SIMULIA

3DEXPERIENCE®

ABAQUS 2025 FD04
EXECUTION GUIDE

DASSAULT SYSTEMES

Contents

Trademarks and Legal Notices....4
Abaqus Execution Guide....5
What's New....6
About Execution Procedures....9
Conventions....9
Environment Settings....10
Selecting TCP/UDP Port Numbers....10
Execution Procedures....11
Abaqus Execution....12
Obtaining Information....13
Abaqus/Standard and Abaqus/Explicit Execution....15
Abaqus Job Execution Control....23
Abaqus/CAE Execution....24
Abaqus/Viewer Execution....27
Making User-Defined Executables and Subroutines....29
Optimization Execution....31
Parametric Studies....33
Abaqus Documentation....34
Abaqus Co-Simulation....36
SIMULIA Co-Simulation Engine Director Execution....37
Abaqus/Standard, Abaqus/Explicit, and FMU Co-Simulation Execution....39
SIMULIA Co-Simulation Engine FMU Execution....46
Abaqus Utilities....48
Hardware System Verification Process....49
Python Execution....50
Licensing Utilities....53
Querying the Keyword/Problem Database....54
Fetching Sample Input Files....56
Reducing Analysis Database File Sizes for Restart....58
Output Database Upgrade Utility....60
SIM Database Utilities....61
Translating ODB Output Database Files to SIM Format....63
Generating Output Database Reports....65
Joining Output Database (.odb) Files from Restarted Analyses....71
Combining Output from Substructures....73
Recovering Results from a Substructure....76
Combining Data from Multiple Output Databases....80
Network Output Database File Connector....83
Mapping Thermal and Magnetic Loads....85
Element Matrix Assembly Utility....87
Encrypting and Decrypting Abaqus Input Data....88
ASCII Translation of Results (.fil) Files....92
Joining Results (.fil) Files....93

Translating Other Solver Input to Abaqus Input....94
Translating Nastran Data to Abaqus Files....95
Translating ANSYS Input Files to Partial Abaqus Input Files....107
Translating PAM-CRASH Input Files to Partial Abaqus Input Files....110
Translating RADIOSS Input Files to Partial Abaqus Input Files....115
Translating LS-DYNA Data Files to Abaqus Input Files....120
Exchanging Abaqus Data with ZAERO....129
Translating a Simpack Flexible Body to Abaqus Matrix Data....133
Translating Moldflow Data to Abaqus Input Files....134
Translating Abaqus Output to Another Format....141
Translating Abaqus Files to Nastran Bulk Data Files....142
Translating Abaqus Output Database Files to Nastran Output2 Results Files....146
Translating an Abaqus Substructure to an MSC.Adams Flexible Body....148
Translating an Abaqus Substructure to a Simpack Flexible Body....151
Translating an Abaqus Substructure to an EXCITE Flexible Body....153
Environment File Settings....155
Environment Settings Hierarchy....155
Syntax....156
Troubleshooting....156
Command Line Default Parameters....156
System Resource Parameters....158
System Customization Parameters....158
Export of Environment Variables during a Parallel Execution....160
Co-Simulation Parameters....160
Environment File Examples....161
Linux Environment File:....161
Windows Environment File:....161
Managing Memory and Disk Resources....162
Understanding Resource Use....162
Abaqus Data....162
Requirements and Considerations....162
Resource Management Parameters....163
Memory Management Parameters....163
Disk Management Parameters....163
Input File Processing and data check....164
Guidelines for Memory Settings....164
Abaqus/Standard Analysis....164
Guidelines for Memory Settings....165
Parallel Execution....166
About Parallel Execution....167
Parallel Processing Support for Abaqus Features....167
Parallel Execution on Shared Memory Computers....167
Parallel Execution on Computer Clusters....167
Parallel Execution Using GPGPU Hardware....168
Use with User Subroutines....168
Parallel Execution in Abaqus/Standard....169
Invoking Parallel Processing....169

Thread-Based Parallelization....169
MPI-Based Parallelization....170
Hybrid Parallelization of MPI and Threads....171
GPGPU Acceleration....172
Parallel Execution of Steady-State Dynamic Analyses....172
Consistency of Results....175
Parallel Execution in Abaqus/Explicit....176
Invoking Parallel Processing....176
Domain-Level Parallelization....177
Loop-Level Parallelization....184
Measuring Parallel Performance....184
Output....184
File Extension Definitions....185
File Extensions....185
Fortran Unit Numbers....190
Fortran Unit Numbers....190

Trademarks and Legal Notices

Trademarks

Abaqus, **3D**EXPERIENCE®, the 3DS logo, the Compass icon, IFWE, 3DEXCITE, 3DVIA, BIOVIA, CATIA, CENTRIC PLM, DELMIA, ENOVIA, GEOVIA, MEDIDATA, NETVIBES, OUTSCALE, SIMULIA and SOLIDWORKS are commercial trademarks or registered trademarks of Dassault Systèmes, a European company (Societas Europaea) incorporated under French law, and registered with the Versailles trade and companies registry under number 322 306 440, or its subsidiaries in the United States and/or other countries. All other trademarks are owned by their respective owners. Use of any Dassault Systèmes or its subsidiaries trademarks is subject to their express written approval.

Legal Notices

Abaqus and this documentation may be used or reproduced only in accordance with the terms of the software license agreement signed by the customer, or, absent such agreement, the then current software license agreement to which the documentation relates.

This documentation and the software described in this documentation are subject to change without prior notice.

Dassault Systèmes or its Affiliates shall not be responsible for the consequences of any errors or omissions that may appear in this documentation.

© Dassault Systèmes Americas Corp., 2025.

For a full list of the third-party software contained in this release, please go to the Legal Notices in the Abaqus 2025 HTML documentation, which can be obtained from a documentation installation, or in the SIMULIA Established Products 2025 Program Directory, which is available on *www.3ds.com*.

Abaqus Execution Guide

The Abaqus Execution Guide discusses the execution procedures for Abaqus/Standard, Abaqus/Explicit, Abaqus/CAE, and several utilities that are provided with Abaqus.

This guide is a part of the Abaqus® documentation collection, which describes all the capabilities of the Abaqus finite element analysis technology used in SIMULIA® applications.

Abaqus is executed by using the Abaqus execution procedure. In the following discussion the command to run the execution procedure is assumed to be **abaqus**. However, you can customize the execution procedure to run Abaqus using any alias you choose. (See the *Abaqus Configuration Guide* for details.)

What's New

This page describes recent changes in Abaqus Execution.

2025 FD04

LS-DYNA to Abaqus Translator Enhancements

You can now translate nonstructural mass and additional set keywords using the LS-DYNA translator. The translation of piecewise linear plastic material now includes rate dependency. Imposition of initial velocity to a rigid body now includes its reference node.

Benefits: The enhancements improve the usability of the translator.

Translation of the following LS-Dyna keywords is now supported:

`*ELEMENT_MASS_PART`

`*ELEMENT_MASS_PART_SET`

`*SET_PART_ADD`

`*SET_NODE_ADD`

In the translation of `*MAT_PIECEWISE_LINEAR_PLASTICITY`, rate dependency data is translated based on the C, P and LCSS parameters associated with the keyword. The translation of `*PARAMETER` and `*PARAMETER_EXPRESSION` is revised to include use of the parameter names in the Abaqus data and is now supported in the translation of `*INITIAL_VELOCITY`. In addition, the imposition of initial velocity to a rigid body now includes the reference node generated in the translation.

For more information, see *Translating LS-DYNA Data Files to Abaqus Input Files*.

2025 FD02

Improved Substructure Results Recovery for Abaqus-Simpack Flexible Body Dynamics

You can now recover results obtained from Simpack dynamic time domain and frequency domain analyses.

Benefits: The enhancements improve the Abaqus-Simpack workflow for flexible body dynamic analysis. The **abaqus substructurerecover** utility now supports recovery of results obtained from Simpack dynamic time domain and frequency domain analyses.

For more information, see *Environment File Settings* and *Recovering Results from a Substructure*.

2025 FD01

Environment File Setting Customizes Warnings Level

You can now specify whether common warnings are reported by the Abaqus analysis input file processor.

Benefits: Reducing the number of warnings makes evaluation of the preprocessed model more convenient.

The Abaqus analysis input file processor now suppresses common warnings for unsupported output requests by default. However, you can use the new environment variable `output_warnings_level` to specify if common warnings are reported.

Many warnings issued by the Abaqus analysis input file processor for invalid output requests, such as those issued for output requests that are not supported for specific element or procedure types, are informational only and warrant no action by the user. For large models the number of these warnings can be very high and can make it difficult to evaluate of the preprocessed model.

For more information, see *Environment File Settings*.

PDF Guides Available

You can download PDF versions of the Abaqus guides from each guide overview.

Benefits: You can easily access the PDF versions of the guides.
In earlier releases, PDF versions of the guides were updated only for the GA (General Availability) release and could be downloaded from the Dassault Systèmes Knowledge Base. As of 2025 FD01, the PDF guides will be updated with each release of the HTML versions of the guides.

For more information, see *Abaqus Execution Guide*.

2024 FD04

Optimization and Parallel Execution

For parallel execution using GPGPU acceleration, you now specify the number of GPUs as the total number of GPUs for the analysis rather than specifying the number of GPUs per MPI process.

Benefits: Specifying GPGPU acceleration using the total number of GPUs is more consistent with how you specify the number of CPUs allocated to an analysis and permits greater flexibility for job execution in heterogeneous compute environments.

For more information, see *Abaqus/Standard and Abaqus/Explicit Execution*, *Optimization Execution*, *Environment File Settings*, *Parallel Execution in Abaqus/Standard*, and *Parallel Execution in Abaqus/Explicit*.

2024 FD03

Direct Sparse Solver Support for Steps with Unsymmetric Matrix Storage

For shared memory parallel (SMP) execution of steady-state dynamic direct procedures with unsymmetric matrix storage, Abaqus/Standard now uses the pure thread-based complex solver by default.

Benefits: The pure thread-based complex solver provides a solution that is typically at least twice as fast as the previous default option, the hybrid direct sparse solver.

For more information, see *Parallel Execution in Abaqus/Standard*.

2024 FD02

Execution Procedures User Assistance Reorganized

Content in the Execution Procedures section of the Abaqus Execution Guide is now divided into several subsections.

Benefits: The new organization makes it easier to locate user assistance for the **abaqus** commands you want to run.
Topics are now divided into the following subsections:

- *Abaqus Execution*
- *Abaqus Co-Simulation*
- *Abaqus Utilities*
- *Translating Other Solver Input to Abaqus Input*
- *Translating Abaqus Output to Another Format*

For more information, see *Execution Procedures*.

2024 FD01

Enhanced Translation of LS-DYNA Keyword Files to Abaqus

You can now translate LS-DYNA keyword files that use the i10 format option in reading integer data.

Benefits: The ability to read these data improves the usability of the **abaqus fromdyna** translator.
You can specify the format for individual keywords by appending the percent sign at the end of the keyword, as shown in the following examples:

- `*ELEMENT_BEAM %`
- `*ELEMENT_SHELL %`
- `*ELEMENT_SOLID %`
- `*NODE %`

In addition, you can specify the format globally for all applicable keywords in the LS-DYNA keyword file, by adding an option to the execution command of the translator or by declaring the following keyword option:

```
*KEYWORD i10=y
```

For more information, see *Translating LS-DYNA Data Files to Abaqus Input Files*.

About Execution Procedures

Overview

Abaqus is executed by using the Abaqus execution procedure. In the following discussion the command to run the execution procedure is assumed to be **abaqus**. However, you can customize the execution procedure to run Abaqus using any alias you choose. (See the *Abaqus Configuration Guide* for details.)

The **abaqus** command is described in *Execution Procedures*. The following sections contain further information about running Abaqus jobs:

- *Environment File Settings*
- *Managing Memory and Disk Resources*
- *Parallel Execution*
- *File Extension Definitions*
- *Fortran Unit Numbers*

References:

- *Execution Procedures*
- *Parallel Execution*

Conventions

The following conventions are used in these sections:

- Each discussion includes a “Command summary” section that provides the syntax for the command in the left column and the syntax for its options in the right column. The full command must appear first, followed by the options. In some cases the command has multiple words, such as **abaqus fetch**; you must enter all words of the command before issuing any option statements.
- Options are presented in **boldface**. They can appear in any order and can be abbreviated.
- Default options are underlined (__).
- Items enclosed in square brackets ([ ]) are optional.
- Items appearing in a list separated by bars (|) are mutually exclusive.
- One value must be selected from a list of values enclosed by curly brackets ({ }).
- You must supply values in *italics*.
- Blanks are used as separators between options and must not precede nor follow an equal sign.
- An alternate syntax of **-option***value* can be used instead of the **option=***value* format.

The **abaqus** procedure will prompt for any information required that is not provided on the command line. If **abaqus** is typed with no options, prompts are issued for all options.

Environment Settings

The Abaqus execution procedure uses environment parameters to customize the execution of a job. These settings can be changed using one of the Abaqus environment files: `custom_v6.env` or `abaqus_v6.env`. See *Using the Abaqus environment files* for more details.

If the same job parameter is defined in more than one environment file or is defined more than once within the same environment file, the last definition encountered will be used. Some exceptions to this rule are noted in *Environment File Settings*. These environment files can be used to customize the behavior of Abaqus, including modification of the default options. See *Environment File Settings* for further information on the environment files.

Selecting TCP/UDP Port Numbers

Several of the execution procedure command line options, such as **port** and **listenerport**, require that you specify a port number. TCP/UDP port numbers can range from 0 to 65535.

Port numbers 0 to 1023 are well-known ports used by system processes (such as FTP, SSH, SMTP, etc.) and should never be used. Port numbers 1024 to 49151 are registered ports with the Internet Assigned Number Authority (IANA) by software vendors. These ports can be used, but you should be careful that you are not conflicting with any software installed on your system that may be using this port. Port numbers 49152 to 65535 are unreserved and can be used freely, as long as no other application uses them.

Ports may be blocked by a firewall. Contact your system administrator to ensure that the ports that you want to specify are not blocked.

You can use the `netstat` command to obtain information on TCP/UDP network connections.

Execution Procedures

This section describes using the **abaqus** command to run Abaqus.

In this section:

- *Abaqus Execution*
- *Abaqus Co-Simulation*
- *Abaqus Utilities*
- *Translating Other Solver Input to Abaqus Input*
- *Translating Abaqus Output to Another Format*

Abaqus Execution

This section describes using the **abaqus** command to run Abaqus and the **abaqus doc** command to run Abaqus documentation.

In this section:

- *Obtaining Information*
- *Abaqus/Standard and Abaqus/Explicit Execution*
- *Abaqus Job Execution Control*
- *Abaqus/CAE Execution*
- *Abaqus/Viewer Execution*
- *Making User-Defined Executables and Subroutines*
- *Optimization Execution*
- *Parametric Studies*
- *Abaqus Documentation*

Obtaining Information

Products: Abaqus/Standard Abaqus/Explicit

References:

- *About Execution Procedures*

Overview

The Abaqus execution procedure can be used to obtain help regarding command syntax or information about the installation and computing environment.

Command Summary

abaqus {**help** | **information =** {`environment` | `local` | `memory` | `release` | `support` | `system` | `all`} **[job =** *job-name*] | **whereami**}

Command Line Options

help This option prints a summary of the **abaqus** command syntax.

information This option writes information about the installation and the environment that is in effect to the screen. The following information is output for all information requests: the current release, the directory in which Abaqus is located, and the directory in which the information files are located.

- If **information**=`environment`, the current settings of the environment file options are displayed.
- If **information**=`local`, the local installation notes are output.
- If **information**=`memory`, some suggestions for setting memory parameters for analysis jobs are output.
- If **information**=`release`, information is provided about where to locate the current release notes.
- If **information**=`support`, information on diagnosing hardware-related issues is provided. Please send this information to systems support when requesting assistance.
- If **information**=`system`, information is provided about system software and hardware resources (operating system level, compiler levels, processor type, graphics board, memory, etc).
- If **information**=`all`, information on all of the above information topics is output.

job If a *job-name* is specified, the information text is written to the file *job-name*`.log`.

whereami This option prints the location of the Abaqus release directory.

Examples

Use the following command to display the local installation notes:

```
abaqus information=local
```

The following command will write the local installation notes to the file `support.log`:

```
abaqus information=local job=support
```

Abaqus/Standard and Abaqus/Explicit Execution

Products: Abaqus/Standard Abaqus/Explicit

References:

- *About Execution Procedures*

Overview

Abaqus/Standard and Abaqus/Explicit are executed by running the Abaqus execution procedure. Several parameters can be set either on the command line or in the environment file (see *Environment File Settings*). Alternatively, you can use the convenient Abaqus/CAE user interface to submit an Abaqus analysis from an input file and set the analysis parameters; see *Understanding analysis jobs*.

Abaqus enforces a character limit on file names. For any command line reference to a file, the total length of the file name, including the path description, cannot exceed 256 characters.

Command Summary

abaqus

job = *job-name*
[analysis] | [datacheck] | [parametercheck] | [continue] | [convert = {`select` | `odb` | `state` | `all`}**] | [recover] | [syntaxcheck] | [information =** {`environment` | `local` | `memory` | `release` | `support` | `system` | `all`}**]**
[input = *input-file*]
[user = {*source-file* | *object-file*}**]**
[uniquelibs]
[oldjob = *oldjob-name*]
[fil = {`append` | `new`}**]**
[globalmodel = {*results file-name* | *ODB output database file-name* | *SIM database file-name* }**]**
[cpus = *total number-of-cpus*]
[parallel = {`domain` | `loop`}**]**
[domains = *number-of-domains*]
[dynamic_load_balancing = {`on` | `off`}**]**
[threads_per_mpi_process = *number of threads per mpi process*]
[standard_parallel = {`all` | `solver`}**]**
[gpus = *total number-of-gpgpus*]
[memory = *memory-size*]
[interactive] | [background] | [queue [= *queue-name* **][after =** *time*]]
[double =** {`explicit` | `both` | `off` | `constraint`}**]**
[scratch = *scratch-dir*]
[output_precision = {`single` | `full`}**]**
[resultsformat = {`odb` | `sim` | `both`}**]**
[port = *co-simulation port-number*]
[host = *co-simulation hostname*]
[csedirector = *Co-Simulation Engine director host:port-number*]
[timeout = *co-simulation timeout value in seconds*]
[unconnected_regions = {`yes` | `no`}**]**
[noFlexBody]
[license_type = {`token` | `credit`}**]**

[ssd_split = *number-of-frequency-partitions***]**
[ssd_partition = *frequency-partition-number***]**

Command Line Options

Required Option

job The value of this option specifies the name of all files generated during the run and the name of files that are read in the **continue**, **convert**, and **recover** phases.

If this option is omitted from the command line, you will be prompted for its value (except when only the informational options described in *Obtaining Information* are used). If the **input** option is not supplied, the procedure will look for an input file called *job-name*`.inp` in the current directory.

Mutually Exclusive Options That Determine Which Phases of an Analysis Are Performed

All options are order independent. If none of these options is present, the **analysis** option is assumed. The **convert** option is an exception to the mutual exclusion rule: **convert** can appear with any option except **datacheck**, **parametercheck**, **syntaxcheck**, and **information**.

analysis This option indicates that a complete Abaqus analysis (or a restart of an Abaqus analysis) is to be performed.

datacheck This option indicates that the run is for data checking only. No analysis will be performed. If this option is used, all files necessary to continue the analysis are saved.

parametercheck This option indicates that the run is for input parameter checking only (parameter definitions must have been used; see *Parametric Modeling*). No analysis or data checking will be performed.

continue This option indicates that the run is to begin at the point at which a previous data check run ended.

convert The value of this parameter indicates which files will be postprocessed.

Results can be converted either immediately following an analysis run, as a separate run subsequent to an analysis run, or while an analysis is running as follows:

1. To run an analysis including a subsequent conversion of the results, use the **convert** option in conjunction with the **job** and **analysis** options.
2. To convert the results of a previously run analysis, use the **convert** option in conjunction with the **job** option.
3. To convert results from a job that is currently running, use the **convert** option in conjunction with the **oldjob** option (to name the running job) and the **job** option (to supply a new name for the files generated by the **convert** option).

If **convert**=`select`, the Abaqus/Explicit selected results file (*job-name*`.sel`) will be converted into a standard Abaqus results file (*job-name*`.fil`). If the analysis is run in parallel with **parallel**=`domain`, the separate selected results files (*job-name*`.sel.`*n*) will be converted into a single selected results file (*job-name*`.sel`) prior to being converted into a standard Abaqus results file.

If **convert**=`odb`, the output database (*job-name*`.odb`) will be converted using the postprocessing calculator (see *The Postprocessing Calculator*). This conversion is necessary only if the types of output listed in *The Postprocessing Calculator* are requested.

If **convert**=`state`, the separate Abaqus/Explicit state files (*job-name*`.abq.`*n*) will be converted into a single Abaqus/Explicit state file (*job-name*`.abq`) if the analysis is run in parallel with **parallel**=`domain`.

If **convert**=`all`, all of the applicable convert options will be executed.

recover This option applies only to Abaqus/Explicit. It indicates that an analysis is to be restarted at the last available step and increment in the state file. This capability is available to restart after a catastrophic failure, such as exceeding a CPU limit or a disk quota (see *Restarting an Analysis*). If the original analysis was run in parallel with **parallel**=`domain`, it must be restarted with **parallel**=`domain` and the same number of processors.

syntaxcheck This option indicates that the run is for checking the syntax of the input file only. This option does not use any license tokens. No analysis will be performed, and the **continue** option cannot be used to continue with an analysis. Only the data (`.dat`) and output database (`.odb`) files are generated for viewing. In an Abaqus/Explicit analysis, the model data in the output database may not be complete.

information This option writes information about the installation and the environment that is in effect to the screen or to the file *job-name*`.log`. For output information for each value of this option, see *Obtaining Information*. If the **information** option is used in conjunction with the **analysis** option, the job must be run in the background to write the information text to the log file.

Additional Options Available for the Analysis Module

input This option is used to specify the input file name, which may be given with or without the `.inp` extension (if the extension is not supplied, Abaqus will append it automatically). If this option is not supplied, the procedure will look for an input file called *job-name*`.inp` in the current directory. If *job-name*`.inp` cannot be found, the procedure will prompt for the input file name.

user This option specifies the name of a source or object file that contains any user subroutines to be used in the analysis. The name of the user routine may contain a path name and may be given with or without a file extension. Abaqus/Standard and Abaqus/Explicit accept user subroutines written in C, C++, or Fortran.

If an extension is given, the program will take the appropriate action based on the file type. If the file name has no extension, the program will search for a C, C++, or Fortran source file. If the source file does not exist, an object file will be searched for instead. The execution procedure creates a shared library using the user subroutine file that is used by the analysis during execution.

If the same user subroutine will be needed often, consider setting the **usub_lib_dir** environment file parameter and using the **abaqus make** execution procedure to create a shared library containing the user subroutine. This will avoid the need to recompile and/or relink the user subroutine each time it is needed. The **user** option is not required if the user subroutine called by the analysis is contained in the user library. User libraries contained in the directory given by the **usub_lib_dir** environment file parameter will not be used if the **user** option is specified.

The **user** option cannot be used to specify an object file when the **double** option is used to run an Abaqus/Explicit analysis because Abaqus/Explicit double precision runs need both single precision and double precision objects. In this case you must set the **usub_lib_dir** environment file parameter and place the single and double precision object files in the specified directory; alternatively, you can supply the user subroutine source.

uniquelibs This option indicates that input file execution is to use libraries created with the **uniquelibs** option in the **abaqus make** execution procedure.

oldjob This option identifies the files from a previous run from which an import, restart, or postprocessing run (Abaqus/Standard only; see *Recovering Additional Results Output from Restart Data in Abaqus/Standard*) is to start or from which to import results. You specify the *oldjob-name* (without a file extension). For an import or restart analysis, you can include a full or a relative path. The *oldjob-name* must be different from the current *job-name*.

This option is required when a restart, postprocessing, or symmetric model generation analysis reads data from the restart or the results file.

For import analysis, this option can be used when importing from a single previous analysis. This option should not be used when importing from multiple previous analyses.

fil This option specifies whether the data from the old results file specified in a restart run are included at the beginning of the new results file (default). If **fil=**`new` is used, the new results file will contain only the data from the point in the analysis where the restart occurred. This feature is used for Abaqus/Standard runs to join the output from restarted analyses into a single, continuous results file. Non-restart jobs cannot use this feature to append results file output to an old results file; the **abaqus append** execution procedure must be used for this purpose. Setting **fil=**`new` is not allowed for Abaqus/Explicit runs.

globalmodel This option specifies the name of the global model's results file, ODB output database file, or SIM database file from which the results are to be interpolated to drive a submodel analysis. This option is required whenever a submodel analysis or submodel boundary condition reads data from the global model's results.

The file extension is optional. If you omit the file extension, Abaqus uses the results file. If the results file does not exist, Abaqus uses the SIM output database file. If both the results file and the SIM output database file do not exist, Abaqus uses the ODB database file.

cpus This option specifies the total number of processors to use during an analysis run if parallel processing is available. The default value for this parameter is 1 and can be changed in the environment file (see *Environment File Settings*).

parallel This option specifies the method to use for thread-based parallel processing in Abaqus/Explicit. The possible values are `domain` and `loop`. The default value is `domain`, which can be changed in the environment file (see *Environment File Settings*).

If **parallel=**`domain`, the domain-level method is used to break the model into geometric domains. If **parallel=**`loop`, the loop-level method is used to parallelize low-level loops. For more information on these methods, see *Parallel Execution in Abaqus/Explicit*.

domains This option specifies the number of parallel domains in Abaqus/Explicit. If the value is greater than 1, the domain decomposition will be performed regardless of the values of the **parallel** and **cpus** options. However, if **parallel=**`domain`, the value of **cpus** must be evenly divisible into the value of **domains**. The default value is set equal to the number of processors used during the analysis run if **parallel=**`domain` and 1 if **parallel=**`loop`. The default value can be changed in the environment file (see *Environment File Settings*). A restart analysis uses the

same number of parallel domains as the original analysis, and the value specified with this option will be ignored.

dynamic_load_balancing For domain-parallel execution in Abaqus/Explicit (**parallel**=`domain`) where the number of domains is larger than the number of cpus, this option activates/deactivates the dynamic load balancing scheme. The default value is `on`. Abaqus/Explicit will attempt to improve computational efficiency by periodically reassigning domains to processors in a way that minimizes load imbalance (see *Parallel Execution in Abaqus/Explicit*).

threads_per_mpi_process This parameter is used to select the parallel execution mode from the following options: pure MPI mode, pure thread mode, or a hybrid mode. The thread mode of execution requires that all cores be limited to a single compute node, whereas the cores may span multiple compute nodes for the other two modes.

A value of *1* results in pure MPI mode. The pure MPI mode is active by default for Abaqus/Explicit (see *Parallel Execution in Abaqus/Explicit*).

A value equal to the **cpus** parameter results in pure thread mode.

A value in between 1 and the **cpus** parameter, divisible into the **cpus** parameter value, results in hybrid mode. The hybrid mode is active by default for Abaqus/Standard with the **threads_per_mpi_process** value selected automatically (see *Parallel Execution in Abaqus/Standard*).

standard_parallel This option specifies the parallel execution mode in Abaqus/Standard. The possible values are `all` and `solver`. If **standard_parallel**=`all`, both the element operations and the solver will run in parallel. If **standard_parallel**=`solver`, only the solver will run in parallel. The default value is **standard_parallel**=`all` on platforms where MPI-based parallelization is supported.

The parallel execution mode can also be set in the environment file (see *Environment File Settings*).

gpus This option specifies acceleration of the Abaqus/Standard direct solver. This option is meaningful only on computers equipped with appropriate GPGPU hardware. By default, GPGPU solver acceleration is not activated. The value of this parameter is the total number of GPGPUs to be used in an Abaqus/Standard analysis. In an MPI-based analysis, this parameter is the number of GPGPUs across all hosts.

GPGPU-based solver acceleration can also be set in the environment file (see *Environment File Settings*).

memory Maximum amount of memory or maximum percentage of the physical memory that can be allocated during the input file preprocessing and during the Abaqus/Standard analysis phase (see *Managing Memory and Disk Resources*). The default values can be changed in the environment file (see *Environment File Settings*).

interactive This option will cause the job to run interactively. For Abaqus/Standard the log file will be output to the screen; for Abaqus/Explicit the status file and the log file will be output to the screen. The default **run_mode** can be set in the environment file (see *Environment File Settings*).

background This option will submit the job to run in the background, which is the default. Log file output will be saved in the file *job-name*`.log` in the current directory. The default method for submitting the job can be set in the environment file by using the **run_mode** parameter (see *Environment File Settings*).

queue This option will submit the job to a batch queue. If the option appears with no value, the job will be submitted to the system default queue. Quoted strings are allowed. The available queues are site specific. Contact your site administrator to find out more about local queuing capabilities. Use **information**=`local` to see what local queuing capabilities have been installed. The default method for submitting the job can be set in the environment file by using the **run_mode** parameter (see *Environment File Settings*).

after This option is used in conjunction with the **queue** option to specify the time at which the job will start in the selected batch queue. This capability is supported for each individual site through the Abaqus environment file. (See the *Abaqus Configuration Guide* for details.)

double This option is used to specify that the double precision executable is to be used for Abaqus/Explicit. The possible values are `both`, `constraint`, `explicit`, and `off`. This capability is also supported through the Abaqus environment file with the environment variable **double_precision** (see *Environment File Settings*).

If **double**=`both`, both the Abaqus/Explicit packager and analysis will run in double precision.

If **double**=`constraint`, the constraint packaging and constraint solver in Abaqus/Explicit will run in double precision, while the Abaqus/Explicit packager and Abaqus/Explicit analysis continue to run in single precision.

If **double**=`explicit`, the Abaqus/Explicit analysis will run in double precision, while the packager will still run in single precision. The default value is `explicit`.

If **double**=`off`, the environment file setting is overridden if necessary to invoke both the Abaqus/Explicit packager and Abaqus/Explicit analysis in single precision. For a discussion of when to use the double precision executable, see *Defining an Analysis*.

scratch This option is used to specify the name of the directory used for scratch files. On Linux platforms the default value is the value of the `$TMPDIR` environment variable or `/tmp` if `$TMPDIR` is not defined. On Windows platforms the default value is the value of the `%TEMP%` environment variable or `\TEMP` if this variable is not defined. During the analysis a subdirectory will be created under this directory to hold the analysis scratch files. The default value for this parameter can be set in the environment file (see *Environment File Settings*).

output_precision This option specifies the precision of the field output written to the output database file (*job-name*`.odb`). Using **output_precision**=`full` results in double precision nodal and element field output for Abaqus/Standard analyses. To obtain double precision nodal field output for Abaqus/Explicit analyses, use the **double** option in addition to using **output_precision**=`full`. The element field output in double precision is not applicable for Abaqus/Explicit. History output is available only in single precision. This option cannot be used with the **recover** option.

resultsformat This option specifies the output format of the results. If **resultsformat**=`odb`, the output is written in ODB format only. If **resultsformat**=`sim`, the output is written in SIM format only. If **resultsformat**=`both`, the output is written in both ODB and SIM formats. The default value is `odb`. For more information, see *The Output Database*.

port This option is used to specify the TCP/UDP port number for co-simulation between solvers using the direct coupling interface, which includes co-simulation between Abaqus and certain third-party analysis programs. Set **port** equal to the port number

used for the connection. The default value is `48000`. The default port number that Abaqus uses to initiate communication can be set with the **cosimulation_port** parameter in the environment file (see *Environment File Settings*). This option is used in conjunction with the **host** option. For more information, see *Selecting TCP/UDP Port Numbers*.

host This option is used to specify the host name for co-simulation between solvers using the direct coupling interface, which includes co-simulation between Abaqus and certain third-party analysis programs. This option specifies the name of the machine that is hosting the connection. Refer to the third-party program documentation to determine if the **host** option is required. This option is used in conjunction with the **port** option.

csedirector This option is used to specify the connection (e.g., host:port) for the SIMULIA Co-Simulation Engine director process when performing a co-simulation using the SIMULIA Co-Simulation Engine. The **csedirector** entry identifies the host name and the TCP/UDP port number for the listening port of the SIMULIA Co-Simulation Engine director process.

timeout This option is used to specify a timeout value in seconds for establishing the co-simulation connection using the direct coupling interface or the SIMULIA Co-Simulation Engine. Abaqus terminates if it does not receive any communication from the coupled analysis program during the time specified. The default value is 3600 seconds. The default timeout value that Abaqus uses can be set with the **cosimulation_timeout** parameter in the environment file when using the direct coupling interface (see *Environment File Settings*).

noFlexBody This option is used to indicate that no translation to generate a flexible body should be done. By default, the **abaqus** execution procedure executes the **tosimpack** or **toexcite** translators with default arguments on completion of an Abaqus/Standard flexible body generation analysis set to generate a flexible body in the Simpack or AVL EXCITE™ format, respectively.

license_type This option is used to control the type of license Abaqus uses for the SimUnit license model. The possible values are `token` and `credit`. The default value is `token`. This capability is also supported through the Abaqus environment file with the environment variable **license_type**. For more information, see *License management parameters*.

ssd_split This option specifies a value for splitting the frequencies into partitions to execute a steady-state dynamic analysis by running multiple Abaqus jobs.

ssd_partition This option specifies a particular frequency partition number in conjunction with the frequency split specified by the **ssd_split** option.

Additional Option Available for the Datacheck Module

unconnected_regions This option is used to request that Abaqus/Standard create element and node sets for unconnected regions in the analysis output database. Set **unconnected_regions**=`yes` to create element and node sets that are named MESH COMPONENT *N*, where *N* is the component number.

Examples

The following examples illustrate the different functions and capabilities of the **abaqus** execution procedure.

Running Analyses in Abaqus/Standard

Use the following command to run a heat transfer analysis called “c8” in the background:

```
abaqus analysis job=c8 background
```

The following command runs the job c8 in the background and output the current environment settings to the log file:

```
abaqus analysis job=c8 information=environment background
```

The follow-up analysis to the heat transfer analysis c8 is “c10,” which is a static analysis that uses temperature data from c8 as input. The temperature data are read in from the c8 results file as predefined fields. The execution procedure scans the Abaqus/Standard input file for file dependencies of this sort. In this example the procedure looks for the c8 results file in the current directory with the extension `.fil`. The results file identifier can include a path name (see *Input Syntax Rules*), and the execution procedure then looks in the directory specified. In either case an error message is issued if the file does not exist. The following command is used to run the job c10 in the “long” queue:

```
abaqus analysis job=c10 queue=long
```

This job is next restarted as “c11,” using the final results from c10 as the starting point for a creep analysis. The following command is used to run this job in the default queue:

```
abaqus analysis job=c11 oldjob=c10 queue=
```

The following command is used to run an Abaqus/Standard analysis called “draw_imp” that imports the results from a previously run Abaqus/Explicit analysis called “draw_exp”:

```
abaqus analysis job=draw_imp oldjob=draw_exp
```

Running Analyses in Abaqus/Explicit

Use the following command to submit an Abaqus/Explicit analysis called “beam” to the default queue:

```
abaqus analysis job=beam convert=all queue=
```

Equivalent results would be obtained from the following series of commands:

```
abaqus datacheck job=beam interactive
abaqus continue job=beam queue=
abaqus convert=all job=beam interactive
```

The CPU-intensive analysis option is run in batch, while the other options are run interactively.

Running Different Phases of an Analysis

Use the following command to perform a parameter check run on an input file called “parmodel”:

```
abaqus job=parmodel parametercheck
```

Use the following command to perform a data check run on an input file called “parmodel” (the parameter check is done again if this job is run after the previous one):

```
abaqus job=parmodel datacheck
```

The following command continues the previous **datacheck** job to execute the analysis:

```
abaqus job=parmodel
continue
```

Abaqus Job Execution Control

Products: Abaqus/Standard Abaqus/Explicit

References:

- *About Execution Procedures*

Overview

The execution procedures for job execution control include **abaqus suspend**, **abaqus resume**, and **abaqus terminate**. These utilities are used to suspend, resume, and terminate Abaqus analysis jobs. Suspending an analysis job will stop its execution and release its license tokens to the free-token pool. Resuming an analysis will reactivate a suspended job and check out license tokens for that job if they are available. The job will be placed in the license queue if license tokens are not available. Terminating an analysis job will stop the executable for the analysis and release its license tokens. A terminated analysis job cannot be resumed.

You can execute these commands remotely; you are not required to execute these commands on the machine running the analysis.

Command Summary

abaqus {suspend | resume | terminate} {**job =** *job-name* | **host =** *hostname* **port =** *port-number*}

Command Line Options

job This option is used to specify the name of the analysis job to suspend, resume, or terminate. When using this option, the command must be executed from the working directory of the job. The command reads the *job-name*.`cid` file to obtain the host name and port number used to signal the job.

host This option is used to specify the host name for the connection running the analysis job. This option must be used in conjunction with the **port** option. The host name can be identified by reading the last line of the *job-name*.`cid` file, which lists *hostname*:*port-number*.

port This option is used to specify the port number for the connection running the analysis job. This option must be used in conjunction with the **host** option. The port number can be identified by reading the last line of the *job-name*.`cid` file, which lists *hostname*:*port-number*.

Abaqus/CAE Execution

Products: Abaqus/CAE

References:

- *About Execution Procedures*

Overview

Abaqus/CAE, an interactive environment for creating, submitting, monitoring, and evaluating results from Abaqus simulations, is executed by running the Abaqus execution procedure and specifying the **cae** parameter.

Command Summary

abaqus cae **[database =** *database-file*]
[replay = *replay-file*]
[recover = *journal-file*]
[startup = *startup-file*]
[script = *script-file*]
[noGUI [= *noGUI-file*]]
[noenvstartup]
[noSavedOptions]
[noSavedGuiPrefs]
[noStartupDialog]
[custom = *script-file*]
[guiTester [= *GUI-script*]]
[guiRecord]
[guiNoRecord]

Command Line Options

database This option specifies the name of the model database file or output database file to open. To specify a model database file, include either the `.cae` file extension or no file extension in the file name. To specify an output database file, include the `.odb` file extension in the file name.

replay This option specifies the name of the file from which Abaqus/CAE commands are to be replayed. The commands in *replay-file* will execute immediately upon startup of Abaqus/CAE. If no file extension is given, the default extension is `.rpy`. You cannot use the `replay` option to execute a script with control flow statements.

recover This option specifies the name of the file from which a model database is to be rebuilt. The commands in *journal-file* will execute immediately upon startup of Abaqus/CAE. If no file extension is given, the default extension is `.jnl`.

startup This option specifies the name of the file containing Python configuration commands to be run at application startup. Commands in this file are run after any configuration commands that have been set in the environment file. Abaqus/CAE does not echo the commands to the replay file when they are executed.

Arguments can be passed into the file by entering `--` on the command line, followed by the arguments separated by one or more spaces. These arguments will be ignored by the Abaqus/CAE execution procedure, but they will be accessible within the script.

script Same as **startup**.

noGUI This option specifies that Abaqus/CAE is to be run without the graphical user interface (GUI). If no file name is specified, an Abaqus/CAE license is checked out and the Python interpreter is initialized to allow interactive entry of Python or Abaqus Scripting Interface commands.

If a file name is specified, Abaqus/CAE runs the commands in the file and exits upon their completion. If no file extension is given, the default extension is `.py`. This option is useful for automating pre- or post-analysis processing tasks without the added expense of running a display. Since no interface is provided, the scripts cannot include any user interaction. If you use the `noGUI` option, Abaqus/CAE ignores any other command line options that you provide.

Arguments can be passed into the file by entering `--` on the command line, followed by the arguments separated by one or more spaces. These arguments will be ignored by the Abaqus/CAE execution procedure, but they will be accessible within the Python script. If you are using the `noGUI` option, you can use an argument to pass in a variable that would otherwise be provided by a command line option. For example, you can pass in the name of a file that would otherwise be specified by the `script` option.

noenvstartup This option specifies that all configuration commands in the environment files should not be run at application startup. This option can be used in conjunction with the **script** command to suppress all configuration commands except those in the **script** file.

noSavedOptions This option specifies that Abaqus/CAE should not apply the display options settings stored in `abaqus_2025.gpr` (for example, the render style and the display of datum planes). For more information, see *Saving your display options settings*.

noSavedGuiPrefs This option specifies that Abaqus/CAE should not apply the GUI settings stored in `abaqus_2025.gpr` (for example, the size and location of the Abaqus/CAE main window or its dialog boxes).

noStartupDialog This option specifies that the **Start Session** dialog box for Abaqus/CAE should not be displayed.

custom This option specifies the name of the file containing Abaqus GUI Toolkit commands. This option executes an application that is a customized version of Abaqus/CAE. For more information, see *Introduction*.

guiTester This option starts a separate user interface containing the Abaqus Python development environment along with Abaqus/CAE. The Abaqus Python development environment allows you to create, edit, step through, and debug Python scripts. For more information, see *The Abaqus Python Development Environment*.

You can specify a script as the argument for this option, which prompts Abaqus/CAE to run a GUI script. Abaqus/CAE closes when the end of the script is reached.

guiRecord This option enables you to record your actions in the Abaqus/CAE user interface in a file named `abaqus.guiLog`. You can also set this option at startup by using the environment variable `ABQ_CAE_GUIRECORD`. The **guiRecord** option cannot be used with the **guiTester** option.

guiNoRecord This option enables you to disable user interface recording when the environment variable `ABQ_CAE_GUIRECORD` is set.

Examples

The following examples illustrate the command line options of the **cae** execution procedure and how arguments are passed to Abaqus/CAE.

