SIMULIA

3DEXPERIENCE

ABAQUS 2025 FD02
CONFIGURATION GUIDE

DASSAULT SYSTEMES

Contents

Trademarks and Legal Notices

Trademarks

Legal Notices

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*.

What's New

This page describes recent changes in Abaqus Configuration.

2025 FD01

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 .

2024 FD01

threads_per_mpi_process Parallelization Parameter

The threads_per_mpi_process parameter can be used to select between three different parallel execution modes: pure MPI-based (DMP), pure Threads-based (SMP), and a hybrid of MPI and threads (HMP).

Benefits: The new parameter helps users customize parallel execution in Abaqus.

For more information, see *Parallelization parameters*.

Abaqus Configuration Guide

The Abaqus Configuration Guide contains information about customizing Abaqus 2025.

In addition, the guides in the Abaqus® documentation collection describe the capabilities of the Abaqus finite element analysis technology used in SIMULIA® applications.

For a description of the installation procedures, see the *SIMULIA Installation Guide*.

The environment file provides options for customizing an Abaqus installation. Specifying the file system to be used to store scratch files is particularly important. For details on how to set local defaults and adjust the system configuration to run Abaqus jobs efficiently, see *Customizing the Abaqus environment*.

For more information on customizing the Abaqus user interface, printer configuration, and graphic card tuning, see *Customizing Abaqus/CAE and Abaqus/Viewer*.

Customizing the Abaqus environment

This chapter describes how to use environment file parameters to customize the Abaqus execution procedure, including how to define analysis batch queues.
Example files are provided at the end of the chapter.

In this section:

Using the Abaqus environment files

The Abaqus execution procedure reads environment files to determine the various parameters that are used to run a job.
(See the *Abaqus Execution Guide*.)

When Abaqus starts, it searches three directories for the main environment file, `abaqus_v6.env`, in the following order:

1. The *install_dir*`/`*os*`/SMA/site/` subdirectory of the installation (see *Installation subdirectories*). An `abaqus_v6.env` file must exist in this directory.

   The parameter settings in this file are ignored when a job is submitted to a remote queue; in this case, the settings in *install_dir*`/`*os*`/SMA/site/abaqus_v6.env` on the remote computer are used instead.

   Starting in Abaqus 2016, the `abaqus_v6.env` file imports and uses any parameter definitions found in the following auxiliary environment files:

   - `custom_v6.env` should contain your site-specific settings such as licensing and documentation parameters
   - `basic_v6.env` contains general parameters common to all platforms
   - `win86_64.env` or `lnx86_64.env` contain platform-specific parameters such as compiler and MPI settings
   - `graphicsConfig.env` contains the onCaeGraphicsStartup() function to configure graphics-card-specific settings for Abaqus/CAE and Abaqus/Viewer

   You can add your parameter settings directly in the *install_dir*`/`*os*`/SMA/site/abaqus_v6.env` file; but the recommended best practice is to place your site-specific settings in the *install_dir*`/`*os*`/SMA/site/custom_v6.env` file.

2. An `abaqus_v6.env` file in the user's home directory. This environment file is optional and will affect all jobs submitted from the user's account. Each user should include in this file only the parameters that they specifically want to change. This file should not be a complete copy of the *install_dir*`/`*os*`/SMA/site/abaqus_v6.env` file.

   On Windows platforms, for Abaqus to locate the environment file in the user's home directory, the full path to the user's home directory must be specified using the `HOME` environment variable or a combination of the `HOMEDRIVE` and `HOMEPATH` environment variables.

3. An `abaqus_v6.env` file in the current working directory. This environment file is optional and will affect all jobs submitted from the current working directory. Each user should include in this file only the parameters that they specifically want to change. This file should not be a complete copy of the *install_dir*`/`*os*`/SMA/site/abaqus_v6.env` file.

If the same parameter is defined in more than one of these environment files or is defined more than once within a file, the last definition encountered will be used.

An individual user can override the site-wide parameter settings in the *install_dir*`/`*os*`/SMA/site/custom_v6.env` and *install_dir*`/`*os*`/SMA/site/abaqus_v6.env` files by creating a new file named `abaqus_v6.env` in their home directory or in the current working directory. Any parameters set in these files will be read last by Abaqus (in the order described above), overriding the site-wide values.

If you are upgrading from an earlier release, do not simply include parameters from the `abaqus_v6.env` file from previous releases of Abaqus without checking that they are necessary.

In this section:

Environment file syntax

The environment file for Abaqus uses Python syntax. See the *Abaqus Scripting User's Guide* for more detailed information on Python syntax.

Environment file entries have the following syntax:

```
parameter=value
```

All parameters must have a value. The following syntactic rules also apply:

- All parameters are case sensitive.
- 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.
- Lists must be enclosed in square brackets ([ ]). Individual items in the list are separated by commas. Entries take the form:

  ```
  parameter=[value1, value2, value3]
  ```

- Tuples must be enclosed in parentheses (( )). Individual items in the tuple are separated by commas. If the tuple is enclosed in parentheses and contains only one value, a comma has to follow the value. Entries take the form:

  ```
  parameter=(value1, value2)
  ```

- 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. Strings in a list or a tuple must be enclosed in quotes.
- Triple-quoted (""") strings can span more than one line, and no special treatment of quotes within the string is necessary. Entries take the form:

  ```
  parameter="""
  multi-line
  value
  """
  ```

Examples of many of the environment file parameters are available in the sample `abaqusinc.env` file found in the installation subdirectory *install_dir*`/`*os*`/SMA/site/`. Care should be taken when merging customized `abaqus_v6.env` settings from an earlier release into the current release. Settings from the earlier release may not be compatible with the new release.

Memory and disk management parameters

Management of memory and disk resources for Abaqus/Standard and Abaqus/Explicit is discussed in detail in *Managing Memory and Disk Resources*. The relevant parameters are listed here along with a single parameter, **abq_ker_memory**, that is used to manage memory in Abaqus/CAE and Abaqus/Viewer.

The available units for memory sizes are mb (megabytes) and gb (gigabytes). If the units are not specified, the size is assumed to be in megabytes.

abq_ker_memory

The maximum amount of memory that can be allocated by the Abaqus/CAE and Abaqus/Viewer kernel, specified in MB (megabytes). If the limit is exceeded, Abaqus/CAE displays an error message.

If the kernel memory size reaches the **abq_ker_memory** value or the virtual memory limit of the machine, the following message will be displayed: `Operation did not complete due to a memory allocation failure.`

For optimal performance, the memory limit should be set to a value less than the physical amount of memory on the machine. The minimum setting allowed is 256 MB.

scratch

Full path name of the directory to be used for scratch files. The default value on Linux is the value of the `$TMPDIR` environment variable or `/tmp` if `$TMPDIR` is not defined. On Windows the default value is the value of the `%TEMP%` environment variable or `\TEMP` if `%TEMP%` 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 the user's user name, the job ID, and the job's process number. The subdirectory and its contents are deleted upon completion of the analysis.

