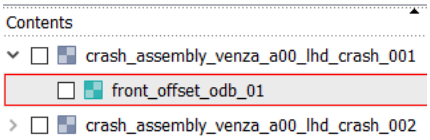
The **image** key is optional and can be used in order to display an icon different than the default icon of the DM Object Type. It can either be the name of one of the embedded icons of the BETA Suite or the file name of a custom icon that should be placed within the DM Configurations folder.

Finally, the **actions_index** key is optional and enables the addition of custom actions in the context menu of the listed item. Custom actions are defined in the **Right_click_actions** element.

Data_File node > Contained_Data_File sub-node

The example below shows the set-up required for the display of the contained items of a DM Object as checkable list items, allowing them to be marked and downloaded.

```
<view>
    <view_name>Flat</view_name>
    <entry_type>Simulation_Model</entry_type>
    <sdm_system>AnsaDM</sdm_system>
    <tab_settings_index>Simulation_Model_Default</tab_settings_index>
    <node>
        <value_type>Attribute_Label</value_type>
        <attribute_field>Name</attribute_field>
        <view_item>Data_File</view_item>
        <image>mb_sim_model_full_small.svg</image>
        <node>
            <value_type>Attribute_Label</value_type>
            <attribute_field>Name</attribute_field>
            <view_item>Contained_Data_File</view_item>
            <entry_type>LoadCase</entry_type>
            <image>mb_loadcase_full_small.svg</image>
            <tab_settings_index>LoadCase_Default</tab_settings_index>
        </node>
    </node>
</view>
```



The available settings for the tuning of **Containing_Data_File** nodes are shown below:

File structure
XMLData
view
node
view_item = Data_File
…
node
view_item = Conataing_Data_File
value_type = Attribute_Label
attribute_field
entry_type
actions_index
tab_settings_index
show_in_parenthesis

The **value_type** key is required and determines the label of the Data_File list item. It can only get the value `Attribute_Label`, to indicate that the label of the list item will be the value of an attribute of the DM Object. The attribute that will be used is defined in the **attribute_field** key.

The **entry_type** key is required and defines the DM Object Type for which the sub-node is defined. The type should be given with the exact same spelling as the DM Object Types are defined in the DM Schema. E.g. Subsystems, parts, LoadCase, Library_File etc.

The **tab_settings_index** key is optional and enables the control of the default visible columns and the structure and content of the Details tab for the containing data file. The value of this key is the name of a *Tab* element defined in the **GUI_settings** main element.

Finally, the **actions_index** key is optional and enables the addition of custom actions in the context menu of the listed item. Custom actions are defined in the **Right_click_actions** element.

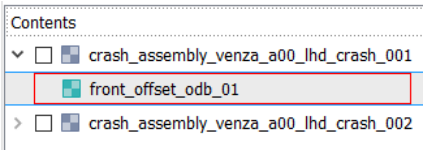
! **NOTE:** The contained items (DM Objects) of a DM Object should not be confused with the DM object's hierarchy. The contained items of DM Objects are those listed in the Containing Items tab of the *DM Schema Editor* and represent the hierarchical structure of Modular Run Management data. More specifically:

- A Simulation Model contains LoadCases and Reports
- A LoadCase contains a Simulation Runs and Reports
- A Simulation Run contains Reports

Data_File node > Containing_Data_Item sub-node

The example below shows the set-up required for the display of a containing item as a child item to the DM Object listed. The contained items of DM objects are those listed in the Containing Items tab of the *DM Schema Editor*. The only difference from the Containing_Data_File is that the listed item will not be checkable. Note that since v21.0.0 it's not necessary for a DM Object to be marked (checked) to be downloaded, as downloading works directly on the selected items. Marking is a handier way for selecting multiple DM Objects for Download as multi-selection is not convenient at all times.

```
<view>
    <view_name>Flat</view_name>
    <entry_type>Simulation_Model</entry_type>
    <sdm_system>AnsaDM</sdm_system>
    <tab_settings_index>Simulation_Model_Default</tab_settings_index>
    <node>
        <value_type>Attribute_Label</value_type>
        <attribute_field>Name</attribute_field>
        <view_item>Data_File</view_item>
        <image>mb_sim_model_full_small.svg</image>
        <node>
            <value_type>Attribute_Label</value_type>
            <attribute_field>Name</attribute_field>
            <view_item>Containing_Data_Item</view_item>
            <entry_type>LoadCase</entry_type>
            <image>mb_loadcase_full_small.svg</image>
            <tab_settings_index>LoadCase_Default</tab_settings_index>
        </node>
    </node>
</view>
```

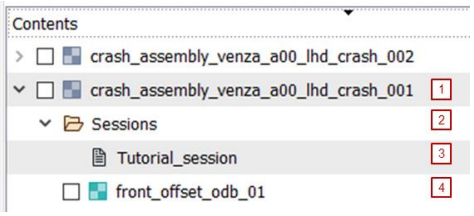


The available settings for the tuning of **Containing_Item** nodes are shown below:

File structure
XMLData
view
node
view_item = Data_File
…
node
view_item = Containing_Item
value_type = Attribute_Label
attribute_field
entry_type
tab_settings_index
show_in_parenthesis

In combination the two sub-nodes can be utilized to list more than one containing items under the DM Object as shown in the picture below:

```
<view>
    <view_name>Flat</view_name>
    <entry_type>Simulation_Model</entry_type>
    <sdm_system>AnsaDM</sdm_system>
    <tab_settings_index>Simulation_Model_Default</tab_settings_index>
    <node>
        <value_type>Attribute_Label</value_type>
1       <attribute_field>Name</attribute_field>
        <view_item>Data_File</view_item>
        <image>mb_sim_model_full_small.svg</image>
        <node>
            <value_type>Fixed_Label</value_type>
            <fixed_field>Sessions</fixed_field>
2           <view_item>Subfolder</view_item>
            <image>filemng_open_16.png</image>
            <node>
                <value_type>Attribute_Label</value_type>
                <attribute_field>Name</attribute_field>
3               <image>file_session_small.svg</image>
                <view_item>Containing_Item</view_item>
                <entry_type>Session</entry_type>
                <should_not_appear_if_empty>true</should_not_appear_if_empty>
            </node>
        </node>
        <node>
            <value_type>Attribute_Label</value_type>
            <attribute_field>Name</attribute_field>
4           <view_item>Containing_Data_File</view_item>
            <entry_type>LoadCase</entry_type>
            <image>mb_loadcase_full_small.svg</image>
            <tab_settings_index>LoadCase_Default</tab_settings_index>
        </node>
    </node>
</view>
```



5.2.1.4. Configuring the Details and default visible columns of a View

The **GUI_settings** element occurs once and is used to control the attributes that will be displayed in the *Details* tab, their order and structure, as well as the default visible columns in the main list.

The definition of a **GUI settings** element is shown below:

File structure	Sample
Gui_settings	
Tab	
tab_settings_index	
exactly_follow_details	
show_column	
show_detail	
object_type	
details_group	
details_group_name	
details	
show_detail	

```
<Tab>
    <tab_settings_index>Subsystems_Default</tab_settings_index>
    <exactly_follow_details>true</exactly_follow_details>
    <show_column>Owner</show_column>
    <show_detail>Name</show_detail>
    <object_type>Subsystems<object_type>
    <details_group>
        <details_group>
            <details_group_name>DM Properties</details_group_name>
            <details>
                <show_detail>Module Id</show_detail>
                <show_detail>Project</show_detail>
                <show_detail>Release</show_detail>
                <show_detail>Variant</show_detail>
                <show_detail>Iteration</show_detail>
                <show_detail>Loadcase Variant</show_detail>
                <show_detail>Representation</show_detail>
                <show_detail>File Type</show_detail>
                <show_detail>File</show_detail>
            </details>
        </details_group>
    </details_group>
</Tab>
```

The `Gui_settings` element consists of one or more tabs. Every **Tab** has a characteristic name defined by the **tab_settings_index** key. As we have already seen, the **tab_settings_index** key is used for the definition of a view in order to link this view with a specific tab, but it could also be used inside an element with **view_type** `Data_File`, `Hierarchy`, `Containing_Data_File` or `Containing_Item` in order to customize the Details tab when the user selects a listed item.

The **object_type** element associates this Tab definition with a DM Object type, so that even in cases where the Details tab is populated after the selection of an item in a Graph View (and not in a list), the information in Details is presented

according to the instructions for the respective DM Object type. The type should be given with the exact same spelling as the DM Object Types are defined in the DM Schema. E.g., Subsystems, parts, LoadCase, Library_File, etc.

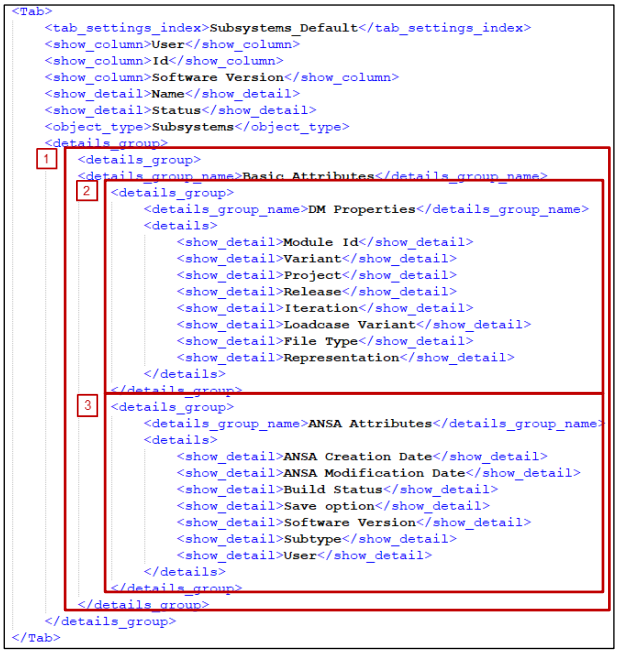
The **exactly_follow_details** element restricts the Details shown to exactly what's defined in the `dm_views.xml` file. When `true`, any attributes of the DM Object that are not listed in the Tab definition will not be displayed at all.

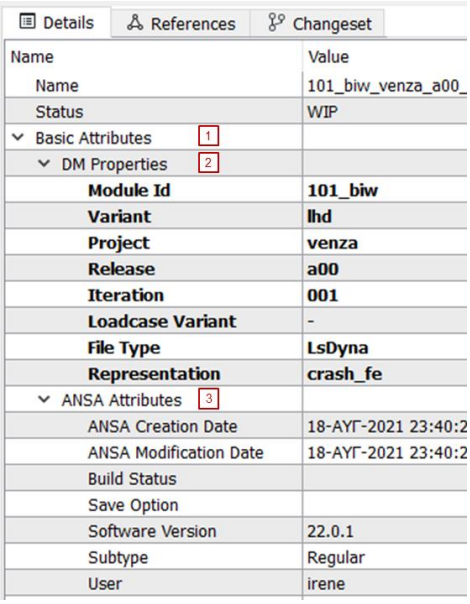
The **show_column** key specifies the visible columns for this tab. Note that this element only suggests the default visible columns, as the columns shown finally in the list may be the ones stored in the user's GUI customization file (*ANSA.xml*), which takes precedence over the `dm_views.xml`.

The **show_detail** key is used to display the attributes and properties of the DM Object in the given order.

The use of the **object_type** key on a particular **<Tab>** element indicates that the structure of the *Details* tab for a specified DM item type will be controlled by the keywords included in the element independently from the view modes of the *DM Browser*. To connect more DM item types with a specific *Details* tab structure, more object_type keywords can be added in the same <Tab> element.

It is possible to group the details utilizing the **details_group** key. To assign a name to a group we use the **details_group_name** key. Grouping of attributes is possible in multiple levels, by nesting a group under a details group as shown in the picture below.





Defining a Files/Folders GUI Settings element

To control the structure of the Details tab for plain Files and Folders, a new *Tab* element should be added, under GUI Settings element, with an **object_type** keyword value of **Files** or **Folders**, respectively.

```
<Tab>
    <tab_settings_index>Library Files</tab_settings_index>
    <object_type>Files</object_type>
    <exactly_follow_details>true</exactly_follow_details>
    <show_column>Id</show_column>
    <show_column>Owner</show_column>
    <details_group>
        <details_group>
            <details_group_name>Properties</details_group_name>
            <details>
                <show_detail>Name</show_detail>
            </details>
        </details_group>
        <details_group>
            <details_group_name>System Attributes</details_group_name>
            <details>
                <show_detail>User</show_detail>
                <show_detail>Size</show_detail>
            </details>
        </details_group>
        <details_group>
            <details_group_name>Path</details_group_name>
            <details>
                <show_detail>Is Folder</show_detail>
                <show_detail>Relative Path</show_detail>
            </details>
        </details_group>
    </details_group>
</Tab>
```

```
<Tab>
    <tab_settings_index>Library Folders</tab_settings_index>
    <object_type>Folders</object_type>
    <exactly_follow_details>true</exactly_follow_details>
    <show_column>Id</show_column>
    <show_column>Owner</show_column>
    <details_group>
        <details_group>
            <details_group_name>Properties</details_group_name>
            <details>
                <show_detail>Name</show_detail>
            </details>
        </details_group>
        <details_group>
            <details_group_name>System Attributes</details_group_name>
            <details>
                <show_detail>User</show_detail>
                <show_detail>Size</show_detail>
            </details>
        </details_group>
        <details_group>
            <details_group_name>Path</details_group_name>
            <details>
                <show_detail>Is Folder</show_detail>
                <show_detail>Relative Path</show_detail>
            </details>
        </details_group>
    </details_group>
</Tab>
```

This Tab is not linked with any view, so the **tab_settings_index_key** is only to give the Tab a characteristic name.

5.2.1.5. Defining custom right-click actions

The definition of a **Right_click_actions** element is shown below:

File structure	Sample
Right_click_actions	<Right_click_actions> <Action> <actions_index>Simulation_Run_Results_Actions</action <action_menu>My Actions</action_menu> <action_name>Action 1</action_name> <function_name>Action_Function</function_name> <script_file_path>Actions_File.py</script_file_path> <keep_alive>YES</keep_alive> <run_in_separate_process>NO</run_in_separate_process> <object_type>Simulation_Run</object_type> </Action> </Right_click_actions>
Action	
actions_index	
action_menu	
action_name	
function_name	
script_file_path	
keep_alive	
run_in_seperate_process	
object_type	

Every context menu action requires an **Action** element in the `dm_views.xml`.