Opening a Model Database

The following command will execute Abaqus/CAE and load the model database file called “beam”:

```
abaqus cae database=beam
```

Passing Arguments to a Script

The following command will run the Python script in a file named “try.py” at application startup and pass “argument1” to the script:

```
abaqus cae script=try.py -- argument1
```

The above command will print `argument1` if “try.py” is defined as

```
import sys
print sys.argv[-1]
```

Running Abaqus/CAE without the Graphical User Interface

The following command will run the Python script in a file named “checkPartValidity.py” and pass arguments to the script specifying the model database, the model, and the part. The script is executed by Abaqus/CAE; however, the graphical user interface is never displayed.

```
abaqus cae noGui=checkPartValidity.py -- test.cae Model-1 Part-1
```

The above command will print `Part-1 is valid` if “checkPartValidity.py” is defined as

```
import sys
import os

myMdb= sys.argv[-3]
myModel = sys.argv[-2]
myPart = sys.argv[-1]

mdb = openMdb(myMdb)
model = mdb.models[myModel]
part = model.parts[myPart]

if part.geometryValidity:
    sys.__stderr__.write('%s is valid\n' % myPart)
else:
    sys.__stderr__.write('%s is invalid\n' % myPart)
```

Abaqus/Viewer Execution

Products: Abaqus/Viewer

References:

- *About Execution Procedures*

Overview

Abaqus/Viewer, a subset of Abaqus/CAE that contains only the postprocessing capabilities of the Visualization module, is executed by running the Abaqus execution procedure and specifying the **viewer** parameter.

Command Summary

abaqus viewer
[database = *database-file*]
[replay = *replay-file*]
[startup = *startup-file*]
[script = *script-file*]
[noGUI [= *noGUI-file*]]
[noenvstartup]
[noSavedOptions]
[noSavedGuiPrefs]
[noStartupDialog]
[custom = *script-file*]
[guiTester [= *GUI-script*]]
[guiRecord]
[guiNoRecord]

Command Line Options

database This option specifies the name of the output database file to use if it is different from *job-name*. The procedure searches for *database-file* as entered on the command line with the `.odb` file extension.

replay This option specifies the name of the file from which Abaqus/Viewer commands are read. The commands in *replay-file* will execute immediately upon startup of Abaqus/Viewer. If no file extension is given, the default extension is `.rpy`. You cannot use the `replay` option to execute a script with control flow statements.

startup This option specifies the name of the file containing Python configuration commands to be run at application startup. Commands in this file are run after any configuration commands that have been set in the environment file. Abaqus/Viewer does not echo the commands to the replay file when they are executed.

Arguments can be passed into the file by entering `--` on the command line, followed by the arguments separated by one or more spaces. These arguments will be ignored by the Abaqus/Viewer execution procedure, but they will be accessible within the script.

script Same as **startup**.

noGUI This option specifies that Abaqus/Viewer is to be run without the graphical user interface (GUI). If no file name is specified, an Abaqus/Viewer license is checked out and the Python

interpreter is initialized to allow interactive entry of Python or Abaqus Scripting Interface commands.

If a file name is specified, Abaqus/Viewer runs the commands in the file and exits upon their completion. If no file extension is given, the default extension is `.py`. This option is useful for automating post-analysis processing tasks without the added expense of running a display. Since no interface is provided, the scripts cannot include any user interaction.

noenvstartup This option specifies that all configuration commands in the environment files should not be run at application startup. This option can be used in conjunction with the **script** command to suppress all configuration commands except those in the **script** file.

noSavedOptions This option specifies that Abaqus/Viewer should not apply the display options settings stored in `abaqus_2025.gpr` (for example, the render style and the display of boundary conditions). For more information, see *Saving your display options settings*.

noSavedGuiPrefs This option specifies that Abaqus/Viewer should not apply the GUI settings stored in `abaqus_2025.gpr` (for example, the size and location of the Abaqus/CAE main window or its dialog boxes).

noStartupDialog This option specifies that the **Start Session** dialog box for Abaqus/Viewer should not be displayed.

custom This option specifies the name of the file containing Abaqus GUI Toolkit commands. This option executes an application that is a customized version of Abaqus/Viewer. For more information, see *Introduction*.

guiTester This option starts a separate user interface containing the Python development environment along with Abaqus/Viewer. The Python development environment allows you to create, edit, step through, and debug Python scripts. For more information, see *The Abaqus Python Development Environment*.

You can specify a script as the argument for this option, which prompts Abaqus/Viewer to run a GUI script. Abaqus/Viewer closes when the end of the script is reached.

guiRecord This option enables you to record your actions in the Abaqus/Viewer user interface in a file named `abaqus.guiLog`. You can also set this option at startup by using the environment variable `ABQ_CAE_GUIRECORD`. The **guiRecord** option cannot be used with the **guiTester** option.

guiNoRecord This option enables you to disable user interface recording when the environment variable `ABQ_CAE_GUIRECORD` is set.

Making User-Defined Executables and Subroutines

Products: Abaqus/Standard Abaqus/Explicit

References:

- *About Execution Procedures*

Overview

The **abaqus make** utility is used to create user postprocessing executables and user-defined libraries of Abaqus user subroutines. The commands used to compile and link a user-supplied program or user subroutine source file can be changed using the appropriate Abaqus environment file parameters; i.e., **compile_cpp**, **compile_fortran**, **link_exe**, and **link_sl**. You can skip the compilation step by providing a precompiled object as input for postprocessing programs.

Postprocessing executables created using this procedure must be run using the Abaqus execution procedure. This is necessary to set the operating system environment variables for finding the Abaqus utility libraries. To run a user postprocessing program, use the following command:

```
abaqus job-name
```

User subroutine shared libraries created using this procedure are used by specifying the **usub_lib_dir** variable in the Abaqus environment file. The advantage of doing this is that an analysis using user subroutines can execute without having to compile or link the user subroutine.

User subroutine shared libraries created using the **uniquelibs** option must be run using the Abaqus execution procedure as follows:

```
abaqus job-name uniquelibs
```

Command Summary

abaqus make {**job =** *job-name* | **library =** *source-file*}
[user = {*source-file* | *object-file*}**]**
[directory = *library-dir***]**
[object_type = {`fortran` | `c` | `cpp`}**]**
[uniquelibs]

Command Line Options

job This option is used to create a user-supplied postprocessing program. The value of the option specifies the name of the executable created by this procedure. It is also used as the default source file name.

If no option is given on the command line, you will be prompted for this value.

library This option is used to create user subroutine object files and shared libraries. The value of the option specifies the name of the user subroutine source file to be compiled and linked. The resulting object and shared library files are placed in the directory given by the command line **directory** option. If the **directory** option is not used, the files are placed in the current working directory.

The object file or files created have a suffix indicating if the user subroutine is for Abaqus/Standard or Abaqus/Explicit. The Abaqus/Standard object file suffix is `-std`. Abaqus/Explicit has single and double precision object files; the object file suffixes are `-xpl` and `-xplD`. The Abaqus/Standard user subroutine shared library that is created is called `standardU`, and the

Abaqus/Explicit shared libraries are called `explicitU` and `explicitU-D`. If the **directory** option is used and it contains object files with the appropriate suffix for the shared library that is being created, those files are linked to the shared library.

user This option is valid only when used in conjunction with the **job** option. It is used to specify the name of the source or object file containing your program if it is different from *job-name*. If a file extension is not provided, the option value with a Fortran source file extension is sought. If a file by this name is not found, the option value with an object file extension is sought.

directory This option is valid only when used in conjunction with the **library** option. It is used to specify the absolute path of the destination of the user subroutine object and shared library files that will be created by the procedure. It is also used to specify the location of additional object files that are to be linked to the shared library or libraries being created. If the option is omitted, the files created by the procedure are placed in the current working directory.

object_type This option is valid only when used in conjunction with the **job** option. It is used to specify the type of object file, either Fortran, C, or C++, given by the **job** or **user** option.

uniquelibs This option is valid only when used in conjunction with the **library** option. It is used to create Abaqus user subroutines in individual dynamic link libraries or shared objects. These libraries contain any utility routines included in the user subroutine. The library naming conventions are *user_subroutine_name*`_usub.dll` and *user_subroutine_name*`_usub.so`.

To direct Abaqus to use these user subroutines during input file execution, you must specify the **uniquelibs** option in the **abaqus** execution procedure (see *Abaqus/Standard and Abaqus/Explicit Execution*).

The use of global data between user subroutines via common blocks usage in Fortran routines is not allowed. If encountered, Abaqus issues a warning during the build process. To ensure the sharing of global data transfer among user subroutines, you must declare `DLLExport` and `DLLImport` statements accordingly.

Examples

To create an executable called “pprocess” given a Fortran source file of the same name, use the following command:

```
abaqus make job=pprocess
```

This program can then be run using the command

```
abaqus pprocess
```

Optimization Execution

Products: Abaqus/CAE

References:

- *About Execution Procedures*

Overview

Optimization using TOSCA for Abaqus is executed by running the Abaqus execution procedure and by providing the name of the parameter file, which, in turn, refers to an Abaqus input file. You can define your optimization model using Abaqus/CAE, and you can use the **Optimization Process Manager** to create the parameter file and the Abaqus/CAE input file.

Command Summary

abaqus optimization

task = *parameter file*
job = *results folder*
[cpus = *total number-of-cpus***]**
[gpus = *total number-of-gpgpus***]**
[memory = *memory-size***]**
[interactive]
[globalmodel = {*results file-name* **|** *ODB output database file-name* **|** *SIM database file-name* **}]**
[scratch = *scratch-dir***]**

Command Line Options

task

This option is used to specify the file containing the parameters that are used to execute the optimization. The parameter file also contains information about the input file that is associated with the optimization.

If this option is omitted from the command line, you will be prompted for this value.

job

This option is used to specify the name of the folder in which the results of the optimization are stored.

If this option is omitted from the command line, you will be prompted for this value.

cpus

This option specifies the total number of processors to use during an analysis run if parallel processing is available. The default value for this parameter is 1 and can be changed in the environment file (see *Environment File Settings*).

gpus

This option specifies acceleration of the Abaqus/Standard direct solver. This option is meaningful only on computers equipped with appropriate GPGPU hardware. By default, GPGPU solver acceleration is not activated. The value of this parameter is the total number of GPGPUs to be used in an Abaqus/Standard analysis. In an MPI-based analysis, this parameter is the total number of GPGPUs to be used across all hosts.

GPGPU-based solver acceleration can also be set in the environment file (see *Environment File Settings*).

memory Maximum amount of memory or maximum percentage of the physical memory that can be allocated during the input file preprocessing and during the Abaqus/Standard analysis phase (see *Managing Memory and Disk Resources*). The default values can be changed in the environment file (see *Environment File Settings*).

interactive This option will cause the job to run interactively. The Tosca execution output will be output to the screen. The default **run_mode** can be set in the environment file (see *Environment File Settings*).

globalmodel This option specifies the name of the global model's results file, ODB output database file, or SIM database file from which the results are to be interpolated to drive a submodel analysis. This option is required whenever a submodel analysis or submodel boundary condition reads data from the global model's results.

The file extension is optional. If you omit the file extension, Abaqus uses the results file. If the results file does not exist, Abaqus uses the ODB output database file. If both the results file and the ODB output database file do not exist, Abaqus uses the SIM database file.

scratch This option is used to specify the name of the directory used for scratch files. On Linux platforms the default value is the value of the `$TMPDIR` environment variable or `/tmp` if `$TMPDIR` is not defined. On Windows platforms the default value is the value of the `%TEMP%` environment variable or `\TEMP` if this variable is not defined. During the analysis a subdirectory will be created under this directory to hold the analysis scratch files. The default value for this parameter can be set in the environment file (see *Environment File Settings*).

Examples

To run the optimization with the parameter file `WeightReduction` and the results folder `WeightReductionResults`, use the following command:

```
abaqus optimization task=WeightReduction job=WeightReductionResults
```

Parametric Studies

Products: Abaqus/Standard Abaqus/Explicit

References:

- *About Execution Procedures*
- *Abaqus/Standard and Abaqus/Explicit Execution*

Overview

The **abaqus script** facility indicates that a parametric study is to be done (see *Scripting Parametric Studies*). Each analysis involved in the design can be executed using the `execute` command (see *aStudy.execute()*). You can add any necessary Abaqus execution options (refer to *Abaqus/Standard and Abaqus/Explicit Execution*) to the execution command for each of the analyses by specifying them on the **execOptions** option of the `execute` command. If the script file contains references to other input files, these files must be located in the same directory as the script file. The files created by the execution of the script file are placed in the directory from which the Abaqus execution procedure is run.

Command Summary

abaqus script **[**=*script-file*]
[startup = *startup file-name*]
[noenvstartup]

Command Line Options

script-file When a script file name is specified, the parametric study module is imported and the instructions in the parametric study script file are executed. If the script file name is omitted from the command line, the Python interpreter is initialized by importing the parametric study module.

startup This option specifies the name of the file containing Python configuration commands to be run at application startup. Commands in this file are run after any configuration commands that have been set in the environment file.

noenvstartup This option specifies that all configuration commands in the environment files should not be run at application startup. This option can be used in conjunction with the **startup** command to suppress all configuration commands except those in the **startup** file.

Examples

Use the following command to execute the Python script in a file named “parstudy.psf”:

```
abaqus script=parstudy
```

The following command will initiate a Python scripting session:

```
abaqus script
```

In a Python scripting session the following command will execute the Python script in a file named “scriptfile”:

```
script("scriptfile")
```

Abaqus Documentation

Products: Abaqus/Standard Abaqus/Explicit Abaqus/CAE

References:

- *About Execution Procedures*
- *Getting help*

Overview

Abaqus documentation is installed separately from the product and is viewed through a web browser.

Using Abaqus Documentation

The documentation consists of the following guides:

- *Abaqus/CAE User's Guide*
- *Abaqus Analysis Guide*
- *Abaqus Benchmarks Guide*
- *Abaqus Constraints Guide*
- *Abaqus Elements Guide*
- *Abaqus Example Problems Guide*
- *Abaqus Execution Guide*
- *Getting Started with Abaqus/CAE*
- *Abaqus GUI Toolkit User's Guide*
- *Abaqus GUI Toolkit Reference Guide*
- *Abaqus Interactions Guide*
- *Abaqus Introduction & Spatial Modeling Guide*
- *Abaqus Keywords Guide*
- *Abaqus Materials Guide*
- *Abaqus Output Guide*
- *Abaqus Prescribed Conditions Guide*
- *Abaqus Release Notes*
- *Abaqus Scripting User's Guide*
- *Abaqus Scripting Reference Guide*
- *Abaqus Theory Guide*
- *Abaqus User Subroutines Guide*
- *Abaqus Verification Guide*
- *Abaqus Configuration Guide*

To view the documentation:

1. Type **abaqus doc**.

   The documentation opens in a web browser.
2. Click the title of a book to display it.
3. Expand the topic headings in the table of contents. To jump directly to a section whose title is displayed in the table of contents, click that title.

Configuration of Documentation Application

The **abaqus doc** command locates a web browser executable.

Configuration of Web Browser

If the HTML documentation was installed and configured by your system administrator, the **abaqus doc** command will locate a web browser executable as follows:

Windows platforms

The **abaqus doc** command uses your default web browser.

Linux platforms

The **abaqus doc** command searches the system path for Firefox. If the help system cannot find Firefox, an error is displayed.

The **browser_type** and **browser_path** variables can be set in the Abaqus environment file to modify the behavior of this command. For more information, see *System customization parameters*.

Command Summary

abaqus doc

Abaqus Co-Simulation

This section desribes using commands to run Abaqus co-simulation procedures.

In this section:

- *SIMULIA Co-Simulation Engine Director Execution*
- *Abaqus/Standard, Abaqus/Explicit, and FMU Co-Simulation Execution*
- *SIMULIA Co-Simulation Engine FMU Execution*

SIMULIA Co-Simulation Engine Director Execution

Products: Abaqus/Standard Abaqus/Explicit

References:

- *About Execution Procedures*

Overview

Co-simulation between Abaqus/Standard and Abaqus/Explicit is governed by an additional process, the SIMULIA Co-Simulation Engine (CSE) director. Typically, you are not required to invoke the CSE director process; it is invoked automatically when you run the Abaqus co-simulation procedure (*Abaqus/Standard, Abaqus/Explicit, and FMU Co-Simulation Execution*) or if you submit the co-simulation from Abaqus/CAE.

If you are unable to use the Abaqus co-simulation procedure or Abaqus/CAE and are required to submit the co-simulation analyses separately using the Abaqus execution procedure (*Abaqus/Standard and Abaqus/Explicit Execution*), you must invoke the CSE director as described in this section.

Command Summary

abaqus cse

job = *cosim-job-name*
configure = *configuration file-name*
listenerport = *listener port-number*
[datacheck]
[interactive]
[license_type = {`token` | `credit`**}]**

Command Line Options

job The value of this option specifies the name of the co-simulation summary log file generated during the run. If this option is omitted from the command line, you will be prompted for its value.

configure This option specifies the name of the SIMULIA Co-Simulation Engine configuration file that governs the co-simulation. For more information, see *Using the SIMULIA Co-Simulation Engine Configuration File*.

listenerport This option is used to specify the TCP/UDP port number for co-simulation inbound messages to the director. Set **listenerport** equal to the port number used for the connection.

datacheck This option causes the director to check the correctness of the configuration file only.

interactive This option causes the director to run interactively.

license_type This option is used to control the type of license Abaqus uses for the SimUnit license model. The possible values are `token` and `credit`. The default value is `token`. This capability is also supported through the Abaqus environment file with the environment variable **license_type**. For more information, see *License management parameters*.

Examples

The following example illustrates the different functions and capabilities of the co-simulation director execution procedure when you are required to submit the co-simulation analyses separately.

Running an Abaqus/Standard to Abaqus/Explicit Co-Simulation

Use the following command for the first Abaqus analysis, running on machine “earth,” to connect to the co-simulation director, running on machine “mercury” and listening on port 44444:

```
abaqus job=explicit csedirector=mercury:44444
```

Use the following command for the second Abaqus analysis, running on machine “venus,” to connect to the co-simulation director, running on machine “mercury” and listening on port 44444:

```
abaqus job=standard csedirector=mercury:44444
```

Use the following command for the co-simulation director, running on machine “mercury,” to operate according to the co-simulation configuration defined in the file `explicit_standard_config.xml` and to receive communication through port 44444:

```
abaqus cse job=cosim listenerport=44444
   configure=explicit_standard_config.xml
```

Abaqus/Standard, Abaqus/Explicit, and FMU Co-Simulation Execution

Products: Abaqus/Standard Abaqus/Explicit

References:

- *About Execution Procedures*

Overview

Co-simulation between Abaqus/Standard and Abaqus/Explicit and co-simulation format Functional Mockup Unit (FMU) files can be executed by running the Abaqus co-simulation procedure. Several parameters can be set either on the command line or in the environment file (see *Environment File Settings*). See *About Co-Simulation* for details about supported co-simulation interactions. Refer to `http://fmi-standard.org` for more information on the Functional Mockup Interface standard.

A co-simulation analysis executes the specified "child" analyses and directs the communication of the processes according to the co-simulation configuration file specifications. The co-simulation execution procedure allows you to enter a single command to run the co-simulation and should be used whenever possible (see *Limitations* below). If you are unable to use the Abaqus co-simulation execution procedure, you are required to do the following:

- submit the co-simulation analyses separately using the Abaqus execution procedure (*Abaqus/Standard and Abaqus/Explicit Execution*) or the FMU execution procedure (*SIMULIA Co-Simulation Engine FMU Execution*), and
- invoke the SIMULIA Co-Simulation Engine (CSE) director (*SIMULIA Co-Simulation Engine Director Execution*).

The co-simulation execution procedure supports a subset of the options that are available for the Abaqus execution procedure; these options are included in the command summary below.

Composition of Command Line Arguments

You can use comma-separated lists of options for command line arguments for the participating child analyses. Command line arguments (see the command line options below) are categorized as either

- global options, taking a single argument;
- applicable to Abaqus child analyses, taking a comma-separated list whose length is the number of participating Abaqus jobs; or
- applicable to FMUs, taking a comma-separated list whose length is the number of participating FMUs.

Therefore, the length of command line argument lists generally varies for co-simulation analyses that use Abaqus products and FMUs.

Allocating CPUS for Parallel Processing with Abaqus Jobs

Your CPU allocation specification applies to Abaqus/Standard and Abaqus/Explicit child jobs. Functional Mockup Units (FMUs) do not support multiple-CPU use. Three methods are available for allocating CPUs to child analysis jobs for parallel processing: specifying the number of CPUs for each job, distributing CPUs between analysis jobs, and distributing CPUs between analysis products.

Specifying the Number of CPUs for Each Job

The most direct method of allocating CPUs is to specify the number of CPUs to be used for each child analysis. You provide a comma-separated list of values using the **cpus** parameter.

Distributing CPUs between Analysis Jobs

You can specify the total number of CPUs to be used for your co-simulation analysis and weighting factors that determine the distribution of the CPUs between the child analyses. This method enables you to specify a CPU count that relates directly to your resource limits and to describe the relative computational needs of the child analyses. You provide one value for the number of CPUs to allocate for the co-simulation using the **cpus** parameter, and you define weight factors using the **cpuratio** parameter.

Weight factors are floating point numbers and are considered in a normalized sense. For example, if you want to specify that the CPU allocation for the first child job is four times that of the second job, you can provide any of the following pairings:

```
cpuratio=4.0,1.0
cpuratio=16,4
cpuratio=0.8,0.2
```

Distributing CPUs between Analysis Products

You can specify the total number of CPUs to be used for your co-simulation analysis and weighting factors that determine the distribution of the CPUs between the analysis products involved in the co-simulation. This method enables you to specify a CPU count that relates directly to your resource limits and to describe the relative computational needs of the child analyses based on the analysis product used (Abaqus/Standard and Abaqus/Explicit). You provide one value for the number of CPUs to allocate for the co-simulation using the **cpus** parameter, and you define weight factors in the environment file using the **cpus_weight_std** and **cpus_weight_xpl** environment variable parameters (see *Co-Simulation Parameters*).

Weight factors are interpreted in a normalized sense. For example, if you want to specify that the CPU allocation for the Abaqus/Standard analysis is twice that of the Abaqus/Explicit analysis, you define the parameters in the environment file as follows:

```
cpus_weight_xpl=1
cpus_weight_std=2
```

Rounding Considerations for Distributing CPUS

In cases where the distribution of the CPUs between analysis jobs or analysis products does not result in whole numbers, Abaqus rounds down the CPU allocation for the first jobs listed in the **job** parameter and rounds up the allocation for the final job listed. For example, if 8 CPUs are allocated and the CPU allocation for the Abaqus/Standard analysis is twice that of the Abaqus/Explicit analysis, the distribution between Abaqus/Explicit and Abaqus/Standard is 2/6 if the Abaqus/Explicit job is listed first and is 3/5 if the Abaqus/Standard job is listed first.

Specifying Options for Abaqus Child Analyses

Command line options that pertain to the child analyses require you to enter a comma-separated list of values. The number and order of entries in the lists must be consistent for all child analysis options to obtain the desired co-simulation execution behavior. For example, in an Abaqus/Standard to Abaqus/Explicit co-simulation, if you specify the job name for the Abaqus/Standard analysis as the first entry for the job parameter, the first entry for the remainder of the child analysis options applies to the Abaqus/Standard analysis.

If an option is relevant for only one of the child analyses, you can enter a value of `NONE` for the analysis in which the option is not relevant. In cases where you want to use the default settings for an option for all child analyses or want to use environment settings to control the behavior, you need not provide that option in the command line.

Specifying Options for Functional Mockup Unit (FMU) Files

You specify command line option **fmu** and, optionally, **fmuinstance** to use Functional Mockup Unit (FMU) files in the co-simulation analysis. You use a comma-separated list to specify multiple FMUs.

Limitations

The following limitations apply to the co-simulation execution procedure:

- The analyses can be run only on a single machine or a compute cluster where the head node can be shared by both child analysis jobs.
- Co-simulation with third-party applications is not supported with this execution procedure; for information on Abaqus job execution for co-simulation with third-party applications, consult the third-party program documentation.

Command Summary

abaqus cosimulation

cosimjob = *cosim-job-name*
configure = *co-simulation configuration file name*
job = *comma-separated list of Abaqus job names*
[cpus = {_number-of-cpus_ | *comma-separated list of number-of-cpus for the Abaqus jobs*}]
[cpuratio = *comma-separated list of weight factors specifying cpu allocation to Abaqus child analyses*]
[interactive] | **[background]** | **[queue [=** *queue-name* **][after =** *time*]]
[timeout = *co-simulation timeout value in seconds*]
[listenerport = *co-simulation director listener socket port number*]
[portpool = *colon-separated pair of socket port numbers*]
[license_type = {`token` | `credit`**}]**
[input = *comma-separated list of Abaqus-job input-file names*]
[user = *comma-separated list of {source-file | object-file} names for the Abaqus jobs*]
[globalmodel = *comma-separated list of {results file | ODB output database file | SIM database file} names for the Abaqus jobs*]
[memory = *comma-separated list of memory-sizes for the Abaqus jobs*]
[oldjob = *comma-separated list of oldjob-names for the Abaqus jobs*]
[double = *comma-separated list of double precision executable settings for the Abaqus jobs*]
[scratch = *comma-separated list of scratch-dir names for the Abaqus jobs*]
[output_precision = *comma-separated list of {*`single` | `full`*} for the Abaqus jobs*]
[fmu = *comma-separated list of functional mockup unit files*]
[fmuinstance = *comma-separated list of functional mockup instance names*]

Command Line Options

Required Global Options

cosimjob This option specifies the name of the co-simulation summary log file generated during the run. If this option is omitted from the command line, you will be prompted for its value.

configure This option specifies the name of the co-simulation configuration file. If this option is omitted from the command line, an attempt is made to automatically generate a configuration file based on the co-simulation controls specified in the input files for the child analyses. If successful, the generated configuration file is used in the co-simulation; if unsuccessful, you will be prompted for a value.

Required Option for Abaqus Child Analyses

job The comma-separated values of this option specify the names of all Abaqus child analysis files generated during the run. If this option is omitted from the command line, you will be prompted for its value.

Parallel Processing Options

cpus This option is used to specify how CPUs are allocated for the co-simulation during parallel processing. The default value for this parameter is the number of Abaqus child analyses, and can be changed to a different value in the environment file (see *Environment File Settings*).

If this option is set equal to a single value, that value specifies the total number of processors allocated for the co-simulation, which can be distributed between child analyses or between analysis products. The distribution of the CPUs between child analyses is split evenly by default and may be further controlled either by using the **cpuratio** parameter or by defining the distribution of the CPUs between analysis products by setting the **cpus_weight_std** and **cpus_weight_xpl** environment file parameters (see *Co-Simulation Parameters*).

If this option is set equal to a comma-separated list of values, these values specify the number of processors to be used for each child analysis.

cpuratio The comma-separated values of this option specify the relative weighting of the distribution of processors allocated to each Abaqus child analysis. This option is valid only when the **cpus** option is set to a single value.

Additional Global Options Available

interactive This option causes the co-simulation job to run interactively. A summary log file will be output to the screen, and the child analysis summary output will be written to their separate log files.

background This option submits the co-simulation job to run in the background, which is the default. Log file output is saved for the co-simulation job in the file *cosim-job-name*`.log` and in the child analysis files *job-name*`.log` in the current directory.

queue This option submits the co-simulation job to a batch queue. If the option appears with no value, the job is submitted to the system default queue. Quoted strings are allowed. The available queues are site specific. Contact your site administrator to find out more about local queuing capabilities.

after This option is used in conjunction with the **queue** option to specify the time at which the job will start in the selected batch queue. This capability is supported for each individual site through the Abaqus environment file. (See the *Abaqus Configuration Guide* for details.)

timeout This option is used to specify a timeout value in seconds for the co-simulation connection. Abaqus terminates if it does not receive any communication between the child analysis processes during the time specified. The default value is 3600 seconds. The default timeout value that Abaqus uses can be set with the **cosimulation_timeout** parameter in the environment file (see *Environment File Settings*).

listenerport This option is used to specify a TCP/UDP port numbers to be used by the co-simulation director process. This option can be used to enable additional co-simulation analyses, for example a third-party analysis job, to join a co-simulation initiated with this procedure. If this option is not specified the listenerport value is determined based on available open socket ports, and any constraints you set with the **portpool** option.

portpool This option is used to specify a colon-separated pair of TCP/UDP port numbers that represent the start and end value of port numbers to be used when establishing connections between the child processes. The default range is `51000:52000`. The default range that Abaqus uses can be set with the **portpool** parameter in the environment file (see *Environment File Settings*).

license_type This option is used to control the type of license Abaqus uses for the SimUnit license model. The possible values are `token` and `credit`. The default value is `token`. This capability is also supported through the Abaqus environment file with the environment variable **license_type**. For more information, see *License management parameters*.

Additional Options for Abaqus Child Analyses

input The comma-separated values of this option specify the Abaqus child analysis input file names, which may be given with or without the `.inp` extension (if the extension is not supplied, Abaqus appends it automatically). For each child analysis, if this option is not supplied, the procedure looks for an input file called *job-name*`.inp` in the current directory. If *job-name*`.inp` cannot be found, the procedure prompts for the input file name.

user The comma-separated values of this option specify the names of Fortran source or object files that contain any user subroutines to be used in the analysis. The names of the user routines may contain a path name and may be given with or without a file extension.

globalmodel The comma-separated values of this option specify the names of the global model's results (`.fil`) file, ODB output database (`.odb`) file, or SIM database (`.sim`) file from which the results are to be interpolated to drive a submodel analysis. This option is required whenever a submodel analysis or submodel boundary condition reads data from the global model's results.

The file extension is optional. If you omit the file extension, Abaqus uses the results file. If the results file does not exist, Abaqus uses the SIM output database file. If both the results file and the SIM output database file do not exist, Abaqus uses the ODB database file.

memory The comma-separated values of this option specify the maximum amount of memory or maximum percentage of the physical memory that can be allocated during the input file preprocessing and during the Abaqus/Standard analysis phase (see *Managing Memory and Disk Resources*).

oldjob The comma-separated values of this option identify the files from a previous run from which a restart run is to start or from which to import results. You specify the *oldjob-names* (without file extensions). For an import or restart analysis, you can include a full or a relative path. This option is required when a restart or import analysis reads data from the restart file. The *oldjob-names* must be different from the current *job-names*.

double This option is applicable only for an Abaqus/Explicit analysis.

The comma-separated values of this option specify the double precision executable settings to use; the value for the Abaqus/Standard analysis is always `NONE`. The possible values for the Abaqus/Explicit analysis are `both`, `constraint`, `explicit`, and `off`. This capability is also supported through the Abaqus environment file with the environment variable **double_precision** (see *Environment File Settings*).

If the **double** option is omitted for an Abaqus/Standard to Abaqus/Explicit co-simulation, the Abaqus/Explicit packager and analysis run in double precision.

If **double**=`both`, both the Abaqus/Explicit packager and analysis run in double precision.

If **double**=`constraint`, the constraint packaging and constraint solver in Abaqus/Explicit run in double precision, while the Abaqus/Explicit packager and Abaqus/Explicit analysis continue to run in single precision.

If **double**=`explicit`, the Abaqus/Explicit analysis runs in double precision, while the packager still runs in single precision.

If **double**=`off`, the environment file setting is overridden if necessary to invoke both the Abaqus/Explicit packager and Abaqus/Explicit analysis in single precision. For a discussion of when to use the double precision executable, see *Defining an Analysis*.

scratch The comma-separated values of this option specify the names of the directories used for scratch files. On Linux platforms the default value is the value of the `$TMPDIR` environment variable or `/tmp` if `$TMPDIR` is not defined. On Windows platforms the default value is the value of the `%TEMP%` environment variable or `\TEMP` if this variable is not defined. During the analysis a subdirectory will be created under this directory to hold the analysis scratch files.

output_precision The comma-separated values of this option specify the precision of the field output written to the output database files (*job-name*`.odb`). Using **output_precision**=`full` results in double precision nodal and element field output for Abaqus/Standard analyses. To obtain double precision nodal field output for Abaqus/Explicit analyses, use the **double** option in addition to using **output_precision**=`full`. The element field output in double precision is not applicable for Abaqus/Explicit. History output is available only in single precision.

Additional Options for Functional Mockup Unit (FMU) Files

fmu The comma-separated values of this option specify the co-simulation FMU file names, which may be given with or without the `.fmu` extension. If you intend to instantiate an FMU file multiple times, you must repeat its file name in this list and provide the FMU instance names using the **fmuinstance** option.

fmuinstance The comma-separated values of this option specify the FMU instance names. The entries for this option are interpreted as having a list-order association with the list provided with the **fmu** option. If you do not provide this option, the FMU instance names will be the file names (without the file extension) specified with the **fmu** option.

Examples

The following examples illustrate the different functions and capabilities of the **abaqus cosimulation** execution procedure.

Running an Abaqus/Standard to FMU Co-Simulation Interactively

Use the following command to run a co-simulation between a heat transfer analysis called “plant” and an FMU file named “controller.fmu”:

```
abaqus cosimulation cosimjob=cosim_controls
   job=plant configure=cosim_controls fmu=controller interactive
```

Submitting an Abaqus/Standard to Abaqus/Explicit Co-Simulation to a Batch Queue

Use the following command to submit a co-simulation for an Abaqus/Explicit analysis called “beam” and an Abaqus/Standard analysis called “beam2” to a batch queue named “long” and to allocate 8 cores to the Abaqus/Explicit analysis and 4 cores to the Abaqus/Standard analysis:

```
abaqus cosimulation cosimjob=beam job=beam,beam2
   configure=beam_config cpus=8,4
queue=long
```

SIMULIA Co-Simulation Engine FMU Execution

Products: Abaqus/Standard Abaqus/Explicit

References:

- *About Execution Procedures*
- *SIMULIA Co-Simulation Engine Director Execution*

Overview

Any number of Functional Mockup Unit (FMU) files can be used in a signals/actuators co-simulation with Abaqus/Standard or Abaqus/Explicit. FMU files conform to the Functional Mockup Interface (FMI) standard defined by the Modelisar organization.
For more information on this standard or for a list of tools that provide the FMU file export capability, refer to `http://fmi-standard.org/`. You will use separate invocations of the FMU execution procedure for each FMU instance participating in your co-simulation.



Warning:

Use of this procedure is recommended only in cases where the more general co-simulation execution procedure (*Abaqus/Standard, Abaqus/Explicit, and FMU Co-Simulation Execution*) does not support your co-simulation use of FMUs.

The FMU capability is intended primarily for co-simulation analyses involving Abaqus solvers; however, if you have a co-simulation containing only FMUs, then the only supported time negotiation method is constant time step.

Command summary

abaqus fmu
csedirector = *CSE director hosts:port-number*
fmu = *fmu-file-name*
instance = *co-simulation-instance-name*
[interactive]
[timeout = *timeout value in seconds***]**

Command line options

csedirector This option is used to specify the connection for co-simulation messages between the CSE director and the FMU execution. The CSE director entry consists of a host name and the listener TCP/UDP port number separated by a colon (e.g., `earth:30000` indicates that the CSE director is running on machine “earth” and using a listener port of “30000”).

fmu This option specifies the FMU file name to be used in the co-simulation.

instance This option specifies the instance name for the FMU. This instance name will correspond to the `componentInstance` name in the co-simulation configuration file provided to the CSE director. You can employ a single FMU file multiple times in a co-simulation by making repeated use of the FMI execution procedure with unique instance names.

interactive This option causes the director to run interactively.

timeout This option is used to specify a timeout value in seconds for the co-simulation director connection. The FMI execution terminates if it does not receive any communication from the CSE director during the time specified. The default value is 3600 seconds.

Examples

The following example illustrates using the FMU execution procedure in a co-simulation involving Abaqus/Standard and an FMU. This example considers the FMU to be interacting with the Abaqus/Standard job; Abaqus/Standard provides an interface to FMUs.

Running the Co-Simulation Director

Use the following command for the co-simulation director running on machine "mercury" to operate according to the co-simulation configuration defined in the file cosim_config.xml and to receive communication via port 44444:

```
abaqus cse job=cosim listenerport=44444
   configure=cosim_config.xml
```

Based on the contents of the file cosim_config.xml, the director will await connections from Abaqus/Standard and an FMU.

Running the First FMU-Based Simulation

Use the following command for the FMI analysis, which will connect to the CSE director on machine "mercury" via port 44444:

```
abaqus fmu fmu=controller.fmu instance=controller1
   csedirector=mercury:44444
```

Running the Second FMU-Based Simulation

Use the following command for the FMI analysis, which will connect to the CSE director on machine "mercury" via port 44444:

```
abaqus fmu fmu=controller.fmu instance=controller2
   csedirector=mercury:44444
```

Running the Abaqus/Standard Simulation

Use the following command for the Abaqus/Standard analysis, which will connect to the CSE director on machine "mercury" via port 44444:

```
abaqus job=standard csedirector=mercury:44444
```

Abaqus Utilities

This section describes using commands to run Abaqus utilities.

In this section:

- *Hardware System Verification Process*
- *Python Execution*
- *Licensing Utilities*
- *Querying the Keyword/Problem Database*
- *Fetching Sample Input Files*
- *Reducing Analysis Database File Sizes for Restart*
- *Output Database Upgrade Utility*
- *SIM Database Utilities*
- *Translating ODB Output Database Files to SIM Format*
- *Generating Output Database Reports*
- *Joining Output Database (.odb) Files from Restarted Analyses*
- *Combining Output from Substructures*
- *Recovering Results from a Substructure*
- *Combining Data from Multiple Output Databases*
- *Network Output Database File Connector*
- *Mapping Thermal and Magnetic Loads*
- *Element Matrix Assembly Utility*
- *Encrypting and Decrypting Abaqus Input Data*
- *ASCII Translation of Results (.fil) Files*
- *Joining Results (.fil) Files*

Hardware System Verification Process

Products: Abaqus/Standard Abaqus/Explicit

References:

- *About Execution Procedures*

Overview

The hardware system verification tool provides a way to ensure that the hardware running the Abaqus solvers provides the same results data as that obtained at SIMULIA during Quality Assurance testing. Systems in a shared resource pool may have several variations of hardware and software packages installed to accommodate numerous software applications. This procedure will provide confidence that the Abaqus solver products are running and generating the same results as those obtained at SIMULIA.

Tests can be executed in serial mode or in parallel mode as specified by the **cpus** option. The tool can be submitted via the **queue** option to test various resources in the shared pool. Part of the validation is to execute with distributed memory parallel using MPI technology.

Command Summary

abaqus sysVerify **[cpus =** *number-of-cpus***]**
[interactive] | [background] | [queue [= *queue-name* **]]**

Command Line Options

cpus This option is used to specify the number of processors to use during the verification. If the **cpus** option is not defined or is set to 1, only the serial tests are executed during the validation. If the **cpus** option is set to 4, both serial and parallel tests are executed with the parallel tests using MPI with 2 cores on two nodes.

interactive This option is used to run the process interactively with the actual analysis still running in the background. The status of the testing is displayed to the screen as the tests are executed. The default **run_mode** can be set in the environment file (see *Environment File Settings*).

background This option is used to submit the process to run in the background, which is the default. Log file output is saved in the file `systemValidation.log` in the current directory. The default method for submitting the job can be set in the environment file by using the **run_mode** parameter (see *Environment File Settings*).

queue This option is used to submit the process to a batch queue. If the option appears with no value, the process is submitted to the system default queue. Quoted strings are allowed. The available queues are site specific. Contact your site administrator to find out more about local queuing capabilities. Use **information**=`local` to see what local queuing capabilities have been installed. The default method for submitting the process can be set in the environment file by using the **run_mode** parameter (see *Environment File Settings*).

Python Execution

Products: Abaqus/Standard Abaqus/Explicit

References:

- *About Execution Procedures*

Overview

The Python language is used throughout Abaqus: in the Abaqus Scripting Interface, in the Abaqus environment file, and to perform parametric studies. The **abaqus python** facility is used to access the Python interpreter.

You can generate, modify, and read SIM documents using the SIM Results Python API.

For more information on standard Python command line options, see the documentation on the official Python website (*https://www.python.org/*).

Simulation Interchange Model

The Simulation Interchange Model (SIM) is the SIMULIA data storage and exchange mechanism used by **3D**EXPERIENCESIMULIA apps. It is a database designed to handle large volumes of data efficiently. It supports both model and results data.

The data access starts with accessing a file named `.SMAManifest`. This is an `xml` file that groups other files that make up the database: binary files with extensions `.SMAFocus` and `.SMABulk`.

You generate a SIM document by running a **3D**EXPERIENCE simulation or by running a stand-alone Abaqus execution using the **resultsFormat**=`SIM` option (see *Abaqus/Standard and Abaqus/Explicit Execution*). You can read, access, or modify the SIM document using the SIM Results API.

SIM Results Python API

A SIM document can store both model and results data. You can visualize the results SIM document in the Physics Results Explorer app on the **3D**EXPERIENCE platform. While results documents are usually generated by SIMULIA apps, they can also be generated, modified, and read using the SIM Results API, available in both C++ and Python. This section discusses the SIM Results Python API.

The SIM Results Python API has two main classes: SIMResultsReader and SIMResultsWriter, which allow you to read and write results.

Detailed instructions for using the SIM Results Python API are available in the Developer Assistance documentation at *https://www.3ds.com/support/documentation/developer-guides/*. Select the current release, and navigate to **Native Apps Automation > Simulation > Physics Simulation > Simulation Interchange Model** to view the following topics:

- *Reading Simulation Results from a SIM file (Python)*
- *Writing Simulation Results to a SIM file (Python)*
- *SIMResultsReader reference (Python)*
- *SIMResultsWriter reference*
- *SIM Utilities reference (Python)*

The *SIMULIA Established Products CAA API* media contains the SIM Results Python API. For more information, see the SIMULIA Installation Guide.

You can run a script generated by this API from the command prompt using the following command:

```
abaqus python python_file_name sim=sim_file_name log=log_file_name
```

Features

The SIM Results Python API:

- uses Numpy arrays for efficiency,
- throws clear exceptions during the reading and writing phase,
- supports Unicode characters,
- uses very few classes as most of the methods are directly under reader and writer classes,
- supports multiple field classes per output location, and
- provides functionality to read or write basic model data (such as nodes, elements, materials, and sections) and supports full results data.

SIM Data Model

Figure 1 shows the SIM data model.