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. The default value is 80% of the machine physical memory.

Parallelization parameters

You can set parameters to customize parallel execution in Abaqus.

Parallelization in Abaqus is discussed in detail in the following sections:

- *Parallel Execution in Abaqus/Standard*
- *Parallel Execution in Abaqus/Explicit*

The relevant parameters are listed here.

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 if required. The default value is `ON`.

cpus

The total number of processors to use during an analysis run if parallel processing is available. The default value for this parameter is 1.

direct_solver

Set this variable equal to `DMP` to choose the hybrid direct sparse solver explicitly. Set this variable equal to `SMP` to choose the pure thread-based direct sparse solver explicitly. If this parameter is omitted, the default is to use the pure thread-based direct sparse solver.

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** variables. 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`.

gpus

Activate direct solver acceleration using GPGPU hardware in Abaqus/Standard. The value of this parameter should be the number of GPGPUs to use for an analysis. In an MPI-based parallel Abaqus/Standard analysis, this is the total number of GPGPUs to use across all hosts.

max_cpus

Maximum number of processors allowed if parallel processing is available. If this parameter is not set, the number of processors allowed equals the number of available processors on the system.

mp_file_system

Type of file system available for an MPI-based parallel Abaqus analysis. The parameter must be set to a tuple; for example,

```
mp_file_system=(SHARED,LOCAL)
```

The first item in the tuple refers to the directory where the job was submitted, while the second refers to the job's scratch directory. If the file system hosting a directory is `LOCAL`, Abaqus will copy the required analysis files to the remote host machines and, at the end of the run, copy the output files back. In this case it must be possible to create the job's directory structure on all the hosts in **mp_host_list**. A `SHARED` file system means that the host machines share the same file system and file transfer is not necessary. With the recommended default (`DETECT`, `DETECT`) setting, Abaqus will determine the type of file system that exists. An MPI-based parallel Abaqus/Explicit analysis will use the scratch directory only if a user subroutine is used, whereas Abaqus/Standard normally writes large temporary files in this directory. Running on a local file system will generally improve the performance.

mp_host_list

List of host machine names, typically set by the job submission system, to be used for a parallel Abaqus analysis, along with 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 to be greater than or equal to the number of **cpus** specified for the analysis. **mp_host_list** is modified to account for **threads_per_mpi_process** value as needed. 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.

mp_mpi_implementation

A dictionary or constant to specify the underlying MPI implementation to use. Generally, this variable does not need to be specified.

Dictionary values are used to apply an MPI implementation according to the solver used. Dictionary Keys are `STANDARD`, `EXPLICIT`, or `DEFAULTMPI`. Dictionary Values are `PMPI`, `IMPI`, or `NATIVE`. On Linux, the default setting is the following dictionary:

```
mp_mpi_implementation={DEFAULTMPI: IMPI, STANDARD: PMPI}
```

The constant values can be `PMPI`, `IMPI`, or `NATIVE`. A Constant value is used when applicable to all solvers. For example,

```
mp_mpi_implementation=PMPI
```

mp_mpirun_options

A dictionary or string of options passed to the MPI launcher for an MPI-based parallel Abaqus analysis. Generally, this variable does not need to be set.

Dictionary Keys are constants `PMPI`, `IMPI`, or `NATIVE`. For example,

```
mp_mpirun_options={PMPI: '-v -d -t', IMPI: '-verbose'}
```

A string is used when applicable to all MPI implementations. For example,

```
mp_mpirun_options='-v'
```

mp_mpirun_path

A dictionary to define the full path to the MPI launcher for a given MPI implementation. For example,

```
mp_mpirun_path={NATIVE: 'C:\\Program Files\\Microsoft MPI\\bin\\mpiexec.exe'}
```

```
mp_mpirun_path={PMPI:
'/apps/SIMULIA/Abaqus/2023/linux_a64/code/bin/SMAExternal/pmpi/bin/mpirun',

                IMPI:
'/apps/SIMULIA/Abaqus/2023/linux_a64/code/bin/SMAExternal/impi/bin/mpirun'}
```

mp_num_parallel_ftps

When performing parallel file staging using MPI-based parallelization, this parameter controls the number of simultaneous MPI file transfers. The first item controls the transfer of files to and from the temporary scratch directory. The second item controls the transfer of files to and from the analysis working directory. Setting either value to 1 disables the parallel file staging process. The use of file staging depends on the values specified in **mp_file_system**.

mp_rsh_command

Preferred command to open a remote shell on the machines specified by **mp_host_list**. Abaqus needs to open a remote shell to create and remove directories and files if the file system is not shared. The default value for this option is platform- dependent; for example,

```
mp_rsh_command='ssh -n -l %U %H %C'
```

The following placemarkers are used:

%U

Username.

%H

The host where the remote shell is opened.

%C

The command to be executed on the host.

Abaqus automatically uses secure copy (`scp`) to copy files to and from remote hosts if this parameter is set to use secure shell. By default, this parameter is ignored in favor of built-in MPI `rsh/scp` commands.

order_parallel

The default direct solver ordering mode in Abaqus/Standard if you do not specify the parallel ordering mode on the **abaqus** command line. If this parameter is set equal to `OFF`, the solver ordering will not be performed in parallel. If this parameter is set equal to `ON`, the solver ordering will be run in parallel. The default for parallel solver ordering is `ON`.

parallel

The default parallel equation solution method in Abaqus/Explicit if the user does not specify the parallel method on the **abaqus** command line. Possible values are `DOMAIN` or `LOOP`; the default value is `DOMAIN`.

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`.

threads_per_mpi_process

The **threads_per_mpi_process** option can be used with the **cpus** option to configure 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. For Abaqus/Explicit, the default value for this parameter is 1, resulting in pure MPI-based parallelization; whereas for Abaqus/Standard a suitable value is automatically set by default, with hybrid parallelization preferred when possible.

Job customization parameters

Job customization parameters are available to customize Abaqus analysis jobs.

abq_cosimulation_lower_port

This variable specifies the lowest port number in the range of TCP/UDP port numbers available for co-simulation between two Abaqus analyses; it is valid only for Abaqus/CAE. The default value is 48000.

abq_cosimulation_upper_port

This variable specifies the highest port number in the range of TCP/UDP port numbers available for co-simulation between two Abaqus analyses; it is valid only for Abaqus/CAE. If this value is not specified, it is set to 1000 more than **abq_cosimulation_lower_port**.

