Appendix X: Multistage Analysis

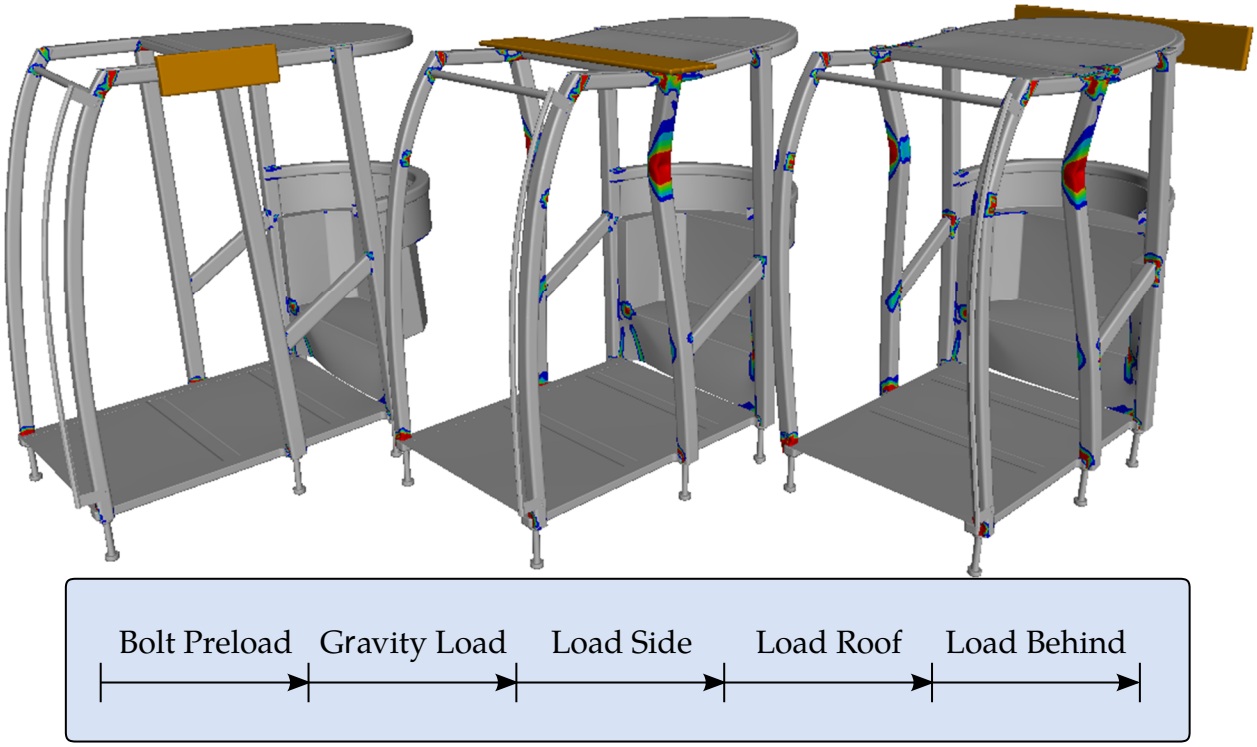

Figure 75-1. Example of Roll-Over Protection Systems simulation with Bolt Preload and Gravity Load

WHY USE STAGES

A typical LS-DYNA simulation is a boundary value problem with initial conditions. It is solved from an initial time 0 (zero) to an appropriate end time T. The initial configuration is usually assumed to be *stress free*, which basically means that all internal variables in all active features are trivially initialized. If the application is sufficiently complex, setting up the input can be cumbersome. The resulting simulation can also be difficult to tackle. In those situations, we can divide the entire problem into several *stages.* A stage is associated with an isolated and well-defined phase of the real physical process. In this sense, a stage naturally defines a *standard* analysis in the following ways:

- No birth or death time in any feature[2]
- Few and simple boundary conditions
- Only implicit or only explicit with no switching between

[2] The only exception is turning on and off dynamics in implicit, which is regarded as a stabilization technique for solving implicit static analyses. See Appendix P for details.

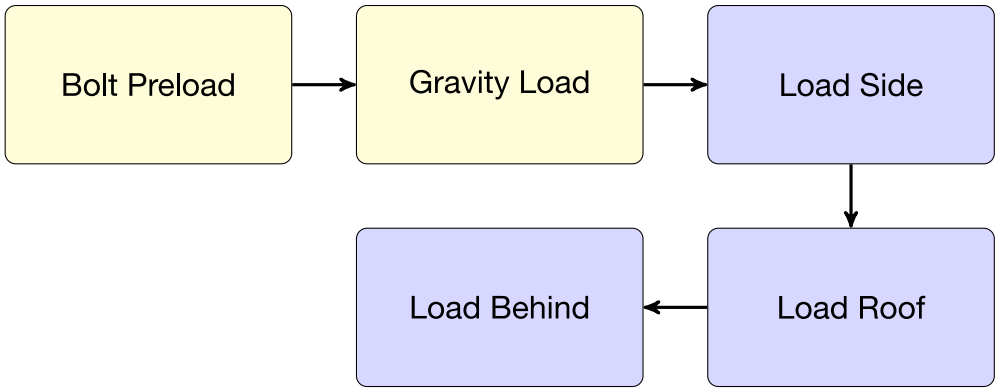

Figure 75-2. Flowchart of how to separate the ROPS test simulation into 5 stages

- Using well defined loadings with simple curves

Each stage is itself a boundary value problem with initial conditions. The only caveat is that the initial configuration may now be *prestressed* because of what occurred in the previous stage. In other words, internal variables must be propagated from the previous stage in order to make sense of future results. Apart from this, each stage can be executed and investigated separately instead of having to complete the entire simulation in a single run.

Example of Splitting into Stages:

To make things less abstract, consider the simple example of a hypothetical cab design subjected to a *ROPS (Roll-Over Protective Systems)* test, ISO requirement 3471, as depicted in Figure 75-1. The legal requirements state conditions for sequential quasi-static loads from the *side*, the *roof* and *behind*, which are graphically illustrated. To make things a little more interesting, we add *bolt preload* and *gravity load* before applying the external loads. Assume for the sake of simplicity that the entire process takes 10 seconds. As indicated, completing the entire simulation in one single run would require everything to be set up accordingly, with the potential risk of having to restart from the beginning in case of execution failure.

The alternative is to identify the 5 stages. Assuming each stage takes 2 seconds, you then set up a separate input for all of them. You run these input files sequentially in the order they apply. Figure 75-2 illustrates this method for setting up the simulation. For the moment we don't worry about the problem of inheriting the internal variables but instead think of what advantages this approach might bring:

- If the simulation fails, restart can be made from the last successful stage instead of rerunning the entire simulation. Thus, turnaround time decreases.
- Some stages may require only a subset of the entire model. Thus, this method reduces simulation time.