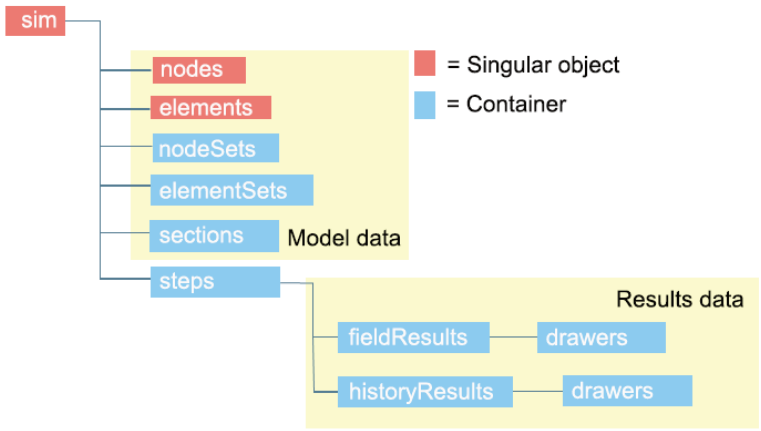


Figure 1: SIM data model.

Limitations

The SIM Results Python API has the following limitations:

- The API does not provide the ability to read and write many model attributes such as loads, boundary conditions, interactions, material properties, and surfaces. The *SIMResultsReader reference* and *SIMResultsWriter reference* topics detail the coverage attributes.
- There are no methods to add part and assembly information using the SIMResultsWriter API.

Command Summary

abaqus python	[*python_file_name*] **[sim =** *sim_file_name***]** **[log =** *log_file_name***]**

Command Line Options

python_file_name	The Python interpreter executes the instructions in the specified *python_file_name*. If this option is omitted from the command line, the Python interpreter is started in interactive mode.
sim	This option specifies the name of the SIM file.
log	This option specifies the name of the log file. The API adds warning and error messages and other output information to this file.

Licensing Utilities

Products: Abaqus/Standard Abaqus/Explicit

References:

- *About Execution Procedures*

Overview

The **abaqus licensing** utilities provide management and monitoring tools for both types of Abaqus licensing: FLEXnet and Dassault Systèmes licensing.

Executing the **abaqus licensing** command without additional arguments displays a command usage summary of all available utilities.

For a detailed description of all the FLEXnet Licensing utilities, refer to the FLEXnet End User Guide available from *https://www.3ds.com/products-services/simulia/*. The command summary below lists several of the most useful licensing utilities.

For more information, see the following guide in the user assistance:

SIMULIA Licensing

Command Summary

abaqus licensing **[lmstat] | [lmdiag] | [lmpath] | [lmtools] | [dslsstat] | [reporttool]**

Command Line Options

lmstat This option displays information about the location and features served by the FLEXnet Licensing servers used to serve the Abaqus license. Additional arguments may be used with this command to generate more license usage information.

lmdiag This option displays information relating to the various FLEXnet Licensing features and indicates whether or not the feature may be checked out.

lmpath This option can be used to control where Abaqus looks for licenses. Additional arguments are used to print, set, or add license location information. Running the command without arguments will display the command summary for each action.

lmtools This option starts the FLEXnet Licensing toolchest on Windows platforms. This application can be used to invoke most FLEXnet Licensing administration tool functions.

dslsstat This option displays information about the location and features served by the Dassault Systèmes license server (DSLS). For more information, see the following guide in the user assistance: *Installation, Licensing & Configuration | SIMULIA Licensing*.

reporttool This option is used to generate reports from license usage history. For more information, see the following guide in the user assistance: *Installation, Licensing & Configuration | SIMULIA Licensing*.

Querying the Keyword/Problem Database

Products: Abaqus/Standard Abaqus/Explicit

References:

- *About Execution Procedures*

Overview

The **abaqus findkeyword** utility queries a keyword/problem database that contains information on Abaqus/Standard and Abaqus/Explicit example problems, verification problems, problems used in training seminars, problems shown in the Abaqus technology briefs and benchmark timing problems. You specify which keywords, parameters, and values are of interest; and this utility will list the input files that contain those keywords, parameters, and values. You can specify multiple keywords, which causes the **findkeyword** utility to list those input files that contain all of the specified keywords. You can then use the **abaqus fetch** utility to fetch the input files (see *Fetching Sample Input Files*). The output is grouped into problem sets; e.g., Abaqus Example Problems or Abaqus/Standard Technology Brief Problems.

Command Summary

abaqus findkeyword **[job =** *job-name***]**
[maximum = *maximum-output***]**
keyword data lines

Command Line Options

job This option is used to specify the output file name for the output listing. If this option is omitted from the command line, the output will be printed to the standard output device.

maximum This option is used to limit the number of sample problems that are listed for each set. If this option is omitted, a maximum of 100 sample problems are listed for each set.

keyword data lines The *keyword data lines* specify which Abaqus keywords, parameters, and values are of interest to the user. The names of sample problems that contain the specified keywords, parameters, and values are printed to the standard output device or to the file indicated by the **job** command line parameter. The keyword is required, but parameters and values are optional. If a keyword is specified without a parameter or a value, all sample problems that use that keyword (with or without parameters and values) will be listed. If a parameter is specified without a value, all sample problems that use that parameter with any value will be listed. Parameter values that are user-specified data (e.g., numeric data, set names, orientation names, etc.) are ignored. The end of the keyword data lines is indicated by an empty line or an end of file.

Examples

The following examples illustrate the different types of search criteria utilized by the **findkeyword** execution procedure.

Querying for Keywords and Parameters

To list the sample problems that use the **RESTART* option with the WRITE parameter, type the following command and data lines:

```
abaqus findkeyword
*RESTART,WRITE
```

To generate a list of sample problems that contain two keyword lines in the same file, both keywords are included as data lines. For example,

```
abaqus findkeyword
*RESTART,WRITE
*NGEN
```

To list all sample problems that use a keyword and parameter with a value, the value must be included on the data line. For example,

```
abaqus findkeyword job=beam
*BEAM SECTION,SECTION=ARBITRARY
```

The output is written to the file `beam.dat`.

Querying for User-Specified Parameter Values

User-specified parameter values (e.g., numeric data, set names, orientation names, etc.) are ignored. The following two examples are equivalent because the value `MYSET` is an element set name.

```
abaqus findkeyword
*ELSET,ELSET=MYSET
```

```
abaqus findkeyword
*ELSET,ELSET
```

Fetching Sample Input Files

Products: Abaqus/Standard Abaqus/Explicit

References:

- *About Execution Procedures*

Overview

The **abaqus fetch** utility is used to extract sample Abaqus input files, user subroutine files, journal files, parametric study script files, or postprocessing programs from the compressed archive files provided with the release (for problems in the *Abaqus Example Problems Guide*, the *Abaqus Benchmarks Guide* Guide, and the *Abaqus Verification Guide* Guide). File names are specified in the guides. If no file extension is specified, all files corresponding to the name given will be extracted.

Wildcard expressions can be used when specifying the file names and include the following:

- An asterisk (*) matches a sequence of zero or more characters.
- A question mark (?) matches exactly one character.
- A bracketed item [...] matches any single character found inside the brackets; ranges are specified by a beginning character, a hyphen, and an ending character. If an exclamation point (!) or a caret (^) follow the left bracket, the range of characters within the brackets is complemented; that is, anything except the characters inside the brackets is considered a match.

Any character that might otherwise be interpreted or modified by the operating system, particularly on Linux platforms, should be placed inside quotation marks. If no matches are found using the wildcard expressions, the **abaqus fetch** utility attempts to extract a file with the name specified.

Command Summary

abaqus fetch	**job =** *job-name* **[input =** *input-file***]**

Command Line Options

job
This option is used to specify the output file name for the fetched input file or files. It is also the default name of the input file to fetch.

If this option is omitted from the command line, you will be prompted for this value.

input
This option is used to specify the name of the input file or files to fetch if it is different from the *job-name*.

Examples

To fetch the example input file `c2.inp` from the archive files, use the following command:

```
abaqus fetch job=c2.inp
```

To fetch all files associated with job `c8` from the archive files, do not specify a file extension. The following command will extract both the input file (`c8.inp`) and the user subroutine file (`c8.f`):

```
abaqus fetch job=c8
```

To fetch the sample parametric study scripting file `parstudy.psf` from the archive files, use the following command:

```
abaqus fetch
job=parstudy.psf
```

Reducing Analysis Database File Sizes for Restart

Products: Abaqus/Standard

References:

- *About Execution Procedures*

Overview

The **abaqus redistadb** utility efficiently extracts a single frame of restart data from existing analysis database files.

The analysis database for an Abaqus/Standard analysis consists of state (`.stt`) and model (`.mdl`) files and is required for continuation workflows such as restart, import, map solution, and symmetric results transfer. For analyses that have multiple restart frames, these files can get large in size. You can use this utility to create analysis database files (`.stt` and `.mdl` files) that contain only a selected restart frame. You can use these files to run continuation workflows.

The existing analysis database files can be in global form (`.stt` and `.mdl`) or domain form (`stt.0`, `stt.1`, etc. and `mdl.0`, `mdl.1`, etc.). In the case where domain-specific files are used, this utility creates analysis database files pertaining to the specified restart frame in global form (`.stt` and `.mdl`).

Command Summary

abaqus redistadb

oldjob = *oldjob-name*
[newjob = *newjob-name***]**
[input = *input-file-name***]**
[step = *step-number***]**
[increment = *increment-number***]**
[outdir = *outdir-name***]**
[copyfiles = `yes`**]**
[list = `yes`**]**
[help = `yes`**]**

Command Line Options

oldjob
This option is used to specify the name (without a file extension) of the existing analysis database that contains all of the `.stt` and `.mdl` files from a previous analysis.

newjob
This option is used to specify the name (without a file extension) of all the files generated during the redistribution. This name must be different from the **oldjob** name that you specify. If it is omitted, a default name is derived by adding the suffix `_RES` to the **oldjob** name.

input
This option is used to provide the name of an Abaqus input file that contains **RESTART*, READ or **IMPORT* so that the step and increment specified by either of these options can be extracted and used to generate the reduced analysis database.

step
This option is used to specify the step in the existing analysis database from which a frame will be extracted. In combination with the **increment** option, it allows you to specify an exact frame for extraction. It provides an alternate and direct method of reducing the analysis database that does not require an input file. Hence, this option is applicable only if the **input** option is omitted.

If the **step** option is specified but the **increment** option is omitted, the last saved increment of the specified step is extracted.

increment This option is used to specify an increment number in the existing analysis database that will be extracted. In combination with the **step** option, it allows you to specify an exact frame for extraction. It provides an alternate and direct method of reducing the analysis database that does not require an input file. Hence, this option is applicable only if the **input** option is omitted.

outdir This option is used to specify the directory where the previous analysis database files reside and the reduced analysis database files are to be created. The default directory is the working directory.

copyfiles This option is used to make a copy of the `.odb`, `.prt`, `.res`, `.sim`, and `.fil` (if it exists) files under the *newjob-name* or the derived name *oldjob-name*_`RES` from the previous analysis so that a subsequent analysis can run with the reduced analysis database.

You can use an alternate syntax of -**copyfiles** without the value.

list This option is used to list all the saved frames (step and increment) within the existing analysis database from a previous analysis.

You can use an alternate syntax of -**list** without the value.

help This option prints a summary of the **abaqus redistadb** command syntax.

You can use an alternate syntax of -**help** without the value.

Examples

You can run the stand-alone analysis database redistributor as follows:

```
abaqus redistadb oldjob=test_old input=contactdamp_restart copyfiles=yes
```

This command generates the reduced Abaqus analysis database files (`.mdl` and `.stt`) under a default name with the persistent data extracted from the previous analysis database at the frame specified by **RESTART*, READ in the given input file. The command also copies the `.odb`, `.prt`, `.res`, `.sim`, and `.fil` (if it exists) files from the previous analysis.

You can then run the subsequent analysis with the reduced analysis database using the following command:

```
abaqus job=test_res input=contactdamp_restart oldjob=test_old_RES cpus=4
threads_per_mpi_process=2
```

Output Database Upgrade Utility

Products: Abaqus/Standard Abaqus/Explicit

References:

- *About Execution Procedures*

Overview

The **abaqus upgrade** utility will convert an output database file from earlier releases of Abaqus to the current release. Output database files from Abaqus 6.1 or later can be upgraded. The **abaqus upgrade** utility will generate a log file (*job-name*`.log`) that contains error, warning, diagnostic, and informational messages. You should carefully review the conversion log file to ensure that changes made to the older release output database file are appropriate. If no conversions are necessary, a message will be issued to the log file as well as to the screen.

Command Summary

abaqus upgrade **job =** *job-name*
odb = *old-odb-file-name*

Command Line Options

Required Options

job This option is used to specify the name of the upgraded output database file to be output by the utility.

odb This option is used to specify the name of the output database file to be upgraded.

SIM Database Utilities

Products: Abaqus/Standard Abaqus/Explicit

References:

- *About Execution Procedures*
- *Abaqus/Standard and Abaqus/Explicit Execution*

Overview

The **sim_version** utility converts a SIM database file from one release to another, to query a SIM database file for its SIM release level, or to determine the SIM release level of the current code you are using.

Command Summary

abaqus sim_version {**convert =** *old-sim-file-name* | **query =** *sim-file-name* | **-current**}
[out = *new-sim-file-name***]**
[level = *release-level***]**
[help]

Command Line Options

convert This option is used to specify the name of the SIM database file to be converted.

query This option is used to specify the name of the SIM database file to query for its SIM release level.

current This option is used to determine the SIM release level of the current code you are using.

out This option is used to specify the name of the SIM database file to be output by the utility. This option is used in conjunction with the **convert** option.

level This option is used to specify the release level name to which the SIM database file is to be converted.

If this option is omitted from the command line, the SIM database file is converted to the current release level. Release level names are case insensitive. This option is used in conjunction with the **convert** option.

help This option prints a summary of the **abaqus sim_version** command syntax.

Examples

Use the following command to query a SIM database file for the release level number:

```
abaqus sim_version query=sim-file-name
```

Use the following command to determine the SIM level corresponding to the current Abaqus release:

```
abaqus sim_version -current
```

Downgrading SIM Database Files

You can use the **level** option to downgrade SIM database files for use with previous releases of the **3D**EXPERIENCEPhysics Results Explorerapp.

For example, use the following command to downgrade SIM database files from the current release for use with the **3D**EXPERIENCE R2016x Physics Results Explorerapp:

```
abaqus sim_version convert=sim-file-name out=downgraded-sim-file-name
   level=R2016x
```

Translating ODB Output Database Files to SIM Format

Products: Abaqus/Standard Abaqus/Explicit

References:

- *About Execution Procedures*
- *The Output Database*

Overview

The translator converts an output database (`.odb`) file containing model and results information to a SIM database (`.sim`) file, allowing you to use the Physics Results Explorerapp on the **3D**EXPERIENCE platform for high-performance postprocessing of analysis results.

Errors are issued when specific data in the output database cannot be converted to the SIM database file. For example, if the output database does not have sufficient information to translate the data properly, or if support is not yet implemented for particular output in the SIM format. The translator will not stop if these types of errors are issued; however, it will stop (without saving the SIM database file) if a fatal error occurs.

Limitations

The translator is subject to the following limitations:

- All data in the output database file that are needed by the Physics Results Explorerapp are copied to the SIM database file. Other data may not be copied to the SIM file.
- Field and history results for the following output variables are not converted:
  - MMIXDME
  - MMIXDMI
  - NFORC
  - SSPEEQ
  - SSSPRD
  - SSTORQ
  - SSFORC
  - TIME
- Data for the following elements are not converted:
  - ELBOW31C
  - Dashpot elements
  - Spring elements
  - Substructure elements
  - User elements
- Coupling constraints are not converted.

Command Summary

abaqus odb2sim **odb =** *odb-file-name*
sim = *SIM-database-file-name*
[log = *log-file-name***]**
[o2sdebug]

Command Line Options

odb This option is used to specify the name of the output database file to be converted.

sim This option is used to specify the name of the SIM database file to be output by the translator.

log This option is used to specify the name of the log file generated during the translation. If this option is omitted, the information is written to the screen.

o2sdebug This option specifies the level of detail for the messages that Abaqus writes to the log file. Valid settings are as follows:

- `0` - errors only (default)
- `1` - warnings and work-in-progress milestones (e.g., convert sections)
- `2` - more debug information (e.g., field data)

Generating Output Database Reports

Products: Abaqus/Standard Abaqus/Explicit

References:

- *About Execution Procedures*
- *Object model for the output database*

Overview

The output database report utility prints information from an Abaqus output database (`.odb`) file to a formatted report. By default, the report is printed in plain text format; however, you can also create reports in HTML and CSV (comma-separated values) formats.

Output Database Structure

Every output database consists of two main sections: model data and results data. The database is further broken down into a hierarchical structure of containers, as indicated in *Figure 1*.

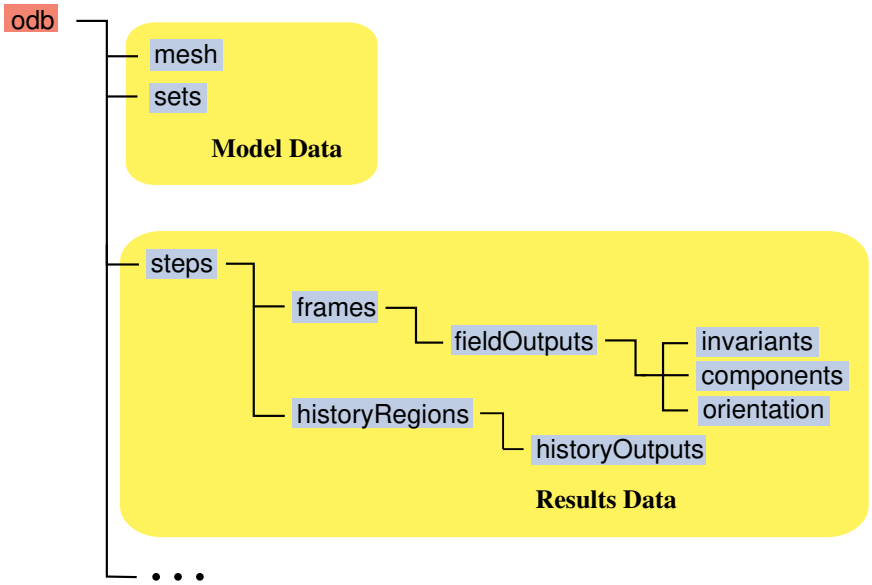


Figure 1: Structure of an output database

The data that can appear in a report reside in the containers at the far right of each branch. These containers can be used to classify the four main branches of the output database:

- The **mesh** branch terminates in a container holding nodal coordinates and element connectivity information for the model.
- The **sets** branch terminates in a container holding the names and node or element labels of the sets and surfaces in the model.
- The **fieldOutputs** branch terminates in a container holding the values of field output variables from the analysis. These values are further broken down into their vector or tensor attributes: **invariants**, **components**, and **orientation**.

- The **historyOutputs** branch terminates in a container holding the values of history output variables from the analysis.

The containers in the model data section of the tree are singular containers: each model has one container for mesh information and one container for sets information. The containers in the results section of the tree, however, represent aggregates of multiple containers. For a multistep analysis, the output database will have a separate **step** container for each step of the analysis. Within each **step** container will be multiple **frames** and **historyRegions** containers. Within each individual **frames** container will be multiple **fieldOutputs** containers, and so on. The output database assigns names or values to these individual containers to help distinguish and identify them.

For a more detailed discussion of the output database structure, see *Object model for the output database*.

Generating Summary Reports

If you generate a report using only the required and file formatting command line options, the report will be a brief summary of the output database. This summary contains a listing of the following information:

- Part instance names
- Number of nodes and elements in the model
- Names of sets and surfaces
- Names of steps and load cases
- Numbers of frames in the steps
- Names of field and history output variables

The information contained in this summary can help you determine the names and values of containers in the output database.

Adding Information to a Report

You can create more comprehensive reports using additional command line options. Most of these options correspond to a container in the output database structure outlined in *Figure 1*. Using these options to specify the name or value of a container instructs the utility to extract the data found in that container and to add it to the generated report. Container names and values are not always unique, and may appear more than once in an output database. For example, a container corresponding to frame 1 will likely appear in every individual step container for a multistep analysis; similarly, a container holding a specific field output variable usually appears inside every frame of the step. The utility will add all instances of these containers to the report.

To refine the container selection, you can combine options. When more than one container from the same branch is indicated on the command line, the utility only reports the data that are common to both containers. For example, if two options specify the container for Step 1 and the container for frame 3, the utility will add results data only from the third frame of the first step to the report. If you specify containers from different branches, the data from each container are added to the report. For example, if the two options specify the sets container and a history region container, both sets data and history output data are added to the report.

You identify specific containers by setting the associated option equal to the name or value of that container. To include multiple containers of the same type, set the option equal to a comma-separated list. The names are case-sensitive. If the names include spaces, you must enclose the entire value in double quotation marks (`"container name"`).

Additional Options

The output database report utility offers some additional options for controlling the organization and details of a report. These options will have no effect unless they are invoked in conjunction with other “container” options.

Command summary

abaqus odbreport **[job =** *job-name*]
[odb = *output-database-file*]
[mode = {`HTML` | `CSV`**}]**
[all]
[mesh]
[sets]
[results]
[step = {*step-name* | `_LAST_`**}]**
[frame = {*number* | *load-case-name* | *description* | `_LAST_`**}]**
[framevalue = {*time* | *mode* | *frequency*}]**
[field [= *field-variable* **]]**
[components]
[invariants]
[orientation]
[histregion = *region-name*]
[history [= *history-variable* **]]**
[instance = {*instance-name* | `_NONE_`**}]**
[blocked]
[extrema]

Command line options

Required Options

You must include at least one of the following options when executing `abaqus odbreport`. They tell the utility where to find the output database and where to print the report. Use both options together to make the report's file name unique from the output database name.

job This option is used to specify the file name of the generated report. If you omit this option, the utility prints the report to the standard output device.

odb This option is used to specify the output database (`.odb`) file from which the report is generated. If you omit this option, the utility looks for an output database called *job-name*`.odb` in the current directory.

File Formatting Option

mode This option specifies the file format of the generated report. If you omit this option, the report is in plain text format with the file extension `.rep`. If **mode=**`HTML`, the report is in HTML format with the file extension `.htm`. If **mode=**`CSV`, the report is in comma-separated values format with the file extension `.csv`.

Option to Generate a Full Output Database Report

all This option is used to report all available model information and results information from every step in the analysis; data from the base state of each step (frame zero) is not included in the report. The report will be very long for large output databases.

Options to Report Model Data

The following options extract information from the model data section of the output database.

mesh This option is used to report the nodal coordinates and element connectivity associated with the model's mesh.

sets This option is used to report the names and contents of all sets and surfaces associated with the model.

Options to Report Results Data

The following options extract information from the results data section of the output database.

results This option is used to report all field and history output variable values from the output database. If you include any other options corresponding to specific results containers, this option is ignored.

step This option is used to report the field and history output variable values for the specified steps. When invoking this option, you must set it equal to at least one step name. If **step=**`_LAST_`, the report includes results from only the last step of the analysis.

The **steps** container is common to both the **fieldOutputs** and **historyOutputs** branches of the output database. If you combine the **step** option with a field output variable option, only field output variable data appear in the report. Similarly, if you combine the **step** option with a history output variable option, only history output variable data appear in the report. If you combine the **step** option with both field and history output variable options, both types of variable data appear in the report.

Options to Report Field Output Variables

The following options extract information from containers in the **fieldOutputs** branch of the output database.

frame This option is used to report field output variable values for the specified frames. When invoking this option, you must set it equal to at least one frame number, load case name, or frame description. The initial (or "zero increment") frame can be identified only by setting **frame=**`0`. If **frame=**`_LAST_`, the report includes results from only the last frame of each included step.

framevalue This option is used to report field output variable values for the specified frame values. Each frame can be identified by a frame value that may be unique from the frame number. The frame value is either the time, eigenmode number, or frequency point associated with a frame.

This option can be used as an alternative or complement to the **frame** option. When invoking this option, you must set it equal to at least one frame value. The values you provide do not need to be exact; the utility will find the frame with the closest frame value.

field This option is used to report the specified field output variable values. If you invoke this option without setting it equal to any variable names, all field variable containers are included in the report.

Options to Report Different Field Variable Attributes

If none of the following options is invoked, the utility automatically reports components and (if applicable) orientations for each field variable. Otherwise, the utility reports only the attributes specified by these options. These options will have an effect only if used in conjunction with other field output variable options. Invariants and orientations are not available for all field variables.

components This option is used to report components for all field output variables.

invariants This option is used to report invariant values for all field output variables.

orientation This option is used to report the local coordinate system for each field output variable.

Options to Report History Output Variables

The following options extract information from containers in the **historyOutputs** branch of the output database.

histregion This option is used to report history output variable values for the specified history region. When invoking this option, you must set it equal to at least one history region name.

history This option is used to report the specified history output variable values. If you invoke this option without setting it equal to any variable names, all history variable containers are included in the report.

Additional Options

The following options add an additional level of control and detail to a report. They are not associated directly with the output database structure and will not add database information to a report. They must be used in conjunction with the previously described options.

instance This option is used to limit reported model and results data to a specific part or assembly instance in the model. It is not directly associated with any output database containers and will not add any data to a report.

When invoking this option, you must set it equal to at least one instance name. If **instance=**`_NONE_`, the report includes data for the whole assembly and model.

blocked This option is used to subdivide tables of field output variables into blocks according to part instance, element type, and section point. It is useful if you are interested in separating output from different areas of a large model. By default, the tables are organized according to variable name and frame.

This option instructs the report utility to access the output database using the field bulk data API. For details about how the field bulk data API operates, see *Using bulk data access to an output database*. An additional benefit of this option is enhanced performance of the utility when dealing with large volumes of field variables, leading to faster report generation. The option has no effect if there are no field output variables in a report, or when the **invariants** option is also specified.

extrema This option is used to report maximum and minimum values at the end of each table of nodal coordinates and field output variables. By default, these extrema do not appear in a report. The option will have no effect if there are no nodal coordinates or field output variables in a report.

Examples

The following examples illustrate the capabilities of the **odbreport** execution procedure and the effects of different option combinations.

File Naming and Formatting

The following command generates a brief summary of the output database `beam.odb` in a plain text file named `beam.rep`:

```
abaqus odbreport job=beam
```

To create the same report in HTML format and with the name `beamreport.htm`, execute the following command:

```
abaqus odbreport job=beamreport odb=beam mode=html
```

Adding Information to a Report

Use additional command line options to add data from specified containers to a report. The following command creates a report listing nodal coordinates and element connectivity from the model and all output variable values associated with the step named **Apply weight**:

```
abaqus odbreport job=beam mesh step="Apply weight"
```

You can refine the results data listed by using combinations of options. In the following example, the utility reports only history output variable values that were output from the history region named **Node350** in the **Apply weight** step:

```
abaqus odbreport job=beam step="Apply weight"
   histregion=Node350
```

If a container is identified by a name or value that is not unique, the generated report will include all occurrences of that container. The following command creates a report listing the values for field variable RF that were output in the third frame of every individual step:

```
abaqus odbreport job=beam frame=3 field=RF
```

To report the magnitude of RF instead of its components, use the **invariants** option:

```
abaqus odbreport job=beam frame=3 field=RF invariants
```

To add multiple containers of the same type to a report, you can set an option equal to a comma-separated list. The following command reports all values of field output variables U and S that were output during the steps **Apply weight** and **Side load**:

```
abaqus odbreport job=beam step="Apply weight","Side load"
   field=U,S
```

Additional Options

Use the **instance** option to limit reported information to a particular section of your model. The following command reports set names and nodes, and values of S in the last frame of every step from the database `motor.odb`. However, only information related to part instance **pistonA** appears in the report:

```
abaqus odbreport job=motor sets frame=_LAST_ field=S
   instance=pistonA
```

Selecting Frames

The **frame** and **framevalue** options can accept a wide variety of value types, making them powerful report-building options. Because of this variety, it is sometimes necessary to invoke both options to specify a particular frame. For example, consider the output database `plate.odb`, the results of a steady-state dynamic analysis. The analysis investigated the response of a plate over a range of 20 different frequencies under three different load cases. The output database, therefore, includes results for the three different load cases at each frequency. You are interested in the response at 45 Hz under the load case named **lc2**. Setting **frame=**`lc2` will report field variables for load case **lc2** at every frequency (a total of 20 frames). Setting **framevalue=**`45` will report field variables for every load case associated with the 45 Hz frequency (a total of three frames). To limit the report to the single frame of interest, you must invoke both options together:

```
abaqus odbreport job=plate frame=lc2 framevalue=45
```

Joining Output Database (.odb) Files from Restarted Analyses

Products: Abaqus/Standard Abaqus/Explicit

References:

- *About Execution Procedures*
- *Continuation of Output on Restart*

Overview

The **abaqus restartjoin** utility appends an output database (.odb) file produced by a restart analysis of a model to the output database produced by the original analysis of that model. Combining the original and restart output database files into a single file enables you to examine all of the output data for the analysis in Abaqus/CAE.

A similar utility, **abaqus append**, is used to join results (.fil) files. See *Joining Results (.fil) Files* for details.

Appending Data When the Analysis Restarts between Steps Versus Midstep

You can append output database files from analyses that restart between steps and from analyses that restart in the middle of a step. While the required syntax is the same for these two types of analyses, Abaqus appends data differently, as follows:

- For an analysis that stops and restarts between steps, Abaqus simply appends the output from the new steps to the output from the existing steps of the original analysis.
- For an analysis that stops and restarts in the middle of a step, the original and restart analyses overlap because the restart analysis resumes at the beginning of the interrupted step. In this case the **abaqus restartjoin** utility retains the results for any completed steps in the original analysis but replaces the results for the interrupted step with the output data produced by the restart analysis.

Customizing the Combined Output Database File

By default, Abaqus appends the output data produced by the restart analysis directly to the original output database file. If you prefer to retain the original output database file, you can create a copy of it and append the restart analysis output data to the copy instead. Abaqus names this copy using the format Restart_*original-odb-filename*; for example, a copy of the original output database file job-1.odb would be named Restart_job-1.odb.

Abaqus omits history data when you combine original and restart output databases; however, you can override this default. You can also control whether Abaqus compresses the combined output database file.

Command Summary

abaqus restartjoin
originalodb = *odb-file-name*
restartodb = *odb-file-name*
[copyoriginal]
[history]
[compressresult]

Command Line Options

originalodb This option specifies the output database file produced by the original analysis. If you omit the **copyoriginal** option, Abaqus appends the output data from the restart output database file directly to the original output database file.

If you omit this option from the command line, Abaqus will prompt you for its value.

restartodb This option specifies the output database file produced by the restart analysis. You can specify only one restart analysis output database file at a time.

If you omit this option from the command line, Abaqus will prompt you for its value.

copyoriginal If this option is specified, Abaqus creates a copy of the output database file specified by the **originalodb** option and appends the contents of the **restartodb** output database file to that copy instead of to the original file. When this option is omitted, Abaqus appends the output data from the restart analysis directly to the original output database file.

Abaqus names the copied output database file by adding the prefix `Restart_` to the name of the original output database file; for example, a copy of the original output database file `original.odb` would be named `Restart_original.odb`.

history If this option is specified, Abaqus copies history data from the restart output database to the original output database or its copy. Abaqus omits history data in the joined output database file unless you specify this option.

compressresult If this option is specified, Abaqus compresses the resulting output database file.

Examples

If your model produced an initial output database file named `Job-1.odb` and a restart output database file named `Job-1_res.odb`, issue the following command to append the contents of the restart database to the initial output database file:

```
abaqus restartjoin originalodb=Job-1.odb restartodb=Job-1_res.odb
```

If you prefer to retain the original output database file, you can create a copy of this original file and append the contents of the restart output database file to the copy instead. Abaqus creates the name of the copied output database file by adding the prefix `Restart_` to the name of the original file; in the preceding example the copy of the original file `Job-1.odb` would be named `Restart_Job-1.odb`. To perform the restart join operation using a copy of the original file, issue the following command:

```
abaqus restartjoin originalodb=Job-1.odb restartodb=Job-1_res.odb
copyoriginal
```

By default, Abaqus does not copy history data to the combined output database. To include history data, issue the following command:

```
abaqus restartjoin originalodb=Job-1.odb restartodb=Job-1_res.odb history
```

Combining Output from Substructures

Products: Abaqus/Standard Abaqus/Explicit

References:

- *About Execution Procedures*
- *Obtaining Output of Results within a Substructure*
- *Combining data from multiple output databases*
- *Using Abaqus/Standard Output Variable Identifiers*

Overview

The **abaqus substructurecombine** utility combines the model and results data produced by two of a model's substructures into a single output database (`.odb`) file. By combining all of a model's substructure analysis output database files, you can display all of the data produced by a substructure analysis in Abaqus/CAE.



Note: Using the **abaqus odbcombine** utility is the preferred method for combining the substructure output database (`.odb`) files.

Abaqus combines output data by adding the contents of the second file you specify (the copy output database) directly into the first file you specify (the base output database). Because this process changes the base output database, consider backing up your data before using this utility.

Combining Data for Models with More than Two Substructures

Because the **abaqus substructurecombine** utility combines data from only two output databases at a time, you must run the utility multiple times to create a single output database from an analysis with more than two substructures. Combine data from two of the substructures first, then repeat the operation to combine the resulting output database file with data from each remaining substructure.

Customizing the Combined Output Database

You can customize the substructure combine operation by adding only a subset of the data from the copy output database into the base output database. Abaqus enables you to add output data to the base output database from a single step or frame in the copy output database. You can also include only output data from the copy output database that relates to a particular variable; for example, you can copy output data related to Mises stress.

Command Summary

abaqus substructurecombine **baseodb =** *odb-file-name*
copyodb = *odb-file-name*
[all]
[step = *step-name***]**
[frame = *frame-number***]**
[variable = *variable-key***]**

Command Line Options

baseodb This option specifies the name of the base output database, to which Abaqus adds the contents of the copy output database.

If you omit this option from the command line, Abaqus will prompt you for its value.

copyodb This option specifies the name of the copy output database, which Abaqus adds to the contents of the base output database. You can specify only one file at a time for this option.

If you omit this option from the command line, Abaqus will prompt you for its value.

all This option indicates that data for all variables within all steps and frames of output should be copied to the combined output database. When you specify this option, Abaqus ignores the **step**, **frame**, and **variable** options.

step This option indicates the name of the step from which Abaqus will copy results data. You can specify only one step; if you omit this option, Abaqus copies data from the last step in the output database.

Abaqus ignores this option if you specify the **all** option.

frame This option indicates the number of the frame from which Abaqus will copy results data. You can specify only one frame; if you omit this option, Abaqus uses the last frame in the step specified by the **step** option.

Abaqus ignores this option if you specify the **all** option.

variable This option indicates the variable key for the variable from which Abaqus will copy results data. If you omit this option, Abaqus copies data for all variables in the output database. Abaqus ignores this option if you specify the **all** option.

Only output variable keys that are valid for output database file output are available for use with **abaqus substructurecombine**. In general, if a key corresponds to a collective output variable, rather than an individual component, it can be used with this execution procedure. The collective output variable keys are distinguished from their individual components by the fact that they have the word "yes" after one of the **.odb** categories in the tables in *Using Abaqus/Standard Output Variable Identifiers*.

Examples

The following examples illustrate different methods of combining substructures using the **abaqus substructurecombine** execution procedure.

Combining Two Substructures

If your model contains two substructures that produce output database files named `subst1.odb` and `subst2.odb`, issue the following command to overwrite `subst1.odb` with the combined contents of the two files:

```
abaqus substructureCombine baseodb=subst1.odb copyodb=subst2.odb
```

Combining More than Two Substructures

If your model contains more than two substructures, you must first combine the output database files from two of the substructures, then combine the combined output database with each of the other substructures' output

databases in turn. In this example the substructure analysis produces four output database files named `subst1.odb`, `subst2.odb`, `subst3.odb` and `subst4.odb`, so you must issue the **abaqus substructure** command a total of three times to combine all four files into a single output database, as shown in the following example:

```
abaqus substructureCombine baseodb=subst1.odb copyodb=subst2.odb
abaqus substructureCombine baseodb=subst1.odb copyodb=subst3.odb
abaqus substructureCombine baseodb=subst1.odb copyodb=subst4.odb
```

Combining Specific Elements of the Substructures

If you want to include only the output data from the step **Step-1** in the combined output database, issue the following command:

```
abaqus substructureCombine baseodb=subst1.odb copyodb=subst2.odb
   step="Step-1"
```

If you want to include only the output data from the **Mises** variable in the combined output database, issue the following command:

```
abaqus substructureCombine baseodb=subst1.odb copyodb=subst2.odb

variable="Mises"
```

Recovering Results from a Substructure

Products: Abaqus/Standard

References:

- *Generating Substructures*
- *Using Substructures*

Overview

The **abaqus substructurerecover** utility recovers results within a substructure using the modal recovery matrices.

Results recovery is performed for static or dynamic time-domain or frequency-domain solutions obtained from Abaqus/Standard and from the following workflows:

- Abaqus-Simpack flexible body dynamics
- Abaqus-ADAMS flexible body dynamics
- Abaqus-EXCITE flexible body dynamics

Using the Utility

The following procedure summarizes the typical usage of the **abaqus substructurerecover** utility:

1. Run an Abaqus/Standard substructure generation analysis to generate the following items:
   - Substructure
   - Flexible body
   - Recovery matrices for the nodal and element results
   - Model data ODB file
2. Do one of the following:
   - Run a flexible body dynamic analysis using Simpack, ADAMS, or EXCITE to obtain a solution.
   - Run an Abaqus/Standard transient dynamic analysis using a substructure, and output the substructure generalized displacements to a file (see *Obtaining Output of the Solution for All of the Substructure Degrees of Freedom*).
3. Run the **abaqus substructurerecover** utility to recover results within the substructure, specifying the substructure name that defines the substructure SIM and model data ODB files and specifying the file that contains the solution for all the substructure degrees of freedom.

Command Summary

abaqus substructurerecover **job =** *job-name*
substructure = *substructure-name*
[input = *input-file-name***]**
[format = {`OP4` | `INP4` | `MDF` | `ASC` | `SIM` | `FBR`**}]**
[eigenproblem]
[minframevalue = *value***]**
[maxframevalue = *value***]**
[resultsformat = {`ODB` | `SIM` | `BOTH`**}]**
[output = *variables-list***]**

[nset = *node-set-names*]
[elset = *element-set-names*]
[body = *body-name*]

Command Line Options

job This option specifies the name of the output database file: *job-name*`.odb`. It also defines the *input-file-name* if the **input** option is not specified.

substructure This option specifies the name of the substructure ("Z-name") that defines the substructure SIM and model data ODB files.

input This option specifies the name of the input file that contains the solution for all the substructure degrees of freedom. If the option is omitted, the default value is *job-name*. The contents and format of the solution input file are specific to the product used to obtain the solution.

The solution input file name can include one of the following supported file extensions: `.op4`, `.inp4`, `.mdf`, `.asc`, `.sim`, and `.fbr`.

format This option specifies the format of the solution input file. The following formats are supported: `OP4`, `INP4`, `MDF`, `ASC`, `SIM`, and `FBR`. If this option is omitted, the format is determined by the extension of the *input-file-name*. If the **input** option is omitted, the default format is `SIM`.

eigenproblem This option specifies recovering the eigenmodes of an unsupported substructure. This calculation does not require input of the dynamic solution. This option is mutually exclusive with the **input** and **format** options.

minframevalue This option specifies the lower boundary of the time or frequency range for results recovery. By default, the minimal available time or frequency value is used.

maxframevalue This option specifies the upper boundary of the time or frequency range for results recovery. By default, the maximal available time or frequency value is used.

resultsformat This option specifies the results output format: `ODB`, `SIM`, or `BOTH`. The default value is `ODB`.

output This option specifies a list of output variables to recover. Variables in the list are separated by commas and/or spaces. If spaces are used, the list must be enclosed in quotes. This option is supported only for **resultsformat**=`ODB`.Supported output requests:

- `U` (displacement)
- `V` (velocity)
- `A` (acceleration)
- `S` (stress)
- `E` (strain)
- `MISES` (Mises stress)
- `SP` (principal stress)
- `EP` (principal strain)
- `SPAM` (principal absolute maximum stress)
- `EPAM` (principal absolute maximum strain)

Principal stresses and strains and Mises stress are applicable only to element recovery averaged at nodes in the Abaqus substructure generation.

nset This option specifies the node set names for results output. The node sets are defined in the Abaqus substructure generation model. Multiple node set names must be separated by commas. By default, results are recovered at all nodes associated with recovery matrices.

elset This option specifies the element set names for results output. The element sets are defined in the Abaqus substructure generation model. Multiple element set names must be separated by commas. By default, results are recovered at all elements associated with recovery matrices.

body This option specifies which flexible body solution to use for recovery. This option is supported only for **format=**`FBR`. By default, results are recovered for the first flexible body solution stored in the solution input file.

Examples

The following examples illustrate different methods of recovering results from a substructure using the **abaqus substructurerecover** utility.

Recovering Results within a Substructure in an Abaqus-Simpack Flexible Body Dynamics Workflow

In an Abaqus-Simpack flexible body dynamics workflow, you might have the substructure generalized solution from time-domain or frequency-domain analyses in a binary file with `.fbr` extension.

Use the following command to recover results within a substructure in an Abaqus-Simpack flexible body dynamics workflow:

```
abaqus substructurerecover job=recover2 substructure=substr_Z1
input=solutions.fbr nset=NSET1,NSET2 body=BODY1
```

Results are recovered with solutions of the flexible body `BODY1` and for node sets `NSET1` and `NSET2` defined in the substructure generation finite element model. The recovered results are saved in a file named `recover2.odb`.

For time-domain analyses you can also have the substructure generalized solution in a text file with a `.txt` extension. In this case, you can use the `.txt` file as input; however, you must use **format=**`ASC` as shown in the example below:

```
abaqus substructurerecover job=recover2 substructure=substr_Z1
input=solutions.txt
format=ASC nset=NSET1,NSET2
```

Alternatively, you can rename the `solutions.txt` file to `solutions.asc` and omit the **format** argument.

Recovering Results within a Substructure in an Abaqus-ADAMS Flexible Body Dynamics Workflow

Use the following command to recover results within a substructure in an Abaqus-ADAMS flexible body dynamics workflow:

```
abaqus substructurerecover job=recover1 substructure=substr_Z1
input=solutions.mdf
minframevalue=0.001 maxframevalue=0.01
```

Results are recovered in the time range from 0.001 to 0.01. The recovered results are saved in a file named `recover1.odb`.

