DM.inspect: customizable quality control of LS-DYNA input files

Steffen Mattern[1], Robert D. Bitsche[2], Marcel Koch[3]

[1]DYNAmore GmbH
[2]Scale GmbH
[3]Dr. Ing. h. c. F. Porsche AG

1 Introduction

LS-DYNA models for industrial applications are often composed from several smaller sub-models. For example, in the automotive industry the different components of a car are usually modelled separately. These “include files” may be built-up in different departments of the same company or even developed by external suppliers. Due to the huge variety of features and functionalities in LS-DYNA, it is a good idea to set up general rules for the sub-models to achieve robust and efficient simulation models for production. Also, the successful assembly of models can be assured by defining generalized modeling guidelines, especially for the interaction of the different include files.

In order to ensure compliance with their modeling guidelines already during the development process of the sub-models, Porsche tasked DYNAmore with the development of a batch program that performs quality checks on LS-DYNA input files. This program is now an inherent part of Porsche’s CAE process chain and is also used by some of their external suppliers. Based on the experiences from this collaboration, DYNAmore has now developed a new software called DM.inspect, a batch program allowing customized quality control for LS-DYNA input files. The software requires the individual definition of quality criteria by the user rather than containing any pre-defined checks. This modularized approach allows the use of DM.inspect for various simulation disciplines with different requirements.

The check procedure performed by DM.inspect can be divided into three subsequent phases. An optional pre-phase where checks from a pre-processor can be applied, a simulation-phase performing an LS-DYNA initialization run and a post-phase with various checks based on the LS-DYNA input and output files. In the end, DM.inspect writes a report containing the results of the performed quality checks.

This paper introduces DM.inspect and describes installation, configuration, and application of the software. Based on a small example, different checks are presented and the report generated by DM.inspect is discussed. Finally, the integration of DM.inspect into the simulation data management system at Porsche is briefly described.

2 ‘Quality Control’ in CAE

The term “quality control” is defined in ISO 9000 as "A part of quality management focused on fulfilling quality requirements" [1]. Performing quality controls means to review the quality of all factors involved in a production process. Applying this definition to CAE, the generation of input files for a Finite Element program – the production process – can undergo quality control, since their properties will directly affect the quality of the simulation results.

In other terms, quality control for CAE input files means to improve the reliability of simulation results by enforcing rules regarding the choice and the configuration of specific features provided by the CAE-software. This may include aspects which directly influence the results such as mesh-size and -quality, element formulations, material models, contact types, among others. Furthermore, the existence and correct denomination of evaluation definitions (history nodes, cross sections, etc.) is also very important for the integration of the simulation model and its results in the often rather complex CAE-processes.

3 DM.inspect

3.1 General Software Description

DM.inspect is programmed in Python and is distributed as a stand-alone executable available for Windows and Linux. It is typically executed from the command line. Besides the input to be checked, it expects a configuration file where the required checks are defined. Also, DM.inspect can be integrated into a simulation data management (SDM) system. At Porsche, DM.inspect is seamlessly integrated into SCALE.model [2][3], the SDM solution from Scale [4]. (SCALE.model was formerly known as LoCo). The tool entirely runs in batch mode without a graphical user interface requiring no human interaction.

The checks are defined in a configuration file in YAML-format as shown in Fig.1:. The file is structured in different check categories (*primer*, *dyna*, *additional*, …) that will be introduced in the following sections. Each check is specified using a unique identifier and can be classified as '*error'* or '*warning'* following the terminology used by LS-DYNA. The parameter '*description*' is optional and may be used for documentation.

```yaml
checks:
  primer:
    TABL_012:
      description: "non-monotonic values in table"
      mestype: warning
    XSEC_060:
      description: "cross section cuts no elements"
      mestype: warning
  dyna:
    '20446':
      description: "x-axis reverses direction"
      mestype: error
    '20006':
      description: "no output interval defined for D3PLOT"
      mestype: warning
  additional:
    added_part_mass:
      description: "added mass of part exceeds 'ratio'"
      mestype: warning
      ratio: 0.10
```