An Action can be associated with the data either through the **actions_index** key or through the **object_type** key.

In order to make an action available for all DM Objects of a particular type independent of the view, the **object_type** key must be used, that associates this action with a particular DM Object Type. To make the same action available to more DM item types, more object_ type elements can be added in the same Action element.

In order to make an action available only to some particular views, the **actions_index** key can be used. The actions_index element holds a label that can be used to groups several actions and is then referenced in the actions_index optional key of the node elements of views.

The **action_name** key is used to define the action label that will appear in the context menu.

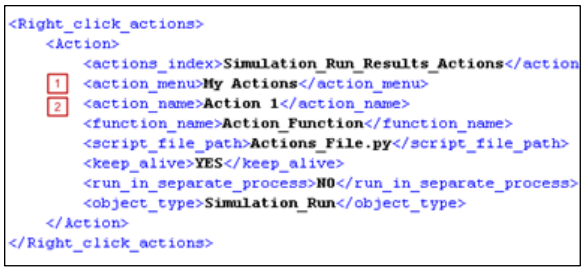
The **script_file_path** is the path to the script that should be expressed relatively to the DM Configuration folder or to the application home directory defined through the environmental variables `ANSA_HOME`, `META_HOME`, `SDM_CONSOLE_HOME` for ANSA, META and KOMVOS respectively.

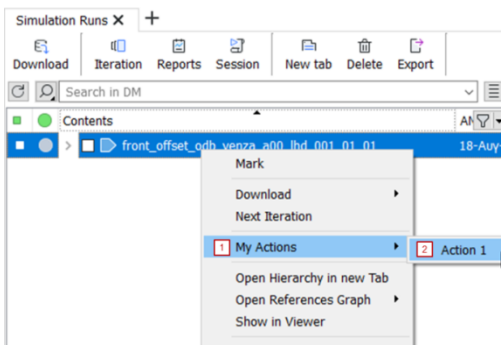
The **function_name** defines the name of the function to be executed.

The **action_menu** key is optional and creates a 2nd level grouping in the context menu of the item.

If the ***keep alive*** option is set to YES through the respective key, the action will be loaded only the first time it is executed, unless the user explicitly reloads the script (default value = NO).

The use of the **run_in_separate_process** key indicates that the action is performed in a separate nogui ANSA session (default value = NO).





Scripts defined as custom actions get as input argument a list that contains one dictionary for each selected DM Object. The sample script below prints the Status of the selected DM Object.

```
def custom_dm_action(sel_dm_objects_key_val):
    for dm_object_key_val in sel_dm_objects_key_val:
        server_id = dm_object_key_val.get("Id")
        if not server_id:
          continue
        dmo = dm.DMObject(server_id = server_id)
        print ('Status of DM item with id {}: {}'.format(server_id, dmo.get_attribute_values(attributes=['Status'])))
```

5.2.1.6. Requesting the use of an icon instead of text

When an attribute that accepts a closed list of values is added as a column in the main list, it is possible to map each of the accepted values to an image and display the image instead of text. This is achieved with the aid of the **value_image_maps** block.

The example below shows such a set up for the *File Type* column.

File structure	Sample
value_image_maps	<value_image_maps> <value_image_map name="File_Type"> <value_image_entry> <value>ANSA</value> <image>logo_ansa.svg</image>
value_image_map name	
value_image_entry	
value	
image	

The images are mapped to the accepted values with the use of the ***value_image_entry*** node. The **value** key corresponds to the accepted value and the ***image*** corresponds to the chosen image for the representation of this value.

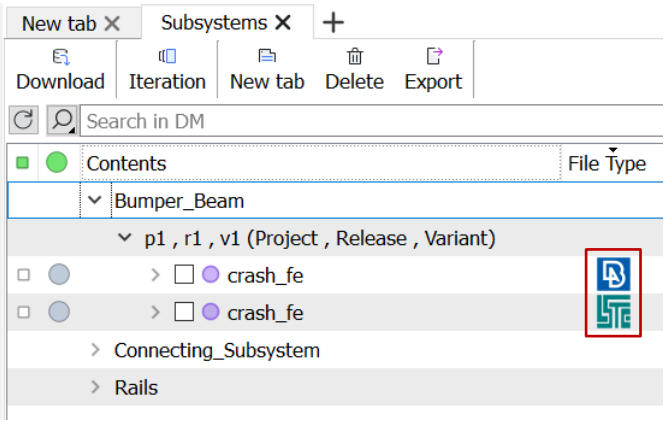
! NOTE: For the mapping of values with icons to be successful, in the **Gui_settings** element under the *Tab* element that correspond to the preferred DM Object Type, *e.g. Subsystem*, the attribute of choice, e.*g. File Type*, have to be marked with the **value_icons** keyword.

Gui_settings

```
<Gui_settings>
    <Tab>
        <tab_settings_index>Subsystems_Default</tab_settings_index>
        <exactly_follow_details>true</exactly_follow_details>
        <object_type>Subsystems</object_type>
        <show_column>DM Modification Date</show_column>
        <show_column value_icons="File_Type">File Type</show_column>
        <show_column>Owner</show_column>
        <show_column>Id</show_column>
        <show_detail>Name</show_detail>
        <show_detail>Status</show_detail>
        <show_detail>Comment</show_detail>
        <show_detail>Owner</show_detail>
        <show_detail>DM Modification Date</show_detail>
    </Tab>
```

value_image_maps

```
<value_image_maps>
    <value_image_map name="File_Type">
        <value_image_entry>
            <value>ANSA</value>
            <image>logo_ansa.svg</image>
        </value_image_entry>
        <value_image_entry>
            <value>Abaqus</value>
            <image>deck_abaqus.svg</image>
        </value_image_entry>
        <value_image_entry>
            <value>LsDyna</value>
            <image>deck_ls_dyna.svg</image>
        </value_image_entry>
    </value_image_map>
</value_image_maps>
```

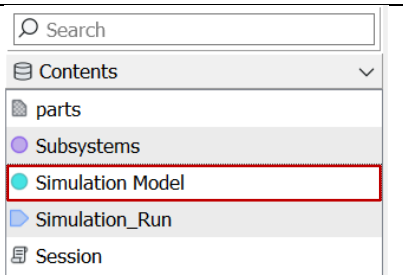


5.2.1.7. General Settings

Customizing the icon of DM Object Types

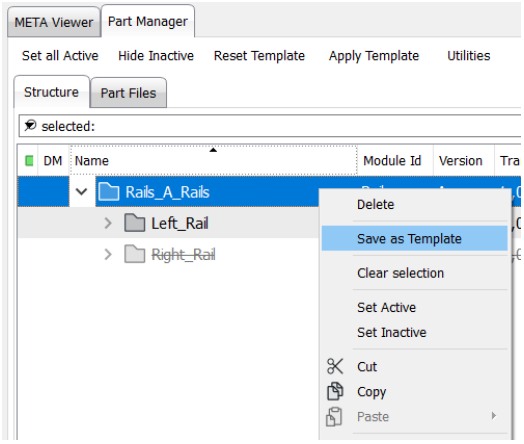
Customization of the icons of DM Object Types displayed in the DM Browser of ANSA/META or the Data Workspace of KOMVOS is possible through the xml element *General_settings*.

To change the default icon displayed for a type, the **Object_type_icons** element can be configured as shown below:

File structure	Sample	Result
General_settings Object_type_icons object_type icon	<General_settings> <Object_type_icons> <object_type>Simulation_Model</object_type> <icon>led_cyan_small.svg</icon> </Object_type_icons> </General_settings>	

Rules for saving XML Templates for Subsystems through KOMVOS

After reading a product definition (PLMXML or other) in the *Part Manager* of KOMVOS, it is possible to apply templates that will filter the CAD structure by activating or deactivating groups and parts in order to prepare it for use under a CAE Subsystem.



Such templates are usually created manually but can also be created based on filtering applied manually, by selecting groups/parts and activating/deactivating them with the context menu options Set Active/Set Inactive respectively. In the example on the left, the group "*Right_Rail*" of the CAD hierarchy was deactivated manually.

The **Save as Template** will save an XML template so that the filtering applied once manually can be re-applied later through the template (see paragraph 6.1.3 of KOMVOS User Guide).

The template generation process is tuned with some settings defined under the *General_settings* element.

These settings are tuned under the **SubmodelTemplateSetting** element, as shown below:

File structure	Sample
General_settings SubmodelTemplateSetting template_setting setting_name setting_value	<General_settings> <SubmodelTemplateSetting> <template_setting> <setting_name>search_in_attribute</setting_name> <setting_value>Name</setting_value> </template_setting> <template_setting> <setting_name>use_delimeter</setting_name> <setting_value>_</setting_value> </template_setting> <template_setting> <setting_name>section_index</setting_name> <setting_value>1</setting_value> </template_setting> </SubmodelTemplateSetting> </General_settings>

Based on these settings, the resulting template looks like this:

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

Based on the values of the "*search_in_attribute*", "*use_delimeter*" and "*section index*" keywords specified under the *SubmodelTemplateSetting* element of `dm_views.xml` another keyword named "*for_keyword*" is calculated. These keywords conclude the criteria responsible for the identification of items of the CAD hierarchy.

Upon applying the newly created template, through the *Apply Template* option of Part Manager, the functionality will search for the Top Item to be included as specified by the Action keyword (*Action = "Include"*) and a Child Item to be set as inactive (*Action = "Set_Inactive"*).

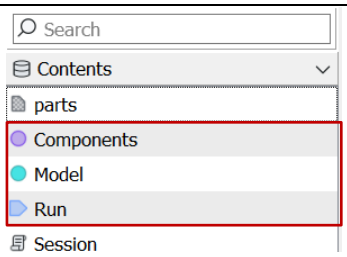
For the detection of items in the CAD hierarchy the following keywords are utilized accordingly:

- **search_in_attribute:** collects the value of the specified attribute
- **after_string:** collects only the section of the value that follows the specified string
- **for_keyword:** the value that identifies the item among the other of the CAD Hierarchy
- **use_delimeter:** the delimeter used along the attribute value and separates the value into sections
- **section_index:** the number of delimeters used in the attribute value
- **match_method:** the method that specifies if the for_*keyword* value should be an exact match with the value calculated by the other keywords. If the match method is "equals", the match should be exact. For values such as "*contains*" or "*ends_with*", the for_*keyword* value must be contained in the calculated string.

5.2.1.8. Customizing the name of DM Object Types of the DM Contents Index

Customization of the names of DM Object Types displayed in the *Contents Index* list of the DM Browser of ANSA/META or the Data Workspace of KOMVOS is possible through the xml element *alternative_type_names*.

To change the names displayed in the *Contents Index* list, the **alternative_type_names** element can be configured as shown below:

File structure	Sample	Result
alternative_type_names alternative_pair original_name alternative_name	<alternative_type_names> <alternative_pair> <original_name>Simulation_Model</original_name> <alternative_name>Model</alternative_name> </alternative_pair> <alternative_pair> <original_name>Simulation_Run</original_name> <alternative_name>Run</alternative_name> </alternative_pair> <alternative_pair> <original_name>Subsystems</original_name> <alternative_name>Components</alternative_name> </alternative_pair> </alternative_type_names>	

Note that this change will only affect the label of the DM Object Type in the *Contents Index* list and not in any other occurrence of items of this type. A global renaming of DM Object Types can be achieved with the aid of DM Object Type aliases in the data model customization file, as described in paragraph 5.1.2.4.

5.2.1.9. Defining custom draw modes in the Viewer

In the embedded META Viewer of the *Database* workspace of KOMVOS, it is possible to view the parts of a Subsystem colored according to the values of one or more of their attributes. The attributes that can be used are the ones defined in the data model either as primary or secondary attributes. This is achieved be creating a new Style in the **Styles** toolbar of the META Viewer.

To create a new Style, a new **Custom_attribute** must be created in the **Custom_attributes_drawing** element of the `dm_views.xml` file and another file, named `palette.xml` must be created, in the KOMVOS application home ($SDM_CONSOLE_HOME) folder, to define the colors that will represent the different attribute values.

There are two types of Styles that can be defined:

A. Styles based on a single attribute

In this case, the criterion for coloring the parts of a Subsystem in the META Viewer is the value of a single attribute. Each attribute value is mapped to a specific color in the `palette.xml` file.

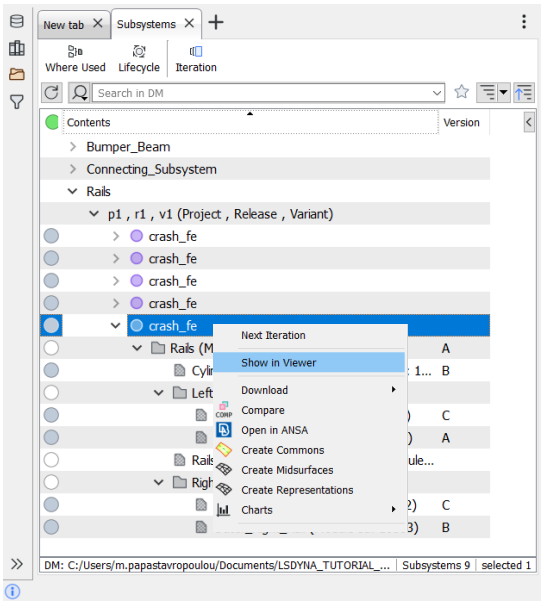
File structure	Description
Custom_attributes_drawing	The **name** element is used to define the name of the view mode that will appear in the View Modes pop-up menu in the META Viewer.
Custom_attribute	
name	The **comment** element is used as a tooltip, for when the user selects this Style in the embedded META Viewer.
comment	The **color_mode** element is used to provide the name of the color mode that is defined in the `palette.xml` file.
color_mode	
attr_keyword	The **attr_keyword** holds the name of the attribute that will be used for determining the color of the parts.