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 necessary. If this parameter is set to `OFF`, no output database postprocessing occurs even if the execution procedure detects that it is necessary to postprocess the output database file. The default value is `ANALYSIS`. In Abaqus/Explicit**auto_calculate**=`ANALYSIS` is equivalent to **auto_calculate**=`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.

cae_error_limit

This variable defines the maximum number of error messages that will be sent from an analysis job to Abaqus/CAE; it is valid only for Abaqus/CAE. The default value is 50.

cae_no_parts_input_file

This variable defines the format of the input file generated by Abaqus/CAE; it is valid only for Abaqus/CAE. If this variable is set to `ON`, Abaqus/CAE will generate an input file without parts and assemblies. The default value is `OFF`. For more information, see *Writing input files without parts and assemblies*.

cae_warning_limit

This variable defines the maximum number of warning messages that will be sent from an analysis job to Abaqus/CAE; it is valid only for Abaqus/CAE. The default value is 200.

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`.

max_history_requests

This parameter specifies the maximum number of history requests allowed in an Abaqus analysis. The default value is 10,000. History output in Abaqus is intended for relatively frequent output requests for small portions of a model and is displayed in *X–Y* plots in the Visualization module of Abaqus/CAE (Abaqus/Viewer). Requesting large amounts of history output will cause performance problems in both analysis and postprocessing of an Abaqus job. For vector- or tensor-valued output variables, each component is considered to be a single request. In the case of element variables, history output will be generated at each integration point. For example, requesting history output of the tensor variable S (stress) for a C3D10M element will generate 24 history output requests: (6 components) × (4 integration points). When requesting history output of vector- and tensor-valued variables, it is recommended that individual components be selected where applicable. In cases where large amounts of history output are required, it is recommended that the data be written to the output database (`.odb`) as field output from which history data can be extracted using the Visualization module of Abaqus/CAE.

odb_output_by_default

If this parameter is set to `ON`, output database output will be generated automatically. If this parameter is set to `OFF`, output database request keywords must be placed in an input file to obtain output database output and to allow the analysis to be restarted. The default value is `ON`.

onCaeGraphicsStartup

Optional function to be executed before Abaqus/CAE or Abaqus/Viewer begins. This function allows the user to change the graphics options. See *Tuning graphics cards*, for more information on this function. This function should not normally be changed.

onCaeStartup

Optional function to be executed before Abaqus/CAE begins. See *Customizing Abaqus/CAE startup*, for examples of this function.

onDesignStartup

Optional function to be executed before Abaqus/Design begins.

onJobCompletion

Optional function to be executed after the Abaqus job completes. A function specified in the Abaqus environment file in the current directory will be executed first, followed by the function in the user's home directory, and then the function in the Abaqus installation environment file. Multiple functions in the same environment file will result in only the last definition being used. See *Job variables*, for a list of variables available to this function.

onJobStartup

Optional function to be executed before the Abaqus job begins. See *Job variables*, for a list of variables available to this function.

printed_output

By default, the values of all **PREPRINT* parameters are NO and no results are printed to the data file. Set the **printed_output** parameter equal to `ON` to obtain the same preprint information in the data file as if

```
*PREPRINT, CONTACT=YES, ECHO=YES, HISTORY=YES, MODEL=YES
```

were included in the input file. Setting **printed_output** equal to `ON` can also cause a large volume of tabular results to be printed to the data file (unless printed output control options are used to limit the output). If the input file is in terms of parts and assemblies, setting **printed_output** equal to `ON` will cause the part-assembly map to print out in the data file regardless of the settings on the **PREPRINT* option; this allows the user to associate the printed output with the part-assembly defined in the input file. The default value for this variable is `OFF`.

run_mode

Default run mode (`INTERACTIVE`, `BACKGROUND`, or `BATCH`) if the user does not specify the run mode on the **abaqus** command line when running the analysis products. The default value is `BACKGROUND`. This variable should not be set to `BATCH` unless batch queues are defined.

split_dat

If this variable is set to `ON`, the data file will be split into two pieces. The output from the user input processing will be put in a file with a `.pre` extension. The analysis output file will still have a `.dat` file extension. The default value is `OFF`.

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.

System customization parameters

System customization parameters are available to customize various system settings, such as preventing unauthorized modification of environment file parameters.

admin

This parameter prevents unauthorized modification of environment file parameters. Set this parameter equal to a list of environment file parameters that cannot be changed in a lower-level `abaqus_v6.env` file. Unless otherwise noted, all system and job customization parameters can be locked out. Commands in the installation directory have the highest precedence, followed by commands in the user's home directory, and then commands in the current working directory. Thus, an Abaqus user cannot change environment file commands that were locked out by the Abaqus account manager.

ask_delete

If this parameter is set to `OFF`, the user will not be asked whether old job files of the same name should be deleted; the files will be deleted automatically. The default value is `ON`.

compile_cpp

C++ compile command. The command used at SIMULIA is included in the platform-specific environment file (`win86_64.env` or `lnx86_64.env`) in the *install_dir*`/`*os*`/SMA/site/` subdirectory of the installation. This command should not normally be changed. It may be either a string or a tuple of strings. If the command is a tuple of strings, each string must represent a single command line argument. The values of the placemarker are determined by the Abaqus execution procedure or by the command line options and cannot be modified by the user. The values of the placemarker replace the placemarker in the **compile_cpp** string. The following placemarker is available:

%I

Search directories for include files.

compile_fortran

Fortran compile command. The command used at SIMULIA is included in the platform-specific environment file (`win86_64.env` or `lnx86_64.env`) in the *install_dir*`/`*os*`/SMA/site/` subdirectory of the installation. This command should not normally be changed. It may be either a string or a tuple of strings. If the command is a tuple of strings, each string must represent a single command line argument.

The compilation of Fortran files using Fortran 90 freeform specifications is not supported by default. The `abaqus_v6.env` file in the `/SMA/site/` subdirectory of the Abaqus installation includes comments that discuss the compile options.

file_format

Format of results file output (`ASCII` or `BINARY`). This parameter is valid only for Abaqus/Standard. The default value is `BINARY`.

link_exe

Command to link a postprocessing program. The command used at SIMULIA is included in the platform-specific environment file (`win86_64.env` or `lnx86_64.env`) in the *install_dir*`/`*os*`/SMA/site/` subdirectory of

the installation. This command should not normally be changed, except in the case where the user is building a postprocessing program without a Fortran compiler on Windows. In this case, see the commented changes suggested in the `win86_64.env` file. This command can be either a string or a tuple of strings. If the command is a tuple of strings, each string must represent a single command line argument. The values of the placemarkers are determined by the Abaqus execution procedure or by the command line options and cannot be modified by the user. User-specified external libraries can be linked with the usual link commands. The following placemarkers are used in **link_exe**:

%J

The job name (in this case the name of the executable to be created).