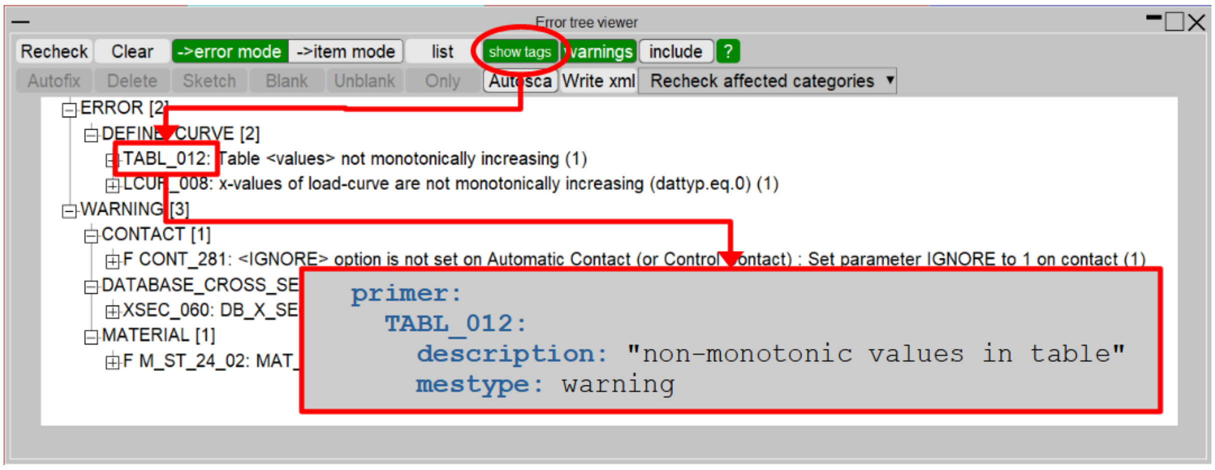
Fig.1: Extract from YAML configuration file for DM.inspect

3.2 Check-Phases

3.2.1 Pre-Phase

The model checks belonging to the pre-phase are currently performed by utilizing the check capabilities of the LS-DYNA pre-processor Oasys PRIMER [5]. Since PRIMER requires a separate license, this part of DM.inspect is optional and may be skipped by a command line option. PRIMER provides more than 7000 LS-DYNA specific checks and may entirely run in batch mode, writing the check results to a file in JSON-format, which allows for a seamless integration into DM.inspect. The different checks are specified in the configuration file by certain identifiers, referred to as '*tags*' in PRIMER. Running PRIMER's checks interactively allows the user to find the desired tags and configure DM.inspect accordingly, see Fig.2:.

One of the ideas behind the pre-phase is to find possible modeling errors as early as possible, i.e., prior to the LS-DYNA initialization run in the next phase. PRIMER has been the first choice as a pre-processor since it is perfectly suited for LS-DYNA models. Other pre-processors may be supported in future releases of DM.inspect and can then either replace PRIMER or work besides PRIMER.



```
ERROR [2]
  DEFINE_CURVE [2]
    TABL_012: Table <values> not monotonically increasing (1)
    LCUR_008: x-values of load-curve are not monotonically increasing (dattyp.eq.0) (1)
WARNING [3]
  CONTACT [1]
    F CONT_281: <IGNORE> option is not set on Automatic Contact (or Control Contact) : Set parameter IGNORE to 1 on contact (1)
  DATABASE_CROSS_SE
    XSEC_060: DB_X_SE
  MATERIAL [1]
    F M_ST_24_02: MAT_
```

```
primer:
  TABL_012:
    description: "non-monotonic values in table"
    mestype: warning
```

Fig.2: PRIMER-GUI error tree view and transfer to DM.inspect YAML file

3.2.2 Simulation-Phase

In the following phase, an LS-DYNA initialization run is performed using the provided input. As every external application, the command for executing LS-DYNA can be specified individually by the user. For standard (explicit) applications, it is recommended to use LS-DYNA's command line option '`mcheck=y`' where the program will only run for 10 cycles typically without occupying any licenses [6]. For implicit applications, '`mcheck=y`' will only perform initialization without running the first cycle which will not produce sufficient output for the subsequent steps. The option '`ncycle=1`' is preferable here.

After the initialization is completed, the LS-DYNA output files (*d3hsp*, *d3plot*, *mesXXXX*) are processed using '*check-hsp*' from DYNAmore's LS-DYNA-Tools [7]. With the command line option '`-xml`', this tool writes an XML-file, collecting model information from the above-mentioned output files. The integration of further tools in DM.inspect, such as 'check-c' for contact definitions or 'check-failed' for failed elements, is planned.