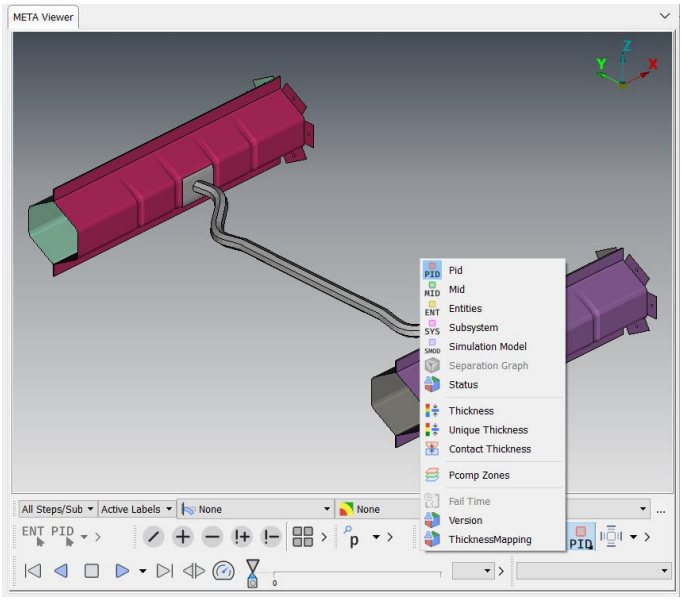
Extract from the `dm_views.xml` file:

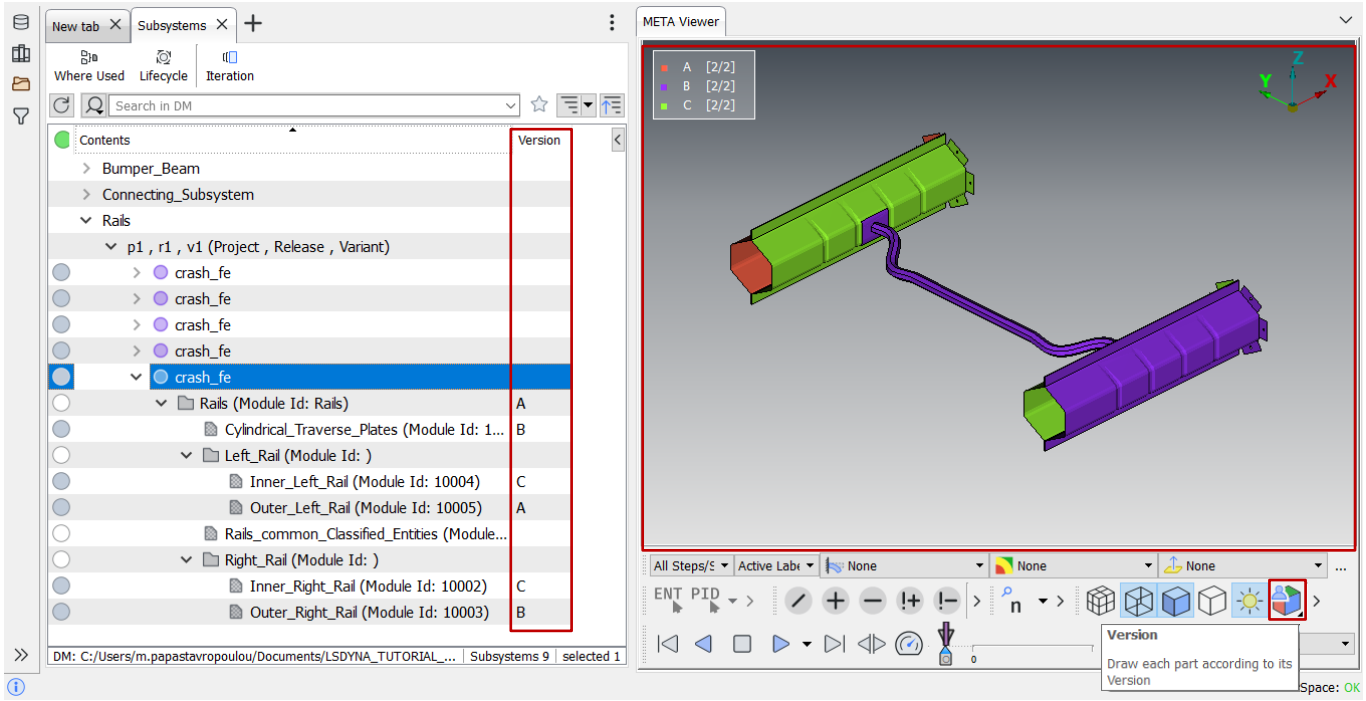
```
<Custom_attribute>
    <name>Version</name>
    <attr_keyword>Version</attr_keyword>
    <comment>Draw each part according to its Version</comment>
    <color_mode>Version</color_mode>
</Custom_attribute>
```

Sample from the `palette.xml` file:

```
<colorMode name="Version" extraInfo="visibleEntities">
    <palette default="1" name="VersionColours">
        <color name="A"     r="255" g="99" b="71"/>
        <color name="B"     r="153" g="51"  b="255"/>
        <color name="C"     r="153" g="255" b="51"/>
    </palette>
</colorMode>
```







B. Styles based on a classification done by a user-script

In this case a script function is provided to classify the parts of the Subsystem based in a custom way e.g., based on the maximum thickness, stated in the *Thickness* attribute, parts can be characterized as *Thin*, *Regular*, or *Thick*. Each of these thickness groups are matched with a specific color in the `palette.xml` file.

File structure	Description
`Custom_attributes_drawing`	The **name** element is used to define the name of the view mode that will appear in the View Modes pop-up menu in the META Viewer.
`Custom_attribute`	The **comment** element is used as a tooltip, for when the user selects this Style in the embedded META Viewer.
`name`	The **color_mode** element is used to provide the name of the color mode that is defined in the `palette.xml` file.
`comment`	The **attr_keyword** holds the name of the attribute that will be used for determining the color of the parts.
`color_mode`	The **script_file_path** element is used to provide the script file name.
`script_file_path`	The **function_name** holds the name of the script function to be used.
`function_name`	

Sample from the `dm_views.xml` file:

```
<Custom_attribute>
    <name>ThicknessMapping</name>
    <comment>Draw parts according to their thickness</comment>
    <color_mode>ThicknessColors</color_mode>
    <script_file_path>CustomAttributeDrawing.py</script_file_path>
    <function_name>ThicknessMap</function_name>
</Custom_attribute>
```

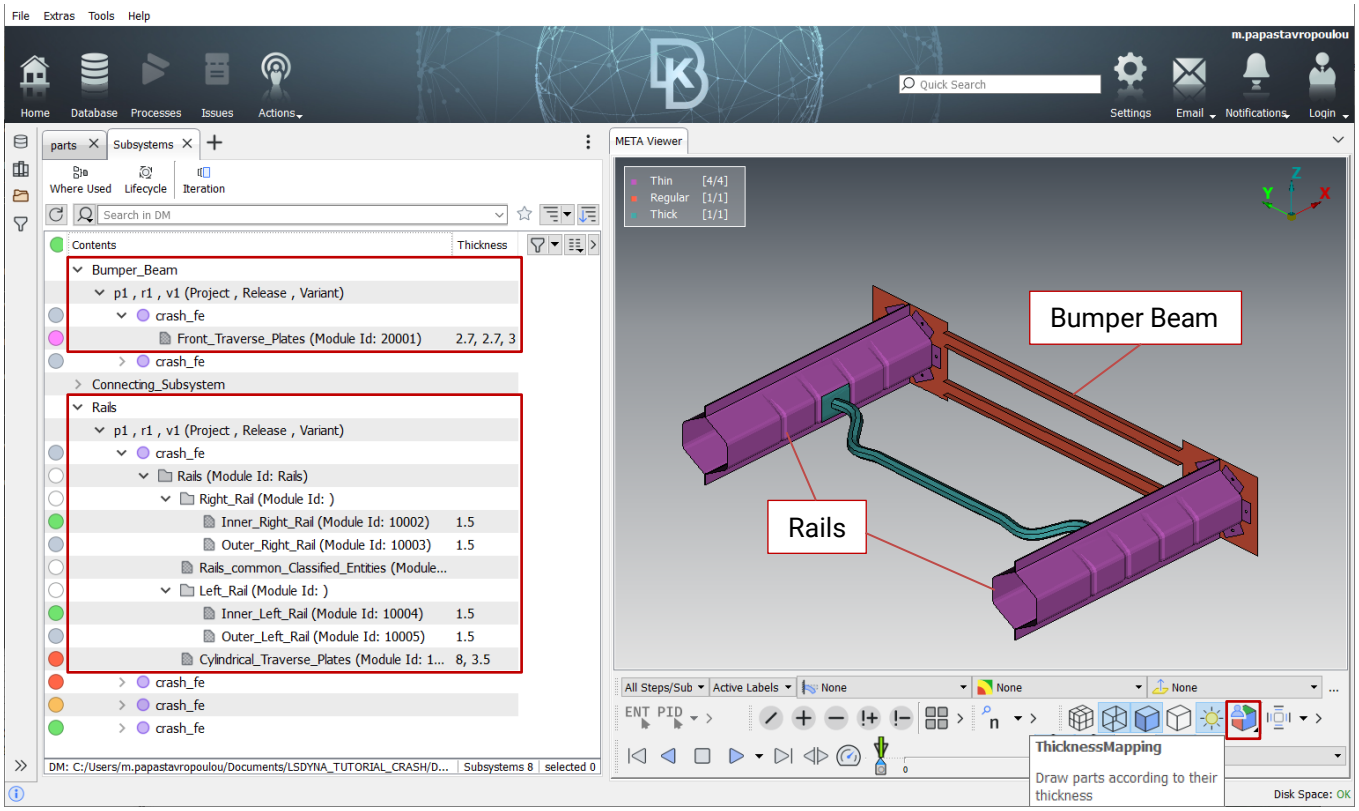
Sample from the `palette.xml` file:

```
<colorMode name="ThicknessColors" extraInfo="visibleEntities">
    <palette default="1" name="ThicknessColors">
        <color name="Thin"      r="192" g="92"  b="192"/>
        <color name="Regular"     r="155" g="200" b="196"/>
        <color name="Thick"     r="69"  g="169" b="169"/>
        <color name="N/A"       r="120" g="40"  b="120"/>
    </palette>
</colorMode>
```

Sample from the script file:

```
def ThicknessMap(list_of_full_inclusive_maps):
    retval = []
    for run_map_of_all_attrs in list_of_full_inclusive_maps:
        thickness_exists, run_thickness = getAttributeValueBasedOnAttrMap('Thickness', run_map_of_all_attrs)
        if thickness_exists:
            thickness_of_parts = run_thickness.split(", ")
            max_thickness = max(thickness_of_parts)
            if float(max_thickness) <= float(2):
                desired_value = "Thin"
            elif float(max_thickness) <= float(3):
                desired_value = "Regular"
            else:
                desired_value = "Thick"
            retval.append(desired_value)
    return retval

def getAttributeValueBasedOnAttrMap(attr_keyword, map_of_attrs):
    exists = False
    attr_value = ''
    if attr_keyword in map_of_attrs:
        exists = True
        attr_value = map_of_attrs[attr_keyword]
    else:
        print('Warning!!No ', attr_keyword, ' attribute found for this: ')
        print(map_of_attrs)
    return exists, attr_value
```



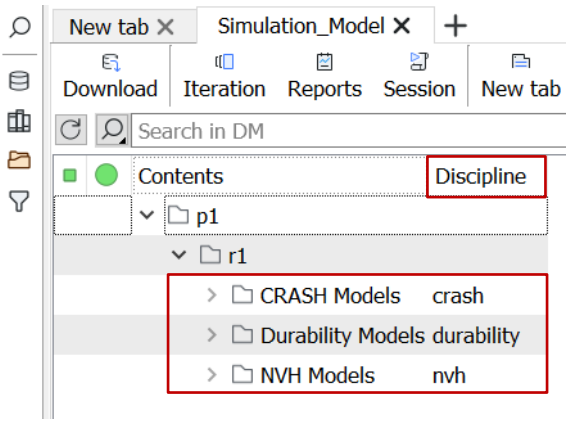
5.2.1.10. Defining nicknames for attribute values

The **attribute_value_nick_names** element is used to map an attribute's value to a different value that will be displayed in the list, e.g. "Experiment" to "Experiments". The definition of an attribute_value_nick_names element is shown below:

File structure	Description
attribute_value_nick_names	The **value** element is used to define the value of the attribute. The **text** element is used to define the text that will be displayed in the *Contents* column.
attribute_value_map	
entry	
value	

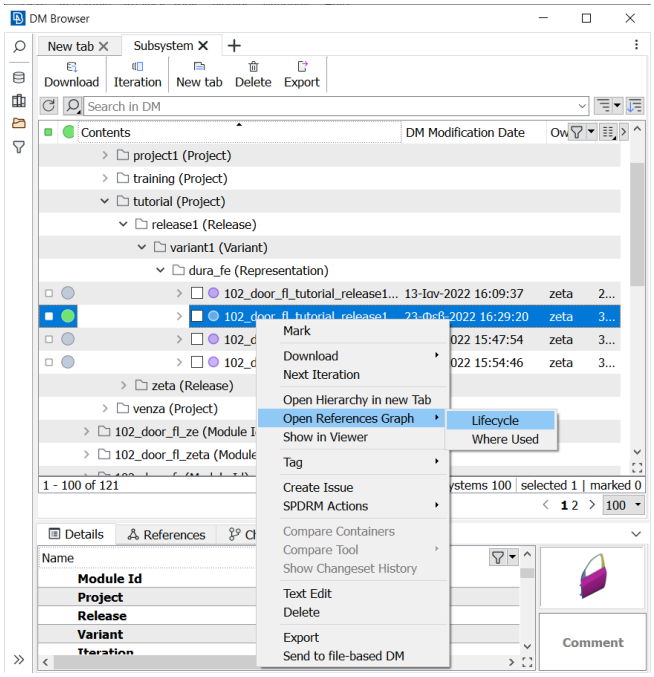
text

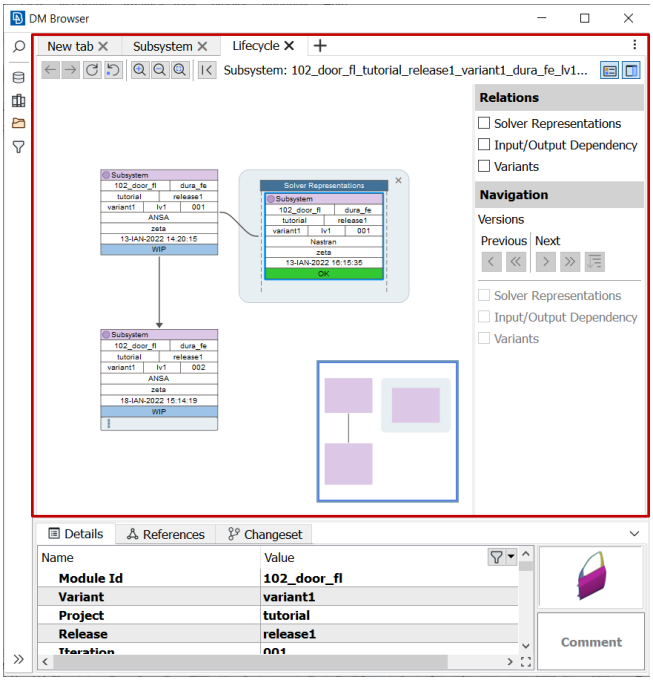
```
<attribute_value_nick_names>
    <attribute_value_map name="Discipline">
        <entry>
            <value>crash</value>
            <text>CRASH Models</text>
        </entry>
        <entry>
            <value>durability</value>
            <text>Durability Models</text>
        </entry>
        <entry>
            <value>nvh</value>
            <text>NVH Models</text>
        </entry>
        <entry>
            <value>cfd</value>
            <text>CFD Models</text>
        </entry>
    </attribute_value_map>
</attribute_value_nick_names>
```



5.2.2. Graph View customization with the data_views.xml file

The Lifecycle Graph displays the evolution of a data object. It is accessed through the context menu of all data objects under *Open References Graph > Lifecycle*. In this view, each data Object is represented by a table that displays its key attributes. When SPDRM is used as a back-end, it is possible to control the attributes that will be shown and the layout of this table. This is done through the `data_views.xml` file, located in the SPDRM installation directory */server/Wildfly/standalone/configuration/*`data_views.xml`.