Recovering Results within a Substructure in an Abaqus-EXCITE Flexible Body Dynamics Workflow

Use the following command to recover results within a substructure in an Abaqus-EXCITE flexible body dynamics workflow:

```
abaqus substructurerecover job=recover3 substructure=substr_Z1
input=solutions.inp4
output=U,MISES,SPAM
```

Results are recovered only for output variables `U`, `MISES`, and `SPAM`. The recovered results are saved in a file named `recover3.odb`.

Combining Data from Multiple Output Databases

Products: Abaqus/Standard Abaqus/Explicit

References:

- *About Execution Procedures*
- *Joining Output Database (.odb) Files from Restarted Analyses*
- *Combining data from multiple output databases*

Overview

The **abaqus odbcombine** utility combines the results data in two or more Abaqus output database files (`.odb`) into a single output database (`.odb`) file. The **abaqus odbcombine** utility is intended for the combination of output databases containing different results. If you want to combine output databases from the same analysis before and after a restart, use the **abaqus restartjoin** execution procedure instead. For more information, see *Joining Output Database (.odb) Files from Restarted Analyses*.

Abaqus includes all model data from the selected output databases in the combined output database; however, for results data you can choose to include a subset of the data from the output databases that you specify. Abaqus/CAE determines which results data are included in the combined output database based on two factors: the filtering options you specify and your selection of primary output database.

Filters

You can filter the data that the utility includes in the combined output database to include results only from selected steps or frames, from selected output variables, or from a combination of these options. For example, a filter can enable you to include results data only from the last step and the last frame of the specified output databases, and the same filter can dictate that only Mises stress results are included in the combined output database. You can also establish multiple filters if you want to set up different filtering conditions for the first step than in the second step.

The **abaqus odbcombine** utility also provides two levels of filtering: output database–specific filters, which filter results from only a single output database; and default filters, which apply to the entire job. The output database–specific filters take precedence over the default filters, so Abaqus/CAE employs the settings in the default filters only when the default filter you define does not conflict with filters for one of the individual output databases.

The filtering syntax is flexible enough to allow you to specify multiple steps, frame, or output variable values. You can specify multiple step names in a comma-separated list, such as `Step-1, Step-2, Step-4`. For frames you can include ranges or individual values; for example, entering `1, 3, 5, 7:9` returns frames 1, 3, 5, 7, 8, and 9 to the combined output database.

You can also use the symbolic constants `'ALL'`, `'FIRST'`, and `'LAST'` as shortcuts to specify the data you want to include. These options enable you to include results data from all steps or frames and data from all output variables rather than one or more selected variables.

Primary Output Database

One output database in every combine operation is designated as the primary output database. The utility first transfers all field output data, subject to filtering selections, from the primary output database to the combined output database. The utility then locates results data from matching steps and frames in the subsequent output databases and copies only those data into the combined output database. This strategy provides a more coherent structure for the combined results data.

Configuration File Usage

The **abaqus odbcombine** utility uses data in configuration files to determine which output databases to combine, the file to designate as the primary output database, and the filtering options to enforce by default and for each output database. The configuration file must be in `.xml` format, and it can have three types of elements in the following order:

- The `<DefaultFilters>` element specifies one or more default filtering definitions. This section is optional, but you must include it if you want to set up default filtering for your combine operation. If this section is omitted, the default value used for Steps is `'LAST'`, for Frames is `'LAST'`, and for Vars is `'ALL'` (that is, results of all variables from the last frame of the last step are included).
- The `<PrimaryOdb>` element specifies the location of the primary output database and, if desired, one or more filtering definitions for the data in that output database. This section is required.
- One `<Odb>` element is required for each additional output database that you want to include in the combine operation.

You can then specify default filters for output database–specific filters by embedding `<Filter>` elements within the `<DefaultFilters>` element or within one of the output database elements.

Configuration File Template

The following example illustrates the structure of the configuration file for the **abaqus odbcombine** utility.

```
<?xml version='1.0' encoding='UTF-8'?>
Your XML file declaration may differ from this one.
<OdbInput>
    <DefaultFilters>
    The default filtering element is optional. If you include this element in the configuration file,
    you must include at least one <Filter> element within this section. Filter elements can
    use the Steps or Frames attributes to refer to symbolic constants or the StepName or
    FrameIndex attributes to refer to individual steps or frames, as shown in the following examples:
        <Filter Steps='step names or symbolic constants'
            Frames='frame numbers or symbolic constants'
            Vars='variable or symbolic constants' />
        <Filter StepName='full step name'
            FrameIndex='individual frame number'
            VariableName='variable' />
    </DefaultFilters>
    <PrimaryOdb Name='path to primary output database'>
    Filtering elements for the primary output database are optional. If you want to filter
    the data from this output database, include a <Filter> element within this section
    for each filtering option you want to define.
    </PrimaryOdb>
    <Odb Name='path to output database'>
    Filtering elements for the output database are optional. If you want to filter
    the data from this output database, include a <Filter> element within this section
    for each filtering option you want to define.
    </Odb>
    Append an <Odb> element for each additional output database you want to include.
</OdbInput>
```

Data Not Included in Combined Output Databases

The following types of output data are not included when you combine output database files:

- History output.
- Surface data.
- Data from analytical rigid part instances.
- Local coordinate systems associated with field output data.

Combining Output Results for Frequency Partitions

When you run a steady-state dynamic analysis, you can use the frequency split method to combine the output files for different frequency partitions. You invoke this method using the **ssd_split** and **ssd_partition** parameters on the Abaqus execution procedure. All types of output data supported for steady-state dynamic analyses are included when you combine output database files for multiple frequency partitions.

Command Summary

abaqus odbcombine **job =** *job-name*
[input = *configuration-file-name***]**
[verbose = *level***]**
[frequencyrange = *consecutive-list-of-odb-files***]**

Command Line Options

job This option specifies the name of the resulting combined output database and the name of the log file. Abaqus also searches for a configuration file by this name.

If you omit this option from the command line, Abaqus will prompt you for its value.

input This option specifies the name of the configuration file that specifies the output databases you want to combine and the steps, frames, and output variables to be included in the combination. The configuration file must be in `.xml` format.

verbose This option specifies the level of detail for the messages that Abaqus writes to the log file. Possible values are `1` or `2`. If you specify `1`, Abaqus writes only errors and warnings to the log file; if you specify `2`, Abaqus also records the filtering options you select and lists the model data and field output data that were successfully copied to the combined output database.

frequencyrange This option is exclusively used when steady-state dynamic analysis is executed with parameters **ssd_split** and **ssd_partition**. A list of output databases is specified as *job-name1.odbjob-name2.odbjob-name3.odb ...job-nameN.odb*. The output databases should contain results for different frequency partitions of the same steady-state dynamic analysis, and the files should be specified in the ascending order of the frequency partition numbers. The combined results are stored on *job-name1.odb*

Network Output Database File Connector

Products: Abaqus/CAE Abaqus/Viewer

References:

- *About Execution Procedures*
- *Accessing an output database on a remote computer*

Overview

A network ODB connector creates a connection to a network ODB server that can be used to access a remote output database. The **abaqus networkDBConnector** command is used to start the network ODB server. A network ODB connector can be created from any platform—Windows or Linux; however, the network ODB server must reside on a Linux platform.

Abaqus uses password files to authenticate the connection between the client and the server. The password on the network ODB server must be stored in a file called `.abaqus_net_passwd` in your home directory on the remote system. You must update this file after 30 days, and the password must be at least 8 characters long.

In addition, your home directory on the local client machine can contain either of the following:

- A file called `.abaqus_`*hostname*`_passwd`. This file allows you to connect to the remote server on the machine called *hostname*.
- A file called `.abaqus_net_passwd`. This file allows you to connect to the network ODB server on any machine.

The contents of the password file on both the server and the client must be identical. In addition, Abaqus checks that you are the only user with permission to read from or to write to the password files. If neither file exists, Abaqus tries to use remote and secure shell commands to read the password from the network ODB server. However, the security configuration at your site may prevent Abaqus from reading the password.

Command Summary

abaqus networkDBConnector **port =** {*serverPortNumber* | `auto_assigned`}
[timeout = *time out value in seconds***]**
[host = *hostname***]**
[stop]
[ping]

Command Line Options

port This option specifies the port number on the network ODB server. If **port=**`auto_assigned`, Abaqus automatically assigns the port number.

timeout This option specifies the timeout period in seconds for the network ODB server. The server exits if it does not receive any communication from the client during the time specified. A timeout value of zero indicates that the server will run until it is terminated explicitly using the **stop** option.

host This option specifies the name of the machine that is hosting the network ODB server. This option is used with the **stop** and **ping** options. If this option is not provided, Abaqus uses the name of the machine from which the execution procedure was issued.

stop This option specifies that Abaqus should stop the network ODB server that was established using the specified host name and port number.

ping This option queries the network ODB file server that was established using the specified host name and port number. Use this option to confirm that the network ODB server exists and that communications have been established.

Mapping Thermal and Magnetic Loads

Products: Abaqus/Standard

References:

- *About Execution Procedures*
- *Eddy Current Analysis*
- *Predefined Loads for Sequential Coupling*
- **CFLUX*
- **CLOAD*

Overview

The **abaqus emloads** utility converts results output from a time-harmonic eddy current analysis for use as loads in a subsequent heat transfer, coupled temperature-displacement, or stress/displacement analysis. For example, magnetic body force intensity output is converted to point loads. You specify the names of the time-harmonic eddy current analysis results output database (`.odb`) file and the input file for the subsequent analysis on the command line. The utility creates an output database file containing a mesh that matches the mesh in your subsequent analysis and steady-state concentrated nodal fields consistent with the time-harmonic eddy current analysis results. Your time-harmonic eddy current and subsequent analysis meshes can be dissimilar, and results transfer ensures global conservation of the flux quantities when your model domains match; i.e., the model boundaries are the same. You can then use this new output database file to apply concentrated loads and concentrated heat fluxes in the subsequent analysis.

Results Conversion

The utility converts whole element output quantities from a time-harmonic eddy current analysis to nodal results. You use the options listed in *Table 1* in the subsequent analysis to specify the output database file (and optionally the step and increment) from which the data are to be read.

Table 1: Supported results conversion.

Electromagnetic analysis output variable	Converted output variable	Input file option
Rate of Joule heat dissipation EMJH	Concentrated heat flux CFL11	**CFLUX*, FILE=*odb-name*, STEP=*step-number*, INC=*inc*
Magnetic body force intensity EMBF	Point load components CF	**CLOAD*, FILE=*odb-name*, STEP=*step-number*, INC=*inc*

Utility Execution

The utility executes in two phases. Abaqus writes progress information and, if appropriate, error messages to the screen during each phase.

In the first phase a datacheck analysis is performed on your subsequent analysis input file to create an output database representation of a “target” mesh. This phase requires that your input file be sufficiently complete to successfully run **abaqus datacheck**, with the exception that you can have **CFLUX* and **CLOAD* options that include the FILE parameter to refer to files that are not available. If this phase is successful, the utility proceeds to the second phase; otherwise, an error message is issued.

In the second phase time-harmonic eddy current analysis load data are mapped from the source to the target output database. In this phase all steps and increments found in the original analysis are defined in the target output database. This phase requires that your target model domain lie within the source model domain. If it does not, an appropriate error message is issued.

Command Summary

abaqus emloads

job = *target-odb-name*
input = *subsequent analysis input-file-name*
sourceodb = *time-harmonic eddy current analysis odb-file-name*

Command Line Options

job This option specifies the name of the resulting “target” output database file.

input This option specifies the name of the subsequent analysis Abaqus input file. This file must be sufficiently complete to successfully run, as described above.

sourceodb This option specifies the name of the time-harmonic eddy current analysis output database file.

Element Matrix Assembly Utility

Products: Abaqus/Standard

References:

- *Matrices*

Overview

The **abaqus mtxasm** utility assembles element matrices contained in a SIM document and, optionally, writes the assembled matrices to text files. If assembled matrices are already available in a SIM document, you can use this utility to write them to text files.

Command Summary

abaqus mtxasm	**job =** *name-of-assembled-mtx-sim-doc* **[oldjob =** *name-of-element-mtx-sim-doc***]** **[text]**

Command Line Options

job This option is used to specify the name of the SIM document containing assembled matrices or to which assembled matrices will be written, depending on the mode of operation.

oldjob This option is used to specify the name of the SIM document containing element matrices to be assembled. If this option is not used, the SIM document specified using the **job** option must already exist and contain assembled matrices.

text This option is used to write assembled matrices to text files in the matrix input format. Each matrix is written to a file that follows the naming convention *jobname_matrixN*.`mtx`, where *jobname* is the name specified using the **job** option, *matrix* is a four-letter identifier (`STIF`, `MASS`, `DMPV`, `DMPS`, or `LOAD`) indicating the matrix type relevant for structural or thermal matrices, and *N* is the increment number.

Encrypting and Decrypting Abaqus Input Data

Products: Abaqus/Standard Abaqus/Explicit

References:

- *About Execution Procedures*
- *Including an Encrypted Data File*
- **INCLUDE*

Overview

You can use the **abaqus encrypt** utility to prevent the unauthorized use of Abaqus input data. The utility converts a data file into an encrypted, password-protected format that only authorized Abaqus input parties can access.

The utility is intended for the encryption of data that you include by reference in input (`.inp`) files or in other data files. For example, you could encrypt a file that contains all of the proprietary material data for your model, then include the encrypted data file by reference in an unencrypted Abaqus input file. See *Including an Encrypted Data File* for information on how to include an encrypted data file in an Abaqus input file.

You can encrypt any input file. However, Abaqus cannot run an encrypted Abaqus input file directly; the encrypted file must be included in an unencrypted file. You cannot use parameterized input in the encrypted file.

When you encrypt an Abaqus input file, each line becomes longer as part of the encryption process. A valid input file line may become invalid if its length is expanded beyond the 256 character limit. In this case shorten the line before encrypting the input file.

Specifying Additional Access Levels and Controls

You can customize your encryption so that only users with a license for a particular Abaqus feature or from a particular site can include or decrypt the file. For example, you can specify that only Abaqus/Standard users can access the file. You can also prevent decryption of an encrypted file by any user, regardless of their license and site; end users can still use the encrypted data in an analysis by including it by reference in an unencrypted Abaqus input file, provided that the users know the encrypted file's password.

Security and Support Considerations

The primary intent of the Abaqus encryption implementation is to prevent unauthorized use of encrypted input data, not to prevent disclosure of encrypted data to authorized users. Running an Abaqus analysis input using encrypted data may produce output files that are not encrypted. Only material and connector behavior information contained within an encrypted input file is prevented from being visible in the output. This approach means that recipients of encrypted data who satisfy the access criteria, such as the password, license feature, or SiteID, will be able to reconstruct some input in an unencrypted form. Providers of encrypted data should consider establishing contractual agreements to protect proprietary data. Users of encrypted data must accept responsibility for security of files produced from encrypted input and should consider restricting access to resulting analysis files.

Abaqus technical support cannot retrieve lost passwords for encrypted data files. Users receiving encrypted data should contact the data provider for any technical support issues.

Adding Comments to the Header of an Encrypted File

When you encrypt a file, Abaqus adds the following unencrypted comment line to the beginning of the file:

```
** encrypted input
```

Do not modify or delete this header comment. You can, however, insert additional comment lines between this header comment and the first line of encrypted data. These post-encryption comment lines can describe the encrypted file's contents, provide release numbers, or display copyright and legal information about the encrypted data. For more information about comment line syntax, see *Input Syntax Rules*.

You should not, however, add post-encryption comment lines within the lines of encrypted data. If you want to edit or amend the comment lines within the data itself, you must first decrypt the data.

Command Summary

abaqus {encrypt | decrypt} **input =** *input-file-name*
output = *output-file-name*
password = *password*
[license = *feature_list***]**
[siteid = *site-id_list***]**
[include_only]
[expiration = *expiration_date***]**

Command Line Options

input This option specifies the name of the data file that you want to encrypt or decrypt.

If you omit this option from the command line, Abaqus will prompt you for its value.

output This option specifies the name of the data file after encryption or decryption.

If you omit this option from the command line, Abaqus will prompt you for its value.

password This option specifies the password for this encryption or decryption. Passwords are case-sensitive.

If you omit this option from the command line while encrypting data, Abaqus will prompt you for its value. If you enter the password incorrectly or omit it from the command line while decrypting data, Abaqus reports that the input file is either corrupted or the password is incorrect.

license This option applies only to file encryption.

This option specifies the Abaqus feature or features for which end users must be licensed if they want to include or decrypt this encrypted data file. You can use a comma-separated list to allow access to the file by licensees of any one of a series of Abaqus features.

Any feature name that appears in an Abaqus license file is valid. These might include the following features: `foundation`, `standard`, `explicit`, `design`, `aqua`, `ams`, `cae`, `viewer`, `cae_nogui`, `cmold`, `moldflow`, `safe`, `cadporter_catia`, `cadporter_catiav5`, `cadporter_ideas`, `cadporter_parasolid`, `cadporter_proe`, `afcv5_structural`, and `afcv5_thermal`.

siteid This option applies only to file encryption.

This option specifies the Abaqus Site ID or IDs where end users can include or decrypt this encrypted data file. You can use a comma-separated list to allow multiple sites access to the file. You can use this option only when you also use the **license** option.

To determine your Abaqus Site ID, run `abaqus whereami` from a command prompt.

include_only This option applies only to file encryption.

This option specifies that encrypted input data cannot be decrypted using the **abaqus decrypt** execution procedure; such data can only be included in an Abaqus input file.

If you attempt to decrypt a file that was encrypted with the **include_only** option, Abaqus issues an error message stating that the input file can be included in an analysis but is not eligible for decryption.

expiration This option applies only to file encryption.

This option specifies the date after which the end users can no longer decrypt or include the encrypted data file. The date must be provided in the formYYYY-MM-DD.

Examples

The following examples illustrate the different encryption methods that are possible using the **encrypt** execution procedure.

Creating Encrypted Files

In the simplest encryption scenario an Abaqus user creates an encrypted copy of a file named `material_data.inp`, which contains all of the material data for a model, before sending the encrypted version to an authorized end user. Encryption prevents unauthorized users from accessing the encrypted file during its transmission. To create an encrypted copy of `material_data.inp` named `material_data_enc.inp`, issue the following command:

```
abaqus encrypt input=material_data.inp
   output=material_data_enc.inp password=e1No9c2z
```

Upon receiving the file, the end user can run the **abaqus decrypt** execution procedure to create a copy of the original, non-encrypted material data file. Because of the encryption options selected in this example, the end user requires only the encrypted file's password to decrypt it. To decrypt the encrypted data file `material_data_enc.inp`, producing the non-encrypted file `material_data.inp`, issue the following command:

```
abaqus decrypt input=material_data_enc.inp
   output=material_data.inp password=e1No9c2z
```

Alternatively, the end user can skip the decryption and run an analysis that includes the encrypted data by reference. To include the encrypted file by reference in an Abaqus input file, add the following statement to the input file:

```
*INCLUDE, INPUT=material_data_enc.inp, PASSWORD=e1No9c2z
```

Limiting Access to Decrypted Files by License Feature or Site ID

You can specify that end users cannot access the file unless they have a valid license for a particular Abaqus feature, run Abaqus at a particular site, or satisfy both of these criteria. To encrypt a data file that can be accessed only by users who have an Abaqus/Explicit license and who run the software at site `09YYY`, issue the following command:

```
abaqus encrypt input=material_data.inp
   output=material_data_enc.inp password=e1No9c2z
   license=explicit siteid=09YYY
```

An end user can attempt to access the file `material_data_enc.inp` using the same decryption or inclusion syntax specified in the previous example. For this encrypted file, Abaqus would validate that the end user has an Abaqus/Explicit license and is running Abaqus at site `09YYY` before providing access to the file. If the end user's license or site settings do not match those specified during encryption, Abaqus issues an error message that lists the licenses or sites that are required to access the file.

Creating Encrypted Files That Must Be Included to Be Used by Abaqus

You can use the **include_only** option to prevent end users from decrypting the file directly using **abaqus decrypt**. Authorized users can access a file encrypted with the **include_only** option by including the file by reference in an Abaqus input file. Material and connector behavior definitions within an encrypted input file are not written to the output database. In addition, all material and connector behavior definitions output to the data file are suppressed if an encrypted file is used as input for any portion of the model. To create an encrypted file that is available only for inclusion by reference in other input files, issue the following command:

```
abaqus encrypt input=material_data.inp
   output=material_data_enc.inp password=e1No9c2z include_only
```

The resulting encrypted file can be included by reference in an Abaqus input file using the same syntax as in the previous example. If you attempt to decrypt a file that was encrypted with the **include_only** option, Abaqus returns an error message.

ASCII Translation of Results (.fil) Files

Products: Abaqus/Standard Abaqus/Explicit

References:

- *About Execution Procedures*

Overview

The **abaqus ascfil** translation facility:

- is provided to convert results (.fil) files (produced by an Abaqus analysis) to ASCII format for porting between dissimilar operating systems;
- permits the movement of results data to a different system for postprocessing; and
- can also be used to convert a results file in ASCII format to binary format to save disk space.

Command Summary

abaqus ascfil **job =** *job-name*
[input = *input-file***]**

Command Line Options

job This option specifies the input and output file names to use during results file translation. The *job-name* value is used as the default input file name. The translated output file will have the name *job-name*.fin.

If the input file is in binary format (default), this utility will create the *job-name*.fin file in ASCII format. To transfer the results file back to binary format after porting to a dissimilar operating system, rename the *job-name*.fin file to *job-name*.fil, and use this utility again; the resulting *job-name*.fin file will be in binary format.

If this option is omitted from the command line, you will be prompted for this value.

input This option specifies the name of the input file if it is different from *job-name*.

Examples

To convert the results file c4.fil from binary to ASCII format, use the following command:

```
abaqus ascfil job=c4
```

The translated file will have the name c4.fin.

Joining Results (.fil) Files

Products: Abaqus/Standard Abaqus/Explicit

References:

- *About Execution Procedures*

Overview

The **abaqus append** postprocessing facility:

- is provided to join results (.fil) files into a single file;
- permits two results files that may be either ASCII or binary files, or a combination of ASCII and binary, to be joined for further postprocessing; and
- will write a results file in the same format as the file specified with the **oldjob** option.

A similar utility, **abaqus restartjoin**, is used to join output database (.odb) files. See *Joining Output Database (.odb) Files from Restarted Analyses* for details.

Command Summary

abaqus append	**job =** *job-name* **oldjob =** *oldjob-name* **input =** *input-file*

Command Line Options

job This option specifies the output file name to use during execution. The *job-name* value is used as the output file name. The joined output file will have the name *job-name*.fil.

If this option is omitted from the command line, you will be prompted for this value.

oldjob This option specifies the name of the first results file to use during execution. The *oldjob-name* value is used as the results file name.

If this option is omitted from the command line, you will be prompted for this value.

input This option specifies the name of the second results file to use during execution. The *input-file* results file will be appended to the *oldjob-name* results file.

If this option is omitted from the command line, you will be prompted for this value.

Examples

The following command will append the history contents of the fjoin003.fil results file to the end of the fjoin002.fil results file and create the file fjoin001.fil:

```
abaqus append job=fjoin001 oldjob=fjoin002
input=fjoin003
```

Translating Other Solver Input to Abaqus Input

This section describes using commands to convert content from other solvers into formats that can be accessed in Abaqus.

In this section:

- *Translating Nastran Data to Abaqus Files*
- *Translating ANSYS Input Files to Partial Abaqus Input Files*
- *Translating PAM-CRASH Input Files to Partial Abaqus Input Files*
- *Translating RADIOSS Input Files to Partial Abaqus Input Files*
- *Translating LS-DYNA Data Files to Abaqus Input Files*
- *Exchanging Abaqus Data with ZAERO*
- *Translating a Simpack Flexible Body to Abaqus Matrix Data*
- *Translating Moldflow Data to Abaqus Input Files*

Translating Nastran Data to Abaqus Files

The translator from Nastran to Abaqus converts finite element entities in a Nastran text input file into their equivalent in an Abaqus text input file. It can also convert matrix data (DMIGs) to their equivalent in an Abaqus binary SIM file.

In this section:

- *Translating Nastran Bulk Data in Text Files to Partial Abaqus Input Files*
- *Translating Nastran DMIG Data in Bulk Data Files to an Abaqus Binary SIM File*
- *Translating Nastran DMIG Matrix Data in Output2 Binary Files to an Abaqus Binary SIM File*

Translating Nastran Bulk Data in Text Files to Partial Abaqus Input Files

Products: Abaqus/Standard Abaqus/Explicit

References:

- *About Execution Procedures*
- *Translating an Abaqus Input File to a Partial Nastran Bulk Data Text File*
- *Importing a model from a Nastran input file*

Overview

The translator from Nastran to Abaqus converts finite element entities in a Nastran text input file into their equivalent in an Abaqus text input file.

Overview

The Nastran data must be in a file with the extension `.bdf`, `.dat`, `.nas`, `.nastran`, `.blk`, or `.bulk`. The Nastran data entries that are translated are listed in the tables below. Other valid Nastran data are skipped over and noted in the log file.

The translator is designed to translate a complete Nastran input file. If only bulk data are present, the first two lines in the file should be the terminators for the executive control and case control sections, namely:

```
CEND
BEGIN BULK
```

For normal termination, end the Nastran input data with the line

```
ENDDATA
```

Two levels of `INCLUDE` are supported.

Summary of Nastran Entities Translated

Table 1: Executive control data.

Nastran Statement		Abaqus Equivalent
SOL		
1	(STATICS1)	**STATIC*
24	(STATICS)	
101	(SESTATIC)	
106	(NLSTATIC)	
3	(MODES)	**FREQUENCY*
25	(OLDMODES)	
103	(SEMODES)	
5	(BUCKLING)	**BUCKLE*
105	(SEBUCKL)	
26	(DFREQ)	**STEADY STATE DYNAMICS*, DIRECT
108	(SEDFREQ)	

Nastran Statement		Abaqus Equivalent
27	(DTRAN)	**DYNAMIC*
109	(SEDTRAN)	
107	(SEDCEIG)	**COMPLEX FREQUENCY*
110	(SEMCEIG)	
30	(DFREQ)	**FREQUENCY* and **STEADY STATE DYNAMICS*
111	(SEMFREQ)	
31	(MTRAN)	**FREQUENCY* and **MODAL DYNAMIC*
112	(SEMTRAN)	

Table 2: Case control data.

Nastran Command	Comment
SPC	Selects SPC sets alone or in combinations
LOAD	Selects individual loads and load combinations
METHOD	Selects EIGRL, EIGR, or EIGB from bulk data for eigenfrequency extraction and eigenvalue buckling prediction procedures
SUBCASE	Delimiter for steps or load cases; optional if there is only one step
TITLE	Echoed as comment at top of input file and for each step
SUBTITLE	Echoed as comment for the step to which it applies
LABEL	Used as text following the **STEP* option
DLOAD	Selects dynamic loads from bulk data
LOADSET	
FREQUENCY	Selects forcing frequencies from bulk data
MPC	Selects MPCADD and MPC from bulk data if referenced in the first SUBCASE
SUPORT1	Selects SUPORT1 from bulk data
TSTEP	Selects TSTEP from bulk data
K2GG	Selects DMIG from bulk data using the matrix name from the first SUBCASE
K2PP	
M2GG	
M2PP	
B2GG	
B2PP	
K42GG	
TEMPERATURE	Selects nodal temperatures from bulk data
SET	Selects nodal quantities for output
DISPLACEMENT	
VELOCITY	
ACCELERATION	
SPCFORCES	

Nastran Command	Comment
PRESSURE	

Table 3: Bulk data.

Nastran Data Entry	Comment
PARAM	Ignored except for: 1. WTMASS, which can be used to modify density, mass, and rotary inertia values if the **wtmass_fixup** command line parameter is used 2. INREL, which if equal to −1 or −2 will create inertia relief loads 3. G, which is translated to **GLOBAL DAMPING*, STRUCTURAL, FIELD=MECHANICAL 4. GFL, which is translated to **GLOBAL DAMPING*, STRUCTURAL, FIELD=ACOUSTIC
CDAMP1	DASHPOT1/DASHPOT2 and **DASHPOT* (CDAMP2 at SPOINTs are translated to **MATRIX INPUT* viscous damping terms.)
CDAMP2	
PDAMP	
PDAMPT	
CELAS1	SPRING1/SPRING2 and **SPRING* (CELAS2 at SPOINTs are translated to **MATRIX INPUT*, stiffness, and/or structural damping terms.)
CELAS2	
PELAS	
PELAST	
CMASS2	**MATRIX INPUT* mass terms
CBUSH	CONN3D2 and **CONNECTOR SECTION*
PBUSH	
PBUSHT	
CWELD	**FASTENER* and **FASTENER PROPERTY*
PWELD	
CONM1	MASS and/or ROTARY INERTIA and/or UEL
CONM2	MASS and/or ROTARY INERTIA
CHEXA	C3D8I/C3D20R/C3D6/C3D15/C3D4/C3D10 and **SOLID SECTION*
CPENTA	
CTETRA	
PSOLID	
PLSOLID	
CQUAD4	S4/S3R/S8R/STRI65, and **SHELL SECTION*, **SHELL GENERAL SECTION*, or **MEMBRANE SECTION*.
CTRIA3	
CQUAD8	
CTRIA6	
CQUADR	
CTRIAR	
PSHELL	
PCOMP	
PCOMPG	

Nastran Data Entry	Comment
CSHEAR	**USER ELEMENT*, LINEAR and **MATRIX*, TYPE=STIFFNESS and TYPE=MASS, or SHEAR4 and **SHELL GENERAL SECTION*
PSHEAR	
CBAR	B31 and **BEAM SECTION* or **BEAM GENERAL SECTION*
CBEAM	
PBAR	
PBARL	
PBEAM	
BEAML	
CROD	T3D2 and **SOLID SECTION*
CONROD	
PROD	
CGAP	GAPUNI and **GAP*
PGAP	
RBAR	**COUPLING* or **MPC*, type BEAM
MAT1	**ELASTIC*, TYPE=ISO; **EXPANSION*, TYPE=ISO; **DENSITY*; and **DAMPING* (G is used only for **BEAM GENERAL SECTION*)
MAT2	When used alone in a PSHELL, MAT2 is translated to **ELASTIC*, TYPE=LAMINA or **ELASTIC*, TYPE=ANISOTROPIC. When used in combination with other materials, the coefficients relating midsurface strains and curvatures to section forces and moments are computed and entered following the **SHELL GENERAL SECTION* option.
MAT8	**ELASTIC*, TYPE=LAMINA; **EXPANSION*, TYPE=ORTHO; **DENSITY*; and **DAMPING*
MAT9	**ELASTIC*, TYPE=ANISOTROPIC unless the data are found to be orthotropic, in which case the data are analyzed to create **ELASTIC*, TYPE=ENGINEERING CONSTANTS. Also **DENSITY*; **EXPANSION*, TYPE=ANISO or ORTHO; and **DAMPING*.
MAT11	**ELASTIC*, TYPE=ENGINEERING CONSTANTS. Also **DENSITY*; **EXPANSION*, TYPE=ANISO or ORTHO; and **DAMPING*.
ACMODL	**TIE* between a **SURFACE*, TYPE=ELEMENT defining the exterior surfaces of all acoustic solid elements and a **SURFACE* defined by the SET1 referenced by the SSID.
NSM	**NONSTRUCTURAL MASS*
NSM1	
NSML	
NSML1	
NSMADD	
GRID	**NODE* and **SYSTEM*
CORD1R	**SYSTEM* for nodes; **TRANSFORM* if referred to on GRID; **ORIENTATION* for some elements
CORD1C	
CORD1S	
CORD2R	
CORD2C	
CORD2S	

Nastran Data Entry	Comment
RBE2	**COUPLING* and **KINEMATIC*; or **KINEMATIC COUPLING* (If the RBE2 has only two nodes and neither node has rotational stiffness, the RBE2 is translated to **MPC*, type LINK)
RBE3	**COUPLING* and **DISTRIBUTING*; or DCOUP3D and **DISTRIBUTING COUPLING*
SPCADD	Used to combine SPC/SPC1/SPCD data into a new set
SPC	**BOUNDARY*
SPC1	
SPCD	
LOAD	Used to combine FORCE, MOMENT, etc. data into a new set
FORCE	**CLOAD*
FORCE1	
FORCE2	
MOMENT	
MOMENT1	
MOMENT2	
PLOAD	**DLOAD*
PLOAD1	
PLOAD2	
PLOAD4	
RFORCE	
DLOAD	Dynamic loads as functions of time or frequency
DAREA	
LSEQ	
RLOAD1	
RLOAD2	
TLOAD1	
TABLED1	
TABLED2	
TABLED4	
DELAY	
DPHASE	
TEMP	**INITIAL CONDITIONS*, TYPE=TEMPERATURE and **TEMPERATURE*
TEMPD	
TSTEP	Time step size for dynamic and modal dynamic procedures
EIGB	**BUCKLE*
EIGR	**FREQUENCY*
EIGRL	
EIGC	**COMPLEX FREQUENCY*

Nastran Data Entry	Comment
TABDMP1	**MODAL DAMPING*
FREQ	Forcing frequencies for steady-state dynamic procedures
FREQ1	
FREQ2	
FREQ3	
FREQ4	
FREQ5	
MPCADD	**EQUATION*
MPC	
SUPORT	**INERTIA RELIEF* and **BOUNDARY*
SUPORT1	
DMIG	**MATRIX INPUT* and **MATRIX ASSEMBLE*
GENEL	**USER ELEMENT*, LINEAR and **MATRIX*, TYPE=STIFFNESS
PLOTEL	CONN3D2 and **CONNECTOR SECTION* (Ignored unless the command line option **plotel**=ON.)
PLOTEL3 PLOTEL4	SFM3D3/SFM3D4 and **SURFACE SECTION* (Ignored unless the command line option **plotel**=ON.)

Command Summary

abaqus fromnastran

job = *job-name*
[input = *input-file***]**
[wtmass_fixup = {OFF | ON**}]**
[loadcases = {OFF | ON**}]**
[pbar_zero_reset [= *small-real-number* **]]**
[surface_based_coupling = {OFF | ON**}]**
[beam_offset_coupling = {OFF | ON**}]**
[beam_orientation_vector = {OFF | ON**}]**
[cbar = *2-node-beam-element***]**
[cquad4 = *4-node-shell-element***]**
[chexa = *8-node-brick-element***]**
[cpyram = *5-node-pyramid-element***]**
[ctetra = *10-node-tetrahedron-element***]**
[cshear = {UEL | SHEAR4**}]**
[plotel = {OFF | ON**}]**
[cdh_weld = {OFF | RIGID | COMPLIANT**}]**

Command Line Options

job

This option is used to specify the name of the Abaqus input file to be output by the translator. It is also the default name of the file containing the Nastran data. Diagnostics created by the translator will be written to a file named *job-name*`.log`.

input

This option is used to specify the name of the file containing the Nastran data if it is different from *job-name*.

wtmass_fixup If **wtmass_fixup**=ON, the value on the Nastran data line `PARAM, WTMASS, value` is used as a multiplier for all density, mass, and rotary inertia values created in the Abaqus input file.

This option can be defined in the Abaqus environment file as follows:

```
fromnastran_wtmass_fixup={OFF | ON}
```

loadcases By default, each `SUBCASE` is translated to a **STEP* option in Abaqus. If **loadcases=**ON, this behavior is altered for linear static analyses: each `SUBCASE` is translated to a **LOAD CASE* option, and all such **LOAD CASE* options are grouped in a single **STEP* option.

This option can be defined in the Abaqus environment file as follows:

```
fromnastran_loadcases={OFF | ON}
```

pbar_zero_reset Nastran allows beams to have zero values for cross-sectional area or moments of inertia; Abaqus does not. Set this option equal to a small real number to reset any zero values for A, I_1, I_2, or J to the specified small real number. If this option is omitted or present without a value, the default value of 1.0×10^{-20} is used in place of the zeros. To retain the zeros in the translated Abaqus input file, set **pbar_zero_reset**=0.

This option can be defined in the Abaqus environment file as follows:

```
fromnastran_pbar_zero_reset=small-real-number
```

surface_based_coupling Certain Nastran rigid elements have more than one equivalent in Abaqus. If **surface_based_coupling**=ON, RBE2 and RBE3 elements translate to **COUPLING* with the appropriate parameters. Otherwise, RBE2 elements translate to **KINEMATIC COUPLING* and RBE3 elements translate to **DISTRIBUTING COUPLING*. This translation behavior also applies to "implied"RBE2-type rigid elements used for offsets on CBAR, CBEAM, and CONM2 elements.

For input files created with **surface_based_coupling**=ON, the translated elements can be visualized and manipulated in Abaqus/CAE. However, large numbers of these elements may cause slower performance.

This option can be defined in the Abaqus environment file as follows:

```
fromnastran_surface_based_coupling={OFF | ON}
```

beam_offset_coupling If **beam_offset_coupling**=ON, beam element offsets are translated by creating new nodes at the offset locations, changing the beam connectivity to the new nodes, and rigidly coupling the new and original nodes.

If **beam_offset_coupling**=OFF, beam element offsets are translated to the **CENTROID* and **SHEAR CENTER* options, which are suboptions of the **BEAM GENERAL SECTION* option.

The setting for this parameter is ignored if the beam element references a PBARL or PBEAML property or if the beam offset has a significant component in the direction of the beam axis. In these situations the beam offsets are always translated as if **beam_offset_coupling**=ON.

This option can be defined in the Abaqus environment file as follows:

```
fromnastran_beam_offset_coupling={OFF | ON}
```

beam_orientation_vector If **beam_orientation_vector**=`OFF`, beam cross-section orientations are translated by creating new nodes at the tips of vectors defining the first principal direction of the cross-section and changing the beam connectivity to the new nodes.

If **beam_orientation_vector**=`ON`, beam cross-sections are translated by defining vectors on the **BEAM SECTION* and **BEAM GENERAL SECTION* options.

This option can be defined in the Abaqus environment file as follows:

```
fromnastran_beam_orientation_vector={OFF | ON}
```

cbar This option is used to define the 2-node beam that is created from CBAR and CBEAM elements. The default is B31.

This option can be defined in the Abaqus environment file as follows:

```
fromnastran_cbar=2-node-beam-element
```

cquad4 This option is used to define the 4-node shell that is created from CQUAD4 elements. The default is S4R. If a reduced-integration element is chosen, the enhanced hourglass formulation is applied automatically.

This option can be defined in the Abaqus environment file as follows:

```
fromnastran_cquad4=4-node-shell-element
```

chexa This option is used to define the 8-node brick that is created from CHEXA elements. The default is C3D8I. If a reduced-integration element is chosen, the enhanced hourglass formulation is applied automatically.

This option can be defined in the Abaqus environment file as follows:

```
fromnastran_chexa=8-node-brick-element
```

cpyram This option is used to define the 5-node pyramid that is created from CPYRAM elements. The default is C3D5. You can specify an 8-node brick element type, which degenerates to the appropriate pyramid shape.

ctetra This option is used to define the 10-node tetrahedron that is created from CTETRA elements. The default is C3D10.

This option can be defined in the Abaqus environment file as follows:

```
fromnastran_ctetra=10-node-tetrahedron-element
```

cshear By default, CSHEAR elements are translated to user elements, as described in *Table 3*. If **cshear**=`SHEAR4`, CSHEAR elements are translated to SHEAR4 elements.

plotel By default, PLOTEL elements are not translated. If **plotel**=`ON`, PLOTEL, PLOTEL3, and PLOTEL4 elements are translated as described above.

cdh_weld By default, CHEXA elements with RBE3 elements at all eight corner nodes are translated to the type of 8-node element specified in the **chexa** parameter. If **cdh_weld**=`RIGID`, CHEXA elements with RBE3 elements at all eight corner nodes are translated to rigid fasteners in Abaqus. If **cdh_weld**=`COMPLIANT`, CHEXA elements with RBE3 elements at all eight corner nodes are translated to compliant fasteners in Abaqus.

Translating Nastran DMIG Data in Bulk Data Files to an Abaqus Binary SIM File

Products: Abaqus/Standard Abaqus/Explicit

References:

- *About Execution Procedures*
- *Translating an Abaqus Input File to a Partial Nastran Bulk Data Text File*
- *Importing a model from a Nastran input file*

Overview

The translator from Nastran to Abaqus converts matrix data (DMIGs) to their equivalent in an Abaqus binary SIM file.

Overview

You can specify that the translator create a SIM file of matrix data in addition to a text input file. The matrix data can be translated to:

- a SIM file structured as if it were created from a **MATRIX GENERATE* step in an Abaqus input file or
- a SIM file equivalent to that resulting from an Abaqus analysis with a **SUBSTRUCTURE GENERATE* step.

Command Summary

abaqus fromnastran **job =** *job-name*
[input = *input-file***]**
[dmig2sim = {`GENERIC` | `SUBSTRUCTURE`**}]**

Command Line Options

job This option is used to specify the name of the Abaqus input file to be output by the translator. It is also the default name of the file containing the Nastran data. Diagnostics created by the translator will be written to a file named *job-name*`.log`.

input This option is used to specify the name of the file containing the Nastran data if it is different from *job-name*.

dmig2sim This option is used to write DMIG matrix data to a binary SIM file for further processing by Abaqus.

If **dmig2sim=**`GENERIC`, a SIM file with a generic matrix system equivalent to that produced by a **MATRIX GENERATE* step is created.

If **dmig2sim=**`SUBSTRUCTURE`, a SIM file with a substructure matrix system equivalent to that produced by a **SUBSTRUCTURE GENERATE* step is created.

Translating Nastran DMIG Matrix Data in Output2 Binary Files to an Abaqus Binary SIM File

Products: Abaqus/Standard Abaqus/Explicit

References:

- *About Execution Procedures*
- *Translating an Abaqus Input File to a Partial Nastran Bulk Data Text File*
- *Importing a model from a Nastran input file*

Overview

The translator from Nastran to Abaqus converts matrix data (DMIGs) to their equivalent in an Abaqus binary SIM file.