Besides expecting a normal termination of the initialization run, DM.inspect allows to check for any LS-DYNA warning in the check category '*dyna*'. For example, the warning

```
*** Warning 20446 (STR+446)
     load curve 211 x-axis reverses direction at point, 102
     with x-ordinate, 0.0000E+00. Please check that this curve
     definition is correct.
```

can be captured by DM.inspect by inserting the following in the YAML-File:

```
dyna:
    '20446':
      description: "x-axis reverses direction"
      mestype: error
```

In this way, any LS-DYNA warning can even be promoted to an error via '*mestype*'.

3.2.3 Post-Phase

Most checks belonging to the post-phase use information from the XML-file created by 'check-hsp' during the simulation-phase. Here the user can, for instance, limit the allowed amount of added mass due to mass scaling, enforce the existence and correct designation of output definitions (history nodes, cross-sections, etc.), check for name, value, and type of specific parameters, among others. Also, specific attributes for parts regarding element formulations, hourglass definitions, etc., can be specified as well as a list of forbidden material models.

For several checks, the input file is parsed directly, such as identifying encrypted content, or explicitly expected or forbidden keywords. Also, the existence of specific part sets, for instance, required to contribute to global contact definitions, is checked by reading the keyword file since this information is currently not available from the d3hsp file.

Checking the compliance with specified numbering ranges is also located in the post-phase, even if performed with the help of the pre-processor LS-PrePost. If preferred, this should also be possible utilizing PRIMER in the pre-phase.

3.3 Installation/ Configuration

Before the first start, DM.inspect must be configured, since, as described in the previous sections, a couple of external applications are supposed to be executed during one batch check. The program itself is distributed as a single binary available for Windows and Linux and does not require any further installation procedure. Additionally, a license file is provided whose location is to be specified by setting an environment variable.

The external applications (PRIMER, LS-DYNA, check-hsp, etc.) may be installed differently in each software environment, so the code expects a configuration file '*DM.inspect.conf*', located in the installation directory, next to the binary. Here, the execution commands as well as some individual settings for the external applications, are specified. Further instances of this file may be located in the user's home directory ($HOME in Linux, %USERPROFILE% in Windows) and in the current working directory, i.e., where the check is performed, respectively. Settings from the working directory supersede settings from the home directory which again overrule what is defined in the installation directory. This allows to adjust the configuration settings individually for each user.

Especially for the LS-DYNA command, it has proven beneficial to provide a separate script for submitting the initialization run. Then it is even possible to run DM.inspect on a Windows environment while submitting the job to a Linux cluster, which is a common setup for many users.

4 Demonstration Example

4.1 Example description

In the following example, DM.inspect will be used to check the small example model, shown in Fig.3:. The model consists of a plate modelled with shell elements impacted by four different solid and shell parts. All the structural information (nodes, elements, material cards, etc.) is located in a single include file '*impact.k*' which is referenced in the main input '*main.k*'. Besides the `*INCLUDE` keyword, the main file only contains control and database cards.

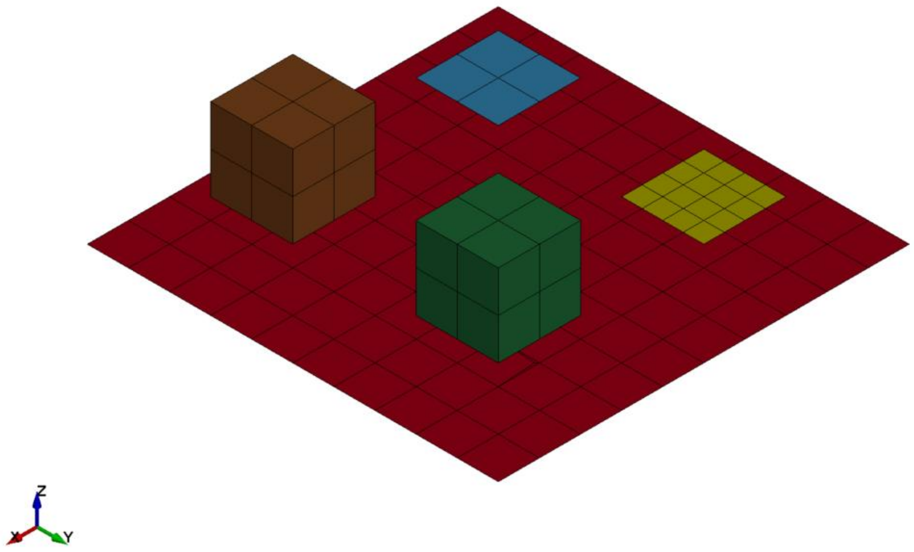


Fig.3: Demonstration example for DM.inspect

Several checks are specified in the file '*demo.yaml*' which will be discussed in detail in the following sections. DM.inspect is executed with the command

```
<DM.inspect> -i main.k -k impact.k -c demo.yaml
```