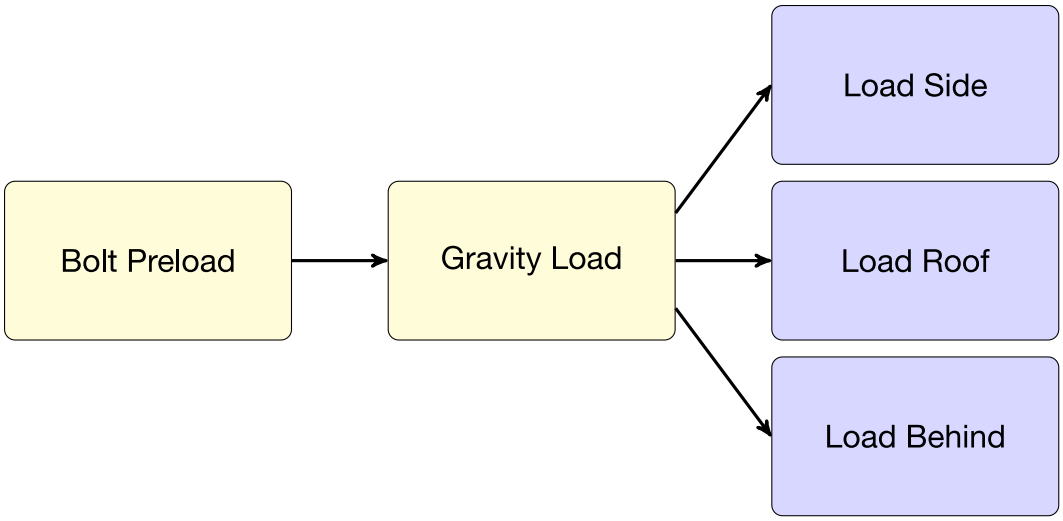

Figure 75-3. Flowchart of a multiple case staged analysis

- Some parts can be rigid during stages, saving simulation time.
- Data from a stage can be reused for many different types of simulations, such as different load cases and eigenvalue analyses.
- A prestressed component can be duplicated in subsequent stages.

So far, we only considered the situation where the stages are executed sequentially, and each case depends on the results of the previous. It is, however, easily seen that a *multiple case* analysis with preloads fits within this framework. Assume, for instance, that you want the response from each of the three external loads *independently*, but with the effect from both bolt preload and gravity load. This situation is illustrated in Figure 75-3. The only difference is that internal variables resulting from the gravity load is imported to each of the three different load cases. The three load cases can be executed in parallel if you have sufficient computer resources.

Simulating a particular stage is on the one hand rather straightforward, but we are left to explain how internal variables are exported from one stage and imported to the next. Although the principles are simple and much can be figured out from logical reasoning, certain situations and features call for a more thorough explanation. With this appendix, we intend to provide the basics and then meticulously go through each feature and situation that requires a more in-depth treatment. We will also explain how all stages can be simulated seamlessly without manual interaction, which is handy if submitting overnight runs.

THE "DYNAIN" APPROACH

The "dynain" approach is a well-known concept among experienced LS-DYNA users, especially in the forming simulation community. The common usage is to output

geometry and stresses of a blank from a forming simulation to a file called dynain, and then include this file in a subsequent simulation, such as springback. This is the simplest form of a multistage analysis, that is, porting stress between two simulations. Here we describe this approach for general situations.

Formats:

The first thing to observe is that the format of a dynain file can be one out of three: *ascii, binary* or *lsda* format.

1. The *ascii* format is popular because you can open it in a text editor and, thereby, inspect and edit the file. It is not, however, capable of representing the full *system state* of a particular simulation result. The *system state* is not only stresses and strains, but everything that is needed to seamlessly continue the process without losing any information. In particular, it requires the state from the contacts, like friction history and information from tied contacts.

2. The *binary* format is not used extensively and is therefore of no interest here.

3. The format we want to use for multistage analysis is the *lsda* format, since it handles most of the internal variables for obtaining an acceptable result. From now on we will refer to the file as the dynain.lsda file.

LSDA format:

The dynain.lsda file is in a sense "just" a regular keyword file. It is bound to the rules and regulations for that particular reader.

It is generated by LS-DYNA from a previous run, but the content is whatever you may find in any other keyword file, such as nodes, elements, initial stresses, etc. It can be included from any other keyword file. Given this prerequisite, and that you are familiar with the keyword format, much of what is about to be presented will make sense from your own experience.

It may contain keywords that are not available in the ascii format. These keywords are for representing the system state as accurately as possible. In particular, it contains stabilization forces and contact history. These are yet to be documented, but the lsda format does not currently have documentation, so this is not a showstopper.

Finally, it is a high-level *binary* format. The content is organized like a file system. To process it, you need an lsda *reader*. Both LS-PrePost and LS-DYNA have built-in lsda readers, but at this moment LS-PrePost is not able to *interpret* all the data correctly. We have an external reader called *ioq*, that can be used to parse through the hierarchical structure of the file with standard linux commands such as *cd, ls* and *cat*. This reader cannot

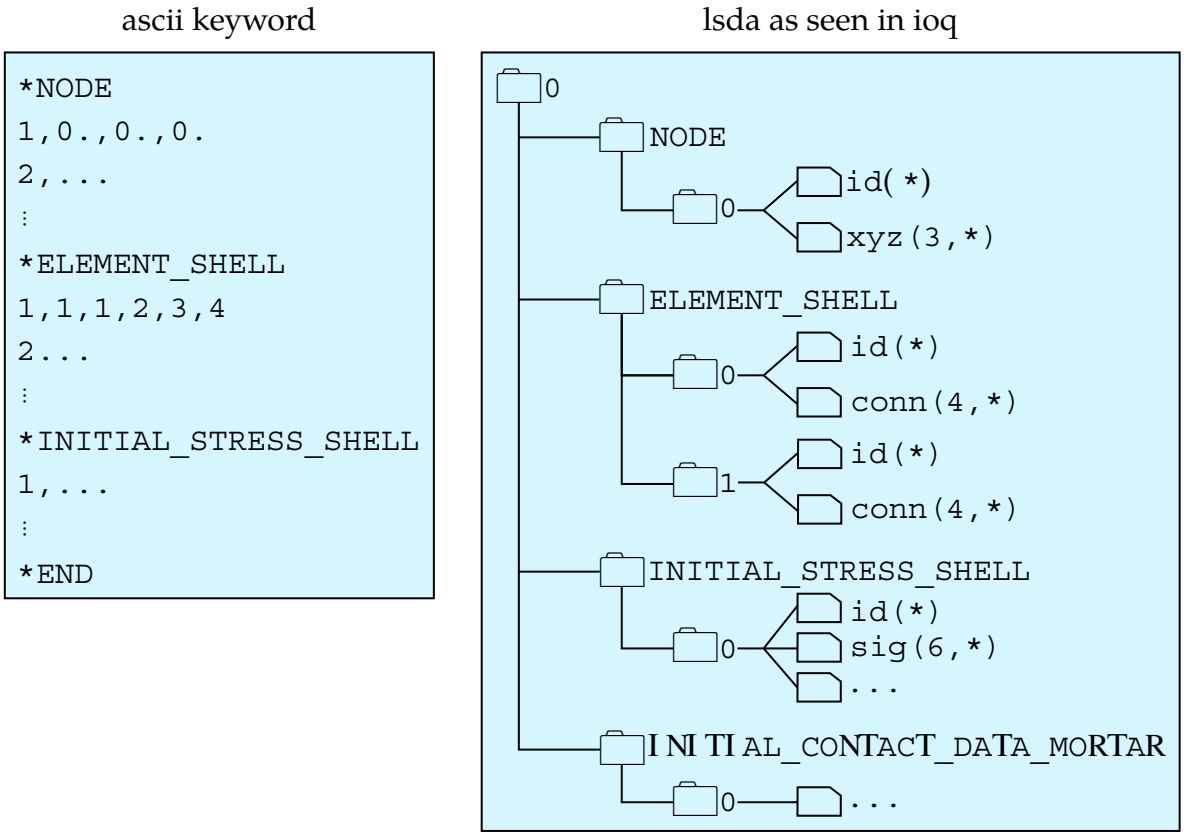

Figure 75-4. Comparison of the structure of *ascii* and *lsda* formats for the dynain file