%F

The name of the object file created from the user's source file.

%M

The name of the internally created main object file.

%B

The name of the shared library of utility functions.

%O

The list of utility shared libraries for the output database application public interface.

%L

The list of directories containing shared libraries (HP only).

link_sl

Command to link a shared library. The command used at SIMULIA is included in the platform-specific environment file (`win86_64.env` or `lnx86_64.env`) in the *install_dir*`/`*os*`/SMA/site/` subdirectory of the installation. This command, once its placemarkers have been exchanged, must be a valid command on the computer system where the shared library is linked. This command can be either a string or a tuple of strings. If the command is a tuple of strings, each string must represent a single command line argument. The values of the placemarkers are determined by the Abaqus execution procedure or by the command line options and cannot be modified by the user. The values of the placemarkers replace the placemarkers in the **link_sl** string. User-specified external libraries can be linked with the usual link commands. The following placemarkers are available:

%F

The name of the compiled object file.

%U

The name of the shared library.

%A

The name of the archive of compiled object files to be linked into the shared library.

%B

The name of the shared library of utility functions.

%E

The name of the file containing the names of the symbols to be exported from the shared library.

%L

The list of directories containing shared libraries (HP only).

nodb_cache_limit

Maximum size of the cache in the temporary file directory. Abaqus/CAE uses this cache for local data storage when you use a network ODB connector to read from a remote output database. Set the **nodb_cache_limit** parameter to the number of megabytes to which the cache size will be limited. The minimum value of **nodb_cache_limit** is 500, indicating that the cache size is limited to 500 MB. If you set the maximum cache size to be greater than the available free space, Abaqus/CAE reduces it to a value that is equal to the available free space.

plugin_central_dir

Full pathname or list of pathnames of directories containing Abaqus/CAE plug-ins. In most cases this is one or more directories at a central location that is accessible to all users at your site. For more information, see *The Plug-in toolset*.

usub_lib_dir

Full path name of the directory containing optional user-defined libraries of Abaqus/Standard and/or Abaqus/Explicit user subroutines. Valid user subroutine libraries are platform specific, but the file base names are the same for all platforms. The base names are `standardU`, `explicitU`, and `explicitU-D`. Use this variable to avoid the cost of recompiling and/or relinking frequently used user subroutines. The **abaqus make** utility is used to create the shared libraries for use with this variable (see *Making User-Defined Executables and Subroutines*). User libraries created by the **user** option of the Abaqus/Standard and Abaqus/Explicit execution procedure will supersede any user libraries in this directory.

verbose

If this parameter is in the environment file, the execution procedure will print more information on job submission. Possible values are `ON`, `OFF`, `1`, `2`, and `3`. Set the value to `ON` or `1` to print the commands used to run application executables and some performance data. Set the value to `2` to print licensing transaction information. Set the value to `3` to print operating system environment settings. The output associated with the **verbose** parameter is written to standard output. The default value is `OFF`.

The following system customization parameters are used during documentation installation and to specify a web browser to display the SIMULIA HTML documentation and context-sensitive help in Abaqus/CAE.

browser_path

Full path to the web browser executable on Linux platforms. The value of this parameter can be either a string or a list of strings. If a list of strings is specified, the first string must be the full path to the web browser executable and subsequent strings are arguments to customize the browser behavior. If any argument strings are included, at least one of them must contain `%s` for which the full uniform resource locator (URL) to the SIMULIA HTML documentation will be substituted.

Abaqus automatically configures supported browsers to correctly display the HTML documentation. The configurations are different for each browser. This parameter can be used in conjunction with **browser_type** to clarify the browser being used. If **browser_path** is set equal to a string and **browser_type** is not set, the system checks the specified browser path for Firefox. If Abaqus does not detect Firefox, Abaqus assumes that an unsupported browser will be used and does not perform an automatic configuration. This parameter is ignored on Windows platforms.

browser_type

Web browser on Linux platforms. To correctly display the HTML documentation, Abaqus automatically configures the browser according to the specified browser type. The possible settings are `FIREFOX` and `CUSTOM_BROWSER`. If you set **browser_type**=`CUSTOM_BROWSER` to use a web browser other than Firefox, no support or automatic configuration is provided. For more information, see *Using a web browser not supported by Abaqus*.

This parameter can be used in conjunction with **browser_path** to directly specify an executable command for the browser. If **browser_type** is set to Firefox and **browser_path** is not set, the system searches the system path for Firefox. If the specified browser is not found, an error is displayed. This parameter is ignored on Windows platforms.

doc_root

Full uniform resource locator (URL) or path to the SIMULIA HTML documentation.

URL example:

```
doc_root="http://<servername>:4040/English"
```

This variable affects only the connection from Abaqus to documentation, not other Established Products; for example, Tosca and fe-safe. To configure the connection for all Established Products, see

Installation, Licensing & Configuration > SIMULIA Installation > Established Products Installation and Configuration > Manually Configuring Documentation.

doc_lang

Language of the documentation, if more than one is available. The default is "English".

The following parameters disallow the execution of the corresponding modules prior to testing for license activation through the startup file. They can be used to provide “friendly” messages if an attempt is made to execute an analysis module for which your site does not have a license. Use these parameters in conjunction with the **admin** parameter to ensure uniformity across your site.

no_aqua

Block execution of Abaqus/Aqua if value is set to `ON`.

no_background

Block background execution of Abaqus analysis jobs if value is set to `ON`.

no_batch

Block batch queue execution of Abaqus analysis jobs if value is set to `ON`.

no_cae

Block execution of Abaqus/CAE if value is set to `ON`.

no_design

Block execution of Abaqus/Design if value is set to `ON`.

no_explicit

Block execution of Abaqus/Explicit if value is set to `ON`.

no_interactive

Block interactive execution of Abaqus analysis jobs if value is set to `ON`.

no_standard

Block execution of Abaqus/Standard if value is set to `ON`.

no_viewer

Block execution of Abaqus/Viewer if value is set to `ON`.

Executable parameters

The Abaqus executables for the licensed modules are placed automatically into the `/code/bin/` subdirectory of the Abaqus installation. Prefix parameters (***_prefix**) can be set to `"time"` on Linux systems to determine the execution time for a particular stage of an Abaqus run.

exe_prefix

Optional executable prefix for all Abaqus analysis executables. The default value is an empty string.

explicit_prefix

Optional executable prefix for Abaqus/Explicit. The value given for this prefix will override the value given for **exe_prefix**.

explicit_dp_prefix

Optional executable prefix for the double precision version of Abaqus/Explicit. The value given for this prefix will override the value given for **exe_prefix**.

package_prefix

Optional executable prefix for the Abaqus/ExplicitPACKAGE program. The value given for this prefix will override the value given for **exe_prefix**.

pre_prefix

Optional executable prefix for the Abaqusanalysis input file processor. The value given for this prefix will override the value given for **exe_prefix**.

standard_prefix

Optional executable prefix for the Abaqus/Standard program. The value given for this prefix will override the value given for **exe_prefix**.