4.2 Check-Phases

4.2.1 Pre-Phase

After initialization, the tool reads the whole model ('*main.k*') in PRIMER and performs a full model check writing the results to a JSON-file. These results are then scanned for the messages specified in the YAML-file shown in Fig.4:. In the present example, the model is checked for non-monotonic values in `*DEFINE_TABLE` (`TABL_012`) as well as `*DATABASE_CROSS_SECTION` definitions not cutting any elements (`XSEC_060`). Both items are specified as warning.

```
primer:
  TABL_012:
    description: "non-monotonic values in table"
    mestype: warning
  XSEC_060:
    description: "cross section cuts no elements"
    mestype: warning
```

Fig.4: PRIMER-checks defined in 'demo.yaml'

As long as every reported message is of type '*warning*', the check will continue with the next phase afterwards. If at least one 'error' is found in the pre-phase, DM.inspect will terminate before entering the next phase. The general idea is to specify the criteria strictly enough to guarantee a successful initialization with LS-DYNA.

If PRIMER is not available, this phase can entirely be skipped via the command line option '`--nopri`' or the configuration file '*DM.inspect.conf*' and DM.inspect will directly start with the second phase.

4.2.2 Simulation-Phase

The second phase starts with an LS-DYNA initialization run using '*main.k*' as input and the command line option '`mcheck=y`'. After 10 cycles and a normal termination, '*check-hsp*' is called with the option '`–xml`', writing the model information to an XML-file. For the example, this XML-file is scanned for two different LS-DYNA warnings as shown in Fig.5:. While a missing output interval for the d3plot lead to a warning, curve definitions with non-monotonic x-values are considered an error.

```
dyna:
  '20446':
    description: "x-axis reverses direction"
    mestype: error
  '20006':
    description: "no output interval defined for D3PLOT"
    mestype: warning
```

Fig.5: Checks for simulation phase defined in 'demo.yaml'

An error termination of the initialization run will always be considered an error for DM.inspect, this does not have to be specified explicitly. If possible, the reason for the error termination will be reported by DM.inspect.

4.2.3 Post-Phase

The third and last phase is performed after the initialization run and currently provides four different groups of checks. For the example, some checks are picked from each group, see Fig.6: for the corresponding lines of the configuration file '*demo.yaml*'.

The '*additional'* checks are based on the XML-File, written by check-hsp and provide criteria beyond the evaluation of warnings from the category '*dyna*'. The first check, '*added_part_mass*', limits the ratio of added mass due to mass scaling and actual mass for each part to a specified value. With ‘*smallest_time_step*’, we compare the actual compute time step size with the smallest time step size as reported in the d3hsp file. Both checks only make sense in the presence of mass scaling, which is, for instance, typically used in automotive crash simulation. With ‘*part_attributes*’, we can specify allowed combinations of element formulations and corresponding settings for different element types. For the example, we expect fully integrated shell parts (`ELFORM=16`) to use hourglass type 8 and at least 2 but not more than 7 integration points through the thickness. Reduced integrated shell parts (`ELFORM=2`) are supposed to have hourglass type 4 and between 3 and 5 integration points.

```yaml
additional:
  added_part_mass:
    description: "added mass of part exceeds 'ratio'"
    mestype: warning
    ratio: 0.10
  smallest_time_step:
    description: "smallest time step smaller than 'ratio'*solution_time_step"
    mestype: warning
    ratio: 0.50
  part_attributes:
      description: "check if parts are defined using allowed settings (by elform)"
      shell:
          elforms:
              - 16: {hgtyp: 8, nip_min: 2, nip_max: 7}
              - 2: {hgtyp: 4, nip_min: 3, nip_max: 5}
          mestype: warning
thdata:
  duplicate_th_labels:
    description: "check for duplicate labels"
    mestype: warning
    check:
      - node
      - element
      - cross_section
      - contact
  no_labels:
    description: "check for history entities without labels"
    mestype: warning
    check:
      - node
      - element
      - cross_section
      - contact
numbering:
  node:
    description: "check numbering range of node keyword"
    mestype: warning
    minid: 100
    maxid: 999
  element_shell:
    description: "check numbering range of element shell keyword"
    mestype: warning
    minid: 1
    maxid: 99
keyword:
  keyword_exists:
      description: "check for the existence of a specific keyword"
      required:
          - [airbag, warning]
      forbidden:
          - [title, warning]
  crypted_content:
      description: "reports line numbers where PGP-Blocks start"
      mestype: warning
```