Overview

The Nastran matrix data can be in one or two binary Output2 files. The DMIG matrix data are assumed to be written to an Output2 file using a command as follows:

```
ASSIGN OUTPUT2='jobname_matrixdata.op2',UNIT=30
…
EXTSEOUT(STIFFNESS,
         MASS,
         DAMPING,
         K4DAMP,
         LOADS,
         ASMBULK,
         EXTBULK,
         EXTID=10,
         DMIGOP2=30)
```

The nodal coordinate data may be in a second Output2 file, created with a command as follows:

```
ASSIGN OUTPUT2='jobname.op2',UNIT=12
…
DISP(PLOT) = ALL
…
PARAM,POST,-2
```

These Output2 files are referenced by the **op2file1** and **op2file2** options. The use of **op2file2** is optional. Using the file names from the example above, you specify the command line options as follows:

```
op2file1=jobname_matrixdata.op2 op2file2=jobname.op2
```

The **op2target** option determines the type of matrix data to create. The matrix data can be translated to:

- a partial Abaqus input file with a **MATRIX INPUT* representation of the matrix data,
- a SIM file structured as if it were created from a **MATRIX GENERATE* step in an Abaqus input file, or
- a SIM file equivalent to that resulting from an Abaqus analysis with a **SUBSTRUCTURE GENERATE* step.

Command Summary

abaqus fromnastran **job =** *job-name*
[input = *input-file***]**
[dmig2sim = {GENERIC | SUBSTRUCTURE**}]**
[op2file1 = *op2-filename-1***]**
[op2file2 = *op2-filename-2***]**

[op2target = {`INPUT` | `GENERIC` | `SUBSTRUCTURE`**}]**

Command Line Options

job This option is used to specify the name of the Abaqus input file to be output by the translator. It is also the default name of the file containing the Nastran data. Diagnostics created by the translator will be written to a file named *job-name*`.log`.

input This option is used to specify the name of the file containing the Nastran data if it is different from *job-name*.

op2file1 This option is used only in a workflow that reads DMIG matrix data in an Output2 file. It specifies the name of an Output2 file containing DMIG matrix data. The complete file name with the extension must be given.

op2file2 This option is used to give the name of a second Output2 file containing nodal coordinate data associated with DMIG entries. If the **op2file1** option is present, specifying **op2file2** is optional. If specified, the complete file name with the extension must be given.

op2target This option controls the translation behavior for the DMIG matrix data in an Output2 file.

If **op2target=**`INPUT`, a partial Abaqus input file containing a **MATRIX INPUT* option is created.

If **op2target=**`GENERIC`, a SIM file with a generic matrix system equivalent to that produced by a **MATRIX GENERATE* step is created.

If **op2target=**`SUBSTRUCTURE`, a SIM file with a substructure matrix system equivalent to that produced by a **SUBSTRUCTURE GENERATE* step is created.

Translating ANSYS Input Files to Partial Abaqus Input Files

Products: Abaqus/Standard Abaqus/Explicit

References:

- *About Execution Procedures*

Overview

The translator from ANSYS to Abaqus converts certain entities in an ANSYS blocked coded database file into their equivalent in an Abaqus input file.

Using the Translator

The **abaqus fromansys** translator can convert ANSYS blocked coded database files (`.cdb`) into a "flat" Abaqus input file; that is, an Abaqus input file that is not written in terms of parts and assemblies. The `.cdb` file must be created in ANSYS using the following command:

```
CDWRITE , , <jobname>, cdb
```

The second field of the `CDWRITE` command may contain `ALL` or `DB`. The eighth field may contain `BLOCKED`. Any other use of the `CDWRITE` command will create problems for the translator.

Summary of ANSYS Entities Translated

The translator from ANSYS to Abaqus supports the mappings shown in the tables below.

Table 1: Nodal data mapping for ANSYS commands.

ANSYS command	Abaqus equivalent
NBLOCK	**NODE* **TRANSFORM*

Table 2: Element data mapping for ANSYS structural lines.

ANSYS command	Abaqus equivalent
LINK1	**ELEMENT*, TYPE=T2D2
LINK8	**ELEMENT*, TYPE=T3D2
LINK10	**ELEMENT*, TYPE=T3D2
LINK11	**ELEMENT*, TYPE=T3D2
LINK180	**ELEMENT*, TYPE=T3D2

Table 3: Element data mapping for ANSYS structural beams.

ANSYS command	Abaqus equivalent
BEAM3	**ELEMENT*, TYPE=B21
BEAM4	**ELEMENT*, TYPE=B31
BEAM23	**ELEMENT*, TYPE=B21

ANSYS command	Abaqus equivalent
BEAM24	**ELEMENT*, TYPE=B31
BEAM188	**ELEMENT*, TYPE=B31 or B32
BEAM189	**ELEMENT*, TYPE=B32

Table 4: Element data mapping for ANSYS structural shells.

ANSYS command	Abaqus equivalent
SHELL43	**ELEMENT*, TYPE=S4 or S3
SHELL63	**ELEMENT*, TYPE=S4, S3, M3D4, or M3D3
SHELL93	**ELEMENT*, TYPE=S8R or STRI65
SHELL181	**ELEMENT*, TYPE=S4R or S3R

Table 5: Element data mapping for ANSYS structural pipes.

ANSYS command	Abaqus equivalent
PIPE16	**ELEMENT*, TYPE=PIPE32
PIPE20	**ELEMENT*, TYPE=PIPE31
PIPE59	**ELEMENT*, TYPE=PIPE31

Table 6: Element data mapping for ANSYS planar elements.

ANSYS command	Abaqus equivalent
PLANE42 PLANE82 PLANE182 PLANE183	**ELEMENT*, TYPE=CPS*n*, CAX*n*, or CPE*n*

Table 7: Element data mapping for ANSYS solid elements.

ANSYS command	Abaqus equivalent
SOLID45	**ELEMENT*, TYPE=C3D8I, C3D4, or C3D6
SOLID65	**ELEMENT*, TYPE=C3D8I, C3D4, or C3D6
SOLID92	**ELEMENT*, TYPE=C3D10
SOLID95	**ELEMENT*, TYPE=C3D20, C3D10, or C3D15
SOLID147	**ELEMENT*, TYPE=C3D20, C3D10, or C3D15
SOLID148	**ELEMENT*, TYPE=C3D10
SOLID185	**ELEMENT*, TYPE=C3D8, C3D4, or C3D6
SOLID186	**ELEMENT*, TYPE=C3D20R, C3D10, or C3D15
SOLID187	**ELEMENT*, TYPE=C3D10

Table 8: Load and boundary condition data mapping.

ANSYS command	Abaqus equivalent
SFE, *ELEM*, *LKEY*, PRES, *KVAL*, *VAL1*, *VAL2*, *VAL3*, *VAL4*, where *VAL1=VAL2=VAL3=VAL4=n*	**SURFACE* and **DSLOAD*

ANSYS command	Abaqus equivalent
SFE, *ELEM*, *LKEY*, HFLU, *KVAL*, *VAL1*, *VAL2*, *VAL3*, *VAL4*, where *VAL1=VAL2=VAL3=VAL4=n*	**SURFACE* and **DSFLUX*
BF, *NODE*, TEMP, *VAL1*, *VAL2*, *VAL3*, *VAL4*	**TEMPERATURE* and **CFLUX*
BFE, *NODE*, HGEN, *STLOCVAL1*, *VAL2*, *VAL3*, *VAL4*	**DFLUX*
ACEL, *1-component*, *2-component*, *3-component*	**DLOAD*
F, *NODE*, *Lab*, *VALUE*, *VALUE2*, *NEND*, *NINC*, where *Lab*=FX, FY, or FZ	**CLOAD*
D, *NODE*, *Lab*, *VALUE*, *VALUE2*, *NEND*, *NINC*, where *Lab*=UX ,UY, UZ, ROTX, ROTY, or ROTZ	**BOUNDARY*

Table 9: Material data mapping.

ANSYS command	Abaqus equivalent
MPTEMP, … MPDATA, … , EX MPDATA, … , NUXY or PRXY	**MATERIAL* and **ELASTIC* Minor Poisson's ratios (such as NUXY), if present, are automatically converted to major Poisson's ratios (such as PRXY).
MPTEMP, …. MPDATA, … , EX MPDATA, … , EY MPDATA, … , EZ MPDATA, … , NUXY or PRXY MPDATA, … , NUXZ or PRXZ MPDATA, … , NUYZ or PRYZ MPDATA, … , GXY MPDATA, … , GXZ MPDATA, … , GYZ	**MATERIAL* and **ELASTIC*, TYPE=ENGINEERING CONSTANTS Minor Poisson's ratios (such as NUXY), if present, are automatically converted to major Poisson's ratios (such as PRXY).
MPTEMP, … MPDATA, … , KXX	**MATERIAL* and **CONDUCTIVITY*
MPTEMP, … MPDATA, … , DENS	**DENSITY*
MPTEMP, … MPDATA, … , C	**SPECIFIC HEAT*
MPTEMP, … MPDATA, … , CTEX or ALPX	**EXPANSION*

Command Summary

abaqus fromansys **job =** *job-name*
[input = *input-file***]**

Command Line Options

job This option is used to specify the name of the Abaqus input file to be output by the translator. It is also the default name of the input file containing the ANSYS data. Diagnostics created by the translator will be written to a file named *job-name*`.log`.

input This option is used to specify the name of the file containing the ANSYS data if it is different from *job-name*.

Translating PAM-CRASH Input Files to Partial Abaqus Input Files

Products: Abaqus/Explicit

References:

- *About Execution Procedures*

Overview

The translator from PAM-CRASH to Abaqus converts certain keywords in a PAM-CRASH input file into their equivalent in Abaqus/Explicit.

Using the Translator

The translator requires an input file created by PAM-CRASH Version 2002 or later. The input file can have any name and extension.

The PAM-CRASH data entries that are translated are listed in the tables below. Other PAM-CRASH keywords and data are skipped over and noted in the log file.

The translator creates a partial Abaqus input file that contains only the model data. You must provide history data (including output data) to complete the input.

Element Numbering and Grouping

All elements must have unique element numbers. Elements that are assigned the same PART identification number are grouped together in an element set.

Except for connector elements that result from the translation of SPRING and KJOIN, section properties need to be entered in the PART section rather than individually in the element section. Elements that have different material or section properties should be given different PART identification numbers; that is, the same material and section properties must be applicable to all elements grouped in the same element set.

If elements that result from the translation of SPRING and KJOIN have different element data (such as frame numbers used to define local directions), and they are assigned the same PART identification number, the translator automatically separates them into different element sets.

Material Models

The translator supports only the material models shown in *Table 3*. All unsupported material models between Types 1 and 99 are translated as bilinear elastic-plastic, and all other material types are translated as linear elastic if a stress-strain law definition is required. In these cases the translator provides nominal values for the material properties.

History Section Data

The translator creates a history section based partially on keywords (except TITLE) from the control section of the PAM-CRASH file as shown in *Table 1*. Other control data are unsupported.

Summary of PAM-CRASH Entities Translated

Table 1: Control section data.

PAM-CRASH keyword	Abaqus equivalent
TITLE	**HEADING*
RUNEND	**DYNAMIC*, EXPLICIT time period
TCTRL / DYNA_MASS_SCALE	**VARIABLE MASS SCALING*
ECTRL / RATEFILTER	**MATERIAL*, SRATE FACTOR

Table 2: Part section data.

PAM-CRASH keyword	Abaqus equivalent
PART / BAR	Truss element properties and grouping data
PART / BEAM	Beam element properties and grouping data
PART / SPRING	Connector behavior and grouping data
PART / KJOIN	Connector type, behavior, and grouping data
PART / SOLID	Solid element properties and grouping data
PART / SHELL	Shell element properties and grouping data
PART / MEMBR	Membrane element properties and grouping data
PART / TIED	Mesh tie constraint data and parameters
PART / PLINK	Mesh-independent fastener data and parameters

Table 3: Material section data.

PAM-CRASH keyword	Abaqus equivalent
MATER / Types 1, 16, 41, 99	C3D4/C3D6/C3D8R; solid material model data
MATER / Types 100, 101, 102, 103, 105	S3RS/S4RS; shell material model data
MATER / Types 150, 151	M3D3/M3D4/M3D4R and **USER MATERIAL*
MATER / Types 200, 201, 202	T3D2/B31; beam and truss material model data
MATER / Types 203, 204, 205, 230	CONN3D2; connector behavior data
MATER / Types 212, 213	B31; beam material model data
MATER / Type 302[1]	CONN3D2; connector behavior data
[1] Material type 302 supports the use of a rupture model (see RUPMO in *Table 10*).	

Table 4: Node section data.

PAM-CRASH keyword	Abaqus equivalent
FRAME	**ORIENTATION* and **TRANSFORM*
NODE	**NODE*
MASS	**MASS* and **ROTARY INERTIA*
NSMAS	**NONSTRUCTURAL MASS*

PAM-CRASH keyword	Abaqus equivalent
INVEL	**INITIAL CONDITIONS*, TYPE=VELOCITY or ROTATING VELOCITY
BOUNC	**BOUNDARY*
DIS3D	**BOUNDARY* and **AMPLITUDE*
VEL3D	**BOUNDARY* and **AMPLITUDE*
DAMP	**DLOAD* and **AMPLITUDE*
TRSFM	**NODE* with transformed coordinates

Table 5: Element section data.

PAM-CRASH keyword	Abaqus equivalent
SOLID	C3D4/C3D6/C3D8R and **SOLID SECTION*
TETR4	C3D4 and **SOLID SECTION*
SHELL	S3RS/S4RS and **SHELL SECTION*
MEMBR	M3D3/M3D4R and **MEMBRANE SECTION*
BEAM	B31 and **BEAM SECTION*, SECTION=CIRC
BAR	For MATER / Types 203 and 204: CONN3D2 and **CONNECTOR SECTION* [AXIAL]For all other MATER / Types: T3D2 and **SOLID SECTION*
SPRING	CONN3D2 and **CONNECTOR SECTION* [CARTESIAN + CARDAN]
KJOIN	CONN3D2 and **CONNECTOR SECTION*
PLINK	**FASTENER* and **FASTENER PROPERTY*; CONN3D2 and **CONNECTOR SECTION*

Table 6: Constraint section data.

PAM-CRASH keyword	Abaqus equivalent
RWALL (Stationary, segmented finite rigid wall) Velocity flag=0 Wall description=20	**RIGID BODY* and **CONTACT*
RBODY Types 0, 3	**RIGID BODY* and/or **MPC* (type BEAM) To define a group of elements as a rigid body, enter the part identification number of that element group as the PART entity[1]. To define an element as a rigid body, enter the element number as the ELE entity or enter all the element node numbers as the NOD entity[2].
RBODY Type 1	CONN3D2, **CONNECTOR SECTION* [PROJECTION CARTESIAN + PROJECTION FLEXION-TORSION], **CONNECTOR DAMAGE INITIATION*, and **CONNECTOR DAMAGE EVOLUTION*
CNTAC Sliding interface types: 33, 34, 36, 37, 46	**CONTACT*, **CONTACT INCLUSIONS*, **CONTACT EXCLUSIONS*, **CONTACT PROPERTY ASSIGNMENT*, **CONTACT FORMULATION*, **SURFACE INTERACTION*, and **SURFACE PROPERTY ASSIGNMENT*
TIED	**TIE*
[1] If PART entities are used to define a rigid body, RBODY is translated as **RIGID BODY*.	
[2] If the ELE and NOD entities constitute all elements in a part, RBODY is translated as **RIGID BODY*. If the ELE and NOD entities do not constitute all elements in a part (i.e., if the part consists of both rigid and deformable elements), RBODY is translated as **MPC* (MPC type BEAM), a beam-type multi-point constraint for the set of nodes that consists of all input NOD entities and nodes extracted from all ELE entities.	

Table 7: Nodes/faces/elements entity selection data.

PAM-CRASH keyword	Abaqus equivalent
ELE	**ELSET*; data for elements to be grouped in a set using **ELSET*
PART	Data for selecting element sets (**ELSET*) already defined
NOD	Data for nodes to be grouped in a set using **NSET*
ELE>NOD	Same procedure as ELE
PART>NOD	Same procedure as PART
DELELE	**ELSET* and **NSET*
DELPART	**ELSET* and **NSET*
DELNOD	**ELSET* and **NSET*
GRP	Named set of entities defined in GROUP

Table 8: Airbag data.

PAM-CRASH keyword	Abaqus equivalent
GASPEC	**FLUID BEHAVIOR*, **MOLECULAR WEIGHT*, and **CAPACITY*
BAGIN	**PHYSICAL CONSTANTS* and **FLUID CAVITY*
GEN_INI_COND	**INITIAL CONDITIONS*
GAS	**FLUID CAVITY*, BEHAVIOR or MIXTURE
CHAMBER	**NODE*, NSET=*ref_node_name*; **SURFACE*, TYPE=ELEMENT; and **FLUID CAVITY*
EXT_SKIN	M3D3/M3D4 and **SURFACE*, TYPE=ELEMENT
WALL_OPENING	**FLUID EXCHANGE*, **FLUID EXCHANGE ACTIVATION*, and **FLUID EXCHANGE PROPERTY*
WALL_FABRIC	**FLUID EXCHANGE*, **FLUID EXCHANGE ACTIVATION*, and **FLUID EXCHANGE PROPERTY*
LEAKAGE	**FLUID EXCHANGE*, **FLUID EXCHANGE ACTIVATION*, and **FLUID EXCHANGE PROPERTY*
INI_COND	**INITIAL CONDITIONS*
INFLATOR	**FLUID INFLATOR*, **FLUID INFLATOR ACTIVATION*, **FLUID INFLATOR MIXTURE*, and **FLUID INFLATOR PROPERTY*

Table 9: Seat belt data.

PAM-CRASH keyword	Abaqus equivalent
SLIPR	**ELEMENT*, TYPE=CONN3D2; **CONNECTOR SECTION*; and **BOUNDARY*
RETRA	**ELEMENT*, TYPE=CONN3D2; **CONNECTOR SECTION*; and **BOUNDARY*

Table 10: Miscellaneous data.

PAM-CRASH keyword	Abaqus equivalent
GROUP	Convert entities to Abaqus equivalents
METRIC	**INITIAL CONDITIONS*, TYPE=REF COORDINATE

PAM-CRASH keyword	Abaqus equivalent
SENSOR	Type-1: use activation time in **AMPLITUDE* Type-4: use belt feed rate in **CONNECTOR LOCK*
FUNCT	Data for material properties and time-dependent parameters, such as **AMPLITUDE*, **CONNECTOR ELASTICITY*, **PLASTIC*, and **FLUID EXCHANGE PROPERTY*
RUPMO	Data for connector behavior, such as **CONNECTOR DAMAGE INITIATION*, **CONNECTOR DAMAGE EVOLUTION*, **CONNECTOR POTENTIAL*, and **CONNECTOR HARDENING*
THELE	Element sets defined as **ELSET*; output quantities are not specified for the element set
THNOD	Node sets defined as **NSET*; output quantities are not specified for the node set

Command Summary

abaqus frompamcrash **job =** *job-name*
input = *input-file*
[pLinkConnectors = {`OFF` | `ON`}**]**
[splitAirbagElements = {`OFF` | `ON`}**]**
[autoKJoinStops = {`OFF` | `ON`}**]**

Command Line Options

job This option is used to specify the name of the Abaqus input file to be output by the translator. The name of the Abaqus input file must be given without the `.inp` extension. Diagnostics created by the translator are written to a file named *job-name*`_frompam.log`.

input This option is used to specify the name of the file containing the PAM-CRASH data. The name of the file must be given with the file extension.

pLinkConnectors This option is used to specify the inclusion of connector elements in the PLINK translation. The default value is `ON`.

splitAirbagElements This option is used to specify the splitting of 4-node airbag membrane elements into two 3-node airbag membrane elements. The default value is `ON`. Airbag membrane elements result from the translation of MEMBR and MATER / Types 150 and 151. This option is valid only if the keyword BAGIN is specified in the PAM-CRASH input file.

autoKJoinStops This option is used to add connector stops to the behavior of all KJOIN connector elements. If the stiffness interpolated at an endpoint on the force-displacement curve exceeds the stiffness interpolated at an adjacent point by a factor of 10, a connector stop is defined at the point adjacent to the endpoint. The default value is `OFF`.

Translating RADIOSS Input Files to Partial Abaqus Input Files

Products: Abaqus/Explicit

References:

- *About Execution Procedures*

Overview

The translator from RADIOSS to Abaqus converts certain keywords in a RADIOSS input file into their equivalent in Abaqus/Explicit.

Using the Translator

The translator requires an input file in block format created by RADIOSS Version 4.4 or 5.1. The input file can have any name and an optional extension.

The RADIOSS data entries that are translated are listed in the tables below. Other RADIOSS keywords and data are skipped over and noted in the log file.

The translator creates a partial Abaqus input file that contains only the model data and time history output data. You can provide additional output data to complete the input.

Element Numbering and Grouping

All elements in the generated Abaqus input file will have unique element numbers. New element numbers will be assigned automatically to elements with nonunique element numbers in the RADIOSS input. Elements that are assigned the same PART identification number are grouped together in an element set.

Elements that have different material or properties must be given different PART identification numbers; that is, the same material and properties must be applicable to all elements grouped in the same element set.

If elements that result from the translation of SPRING have different element properties (such as skew systems used to define local directions) and are assigned the same PART identification number, the translator automatically separates them into different element sets.

Material Models

The translator supports only the material models shown in *Table 1*. All unsupported material models are translated as linear elastic if a stress-strain law definition is required. In these cases the translator provides nominal values for the material properties.

Include Files

The translator supports the use of *#include* to include an individual data file. Multiple files can be included. The translator reads the data in all the include files sequentially, processes the data altogether, and translates the data collectively into one Abaqus input file.

Summary of RADIOSS Entities Translated

Table 1: Material data.

RADIOSS keyword	Abaqus equivalent
MAT / LAW01 (ELAST)	**ELASTIC*
MAT / LAW02 (PLAS_JOHN)	**PLASTIC*, HARDENING=JOHNSON COOK
MAT / LAW03 (HYDPLA)	**EOS*, **TENSILE FAILURE*, **DAMAGE INITIATION*, and **DAMAGE EVOLUTION*
MAT / LAW19 (FABRI)	**USER MATERIAL*
MAT / LAW22 (DAMA)	**PLASTIC*, HARDENING=JOHNSON COOK; **RATE DEPENDENT*, TYPE=JOHNSON COOK; **DAMAGE INITIATION*; and **DAMAGE EVOLUTION*
MAT / LAW35 (FOAM_VISC)	**HYPERFOAM* and **VISCOELASTIC*
MAT / LAW36 (PLAS_TAB)	**PLASTIC*, HARDENING=ISOTROPIC

Table 2: Property data.

RADIOSS keyword	Abaqus equivalent
PROP / TRUS	Truss element properties and grouping data
PROP / BEAM	Beam element properties and grouping data
PROP / SPRING	Connector behavior and grouping data
PROP / SPR_BEAM	Connector behavior and grouping data
PROP / SPR_GENE	Connector behavior and grouping data
PROP / SOLID	Solid element properties and grouping data
PROP / SOL_ORTH	Solid element properties and grouping data
PROP / SHELL	Shell element properties and grouping data
PROP / SH_ORTH	Shell element properties and grouping data

Table 3: Nodal data.

RADIOSS keyword	Abaqus equivalent
NODE	**NODE*
ADMAS	**MASS* and **ROTARY INERTIA*
BCS	**BOUNDARY*
IMPDISP	**BOUNDARY* and **AMPLITUDE*
IMPVEL	**BOUNDARY* and **AMPLITUDE*
INIVEL	**INITIAL CONDITIONS*, TYPE=VELOCITY or ROTATING VELOCITY
CLOAD	**CLOAD* and **AMPLITUDE*
GRAV	**DLOAD* and **AMPLITUDE*
SKEW	**ORIENTATION* and **TRANSFORM*
FRAME	**ORIENTATION* and **TRANSFORM*

Table 4: Element data.

RADIOSS keyword	Abaqus equivalent
BRICK	C3D4/C3D6/C3D8R and **SOLID SECTION*
TETRA4	C3D4 and **SOLID SECTION*
SHELL[1]	S3RS/S4RS and **SHELL SECTION*; or M3D3/M3D4/M3D4R and **MEMBRANE SECTION*
SH3N[1]	S3RS and **SHELL SECTION*; or M3D3 and **MEMBRANE SECTION*
BEAM	B31 and **BEAM SECTION*, SECTION=CIRC
TRUSS	T3D2 and **SOLID SECTION*
SPRING	CONN3D2 and **CONNECTOR SECTION*
[1]Shell elements with one integration point through the thickness are translated as membrane elements.	

Table 5: Constraint data.

RADIOSS keyword	Abaqus equivalent
RWALL	**RIGID BODY* and **CONTACT*
RBODY	**RIGID BODY* and/or **MPC* (type BEAM) To define an element as a rigid body, enter all the element node numbers in the node group associated with the rigid body.
INTER / Type 2	**TIE* and **FASTENER*
INTER / Types 7, 10, 11	**CONTACT*, **CONTACT CONTROLS ASSIGNMENT*, **CONTACT FORMULATION*, **CONTACT INCLUSIONS*, **CONTACT EXCLUSIONS*, **CONTACT PROPERTY ASSIGNMENT*, **SURFACE INTERACTION*, and **SURFACE PROPERTY ASSIGNMENT*
CYL_JOINT	CONN3D2 and **CONNECTOR SECTION*

Table 6: Group data.

RADIOSS keyword	Abaqus equivalent
SUBSET	**ELSET*; data for elements to be grouped in a set using **ELSET*
PART	**ELSET*; data for elements to be grouped in a set using **ELSET*
MAT	**ELSET*; data for elements to be grouped in a set using **ELSET*
PROP	**ELSET*; data for elements to be grouped in a set using **ELSET*
NODE	**NSET*; data for elements to be grouped in a set using **NSET*
SH3N	**ELSET*; data for elements to be grouped in a set using **ELSET*
SHEL	**ELSET*; data for elements to be grouped in a set using **ELSET*
GRNOD	**NSET*; data for elements to be grouped in a set using **NSET*
GRSH3N	**ELSET*; data for elements to be grouped in a set using **ELSET*
GRSHEL	**ELSET*; data for elements to be grouped in a set using **ELSET*
GRSPRI	**ELSET*; data for elements to be grouped in a set using **ELSET*
GENE	**NSET*; data for elements to be grouped in a set using **NSET*
SEG	**ELSET*; data for elements to be grouped in a set using **ELSET*
SURF	**ELSET* and **NSET*

Table 7: Monitored volume and seat belt data.

RADIOSS keyword	Abaqus equivalent
MONVOL / GAS MONVOL / AIRBAG	**FLUID BEHAVIOR*, **FLUID CAVITY*, **FLUID EXCHANGE*, **FLUID EXCHANGE ACTIVATION*, **FLUID EXCHANGE PROPERTY*, **FLUID INFLATOR*, **FLUID INFLATOR ACTIVATION*, **FLUID INFLATOR MIXTURE*, **FLUID INFLATOR PROPERTY*, **MOLECULAR WEIGHT*, **CAPACITY*, and **PHYSICAL CONSTANTS*
SPRING with property SPR_PUL	**ELEMENT*, TYPE=CONN3D2; **CONNECTOR SECTION*; and **BOUNDARY*

Table 8: Miscellaneous data.

RADIOSS keyword	Abaqus equivalent
TITLE	**HEADING*
ACCEL	CONN3D2 and connector type ACCELEROMETER
FUNCT	Data for material properties and time-dependent parameters, such as **AMPLITUDE*, **CONNECTOR ELASTICITY*, **PLASTIC*, and **FLUID EXCHANGE PROPERTY*
SECT	**INTEGRATED OUTPUT SECTION*
SENSOR / Type 0	Use activation time in **AMPLITUDE*
TH	Data for time history output, such as **OUTPUT*, HISTORY; **NODE OUTPUT*; **ELEMENT OUTPUT*; and **ENERGY OUTPUT*

Command Summary

abaqus fromradioss

job = *job-name*
input = *input-file*
[splitAirbagElements = {`OFF` | `ON`}**]**
[readAbaqusDat = *data-file***]**
[userDefaultMass = *real-number***]**
[userDefaultInertia = *real-number***]**
[userHistoryTime = *real-number***]**

Command Line Options

job
This option is used to specify the name of the Abaqus input file to be output by the translator. The name of the Abaqus input file must be given without the `.inp` extension. Diagnostics created by the translator are written to a file named *job-name*`_fromradioss.log`.

input
This option is used to specify the name of the file containing the RADIOSS data. The file extension is optional.

splitAirbagElements
This option is used to specify the splitting of 4-node airbag membrane elements into two 3-node airbag membrane elements. The default value is `ON`. Airbag membrane elements result from the translation of SHELL or SH3N with one integration point through the thickness. This option is valid only if the keyword MONVOL/AIRBAG is specified in the RADIOSS input file.

readAbaqusDat
This option enables the use of an Abaqus data (`.dat`) file from a previous Abaqus analysis to reformulate spot weld definitions. The data file should identify spot welds

that could not be formed. Using this option, the attachment points for the identified spot welds are translated using distributed coupling constraints.

userDefaultMass This option is used to specify the nodal mass that is assigned to additional nodes generated during the translation that require nonzero mass. This value should be small (typically 10^{-6} times the mass for the entire model). If this option is omitted, the default mass is set to 10^{-4}.

userDefaultInertia This option is used to specify the rotary inertia that is assigned to additional nodes generated during the translation that require nonzero rotary inertia. This value should be small (typically 10^{-6} times the inertia for the entire model). If this option is omitted, the default rotary inertia is set to 10^{-3}.

userHistoryTime This option is used to specify the time interval used for time history output. If this option is omitted, the time history interval is set to 10^{-5}.

Translating LS-DYNA Data Files to Abaqus Input Files

Products: Abaqus/Explicit

References:

- *About Execution Procedures*

Overview

The translator from LS-DYNA to Abaqus converts a set of supported keywords in an LS-DYNA input file into their equivalent in Abaqus.

Using the Translator

The translator supports translation of input files created by LS-DYNA Version 971 Rev 5 or earlier. The input file can have any name and an optional extension.

The LS-DYNA keywords that are supported are listed in the tables below. Other LS-DYNA keywords and data are skipped over and noted in the log file.

The translator creates an Abaqus input file that contains both the model data and history data. However, the translator does not create exact Abaqus equivalents for specific output quantities for nodal output, element output, and contact output; it uses preselected variables instead. You can provide additional output entities to complete the requests.

Element Numbering and Grouping

All elements in the generated Abaqus input file have unique element numbers. New element numbers are assigned automatically to elements with nonunique element numbers in the LS-DYNA input; all element number reassignments are noted in the log file.

Elements that are assigned the same PART identification number are grouped together in an element set. Elements that have different material or properties must be given different PART identification numbers; that is, the same material and properties must be applicable to all elements grouped in the same element set.

When a PART references a rigid material, the part is considered rigid. The element set that corresponds to the part is used in the rigid body definition.

Material Models

The translator supports only the material models shown in *Table 1*. All unsupported material models are translated as linear elastic if a stress-strain law definition is required. In these cases, the translator provides nominal values for the material properties.

Mapping LS-DYNA Elements That End in _ID or _TITLE

Many LS-DYNA keywords include the options _ID, _TITLE, or both of these options. Unless the LS-DYNA keyword with _ID or _TITLE is specified in the mapping tables in this document, the translator maps data from these options to the same Abaqus keywords specified for the main LS-DYNA keyword.

Reading Integer Data With the i10 Format

The translator supports the following specification of the i10 format in reading integer data:

- Activate the option i10=ON in the execution command of the translator. This corresponds to a global specification applicable to all keywords in the input file that support usage of the format.
- Alternatively, the following keyword option in the input file will trigger global specification of the i10 format:

  ```
  *KEYWORD i10=y
  ```

- If the symbol, %, appears at the end of an individual keyword in the input file, all following cards will use the i10 format up to next keyword. For example,

  ```
  *NODE %
  ```

  specifies a read format of (i10,3f16,2i10) instead of the standard (i8,3f16,2i8) for all nodal data input under that *NODE keyword.

Additional Information

The LS-DYNA keyword *PART_CONTACT listed in *Table 2* is always used in conjunction with the contact keywords listed in *Table 12*. The translation of the contact keywords results in **CONTACT* and **CONTACT INCLUSIONS* in the Abaqus input, and these contact keywords are not listed in *Table 2*.

Summary of LS-DYNA Entities Translated

The translator from LS-DYNA to Abaqus supports the mappings shown in the tables below.

Table 1: Material data.

LS-DYNA Keyword	Abaqus Equivalent
*MAT_ANISOTROPIC_VISCOPLASTIC	**ELASTIC*
	**PLASTIC*
	**RATE DEPENDENT*
*MAT_BLATZ-KO_RUBBER	**HYPERELASTIC*, NEO HOOKE
*MAT_CABLE_DISCRETE_BEAM	**ELASTIC*
*MAT_DAMPER_NONLINEAR_VISCOUS	**CONNECTOR DAMPING*, NONLINEAR
*MAT_DAMPER_VISCOUS	**CONNECTOR DAMPING*
*MAT_ELASTIC	**ELASTIC*
*MAT_ELASTIC_PLASTIC_THERMAL	**ELASTIC*
	**PLASTIC*
	**EXPANSION*
*MAT_FABRIC	**FABRIC*
	**UNIAXIAL*
	**LOADING DATA*
*MAT_FU_CHANG_FOAM	**LOW DENSITY FOAM* and **UNIAXIAL TEST DATA*
*MAT_HONEYCOMB	Built-in *VUMAT* user material model ABQ_HONEYCOMB[1]
*MAT_JOHNSON_COOK	**PLASTIC*, HARDENING=JOHNSON COOK
	**RATE DEPENDENT*, TYPE=JOHNSON COOK
	**SHEAR FAILURE*, TYPE=JOHNSON COOK
	**TENSILE FAILURE*, TYPE=JOHNSON COOK

LS-DYNA Keyword	Abaqus Equivalent
*MAT_LINEAR_ELASTIC_DISCRETE_BEAM	**CONNECTOR ELASTICITY* and **CONNECTOR DAMPING*
*MAT_LOW_DENSITY_FOAM	**HYPERFOAM* and **UNIAXIAL TEST DATA*
*MAT_NONLINEAR_ELASTIC_DISCRETE_BEAM	**CONNECTOR ELASTICITY* and **CONNECTOR DAMPING*
*MAT_NULL	**ELASTIC*
	Shell elements that reference a null material are translated as surface elements
*MAT_OGDEN_RUBBER	**HYPERELASTIC*, OGDEN
*MAT_PIECEWISE_LINEAR_PLASTICITY	**PLASTIC*, HARDENING=ISOTROPIC
	**RATE DEPENDENT*
*MAT_PLASTIC_KINEMATIC	**PLASTIC*, HARDENING=KINEMATIC
*MAT_RIGID	**ELASTIC*
	**RIGID BODY* (for LS-DYNA parts that refer to a rigid material)
*MAT_SEATBELT	**CONNECTOR ELASTICITY*, NONLINEAR
*MAT_SPOTWELD	**CONNECTOR ELASTICITY*, RIGID
*MAT_SPRING_ELASTIC	**CONNECTOR ELASTICITY*
*MAT_SPRING_GENERAL_NONLINEAR	**CONNECTOR ELASTICITY*
*MAT_SPRING_NONLINEAR_ELASTIC	**CONNECTOR ELASTICITY*, NONLINEAR
*MAT_VISCOELASTIC	**VISCOELASTIC*, TIME=PRONY
[1] For more information about using ABQ_HONEYCOMB, refer to “Abaqus/Explicit honeycomb material model,” which is available in the Dassault Systèmes Knowledge Base at *https://support.3ds.com/knowledge-base/*.	

Table 2: Part data.

LS-DYNA Keyword	Abaqus Equivalent
*PART	**ELSET* and the corresponding type of element section
*PART_CONTACT	**ELSET* and the corresponding type of element section
	**SURFACE INTERACTION*, PROPERTIES
	**SURFACE PROPERTY ASSIGNMENT*, PROPERTY=THICKNESS
	**SURFACE PROPERTY ASSIGNMENT*, PROPERTY=FRICTION
*PART_INERTIA	**ELEMENT*, TYPE=MASS
	**ELEMENT*, TYPE=ROTARYI
*PART_PRINT	**ELSET* and the corresponding type of element section

Table 3: Auxiliary data.

LS-DYNA Keyword	Abaqus Equivalent
*DEFINE_COORDINATE_NODES	**ORIENTATION*, DEFINITION=NODES
*DEFINE_COORDINATE_SYSTEM	**ORIENTATION*, DEFINITION=COORDINATES
*DEFINE_COORDINATE_VECTOR	**ORIENTATION*, DEFINITION=COORDINATES
*DEFINE_CURVE	Data from a single curve used in the following keywords:

LS-DYNA Keyword	Abaqus Equivalent
	**AMPLITUDE*
	**CONNECTOR DAMPING* (nonlinear)
	**CONNECTOR ELASTICITY* (nonlinear)
	**SURFACE BEHAVIOR*
	**UNIAXIAL TEST DATA*
*DEFINE_SD_ORIENTATION	**ORIENTATION*
*DEFINE_TABLE	Multi-curve data used in conjunction with **PLASTIC* and **LOW DENSITY FOAM* in which the stress-strain relationship is defined for various strain rates

Table 4: Section data.

LS-DYNA Keyword	Abaqus Equivalent
*SECTION_BEAM	Beam elements: **BEAM SECTION* or **BEAM GENERAL SECTION*
	Truss elements: **SOLID SECTION*
*SECTION_DISCRETE	**CONNECTOR SECTION*
*SECTION_SEATBELT	**CONNECTOR SECTION*
*SECTION_SHELL	Shell elements: **SHELL SECTION*
	Membrane elements: **MEMBRANE SECTION*
	Surface elements: **SURFACE SECTION*
*SECTION_SOLID	**SOLID SECTION*
*SECTION_TSHELL	**SHELL SECTION*

Table 5: Nodal data.

LS-DYNA Keyword	Abaqus Equivalent
*NODE	**NODE*

Table 6: Output options data.

LS-DYNA Keyword	Abaqus Equivalent
*DATABASE_BINARY_D3PLOT	**OUTPUT*, FIELD and **ELEMENT OUTPUT*
*DATABASE_BINARY_D3THDT	**OUTPUT*, FIELD and **ELEMENT OUTPUT*
*DATABASE_DEFORC	**OUTPUT*, FIELD and **ELEMENT OUTPUT*
*DATABASE_ELOUT	**OUTPUT*, FIELD and **ELEMENT OUTPUT*
*DATABASE_HISTORY_BEAM	**OUTPUT*, HISTORY and **ENERGY OUTPUT*
*DATABASE_HISTORY_BEAM_ID	
*DATABASE_HISTORY_BEAM_SET	
*DATABASE_HISTORY_DISCRETE	**OUTPUT*, HISTORY and **ENERGY OUTPUT*
*DATABASE_HISTORY_DISCRETE_ID	
*DATABASE_HISTORY_DISCRETE_SET	

LS-DYNA Keyword	Abaqus Equivalent
*DATABASE_HISTORY_NODE	**OUTPUT*, HISTORY
*DATABASE_HISTORY_NODE_ID	
*DATABASE_HISTORY_NODE_SET	
*DATABASE_HISTORY_SHELL	**OUTPUT*, HISTORY and **ENERGY OUTPUT*
*DATABASE_HISTORY_SHELL_ID	
*DATABASE_HISTORY_SHELL_SET	
*DATABASE_HISTORY_SOLID	**OUTPUT*, HISTORY and **ENERGY OUTPUT*
*DATABASE_HISTORY_SOLID_ID	
*DATABASE_HISTORY_SOLID_SET	
*DATABASE_HISTORY_TSHELL	**OUTPUT*, HISTORY and **ENERGY OUTPUT*
*DATABASE_HISTORY_TSHELL_ID	
*DATABASE_HISTORY_TSHELL_SET	
*DATABASE_NODOUT	**OUTPUT*, FIELD and **NODE OUTPUT*

Table 7: Element data.

LS-DYNA Keyword	Abaqus Equivalent
*ELEMENT_BEAM	Beam elements: **ELEMENT*, TYPE=B31
	Truss elements: **ELEMENT*, TYPE=T3D2
*ELEMENT_BEAM_PID	**ELEMENT*, TYPE=CONN3D2 and **FASTENER*
*ELEMENT_DISCRETE	**ELEMENT*, TYPE=CONN3D2
*ELEMENT_MASS	**ELEMENT*, TYPE=MASS and **MASS*
*ELEMENT_MASS_PART	**NONSTRUCTURAL MASS*, UNITS=TOTAL MASS
	**MASS ADJUST*
*ELEMENT_MASS_PART_SET	**NONSTRUCTURAL MASS*, UNITS=TOTAL MASS
	**MASS ADJUST*
*ELEMENT_SEATBELT	**ELEMENT*, TYPE=CONN3D2
*ELEMENT_SEATBELT_ACCELEROMETER	**ELEMENT*, TYPE=CONN3D2
*ELEMENT_SHELL	Shell elements: **ELEMENT*, TYPE=S3R or S4R
	Membrane elements: **ELEMENT*, TYPE=M3D3 or M3D4R
	Surface elements (with *MAT_NULL): **ELEMENT*, TYPE=SFM3D3 or SFM3D4
*ELEMENT_SHELL_BETA	**ELEMENT*, TYPE=S3R or S4R
	**SHELL SECTION* with orientation definition
	**DISTRIBUTION*
	**ORIENTATION*, DEFINITION=OFFSET TO NODES
*ELEMENT_SHELL_THICKNESS	**ELEMENT*, TYPE=S3R or S4R
	**SHELL SECTION*, NODAL THICKNESS, and orientation definition
	**NODAL THICKNESS*
	**DISTRIBUTION*

LS-DYNA Keyword	Abaqus Equivalent
	**ORIENTATION*, DEFINITION=OFFSET TO NODES
*ELEMENT_SOLID	**ELEMENT*, TYPE=C3D4, C3D6, C3D8R, or C3D10M
*ELEMENT_TSHELL	**ELEMENT*, TYPE=SC6R or SC8R

Table 8: Prescribed conditions data.