edit the file. Figure 75-4 schematically illustrates the contents of a dynain file for the ascii and lsda formats, respectively. Note that the enumerations "0" and "1" for some directories are for organizational purposes. We also highlight that the lsda file may contain keywords not supported by the ascii format (here exemplified by INITIAL_CONTACT_-DATA_MORTAR).

Discussion

Juggling many stages soon becomes overwhelming. We recommend giving each stage a *job ID*, which will put a prefix on the dynain.lsda (and all other output) files so that you can distinguish one stage from another. One may either use jobid=A on the command line when executing the stage, or use

```
*KEYWORD_ID
A
```

in the main input file to make sure the dynain.lsda file will be named A.dynain.lsda to indicate that this is data from stage *A*. Using *CASE statements is another way of organizing the stages. *CASE will be described later.

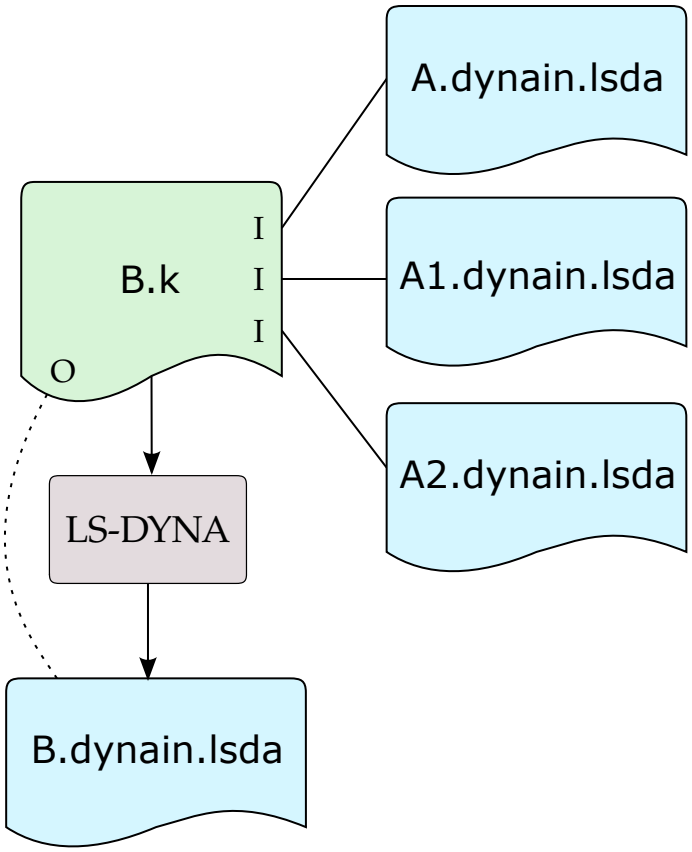

Figure 75-5. Basic schematic of including dynain.lsda files from previous stages in stage *B*. The I in the B.k box stands for an instantiation of *INCLUDE. In this case, A.dynain.lsda, A1.dynain.lsda, and A2.dynain.lsda are output by previous stages and are included in stage *B*. The O in the figure stands for output through *INTERFACE_SPRINGBACK_LSDYNA. With this keyword in the input deck for stage *B*, LS-DYNA exports B.dynain.lsda.

Remember that dynain.lsda *represents* the keyword format in a different way than its ascii counterpart, but that it is nevertheless processed by the keyword reader and should be seen as a natural inclusion to a standard keyword input file. A strong argument in favor of this approach, when compared to other multistage functionalities, is *repeatability*. By restricting all inputs to be keyword and be processed by the same reader facilitates maintenance of the software and thus reduces the risk of i/o problems and related bugs. Stages are advantageous since they constitute a *simple* keyword input. With stages, we can avoid complexities associated with activating/deactivating, switching, using complicated curves and other forms of out-of-the-box solutions that are not part of main stream simulations.

This being said, the dynain approach is by no means trivially understood and failsafe, especially without an appropriate graphical user interface (GUI) to aid the user. But with the right information and tools at hand, it is a powerful framework to deal with complicated processes. We now intend to provide this information and present the tools for dealing with multistage analyses. We refer to Figure 75-5 to enhance understanding. In

particular, we will meticulously go through the details for exporting data from one stage and importing this data to the next.

EXPORTING DATA

Referring to Figure 75-5, assume that we are about to simulate stage *B*, represented by the main keyword file B.k together with include files generated by previous *A* stage(s). For the sake of simplicity, we assume that only dynain.lsda files are included, and B.k is the only ascii file. The keyword required for exporting the results from stage *B* for use in a subsequent stage is *INTERFACE_SPRINGBACK_LSDYNA. It should be part of the input in B.k. A more detailed description of this keyword can be found elsewhere in the keyword manual, but here we provide a recommended setup and discuss the various parameters.

Outputting Data with *INTERFACE_SPRINGBACK_LSDYNA

In the two tables below, we list the needed input in the data cards for *INTERFACE_SPRINGBACK_LSDYNA with recommended below. In this section we will give motivation for the recommended values of each of the variables.

Card 1	1	2	3	4	5	6	7	8
Variable	PSID	NSHV	FTYPE					
Value	999	999	3					

Card 2	1	2	3	4	5	6	7	8
Variable	OPTC				NDFLAG	CFLAG	HFLAG	
Value	OPTCARD				1	1	1	

PSID = 999
We recommend reserving 999 (or some other favorite number) as the ID for a part set (see *SET_PART) containing parts to be carried over from stage *B* to stage *C*. The resulting B.dynain.lsda file will have the nodes with coordinates and elements with stresses/forces for *at least* those particular parts at the end of the simulation. More nodes will be output (see discussion of NDFLAG below) but not more elements.

Note that none of the *PART, *SECTION, and *MAT keywords are output to the B.dynain.lsda file. They must be inserted in the main file C.k for stage *C* or the simulation will error terminate. Using the same section and material for the same part throughout the entire process is convenient and often sufficient *but not mandatory*. For instance, when going from a quasi-static implicit stage to a dynamic explicit stage, you may want to switch from a fully integrated element formulation to a formulation with hourglass control. In this case, the stresses will be mapped onto the new integration point configuration, resulting in loss of accuracy. In rare situations you might want to alter the material as well. This situation is discussed in conjunction with NSHV below.