Defining the lifecycle_view element

The customization of the Lifecycle Graph takes place in the **lifecycle_view** element. Its definition is shown below:

File structure	Sample
⊟ lifecycle_view	
⊟ dm_object	
⊟ table	
cell_content	
⊟ highlight_value_cell	
⊟ condition	
value	
color	

```
<lifecycle_view>
    <dm_object name="Simulation_Model">
        <table rows="3" columns="3">
            <cell_content row="1" column="1" name="Type" value_type="spdrm_attribute"/>
            <cell_content row="2" column="1" name="Variant" value_type="dm_property"/>
            <cell_content row="3" column="1" name="Status" value_type="dm_attribute"/>
            <highlight_value_cell name="Type">
                <condition>
                    <value>Simulation_Model</value>
                    <color red="220" green="220" blue="220"></color>
                </condition>
            </highlight_value_cell>
        </table>
    </dm_object>
</lifecycle_view>
```

One **dm_object** element is required for each DM Object Type whose Lifecycle Graph representation needs to be customized.

The **table** element defines the layout of the table. The total number of table rows and columns is defined in the **rows** and **columns** attributes. These values are not restrictive and they only indicate that the maximum size of the table (i.e. maximum number of cells that can hold a value) can be *rows x columns*. The actual number of cells of the table is determined in a dynamic way from the number of cells that are filled with values. So, the sample above indicates that although the table was initially declared as of size 3x3, only one column and three rows are used so the table will be of size 3x1.

The **cell_content** key is used to define the content of a cell. To locate the cell, the **row** and **column** numbers are given (note that numbering starts from 1). The cell content is defined through the **name** and **value_type** attributes. The

name attribute indicates the name of the attribute and the **value_type** the type of the attribute. The value types can be categorized accordingly:

- **spdrm_attribute:** an SPDRM system attribute, e.g. Type, Id, DM Creation Date, etc.
- **dm_attribute:** a secondary or additional attribute of the DM object
- **dm_property:** a primary attribute of the DM object

The definition of the value type is required because a DM Object type may have more than one attributes with the same name in different *namespaces*. For example, the attribute *Type* that may be a dm_property but at the same time, it is also an spdrm_attribute.

The **highlight_value_cell** element is used to define rules for the coloring of cells based on the displayed value.

The **condition** key is used to keep the **color** to be mapped to each attribute **value**.

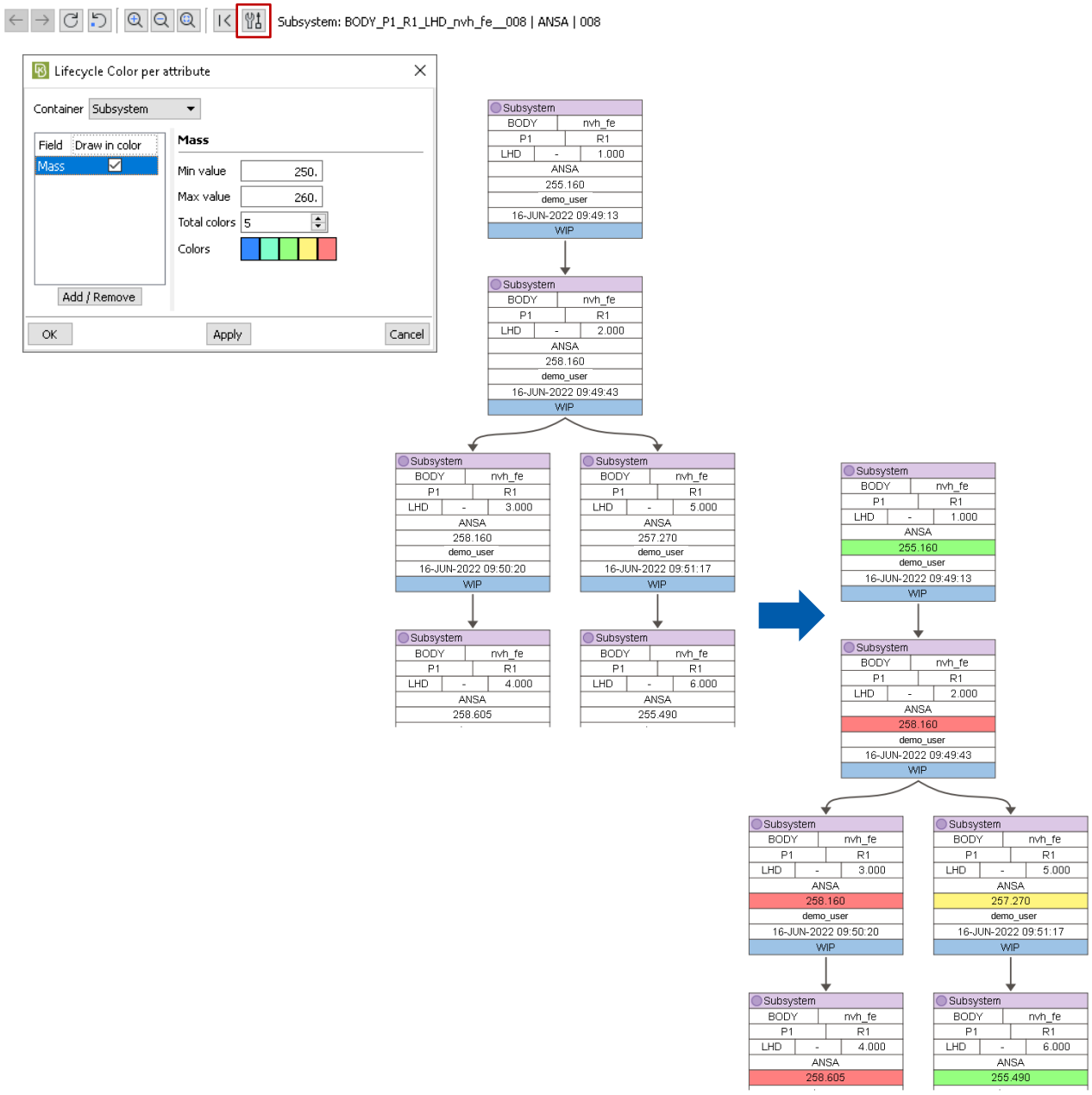
The **color** is described by its RGB index.

```
<lifecycle_view>
    <dm_object name="Simulation_Model">
        <table rows="3" columns="3">
            <cell_content row="1" column="1" name="Type" value_type="spdrm_attribute">
            <cell_content row="2" column="1" name="Variant" value_type="dm_property">
            <cell_content row="3" column="1" name="Status" value_type="dm_attribute">
        </table>
        <highlight_value_cell name="Type">
            <condition>
                <value>Simulation_Model</value>
                <color red="220" green="220" blue="220"></color>
            </condition>
        </highlight_value_cell>
        <highlight_value_cell name="Status">
            <condition>
                <value>Frozen</value>
                <color red="220" green="220" blue="220"></color>
            </condition>
            <condition>
                <value>Valid</value>
                <color red="220" green="220" blue="220"></color>
            </condition>
            <condition>
                <value>Error</value>
                <color red="220" green="220" blue="220"></color>
            </condition>
            <condition>
                <value>Draft</value>
                <color red="220" green="220" blue="220"></color>
            </condition>
        </highlight value cell>
    </dm_object>
</lifecycle_view>
```

In the example on the left, the Status cell gets its background color determined based on the Status value. As the Status can get four alternative values, four conditions are defined.

Note that the method above is only applicable to cells that display attributes that only accept a closed list of values. For attributes that hold numeric values (e.g. Mass), cell coloring according to attribute value is possible through the **Graph View settings** button, accessed from the top left corner of Lifecycle view as shown in the picture below.

At the *Containers* field the user can select the DM Object Type of reference and then select to **Add/Remove** any of its attributes, that hold numeric values, to be colored according to settings. It is possible to select the number of different colors used through the **Total colors** field and define the range **(Min/Max value**) of attribute values to be included for the coloring.


Subsystem: BODY_P1_R1_LHD_nvh_fe__008 | ANSA | 008
Lifecycle Color per attribute
Container Subsystem
Field Draw in color
Mass
Mass
Min value 250.
Max value 260.
Total colors 5
Colors
Add / Remove
OK
Apply
Cancel
Subsystem
BODY nvh_fe
P1 R1
LHD - 1.000
ANSA
255.160
demo_user
16-JUN-2022 09:49:13
WIP
Subsystem
BODY nvh_fe
P1 R1
LHD - 2.000
ANSA
258.160
demo_user
16-JUN-2022 09:49:43
WIP
Subsystem
BODY nvh_fe
P1 R1
LHD - 3.000
ANSA
258.160
demo_user
16-JUN-2022 09:50:20
WIP
Subsystem
BODY nvh_fe
P1 R1
LHD - 5.000
ANSA
257.270
demo_user
16-JUN-2022 09:51:17
WIP
Subsystem
BODY nvh_fe
P1 R1
LHD - 4.000
ANSA
258.605
Subsystem
BODY nvh_fe
P1 R1
LHD - 6.000
ANSA
255.490

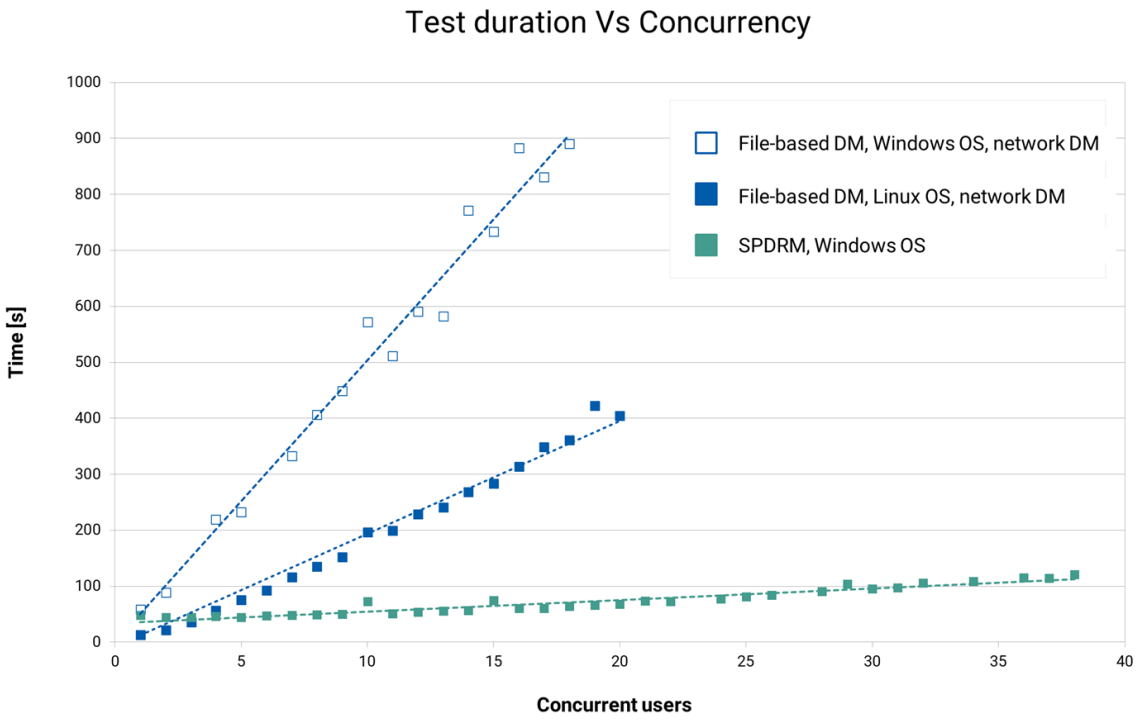
6. Administration topics for file-based DM

6.1. Recommendations

File-based DM is an entry level solution, suitable for small projects and collocated teams of a size up to 10-15 users. The database engine utilized by file-based DM is SQLite, which is a small, fast, self-contained, high-reliability, full-featured SQL database engine.

Although the use of this technology makes the set-up and maintenance of file-based DM an extremely easy task, poor or questionable network reliability may hinder the performance of DM operations at a high extent. Therefore, when such conditions apply, the recommended solution is the client/server SDM solution based on SPDRM.