Fig.6: Checks for Post-Phase defined in ‘demo.yaml’

The group ‘*thdata*’ provides checks regarding the definition of time history outputs, e.g., via `*DATABASE_HISTORY_NODE`, `*DATABASE_CROSS_SECTION`, etc. With ‘*duplicate_th_labels*’ we do not allow two time-history definitions to use the same label which may be a problem for post-processing. With ‘*no_labels*’, only time-history definitions with labels will be accepted.

The allowed numbering range for different entities can be defined using checks from the group ‘numbering’. In the example, nodes may be numbered within the range 100-999 while shell elements are only allowed from 1 to 99.

For the checks in the last section, ‘*keyword*’, the required information is taken directly from the checked include file specified through the command line option ‘`-k`’. Here, we expect the input to contain at least one keyword `*AIRBAG` while a keyword `*TITLE` will lead to a warning. Additionally, encrypted content will not be accepted within the input. If found, DM.inspect will report the line where the encrypted content starts.

4.2.4 Results in report file

The check results are reported in an ASCII file ‘*input.k.report*’ where the file name is generated from what is defined via the command line option ‘`-k`’. Fig.7: exemplarily shows the report for the category ‘*additional*’.

First of all, this report is supposed to help identifying and fixing possible mistakes in the input. Furthermore, it may also serve as a record to document the compliance of the model with certain requirements. In practice, a company may provide a certain DM.inspect configuration file to an external supplier stipulating that the models delivered must pass all checks defined by that configuration file. This strategy helps to avoid the cumbersome and time-consuming process of sending models back and forth until the models satisfy the company’s requirements.

```
[...]
----------------------------------------------------------------------
4: reporting additional messages of type "warning"
----------------------------------------------------------------------

4.1. (added_part_mass):
   added mass exceeds 0.1 x structural mass for Parts
   # 4

4.2. (smallest_time_step):
   smallest time step is 0.145 * compute time step
    --> ratio is smaller than 0.5

4.3. (part_attributes_shell):
   For the following shell parts, the defined hourglass formulation
    is not allowed:
   # 2 (expected: 8)
[...]
```