LS-DYNA Keyword	Abaqus Equivalent
*BOUNDARY_PRESCRIBED_MOTION_NODE	**BOUNDARY*, TYPE=DISPLACEMENT, VELOCITY, or ACCELERATION
*BOUNDARY_PRESCRIBED_MOTION_RIGID	**BOUNDARY* for reference node of rigid body
*BOUNDARY_PRESCRIBED_MOTION_RIGID_LOCAL	**BOUNDARY* for reference node of rigid body
*BOUNDARY_PRESCRIBED_MOTION_SET	**BOUNDARY*, TYPE=DISPLACEMENT, VELOCITY, or ACCELERATION
*BOUNDARY_SPC_NODE	**BOUNDARY*
*BOUNDARY_SPC_SET	**BOUNDARY*
*INITIAL_STRESS_SHELL	**INITIAL CONDITIONS*, TYPE=HARDENING, SECTION POINTS
	**INITIAL CONDITIONS*, TYPE=STRESS, SECTION POINTS
*INITIAL_VELOCITY	**INITIAL CONDITIONS*, TYPE=VELOCITY
*INITIAL_VELOCITY_GENERATION	**INITIAL CONDITIONS*, TYPE=ROTATING VELOCITY
*INITIAL_VELOCITY_NODE	**INITIAL CONDITIONS*, TYPE=VELOCITY

Table 9: Miscellaneous constraints data.

LS-DYNA Keyword	Abaqus Equivalent
*CONSTRAINED_EXTRA_NODES_NODE	Node set used as TIE NSET in the definition of **RIGID BODY*
*CONSTRAINED_EXTRA_NODES_SET	Node set used as TIE NSET in the definition of **RIGID BODY*
*CONSTRAINED_JOINT_CYLINDRICAL	**ELEMENT*, TYPE=CONN3D2
*CONSTRAINED_JOINT_REVOLUTE	**ELEMENT*, TYPE=CONN3D2
*CONSTRAINED_JOINT_SPHERICAL	**ELEMENT*, TYPE=CONN3D2
*CONSTRAINED_JOINT_SPHERICAL_LOCAL	**ELEMENT*, TYPE=CONN3D2
*CONSTRAINED_JOINT_STIFFNESS_GENERALIZED	**ELEMENT*, TYPE=CONN3D2
	**CONNECTOR SECTION*, BEHAVIOR
*CONSTRAINED_JOINT_STIFFNESS_TRANSLATIONAL	**ELEMENT*, TYPE=CONN3D2
	**CONNECTOR SECTION*, BEHAVIOR
*CONSTRAINED_JOINT_TRANSLATIONAL	**ELEMENT*, TYPE=CONN3D2
*CONSTRAINED_JOINT_UNIVERSAL	**ELEMENT*, TYPE=CONN3D2
*CONSTRAINED_NODAL_RIGID_BODY	**RIGID BODY* with TIE NSET for the case of no release of displacements or rotations
	**COUPLING* and **KINEMATIC* for the case with release of displacements or rotations
*CONSTRAINED_NODE_SET	**EQUATION*

LS-DYNA Keyword	Abaqus Equivalent
*CONSTRAINED_RIGID_BODIES	Merged element set used in the definition of **RIGID BODY*
*CONSTRAINED_SPOTWELD	**MPC* type BEAM

Table 10: Load data.

LS-DYNA Keyword	Abaqus Equivalent
*LOAD_BODY_PARTS	**ELSET* for **DLOAD*
*LOAD_BODY_X	**DLOAD*
*LOAD_BODY_Y	**DLOAD*
*LOAD_BODY_Z	**DLOAD*
*LOAD_NODE_POINT	**CLOAD* with node data
*LOAD_NODE_SET	**CLOAD* with node set data

Table 11: Set data.

LS-DYNA Keyword	Abaqus Equivalent
*SET_NODE_ADD	**NSET* with node set data
*SET_NODE_LIST	**NSET* with node data
*SET_NODE_LIST_GENERATE	**NSET* with node data
*SET_PART	**ELSET* with element set data
*SET_PART_ADD	**ELSET* with element set data
*SET_PART_LIST	**ELSET* with element set data
*SET_PART_LIST_GENERATE	**ELSET* with element set data
*SET_SEGMENT	**ELSET* with element data
*SET_SHELL_LIST	**ELSET* with element data
*SET_SHELL_LIST_GENERATE	**ELSET* with element data
*SET_SOLID_LIST	**ELSET* with element data

Table 12: Contact data.

LS-DYNA Keyword	Abaqus Equivalent
*CONTACT_AUTOMATIC_GENERAL	**CONTACT*
	**CONTACT INCLUSIONS*
	**CONTACT PROPERTY ASSIGNMENT*
	**SURFACE INTERACTION*
	**SURFACE PROPERTY ASSIGNMENT*
*CONTACT_AUTOMATIC_NODES_TO_SURFACE	**CONTACT*
	**CONTACT INCLUSIONS*
	**CONTACT PROPERTY ASSIGNMENT*
	**SURFACE INTERACTION*
	**SURFACE PROPERTY ASSIGNMENT*

LS-DYNA Keyword	Abaqus Equivalent
*CONTACT_AUTOMATIC_SINGLE_SURFACE	**CONTACT*
	**CONTACT INCLUSIONS*
	**CONTACT PROPERTY ASSIGNMENT*
	**SURFACE INTERACTION*
	**SURFACE PROPERTY ASSIGNMENT*
*CONTACT_AUTOMATIC_SURFACE_TO_SURFACE	**CONTACT*
	**CONTACT INCLUSIONS*
	**CONTACT PROPERTY ASSIGNMENT*
	**SURFACE INTERACTION*
	**SURFACE PROPERTY ASSIGNMENT*
*CONTACT_NODES_TO_SURFACE	**CONTACT*
	**CONTACT INCLUSIONS*
	**CONTACT PROPERTY ASSIGNMENT*
	**SURFACE INTERACTION*
	**SURFACE PROPERTY ASSIGNMENT*
*CONTACT_RIGID_NODES_TO_RIGID_BODY	**CONTACT*
	**CONTACT INCLUSIONS*
	**CONTACT PROPERTY ASSIGNMENT*
	**SURFACE INTERACTION*
	**SURFACE PROPERTY ASSIGNMENT*
*CONTACT_SINGLE_SURFACE	**CONTACT*
	**CONTACT INCLUSIONS*
	**CONTACT PROPERTY ASSIGNMENT*
	**SURFACE INTERACTION*
	**SURFACE PROPERTY ASSIGNMENT*
*CONTACT_SURFACE_TO_SURFACE	**CONTACT*
	**CONTACT INCLUSIONS*
	**CONTACT PROPERTY ASSIGNMENT*
	**SURFACE INTERACTION*
	**SURFACE PROPERTY ASSIGNMENT*
*CONTACT_TIED_NODES_TO_SURFACE	**TIE*
*CONTACT_TIED_SHELL_EDGE_TO_SURFACE	**TIE*
*CONTACT_TIED_SURFACE_TO_SURFACE	**TIE*

Table 13: Miscellaneous data.

LS-DYNA Keyword	Abaqus Equivalent
*CONTROL_BULK_VISCOSITY	**BULK VISCOSITY*
*CONTROL_TERMINATION	Time period entered in **DYNAMIC*, EXPLICIT

LS-DYNA Keyword	Abaqus Equivalent
*END STEP	**END STEP*
*INCLUDE	Process multiple LS-DYNA files
*KEYWORD	None
*PARAMETER	**PARAMETER*[1]
*PARAMETER_EXPRESSION	**PARAMETER*[1]
*TITLE	**HEADING*
[1]Translation of keyword data in terms of parameters is supported for LS-DYNA *INITIAL family of keywords given in *Table 8*	

Command Summary

abaqus fromdyna

job = *job-name*
input = *dyna-input-file*
[splitFile = {OFF | ON**}]**
[i10 = {OFF | ON**}]**

Command Line Options

job This option is used to specify the name of the Abaqus input file to be output by the translator. The name of the Abaqus input file must be given without the `.inp` extension. Diagnostics created by the translator are written to a file named *job-name*`.log`.

input This option is used to specify the name of the file containing the LS-DYNA keyword data. The LS-DYNA input file can have an extension.

splitFile This option specifies whether the Abaqus input file is to be split into multiple files. The default value is `OFF`. If **splitFile=**`ON`, the following files are output:

- `job-name_nodes.inc`: include file that contains the nodal data
- `job-name_elements.inc`: include file that contains the element data
- `job-name_model.inc`: include file that contains the remaining model data
- `job-name.inp`: Abaqus input file that includes all of the above include files and the history data

i10 This option specifies whether the translator should use i10 format in reading integer data for all keywords in the input file for which use of the format is supported. The default value is `OFF`.

Exchanging Abaqus Data with ZAERO

Products: Abaqus/Standard

References:

- *About Execution Procedures*

Overview

The **abaqus tozaero** interface enables you to exchange aeroelastic data between the Abaqus and ZAERO analysis products. By using this interface between the applications, you can perform structural modal analysis on a model in Abaqus, transfer the model to ZAERO for aeroelastic analysis, then transfer it back to Abaqus for stress analysis.

Universal File

The universal file is the means of data exchange between Abaqus and ZAERO. It consists of four data sets: 2411, which describes node and coordinate data; 2414, which describes mass-normalized mode shapes; 2420, which describes the global coordinate system; and 2453, which describes the mass matrix in text format, or 2453b, which describes the mass matrix in binary format.

You can specify the universal file's output format by using the **mode** parameter. Choosing text format enables you to modify the universal file in a text editor but increases the file size to over twice that of similar files in binary format. Text is the default format and the only format supported by ZAERO. *Table 1* and *Table 2* describe the mass matrix data set text format and binary format, respectively.

Table 1: Format for data set 2453 (text).

Record	Field	Description	Format
1	1	Matrix Identifier 1: DOF 131: Mass 139: Stiffness 147: Back-expansion	(I10)
2	1	Matrix Data Type 1: Integer 2: Real 4: Double Precision 5: Complex 6: Complex Double Precision	(6I10)
	2	Matrix Form 3: General Rectangular	
	3	Number of rows	
	4	Number of columns	
	5	Storage Key 1: Row 2: Column 11: Sparse (not supported for IMAT=1)	
	6	Matrix Size Parameter For IMAT=1 this is the number of dynamic modes. For sparse this is the number of matrix entries. Otherwise, 0.	

Record	Field	Description	Format
3 for storage keys 1 and 2	N/A	Matrix Data	For data type 1: (8 I10) For data type 2: (4 E20.12) For data type 4: (4 D20.12) For data type 5: (2 (2 E20.12)) For data type 6: (2 (2 D20.12))
3 for storage key 11	1	Row	For data type 1: (2 (2I10 1I10)) For data type 2: (2 (2I10 1E20.12)) For data type 4: (2 (2I10 1D20.12)) For data type 5: (1 (2I10 2E20.12)) For data type 6: (1 (2I10 2D20.12))
	2	Column	
	3	Value at cell	

Table 2: Format for data set 2453b (binary).

Record	Field	Description	Format
Header	1	2453	(I6)
	2	Lowercase b	(IA1)
	3	Byte Ordering Method 1: Little Endian	(I6)
	4	Floating Point Format 1: IEEE 754	(I6)
	5	Number of ASCII lines following 2 for data set 2453b	(I12)
	6	Number of bytes following ASCII lines	(I12)
	7–10	Not used (fill with zeros)	
1	1	Matrix Identifier 1: DOF 131: Mass 139: Stiffness 147: Back-expansion	(I10)
2	1	Matrix Data Type 1: Integer 2: Real 4: Double Precision 5: Complex 6: Complex Double Precision	(6I10)
	2	Matrix Form 3: General Rectangular	
	3	Number of rows	
	4	Number of columns	
	5	Storage Key 1: Row 2: Column 11: Sparse (not supported for IMAT=1)	

Record	Field	Description	Format
	6	Matrix Size Parameter For IMAT=1 this is the number of dynamic modes. For sparse this is the number of matrix entries. Otherwise, 0.	
3 (Binary Matrix Data)	1 (4 bytes)	Row	For data type 1: (2 Int32 1 Int32) For data type 2: (2 Int32 1 Flt32) For data type 4: (2 Int32 1 Dbl64) For data type 5: (2 Int32 2 Flt32) For data type 6: (2 Int32 2 Dbl64)
	2 (4 bytes)	Column	
	3	Value at cell	

Preparing the Abaqus Analysis Input File

Before the interface can create the universal file, you must make the following additions to your Abaqus input (`.inp`) file, then run Abaqus:

- Normalize the eigenvectors in the eigenfrequency extraction analysis with respect to the structure's mass matrix. This normalization is necessary because the translator assumes the mode shapes are mass normalized; if you skip this step before the Abaqus run, the modes translated will be incorrect and will give incorrect results with no warnings or errors. For more information, see *Natural Frequency Extraction*.
- Include the following line in the analysis step:

  ```
  *ELEMENT MATRIX OUTPUT, ELSET=allelements, MASS=YES,
  OUTPUT FILE=USER DEFINED, FILE NAME=mtx-file-name
  ```

  where `allelements` is a defined element set containing all the elements that should be included in the global mass matrix. The matrix output will be placed into the file *mtx-file-name* `.mtx`; you should not specify the `.mtx` extension since Abaqus adds it automatically.

Workflow

This section describes the input and output of the three main steps in the workflow between Abaqus and ZAERO.

Modal Analysis in Abaqus

The Abaqus modal analysis uses an Abaqus input file and outputs the following data to an output database (`.odb`) file and matrix (`.mtx`) file: structural model nodes, coordinate systems, mode frequencies, generalized mass, mode shapes, and the mass matrix.

Aeroelastic Analysis in ZAERO

Aeroelastic analysis requires a ZAERO input file and the universal file created by toZAERO. ZAERO outputs force and moment data on structural nodes due to aeroelastic forces to the universal file.

Stress Analysis in Abaqus

The forces and moments output from ZAERO can then be used in a static (linear or nonlinear) Abaqus analysis to calculate deflections, stresses, and loads.

Command Summary

abaqus tozaero **job =** *job-name*
[unvfile = *unv-file-name***]**
[odbfile = *odb-file-name***]**
[mtxfile = *mtx-file-name***]**
[step = *step-number***]**
[mode = {`text` | `binary`**}]**

Command Line Options

job This option is used to specify the name of the Abaqus input file. It is also the default name for the universal output database and mass matrix files.

unvfile This option is used to specify the name of the universal file if it is different from *job-name*. If the `.unv` extension is not supplied, Abaqus adds it automatically.

odbname This option is used to specify the name of the Abaqus output database file if it is different from *job-name*. If the `.odb` extension is not supplied, Abaqus adds it automatically.

mtxfile This option is used to specify the file containing the element mass matrices generated by Abaqus. If the `.mtx` extension is not supplied, Abaqus adds it automatically.

step This option specifies the step number containing the eigenfrequency extraction results from Abaqus. The default value is 1.



Note:

You must normalize the eigenvectors in the eigenfrequency extraction analysis with respect to the structure's mass matrix. For more information, see *Natural Frequency Extraction*.

mode This option specifies the output format of the universal file. If this option is set equal to `binary`, Abaqus writes a portion of the universal file in binary format to save space. If this option is set equal to `text`, Abaqus writes the entire file in all text format. The default value is `text`, which is the only mode currently supported by ZAERO.

Translating a Simpack Flexible Body to Abaqus Matrix Data

Products: Abaqus/Standard

References:

- *About Execution Procedures*

Overview

The **abaqus fromsimpack** translator converts a Simpack flexible body to matrix data in an Abaqus SIM file.

The translator reads flexible body data in a Simpack Flexible Body Interface (FBI) file and writes Abaqus matrix data to a SIM file.

Using the Translator

The translator reads flexible body data defining stiffness, mass, viscous damping, and structural damping operators and creates equivalent matrix data in an AbaqusSIM file.

Command Summary

abaqus fromsimpack	**job =** *job-name* **[fbi =** *fbi-filename***]**

Command Line Options

job This option specifies the input and output file names to use during results translation. The *job-name* value is used to construct the SIM database file name, *job-name*`.sim`. It is also the default name for the flexible body interface file, *job-name*`.fbi`.

If this option is omitted from the command line, you will be prompted for this value.

fbi This option specifies the name of the Simpack Flexible Body Interface (`.fbi`) file if it is different from *job-name*`.fbi`.

Translating Moldflow Data to Abaqus Input Files

Products: Abaqus/Standard

References:

- *About Execution Procedures*
- *Moldflow translation examples*

Overview

Moldflow Plastics Insight (referred to as Moldflow in this section) from Autodesk, Inc. models the plastics injection mold-filling process. The results of a Moldflow simulation include calculations of material properties and residual stresses in the plastic part.

The **abaqus moldflow** translator transforms finite element model information from a Moldflow analysis into a partial Abaqus input file. The translator requires the Moldflow interface files that are created by the Moldflow analysis. (See *The Moldflow Interface Files* for more information.)

For midplane simulations the **abaqus moldflow** translator reads the interface (`.pat` and `.osp`) files created by **abaqus moldflow** translator Version MPI 3 or later.

For three-dimensional solid simulations using Moldflow Version MPI 6 the translator reads the interface (`.inp` and `.xml`) files created using the Visual Basic script `mpi2abq.vbs`. This script is part of an Abaqus installation and is typically found in the *moldflow_install_dir*`/Plastic Insight 6.0/data/commands` directory.

Using the Translator

The following procedure summarizes the typical usage of the **abaqus moldflow** translator:

1. Run a Moldflow simulation.
2. Export the data as follows:
   - For a midplane Moldflow simulation export the finite element mesh data to a file named *job-name*`.pat` and the results data (material properties and residual stresses) to a file named *job-name*`.osp`.
   - For a three-dimensional solid Moldflow simulation using Moldflow Version MPI 6 run the Visual Basic script `mpi2abq.vbs` to export the finite element mesh data to a file named *job-name*`_mesh.inp` and the results data to `.xml` files.
3. Run the **abaqus moldflow** translator to create a partial Abaqus input file from the Moldflow interface files.
4. Edit the Abaqus input file to add appropriate data for the analysis (for example, add boundary conditions and step data).
5. Submit the Abaqus input file for analysis.

The Moldflow Interface Files

The Moldflow interface files contain finite element mesh data, material property data, and residual stress data.

For midplane simulations you must use Moldflow to create two interface files: *job-name*`.pat` and *job-name*`.osp`. Both files must use the same units.

For three-dimensional solid simulations using Moldflow Version MPI 6, the mesh and results files for filled and unfilled models are listed in *Table 1*.

Table 1: Interface files generated using the Visual Basic script for Moldflow Version MPI 6.

Data type	Filled model	Unfilled model
Finite element mesh data	*job-name*_mesh.inp	*job-name*_mesh.inp
Results data	*job-name*_v12.xml	*job-name*_PoissonRatios.xml
	*job-name*_v13.xml	
	*job-name*_v23.xml	
	*job-name*_g12.xml	*job-name*_ShearModuli.xml
	*job-name*_g13.xml	
	*job-name*_g23.xml	
	*job-name*_ltec_1.xml	*job-name*_Ltecs.xml
	*job-name*_ltec_2.xml	
	*job-name*_ltec_3.xml	
	*job-name*_e11.xml	*job-name*_Moduli.xml
	*job-name*_e22.xml	
	*job-name*_e33.xml	
	*job-name*_initStresses.xml	*job-name*_initStresses.xml
	*job-name*_principalDirections.xml	

Finite Element Mesh Data

The Moldflow interface files contain finite element mesh data.

- For midplane simulations the mesh data are in a Patran neutral file containing nodal coordinates, element topology, and element properties.
- For three-dimensional solid simulations the mesh data are in an Abaqus input file containing nodal coordinates, element topology, element properties, and boundary conditions sufficient to eliminate the structure's rigid body modes. Solid elements in the mesh files are always 4-node tetrahedra. The translator has an option to convert these to 10-node tetrahedra.

Material Property Data

The Moldflow interface material property data file contains elastic and thermal expansion coefficients for each element. For midplane simulations these properties may be isotropic or orthotropic. For three-dimensional solid simulations of filled models these properties are orthotropic. For three-dimensional solid simulations of unfilled models, the data files contain orthotropic data adjusted to represent physically isotropic materials.

Residual Stress Data

The **abaqus moldflow** translator calculates residual stresses in the plastic part after it has cooled in the mold. These residual stresses can be translated to initial stresses for the Abaqus structural analysis.

- For midplane simulations a plane stress initial stress state is defined in the same directions as the material properties. The stress state in the material coordinates is defined in terms of the principal stresses (the shear stress is zero).

- For three-dimensional solid simulations residual stresses for each element in *job-name*_`initStresses.xml` are in global coordinates. The translator transforms these coordinates to the same directions as the material properties.

Assumptions Used to Translate the Moldflow Data for Midplane Simulations

For midplane simulations, the **abaqus moldflow** translator makes a number of assumptions regarding the topology and properties of the data. These assumptions, listed below, ensure compatibility with the options available in the current release of Abaqus.

- The Moldflow mesh can consist of 3-node, planar, triangular elements as well as 2-node, one-dimensional elements that represent components such as runners and ribs. The **abaqus moldflow** translator converts the triangular elements to an identical mesh of AbaqusS3R shell elements. One-dimensional elements in the Moldflow mesh are not translated.
- The number of layers in the AbaqusS3R shell elements created by the **abaqus moldflow** translator is equal to the number of layers passed by Moldflow, which is 20. As a result, the mechanical properties and stress data passed to the translator apply to 20 layers through the thickness.
- The Abaqus input data created by the **abaqus moldflow** translator depend on the kind of material defined in the interface (`.osp`) file as follows:
  - For unfilled isotropic materials Abaqus assumes the following:
    - A homogeneous shell formulation.
    - Isotropic material constants.
    - Abaqus section point initial stresses are interpolated from the values at the Moldflow through-thickness integration points.
  - For unfilled transversely isotropic materials Abaqus assumes the following:
    - A homogeneous shell formulation.
    - Transversely isotropic material constants defined for the section in terms of material principal directions plus the orientation with respect to the local Abaqus coordinate system.
    - Abaqus section point initial stresses are interpolated from the values at the Moldflow through-thickness integration points.
  - For fiber-filled materials Abaqus assumes the following:
    - A composite shell formulation.
    - Lamina material constants defined for each layer in terms of material principal directions plus the orientation with respect to the local Abaqus coordinate system for each layer.
    - Moldflow through-thickness integration points are taken as the midpoint of each Abaqus layer.
    - Material properties are constant for each layer.
    - Abaqus section point initial stresses are the same as the values at the Moldflow through-thickness integration points and constant through each layer.

The Abaqus input file that the **abaqus moldflow** translator generates does not contain boundary condition and load data. You must add this information to the input file manually.

Assumptions Used to Translate the Moldflow Data for Three-Dimensional Solid Simulations

For three-dimensional solid simulations, the **abaqus moldflow** translator makes a number of assumptions regarding the topology and properties of the data. These assumptions, listed below, ensure compatibility with the options available in the current release of Abaqus.

- The **abaqus moldflow** translator converts the tetrahedral elements to an identical mesh of AbaqusC3D4 or C3D10 solid elements (for more information, see the command line options below).
- Orthotropic material constants are in terms of material principal directions.
- Material properties are constant for each element.
- Orientations are defined in *job-name*_`principalDirections.xml` by giving vectors defining the local 1- and 2-directions.
- Residual stresses computed by the WARP3D module of Moldflow in *job-name*_`initStresses.xml` are transformed from global coordinates to local material directions and used as initial stresses in Abaqus.
- Loads and boundary conditions representing service loads must be added to the input file manually. For simulations using Moldflow Version MPI 6, the Abaqus input file created by the translator contains boundary conditions sufficient to remove rigid body modes from the model so that an analysis can easily solve for the response due to initial stresses.

Files Created for a Midplane Simulation

The **abaqus moldflow** translator reads the Moldflow interface files and creates the relevant files. The files created depend on which options you include on the command line when executing the translator. For a midplane simulation the **abaqus moldflow** translator creates a partial Abaqus input file, a neutral file, and an initial stress file.

Partial Abaqus Input (`.inp`) File

The partial Abaqus input file contains model data consisting of nodal coordinates, element topology, and section definitions. It also contains a **STATIC* step with default output requests. If you are working with isotropic materials, the input file contains material property data. Each input file begins with a series of comments that summarize the data provided by the Moldflow interface files and how the data are translated to the Abaqus input file. Additional data, such as boundary conditions and loads, and nondefault output requests must be added to this file manually.

Neutral (`.shf`) File Containing Material Data for Layered, Spatially Varying Material Properties

Material data are translated into an appropriately formatted ASCII neutral file. This file contains lamina material property data for each layer of each element. The Abaqus **ELASTIC*, TYPE=SHORT FIBER and **EXPANSION*, TYPE=SHORT FIBER options in the Abaqus input file direct Abaqus/Standard to read material data from this file during the initialization step.

Data lines in the neutral file:

1. First line:
2. Number of elements in the `.shf` file.
3. Number of layers in each shell section.

1. Subsequent lines:
2. Element label.

3. Layer identifier.
4. E_1.
5. E_2.
6. ν_{12}.
7. G_{12}.
8. G_{13}.
9. G_{23}.
10. α_1.
11. α_2.
12. Fiber orientation angle (in degrees), measured relative to the default element orientation.

This data line is repeated as often as necessary to define the above parameters for different layers of a shell section within different elements.

Initial Stress (`.str`) File

Residual stress data from the Moldflow analysis are translated into an appropriately formatted ASCII neutral file. These data are defined in terms of the local Abaqus coordinate system at each section point. The Abaqus**INITIAL CONDITIONS*, TYPE=STRESS, SECTION POINTS option in the Abaqus input file directs Abaqus/Standard to read initial stress data from this file during the initialization step.

Files Created for a Three-Dimensional Solid Simulation

The **abaqus moldflow** translator reads the Moldflow interface files and creates the relevant files. The files created depend on which options you include on the command line when executing the translator.

If you are using an unfilled model, the **abaqus moldflow** translator creates only the partial Abaqus input file described below. For a three-dimensional solid simulation using a filled model, the translator may create additional files, as described below.

Partial Abaqus Input File

The partial Abaqus input file contains model data consisting of nodal coordinates, element topology, and section definitions. Additional data, such as service loads and boundary conditions, and nondefault output requests must be added to this file manually.

Boundary condition data sufficient to remove rigid body modes are also included.

Material (`.mpt`) File Containing Orthotropic Material Properties Data

Material data from the Moldflow analysis are collected and placed in a binary file. The data written to the file are in the same form as the information provided for the Abaqus**ELASTIC*, TYPE=ENGINEERING CONSTANTS option. These are defined in terms of the local Abaqus coordinate system of each element.

Orientation (`.opt`) File Containing Element Orientation Data

Orientations defining the directions for material properties and initial stresses are computed and placed in this binary file.

Thermal Expansion (`.tpt`) File Containing Element Thermal Expansion Coefficient Data

The orthotropic thermal expansion data from the Moldflow analysis are collected and placed in a binary file. These are defined in terms of the local Abaqus coordinate system of each element.

Preparing the Abaqus Input File for Analysis

Once the **abaqus moldflow** translator has created the Abaqus input file, you must complete the input file manually before submitting it for analysis (see *Abaqus Model Definition* for details).

Preparing for a Shrinkage and Warpage Analysis

A shrinkage and warpage analysis calculates the deformation caused by the residual stresses in the model after it is removed from the mold. Usually only rigid body modes must be removed.

In this case you must ensure that residual stresses have been translated. For three-dimensional solid Moldflow simulations boundary conditions sufficient to restrain rigid body modes are automatically translated to the input file. In other cases you are required to add appropriate boundary conditions to remove the rigid body modes of the model.

In certain cases problems with convergence can occur when you must account for geometric nonlinearity and large initial stresses are present. You can overcome these problems by using two analysis steps:

- In the first step, constrain all displacement degrees of freedom.
- In the second step, use the OP=NEW parameter to apply boundary conditions that remove the rigid body modes.

Preparing for a Service Loading Analysis

A service loading analysis (with appropriate boundary conditions) assesses the performance of the model. You can perform this analysis with or without initial stresses. You must specify the appropriate boundary conditions and loads as history data in the Abaqus input file.

Preparing for Other Analysis Types

Any Abaqus/Standard analysis procedure can be performed with the translated model provided that you specify the correct boundary conditions and loading in the Abaqus input file. In addition, certain analysis types may require you to specify additional material constants, model data, and/or solution controls in the input file.

Command Summary

abaqus moldflow

job = *job-name*
[input = *input-name***]**
[midplane] | [3D]
[element_order = {`1` **|** `2`**}]**
[initial_stress = {`on` **|** `off`**}]**
[material = `traditional`**]**
[orientation = `traditional`**]**

Command Line Options

job This option is used to specify the name of the Abaqus input file to be output by the translator. It is also the default name of the file containing the Moldflow data.

If this option is omitted from the command line, the user will be prompted for this value.

input This option is used to specify the name of the files containing the Moldflow interface data if it is different from *job-name*.

midplane This option is used to translate the results of a midplane simulation to an Abaqus model with three-dimensional (shell) elements.

3D This option is used to translate the results of a three-dimensional solid simulation to an Abaqus model with solid elements.

element_order This option is used to specify the order of elements created in the partial input file for three-dimensional solid simulations. Possible values are `1` to create first-order elements (C3D4) and `2` to create second-order elements (C3D10). The default value is `2`. This option is valid only when using the **3D** option.

initial_stress This option specifies whether or not initial stress will be included in the model. This option is valid only when using the **3D** option.

If the **initial_stress** option is not included or if **initial_stress**=`off`, initial stresses will not be translated.

If **initial_stress**=`on`, initial stresses will be written to the input file.

material This option is used to specify where the material properties are written. If **material**=`traditional`, the material properties will be written to the input file. Otherwise, the material properties will be written to the (binary) `.mpt` file. Using **material**=`traditional` is not recommended for large models for performance reasons since every element will have its own **MATERIAL* definition.

orientation This option is used to specify where the orientations are written. If **orientation**=`traditional`, the orientations are written to the input file. Otherwise, the orientations will be written to the (binary) `.opt` file. Using **orientation**=`traditional` is not recommended for large models for performance reasons since every element will have its own **ORIENTATION* definition.

Translating Abaqus Output to Another Format

This section describes using commands to convert content from Abaqus into formats that can be accessed in other solvers.

In this section:

- *Translating Abaqus Files to Nastran Bulk Data Files*
- *Translating Abaqus Output Database Files to Nastran Output2 Results Files*
- *Translating an Abaqus Substructure to an MSC.Adams Flexible Body*
- *Translating an Abaqus Substructure to a Simpack Flexible Body*
- *Translating an Abaqus Substructure to an EXCITE Flexible Body*

Translating Abaqus Files to Nastran Bulk Data Files

The translator from Abaqus to Nastran converts certain entities in an Abaqus file into equivalent entities in Nastran. Only “flat” Abaqus input files can be translated; that is, the Abaqus file cannot contain parts and assemblies.

In this section:

- *Translating an Abaqus Input File to a Partial Nastran Bulk Data Text File*
- *Translating Matrix Data in an AbaqusSIM File to a Partial Nastran Bulk Data Text File*

Translating an Abaqus Input File to a Partial Nastran Bulk Data Text File

Products: Abaqus/Standard Abaqus/Explicit

References:

- *About Execution Procedures*
- *Translating Nastran Bulk Data in Text Files to Partial Abaqus Input Files*

Overview

The Abaqus input data must be in a file with the extension `.inp`. The execution procedure translates selected keywords and creates a Nastran bulk data file with the extension `.bdf`.

Summary of Abaqus Keywords Translated

In the **ELEMENT* usages listed below, an italicized *x* indicates that all Abaqus elements beginning with the preceding label will be mapped to the Nastran entity shown. For example, the statement **ELEMENT*, C3D4*x* indicates that the selected Abaqus-to-Nastran translation applies to the Abaqus elements C3D4, C3D4H, and C3D4T.

Table 1: Abaqus keyword–to–Nastran mapping.

Abaqus Keyword	Nastran Complement
**BEAM GENERAL SECTION*	PBEAM or PBEAML
**BEAM SECTION*	PBEAML
**BOUNDARY*	SPC
**CLOAD*	FORCE
**COUPLING*, DISTRIBUTING	RBE3
**COUPLING*, KINEMATIC	RBE2
**ELEMENT*, B31	CBEAM
**ELEMENT*, B33	CBEAM
**ELEMENT*, C3D4*x*	CTETRA
**ELEMENT*, C3D5	CPYRAM
**ELEMENT*, C3D10*x*	CTETRA
**ELEMENT*, C3D6*x*	CPENTA
**ELEMENT*, C3D15*x*	CPENTA
**ELEMENT*, C3D8*x*	CHEXA
**ELEMENT*, C3D20*x*	CHEXA
**ELEMENT*, MASS	CONM2
**ELEMENT*, ROTARYI	CONM2
**ELEMENT*, S3*x*	CTRIA3
**ELEMENT*, S4*x*	CQUAD4
**ELEMENT*, S8*x*	CQUAD8
**ELEMENT*, SHEAR4	CSHEAR

Abaqus Keyword	Nastran Complement
**ELEMENT*, SPRING1 or SPRING2	CELAS
**ELEMENT*, SPRINGA	CROD
**ELEMENT*, STRI65	CTRIA6
**ELEMENT*, T3D2	CROD
**FREQUENCY*	SOL 103
**HEADING*	TITLE
**MATERIAL*, DENSITY	MAT1
**MATERIAL*, ELASTIC, TYPE=ISO	MAT1
**MATERIAL*, ELASTIC, TYPE=LAMINA	MAT8
**MATERIAL*, EXPANSION, TYPE=ISO	MAT1
**MATERIAL*, EXPANSION, TYPE=ORTHO	MAT8
**MATRIX INPUT*	DMIG
**NODE*	GRID
**ORIENTATION*, DEFINITION=COORDINATES	CORD2R, CORD2C, or CORD2S
**SHELL GENERAL SECTION* (Noncomposite)	PSHELL
**SHELL SECTION* (Noncomposite)	
**SHELL SECTION* (Composite)	PCOMP
**SHELL GENERAL SECTION* (Composite)	
**SOLID SECTION*	PSOLID
**SOLID SECTION* (Trusses)	PROD
**STATIC*	SOL 101
**SYSTEM*	CORD2R, CORD2C, or CORD2S
**TRANSFORM*	

Command Summary

abaqus tonastran **job =** *job-name*
[input = *input-file***]**
[bdf_format = {`DOUBLE` | `FREE`}**]**

Command Line Options

job This option is used to specify the name of the Nastran bulk data file to be output by the translator. It is also the default name of the Abaqus file. Diagnostics created by the translator are written to a file named *job-name*`.log`.

input This option is used to specify the name of the file containing the Abaqus data if it is different from *job-name*.

bdf_format This option determines the format of the Nastran `.bdf` file. If **bdf_format**=`DOUBLE` (default), the file is written in double precision fixed format. If **bdf_format**=`FREE`, the file is written in single precision free format.

Translating Matrix Data in an AbaqusSIM File to a Partial Nastran Bulk Data Text File

Products: Abaqus/Standard Abaqus/Explicit

References:

- *About Execution Procedures*
- *Translating Nastran Bulk Data in Text Files to Partial Abaqus Input Files*

Overview

The Abaqus input data must be in a file with the extension `.sim` and must have been created with either a matrix generation or a substructure generation procedure. When you use the **sim2dmig** option, the execution procedure translates the matrix data to Nastran DMIG coefficients and creates a Nastran bulk data file with the extension `.bdf`.

Command Summary

abaqus tonastran **job =** *job-name*
[input = *input-file***]**
[sim2dmig]
[complex = {`YES` | `NO`**}]**

Command Line Options

job This option is used to specify the name of the Nastran bulk data file to be output by the translator. It is also the default name of the Abaqus file. Diagnostics created by the translator are written to a file named *job-name*`.log`.

input This option is used to specify the name of the file containing the Abaqus data if it is different from *job-name*.

sim2dmig This option is used to translate matrix data in an Abaqus `.sim` file into the Nastran bulk data file (`.bdf`) format.

complex This option is used to determine how structural damping terms are represented. If **complex=**`YES` (default), structural damping terms are written as the imaginary part of the stiffness matrix; if **complex=**`NO`, structural damping terms are written as a separate real matrix.

Translating Abaqus Output Database Files to Nastran Output2 Results Files

Products: Abaqus/Standard

References:

- *About Execution Procedures*

Overview

The translator converts certain results from an Abaqus output database (`.odb`) file to the Nastran Output2 file format.

Using the Translator

The toOutput2 translator can only be used to translate Abaqus output database of a **STATIC* or **FREQUENCY* procedure. Results from an Abaqus analysis are written to the Abaqus output database by using the **OUTPUT* option. The following options should be included in the Abaqus input file to ensure that the results to be translated are available in the Abaqus output database:

```
*OUTPUT, FIELD
*NODE OUTPUT
U,
RF,
CF,
*ELEMENT OUTPUT
S,
E,
SF,
NFORC,
```

Results in the Abaqus output database other than those specified above are skipped during translation. Only results from spring elements and three-dimensional continuum, shell, membrane, beam, and truss elements are translated.

For shell elements, the translator treats stresses and strains at the lowest numbered section point as being at the bottom surface and stresses and strains at the highest numbered section point as being at the top surface. Midsurface stresses and strains translated to the Output2 file are computed as the averages of the stresses and strains at the bottom and top surfaces.

Nodal results are always in global coordinates. By default, element tensor results are in the Abaqus element coordinate system. The **quad4stress_bisector** option can be used to change the default coordinate system for element results for QUAD4 and QUAD8 elements.

Model data from the output database (nodal coordinates, element topology, material properties, and element properties) are written to the Output2 file when applicable records exist.

Command Summary

abaqus toOutput2 **job =** *job-name*
[odb = *odb-name***]**
[step = *step-number***]**
[increment = *increment-number***]**
[slim]
[quad4corner]
[quad4stress_bisector]

Command Line Options

job This option specifies the name of the Nastran Output2 file to be created by the translator. It is also the default name for the Abaqus output database.

odb This option specifies the name of the Abaqus output database if it is different from *job-name*.

step This option specifies the step number of the Abaqus output database for the translator to translate. If the specified step contains multiple load cases, all of the load cases are translated. The default value is the last step of the analysis.

increment This option is valid only when used in conjunction with the **step** option. It is used to specify the increment number of the step in the Abaqus output database for the translator to translate. The default value is the last increment of the specified step.

slim This option is used to include data blocks required for postprocessing in the SLIM/VISION software (available from Third Millennium Productions, Inc.) in the Output2 file.

quad4corner This option is used to request shell output at corner nodes instead of at the centroid. This option is relevant for stress, strain, and section force output.

quad4stress_bisector This option is used to request QUAD4 and QUAD8 element tensor output (stress, strain, and section force) in the Nastran default coordinate system. This system depends on the element connectivity definition. The x-axis of this system is aligned with the line bisecting edges 2 and 4 of each element.

Translating an Abaqus Substructure to an MSC.Adams Flexible Body

Products: Abaqus/Standard

References:

- *About Execution Procedures*
- *Translating Abaqus data to modal neutral file format for analysis in MSC.ADAMS*

Overview

The **abaqus adams** translator converts an Abaqus substructure to a flexible body in a format that can be used by the MSC.Adams multibody dynamics code.

The translator reads Abaqus data from a substructure SIM file and writes data to an MSC.Adams modal neutral (`.mnf`) file.

Using the Translator

The following procedure summarizes the typical usage of the **abaqus adams** translator:

1. Create an Abaqus substructure. (General guidelines for building Abaqus models with substructures are described in *Using Substructures*.)

   The substructure generation step must write at least the mass and stiffness matrices. It can also write the recovery and viscous damping matrices. The **FLEXIBLE BODY* option must be used. For example,

   ```
   *SUBSTRUCTURE GENERATE, MASS MATRIX=YES, RECOVERY MATRIX=YES, VISCOUS
   DAMPING MATRIX=YES
   *FLEXIBLE BODY, TYPE=ADAMS
   ```

   - If TYPE=ADAMS, a flexible body including all high-order inertia invariants is created.
   - If TYPE=ADAMS, FORMULATION=REDUCED, a simplified flexible body (without high-order inertia invariants I5 and I9) is created.

   In addition, you can add the following data to translate stress and/or strain:

   ```
   *ELEMENT RECOVERY MATRIX, POSITION=AVERAGED AT NODES
   S,
   E,
   ```

   If the Abaqus analysis did not use SI units (meters, kilograms, Newtons, and seconds), you must define units when you run the translator using the **length**, **mass**, **force**, and **time** command line options. If the Abaqus analysis uses American or English units (inches, pounds force, and seconds), the mass unit is slinch.
2. Run the Abaqus analysis.
3. Run the **abaqus adams** translator to read the substructure SIM database produced by the analysis and to create the modal neutral file.

Translating Modes with Negative Eigenvalues

Typically, for a nonprestressed, unrestrained substructure in three dimensions, you expect to find six rigid body modes with associated zero eigenvalues. The situation is, in general, different for prestressed substructures, which may have fewer than six modes with zero eigenvalues. Prestressing may change some expected zeroes into values that are significantly positive or negative, depending on the sign of the prestress.

By default, the translator deletes modes with negative eigenvalues and reorthogonalizes the reduced basis. If you want to retain modes with negative eigenvalues, define the environment variable `MDI_MNFWRITE_OPTIONS`.

- On Linux platforms type the following command:
  ```
  setenv MDI_MNFWRITE_OPTIONS negative_roots_OK
  ```
- On Windows platforms type the following command:
  ```
  set MDI_MNFWRITE_OPTIONS=negative_roots_OK
  ```

In this case the translator will treat modes with negative eigenvalues in the same manner as all other modes.

Command Summary

abaqus adams

job = *job-name*
[substructure_sim = *filename***]**
[length = *length-units-name***]**
[mass = *mass-units-name***]**
[time = *time-units-name***]**
[force = *force-units-name***]**
[adams_mnf_block_16 = {`I` **|** `Q`**}]**

Command Line Options

job

This option specifies the input and output file names to use during results translation. The *job-name* value is used to construct the default substructure SIM database file name, *job-name*`.sim`. The output modal neutral file is given the name *job-name*`.mnf`.