SPDRM is the scalable, enterprise-level solution that enables role-based user management, control of data access, collaboration of local and remote teams and high concurrency. The graph below shows the resilience of SPDRM to high concurrency comparing to file-based DM.



6.2. DM folder contents

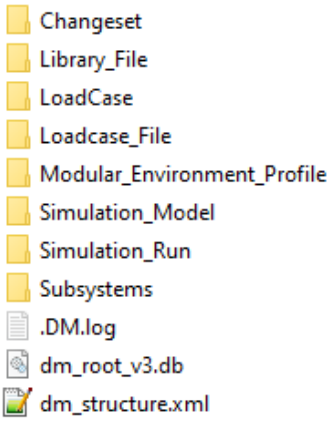
The file structure within the DM root directory is exclusively maintained by ANSA, META or KOMVOS, which, for file-based DMs, incorporate the data management engine and are therefore able to create files and folders during “Save” and “Import” user actions.

The DM folder is accessible by all DM users which means that file browsing within the DM root directory is feasible and direct editing of files through File>Open/Input, with Text Editors or 3rd party pre-processors is also possible.

However, editing files in any other way than those used for checking-in files in DM through ANSA, META or KOMVOS must be strictly avoided, as it is certain to break the DM integrity and lead to data corruption.

6.2.1. Default contents of a file-based DM

The contents of a file-based DM generally look like this:



The **dm_structure.xml** is the data model configuration file that defines the DM Object Types supported by the data manager for this particular DM. When a new, empty directory is set as a DM root folder, ANSA, META or KOMVOS add this file in the DM folder automatically. Detailed information on the contents of the data model xml and its configuration is given in paragraph 5.1.

The **dm_root_v3.db** file is the SQLite database. This file holds the metadata for all DM entries (attributes, relationships, etc.) and is accessed both for reading and writing while a user works with DM.

This file is solely managed by the data manager in ANSA, META or KOMVOS and any attempt to modify it must be strictly avoided, as it will most likely break the DM integrity and lead to data corruption.

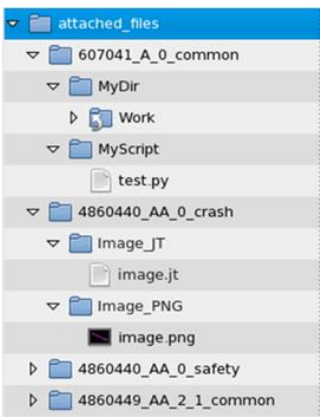
The different folders hold the actual files of the corresponding DM Object type.

Taking a look within the folder of a DM Object Type, the following general observations can be made:

- Files are organized in a structure with a moderate to deep folder nesting, where each folder corresponds to the value of a primary attribute of the particular DM Object Type
- A folder named **attached_files** may exist within the DM Object Type folders of **parts** and **Subsystems**, accommodating all files and folders associated with a DM Object as additional attributes of type FILE or DIRECTORY (e.g. png image, JT file, Definition ANSA File, Interface Representation File, etc.)
- Within the folder of the last primary attribute of DM Object types:
    - For **parts**, the common representation is saved as `common.ansa` and all alternative representations are saved within the folder **repr**
    - For **Subsystems**, the folders **repr**, **hierarchy**, **definition** and **#References** exist, holding the representation file, hierarchy xml, definition attributes xml and references xmls respectively
    - For all other DM Object Types, the folders **repr**, **hierarchy**, **definition** and **#References** exist, holding the representation file, hierarchy xml, definition attributes xml and references xmls respectively and also folders with the name of object attributes of type ATTACH_FILE and ATTACH_DIRECTORY (e.g. Definition ANSA File, Image_PNG, etc.)

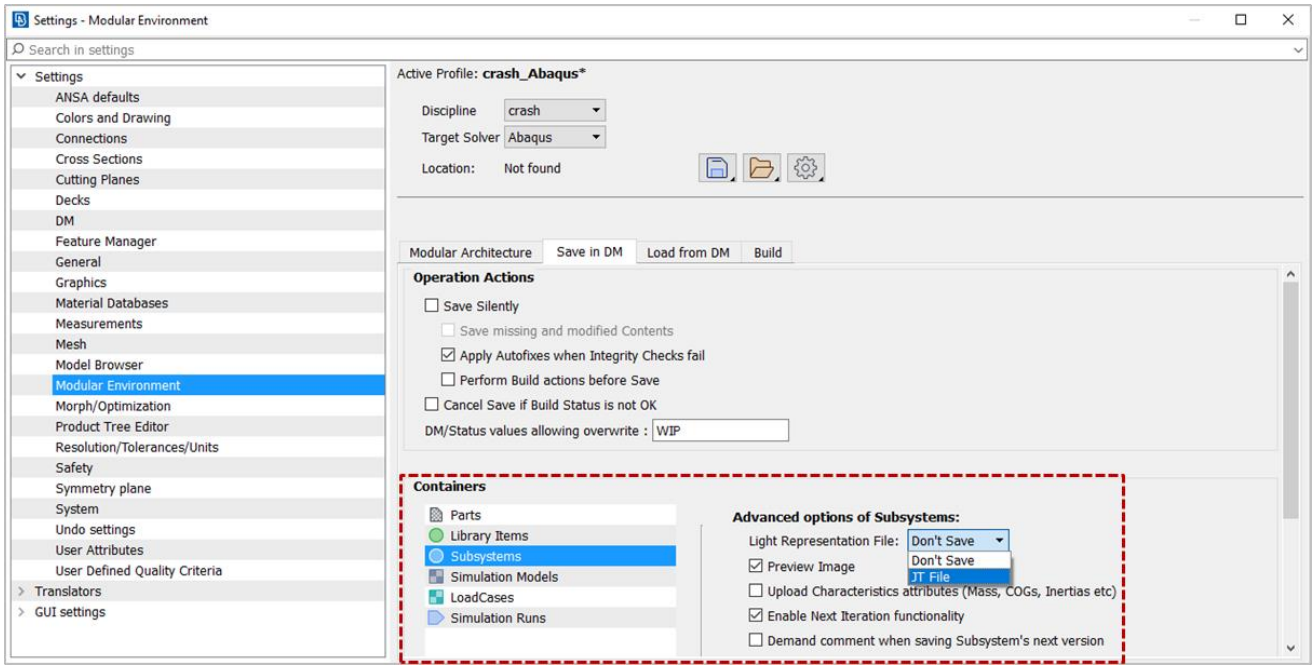
The "attached_files" directory

Within the *attached_files* directory of parts and Subsystems, a separate folder is created for each object saved in DM with some attachments. The folder name is the "signature" of the item (e.g for a part this signature is composed as: *<Module Id>_<Version>_<Study Version>_<Representation>).* Within each of these folders, one sub-folder is created, named after the Attached File or Attached Directory attribute, and within this sub-folder the actual attachment is placed.



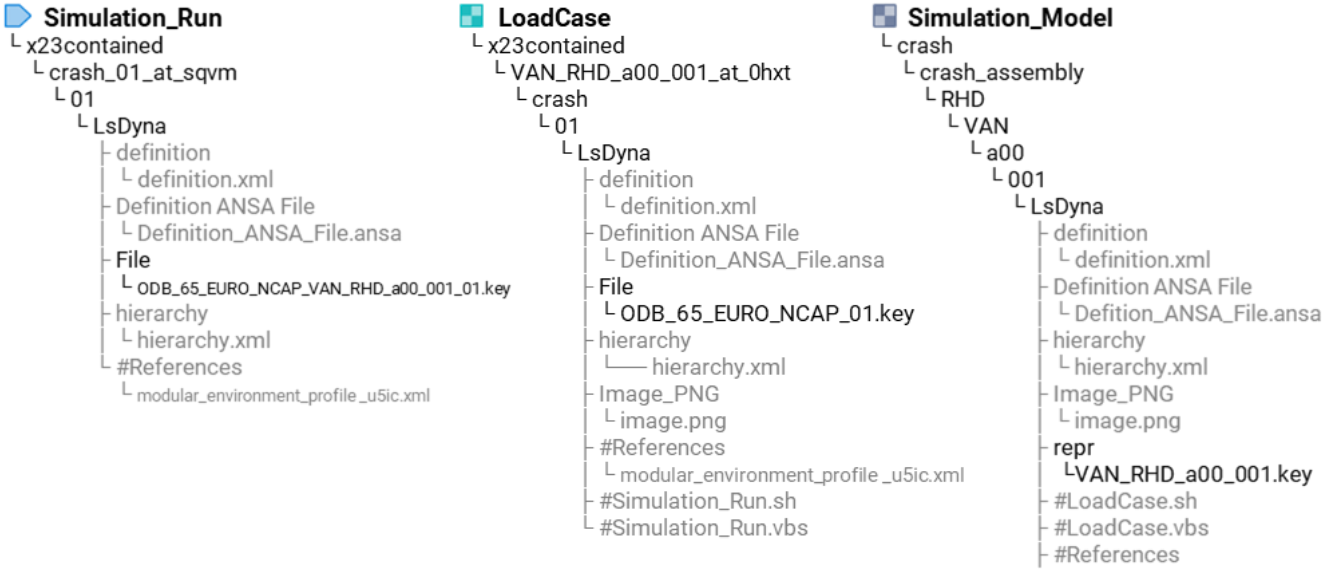
In the example on the left, a Part was saved in DM with two Additional Attributes called "MyDir and "MyScript" of types LINK DIRECTORY and FILE respectively. The values of these attributes point to a specific directory and file, respectively. During Save in DM, for each of these attributes, the data manager has created a folder and has placed inside a link of the folder called Work and a copy of the file test.py, respectively.

When requested through the Modular Environment Profile settings, an image of the saved item, as well as a light-weight JT representation to be used for visualization, are also placed within such sub-folders, as shown on the left.



6.2.2. Customization of the DM directory structure

When data is checked-in in DM, a certain directory structure is created in the DM root directory. For a file-based DM with the default data model, the Simulation Data is stored inside the file-based DM root folder in a structure similar to the one that is shown below.



During save, the data manager creates one sub-directory for each primary attribute (Properties) of the saved item.

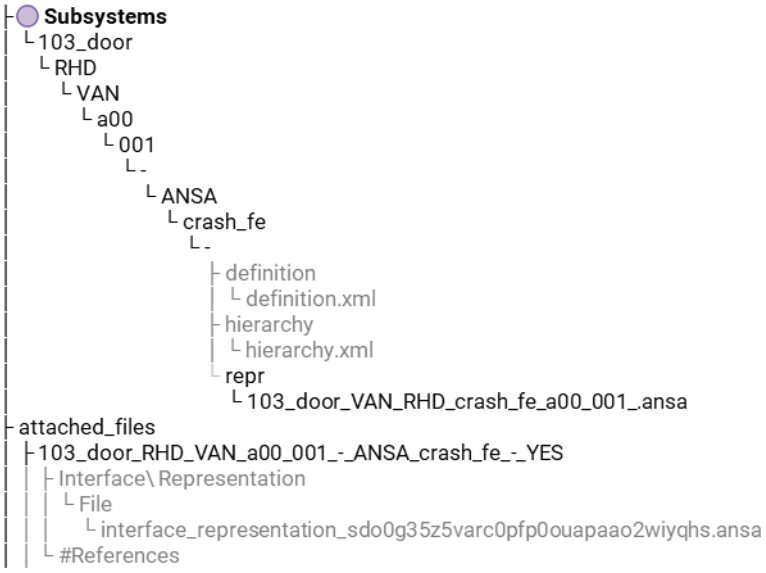
For example, the ANSA file of a Subsystem will be found under the path:

Subsystem/103_door/RHD/VAN/a00/0/ANSA/crash_fe/-/

In order to build this subdirectory, ANSA uses the value of the Subsystem attributes:

Module Id	103_door
Variant	RHD
Project	VAN
Release	a00
Iteration	001
File Type	ANSA
Representation	crash_fe
Loadcase Variant	-

After saving the Subsystem in DM, the sub-directories that are relates to the Subsystems have the following structure:



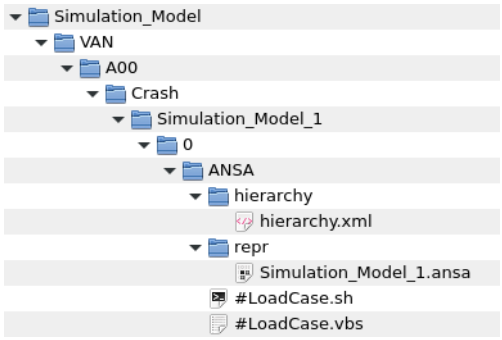
This directory structure is not intended to facilitate data browsing through a File Manager, as check-in and check-out of data within a file-based DM should be solely handled by the DM Browser. However, for those rare cases where manual navigation through DM contents is required, a shallower file structure may be preferred.

In order to reduce the depth of the directories that are created during Save in DM, the user can configure the *dm_structure.xml*, so as to combine multiple properties - which would normally create multiple levels of directories - into a single path component. For example, instead of having three directory levels *VAN/A00/crash*, the user can consolidate them into a single directory by concatenating their labels (e.g. *VAN_ A00_ crash*).

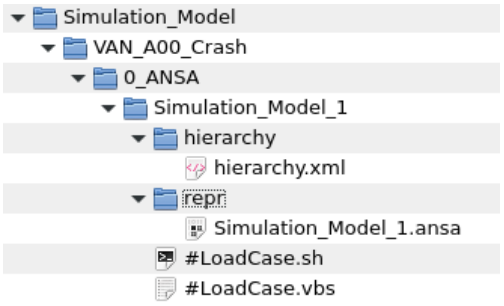
This is achieved by adding the element **Folder_Structure** in the *dm_structure.xml*. Within this element, the user can define how the different path components will be formed when the path towards the appropriate DM entry is synthesized. For example, the following XML snippet dictates the concatenation of three properties for the first level, and the concatenation of two properties for the second level:

```
<Simulation_Model>
  <Folder_Structure>
    <Folder>
      <Property name="Project"/>
      <Property name="Release"/>
      <Property name="Discipline"/>
    </Folder>
    <Folder>
      <Property name="Study Version"/>
      <Property name="File Type"/>
    </Folder>
  </Folder_Structure>
</Simulation_Model>
```

Default folder structure



Modified folder structure



It is possible that not all properties are mentioned in such a *Folder_Structure*. In such a case, all the non-referenced properties that normally have a directory level (i.e. not file related properties or references to other DM entries) are implicitly added at the end, with one path component per property. Finally, the user can quickly define a scheme where all properties are concatenated into a single path component using the following shortcut:

```
<Folder_Structure>
  <Folder generated_from="all_properties"/>
</Folder_Structure>
```

Notes:

- The Folder_Structure element can be used to reduce the nesting of directories for all types except of Parts, Subsystems, Includes and Configurations.
- In order to distinguish different property groups in case they alias into the same component labels, the ***.#attributes.xml*** file is used to record the property name/value pairs that have been used to create this directory.