We *strongly* recommend associating part, node and element numbers with the actual physical components and their material locations throughout the process. By this, we mean that if a particular node number or element number is used for a certain material point or region, then the same number should *not* be reused for some *other* material point or region. For instance, say that we have a forming simulation where the tools have node numbers between 1 and 100000, and the blank between 100001 and 200000. Then you decide to only output the blank to the dynain.lsda file because the tools are no longer of importance. In the next stage you want to mount the blank onto a vehicle as part of the construction process, where some *new* parts are introduced to act as the rest of the car. It is tempting to reuse node numbers 1 through 100000 since these numbers are "available" again. You should use new numbers and *not* reuse the numbers because you will eventually run into trouble with duplicated nodes. The same recommendation applies to element and part numbers.

Selecting parts for PSID is difficult since it presupposes knowledge of what will be performed next. Any part that is *not* included in PSID can in general not be reintroduced without resetting stresses and history. Parts that *are* in PSID *will* be included in the next stage no matter what. But what if the situation demands that some parts are of interest in one particular branch of a continuation, and some other parts of interest in another? You might, for instance, want to do several sub-model analyses on disjoint components given the stresses from a global gravity load. You might also forget to include a certain part that you need. You can always output all parts with:

```
*SET_PART_LIST_GENERATE
999
1,99999999
```

and then *exclude* some parts when importing the data in the subsequent analysis. This strategy might lead to a large dynain.lsda file, but it is convenient. We will discuss it further when dealing with importing data, specifically in the section Excluding Parts in Subsequent Stages g.

NSHV = 999

You should set NSHV to a "large" number to include all history variables for the elements contained in the part set PSID above. 999 should be enough assuming no material has no

more than 999 history variables. We assume that the material models used for a particular element in stages *B* and *C* are not necessarily the same but *compatible.*

Compatible material models are those for which the mapping of stresses and history variables make physical sense. It is difficult to know which ones *are* compatible without consulting expertise. Generally, we only recommend switching from rigid to some deformable material model. This switch is useful when the deformation of some parts can be neglected for some stages. These parts can be defined as rigid until the stage of interest begins. All the stresses and histories for these parts will be trivially initialized. Note that we do not recommend switching a deformable part to a rigid part. A part that is deformable for some duration and then switched to rigid will be *reset* in terms of stress and history variables.

FTYPE = 3
Setting FTYPE = 3 causes the output dynain file to be in lsda format, which is what we *always* want.

NDFLAG = 1
As mentioned earlier, by default, only the nodes and elements belonging to parts in PSID are output to the dynain.lsda file. This behavior is not always sufficient for adequately continuing a process. For instance, some nodes could be part of nodal rigid bodies or interpolation constraints without being associated to *any* part. The updated locations and orientations of these nodes also need to be carried over between stages. Setting NDFLAG = 1 will make sure that *all* nodes are output to the dynain.lsda file. As long as you follow the practice of not reusing a used node, there should be no reason to not *always* set this. Nodes that for some reason are not hooked up to parts or connectors in remaining stages will simply be constrained or massless and should not affect the outcome of the simulation.

CFLAG = 1
This flag is for output of contact history and tied contact pairs. It should *always* be set to not lose any such information. When this flag is active, *all supported* contacts will be output to the dynain.lsda file, regardless of which ones will be of interest in later stages. The philosophy behind this decision is the same as for elements and nodes discussed above. When importing the data, we simply discard everything that is not hooked up to contact entities in later simulations.

Note that the *CONTACT card itself will *not* be output to the dynain.lsda file, only the segments and their associated data. Therefore, you must copy and paste these lines from one main input file to the next in order for simulations to work properly. The pasting can be done with modifications in ways that will be apparent when we discuss importing data, specifically in the section Contact.

APPENDIX X

Because of the abundance of contact formulations in LS-DYNA, the flag applies only to *some* regular and *some* tied contacts. We will discuss these two classes of formulations separately, starting with regular (non-tied) contacts.

The flag is supported for *all* Mortar contacts, but none of the other contact algorithms. The multiple stage approach is in a sense limited to applications where Mortar contact is the preferred choice or is good enough. In situations, where for instance an implicit gravity load is to be followed by a crash simulation, you may need to switch from mortar to a (faster) contact at the expense of losing contact stress history.

The output is organized in chunks of the form *{contact segment ID, data for this segment}* or *{contact node ID, data for this node}*, depending on the type of contact. Since Mortar contact is a segment-to-segment contact, we only have the former situation at the moment. We will only discuss this situation in the following.

The *contact segment ID* is uniquely defined as the pairing *{contact ID, segment node IDs}*, where *contact ID* is simply the identity of the contact given through the option _ID on the *CONTACT card. To properly map the data, you must specify a contact ID *and* use the same identity for the same contact in subsequent stages. The *segment node IDs* are the node numbers as given by you in the input file, reordered so that the smallest number comes first. The mapping of data occurs when a matching *contact segment ID* is found in the dynain.lsda file and among the actual contacts in the input file.

The *data for this segment* is currently specific to the contact of choice. The format is of little interest to the user. It simply contains enough information for properly migrating the state of the contact between stages, and the mapping is performed with no sophistication. It is, for instance, not possible to map data between different *types* of contacts, which essentially means that the same contact type must be used throughout the entire process. There are *some* allowed operations for changing the behavior of the contact between stages. The section concerning contact in importing data will discuss these allowed operations.

Our implementation for tied contacts follows similar conventions. For instance, the *CONTACT card must be copied between main files for the simulations to work properly. Before delving into further details, we need to elaborate on the extent of support for various tied contact formulations. Explicit and implicit tied contacts are in general different formulations which adds some complications. Explicit tied contacts are *incrementally* objective while implicit are *strongly* objective. Because of this, the internal variables used for keeping track of the tied stress logic is different, and it is nontrivial to map between the two. We added IACC = 2 on *CONTROL_ACCURACY to unite the tied contact formulations. IACC = 2 allows for switching between explicit and implicit with no loss of information. It ensures that the implicit contacts are executed even for explicit analysis. We, therefore, recommend always setting IACC to 2 for multiple stage analysis. If IACC = 1, internal variables are not properly ported. The actual *pairs* will be ported

correctly, but the forces/stresses that have been built up during a stage will not be correct in subsequent stages. As an alternative, you could always set IACC to 0 for which the explicit contacts will be executed even for implicit analysis. IACC set to 0 should also be used if all stages are explicit.

The output to the dynain.lsda file for a tied contact is chunks in the form *{contact ID, node ID, segment node IDs, data}*. As before, *contact ID* is from the option _ID on *CONTACT. *{node ID, segment node IDs}* are the tied contact pairs in user node numbers. The *data* field constitutes the internal variables needed for computing the tied contact forces. As soon as a contact using a specific *contact ID* occurs in the dynain.lsda file, the tied contact pairs for that particular contact are generated solely from the information given therein, and *no* new tied contact pairs are generated even if the logic for tying is fulfilled. All mortar tied contacts are also supported, for which the chunks are instead in the form *{contact ID, segment node IDs, segment node IDs, data}* since the tying occurs between segments and not nodes and segments. Except for this, the same treatment applies.