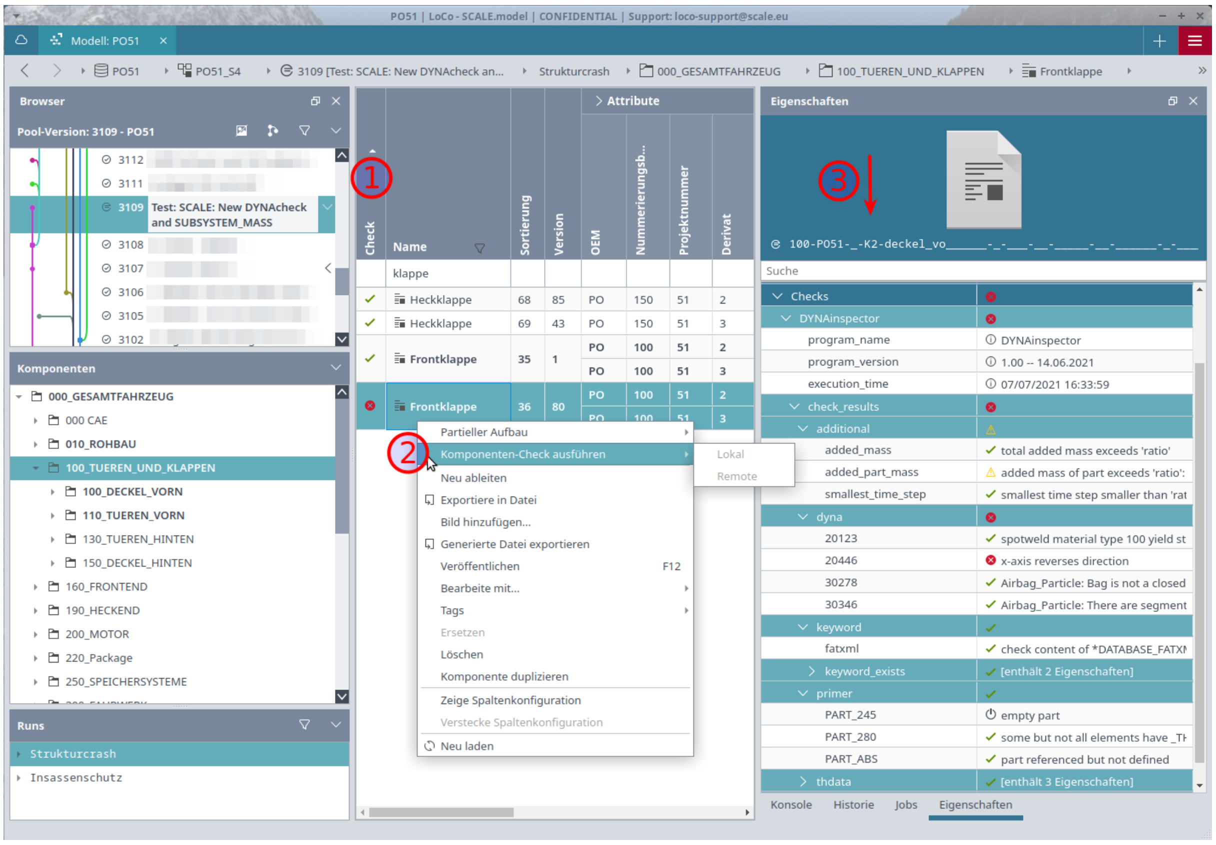
Fig.7: Reported warnings for check category additional

5 DM.inspect integration into Porsche’s Simulation Data Management System

At Porsche DM.inspect has been tightly integrated into the simulation data management (SDM) system SCALE.model [4] (SCALE.model was formerly known as LoCo). SCALE.model has been developed in close collaboration with the German automotive industry and provides features for model assembly, version control, access control and distribution of simulation data.

Fig.8: shows a screenshot of the SCALE.model client software. In the center of the image four lines corresponding to four components are shown. Each component represents a sub-model of a crash simulation vehicle model. The four components in Fig.8: are two hatch doors (“Heckklappe”) and two front hoods (“Frontklappe”). Each of these components has been checked by DM.inspect and the check results are shown in the first column labeled “Check”, symbol ① in Fig.8:. The green check mark next to the first three components indicates successful checks, while the red “X” next to the fourth component indicates that at least one check returned an error.

At Porsche, SCALE.model is configured not to submit models with errors to the High Performance Computing system. Therefore, the user must resolve the problem behind the error reported for the fourth component and recheck.



Check	Name	Sortierung	Version	OEM	Nummerierungsb...	Projektnummer	Derivat
	klappe						
✓	Heckklappe	68	85	PO	150	51	2
✓	Heckklappe	69	43	PO	150	51	3
✓	Frontklappe	35	1	PO	100	51	2
				PO	100	51	3
✗	Frontklappe	36	80	PO	100	51	2
				PO	100	51	3

Checks	✗
DYNAinspector	✗
program_name	DYNAinspector
program_version	1.00 -- 14.06.2021
execution_time	07/07/2021 16:33:59
check_results	✗
additional	⚠
added_mass	✓ total added mass exceeds 'ratio'
added_part_mass	⚠ added mass of part exceeds 'ratio':
smallest_time_step	✓ smallest time step smaller than 'rat
dyna	✗
20123	✓ spotweld material type 100 yield st
20446	✗ x-axis reverses direction
30278	✓ Airbag_Particle: Bag is not a closed
30346	✓ Airbag_Particle: There are segment
keyword	✓
fatxml	✓ check content of *DATABASE_FATXM
keyword_exists	✓ [enthält 2 Eigenschaften]
primer	✓
PART_245	⏻ empty part
PART_280	✓ some but not all elements have _TH
PART_ABS	✓ part referenced but not defined
thdata	✓ [enthält 3 Eigenschaften]