A Simulation Model can be used with several different Loadcases to create different Simulation Runs. Therefore, saving a Simulation Run in DM, always comes with some reference of a Simulation Model and a Loadcase. In order to represent this relationship on the file-system without the need to store the Loadcase and the Simulation Run under the Simulation Model they relate to, avoiding too deep directory nesting, the components of Loadcases and Simulation Runs are not nested entries into a very long path, but instead each entry is placed into the appropriate (i.e. per entry type), top level container.

The relationship between these entries is expressed in two ways:

- Links are added in the parent entries pointing towards the children entry directories. These links are added for the convenience of the user, in case he/she would like to manually navigate in the directory structure.
- Meta information is added in the children directories, recording the fingerprint of the parent entry.

More specifically, in case there are entries that are contained within other entities (e.g. Simulation Run 1 within a Loadcase), ANSA separates them from freestanding ones. As a result, contained entries are not inserted directly in the appropriate top level container, but within a special folder called ***#contained***.

Within this folder, a folder that is named after the parent entry is inserted and within this folder ANSA adds the folders according to the DM structure.

Normally, the child entry would have a folder named according to its type inserted in the parent entry. For example, a folder named Simulation Run 1 should be inserted within the Loadcase entry. Instead of creating a directory, in this case, ANSA creates a link (*#Simulation_Run.sh* for Linux operating systems and *# Simulation_Run.vbs* for Windows) that points to the folder that has been inserted within the ***#contained*** folder of *Simulation Run* directory. The link destinations are relative, and the user is able to move the DM folder without having the link relationships broken.

Regardless of the input string that is used to create the DM structure, valid strings both in Windows and in Linux are generated for directory names. Specifically, it is possible that the original property value contains characters that are not allowed in a file name. For example, the character '/' is not allowed to be part of the directory name, since it is the path separator. In this case as well as in all relevant cases where disallowed characters are used, they are replaced with the underscore character when creating the directory name. Furthermore, in the case that strings are used which are considered to be prohibited by the operating system of the user the '@' character is added as suffix to the original property value.

In all cases where path components that have different origins (i.e. properties of different values) alias into the same directory name due to the sanitizing process described above, a differentiating suffix is added to the original property value which is of the form @<integer>. In order to be able to differentiate between the two directories and see what is the real property value that has led to their creation, an XML file with metadata (*.#attributes.xml*) is added within them, describing the property names / values that were used for naming this directory.

Another source of aliasing is the first level folders within the *#contained* directories, which use the names of the (otherwise) unrelated parents. In order to avoid collisions, some hashing digits are used after the name (e.g. ABQ_SLE_LC30@y4zw). Nevertheless, if any collisions occur they are detected through the fingerprint of the parent entry which is recorded in the .#attributes.xml file which is placed within this directory. As a result, a directory is considered to be a match not in the case that the name of it matches, but also when the fingerprint within the xml file is the correct one.

The default mode is ANSA to fold the paths of the contained items, during save in DM. Nevertheless, the user can deactivate the folding of the paths of the contained items by changing the value of *flatten_contained_items* option which resides in *<DM_Settings>* element inside the dm_structure.xml to “NO”.

```
<DM_Settings>
   <DM_Setting name="flatten_contained_items" value="YES"/>
</DM_Settings>
```

In the case that,

- *flatten_contained_items = YES* is the default value.
- At least one *Folder_Structure* element is present in the dm_structure.xml file, the folding of the paths of the contained items will be activated. This means that if any *Folder_Structure* elements are presented in dm_structure.xml file, then it is implicitly considered that the *flatten_contained_items* setting is set to *YES*.

6.2.3. Maintaining an external file repository

Although the structure of the DM directory is somewhat customizable as described in paragraph 6.2.2, the DM administrator is not given complete freedom while restructuring the directories as all the primary attributes of each DM object must be included in some way in the final file path. The External Folder Structure functionality enables the automatic creation and maintenance of a file repository external to the DM, following a fully customizable directory structure that may contain:

- directories with fixed names
- directories with any number of primary attributes
- preconfigured prefixes, suffixes, or infixes within directory names.

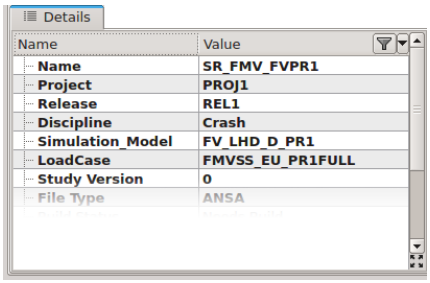
In order for the External Folder Structure functionality to not double the storage space needs, the DM folder only holds hard-links to its files.

The External Folder Structure is activated and configured in the dm_structure.xml by adding the respective element within the element of the DM Object Type to be saved in the external file repository.

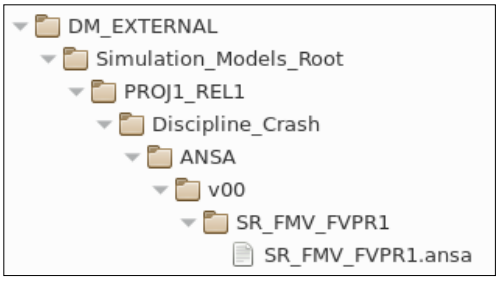
Let's see an example with the definition of an External Folder Structure for Simulation Runs:

```
<XMLData>
   <Simulation_Run>
      <External_Folder_Structure>
         <Root_Folder>/home/user/DM_External/Simulation_Models_Root</Root_Folder>
         <Separator>_</Separator>
         <Folder>
             <Section name="Project" type="Property"/>
             <Section name="Release" type="Property"/>
          </Folder>
          <Folder>
             <Section name="Discipline" type="Fixed_Label"/>
             <Section name="Discipline" type="Property"/>
          </Folder>
          <Folder>
             <Section name="File Type" type="Property"/>
          </Folder>
          <Folder>
             <Section name="Study Version" type="Property" format="v%02d"/>
          </Folder>
          <Folder>
             <Section name="Simulation_Model.Name" type="Property"/>
          </Folder>
      </External_Folder_Structure>
   </Simulation_Run>
</XMLData>
```

A Simulation Run DM item with the properties shown in the image below, is saved in DM.



If we have a look in the external file repository, the saved file and folder structure will look like this.



The optional element ***<External_Folder_Structure>*** in the dm_structure.xml file defines how a DM Object is going to be saved in the external file repository. When this element is set, files of the respective DM Object Type will be saved into the external repository following the folder structure defined, and links will be created in the standard DM root folder.

In the ***<External_Folder_Structure>*** element:

- The location of the external file repository is defined by the ***<Root_Folder>*** element.
- The folder structure that will be created to store the DM Object is defined by all the ***<Folder>*** elements in sequence. All ***<Section*** keys that are grouped under one ***<Folder>*** element are combined into a directory name.
- These sections are concatenated with the defined ***<Separator>***. Each ***<Section*** can be any property, or attribute of the DM Object, or even a fixed string value.
- The "dot notation" (e.g. "Simulation_Model.Name") can be used in Parent-Child cases, to refer to the properties of another entity.

6.2.4. Regenerating the dm_root.db from the file structure

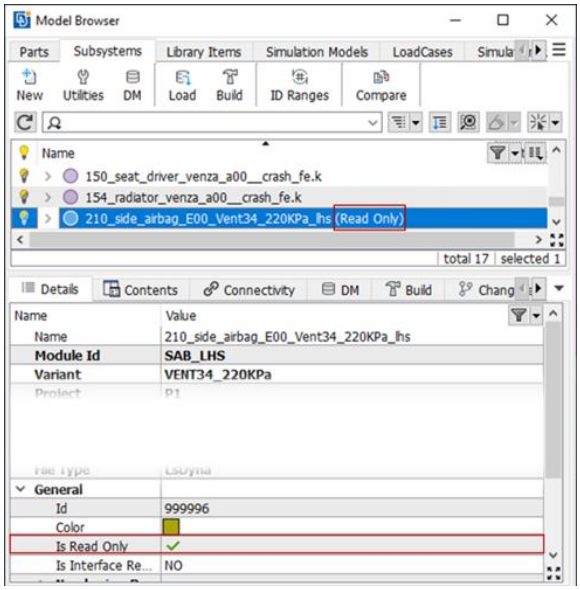
As mentioned in paragraph 6.2 The **dm_root_v3.db** is the “heart” of a data management repository. However, there are cases where the user might copy data in the form of files or folders directly in the DM directory, such actions will lead to a divergence between the contents of the DM’s directory and what is displayed in the DM Browser. In such occasions, it is possible to update the contents of the database file manually by executing the “Rebuild from Disk” functionality from the main button menu of the DM Browser window. This functionality upon confirmation updates the DM database by scanning the DM directory structure. Essentially, it parses the contents of the DM directory in order to rebuild the them and restore any references that may exist between the various DM Objects.

Note! This functionality is available both in regular file-based DMs as well as in Cluster DMs.

6.2.5. Read-only base Modules

In file-based DMs, any modification in the contents of a base module will be identified by the Modular Run Management tools and the respective Subsystems or Library Items will be marked with “Modified” DM Update Status. This marking indicates that that the module was deliberately modified and must be saved in DM during the next Save operation. However, there are cases where a base module can be modified automatically during input, as it may contain unsupported keywords or unsupported numbering schemes. Such modules should never be marked as “pending to be saved”, as such modifications are both unintentional and undesired. In such cases the base modules need to be used as a “black box” and be excluded from the saving process. Such handling may be necessary for Subsystems and Library Items retrieved from suppliers or modules that may contain unsupported keywords or numbering schemes.

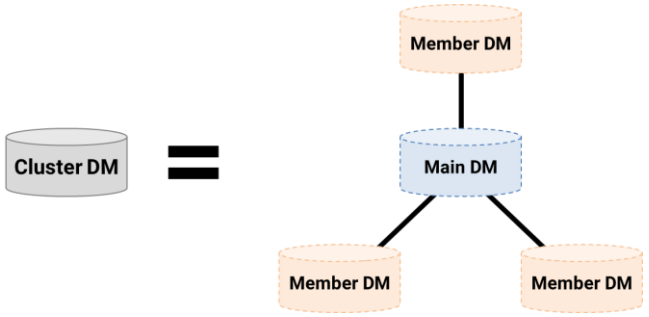
The handling of a base module as a “black box” can be requested with the aid of the “Is Read Only” flag in the ANSA Model Browser.



6.3. Cluster DM

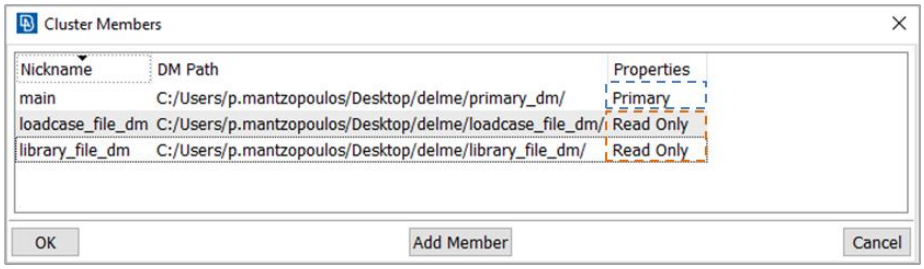
6.3.1. Working with Multiple DMs

Utilization of several source file-based DMs is possible by combining them into a single entity.

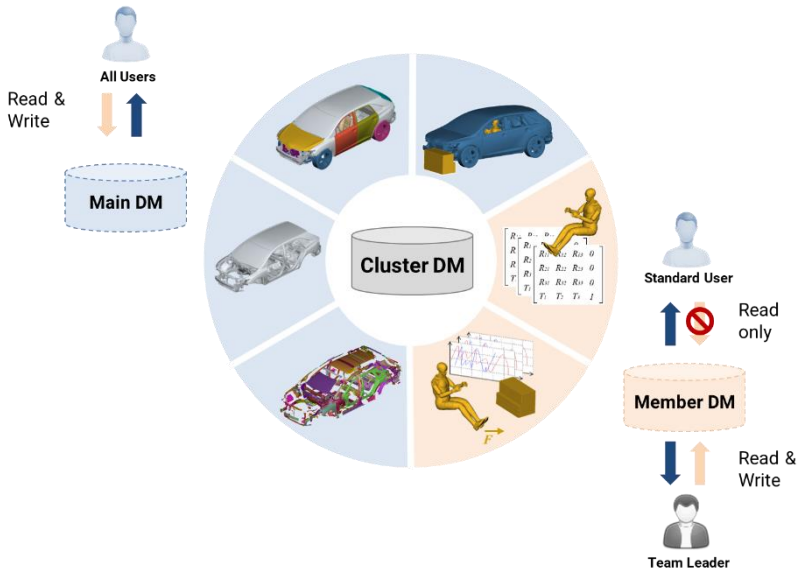


In this set-up, one of the DMs is mandatorily used for both reading and writing data, and the rest DMs operate as read-only sources of DM Objects. A configuration like this is called a **DM Cluster** and the combined DMs are its members. The DM that is used for both reading and writing data is called the **Main DM**, while the read-only DMs are often called **Library DMs** or **Member DMs**.

When connected to a Cluster DM, any operation against the DM Cluster is translated towards the corresponding Member DMs, ultimately allowing the usage of multiple DMs in parallel.