License management parameters

License management customization parameters control the behavior of the Abaqus license server based on current network conditions and user needs. The Abaqus license server is installed with default parameters that should be suitable for most users. The following parameters are provided for customization:

abaquslm_license_file

This parameter provides the host name of the computer running the Abaqus FLEXnet license server and is set during the product installation. This parameter does not apply to DSLS license servers. If a single server is used, this parameter should be set to *port@license_server_host*, where *port* is the port number and *license_server_host* is the name of the computer running the server:

```
abaquslm_license_file="27000@rose"
```

If the port number is between 27000 and 27009, you do not need to include it.

If two standalone servers are used on Windows platforms, separate the servers in the parameter value with a semicolon:

```
abaquslm_license_file="27000@zinnia;27000@marigold"
```

If two standalone servers are used on Linux platforms, separate the servers in the parameter value with a colon:

```
abaquslm_license_file="27000@zinnia:27000@marigold"
```

If three redundant servers are used, separate the servers in the parameter value with a comma:

```
abaquslm_license_file="27000@maple,27000@pine,27000@oak"
```

academic

This parameter indicates whether an academic Abaqus client should use research or teaching license tokens. Setting this parameter to `TEACHING` will force the Abaqus client to use only teaching license tokens. Setting this parameter to `RESEARCH` or removing the parameter will force Abaqus to use only research license tokens. This parameter is set automatically during the product installation: if the license server contains an Abaqus teaching license file, the installation sets the parameter to `TEACHING`; otherwise, the parameter is set to `RESEARCH`. In some cases, it may be necessary to explicitly set this parameter to `RESEARCH` instead of relying on the default unset value.

cae_timeout

The number of minutes that an Abaqus/CAE or Abaqus/Viewer session will remain idle due to no user activity before returning its token to the license server. The default value is 60 minutes.

computer_location

A string that indicates the location of the local client computer. This parameter allows you to subtotal license usage reports by location. The license usage reporting utility compiles and organizes data according to the **computer_location** name. The default value is an empty string. If you do not change this default, the license usage report will not distinguish between different locations in the report.

dsls_license_config

Path to the Dassault Systèmes license server (DSLS) configuration file (`DSLicSrv.txt`). This file determines which Dassault Systèmes license servers to use with Abaqus. For example,

Linux platforms

```
/opt/simulia/license/DSLicSrv.txt
```

Windows platforms

(must use double backslashes)

```
C:\\SIMULIA\\License\\DSLicSrv.txt
```

For more information about the `DSLicSrv.txt` file, see "Configuring Clients" in the *Dassault Systèmes License Server Installation and Configuration Guide* (`DSLS.pdf`).

license_model

This parameter controls which licensing model Abaqus uses. Possible values are `AUTOMATIC` (default), `LEGACY`, and `SIMUNIT`. If this parameter is set to `SIMUNIT`, Abaqus uses SimUnit tokens or SimUnit credits. If this parameter is set to `LEGACY`, Abaqus uses Simulation tokens (tokens available prior to the introduction of SimUnit tokens and SimUnit credits). When set to `AUTOMATIC`, the licensing model is determined automatically by the licenses present on the license server. If SimUnit tokens or SimUnit credits are available on the license server, the SimUnit license model is used. If not, the legacy model is used. This parameter is applicable only when using a DSLS server (see the **license_server_type** parameter).

license_server_type

The type of license server software used by Abaqus clients. Possible values are `FLEXNET` and `DSLS` (default).

license_type

This parameter controls the type of license that Abaqus uses. Possible values are `CREDIT` and `TOKEN` (default). This parameter is applicable only when using the SimUnit license model (see the **license_model** parameter).

lmhanglimit

The number of minutes that an Abaqus client will wait in the license queue to obtain licenses if no licenses are currently available. The default value of 0 forces the job to remain in the license queue indefinitely unless it is killed by the user.

lminteractivequeuing

This parameter indicates whether an interactive Abaqus/CAE or Abaqus/Viewer session should queue for a license if one is not available. To allow Abaqus/CAE or Abaqus/Viewer sessions running interactively to queue for a license, set this parameter equal to `ON`. The default value is `OFF`. The **lmlicensequeuing** parameter is used for queuing sessions running without the graphical user interface.

lmlicensequeuing

This parameter indicates whether an Abaqus analysis job or an Abaqus/CAE or Abaqus/Viewer session using the **noGUI** option should queue for a license if one is not available. The default value is `ON`. If this parameter is set to `OFF`, jobs and Abaqus/CAE or Abaqus/Viewer sessions will terminate immediately if a license is not available. The **lminteractivequeuing** parameter is used for queuing sessions running interactively.

lmlog

This parameter indicates whether license usage data should be written to the FLEXnet debug log file. To use the Abaqus license usage reporting utility, this parameter must be set to `ON`, which is the default value. To suppress license usage data in the debug log file, set this parameter equal to `OFF`.

lmproject

This optional parameter can be used to record information about internal project names or numbers for your company. The **lmproject** parameter can be set to any string value; for example, `lmproject="turbomachinery-project-23"`. This parameter can be set in the environment file in each users' home directory and can be edited whenever necessary to change to a different project name.

The information about Abaqus license checkouts and associated project names is recorded on the license server and can be retrieved by obtaining historical reports using the accessor `project`.

lmqueuesleep

The number of seconds that an Abaqus client will wait before checking the license queue to see if enough tokens are free. The default value is 30 seconds, which is the minimum allowed. Increasing this value will decrease network traffic when license queuing occurs.

lmsvrdownlimit

The number of minutes that an Abaqus client will attempt to connect to the license server if the license server is currently unavailable. The default value of 0 forces the job to attempt to connect indefinitely unless it is killed by the user.

Object-oriented batch queue parameter

The object-oriented interface for defining an analysis batch queue is easily customized and extended. An analysis batch queue is available from the Abaqus execution procedure command line once it has been instantiated and inserted into the dictionary of analysis queues. See *Defining analysis batch queues*, for more information.

queues

Dictionary of queue names and objects. Queue names and their corresponding object instances are inserted into the dictionary as key/value pairs. The special queue name **default** can be used to designate a default queue. The default queue is used when the **run_mode** is set to `BATCH` and the **queue** option is not specified on the command line. It is also used when the **queue** option is specified on the command line but the named queue is blank or does not exist in the **queues** dictionary.