HFLAG = 1
By default, internal variables associated with *integration points*, such as stresses and material related history, are ported between stages. Stabilization forces, such as drilling and hourglass forces, may depend on their own internal variables. These variables may also need to be transferred for a complete description of the system state. We recommend setting HFLAG to 1 to deal with these internal variables. We will proceed to discuss the two mentioned types of stabilization.

In implicit analysis, *drilling* stabilization is on by default, otherwise shell elements would suffer from incompleteness. Usually, everything is taken care of automatically. If all stages are implicit, we can end the discussion here. If the implicit analysis is followed by an explicit analysis, the drilling stabilization will be turned *off* in the latter since explicit drilling resistance is by default handled through the rotational inertia of the shell element nodes. This transition results in loss of equilibrium. In practice this may not mean much, but, if you are picky, you can deal with this by using DRCPSID on *CONTROL_SHELL to invoke drilling stabilization even in explicit analyses. Doing so for all shell parts leads to force continuity between the implicit and explicit stages.

For elements with reduced integration (shells or solids), stiffness based *hourglass* stabilization may be invoked to complete the element. These hourglass models update the internal forces in a similar fashion to how the integration point stresses are incremented. These forces need to be output to the dynain.lsda file. To maintain equilibrium, all stages need to keep the same element formulation and hourglass model. Otherwise, these forces will be lost in the transition. Again, this may be of little practical importance, but nevertheless a good thing to know.

APPENDIX X

Boundary Conditions

Before discussing the importing of data from the previous stage, we need to emphasize that the two lines

```
*INTERFACE_SPRINGBACK_EXCLUDE
BOUNDARY_SPC_NODE
```

must be present in all stage files. Otherwise, the boundary conditions are inherited from the previous stage. With these lines, we are free to use whatever boundary conditions we like for each stage.

IMPORTING DATA

Let's assume stage *B* has been completed. Then, to import stage *B*'s data for stage *C*, add the lines

```
*INCLUDE
B.dynain.lsda
```

to the main file C.k. Assuming no conflict (that is, all element and node numbers are unique), you can include several files, each resulting from separate previous runs

```
*INCLUDE
B.dynain.lsda
B1.dynain.lsda
B2.dynain.lsda
…
```

You can also include the same file several times, but the data must be transformed differently for each include so that again there are no conflicts

```
*INCLUDE
B.dynain.lsda
*INCLUDE_TRANSFORM
B.dynain.lsda
*INCLUDE_TRANSFORM
…
```

Adding these lines is in principle how importing data works with the catch that the include files need to be complemented with accompanying *PART, *SECTION, *MAT, *CONTACT, etc. cards in the main file C.k to render a sensible continuation of the process. In this section we lay out the details concerning which keywords need to be reinserted into C.k and which ones appear in the include file B.dynain.lsda from the previous stage.

Excluding Parts in Subsequent Stages:

First, let's continue the discussion about PSID = 999 from the section on Outputting Data with *INTERFACE_SPRINGBACK_LSDYNA. Recall that only the parts specified in this particular part set in B.k are output to the B.dynain.lsda file. This requirement could be a problem if the parts to be used in subsequent stages are not known *a priori*, or if you want to perform several sub-modelling analyses using data from the same previous run.

We previously indicated that you may output *all* parts and then use only those of interest in subsequent stages. This strategy will circumvent this conundrum. For it to work you will need to include *CONTROL_LSDA in the main file C.k. For details we refer to the corresponding keyword description elsewhere in the manual, but its application is best explained through a simple example.

For the sake of simplicity, assume that stage *B* consists merely of three parts, enumerated from 1 to 3. By include the following in B.k,

```
*SET_PART_LIST
999
1,2,3
*INTERFACE_SPRINGBACK_LSDYNA
999,…
*PART
Part 1
1,1,1
*PART
Part 2
2,2,2
*PART
Part 3
3,3,3
```

LS-DYNA will output all three parts to B.dynain.lsda, including their respective nodal positions and integration point data.

We can now in one particular continuation, say stage *C_1*, insert

```
*CONTROL_LSDA
$ Number of parts to exclude
1
$ Part to exclude
3
*PART
Part 1
1,1,1
*PART
Part 2
2,2,2
```

into C_1.k to actually use only parts 1 and 2. And in another, say stage *C_2*, insert

```
*CONTROL_LSDA
$ Number of parts to exclude
2
$ Parts to exclude
1,2
*PART
Part 3
3,3,3
```

into C_2.k to use only part 3.

This approach is illustrated graphically in Figure 75-6. Note that the *PART card for the active parts need to be reinserted in each continuation.