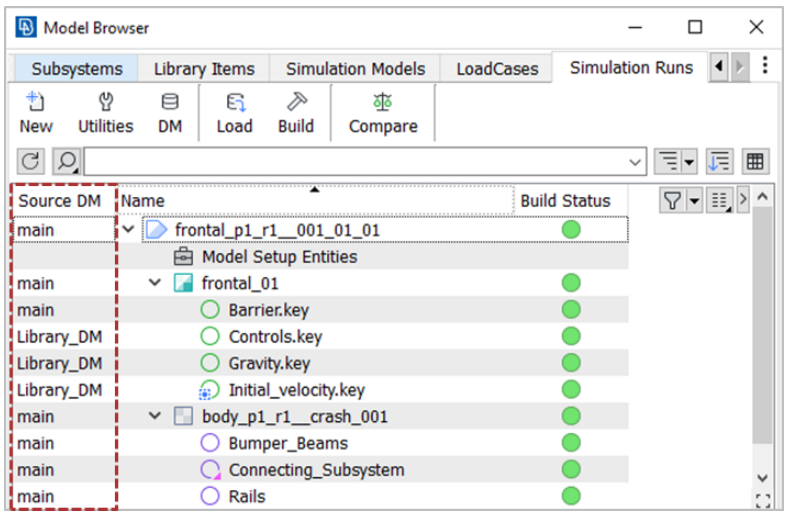
A step-by-step guide on how to set-up a Cluster DM is available in Chapter 30.6 of the ANSA User's Guide.



The most common use case for Cluster DMs is to separate the storage of Project data from that of Library data in order to apply an extra layer of control on who populates the library with data that will be used by the whole team.

With the Cluster DM it is possible for a Team Leader to have exclusive write access to Library DMs, while the whole team uses their content in read-only mode.

Finally, the same Library DMs can be used as read-only members in several Cluster DMs, facilitating the creation of one DM per Project/Release using common library files.



In the image on the left, a Simulation Run is shown in the Model Browser and in the Source DM column one can see information on where each of the Model Container is saved, in terms of Cluster DM members.

When a DM is used as a read-only member of a Cluster DM, overwrite actions of its content should be avoided in all cases. As the data manager is unaware of which of the contents of the read-only member are already used by DM Objects of the Cluster DM when the Team Leader is connected to this read-only member as a primary DM (e.g. to populate it with data), it is not possible to prevent actions that could put the integrity of related Cluster DMs at stake. Hence, it is possible to manually mark such DMs, that are used as read-only members in Clusters, as Library DMs, in order for the data manager to raise warnings when such modifications are attempted. When a DM is marked as a Library DM:

- Deletion of all DM Items is prohibited.
- A warning is shown when an item is about to be overwritten.

A DM is marked as a Library DM by modifying the following DM_Settings in the *dm_structure.xml*.

```
<DM_Settings>
    <DM_Setting name="Software Version" value="22.1.2"/>
    <DM_Setting name="adaptation_key_calculation_method" value="v1"/>
    <DM_Setting name="avoid_special_chars" value="YES"/>
    <DM_Setting name="avoid_special_chars_everywhere" value="NO"/>
    <DM_Setting name="display_types_instead_of_object_types" value="YES"/>
    <DM_Setting name="flatten_contained_items" value="YES"/>
    <DM_Setting name="intermodular_connectivity_links" value="NO"/>
    <DM_Setting name="library_dm" value="YES"/>
    <DM_Setting name="target_point_property_name" value="Target Point"/>
</DM_Settings>
```

Note! Read only-members of a Cluster DM can be removed at any time from the Cluster entity except for cases where hierarchy dependencies exist between the contents of the Member and Primary DM.

6.3.2. Data Model compatibility checks

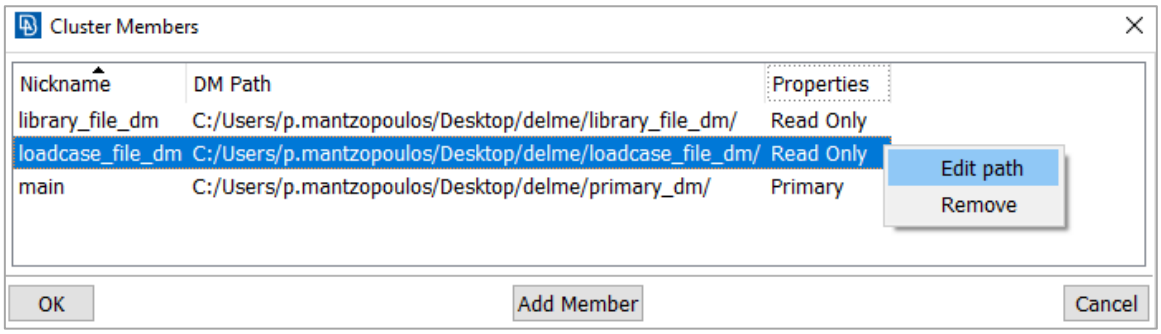
Data Model compatibility is a pre-requisite for the proper configuration of a Cluster DM. For this reason, during the creation of a Cluster DM the following checks must be made:

- All DM Object types defined in the DM Cluster Members are also defined in the Primary Member.
- For each object type, the primary attributes are the same. More specifically:
  - They have the same attribute name
  - They are defined in the same order
  - They have the same type
  - The default value is the same or undefined in both
  - If empty value is accepted in a Member DM, it is checked whether it is also accepted in the Primary
  - Accepted values of an attribute in the Member are a subset of those defined in Primary DM.

Any errors identified during the compatibility checks above, will block the addition of the member DM and will be reported in the My_DM.log file.

6.3.3. Update DM Cluster Member DM Paths

Cluster DMs are defined with absolute paths to all their members. Therefore, if a Cluster DM is moved to another location these paths will no longer be valid and the DM will be unusable. In order to overcome this situation, it is possible to edit and update the paths of each member DM.



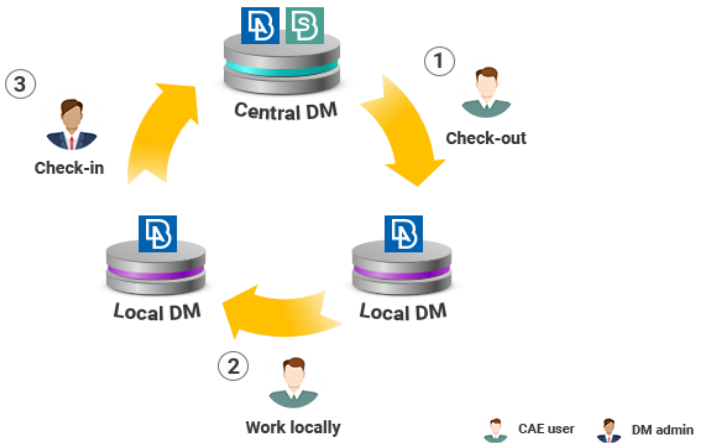
6.4. Moving data between DMs

The **Send to DM** functionality enables the transfer of selected data from one DM to another. With this functionality, it is possible to:

- Enable off-line working mode in cases the central DM is inaccessible at times
- Enable data access to remote users that do not have access to the central DM
- Create optimization studies locally, without cluttering the central storage with temporary data

With **Send to DM** Functionality:

- Selected content from a Central DM can be transferred to a Local DM so that the user works with the extracted data in “offline” mode.
- Data produced in the local DM as new iterations of previously transferred data can be “pushed back” to the central DM.



The functionality is able to handle both base and compound Model Containers. In any case, all the dependencies of transferred Model Containers are identified and transferred along. The target of this action is to keep no

dependencies between the local and the source DM. For instance, by requesting the transfer of a Simulation Run, all its contents (Simulation Model, Loadcase, Library Items and Subsystems) are copied from the Source DM to the target one.

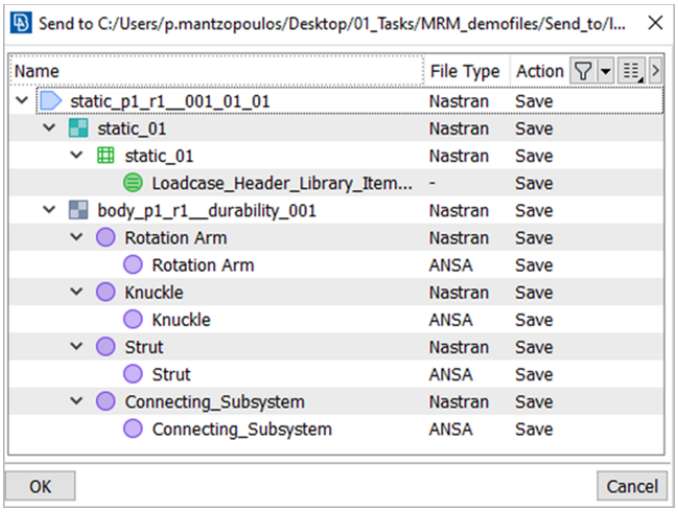
Presently, **Send To DM** functionality is only available within the DM Browser of ANSA.

Important note!

When transferring DM Objects from one DM to another, DM objects with ANSA File Type whose version needs to spin-up in the destination DM, will be re-saved with the same settings of the source DM object.

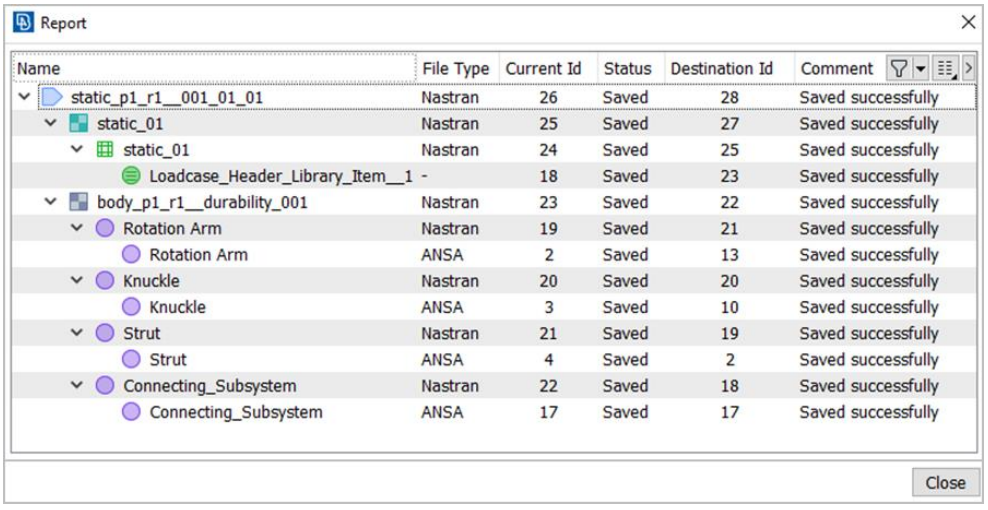
6.4.1. Create a local DM

As mentioned above, using the Send to DM functionality, it is possible to transfer content and metadata from the Source DM to a local one by activating the ***Send to > File-based DM*** functionality after selecting the desired content. During the process, the user is prompted to define the target DM that will host the transferred data. Then, a preview of the actions about to take place is given and once the user confirms, the data transfer starts.



Preview

In the end, a process report is provided, with the status of each transferred DM Object in the local DM. Furthermore, information regarding the new “Id” of each transferred container is available, as well as information regarding the status and any possible errors.



Note! Data transfer may take a while in case of transferred data with many dependencies.

From then on, the newly created DM is fully functional and the user can connect to it and work as usual. This is possible due to the fact that during the transfer process several actions took place in the background in order to ensure the creation of a fully operational file-based DM. In particular:

- In case the target DM is a working DM with existing data, the Data Model compatibility is assured before the transfer begins.
- Any dependencies of the transferred DM Objects, as well as possible additional attached files, are identified and downloaded accordingly.
- Data are exported in the suitable form a DM Root folder structure.
- Referenced Paths written inside the solver keyword file are updated to new paths.

Note! The “Send to” functionality is supported for Nastran, LSDyna and Pamcrash solver keyword files.

A step-by-step guide on how send data to a local DM is available in Chapter 30.5.1 of the ANSA User’s Guide.

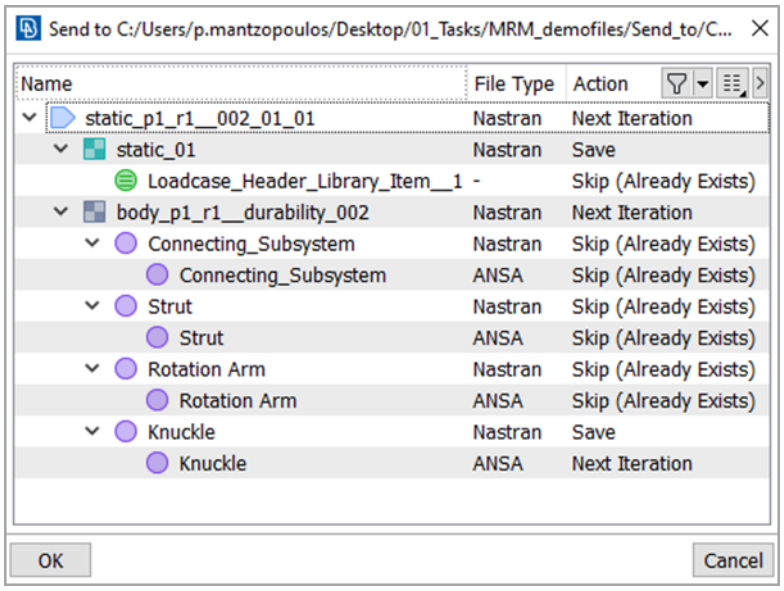
6.4.2. Send data back to the source DM

At some point the user has completed the “offline” work with the data that was initially extracted. Possibly, new data was produced in the local DM as new iterations of the initially transferred DM Objects and some of these data need to be “pushed back” to the central DM. For this operation, ANSA needs to know:

- The location of the source DM (path or server URL)
- The source DM objects being transferred, in order to be able to create the right lifecycle relationships in the source DM

This information is stored in the metadata of the local DM and thus, at any time the user requests some data to be sent back to the source DM, it is retrieved by the function automatically.

In order to send some data from the local DM back to the source DM, the user only needs to choose the desired DM objects in the local DM and select the option **Send to > Source DM**. A preview of the actions about to be performed is displayed and the user can confirm in order for the data transfer to start.



Preview

In the end, a synopsis of all actions that took place and their status is provided to the user.