String-based batch queue parameters

The string-based interface for defining an analysis batch queue allows access to a fixed number of options. The parameters below can be used to construct command strings to control the execution of an Abaqus analysis. The only requirement is the string must be a valid command on the computer system where it is executed. See *Defining a string-based analysis batch queue*, for an example.

after_prefix

Optional prefix to be output as part of the submit command when the **after** option is specified on the command line. The default value is an empty string.

queue_cmd

Default command to be used to submit a batch job when the **run_mode** is set to `BATCH`. This parameter, once its placemarkers have been exchanged, must be a valid command on the computer system where it is executed. The values of the placemarkers are defined by input specified on the command line of the Abaqus execution procedure or by environment file parameters. The values of the placemarkers replace the placemarkers in the **queue_cmd** string. The following placemarkers are available:

%%

A percent (%) character.

%A

The **after_prefix** string is substituted for **%A** when the **after** option is specified on the command line.

%L

The log file name. This token will be replaced by *job-name*`.log`, where *job-name* is the value defined by the **job** command line option.

%P

The **queue_prefix** string (see description below) is substituted for **%P** when the **queue** option is specified on the command line.

%Q

The queue name from the **queue** command line option.

%S

The command script file name. This token will be replaced by *job-name*`.com`, where *job-name* is the value defined by the **job** command line option.

%T

The time from the **after** command line option.

%J

The job name from the **job** command line option.

%E

The token is substituted by the full path to the Python executable.

%O

The full path to the output directory.

queue_name

List of names for batch commands, typically used for submitting jobs to queues other than the default (defined by **queue_cmd**). Then, elsewhere in the environment file, each of these command aliases must appear on the left side of the equal sign, with the desired command string on the right. This command string has the same format as the **queue_cmd** parameter. It can use the replaceable placemarkers in its construction as long as the final result is a valid system command.

queue_prefix

Optional prefix to be output as part of the submit command when a queue name is specified. The default value is an empty string.

Job variables

The following variables can be used in an **onJobStartup** or **onJobCompletion** function:

id

The job identifier specified as the value of the **job** option on the command line.

savedir

The path name to the directory from which the job was submitted.

scrdir

The path name to the scratch directory.

analysisType

The type of analysis to be executed. Possible values are `EXPLICIT` and `STANDARD`.

In addition, for an MPI-based parallel job the following variables are available in an **onJobStartup** or **onJobCompletion** function:

host_list

List of host machine names that were used for the analysis, including the number of processors used on each machine. The format is identical to the **mp_host_list** environment variable (see *Parallelization parameters*).

local_host

List of identifiers used to determine the host machine name from which the job was submitted (e.g., host name, IP address, aliases, etc.).

rsh_command

Command used to open a remote shell on the machines that were used during analysis. The format is identical to the **mp_rsh_command** environment variable (see *Parallelization parameters*).

file_system

Tuple showing the type of file system used for the MPI-based parallel job. The first item in the tuple refers to the directory where the job was submitted, while the second refers to the job's scratch directory. For MPI-based parallel Abaqus/Explicit analyses that do not use a user subroutine, the scratch directory will remain as `DETECT` unless it was set by the user.

cpus

Number of total processors used for the analysis summed across all host machines.

The following variables are available outside of the **onJobStartup** and **onJobCompletion** functions:

abaqus_version

A string that contains the Abaqus release.

analysisType

The type of analysis to be executed. Possible values are `EXPLICIT` and `STANDARD`.

applicationName

The name of the Abaqus execution procedure to be executed; e.g., `analysis`, `cae`, or `viewer`.

Defining analysis batch queues

Analysis batch queues are used to configure the way that Abaqus analysis jobs are run.
They are particularly useful for integrating Abaqus with third-party batch queueing systems.

Two pieces of information are needed to run an Abaqus job: the syntax of the command used to execute the job and the job-specific information. The command used to execute the job is obtained from a queue definition in the Abaqus environment file. The job-specific information is obtained from the command line options and the analysis parameters defined in the Abaqus environment file. The command line options are described in the *Abaqus Execution Guide*.

The command syntax and the job-specific information are used to construct a command to run an Abaqus job in an analysis queue. An object-oriented interface and a string-based interface are available for defining the analysis batch queues. The object-oriented interface is preferred because it is easily customized and extended. This section discusses the object-oriented interface; see *String-based batch queue parameters* for information on the string-based interface.

A set of predefined (built-in) queue classes is available within an Abaqus environment file. These classes can be instantiated to create user-defined queue objects. See *Object-oriented batch queue parameter* for information on the **queues** environment file parameter. In addition to the predefined queue classes, users can create their own queue classes to customize the way analysis jobs are executed. The new queue classes can be derived from the predefined queue classes to minimize coding by the user. The `driverQueues` module must be imported to instantiate a predefined queue or to derive a custom queue class. The predefined classes are described in *Queue class* and *Predefined derived classes*. In addition, examples that illustrate how to extend predefined classes and create custom queue classes are provided in *Using predefined derived classes* and *Deriving and using custom queue classes*, respectively.

In this section:

Queue class

The Queue class is an abstract base class. All other analysis batch queue classes are derived from it. The class has no explicit constructor or members. The following methods of the Queue class are common to all derived classes:

__repr__(...)

This method returns the class name as a string. The string is printed as a description for the queue when **abaqusinformation=**`environment` is executed. Derived classes should override this method to provide a useful description of the queue objects instantiated from them.

createScript(...)

This method creates a Python script named *job-name*`.com` in the current working directory. This script is used to run the analysis. This method is called by the analysis execution procedure prior to the `submit` method. If the *job-name*`.com` file cannot be written in the current working directory, a FileCreationError exception is raised. The following argument is required:

options

A dictionary containing the analysis options.

getDriverName(...)

This method returns the name of the command used to invoke the Abaqus execution procedure.

getPython(...)

This method returns the absolute path to the AbaqusPython interpreter as a string.

getNumRequiredTokens(...)

This method returns the number of license tokens required for an analysis as an integer. The following argument is required:

options

A dictionary containing the analysis options.

spawn(...)

This method executes a command in a new process, waits for it to complete, and returns an integer representing the exit status of the command. If the command cannot be executed, a SpawnError exception is raised. The following arguments are required:

cmd

A valid command string to be executed in the new process. If analyses are submitted to this queue from Abaqus/CAE, the command must return the status immediately; otherwise, the ability to monitor the progress of the analysis jobs in Abaqus/CAE may fail. Examples of commands that return the status immediately are `qsub`, `bsub`, `at`, `batch`, etc.

env

A dictionary of environment variables available to the process.

The following argument is optional:

verbose

A Boolean specifying whether the command string is printed to `stdout`. The default value is OFF.

submit(...)