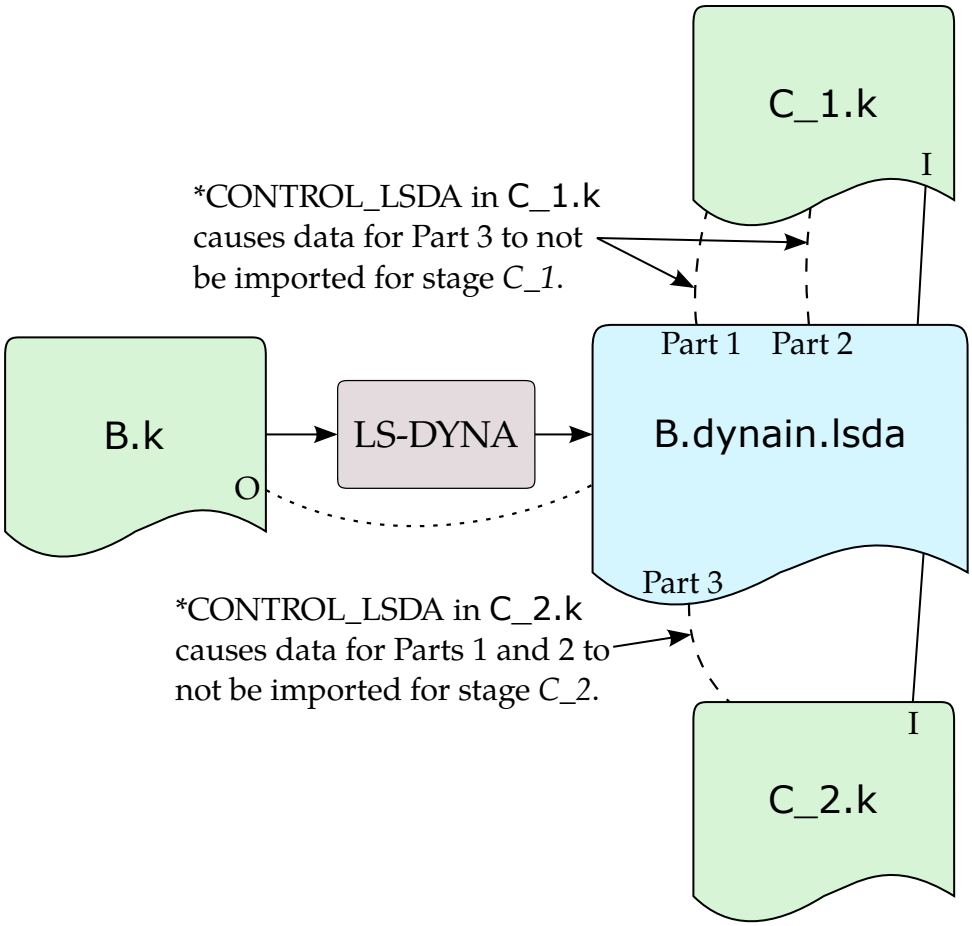

Figure 75-6. Illustration of how parts can be excluded in subsequent analyses. In this example. Stage *B* contains the full model. For stage *C_1* Part 3 is excluded from the analysis due to *CONTROL_LSDA while for stage *C_2* Parts 1 and 2 are excluded.

Changing Element and Material Types:

Just as the *PART card is needed in a continuation, the *SECTION and *MAT cards are needed as well. In almost every situation you would just copy them from the previous stage, assuming things can proceed without sacrificing execution speed or accuracy. In rare cases, you may want to change element formulation or maybe even material model.

The element formulation is preferably changed when switching from an explicit stage to an implicit stage, or vice versa. Fully integrated elements (such as type -16) enhance accuracy in implicit without significant speed penalty while reduced integration elements make explicit computationally efficient while maintaining sufficient accuracy. For example, stage *B* may have

```
*CONTROL_IMPLICIT_GENERAL
1,1.0
*SECTION_SHELL
```

```
1,-16
1.,1.,1.,1.
```

in *B.k*, and stage *C* continues the process with

```
*CONTROL_IMPLICIT_GENERAL
0
*SECTION_SHELL
1,2
1.,1.,1.,1.
```

in *C.k*. The stresses from elements of type -16 in stage *B* will be averaged in the plane for use in the type 2 shells of stage *C*, leading to loss of accuracy. Had it been the other way around, the single point stress in the Belytschko-Tsay shell would be scattered to the four in-plane integration points of the fully integrated shell, impacting the accuracy of the transition. We recommend maintaining section properties between stages as much as possible.

We do not recommend switching materials either in general. As mentioned in the discussion of NSHV = 999 under Outputting Data with *INTERFACE_SPRINGBACK_LS-DYNA, you *may* use a rigid material for a subset of parts during the first few stages, and then switch them to a deformable material when loaded accordingly. Switching back to a rigid material later will rigidize the parts in the deformed state, causing you to lose the internal variables. You must decide if losing this information for subsequent stages is acceptable.

Inertia Properties of Rigid Bodies:

Concerning the *PART card, the particular keyword *PART_INERTIA needs special attention. Again, recall that LS-DYNA will interpret the keyword as written which is a problem if the inertia properties are provided in global coordinates. Consider a rigid body that initially has a center of mass located at

$$\mathbf{x}_c = \begin{pmatrix} 5 \\ 0 \\ 0 \end{pmatrix}$$

and an inertia tensor along the global axes

$$\mathbf{I}_c = \begin{pmatrix} 3 & & \\ & 2 & \\ & & 1 \end{pmatrix}.$$

The part input for this body would be

```
*PART_INERTIA
Global inertia
1
…
$ XC YC ZC TM
5,0,0,1
$ IXX IXY IXZ IYY IYZ IZZ
3,0,0,2,0,0,1
$ No velocities
…
```

Now, if the body translates a distance 1 in the global x direction and rotates 90 degrees about this axis during the first stage, then the rigid body properties at the end of the simulation are

$$\mathbf{x}_c = \begin{pmatrix} 6 \\ 0 \\ 0 \end{pmatrix}$$

and

$$\mathbf{I}_c = \begin{pmatrix} 3 & & \\ & 1 & \\ & & 2 \end{pmatrix}.$$

Consequently, if the *PART_INERTIA card is copied between stages as we have recommended, the inertia properties will always take the values from the beginning of the entire process, resulting in apparent errors. The strategy for circumventing this is to define inertia properties with respect to local positions and a local coordinate system that follows the motion of the body. To do this, put a node at the center of mass and let NODEID refer to this node. Also, set IRCS = 1, and let CID refer to a coordinate system defined by nodes attached to the rigid body in question. The nodes needed for NODEID and CID can be defined by *CONSTRAINED_EXTRA_NODES. The keywords for the example above would be

```
*PART_INERTIA
Local inertia
1
…
$ XC YC ZC TM IRCS NODEID
,,,1,1,999
$ IXX IXY IXZ IYY IYZ IZZ
3,0,0,2,0,0,1
$ No velocities

$ XL YL ZL XLIP YLIP ZLIP CID
0,0,0,0,0,0,99
*NODE
999,5,0,0
9999,6,0,0
99999,5,1,0
*CONSTRAINED_EXTRA_NODES_NODE
1,999
1,9999
1,99999
*DEFINE_COORDINATE_NODES
99,999,9999,99999
```

The above is for the first stage. In all remaining stages the nodes will be contained in corresponding dynain.lsda files, while the other cards are copied between stages.

The same method must be used for *CONSTRAINED_NODAL_RIGID_BODY_INERTIA which has the same keyword data (except the coordinate system in input in field CID2) and *ELEMENT_INERTIA which can similarly be defined using a coordinate system based on nodal coordinates. In sum, care should be taken when simulating components with the INERTIA option.

Deformation Gradient and Stress Update:

All materials have their own set of history variables that are used for computing stresses. We loosely call a material *hyperelastic* if the deformation gradient or stretch tensor is an essential part of the history content and needed for computing the stress.

The deformation gradient can be computed in two different ways, governed by FMATRX on *CONTROL_SOLID. For FMATRX = 1, it is numerically incremented based on the strain and spin rate, while for FMATRX = 2 ,it is computed directly as a relation between the current and reference configurations. In the FMATRX = 1 case, nothing needs to be known about the reference configuration. It would be just like a *hypoelastic* material situation where everything is incrementally updated. For FMATRX = 2, the reference configuration needs to be the same throughout the entire process. Before version R16, this setting affected how to transition stresses between stages. Starting with R16, LS-DYNA handles the treatment of the deformation gradient automatically without requiring user intervention.

The discussion above pertains to solid elements. For shells and beams, the deformation gradient is, in general, numerically incremented, except for fabric materials (*MAT_FABRIC). For fabric materials, the discussion is slightly modified. For instance, no corresponding FMATRX flag exists. The deformation gradient is computed based on the reference geometry from the first stage. Also here, LS-DYNA handles the treatment of reference configurations between stages automatically in versions R16 and later.