If this option is omitted from the command line, the user will be prompted for this value.

substructure_sim

This option specifies the name of the substructure SIM database (`.sim`) file if it is different from *job-name*`.sim`. The file will usually be named *job-name*`_Z`*nn*`.sim`.

length

This option specifies the **length** units for the model. The valid options are as follows:

- `m, meter, meters`
- `mm, millimeter, millimeters`
- `cm, centimeter, centimeters`
- `km, kilometer, kilometers`
- `in, inch, inches`
- `ft, foot, feet`
- `mile, miles`

This option can be defined in the Abaqus environment file as follows:

```
adams_length_units=length-unit
```

mass

This option specifies the **mass** units for the model. The valid options are as follows:

- `kg, kilogram, kilograms`
- `mgg, megagram, megagrams, tonne, tonnes`
- `gram, grams`
- `lbm, pound, pounds, pound_mass`
- `uston, ustons, us_ton`
- `klbm, kpound_mass`

- `ozm, ounce, ounces, ounce_mass`
- `slug`
- `slinch, dozen_slug`

This option can be defined in the Abaqus environment file as follows:

```
adams_mass_units=mass-unit
```

time This option specifies the **time** units for the model. The valid options are as follows:

- `s, sec, second, seconds`
- `ms, millisecond, milliseconds`
- `min, minute, minutes`
- `h, hr, hour, hours`

This option can be defined in the Abaqus environment file as follows:

```
adams_time_units=time-unit
```

force This option specifies the **force** units for the model. The valid options are as follows:

- `n, newton, newtons`
- `kn, knewton, knewtons, kilonewton, kilonewtons`
- `mn, millinewton, millinewtons,`
- `un, micronewton, micronewtons,`
- `nn, nanonewton, nanonewtons,`
- `kgf, kg_force, kilogram_force`
- `lbf, pound, pounds, pound_force`
- `ozf, ounce, ounces, ounce_force`
- `dyne, dynes`
- `klbf, kip, kips, kilopound, kilopounds, kpound_force`

This option can be defined in the Abaqus environment file as follows:

```
adams_force_units=force-unit
```

adams_mnf_block_16 This option defines the contents of the mode shape transformation in block 16 of the modal neutral file. It is meaningful if stress/strain calculations are done following a dynamic analysis in the Adams solver. If the basis for these calculations is modal stress/strain on the modal neutral file, set **adams_mnf_block_16=**`I`. This behavior is the default. If the calculations use stress/strain stored in the ODB, set **adams_mnf_block_16=**`Q`.

Translating an Abaqus Substructure to a Simpack Flexible Body

Products: Abaqus/Standard

References:

- *About Execution Procedures*

Overview

The **abaqus tosimpack** translator converts an Abaqus substructure to a flexible body in a format that can be used by the Simpack multibody dynamics solver.

The translator reads Abaqus data from a substructure SIM file and writes data to a Simpack Flexible Body Interface (FBI) file.

Using the Translator

The following procedure summarizes the typical usage of the **abaqus tosimpack** translator:

1. Create an Abaqus substructure. (General guidelines for building Abaqus models with substructures are described in *Using Substructures*.)

   The substructure generation step must write at least the mass and stiffness matrices. It can also write the recovery, viscous damping, and structural damping matrices. The **FLEXIBLE BODY* option must be used. For example,

   ```
   *SUBSTRUCTURE GENERATE, MASS MATRIX=YES, RECOVERY MATRIX=YES, VISCOUS
   DAMPING MATRIX=YES,
   STRUCTURAL DAMPING MATRIX=YES
   *FLEXIBLE BODY, TYPE=SIMPACK
   ```

   In addition, you can add the following data to translate stress:

   ```
   *ELEMENT RECOVERY MATRIX, POSITION=AVERAGED AT NODES
   S,
   ```

   You can recover results on a subset of the model using the ELSET parameter for the substructure generation and element recovery matrix options shown above.

   The Simpack FBI file is always in SI units. If the Abaqus analysis does not use SI units (meters, kilograms, and seconds), you can define the model units using the **UNIT SYSTEM* option. Alternatively, you can define the model units when you run the translator using the **length**, **mass**, and **time** command line options. If the Abaqus analysis uses American or English units (inches, pounds force, and seconds), the mass unit is slinch.
2. Run the Abaqus analysis.
3. Run the **abaqus tosimpack** translator to read the substructure SIM database produced by the analysis and to create the flexible body interface file.

Command Summary

abaqus tosimpack **job =** *job-name*
[substructure_sim = *filename***]**
[length = *length-units-name***]**
[mass = *mass-units-name***]**
[time = *time-units-name***]**

Command Line Options

job This option specifies the input and output file names to use during results translation. The *job-name* value is used to construct the default substructure SIM database file name, *job-name*`.sim`. The output flexible body interface file is given the name *job-name*`.fbi`.

If this option is omitted from the command line, you will be prompted for this value.

substructure_sim This option specifies the name of the substructure SIM database (`.sim`) file if it is different from *job-name*`.sim`. The file will usually be named *job-name*`_Z`*nn*`.sim`.

length This option specifies the **length** units for the model. The valid options are as follows:

- `m, meter, meters`
- `mm, millimeter, millimeters`
- `cm, centimeter, centimeters`
- `km, kilometer, kilometers`
- `in, inch, inches`
- `ft, foot, feet`
- `mile, miles`

mass This option specifies the **mass** units for the model. The valid options are as follows:

- `kg, kilogram, kilograms`
- `mgg, megagram, megagrams, tonne, tonnes`
- `gram, grams`
- `lbm, pound, pounds, pound_mass`
- `uston, ustons, us_ton`
- `klbm, kpound_mass`
- `ozm, ounce, ounces, ounce_mass`
- `slug`
- `slinch, dozen_slug`

time This option specifies the **time** units for the model. The valid options are as follows:

- `s, sec, second, seconds`
- `ms, millisecond, milliseconds`
- `min, minute, minutes`
- `h, hr, hour, hours`

Translating an Abaqus Substructure to an EXCITE Flexible Body

Products: Abaqus/Standard

References:

- *About Execution Procedures*

Overview

The **abaqus toexcite** translator converts an Abaqus substructure to a flexible body in a format that can be used by the EXCITE multibody dynamics code.

The translator reads Abaqus data from a substructure SIM file and writes data to an EXCITE binary (`.exb`) file.

Using the Translator

The following procedure summarizes the typical usage of the **abaqus toexcite** translator:

1. Create an Abaqus substructure. (General guidelines for building Abaqus models with substructures are described in *Using Substructures*.)

   The substructure generation step must write at least mass and stiffness matrices. It can also write the recovery, viscous damping, and structural damping matrices. The **FLEXIBLE BODY* option must be used. For example,

   ```
   *SUBSTRUCTURE GENERATE, MASS MATRIX=YES, RECOVERY MATRIX=YES,
   VISCOUS DAMPING MATRIX=YES, STRUCTURAL DAMPING MATRIX=YES
   *FLEXIBLE BODY, TYPE=EXCITE
   ```

   - If TYPE=EXCITE, an EXCITE flexible body of type CON6 including high-order inertia invariants is created.
   - If TYPE=EXCITE, FORMULATION=REDUCED, a simplified EXCITE flexible body of type CON6 using only first-order inertia invariants is created.
   - If TYPE=EXCITE, FORMULATION=SMALL MOTION, a small-motion flexible body of type SMOT is created.

2. Run the Abaqus analysis.
3. Run the **abaqus toexcite** translator to read the substructure SIM database produced by the analysis and to create the EXCITE binary file.

Command Summary

abaqus toexcite **job =** *job-name*
[hide_mesh = {OFF | ON}**]**
[recovery_matrix = {OFF | ON | MERGE}**]**
[substructure_sim = *filename***]**

Command Line Options

job This option specifies the input and output file names to use during results translation. The *job-name* value is used to construct the default substructure SIM database file name, *job-name*`.sim`. The output flexible body interface file is given the name *job-name*`.exb`.

If this option is omitted from the command line, you will be prompted for this value.

hide_mesh This option specifies whether or not to write the element nodes, connectivities, materials, and properties to the EXCITE binary file. By default, the mesh is written. If **hide_mesh**=`ON`, the mesh is not written.

recovery_matrix This option specifies whether or not to write the recovery matrix to an EXCITE binary file.

By default, the recovery matrix is written to a separate EXCITE binary file. If **recovery_matrix**=`MERGE`, the recovery matrix is written to the same file as the rest of the flexible body data. If **recovery_matrix**=`OFF`, the recovery matrix is not written.

substructure_sim This option specifies the name of the substructure SIM database (`.sim`) file if it is different from *job-name*`.sim`. The file will usually be named *job-name*_`Z`*nn*`.sim`.

Environment File Settings

Products: Abaqus/Standard Abaqus/Explicit

References:

- *About Execution Procedures*

Overview

The Abaqus environment settings allow you to control various aspects of an Abaqus job's execution. For example, you can

- "Tune" Abaqus to improve its performance by changing memory-related parameters.
- Control where and how scratch files are written.
- Provide default values for job parameters that would otherwise have to be given on the command line.

Many other aspects of a job's execution can be configured through the environment settings. Some of these are discussed in this section; others, which are mainly of interest to the Abaqus site manager, are discussed in detail in the *Abaqus Configuration Guide*.

Environment Settings Hierarchy

Abaqus environment settings are processed in the following order:

1. The host-level environment settings. These settings are applied to all Abaqus jobs run on the designated computer.
2. The user-level environment settings. These settings are applied to all Abaqus jobs run in your account.

   For Abaqus to locate the environment file in your home directory on Windows platforms, the full path to your home directory must be specified using the `HOME` environment variable or a combination of the `HOMEDRIVE` and `HOMEPATH` environment variables.
3. The job-level environment settings. These settings are applied to only the designated Abaqus job.

Environment settings can be specified more than once. The last value processed will be the one used for the setting if it is defined at more than one level or if it is given twice at the same level.

Abaqus environment settings are set using special files in specific directories. The host-level (site-level) settings are set in the *install_dir*`/`*os*`/SMA/site/custom_v6.env` file or `abaqus_v6.env` file in the installation. You can change these settings by creating an environment file named `abaqus_v6.env` in your home directory and/or the current directory. Settings in the home directory file will be applied to all jobs that you run. Settings in the current directory file will be applied only to jobs run from the current directory.

Syntax

The entries given in the environment file must be given using Python language syntax. Entries take the form:

```
parameter=value
```

The following is a brief overview of the Python syntax rules:

- The parameter must always have a value. The value can be any valid Python constant or expression.
- A string value must be enclosed in a pair of double or single quotes.
- Comments are preceded by a number sign (#). All characters following a number sign on a line are ignored. Number signs within a quoted string are part of the string, not the beginning of a comment.
- Blank lines are ignored.
- Embedded single quotes do not require special handling if they are placed within a double quoted string. For example, `"my value's"` is translated as `my value's`. The same holds true for double quotes embedded in a single quoted string. Quotes of the same type as the enclosing quotes can be embedded if they are prefixed by the backslash (\) character.
- Triple quoted (""") strings can span more than one line, and no special treatment of quotes within the string is required. Entries take the form:

  ```
  parameter="""
  multi-line
  value
  """
  ```

- Lists must be enclosed in parentheses (( )) or square brackets ([ ]). Individual items in the list are separated by commas. If the list is enclosed in parentheses and contains only one value, a comma has to follow the value. String list items must be enclosed in quotes. Entries take the form:

  ```
  parameter=(value1, value2, value3)
  ```

Troubleshooting

Problems caused by faulty environment settings can be diagnosed using the command.

```
abaqus information=environment
```

This command prints all the current environment settings.

Command Line Default Parameters

The following parameters provide default values for various settings that you would otherwise have to specify on the command line (see *Abaqus/Standard and Abaqus/Explicit Execution*, and *Abaqus/Standard, Abaqus/Explicit, and FMU Co-Simulation Execution*). Values given on the command line override values specified in the environment files.

cpus

Number of processors to use if parallel processing is available. The default is `2` for the co-simulation execution procedure; otherwise, the default is `1`.

domains

The number of parallel domains in Abaqus/Explicit. If the value is greater than 1, the domain decomposition will be performed regardless of the values of the **parallel** and **cpus** parameters. However, if **parallel**=`domain`, the value of **cpus** must be evenly divisible into the value of **domains**. If this parameter is not set, the number of domains defaults to the number of processors used during the analysis run if **parallel**=`domain` or to 1 if **parallel**=`loop`.

double_precision

The default precision version of Abaqus/Explicit to run if you do not specify the precision version on the **abaqus** command line. Possible values are `EXPLICIT` (only the Abaqus/Explicit analysis is run in double precision), `BOTH` (both the Abaqus/Explicit packager and analysis are run in double precision), `CONSTRAINT` (the constraint packager and constraint solver in Abaqus/Explicit are run in double precision, while the Abaqus/Explicit packager and analysis continue to run in single precision), or `OFF` (both the Abaqus/Explicit packager and analysis are run in single precision). The default is `OFF`.

parallel

The default parallel method in Abaqus/Explicit if you do not specify the parallel method on the **abaqus** command line. Possible values are `DOMAIN` or `LOOP`; the default value is `DOMAIN`.

run_mode

Default run mode (interactive, background, or batch) if you do not specify the run mode on the **abaqus** command line. The default for **abaqusanalysis** and **abaqussysVerify** is `"background"`, while the default for **abaqusviewer** is `"interactive"`.

scratch

Directory to be used for scratch files. This directory must exist (that is, it will not be created by Abaqus) and must have write permission assigned. On Linux platforms the default value is the value of the `$TMPDIR` environment variable or `/tmp` if `$TMPDIR` is not defined. On Windows platforms the default value is the value of the `%TEMP%` environment variable or `\TEMP` if this variable is not defined. During the analysis a subdirectory will be created under this directory to hold the analysis scratch files. The name of the subdirectory is constructed from your user name, the job id, and the job's process identifier. The subdirectory and its contents are deleted on completion of the analysis.

standard_parallel

The default parallel execution mode in Abaqus/Standard if you do not specify the parallel mode on the **abaqus** command line. If this parameter is set equal to `ALL`, both the element operations and the solver will run in parallel. If this parameter is set equal to `SOLVER`, only the solver will run in parallel. The default parallel execution mode is `ALL`.

gpus

The GPGPU solver acceleration setting in Abaqus/Standard if you do not specify the GPGPU solver acceleration option on the **abaqus** command line. By default, GPGPU solver acceleration is not activated. The value of this parameter is the total number of GPGPUs to be used in an Abaqus/Standard analysis. In an MPI-based analysis, this parameter is the number of GPGPUs to be used across all hosts.

unconnected_regions

If this variable is set to `ON`, Abaqus/Standard will create element and node sets in the output database for unconnected regions in the model during a **datacheck** analysis. Element and node sets created with this option are named `MESH COMPONENT N`, where *N* is the component number. The default value is `OFF`.

order_parallel

The ordering mode for the direct sparse solver in Abaqus/Standard if you do not specify the ordering mode on the **abaqus** command line. If this parameter is set equal to `OFF`, the ordering procedure will not run in parallel. If this parameter is set equal to `ON`, the ordering procedure will run in parallel. The default ordering mode is `ON`.

System Resource Parameters

The following environment file variable can be set after the code has been installed to change the resources used by Abaqus and, therefore, to improve system performance. By default, Abaqus detects the physical memory on a machine (or on each compute node in a cluster) and allocates a percentage of the available memory. The default percentage is 80%. You can override the default percentage by specifying a number followed by the percentage sign. The variable can also be defined as the number of megabytes or the number of gigabytes. More detailed information about changing the system resources used by Abaqus is given in *Managing Memory and Disk Resources*.

memory

Maximum amount of memory or maximum percentage of the physical memory that can be allocated during the input file preprocessing and during the Abaqus/Standard analysis phase. For parallel execution on computer clusters, this memory limit specifies the maximum amount of memory that can be allocated on each process.

System Customization Parameters

The following is a discussion of some additional environment file parameters that are commonly used. A complete listing of parameters can be found in the *Abaqus Configuration Guide*.

ask_delete

If this parameter is set equal to `OFF`, you will not be asked whether old job files of the same file name should be deleted; the files will be deleted automatically. The default value is `ON`.

auto_calculate

If this parameter is set to ANALYSIS, all output database postprocessing in Abaqus/Standard is performed as part of analysis execution. If this parameter is set to ON, output database postprocessing is performed by the postprocessing calculator at the end of an analysis if the execution procedure detects that output database postprocessing is required. If this parameter is set to OFF, no output database postprocessing occurs even if the execution procedure detects that it is required to postprocess the output database file. The default value is ANALYSIS. In Abaqus/Explicit**auto_calculate**=ANALYSIS is equivalent to **auto_calculate**=ON.

auto_convert

If this parameter is set equal to ON and an Abaqus/Explicit analysis is run in parallel with **parallel**=domain, the **convert**=select, **convert**=state, and **convert**=odb options will be run automatically at the end of the analysis. The default value is ON.

average_by_section

This parameter is used only for an Abaqus/Standard analysis. If this parameter is set equal to OFF, the averaging regions for output written to the data (.dat) file and results (.fil) file are based on the structure of the elements. If this parameter is set equal to ON, the averaging regions also take into account underlying values of element properties and material constants. In problems with many section and/or material definitions the default value of OFF will, in general, give much better performance than the nondefault value of ON. See *Output to the Data and Results Files* for further details on the averaging scheme.

mp_host_list

List of host machine names to be used for a parallel Abaqus analysis, including the number of processors to be used on each machine; for example,

```
mp_host_list=[['maple',1],['pine',1],['oak',2]]
```

indicates that, if the number of **cpus** specified for the analysis is 4, the analysis will use one processor on a machine called maple, one processor on a machine called pine, and two processors on a machine called oak. The total number of processors defined in the host list must be greater than or equal to the number of **cpus** specified for the analysis. **mp_host_list** is modified to account for the **threads_per_mpi_process** value as required. If the host list is not defined, Abaqus will run on the local system. When using a supported queuing system, this parameter is set by the queuing environment.

odb_output_by_default

If this parameter is set equal to ON, output database output will be generated automatically. If this parameter is set equal to OFF, output database request keywords must be placed in an input file to obtain output database output. The default value is ON.

output_warnings_level

If this parameter is set equal to ALL, all warnings associated with output request validation will be reported by the Abaqusanalysis input file processor. If this parameter is set equal to OFF, no warnings associated with output request validation will be reported. The default value is DEFAULT, which will not include common warnings, such as those for output not supported for specific procedures or element types.

onCaeStartup

Optional function to be executed before Abaqus/CAE begins or any time driverUtils.executeOnCaeStartup is executed in a script. See *Customizing Abaqus/CAE startup* for examples of this function.

Export of Environment Variables during a Parallel Execution

The batch queuing systems typically propagate all environment variables that are not machine specific and that are defined at job submission time to the first execution host only. You must use the Abaqus environment parameter **mp_environment_export** in a local `abaqus_v6.env` file to export all environment variables to all the compute nodes in a parallel execution. In the example below, two environment variables, `MYVAR_1` and `MYVAR_2`, are first defined, and then they are exported to all the compute nodes in a parallel execution. For more information, see the Dassault Systèmes Knowledge Base item *QA00000008589*.

```
import os
os.environ['MYVAR_1']='value1'
os.environ['MYVAR_2']='value2'
mp_environment_export +=('MYVAR_1', 'MYVAR_2')
```

Co-Simulation Parameters

The following environment file variables provide default settings for co-simulation between solvers using the direct coupling interface. This includes Abaqus/Standard to Abaqus/Explicit co-simulation and co-simulation between Abaqus and certain third-party analysis programs.

cosimulation_port

Set **cosimulation_port** equal to the port number used for the connection. The default value is `48000`.

cosimulation_timeout

Set **cosimulation_timeout** equal to the timeout period in seconds. Abaqus terminates if it does not receive any communication from the coupled analysis program during the time specified. The default value is `3600` seconds.

The following environment file variables provide settings that allow you to allocate CPUs for co-simulation jobs submitted using the co-simulation execution procedure. This includes Abaqus/Standard to Abaqus/Explicit co-simulation (see *Abaqus/Standard, Abaqus/Explicit, and FMU Co-Simulation Execution*).

cpus_weight_std

This option controls the allocation of CPUs to Abaqus/Standard analyses. The actual CPU allocation for Abaqus/Standard analyses is made in proportion to this value and considering the settings of **cpus_weight_xpl** and **cpus**.

cpus_weight_xpl

This option controls the allocation of CPUs to Abaqus/Explicit analyses. The actual CPU allocation for Abaqus/Explicit analyses is made in proportion to this value and considering the settings of **cpus_weight_std** and **cpus**.

portpool

Set this variable equal to a colon-separated pair of TCP/UDP port numbers that represents the start and end value of port numbers to be used by the co-simulation execution procedure when establishing connections between the child processes.

Environment File Examples

Example environment files that use some of the previously discussed parameters are shown below. A sample environment file, named `abaqusinc.env`, is included in the `site` subdirectory of the release to show the options used at SIMULIA.

Linux Environment File:

```
ask_delete=OFF
# The following parameter causes the scratch files to
# be written to /tmp.
scratch="/tmp"
```

Windows Environment File:

```
ask_delete=OFF
# The following parameter causes the scratch files to
# be written to the tmp directory on c:.
scratch="c:/tmp"
```

Managing Memory and Disk Resources

Products: Abaqus/Standard Abaqus/Explicit

References:

- *About Execution Procedures*
- *Environment File Settings*

Overview

For small analyses management of computer resources is generally of secondary concern, but with large models intelligent use of disk and memory resources is a critical part of the analysis process. For moderate to large analyses you will find it necessary to modify resource management settings.

Understanding Resource Use

For Abaqus disk and memory are effectively two similar means of storing data. Data that will be required after an analysis completes must eventually be written to disk; but during an analysis, disk and memory provide functionally equivalent storage mechanisms. Typically disk is a more abundant resource, while memory provides faster access to stored data. Management of Abaqus resources hinges on this simple trade-off.

Abaqus Data

There are effectively two types of data generated by an Abaqus analysis. The first is “output” data that must persist after an analysis is complete. Output data are typically either results that you require for postprocessing or data that are necessary to restart an analysis. As mentioned above, these data must be stored on disk before an analysis completes.

In addition, an analysis generates a considerable amount of “scratch” or temporary data. These are data that are needed only while an analysis is running. The scratch data can be subdivided into two groups: performance-critical data and generic data. The performance-critical data are always stored in memory, while the generic data can be stored either in memory or on disk.

Requirements and Considerations

To run an analysis, the following requirements must be satisfied:

- There must be sufficient disk space available to hold the requested output data.
- There must be sufficient memory available to hold all performance-critical data.
- There must be sufficient disk space or memory resource available to hold all generic scratch data.

If the above requirements are satisfied, an analysis can be completed; however, for Abaqus/Standard you may find that allowing Abaqus to use additional memory will often improve performance. With the increased availability of computer clusters, dedicated shared memory computers, and most importantly job queuing systems

that allocate processors and memory for analyses, it makes most sense to be able to use all the memory resources to improve performance.

Typically Abaqus/Standard allocates a large portion of the available system memory on a machine during the analysis phase, but you can manually specify a limit for memory usage with the **memory** parameter (see *Resource Management Parameters* below). No scratch data are written to disk during the Abaqus/Explicit analysis phase, since the majority of scratch data are performance critical.

Resource Management Parameters

Abaqus resource management parameters fall into two classes: memory management and disk management. Each can be adjusted through one environment file parameter. The following sections explain how to best make use of this parameter. For information about the environment file, see *Environment File Settings*.

Memory Management Parameters

The **memory** parameter is used to limit the amount of memory that can be used during the analysis phase of Abaqus/Standard and during the input file processing phase, which is executed before both Abaqus/Standard and Abaqus/Explicit analyses.

If you do not define the **memory** parameter, Abaqus automatically detects the physical memory on the machine and allocates a percentage of this available memory. The default percentage is 80%.

You can override the default memory allocation by specifying the percent of physical memory or by specifying an absolute limit in units of megabytes or gigabytes. Percentages are indicated by a "%" sign following the specified limit. Units of megabytes and gigabytes are indicated by "mb" or "gb" following the specified limit. If no units are specified, megabytes are assumed. For example, with any of the following settings:

```
memory="2048 mb"
memory="2 gb"
memory="25 %"
```

Abaqus uses up to 2 gigabytes of memory on a machine with 8 gigabytes physical memory. The memory setting value must be surrounded by quotes. The values specified for **memory** must be reasonable for the machine being used. Abaqus/Standard does not check the validity of the numerical values. To be consistent with operating system memory measurement tools, a megabyte is defined by Abaqus to be 1,048,576 bytes, not 1,000,000 bytes. A similar rule applies to the unit of gigabyte.

There are no memory management parameters for the Abaqus/Explicit analysis phase, since no scratch data are written to disk during this phase.

Environment file parameters can be set for a host, for a user, or for a particular job (see *Environment File Settings* for further discussion). Because a default memory setting that works well for one machine with a large amount of memory may not be ideal for another machine that has less memory, it may be useful to vary the default memory settings by machine.

Disk Management Parameters

Management of output data is discussed in detail in *About Output*. Output data are written to files in the directory from which you launched the job.

Abaqus/Standard scratch files are written to a separate scratch directory. You can control the directory used to hold the scratch files with the **scratch** environment file parameter. Due to the frequent access of the scratch data throughout the analysis phase, ensuring high I/O speed of the scratch disks is essential to the analysis performance.

As explained above, no scratch data are written to disk for Abaqus/Explicit, so you have to be concerned only with proper management of output data.

Input File Processing and data check

In general, the amount of memory required during input file processing is not large. The amount of memory and disk space needed for the analysis phase of a job is more likely to be a concern. It is not possible for Abaqus to estimate the amount of memory that will be required to complete input file processing. A data check run can be performed by using the **datacheck** parameter in the command for running Abaqus (see *Abaqus/Standard and Abaqus/Explicit Execution*) to obtain an estimate of the required memory for completing the analysis phase. General guidelines for setting the **memory** parameter for performing the data check (which includes the input file processing phase) are given below.

Guidelines for Memory Settings

You will usually not have to change the default memory setting. If a job fails as a result of insufficient memory with the default setting, you will need to find a machine with more memory to run the job. If you need to override the default behavior by specifying a value for the **memory** environment file parameter, *Table 1* lists some typical data check memory settings for problems of various sizes. The actual values required for **memory** may vary considerably from problem to problem depending on the features used in a model.

Table 1: Typical memory settings for performing the data check analysis.

Degrees of freedom	Memory
250,000	250 megabytes
1 million	750 megabytes
2.5 million	1200 megabytes
5 million	2000 megabytes

Abaqus/Standard Analysis

Depending on the execution environment and typical job sizes run on the machine, **memory** can be set by machine or by job. More detailed guidelines are provided in the following section. When setting **memory** by job is needed, you are advised to run a data check analysis and set **memory** based on the memory estimates. These estimates are written to the printed output (`.dat`) file in a table under the heading "MEMORY ESTIMATE." Two columns in this table are relevant to memory use. The first relevant column is labeled "MINIMUM MEMORY REQUIRED" and specifies the **memory** setting that is needed to hold critical scratch data in memory. An attempt to run the analysis with **memory** set below this value will result in a warning, and the job is not likely to run to completion due to the insufficient memory. The second relevant column is labeled "MEMORY TO MINIMIZE I/O" and specifies the **memory** that is required to hold all scratch data, both critical and generic, in memory. If the memory specified by **memory** is larger than the "MINIMUM MEMORY REQUIRED," Abaqus/Standard automatically uses the additional memory up to the memory limit to improve speed of access to generic scratch data that would otherwise be written to disk. When the **memory** is not enough to hold all the generic scratch data in memory, Abaqus/Standard decides which data should be written to disk and which should be kept in memory based on their relative importance with respect to their effect on the analysis performance. Therefore, the actual disk space used by the scratch data can vary from very close to zero to the

"MEMORY TO MINIMIZE I/O" depending on the **memory** setting. The **memory** setting can be changed in an analysis continued from a data check without the need of rerunning the analysis input file processor.

Guidelines for Memory Settings

The **memory** parameter allows you to specify the memory limit that can be used by Abaqus during the input file processing and analysis phases. You can specify the setting that should generally be available to Abaqus on a particular machine in the host environment file. Settings can be modified as necessary for individual jobs in job-specific environment files. Reasonable settings for a particular machine depend on the size of the problems being run and how the machine is being used in addition to the physical memory available on the machine. You should be aware of the difference between physical and virtual memory. When virtual memory is used, a machine's operating system simply uses disk for additional memory. While this can be useful, memory access may require I/O operations that add a considerable performance penalty. Therefore, the guidelines below for managing memory in Abaqus/Standard are always given relative to the physical memory on a machine. Virtual memory should be used only when necessary and with awareness of the associated performance penalty.

Setting memory on Single-User Machines

For a single-user machine that is dedicated to running Abaqus/Standard, using the default setting of **memory** is sensible. If the estimates indicate that the job requires more than this value, the job is too large to run efficiently on this machine. At this point you are urged to move the analysis to another machine with more memory resources.

For a single-user machine that is used to run both Abaqus/Standard and other applications simultaneously, setting a lower **memory** limit makes sense. If an analysis requires more than the specified value, you can decide to increase **memory** and continue the job. However, Abaqus/Standard will have to contend with the other applications for memory, which will impair the efficiency of both Abaqus/Standard and the other applications. If the other applications are interactive, the performance degradation could be problematic. In such a case you might decide to delay continuing the analysis until the machine can be dedicated to running Abaqus/Standard alone.

Setting memory on Multi-User Machines

The guidelines for setting **memory** on a multi-user machine are very similar to those for single-user machines, except that a judgement must be made as to the amount of memory that each user on the machine can expect to have for a single analysis. A reasonable approach might be to divide the machine's physical memory by the number of expected simultaneous jobs. Another sensible approach is to divide the machine's physical memory by the total number of CPUs and then multiply by the number of CPUs used for the current job. If the memory requirement among the simultaneous jobs is not even, you might want to divide the machine's physical memory in an uneven way accordingly. In general, to ensure acceptable performance, users on multi-user machines need to coordinate with each other to properly set the memory limit.

Setting memory When Using Queues

Often queues have an associated memory limit, and determining the appropriate queue for a job requires some judgement. You are advised to run a data check analysis and select a queue based on the estimates provided in the printed output file. However, for large analyses even a data check analysis can require a large amount of memory. Choosing an appropriate queue for a data check analysis requires some experience with particular classes of problems. You may want to submit data check runs initially to queues with very large memory limits to get the necessary estimates. An appropriate queue can then be chosen to actually run the job. If the jobs are to be submitted to shared memory machines, it makes sense to set **memory** to about 90% of the memory limit for the queue. If the jobs are to be submitted to computer clusters, it is reasonable to use the default memory setting.

Parallel Execution

Parallel execution of Abaqus is implemented using two different schemes: threads and message passing.

In this section:

- *About Parallel Execution*
- *Parallel Execution in Abaqus/Standard*
- *Parallel Execution in Abaqus/Explicit*

About Parallel Execution

Products: Abaqus/Standard Abaqus/Explicit

References:

- *Obtaining Information*
- *Environment File Settings*
- *Parallel Execution in Abaqus/Standard*
- *Parallel Execution in Abaqus/Explicit*

Overview

Parallel execution of Abaqus is implemented using three different schemes: threads, message passing, and a combination of both. Threads are lightweight processes that can perform different tasks simultaneously within the same application. Threads can communicate relatively easily by sharing the same memory pool. Thread-based parallelization is readily available on all shared memory platforms.

Parallelization with message passing uses multiple analysis processes that communicate with each other by means of the Message Passing Interface (MPI). This requires MPI components to be installed.

Hybrid parallelization combines MPI-based message passing between multiple analysis processes with shared memory based communication among threads under each analysis process. This requires MPI components to be installed.

Output the local installation notes for your system to learn about local multiprocessing capabilities (see *Obtaining Information*). From the **Support** page at *https://www.3ds.com/products-services/simulia/*, refer to the **System Information** page for the current release of Abaqus for complete information about parallel processing support on various platforms, including information about MPI requirements and availability.

Parallel Processing Support for Abaqus Features

The following Abaqus/Standard features can be executed in parallel: analysis input preprocessing, the direct sparse solver, the iterative solver, the eigensolvers, and element operations. Analysis input preprocessing uses only MPI-based parallelization and will not be executed in parallel if only data checking is performed. For Abaqus/Explicit all the computations other than those involving the analysis input preprocessor. Each of the features that are available for parallel execution has certain limitations, which are documented in detail; see *Parallel Execution in Abaqus/Standard* and *Parallel Execution in Abaqus/Explicit*.

Parallel Execution on Shared Memory Computers

Abaqus/Standard and Abaqus/Explicit can be executed in parallel on shared memory computers by using threads or the MPI. When the MPI is available, Abaqus runs all available parallel features with MPI-based parallelization and activates thread-based parallel implementations for cases where an equivalent MPI-based implementation does not exist.

Parallel Execution on Computer Clusters

Abaqus can be executed in parallel on computer clusters by using MPI-based parallelization. For parallel execution on computer clusters, the list of machines or hosts is given with the **mp_host_list** environment file parameter. This parameter also defines the number of processors to be used on each host.

Parallel Execution Using GPGPU Hardware

The direct solver, the AMS eigensolver, and the modal frequency response solver in Abaqus/Standard can be executed in parallel on computers equipped with compute-capable GPGPU cards.

Use with User Subroutines

User subroutines can be used when running jobs in parallel. In a distributed run, the entire model is decomposed into separate domains (partitions). Each domain is serviced by a separate MPI process. Abaqus provides well-defined synchronization points at which it is possible to exchange information across all MPI ranks, using the MPI communications facilities. All native MPI calls are supported, in both Fortran and C++. In addition, for cases of hybrid execution, user subroutines and any subroutines called by them must be thread safe. This precludes the use of common blocks, data statements, and save statements. To work around these limitations and for guidelines and techniques, see *Ensuring Thread Safety*.

Parallel Execution in Abaqus/Standard

Products: Abaqus/Standard Abaqus/CAE

References:

- *Obtaining Information*
- *Environment File Settings*
- *Controlling job parallel execution*

Overview

Parallel execution in Abaqus/Standard:

- reduces run time for large analyses;
- is available for shared memory computers and computer clusters for the element operations, direct sparse solver, and iterative linear equation solver; and
- can use compute-capable GPGPU hardware on shared memory computers for the direct sparse solver, AMS eigensolver, and modal frequency response solver.

Invoking Parallel Processing

Abaqus/Standard supports both shared memory computers and computer clusters for parallelization. Parallelization is invoked using the **cpus** option, representing the total number of processors; and the **threads_per_mpi_process** option, representing the division of these processors into processes, in the **abaqus** execution procedure. Accordingly, the job submission system is expected to configure the compute resources as reflected by the environment variable **mp_host_list** (see *Environment File Settings*). If **mp_host_list** consists of a single machine host, thread-based parallelization is used within that host if it has more than one processor available. If **mp_host_list** consists of multiple hosts, MPI-based parallelization is executed. In addition, if each host has more than one processor, thread-based parallelization is executed on each host. Thus, the type of parallelization can vary from pure MPI-based, when each host has only one processor, to pure thread-based, when only a single host is used with multiple processors, and to a hybrid mode, when multiple hosts each with multiple processors are used.

Thread-Based Parallelization

You can execute Abaqus/Standard in thread mode within one node of a compute cluster. This approach takes advantage of the shared memory available to the threads that are running on different processors.

In most cases, Abaqus/Standard provides full support for thread-based parallelization; however, this parallelization method is not supported for linear dynamic analysis procedures started using the parameter setting SIM=NO for the eigenvalue extraction analysis procedure.

Two thread-based parallelization options are available: the hybrid direct sparse solver and the pure thread-based direct sparse solver. The pure thread-based direct sparse solver provides better performance than the hybrid direct sparse solver for many workflows, particularly for models dominated by structural elements. All models that you can run in the memory of a single compute node should benefit. The pure thread-based direct sparse solver is the default option. However, there are a number of cases for which the pure thread-based direct sparse

solver is not available. Abaqus/Standard detects these cases automatically and runs thread-based parallelization using the hybrid direct sparse solver instead. For more information, see *Limitations*.

Input File Usage: Enter the following input on the command line:

abaqusjob=*job-name* **cpus**=*n* **threads_per_mpi_process**=*n*

The resulting **mp_host_list** is configured as shown below and the host list consists of only one host (`maple`), which has *n* processors to use. The job runs thread-based parallelization on this host. The default approach is to run the pure thread-based solver. To use the hybrid direct sparse solver, use **direct_solver**=*DMP*

mp_host_list=['maple', *n*]

Abaqus/CAE Usage: Job module: job editor: **Parallelization**: toggle on **Use multiple processors**, and specify the number of processors, *n*; **Multiprocessing mode: Threads**

Limitations

Abaqus/Standard reverts to the hybrid direct sparse solver implementation if your analysis includes any of the following cases (for which the pure thread-based direct sparse solver is not available):

- Eigensolver buckling prediction (*Eigenvalue Buckling Prediction*).
- Natural frequency extraction using EIGENSOLVER=AMS or EIGENSOLVER=SUBSPACE options (*Natural Frequency Extraction*).
- Substructure generation (*Generating Substructures*).
- Matrix generation (*Generating Matrices as a Linear Analysis Step*).
- Element matrix output requests (*Element Matrix Output in Abaqus/Standard*).
- Features activating **SOLVER CONTROLS*, CONSTRAINT OPTIMIZATION; **INERTIA RELIEF*; **COUPLING*; and **DISTRIBUTING*; or any of these element types: C3D4H, C3D4PH, C3D4PHT, C3D10HS.
- Procedure types **SOILS*, **MAGNETOSTATIC*, and **DIRECT CYCLIC*.
- **CO-SIMULATION*.
- **LOAD CASE* when boundary conditions vary between load cases.
- **SOLUTION TECHNIQUE*, TYPE=LCP CONTACT.
- Transient sensitivity analysis **STEP*, SENSITIVITY=ADJOINT; and **DYNAMIC*.
- GPGPU acceleration.
- MPI-based parallelization.
- The inability for all solver data to remain in processor memory.

MPI-Based Parallelization

Abaqus/Standard can also be executed in MPI mode, which uses the message passing interface to communicate between machine hosts. In most cases MPI-based parallelization is fully supported in Abaqus/Standard, except in the following workflows and features:

- Quasi-Newton nonlinear solution technique (*Solving Nonlinear Problems*).
- Response spectrum analysis (*Response Spectrum Analysis*).
- Random response analysis (*Random Response Analysis*).
- Mode-based linear dynamic analysis (*Transient Modal Dynamic Analysis*, *Mode-Based Steady-State Dynamic Analysis*, *Subspace-Based Steady-State Dynamic Analysis*, and *Complex Eigenvalue Extraction*).

- Cavity radiation analyses where parallel decomposition of the cavity is not allowed and writing or restart data is requested (*Cavity Radiation in Abaqus/Standard*).
- Heat transfer analyses where average-temperature radiation conditions are specified (*Thermal Loads*).
- Substructure generation (*Generating Substructures*).
- Matrix generation (*Generating Matrices as a Linear Analysis Step*).
- Element matrix output requests (*Element Matrix Output in Abaqus/Standard*).
- Continuation of output on restart (*Continuation of Output on Restart*).
- Adaptive meshing (*Defining ALE Adaptive Mesh Domains in Abaqus/Standard*).

The Lanczos eigensolver supports MPI-based parallelization with the following exceptions:

- Non-SIM workflow (**FREQUENCY*, SIM=NO).
- Flexible body generation.
- Adjoint Sensitivity analysis.
- Energy output.
- Any subsequent modal analyses after the natural frequency analysis using the Lanczos eigensolver

The subspace iteration eigensolver supports MPI-based parallelization with the following exceptions:

- Non-SIM workflow (**FREQUENCY*, SIM=NO).
- Coupled structural-acoustic analysis (**FREQUENCY*, ACOUSTIC COUPLING=ON and ACOUSTIC COUPLING=PROJECTION).
- Energy output.
- Any subsequent modal analyses after the natural frequency analysis using the subspace iteration eigensolver.

However, MPI-based parallelization is not supported for natural frequency extraction analysis using the AMS eigensolver. After natural frequency analysis using the Lanczos eigensolver or the subspace iteration eigensolver in MPI mode, any subsequent modal analyses can run as a restart analysis using thread-based parallelization.

Input File Usage: Enter the following input on the command line:

abaqusjob=*job-name* **cpus**=*n* **threads_per_mpi_process**=*1*

If *n*=4 and **threads_per_mpi_process**=*1*, the resulting **mp_host_list** is configured as shown below, with the host list consisting of four hosts (`maple`, `pine`, `oak`, and `elm`), and each host having only one processor to use. The job runs MPI-based parallelization between these hosts.

mp_host_list=[['maple', 1], ['pine', 1], ['oak', 1], ['elm', 1]]

Abaqus/CAE Usage: Job module: job editor: **Parallelization**: toggle on **Use multiple processors**, and specify the number of processors, *n*; **Multiprocessing mode: MPI**

Hybrid Parallelization of MPI and Threads

You can further improve performance in hybrid mode when MPI parallelization is executed between hosts and thread-based parallelization is executed within each host.

The **threads_per_mpi_process** option can be used with the **cpus** option to reconfigure the parallelization. The number of **threads_per_mpi_process** should be a divisor of the number of processors on each host and eventually a divisor of the total number of cpus.

Input File Usage: Enter the following input on the command line:

abaqusjob=*job-name* **cpus**=*n* **threads_per_mpi_process**=*m*

If *n*=4 and *m*=2, the resulting **mp_host_list** is configured as shown below, the host list consists of two hosts (`maple` and `pine`), and each host has two processors to use. The job runs MPI-based parallelization between `maple` and `pine` and also runs two thread parallelization separately on both hosts.

mp_host_list=[['maple', 2], ['pine', 2]]

If **mp_host_list** is originally configured as follows:

mp_host_list=['maple', 4]

the original single host (`maple`) is split into two hosts, each with two processors to use, as follows:

mp_host_list=[['maple', 2], ['maple', 2]]

Then the job runs MPI-based parallelization between the first host (`maple`) and second host (`maple`) and also runs two thread parallelization separately on both hosts.