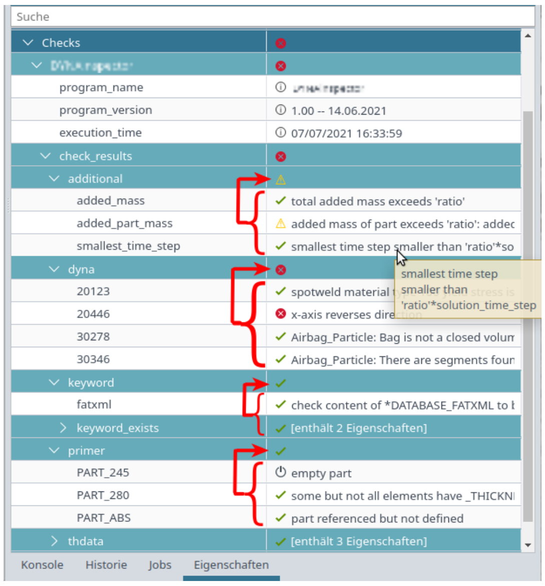
Fig.8: Screenshot of the SCALE.model client software. Parts of the screenshot have been blurred intentionally.

To recheck, the user simply right-clicks on the component and chooses “run component check” in the context menu, symbol ② in Fig.8:. In the background, SCALE.model assembles a minimum viable model including the selected component und calls DM.inspect. Shortly after that, check results for each individual check are shown directly in the SCALE.model client, symbol ③ in Fig.8:. The green tick mark indicates “*passed*”, the yellow triangle indicates “*warning*“, the red cross indicates “*error*“ and the power symbol indicates “*deactivated*“.

The results of individual checks are not only displayed but also aggregated in SCALE.model. This allows check groups like “primer”, “dyna” or “additional” to display a check result symbol as shown in Fig.9: (a). This allows the user to quickly spot areas of concern and take appropriate action.

The check result “*deactivated*“ mentioned above merits further explanation. In the DM.inspect configuration file, individual checks can be assigned a “component type context”. This allows, for instance, to switch off a specific check if the component has a certain component type. As an example, in Fig.10: the check with identifier “CONT_041” is deactivated if the component type of the checked component is “dummy” (referring to a crash test dummy). SCALE.model calls DM.inspect with the additional command line argument “`--comptype`” to enable this feature.

There are different motivations for using DM.inspect’s “component type context” feature. Company modeling guidelines may simply state different sets of rules for different component types. In other cases, the compliance with certain rules may be desirable but cannot be readily achieved because the component in question is provided by an external supplier.

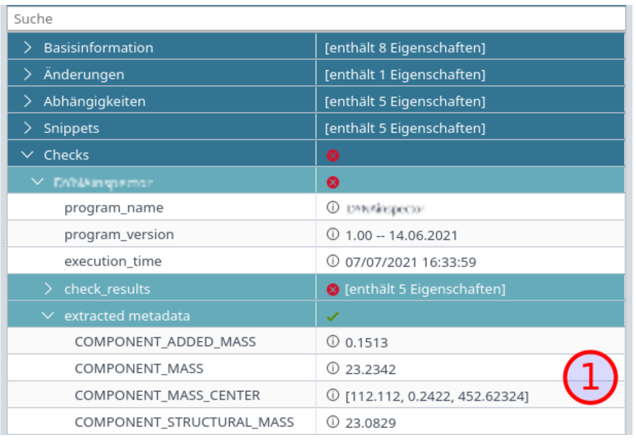


Suche

Checks	✗
[redacted]	✗
program_name	ⓘ [redacted]
program_version	ⓘ 1.00 -- 14.06.2021
execution_time	ⓘ 07/07/2021 16:33:59
check_results	✗
additional	⚠
added_mass	✓ total added mass exceeds 'ratio'
added_part_mass	⚠ added mass of part exceeds 'ratio': addec
smallest_time_step	✓ smallest time step smaller than 'ratio'*so
dyna	✗
20123	✓ spotweld material t
20446	✗ x-axis reverses dire
30278	✓ Airbag_Particle: Bag is not a closed volum
30346	✓ Airbag_Particle: There are segments foun
keyword	✓
fatxml	✓ check content of *DATABASE_FATXML to l
keyword_exists	✓ [enthält 2 Eigenschaften]
primer	✓
PART_245	⏻ empty part
PART_280	✓ some but not all elements have _THICKN
PART_ABS	✓ part referenced but not defined
thdata	✓ [enthält 3 Eigenschaften]

smallest time step smaller than 'ratio'*solution_time_step

(a)