This abstract method must be implemented by a derived class. This method is called by the analysis execution procedure to submit the analysis to a queue. The `submit` method must return an integer; a value of 0 indicates success, and a nonzero value indicates failure. When this method is called, the analysis execution procedure supplies the following required arguments:

options

A dictionary containing the analysis options.

env

A dictionary of environment variables available to the process.

Most derived classes call the `spawn` method from this method and return its exit status.

Predefined derived classes

The following analysis batch queue classes are derived from the Queue base class:

AtQueue class

The AtQueue class executes an analysis using the Linux `at` command. This class overrides the following base class methods:

__repr__(...)

This method returns a string describing the class.

submit(...)

This method executes the Linux `at` command to run the *job-name*`.com` analysis script at the time specified on the command line **after** option. If the **after** option is not specified on the command line, a QueueError exception is raised.

BatchQueue class

The BatchQueue class executes an analysis using the Linux `batch` command. This class overrides the following base class methods:

__repr__(...)

This method returns a string describing the class.

submit(...)

This method executes the *job-name*`.com` analysis script under the Linux `batch` command.

HoldQueue class

The HoldQueue class creates a *job-name*`.com` file and exits. This class overrides the following base class methods:

__repr__(...)

This method is reimplemented to provide a useful description.

submit(...)

This method prints a message stating that the *job-name*`.com` script was not submitted and returns a value of 0.

LSFQueue class

The `LSFQueue` class submits an analysis to the LSF queue named when the object was instantiated. If a name was not specified, the analysis is submitted to the default LSF queue. The following constructor argument is optional:

name

Name of a valid LSF queue.

This class overrides the following base class method:

submit(...)

This method invokes the LSF`bsub` command to submit the *job-name*`.com` analysis script to an LSF batch cluster and returns the exit status of the `bsub` command.

NQSQueue class

The `NQSQueue` class submits an analysis to the NQS queue named when the object was instantiated. If a name was not specified, the analysis is submitted to the default NQS queue. The following constructor argument is optional:

name

Name of a valid NQS queue.

This class overrides the following base class method:

submit(...)

This method invokes the NQS`qsub` command to submit the *job-name*`.com` analysis script to an NQS system and returns the exit status of the `qsub` command.

PBSQueue class

Queues instantiated from the `PBSQueue` class will create a *job-name*`.pbs` script and run the command `qsub job-name.pbs`. The *job-name*`.inp` and *job-name*`.com` files will be copied to the execution host, where the *job-name*`.com` script will be executed. After job completion, all output files will be copied back to the submission host. The following constructor argument is optional:

name

Name of a valid PBS queue.

This class overrides the following base class method:

submit(...)

This method invokes the PBS`qsub` command to submit the *job-name*`.pbs` script to a PBS system and returns the exit status of the `qsub` command.

Examples

The examples included in this section illustrate various uses of the Abaqus environment file.

In this section:

Example environment file

An example Windows environment file is shown below. This file will work on Linux systems as well if you change the `scratch` directory setting appropriately. A sample environment file, `abaqusinc.env`, is included in the installation subdirectory *install_dir*`/`*os*`/SMA/site/` to show the options used at SIMULIA.

```
scratch = "c:/temp"
if applicationName in ('analysis','datacheck','continue'):
    memory = "4 gb"

def onCaeStartup():
    #Graphics preferences
    session.graphicsOptions.setValues(dragMode=AS_IS, displayLists=ON)
    # Print preferences
    session.printOptions.setValues(vpDecorations=OFF,
        vpBackground=OFF, rendition=COLOR,
        printCommand='lpr -S marley -P hp3')
    session.psOptions.setValues(date=OFF)
    # Job preferences
    def setJobPreferences(module, userData):
         session.Queue(name='long', hostName='server',
            queueName='large', directory='/tmp')
    addImportCallback('job', setJobPreferences)
    # Visualization preferences
    def setVisPreferences(module, userData):
        session.defaultOdbDisplay.contourOptions.setValues(
            renderStyle=SHADED, visibleEdges=EXTERIOR,
            contourStyle=CONTINUOUS)
    addImportCallback('visualization', setVisPreferences)
```

The default compile and link environment variables for your computer have also been inserted in your site environment file.

Notifying users when a job is completed

The following is an example of how environment file commands can be used to notify Linux system users when their job is finished. The notification method used depends on how the job was run and if the user is logged in. If the job was run interactively, the user will not be notified that the job has finished. If the user is still logged in when the job completes, a message will be output to the screen. If the user has logged out by the time the job completes, a message will be mailed to the user. The syntax of the `mail` command varies from system to system. Please consult your system documentation to determine the appropriate commands.

```
    def onJobCompletion():
        import os, re

        userName = os.environ['USER']
        msg = 'Job %s has completed' % id

        # Run "who" command, pipe the output, and read into a list
        whopipe = os.popen('who', 'r')
        output = whopipe.readlines()
        whopipe.close()

        # Find out if the user is logged in
        loggedIn = 'no'
        terminal = [ ]

        for line in output:
          columns = re.split('[ ]+', line) # Split into blank separated columns

            name = columns[0]     # User name is in the first column
            if name == userName:
                terminal.append(columns[1]) # Terminal at which user is logged
in
                loggedIn = 'yes'

        # Use "write" command if the user is logged in, use mail otherwise
        if loggedIn == 'no':
           logFile = savedir + id + ".log"
           if os.path.exists(logFile):
              os.system('cat %s | Mail -s "%s" %s' % (logFile, msg, userName))
           else:
              os.system('Mail -s "%s" %s' % (msg, userName))
        else:
           for termNum in terminal:
              os.system('echo "%s" | write %s %s' % (msg, userName, termNum))
```

Customizing Abaqus/CAE startup

The following example for the **onCaeStartup** parameter will establish viewport preferences and print options (including a print command), set up a remote execution queue for running Abaqus jobs, and set preferences for contour plots in the Visualization module:

```
    def onCaeStartup():

        # Print preferences
        session.printOptions.setValues(vpDecorations=OFF,
            vpBackground=OFF, rendition=COLOR,
            printCommand='lpr -S server -P printer')
        session.psOptions.setValues(date=OFF, logo=OFF)

        def initQueues(*args):
            session.Queue(name='long', hostName='server',
               queueName='large', directory='/tmp')
        addImportCallback('job', initQueues)

        # Visualization preferences
        def setVisPreferences(module, userData):
            import visualization
            session.defaultOdbDisplay.contourOptions.setValues(
               renderStyle=SHADED, visibleEdges=EXTERIOR,
               contourStyle=CONTINUOUS)
        addImportCallback('visualization', setVisPreferences)
```

Indented text must be valid Python commands. For more queue examples, see *The Job module*.

Using a web browser not supported by Abaqus

Abaqus provides support only for the Firefox web browser on Linux platforms; however, it does provide the ability to use a web browser not supported by Abaqus for viewing HTML documentation. Examples are shown for setting the **browser_type** and **browser_path** system customization parameters in this situation.