Abaqus/CAE Usage: Hybrid execution is not supported in Abaqus/CAE.

GPGPU Acceleration

GPGPU acceleration is supported in the direct sparse solver (*Direct Linear Equation Solver*), AMS eigensolver (*Selecting the Eigenvalue Extraction Method*), and modal frequency response solver (*Mode-Based Steady-State Dynamic Analysis*).

Input File Usage: Enter the following input on the command line to specify the total number of graphic processing units, *n*:

abaqusjob=*job-name* **gpus**=*n*

Abaqus/CAE Usage: Job module: job editor: **Parallelization**: toggle on **Use GPGPU acceleration**, and specify the total number of graphic processing units, *n*

Parallel Execution of Steady-State Dynamic Analyses

Steady-state dynamic analysis provides the steady-state response (also called frequency response) of a system due to harmonic excitation at a given frequency. Usually such analysis is done as a frequency sweep by applying the loading at a series of different frequencies and calculating the response. Abaqus/Standard supports three types of steady-state dynamic analysis:

- *Direct-Solution Steady-State Dynamic Analysis*
- *Mode-Based Steady-State Dynamic Analysis*
- *Subspace-Based Steady-State Dynamic Analysis*

Calculation of the frequency response for large finite element systems can be time consuming. Parallel execution is a practical option for steady-state dynamic analyses with millions of degrees of freedom, thousands of frequencies, and hundreds of load cases.

In mode-based steady-state dynamic analysis and subspace-based steady-state dynamic analysis, the response calculation is based on solution of the steady-state dynamic equations projected onto a subspace of modes. The modes of the undamped system must first be extracted using the eigenfrequency extraction procedure. The number of modes extracted must be sufficient to model the dynamic response of the system adequately, which is a matter of judgment on your part. In the general case of a finite element model with damping, the projected dynamic system of equations has *number-of-modes* equations and a dense complex matrix.

Parallel Execution of Direct-Solution Steady-State Dynamic Analyses

Direct-solution steady-state dynamic analysis is the most accurate and the most computationally intensive of the steady-state dynamic analysis types. In a direct-solution steady-state analysis, the frequency response is calculated directly in terms of the physical degrees of freedom of the model using the direct sparse solver.

For large-scale direct-solution steady-state dynamic analyses, we recommend execution on computer clusters using hybrid MPI-based and thread-based parallelization and GPGPU acceleration, if the graphic processing units are available. It is recommended to use 1 GPGPU per MPI domain; specifying more GPGPUs per MPI domain may not scale well.

Input File Usage: Enter the following input on the command line to specify the total number of physical cores for all MPI domains *n*, the number of threads per MPI domain, and the total number of GPGPU devices *k* across all MPI domains (if they are available):

abaqusjob=*job-name* **cpus=***n* **threads_per_mpi_process=***m* **gpus=***k*

Example: Parallel Execution of Direct-Solution Steady-State Dynamic Analyses

In this example, you execute a direct-solution steady-state dynamic analysis with 100 frequencies on a cluster of machine hosts with $2 \times 20 = 40$ physical cores. You split the frequencies into three partitions: 33-33-34. You execute the analysis for each partition on two cluster nodes (with a total of $40 \times 2 = 80$ physical cores) using 8 MPI domains (10 physical cores per MPI domain). The combined output results are obtained in *job1.odb*.

Three steady-state dynamic analyses:

```
abaqus job=job1 cpus=80 threads_per_mpi_process=10 ssd_split=3
ssd_partition=1
abaqus job=job2 cpus=80 threads_per_mpi_process=10 ssd_split=3
ssd_partition=2
abaqus job=job3 cpus=80 threads_per_mpi_process=10 ssd_split=3
ssd_partition=3
```

Combining results:

```
abaqus odbcombine frequencyrange job1.odb job2.odb job3.odb
```

Parallel Execution of Mode-Based Steady-State Dynamic Analyses

In a mode-based steady-state dynamic analysis, the steady-state dynamic modal operator is combined from the modal stiffness, mass, and damping operators precalculated in the eigenfrequency extraction procedure. Solving the system of equations with respect to the generalized displacements can be costly for mode-based steady-state dynamic analysis with many thousands of modes, thousands of frequencies, and hundreds of load cases.

Abaqus offers a scalable high-performance hybrid CPU-GPU modal steady-state dynamics solver. Mode-based steady-state dynamic analysis cannot be executed in MPI mode. It can be executed in thread mode only within one node of a compute cluster, and it can take advantage of GPGPU acceleration, if the graphic processing units

are available. To achieve maximal computational efficiency of a mode-based steady-state dynamic analysis, run it on a dedicated computer with sufficient memory and use all available physical cores and GPGPU devices.

Input File Usage: Enter the following input on the command line to specify the total number of physical cores *n*, and the total number of GPGPU devices *k* (if they are available):

abaqusjob=*job-name* **cpus**=*n* **gpus**=*k*

Parallel Execution of Subspace-Based Steady-State Dynamic Analyses

A subspace-based steady-state dynamic analysis provides an approximate way to include frequency-dependent effects (such as frequency-dependent damping and viscoelastic effects) or nonsymmetric stiffness in the model. This solution technique is less accurate than direct-solution steady-state analysis, in particular if significant material damping or viscoelasticity with a high loss modulus is present, but it is computationally cheaper than direct-solution steady-state dynamics. The projection of the dynamic equilibrium equations onto a subspace of selected modes is performed at every frequency, which makes subspace-based steady-state dynamic analysis more expensive than mode-based steady-state dynamics. Only thread-based parallelization is available for subspace-based steady-state dynamic analysis including the operator projection and modal solution phases.

Input File Usage: Enter the following input on the command line to specify the total number of physical cores *n*:

abaqusjob=*job-name* **cpus**=*n*

Splitting Frequency Points Technique for Running Steady-State Dynamic Analyses

When sufficient computer resources are available, you can significantly reduce the time required to run steady-state dynamic analyses with many frequencies by splitting the frequency points into several partitions and executing the analysis for each partition independently. No changes in the job input file are required.

To execute a steady-state dynamic analysis for a partition, you must specify the total number of partitions and the current partition number. If you execute analyses for multiple partitions simultaneously on a computer cluster, it can be significantly faster than executing a single analysis for all frequencies.

Input File Usage: Enter the following input on the command line to specify the total number frequency partitions *s*, and the current partition number *p*; specify a unique *job-name* for every partition:

abaqusjob=*job-name* **ssd_split**=*s* **ssd_partition**=*p*

Splitting Frequency Points with Restart

You can use the frequency splitting technique only for a single-step analysis. If the analysis includes preloading steps or it includes an eigenfrequency extraction step calculating the modal subspace, you can use the restart capability to run a steady-state dynamic analysis with frequency splitting.

Input File Usage: Enter the following input on the command line to specify the total number frequency partitions *s*, and the current partition number *p*; specify a unique *job-name* for every partition but the same *old-job-name* for all partitions:

abaqusjob=*job-name* **oldjob**=*old-job-name* **ssd_split**=*s* **ssd_partition**=*p*

Example: Parallel Execution of Mode-Based Steady-State Dynamic Analyses

In this example you execute a mode-based steady-state dynamic analysis with 1000 frequencies on a cluster of machine hosts with $2 \times 20 = 40$ physical cores and a total of 2 GPGPU devices. You split the frequencies into 10 partitions of 100 frequencies each. The combined output results are obtained in *job1.odb*.

You run multiple steady-state dynamic analyses with a single step restart from the same eigenfrequency extraction analysis *job0* that is executed at a node of the same cluster. The combined output results are obtained in *job1.odb*.

Eigenfrequency extraction analysis:

```
abaqus job=job0
```

Ten steady-state dynamic analyses:

```
abaqus oldjob=job0 job=job1 cpus=40 gpus=2 ssd_split=10 ssd_partition=1
abaqus oldjob=job0 job=job2 cpus=40 gpus=2 ssd_split=10 ssd_partition=2
...
abaqus oldjob=job0 job=job10 cpus=40 gpus=2 ssd_split=10 ssd_partition=10
```

Combining results:

```
abaqus odbcombine frequencyrange job1.odb job2.odb ... job10.odb
```

Combining Results for Frequency Partitions

After the jobs for all partitions complete, you can combine the results into a single output database (`.odb`) file. You can combine only output results from analyses performed with the **resultsformat=**`odb` option.

Input File Usage: Enter the following input on the command line to combine results output for all frequencies. The file *job-name1.odb* will contain the combined result.

abaqusodbcombinefrequencyrange*job-name1.odbjob-name2.odbjob-name3.odb* ...

Consistency of Results

Some physical systems (systems that, for example, undergo buckling, material failure, or delamination) can be highly sensitive to small perturbations. For example, it is well known that the experimentally measured buckling loads and final configurations of a set of seemingly identical cylindrical shells can show significant scatter because of small differences in features such as boundary conditions, loads, and initial geometries. When simulating such systems, the physical sensitivities seen in an experiment can be manifested as sensitivities to small numerical differences caused by finite precision effects. Finite precision effects can lead to small numerical differences when running jobs on different numbers of processors. Therefore, when simulating physically sensitive systems, you might see differences in the numerical results (reflecting the differences seen in experiments) between jobs run on different numbers of processors. To obtain consistent simulation results from run to run, the number of processors should be constant.

Parallel Execution in Abaqus/Explicit

Products: Abaqus/Explicit Abaqus/CAE

References:

- *Obtaining Information*
- *Environment File Settings*
- *Controlling job parallel execution*

Overview

Parallel execution in Abaqus/Explicit:

- reduces run time for analyses that require a large number of increments;
- reduces run time for analyses that contain a large number of nodes and elements;
- produces analysis results that are independent of the number of processors used for the analysis;
- is available for shared memory computers using a thread-based loop level or thread-based domain decomposition implementation;
- is available for both shared memory computers and computer clusters using an MPI-based domain decomposition parallel implementation; and
- is available in hybrid mode using a combination of MPI and threads.

Invoking Parallel Processing

Parallelization in Abaqus/Explicit is implemented in two ways: domain level and loop level. The default and most efficient method is the domain-level parallelization method. It splits the model into topological domains and assigns each domain to a processor. The loop-level method should be used only in the rare cases where the domain-level method is not applicable or is inefficient. It parallelizes low-level loops that are responsible for most of the computational cost. The element, node, and contact pair operations account for the majority of the low-level parallelized routines.

Parallelization is invoked by specifying the number of processors to use.

Input File Usage: Enter the following input on the command line:

abaqusjob=*job-name* **cpus=***n*

For example, the following input runs the job “beam” on two processors:

```
abaqus job=beam cpus=2
```

Abaqus/CAE Usage: Job module: job editor: **Parallelization**: toggle on **Use multiple processors**, and specify the number of processors, *n*

Domain-Level Parallelization

The domain-level method splits the model into a number of topological domains. These parallel domains contain a subset of the nodes and elements and all the modeling features necessary to compute the solution. The domains are distributed evenly among the available processors. The analysis is then carried out independently in each domain. However, information must be passed between the domains in each increment because the domains share common boundaries. The domain-level method is supported with MPI, thread-based parallelization, and in hybrid mode (a combination of MPI and threads).

During initialization, the domain-level method divides the model so that the resulting domains take approximately the same amount of computational expense. The load balance is defined as the ratio of the computational expense of all domains in the most expensive process to that of all domains in the least expensive process. For cases exhibiting significant load imbalance, either because the initial load balancing is not adequate (static imbalance) or because imbalance develops over time (dynamic imbalance), the dynamic load balancing technique might be applied (see *Abaqus/Standard and Abaqus/Explicit Execution*). Dynamic load balancing is based on over-decomposition: the user selects a number of domains that is a multiple of the number of processors. During the calculation, Abaqus/Explicit will regularly measure the computational expense and redistribute the domains over the processors so as to minimize the load imbalance. The following functionality is not supported with dynamic load balancing:

- Selective subcycling (*Selective Subcycling*)
- Co-simulation (*About Co-Simulation*)
- Predefined fields using a results file (*Predefined Fields*)
- Hybrid execution

The efficiency of the dynamic load balancing scheme depends on the load imbalance inherent to the problem, on the degree of overdecomposition, and on the efficiency of the hardware. Most imbalanced problems will see optimal performance improvement when the number of domains is two to four times the number of processors. The efficiency might be significantly reduced on systems with a slow interconnect, such as Gigabit Ethernet clusters. Best results are obtained when an external interconnect is not needed, such as within a multicore node of a cluster, or on a shared-memory system. Applications most likely to benefit from dynamic load balancing are problems with a strongly time-dependent or spatially varying computational load. Examples are models containing airbags (where contact-impact activity is highly localized and time dependent) and coupled Lagrangian-Eulerian problems (where constitutive activity follows the material as it moves through empty space).

Element and node sets are created for each domain and can be inspected in Abaqus/CAE. The sets are named `domain_`*n*, where *n* is the domain number.

Input File Usage: Enter the following input on the command line:

abaqusjob=*job-name* **cpus=***n* **parallel=**`domain` **domains=***m* **dynamic_load_balancing=**`on`

For example, the following input runs the job “beam” on two processors with the domain-level parallelization method:

abaqus job=beam cpus=2 parallel=domain domains=2

The domain-level parallelization method can also be set in the environment file using the environment file parameters **parallel=**`DOMAIN` and **domains**.

Abaqus/CAE Usage: Job module: job editor: **Parallelization**: toggle on **Use multiple processors** and specify the number of processors, *n*; **Number of domains:** *m*;

toggle on **Activate dynamic load balancing**; **Parallelization method**: **Domain**

You can activate dynamic load balancing when the number of domains is a multiple of the number of processors.

Thread-Based Execution

Abaqus/Explicit can be executed in thread mode within one node of a compute cluster and takes advantage of the shared memory available to the threads that are running on different cores.

Input File Usage: Enter the following input on the command line:

abaqusjob=*job-name* **cpus=***n* **threads_per_mpi_process=***n*

For example, the following input runs the job "beam" on 20 cores with thread-based domain-level parallelization using 20 threads:

```
abaqus job=beam cpus=20 threads_per_mpi_process=20
```

Abaqus/CAE Usage: Job module: job editor: **Parallelization**: toggle on **Use multiple processors** and specify the number of processors, *n*; **Number of domains:** *m*; **Multiprocessing mode**: **Threads**

MPI-Based Execution

Abaqus/Explicit can be executed in MPI mode, which uses the message passing interface (MPI) to communicate between processes running on different cores that might be spread over multiple compute nodes on an HPC cluster.

Input File Usage: Enter the following input on the command line:

abaqusjob=*job-name* **cpus=***n*

For example, the following input runs the job "beam" on 20 cores with MPI-based domain-level parallelization using 20 MPI processes:

```
abaqus job=beam cpus=20
```

Abaqus/CAE Usage: Job module: job editor: **Parallelization**: toggle on **Use multiple processors** and specify the number of processors, *n*; **Number of domains:** *m*; **Multiprocessing mode**: **MPI**

Hybrid Execution

Abaqus/Explicit can be executed in hybrid mode using a combination of MPI and threads with each MPI process launching a user-specified number of threads. Execution in hybrid mode is invoked by setting the command line option **threads_per_mpi_process=***m*. The total number of cpus must be divisible by the number of threads per MPI process.

HPC clusters typically consist of compute nodes with multiple sockets, and each socket contains numerous cores. In addition, the cores within a socket can be further organized into multiple NUMA (non-uniform memory

access) nodes with dedicated resources, such as local memory, for faster access. Hybrid execution of Abaqus/Explicit takes advantage of the NUMA architecture and the trend of increasing number of cores available on each NUMA node. It is recommended that the threads per MPI process parameter be set equal to the number of cores in a NUMA node, resulting in one MPI rank per NUMA node. However, when the number of cores in a NUMA node is large, you can experiment to allow multiple MPI ranks within a NUMA node. Furthermore, the hybrid execution is recommended only for jobs engaging one or more compute nodes fully on a homogeneous cluster. When submitting to a batch queuing system, the processors per node (ppn) should be set to the total number of cores available per compute node.

The number of domains is selected by Abaqus/Explicit automatically and does not need to be specified. If specified, Abaqus/Explicit may tune the number of domains for optimal performance. The number of domains is listed in the status (*job-name*`.sta`) file.

Input File Usage: Enter the following input on the command line:

abaqusjob=*job-name* **cpus=***n* **threads_per_mpi_process=***m*

For example, the following input runs the job "beam" on 80 cores with a hybrid MPI- and thread-based domain-level parallelization method using 4 MPI processes and 20 threads per MPI process:

abaqus job=beam cpus=80 threads_per_mpi_process=20

Abaqus/CAE Usage: Job module: job editor: **Parallelization**: toggle on **Use multiple processors** and specify the number of processors, *n*; **threads_per_mpi_process:** *m*; **Multiprocessing mode**: **Hybrid**

Consistency of Results

The analysis results are independent of the number of processors used for the analysis. However, the results do depend on the number of parallel domains used during the domain decomposition. Except for cases in which the single- and multiple-domain models are different because of features that are not yet available with multiple parallel domains (discussed below), these differences should be triggered only by finite precision effects. For example, the order of the nodal force assembly may depend on the number of parallel domains, which can result in differences in trailing digits in the computed force. Some physical systems are highly sensitive to small perturbations, so a tiny difference in the force applied in one increment can result in noticeable differences in results in subsequent increments. Simulations involving buckling and other bifurcations tend to be sensitive to small perturbations.

To obtain consistent analysis results from run to run, the number of domains used in the domain decomposition should be constant. Increasing the number of domains increases the computational cost slightly; therefore, unless dynamic load balancing is being applied, it is recommended that the number of domains be set equal to the maximum number of processors used for analysis execution for optimal performance. If you do not specify the number of domains, the number defaults to the number of processors.

Features that Do Not Allow Domain-Level Parallelization

The use of the domain-level parallelization method is not allowed with the following features:

- Extreme value output. An alternative is to filter the output.
- Steady-state detection. This feature is typically used with ALE adaptive mesh domains.

If you include these features, Abaqus issues an error message.

Features that Cannot Be Split across Domains

Certain features cannot be split across domains. The domain decomposition algorithm automatically takes this into account and forces these features to be contained entirely within one domain. If fewer domains than requested processors are created, Abaqus/Explicit issues an error message. Even if the algorithm succeeds in creating the requested number of domains, the load may be balanced unevenly. If this behavior is not acceptable, the job should be run with the loop-level parallelization method.

If you define tracer particles in the adaptive mesh domains, the adaptive mesh domains cannot span parallel domain boundaries. In this case, the nodes on the boundary between an adaptive mesh domain and a nonadaptive domain as well as the adaptive nodes on the surface of the adaptive mesh domain cannot be shared with another parallel domain. To enforce this in a consistent manner when you specify parallel domains, Abaqus/Explicit sets all nodes shared by adjacent adaptive mesh domains as nonadaptive. The analysis results may be significantly different from that of a serial run with no parallel domains. If this behavior is undesirable, set the number of parallel domains to 1, and switch to the loop-level parallelization method. See *Defining ALE Adaptive Mesh Domains in Abaqus/Explicit* for details.

A contact pair cannot be split across parallel domains, but separate contact pairs are not restricted to be in the same parallel domain. A contact pair that uses the kinematic contact algorithm requires that all the nodes associated with the involved surfaces be within a single parallel domain and not be shared with any other parallel domains. A contact pair that uses the penalty contact algorithm requires that the associated nodes be part of a single parallel domain, but these nodes may also be part of other parallel domains if they are not defined in an adaptive mesh domain. Analyses in which a large percentage of nodes are involved in contact might not scale well if contact pairs are used, especially with kinematic enforcement of contact constraints. General contact does not limit the domain decomposition boundaries.

Nodes involved in kinematic constraints (*About Kinematic Constraints*), except for surface-based shell-to-solid constraints, will be within a single parallel domain; and they will not be shared with another parallel domain. However, two kinematic constraints that do not share nodes can be placed within different parallel domains.

In some cases beam elements that share a node may be forced into the same parallel domain. This happens only for beams whose center of mass does not coincide with the location of the beam node or for beams with additional inertia (see *Adding Inertia to the Beam Section Behavior for Timoshenko Beams*).

User Influence on Domain Decomposition

You can influence the domain decomposition by specifying one or more regions that are independently decomposed into a user-specified number of parallel domains or by specifying that an element set should be constrained to the same parallel domain.

Specifying a domain decomposition region can be useful when a local region of the model is computationally intensive. Performance gains may be achieved by identifying the local region as an independent domain decomposition region, thereby distributing computation of the local regions among all processors. You can specify the domain decomposition region by defining an element set directly, or Abaqus/Explicit can generate the domain decomposition region consisting of all elements within a user-specified box. The part of the model that is not included in any user-specified domain decomposition region is considered as the global region and is also decomposed into the user-specified number of parallel domains. You can specify that each decomposition region can be decomposed using a recursive coordinate bisection (RCB) algorithm or a graph partitioning algorithm that minimizes the number of shared nodes. The RCB algorithm is the default for all domain decomposition regions. You can also specify that each domain decomposition region can be decomposed into

$N * n_{domainsUser}$ domains by specifying a decompose factor N. The domains from each independent domain decomposition are distributed evenly among the available processors, but these domains can be reassigned to different processors during the analysis if dynamic load balancing is activated. The total number of parallel domains for the simulation is

$$n_{domains} = \left(n_{globalRegions} * N_g + \sum_{i=1}^{n_{local\ Regions}} N_i \right) * n_{domainsUser},$$

where

$n_{localRegions}$

is the number of local regions identified as independent domain decomposition regions;

$n_{globalRegions}$

is equal to 1 if any elements are not included in local regions identified as independent domain decomposition regions; otherwise, $n_{globalRegions}$ is 0;

N_i

is the decompose factor for domain decomposition region i;

N_g

is the decompose factor for the global domain decomposition region; and

$n_{domainsUser}$

is the user-specified number of domains per domain decomposition region (see *Domain-Level Parallelization*).

Separate domain decomposition regions might be desired, for example, in bird-strike models (where contact-impact activity is highly localized and time dependent) and coupled Eulerian-Lagrangian problems with localized adaptive mesh refinement (where elements are refined adding to the computational cost). The example below (*Figure 1*) shows a spherical projectile impacting a flat plate with a failure model, thus allowing the projectile to perforate the plate. One of the domains contains the projectile as well as a significant portion of the impact area. Specifying a domain decomposition region consisting of the projectile as well as the computationally intensive impact area results in a more balanced parallel processing (*Figure 2*). In this example $n_{localRegions} = 1$ and $n_{globalRegions} = 1$; therefore, $n_{domains} = 2 * n_{domainsUser}$.

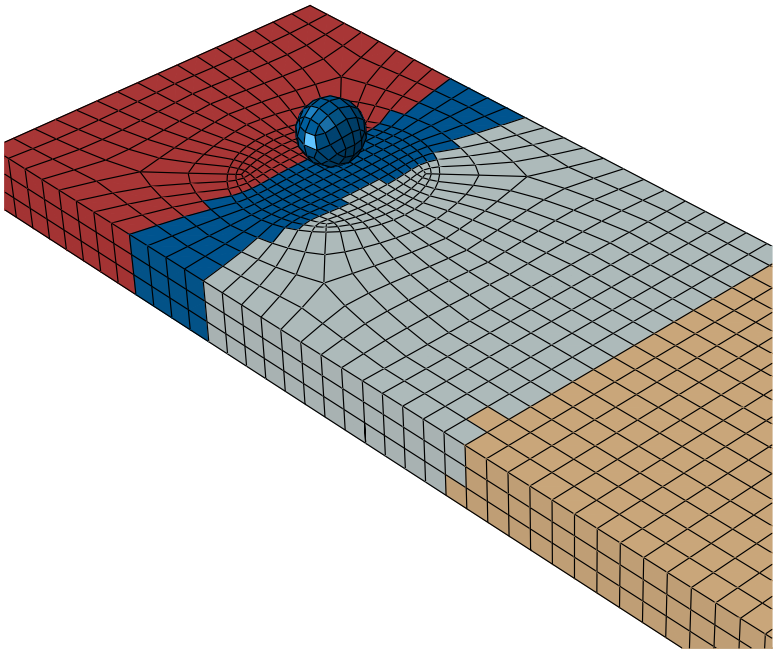


Figure 1: Original domain decomposition.

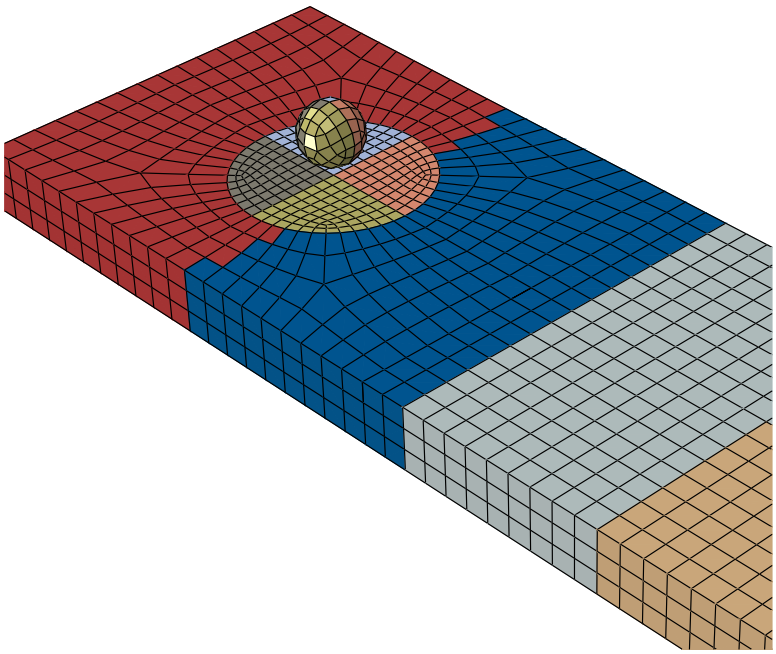


Figure 2: Modified domain decomposition.

Multiple domain decomposition regions can be specified. In the case of overlap between the domain decomposition regions, by default, the first specified decomposition keeps the overlapped elements. Some modeling features

cannot be split across domains, and Abaqus/Explicit automatically merges the domain decomposition regions that contain features that cannot be split.

Input File Usage: Use the following option to define a domain decomposition region using a user-specified element set:

**DOMAIN DECOMPOSITION*, DEFINITION=ELSET,ELSET=*element_set_name*

Use the following option to define a domain decomposition region by specifying that Abaqus/Explicit should generate the domain decomposition region consisting of all elements contained within the user-specified box:

**DOMAIN DECOMPOSITION*, DEFINITION=BOX,ELSET=*element_set_name*

Use the following option to decompose the region using a recursive coordinate bisection algorithm (default):

**DOMAIN DECOMPOSITION*, METHOD=RCB

Use the following option to decompose the region using a graph partitioning algorithm:

**DOMAIN DECOMPOSITION*, METHOD=GRAPH PARTITIONING

Use the following option to specify a decompose factor for the region (default is 1):

**DOMAIN DECOMPOSITION*, DECOMPOSE FACTOR=*N*

Use the following option to constrain an element set to the same parallel domain:

**DOMAIN DECOMPOSITION*
element_set_name, SAME DOMAIN

Abaqus/CAE Usage: Domain decomposition is not supported in Abaqus/CAE.

Restart

There are certain restrictions for restart when using domain-level parallelization. To ensure that optimal parallel speedup is achieved, the number of processors used for the restart analysis must be chosen so that the number of parallel domains used during the original analysis can be distributed evenly among the processors. Because the domain decomposition is based only on the features specified in the original analysis and steps defined therein, features that affect domain decomposition are restricted from being defined in restart steps only if they would invalidate the original domain decomposition. Because the newly added features will be added to existing domains, there is a potential for load imbalance and a corresponding degradation of parallel performance.

The restart analysis requires that the separate state and selected results files created during the original analysis be converted into single files, as described in *Abaqus/Standard and Abaqus/Explicit Execution*. This should be done automatically at the conclusion of the original analysis. If the original analysis fails to complete successfully, you must convert the state and selected results files prior to restart. An Abaqus/Explicit analysis packaged to run with a domain-level parallelization technique cannot be restarted or continued with a loop-level parallelization technique.

Co-Simulation

The co-simulation technique (*About Co-Simulation*) for run-time coupling of Abaqus/Explicit to Abaqus/Standard or to third-party analysis programs can be used with Abaqus/Explicit running either in serial or parallel.

Loop-Level Parallelization

The loop-level method parallelizes low-level loops in the element and contact pair code only. The speedup factor using loop-level parallelization may be significantly less than what can be achieved with domain-level parallelization. The speedup factor will vary depending on the features included in the analysis since not all features utilize parallel loops. Examples are the general contact algorithm and kinematic constraints. The loop-level method may scale poorly for more than four processors depending on the analysis. Using multiple parallel domains with this method will degrade parallel performance and, hence, is not recommended. The loop-level method is not supported on the Windows platform.

Analysis results for this method do not depend on the number of processors used.

Input File Usage: Enter the following input on the command line:

abaqusjob=*job-name* **cpus=***n* **parallel=**`loop`

The loop-level parallelization method can also be set in the environment file using the environment file parameter **parallel=**`LOOP`.

Abaqus/CAE Usage: Job module: job editor: **Parallelization**: toggle on **Use multiple processors**, and specify the number of processors, *n*; **Parallelization method: Loop**

Restart

There are no restrictions on features that can be included in steps defined in a restart analysis when using loop-level parallelization. For performance reasons the number of processors used when restarting must be a factor of the number of processors used in the original analysis. The most common case would be restarting with the same number of processors as used in the original analysis. An Abaqus/Explicit analysis packaged to run with a loop-level parallelization technique cannot be restarted or continued with a domain-level parallelization technique.

Measuring Parallel Performance

Parallel performance is measured by comparing the total time required to run on a single processor (serial run) to the total time required to run on multiple processors (parallel run). This ratio is referred to as the speedup factor. The speedup factor will equal the number of processors used for the parallel run in the case of perfect parallelization. Scaling refers to the behavior of the speedup factor as the number of processors is increased. Perfect scaling indicates that the speedup factor increases linearly with the number of processors. For both parallelization methods the speedup factors and scaling behavior are heavily problem dependent. In general, the domain-level method will scale to a larger number of processors and offer the higher speedup factor.

Output

There are no output restrictions.

File Extension Definitions

Products: Abaqus/Standard Abaqus/Explicit

References:

- *About Execution Procedures*

Overview

The **abaqus** procedure generates several files. Some of these files contain analysis, postprocessing, and translation results and are retained for use by other analysis options, restarting, or postprocessing. This section describes the files that are created and retained by Abaqus.

Other files exist only while Abaqus is executing and are deleted when a run completes. These temporary files are generated in the scratch directory. The number and types of temporary files generated depend on the analysis procedures, memory management parameters, and environment settings.

Certain file extensions used by Abaqus are also used by other software applications. You must handle any file extension conflicts with other applications.

File Extensions

`abq`

State file, only used by Abaqus/Explicit. It is written by the **analysis**, **continue**, and **recover** options. It is read by the **convert** and **recover** options. This file is required for restart.

`axi`

Symmetric model data file, only used by Abaqus/Standard. It is written during symmetric model generation by the **datacheck** and **analysis** options.

`bsp`

Text file containing beam cross-section properties for meshed section profiles. It is written by Abaqus/Standard during meshed beam cross-section generation.

`c`

User subroutine or other special-purpose C file.

`c++`

User subroutine or other special-purpose C++ file.

cax

File containing information required by Abaqus/Standard for continuing a data check run. It is written by the **datacheck** option and is read by the **continue** option.

cpp

User subroutine or other special-purpose C++ file.

cid

Auto-release file, which contains information needed for license recovery and suspension.

com

Command file, created by the Abaqus execution procedure.

dat

Printed output file. It is written by the **analysis**, **datacheck**, **parametercheck**, and **continue** options. Abaqus/Explicit does not write analysis results to this file.

f

User subroutine or other special-purpose Fortran file.

fil

Results file. It is written by the **analysis** and **continue** options in Abaqus/Standard and by the **convert**=`select` and **convert**=`all` options in Abaqus/Explicit.

fin

Results file created when changing the format of the `.fil` file using the **abaqus ascfil** command. It can be in either ASCII or binary format. (See *ASCII Translation of Results (.fil) Files*.) The ASCII format is convenient for data transfer between machines that do not have compatible binary data formats.

inp

Analysis input file. It is read when the **analysis**, **datacheck**, and **parametercheck** options are selected.

ipm

Interprocess message file. It is written when an analysis is run from Abaqus/CAE, and it contains a log of all messages sent from Abaqus/Standard or Abaqus/Explicit to Abaqus/CAE.

lck

Lock file for the output database. This file is written whenever an output database file is opened with write access; it prevents you from having simultaneous write permission to the output database from multiple sources. It is deleted automatically when the output database file is closed or when the analysis that creates it ends. The **ask_delete** environment file parameter setting will not affect the lock file.

log

Log file, which contains start and end times for modules run by the current Abaqus execution procedure.

mdl

Model file, used by Abaqus/Standard and Abaqus/Explicit. It is written by the **datacheck** option. It is read and can be written by the **analysis** and **continue** options in Abaqus/Standard. It is read by the **analysis** and **continue** options in Abaqus/Explicit. Multiple model files may exist if the element operations are executed in parallel in an Abaqus/Standard analysis. In such a case a process identifier is attached to the file name. This file is required for restart.

msg

Message file. It is written by the **analysis**, **datacheck**, and **continue** options in Abaqus/Standard and Abaqus/Explicit. Multiple message files may exist if the element operations are executed in parallel in an Abaqus/Standard analysis. In such a case a process identifier is attached to the file name.

nck

Nickname file used by Abaqus/Standard. It stores a set of internal identifiers for the degrees of freedom in a model.

odb

Output database. It is written by the **analysis** and **continue** options in Abaqus/Standard and Abaqus/Explicit. It is read by the Visualization module in Abaqus/CAE (Abaqus/Viewer) and by the **convert=**`odb` option. This file is required for restart.

pac

Package file, which contains model information and is used by Abaqus/Explicit only. It is written by the **analysis** and **datacheck** options. It is read by the **analysis**, **continue**, and **recover** options. This file is required for restart.

par

Modified version of original parametrized input file showing input parameters and their values.

pes

Modified version of original parametrized input file showing input free of parameter information (after input parameter evaluation and substitution has been performed).

pmg

Parameter evaluation and substitution message file. It is written when the input file is parametrized.

prt

Part file, used by Abaqus/Standard and Abaqus/Explicit. This file is used to store part and assembly information and is created even if the input file does not contain an assembly definition. The part file is

required for restart, import, sequentially coupled thermal-stress analysis, symmetric model generation, and underwater shock analysis, even if the model is not defined in terms of an assembly of part instances. This file may also be needed for submodeling analysis.

psf

Python scripting file. You must create this type of file to define a parametric study.

res

Restart file, which contains information necessary to continue a previous analysis and is used by Abaqus/Standard and Abaqus/Explicit. The restart file is written by the **analysis**, **datacheck**, and **continue** options. It is read by any restarted analysis.

sel

Selected results file, used by Abaqus/Explicit. It is written by the **analysis**, **continue**, and **recover** options and is read by the **convert**=`select` option. This file is required for restart.

sim

Model and results file, used by Abaqus/Standard and Abaqus/Explicit when the **resultsformat**=`sim` or `both` option is specified. It is written by the **datacheck** option. It is read and can be written by the **analysis** and **continue** options. This file is required for restart.

Linear dynamics data file, used by Abaqus/Standard when the **resultsformat**=`odb` option is specified. It is written during the frequency extraction procedure in SIM-based linear dynamics analyses (see *Using the SIM Architecture for Modal Superposition Dynamic Analyses* for details) and is used to store eigenvectors, substructure matrices, and other modal system information. This file is required for restart.

sta

Status file. Abaqus writes increment summaries to this file in the **analysis**, **continue**, and **recover** options.

stt

State file. It is written by the **datacheck** option in Abaqus/Standard and Abaqus/Explicit. It is read and can be written by the **analysis** and **continue** options in Abaqus/Standard. It is read by the **analysis** and **continue** options in Abaqus/Explicit. Multiple state files may exist if the element operations are executed in parallel in an Abaqus/Standard analysis. In such a case a process identifier is attached to the file name. This file is required for restart.

sup

Substructure file, used by Abaqus/Standard.

var

File containing information about the input file variations generated by a parametric study.

023

Communications file, used by Abaqus/Standard and Abaqus/Explicit. It is written by the **analysis** and **datacheck** options and is read by the **analysis** and **continue** options.

Fortran Unit Numbers

Products: Abaqus/Standard Abaqus/Explicit

References:

- *About Execution Procedures*

Overview

Abaqus uses the Fortran unit numbers outlined in the table below. Unless noted otherwise, you should not try to write to these Fortran units from user subroutines.

For Abaqus/Standard, you should specify unit numbers 15–18 or unit numbers greater than 100 .

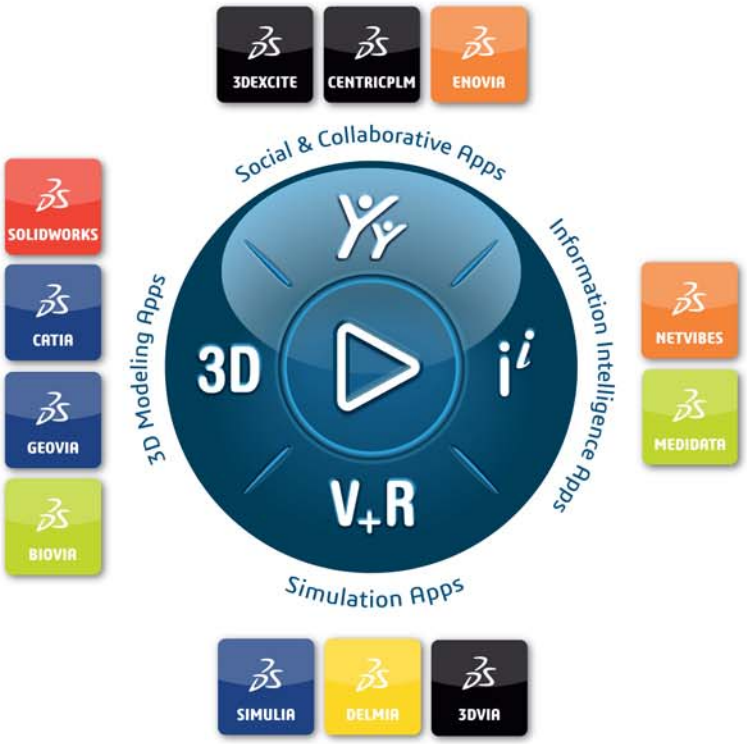
For Abaqus/Explicit, specify units 16–18 or unit numbers greater than 100 ending in 5 to 9, e.g. 105, 268, etc. You cannot write to the `.sta` file.

Fortran Unit Numbers

Code	Unit Number	Description
Abaqus/Standard	1	Internal database
	2	Solver file
	6	Printed output (`.dat`) file (You can write output to this file.)
	7	Message (`.msg`) file (You can write output to this file.)
	8	Results (`.fil`) file
	10	Internal database
	12	Restart (`.res`) file
	19–30	Internal databases (scratch files). Unit numbers 21 and 22 are always written to disk.
	73	Text file containing meshed beam cross-section properties (`.bsp`)

Code	Unit Number	Description
Abaqus/Explicit	6	Printed output (`.log`) .
	12	Restart (`.res`) file
	13	Old restart (`.res`) file, if applicable
	15	Analysis Preprocessor (`.dat` or `.pre`) file
	23	Communications (`.023`) file
	60	Global package (`.pac`) file
	61	Global state (`.abq`) file

Code	Unit Number	Description
	62	Temporary file
	63	Global selected results (`.sel`) file
	64	Message (`.msg`) file
	65	Output database (`.odb`) file
	67	Old package (`.pac`) file, if import from Abaqus/Explicit
	68	Old state (`.abq`) file, if import from Abaqus/Explicit
	69	Internal database; temporary file
If domain-parallel	70	Local package (`.pac.1`) file for CPU #1
	71	Local state (`.abq.1`) file for CPU #1
	73	Local selected results (`.sel.1`) file for CPU #1
	80	Local package (`.pac.2`) file for CPU #2
	81	Local state (`.abq.2`) file for CPU #2
	83	Local selected results (`.sel.2`) file for CPU #2
	...	Add three files, incrementing units by 10, for each additional CPU



Our **3D**EXPERIENCE® platform powers our brand applications, serving 12 industries, and provides a rich portfolio of industry solution experiences.

Dassault Systèmes is a catalyst for human progress. We provide business and people with collaborative virtual environments to imagine sustainable innovations. By creating virtual twin experiences of the real world with our **3DEXPERIENCE** platform and applications, our customers can redefine the creation, production and life-cycle-management processes of their offer and thus have a meaningful impact to make the world more sustainable. The beauty of the Experience Economy is that it is a human-centered economy for the benefit of all – consumers, patients and citizens.

Dassault Systèmes brings value to more than 300,000 customers of all sizes, in all industries, in more than 150 countries. For more information, visit **www.3ds.com**.

Europe/Middle East/Africa
Dassault Systèmes
10, rue Marcel Dassault
CS 40501
78946 Vélizy-Villacoublay Cedex
France

Asia-Pacific
Dassault Systèmes
17F, Foxconn Building,
No. 1366, Lujiazui Ring Road
Pilot Free Trade Zone, Shanghai 200120
China

Americas
Dassault Systèmes
175 Wyman Street
Waltham, Massachusetts
02451-1223
USA

DASSAULT SYSTEMES

©2025 Dassault Systèmes. All rights reserved. **3DEXPERIENCE**, the 3DS logo, the Compass icon, IFWE, 3DEXCITE, 3DVIA, BIOVIA, CATIA, CENTRIC PLM, DELMIA, ENOVIA, GEOVIA, MEDIDATA, NETVIBES, OUTSCALE, SIMULIA and SOLIDWORKS are commercial trademarks or registered trademarks of Dassault Systèmes, a European company (Societas Europaea) incorporated under French law, and registered with the Versailles trade and companies registry under number 322 306 440, or its subsidiaries in the United States and/or other countries.