Suche

Basisinformation	[enthält 8 Eigenschaften]
Änderungen	[enthält 1 Eigenschaften]
Abhängigkeiten	[enthält 5 Eigenschaften]
Snippets	[enthält 5 Eigenschaften]
Checks	✗
[redacted]	✗
program_name	ⓘ [redacted]
program_version	ⓘ 1.00 -- 14.06.2021
execution_time	ⓘ 07/07/2021 16:33:59
check_results	✗ [enthält 5 Eigenschaften]
extracted metadata	✓
COMPONENT_ADDED_MASS	ⓘ 0.1513
COMPONENT_MASS	ⓘ 23.2342
COMPONENT_MASS_CENTER	ⓘ [112.112, 0.2422, 452.62324]
COMPONENT_STRUCTURAL_MASS	ⓘ 23.0829

(b)

Fig.9: (a) Aggregation of DM.inspect check results in SCALE.model.
(b) Component mass und component mass center.

```
checks:
  primer:
    CONT_041:
      description: "nodes on both master and slave side"
      mestype: error
      context:
        component_types: ['dummy',]
        action: exclude
```

Fig.10: Check definition with component type context.

At Porsche, besides its main purpose, DM.inspect is also used to determine the component's mass and mass center (as computed by LS-DYNA) as shown in Fig.9: (b), symbol ①. When the user later assembles a complete vehicle model, the expected mass and mass center of the complete model are computed from the components' values. Subsequently additional non-structural mass is automatically distributed in order to obtain a certain target vehicle mass and target vehicle mass center.

6 Summary and Outlook

DM.inspect has been presented as a tool to check LS-DYNA input files, once the quality criteria, i.e., the desired checks are defined and properly configured by the user. Various checks from different categories are accessible where the applied criteria must be specified and adjusted individually. Especially when working with large models assembled from many sub-models, as it is usually the case in automotive crash simulation, DM.inspect can be integrated in the development process of the sub-models to ensure and enforce certain quality criteria already during their setup. Finding possible issues early in the development process increases the quality of the later assembled full-models and saves time required for cumbersome debugging.

At Porsche, DM.inspect is seamlessly integrated in the simulation data management software SCALE.model and therefore efficiently supports the CAE vehicle development process on component level. The application as a command-line application is also possible and allows DM.inspect to be integrated into other software environments or applied as a stand-alone tool.

The requirements for different CAE simulation disciplines can vary widely and, even for different components of one simulation model, it's hardly possible to define one set of general rules. Thus, DM.inspect does not contain any hard-coded criteria the models are checked against. By setting up the described configuration file as an input for DM.inspect, the user must individually select the checks and specify the criteria suited for the respective application.

The tool is rather new and still under development, so further checks and functionalities will be implemented in the future. Also, the integration of further pre-processing tools besides PRIMER is currently discussed. The results are currently written in the form of a report file in ASCII-text. For the SCALE.model integration at Porsche, the output of a machine-readable format (JSON) has been implemented already, but further output formats (such as XML) may be possible as well if requested.
A manual, where the available checks are documented together with their required YAML-syntax, is also currently under development and will be available soon.

7 Literature

[1] https://en.wikipedia.org/wiki/Quality_control, 2021

[2] M. Koch, S. Mattern, R.D. Bitsche: *Facing Future Challenges in Crash Simulation Engineering – Model Organization, Quality and Management at Porsche*, 15th International LS-DYNA Conference, Detroit, USA, 2018

[3] M. Koch, S. Mattern, R.D. Bitsche: *Model Organization, Quality and Management for Crash Simulation Engineering on different Vehicle Body Lines at Porsche*, German LS-DYNA Forum 2018, Bamberg, Germany, 2018

[4] SCALE.model, Scale GmbH: https://www.scale.eu/en/products/loco-en, 2021

[5] PRIMER, Oasys Ltd.: https://www.oasys-software.com/dyna/software/primer/, 2021

[6] LS-DYNA Keyword User's Manual, Volume 1, Livermore Software Technology (LST), An Ansys Company, 2020

[7] LS-DYNA-Tools, DYNAmore GmbH: https://www.dynamore.de/en/products/pre-and-postprocessors/tools, 2021