Contact:

As discussed in the CFLAG = 1 section, contact histories are ported for a selection of contacts when CFLAG is 1 on *INTERFACE_SPRINGBACK_LSDYNA. The *CONTACT card itself, for the contact with a given ID, must be explicitly included in the keyword file in the continuing simulation. Recall that the dynain.lsda file contains the contact history for all supported contact types from the previous simulation. To read this information, the *CONTACT card ID must match the ID for the history data in the file must match. As discussed earlier, involved contact segments must also match in the sense of having the same user nodal IDs. Let's discuss some hypothetical situations beginning with the penalty contacts.

In a common scenario, the contact definition in subsequent runs is identical to the first run, using the same contact ID, same number of contact segments, and same corresponding user nodal IDs. We *are*, however, allowed to deviate from this restriction. For instance, if a part is added to the subsequent simulation, this part may be included in the contact. This additional part will render additional segments for which there will be a clean sheet of history variables. These segments may come into new contact situations while the other segments will have their history inherited. We can also consider removing a part from the model and consequently from the contact. Any remaining segments

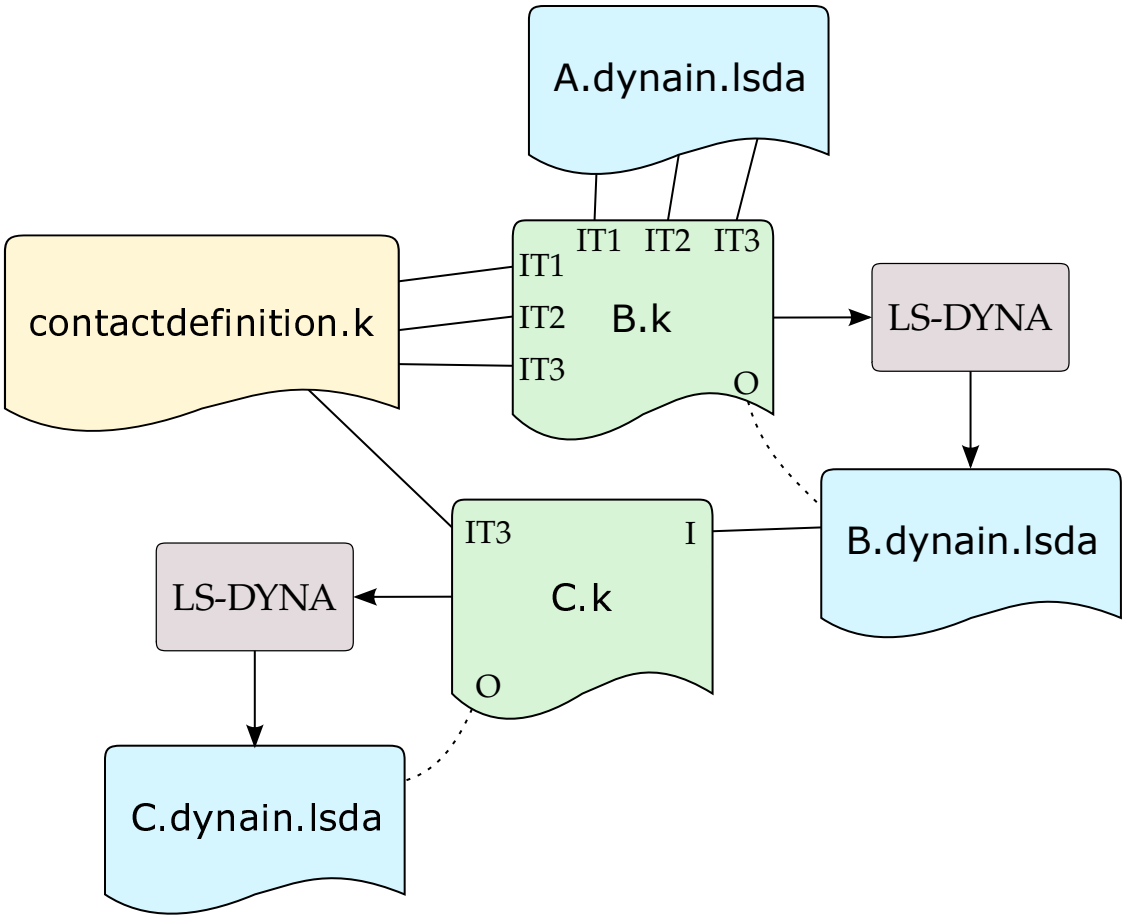

Figure 75-7. Example of including a component several times with transformations. In this example the component from stage *A* is included in Stage *B* three times. Because the same file is being included several times, you must use the keyword *INCLUDE_TRANSFORM. IT*i* indicates a *INCLUDE_TRANSFORM with TRANID *i*. The contact definition must also be included for each component with the same corresponding TRANIDs. For subsequent stage *C*, if we only want the third component, then we include the contact definition again with TRANID = 3. B.dynain.lsda does not need to be transformed since it already contains the transformed component.

that *were* in contact with the deleted part will be given no contact force and will be reset in terms of history variables. Finally, you can change the contact stiffness scale factor SFSA if needed.

For tied contacts, there aren't many exceptions to the rule of keeping the contact as is. Departing from the same contacts tends to result in situations for which the behavior is ambiguous. The mortar tied contacts offer some interesting switch options that may be worth discussing. For instance, you can execute a stage using the keyword *CONTACT_..._MORTAR_TIED_WELD. This contact essentially starts off as a regular, non-tied contact but will tie segments based on weld conditions as a result of the simulation. During the next stage, the keyword may be changed to *CONTACT_..._MORTAR_TIED,

or *CONTACT_..._TIEBREAK_MORTAR, assuming the welding part of the process is now done. The segments that were tied during the first stage are inherited to the next and the behavior will for the remaining process be just as for a regular tied or tiebreak contact.

Sometimes you need to include the same dynain.lsda file several times with different transformations using *INCLUDE_TRANSFORM. For instance, a small, preloaded component may be spatially distributed in a larger assembly. This feature is supported for the state of contact, including ID and nodal offsets, but it requires that *all* segments in contact are transformed as a unified rigid body. Note also that you need a *CONTACT card for each included file where the ID is transformed as the ID in the dynain.lsda file. If many such transformations are required, you can put the contact definition in a separate file and include that the same way as the dynain.lsda file. Figure 75-7 illustrates the case where a component is simulated in stage *A* and distributed in three instances for stage *B*. Note the treatment of the contact definition.

The keyword syntax in B.k would be

```
*INCLUDE_TRANSFORM
A.dynain.lsda
…
$TRANID
1
*INCLUDE_TRANSFORM
A.dynain.lsda
…
$TRANID
2
*INCLUDE_TRANSFORM
A.dynain.lsda
…
$TRANID
3
*INCLUDE_TRANSFORM
contactdefinition.k
…
$TRANID
1
*INCLUDE_TRANSFORM
contactdefinition.k
…
$TRANID
2
*INCLUDE_TRANSFORM
contactdefinition.k
…
$TRANID
3
```

The contact definition file is the same as when simulating stage *A*, just copied between stages. In stage *C*, assume we just want to keep the third component, meaning that we only need to keep the contact definition for that particular contact. The syntax in C.k would then be

```
*INCLUDE
B.dynain.lsda
*INCLUDE_TRANSFORM
```

```
contactdefinition.k
…
$TRANID
3
```

Note that the included dynain.lsda file needs no transformation since it already has the transformed quantities from the first transformation between stages *A* and *B*. This hypothetical example should provide insight into the interrelation between the resulting dynain.lsda files and main keyword ascii files. It should all make sense when recalling the basics of the keyword format.