Note! There are some rules that are followed by the “Send to” functionality in case any conflicts arise during the transfer of DM items between the source and the local DM and vice versa. In particular:

- In case the DM item that is transferred to the target DM is identical to an existing one, then the latter will be reused.
- If the incoming item is not identical to an existing DM Object and is marked as a source DM item, then a new iteration of its source item will be stored.
- If the incoming item is not identical and is not marked as a source DM Object, then a new iteration with no history links will be stored.

A step-by-step guide on how to send data back to Source DM is available in Chapter 30.5.2 of the ANSA User’s Guide.

6.5. Concurrent data access

When a file-based DM is used by a team, there are cases where several users attempt to read and write data at the same time. Reading data simultaneously is never an issue, even if different users attempt to get access to characteristics of the same DM Object at the same time. However, writing data simultaneously may be an issue, especially when different users attempt to save the same DM Object.

Locking mechanism is available to facilitate concurrent writes in DM of objects with the same identity by different users or processes. The locking mechanism, by default inactive, can be activated through **Tools > Settings > DM** by activating the option **Enable locks for concurrent uploads to File based DM**.

The locking mechanism is based on the following rules:

1. When different users attempt to save at the same time a DM Object that already existed in DM with the conflict resolution option “Spin-up”, all users get a new iteration with a history link to the same base object.
2. When different users attempt to save at the same time a DM Object that did not exist in DM in the beginning, one of the involved users gets the first iteration and all other users get a new iteration with no history links with any other object.

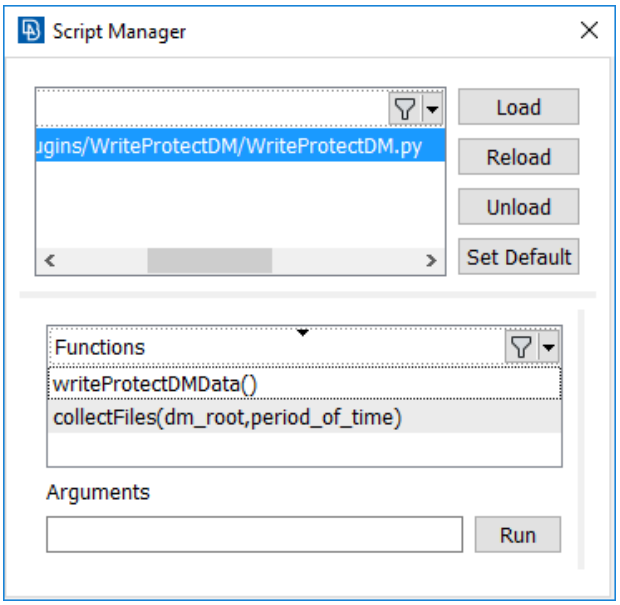
6.6. Access control

As stated in paragraph 6.2.1, the DM Objects stored in DM are comprised of one or more files and certain metadata. In file-based DM, the metadata are kept in the .db files in the root level of the DM folder. Write-protecting data requires the restriction of deletion twofold: First, the deletion of files on the file-system and then, the deletion of related records of metadata from the .db files.

In a Data Management system, management of access rights would enable the control of which users have access to view, modify, delete and execute data objects. Such management of access rights for DM data is supported out-of-the-box in the SPDRM-based implementation, but is generally not possible for file-based DM. Nevertheless, it’s still possible to have a minimum protection from accidental data deletion in file-based DM given that DM operations through ANSA, META or KOMVOS assume that a DM object inherits its access rights form its primary file. Thus, if the file-based DM administrator takes care to write-protect the files of certain DM objects on the file-system, DM operations that attempt deletion or overwrite through ANSA, META or KOMVOS will assume that no change can be made to the metadata of these objects in the .db files or to any of their complementary files.

The collection of the files of certain DM objects in order to change their permissions may be quite tricky, as it requires a good understanding of the overall DM folder structure. Therefore, a script is provided for this purpose in the ANSA installation under *$ANSA_EXEC_DIR/scripts/DM/WriteProtectDM.py*

This script offers 2 functions:



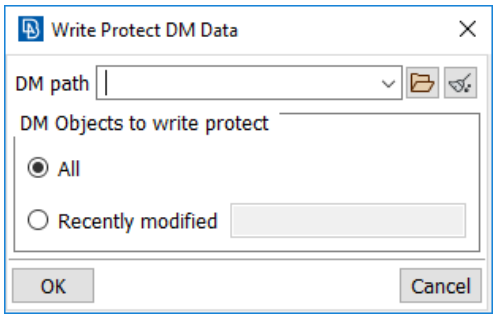
writeProtectDMData: This function can be used to collect the DM files and change their permissions (Linux only).

It is also accessible through a User Script Button.

collectFiles: This function will only collect the DM files and will return them in a Python list. A user could call this function in his/her own script, only to collect all files in DM and manage their permissions. This function accepts two arguments as inputs:

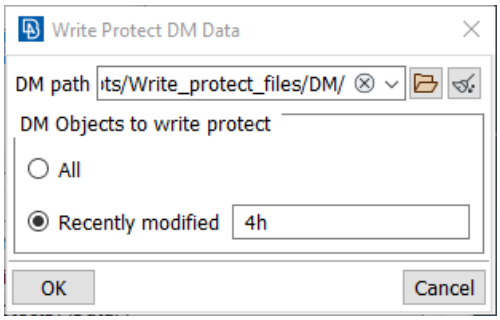
- *dm_root*: this argument refers to the absolute path of the DM repository.
- *period_of_time* (optional): this argument refers to the time period were the collected files have been created or modified. If no time period is specified then All DM files will be collected. The syntax for this argument uses characters h, w, d, m for hours, weeks, days and months respectively along with an integer in front to indicate the extent of the period.

Running the **writeProtectDMData** , the Write Protect DM Data window pops up were the user must specify the DM path along with some options for the collection of DM files.



First, specify the **DM path** of the DM to be write-protected and choose among the two options for the collection of files:

- **All**: All DM objects will be selected in order to write-protect their files.



- **Recently modified**: The DM objects, which were last modified at the time period defined by the user, will be collected.

In order to specify the time period, the following characters are used: h, w, d, m for hours, weeks days and months, respectively, along with an integer in front to indicate the extent of the period.

For example, if the user wants to collect the DM files from the DM objects that were created/modified the last 4 hours he/she will type '4h' as shown in the picture on the left.

The DM root is set when the **OK** button is pressed. All files associated with each DM object will be collected and will be write-protected.

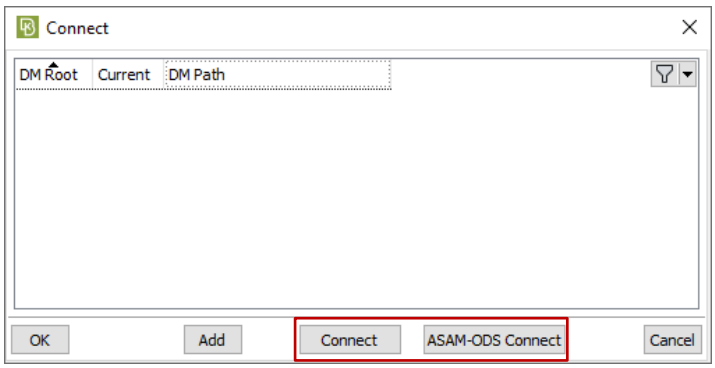
7. Advanced Client Topics

This chapter describes the following client-specific topics that are relevant for all the SDM Clients (KOMVOS, ANSA, META):

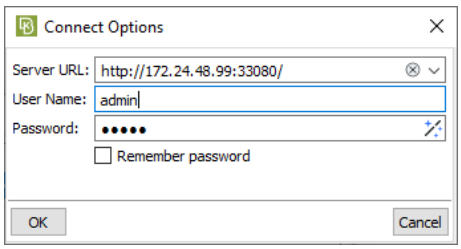
- **Authentication**: This topic is only applicable to the server-based SDM solutions
- **Logging**: This topic is valid for all SDM back-ends, file-based or server-based

7.1. Authentication

In order for a client application to connect to a server-based SDM back-end, the user needs to provide his/her credentials that are sent to the SDM back-end for authentication.

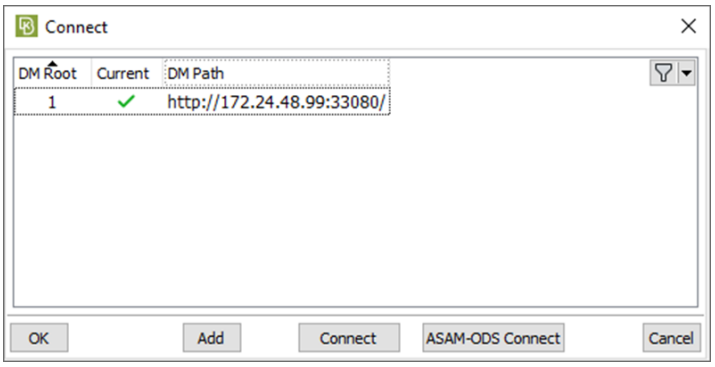


Connect: Opens the **Connect Options** window, used to establish connection to an SPDRM server or to a SimManager server.

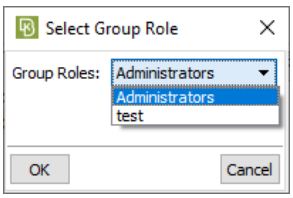


ASAM-ODS Connect: Opens the **ASAM-ODS Connect Options** window, used to establish connection to a test-results server.

The credentials provided are submitted by the client to the server for authentication. If they are authenticated, the client is granted access to the server, the Connect Options window closes and the Server URL is added in the list of DM back-ends in the Connect window with a green tick-mark that indicates that this is the current DM. If the credentials are not authenticated, the client is not granted access to the server and a relevant message is printed in the info window.



Note that when connecting to an SPDRM server, if the user belongs to more than one roles, he/she is prompted to select the role with which the connection should be established.



The moment a successful connection of the client to the server is established, the server returns an authentication token that can be used for later connections to the same server, for as long as it's valid (token expiration is controlled by the token lifecycle policy implemented by each server). The SDM clients store this token in a hidden file in the user's home directory together with some additional information:

- **`.ansa_spdrm`**: The file holds information about the Server URL, the user name and role, the server version and the authorization token (also referred to as a DM ticket). In case the user activated the *Remember Password* option in the Connect Options window, the password is hashed and stored in this file too.
- **`.ansa_simmanager`**: The file contains similar information with the .ansa_spdrm file
- **`.ansa_asamods`**: The file contains all connection settings and the hashed password of the connecting user.

The SDM clients store the list of the SDM back-ends and information on which back-end is marked as "current" in their general settings files:

- KOMVOS: Information is saved automatically on application quit in the file:

```
.BETA/KOMVOS/version_xx.x.x/DM/sdm_settings.db
```

- ANSA / META: Information is saved when "Save Settings" is triggered in the files:

```
.BETA/ANSA/version_xx.x.x/ANSA.defaults
```

```
.BETA/META/version_xx.x.x/META.defaults
```

On launch, the SDM client attempts to connect automatically to the SDM back-end marked as current in its settings with the information stored in the hidden files .ansa_spdrm, .ansa_simmanager, .ansa_asamods.

Automatic connection to an SDM back-end during application launch can be also achieved with the aid of the following command line options:

Argument name	Description
`-dmroot <value>`	SDM back-end server address (URL) for SPDRM or MSC SimManager. Even in case of ASAM-ODS, a string with all connection information can be used for direct connection during launch through `-dmroot`.
`-dmusername <value>`	Defines the user name to be authenticated to the SDM back-end.
`-dmpassword <value>`	Defines the password to be authenticated to the SDM back-end.
`-dmticket <value>`	Defines the authentication token to be used for connection to the SDM back-end.
`-dmrole <value>`	Defines the role with which the user must be connected to the SDM back-end

7.2. Logging

Log files record events and transactions that happen while working with the system.

For server-based SDM systems, different log files are created at server and client level.

For file-based DM, all logging is done client-side.

KOMVOS, ANSA and META all produce a single log file, the **`My_DM.log`** with useful information on events and transactions that take place client-side.

For file-based DMs in particular, an additional log file, the **`.DM.log`**, is saved within each DM location.

7.2.1. My_DM.log

This is the primary file used for logging. It records all DM operations like saving data, loading data, getting info on data, etc. In case of errors related to DM operations, this is the reference file used for investigation.

The scope of this file is “single user / single application / any DM”. It is user specific, and is by default located within the user settings folder (~/.BETA/<app_name>/version_xx.x.x/DM/My_DM.log). A different log file is created for each application in the respective folder. The file is not specific to a particular DM folder and it records events and transactions that may occur in different DMs.

The user can inspect this log file directly from within the application:

In ANSA:

- *Lists > DM > My_DM.log or*
- DM Browser main menu: View My_DM.log

In META:

- DM Browser main menu: View My_DM.log

In KOMVOS:

- Extras > Logs > View My_DM.log

It is possible to change the default location of the My_DM.log file through KOMVOS Preferences *Disk Monitoring>My_DM.log Directory* and through the ANSA Settings *DM > My_DM.log folder*.

Each log message has an associated log level that gives a rough indication of the importance and urgency of the message. The file-based data manager supports the four log levels below, presented in order of increasing detail:

a) LOG_ERROR
b) LOG_WARNING
c) LOG_INFO
d) LOG_DEBUG

The default log level is the LOG_INFO which prints informational messages that highlight the progress of each Data Manager action and will make sense both to end users and system administrators.

In cases where detailed monitoring of the DM operation is required, it is possible to raise the log level to LOG_DEBUG. This is done with script, with the script function `dm.SetLogLevel()` that takes as an argument the corresponding log level:

```
import sdm
from sdm import dm
dm.SetLogLevel(dm.constants.LOG_DEBUG)
```

7.2.2. .DM.log

This file records the changes that affected the content of the DM, e.g. addition or deletion of objects.

The scope of this file is “any user / any application / single DM”. It records DM actions on a particular DM executed from any user, through any application.

The user can inspect this log file directly from within ANSA/META:

In ANSA:

- *Lists > DM > DM Log*
- DM Browser main menu: *DM Log*

In META:

- DM Browser main menu: *DM Log*

BETA
SIMULATION SOLUTIONS

physics on screen

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

physics on screen