Stabilization Forces:

Porting stabilization forces is done more or less automatically. the only concern is the compatibility between implicit and explicit. Following the discussion earlier (HFLAG = 1), we recommend invoking drilling restraints with DRCPSID on *CONTROL_SHELL, even for explicit analysis. Furthermore, using hourglass type 6 for solid elements and type 4 for shell elements is also a good practice. It would lead to continuity in the stabilization forces between stages. Departing from these recommendations may not be a terrible thing to do, but you should be aware of the possible consequences.

CASES

So far we have assumed that you execute runs sequentially and manually. Sometimes simulating all stages in a single run is convenient. For instance, suppose you need to submit an overnight run where you want all stages to be simulated, and you want to take advantage of multistage capability for reasons to be given. The keyword *CASE allows you to do setup a multistage simulation in a single run. Here we exemplify this usage by reconsidering the example at the beginning of this Appendix, the ROPS test. We emphasize that this problem may be set up in many equivalent ways. We will present here merely a suggested organization of files that should serve as a reasonable template. To shorten this section, we will skip the "gravity load" and the "load from behind" for this discussion.

The main process file would contain the definition of all cases (= stages), the card for writing the dynain.lsda file and perhaps other data that is common for all stages. It would look something like

```
*KEYWORD
$
$ Meta data for cases
$
*CASE
1,BoltPreload
*CASE
3,LoadSide
*CASE
4,LoadRoof
$
$ First case is defined all in ascii
```

```
$
*CASE_BEGIN_1
*INCLUDE
BoltPreload.k
*CASE_END_1
$
$ Remaining cases define bulk in ascii,
$ and includes the state from previous dynain.lsda file
$
*CASE_BEGIN_3
*INCLUDE
LoadSide.k
BoltPreload.dynain.lsda
*CASE_END_3
*CASE_BEGIN_4
*INCLUDE
LoadRoof.k
LoadSide.dynain.lsda
*CASE_END_4
$
$ Output "everything" to dynain.lsda
$
*INTERFACE_SPRINGBACK_LSDYNA
…
*INTERFACE_SPRINGBACK_EXCLUDE
BOUNDARY_SPC_NODE
$
$ Mounting of the cab, present in all stages
$
*BOUNDARY
…
*END
```

We then need to define each of the cases. We will start with the bolt preload. The bolt preload is the "easy" load case since it does not involve any inclusions of data from previous stages. Without further ado, BoltPreload.k would look something like

```
*KEYWORD
*TITLE
Bolt preload
$
$ Part data for the cab
$
*PART
…
*PART_INERTIA
…
*SECTION
…
*MAT
…
$
$ Nodes and elements for the cab
$
*NODE
…
*ELEMENT_...
…
$
$ Potential other cards required for the cab
$
*ELEMENT_MASS
…
*ELEMENT_INERTIA
…
*CONTACT_...
```

```
…
*CONSTRAINED_...
…
$
$ The specific load case follows
$
*INITIAL_STRESS_SECTION
…
*DATABASE_CROSS_SECTION
…
*END
```

We will not write out the details. You should imagine the general content of the cab model.

We then set up the load from the side assuming the bolt preload is already finished and the corresponding BoltPreload.dynain.lsda contains everything we now know it contains. The keyword file LoadSide.k needs to contain the complementary information as well as the impactor from the side and associated data. The input file would look like

```
*KEYWORD
*TITLE
Side load
$
$ Part data for the side impactor
$
*PART

1000,…
*SECTION
…
*MAT
…
$
$ Motion of the side impactor
$
*BOUNDARY_PRESCRIBED_MOTION_RIGID
1000,…
$
$ Geometry for the side impactor
$
*NODE
…
*ELEMENT
…
$
$ Contact between the side impactor and cab
$
*CONTACT_..._ID
1000,…
…
$
$ Data for the cab is exactly as in bolt preload,
$ except for nodes and elements which are in the
$ BoltPreload.dynain.lsda file
$
…
*END
```

We will highlight a few things. First, the nodes and elements for the cab are all in the dynain.lsda file, so they do not need to be copied over. Second, keywords like *ELEMENT_MASS/INERTIA and *CONSTRAINED_NODAL_RIGID_-

BODY/INTERPOLATION are to be seen as integrated components of the cab that are *not* written to the dynain.lsda file. These cards refer to quantities that are either updated between stages (nodes) or invariant to any deformation (masses). Therefore, these need to be copied.

Starting the third stage, we tend to get even more fancy because now we not only introduce the roof impactor but need to take out the side impactor. The LoadRoof.k would look like

```
*KEYWORD
*TITLE
Roof load
$
$ Ignore impactor from side in dynain.lsda file,
$ also taking out all of its part and contact data
$
*CONTROL_LSDA
1
1000
$
$ Part data for the roof impactor
$
*PART

2000,…
*SECTION
…
*MAT
…
$
$ Motion of the roof impactor
$
*BOUNDARY_PRESCRIBED_MOTION_RIGID
2000,…
$
$ Geometry for the roof impactor
$
*NODE
…
*ELEMENT
…
$
$ Contact between the roof impactor and cab
$
*CONTACT_..._ID
2000,…
…
$
$ Data for the cab is exactly as in side load,
$ except for nodes and elements which are now in the
$ LoadSide.dynain.lsda file
$
…
*END
```

In principle, this example defines one simulation run in three steps. You may wonder why this is better than defining all in one simulation. Assume you start this simulation overnight, and in the morning you can post-process the results for each of the three cases. What if the first two stages were successful, and the third stage failed? Well, then you can

simply make the modifications necessary in LoadRoof.k that you think takes care of whatever mistakes you made in the first place, and then redefine the main process file as

```
*KEYWORD
$
$ Meta data for cases, first two already successful
$
*CASE
4,LoadRoof
$
$ and we can start directly on the roof load
$
*CASE_BEGIN_4
*INCLUDE
LoadRoof.k
LoadSide.dynain.lsda
*CASE_END_4
*INTERFACE_SPRINGBACK_LSDYNA
…
*INTERFACE_SPRINGBACK_EXCLUDE
BOUNDARY_SPC_NODE
$
$ Mounting of the cab, present in all stages
$
*BOUNDARY
…
*END
```

You can submit this run without having to rerun the first two stages.

Multistage processes also allow you to cleanly organize a process compared to defining everything in a single input with birth/death times, deformable to rigid cards, implicit/explicit switches, complicated curves, etc. Admittedly, either way you do it manipulating text files in text editors is not the way to go. This appendix should be seen as an introductory guide for setting up and troubleshooting this kind of problem. An ultimate goal would be to do the setup in a high-level GUI where much of what has been presented in here as "recommendations" or "requirements" are dealt with automatically.