The first example illustrates the parameter settings to specify a web browser not supported by Abaqus:

```
    browser_type = CUSTOM_BROWSER
    browser_path = ['full_path_to_browser', 'argument1', 'argument2', etc.]
```

where the first string contains the full path to the web browser and subsequent strings are arguments to customize the browser behavior. Refer to the specific web browser documentation for valid arguments.

The second example illustrates the parameter settings to specify the Opera browser (not supported by Abaqus) as the web browser:

```
    browser_type = CUSTOM_BROWSER
    browser_path = ['/usr/local/bin/opera', '-newwindow' '%s']
```

where `/usr/local/bin/opera` is the full path to the browser and the argument `-newwindow` opens a new browser window using the full uniform resource locator (URL) to the Abaqus HTML documentation. Use the argument `-newpage` to open a new page.

The third example illustrates the parameter settings to specify the Konqueror browser (not supported by Abaqus) as the web browser:

```
    browser_type = CUSTOM_BROWSER
    browser_path = '/opt/kde3/bin/konqueror'
```

where `/opt/kde3/bin/konqueror` is the full path to the browser. By default, a new browser window opens using the URL to the Abaqus HTML documentation.

Using predefined derived classes

The following example illustrates the instantiation of some predefined derived classes and their insertion into the **queues** dictionary:

```
    run_mode = BATCH
    from driverQueues import *
    queues['atq']    = AtQueue()
    queues['batchq'] = BatchQueue()
    queues['hold']   = HoldQueue()
```

To submit an analysis using one of the queues, specify the queue name as the value for the analysis execution procedure **queue** parameter.

Deriving and using custom queue classes

To derive a custom queue class, the `driverQueues` module must be imported and the class must inherit directly or indirectly from the Queue class. Derived queues must provide an implementation for the `submit` method. Derived class methods can raise exceptions as needed. The predefined QueueError exception is provided as a general-purpose exception.

The following examples illustrate the derivation and use of custom queue classes:

```
    run_mode = BATCH
    from driverQueues import *
    #~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    class NiceQueue(Queue):
        #~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
        def __repr__(self):
            return 'Executes analysis using Linux nice command.'
        #~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
        def submit(self, options, env):
            job = options['job']
            after = options.get('after', '')
            verbose = options.get('verbose', 0)
            if options.get('after', ''):
                # a descriptive string must be supplied as data when
                # raising a QueueError exception
                #~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
                raise QueueError, \
                      '"after" is not a valid argument for this queue.'
            # run nice under bourne shell to eliminate platform dependencies
            cmd = "/bin/sh -c 'nice %s python ./%s.com 1>./%s.log 2>&1 &'" \
                  % (self.getDriverName(), job, job)
            return self.spawn(cmd, env, verbose)
    #~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    class LSF_ResvQueue(LSFQueue):
        # For integration with LSF. This queue class supports cpu, memory,
        # and license reservations.
        #~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
        def __init__(self, name, memReserve=0, cpusReserve=0):
            LSFQueue.__init__(self, name)
            self.memReserve = memReserve
            self.cpusReserve = cpusReserve
        #~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
        def __repr__(self):
            return 'Submits to LSF %s queue (run "bqueues -l %s" for
description)' \
                   % (self.name, self.name)
        #~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
        def submit(self, options, env):
            job = options['job']
            verbose = options.get('verbose', 0)
            queue = self.name
            cpus = options.get('cpus', '1')
            if self.cpusReserve:
                cpus = self.cpusReserve
            resLst = [ ]
            # license reservation - For the following line to work, LSF
            # must be configured with a static or dynamic resource called
            # "abqtokens".
            #~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
            resLst.append('abqtokens=%d' % self.getNumRequiredTokens(options))
            # memory reservation
```

```
            if self.memReserve:
                from math import ceil
              resLst.append('mem=%d' % int(ceil(self.memReserve/float(cpus))))

            resStr = ''
            if resLst:
                import string
                resStr = '-R rusage[%s]' % string.join(resLst, ':')
            bsub = 'bsub -q %s -J %s -n %s -o %s.log -N %s %s python %s.com' %
\
                   (queue, job, cpus, job, resStr, self.getDriverName(), job)
            return self.spawn(bsub, env, verbose)
   # queue definitions
   queues['default']     = NiceQueue()
   queues['hold']        = HoldQueue()
   queues['benchmark']   = LSF_ResvQueue(name='benchmark', cpusReserve=16)
   queues['dedicated']   = LSF_ResvQueue(name='dedicated')
   queues['a500M']       = LSF_ResvQueue(name='a500M', memReserve=500)
   queues['a1500M']      = LSF_ResvQueue(name='a1500M', memReserve=1500)
   queues['a3000M']      = LSF_ResvQueue(name='a3000M', memReserve=3000)
   queues['a6000M']      = LSF_ResvQueue(name='a6000M', memReserve=6000)
```

Defining a string-based analysis batch queue

The following example illustrates the use of the environment file parameters for string-based analysis batch queue definition:

```
    try:
        queue_name=list(queue_name)
    except:
        queue_name = [ ]
    queue_name=queue_name + ["aba_short", "aba_long", "hold "]
    after_prefix="-a"
    queue_prefix="-q"
    queue_cmd="qsub -nr -me %P %Q %A %T -x -eo -o %L %S"
    aba_short="qsub -nr -me -q short %A %T -x -eo -o %L %S"
    aba_long="qsub -nr -me -q long %A %T -x -eo -o %L %S"
    hold="printf 'Job %S not submitted\n' "
```

The `qsub` command used in this example is available only on certain computer systems. Other commands, such as `at` and `batch`, can be used to configure a queuing system on most Linux platforms. Please refer to your system documentation or contact your hardware vendor for information about queuing systems for your platform.

If the queue specified by the **queue** command line option matches one of the queue aliases in the **queue_name** parameter, that **queue** command is used in place of the default command **queue_cmd**. The following are legal command line options for the above example:

```
abaqus job=qt queue=normal
abaqus job=qt queue=aba_short after=10:00
abaqus job=qt queue=hold
```

The first of these three command line options does not match a defined queue, so the **queue_cmd** string is used to submit the job to the `normal` queue. This queue must have been set up by the systems manager prior to submission of the job. The actual command used to send the job to the `normal` queue for execution on Linux platforms is

```
qsub -nr -me -q normal -x -eo -o qt.log qt.com
```

The value of **%A** is not output if **after=***time* is not specified on the command line.

The second option uses the string defined by **aba_short**, which submits the job to the system predefined `short` queue. The command executed by the Linux platform is

```
qsub -nr -me -q short -a 10:00 -x -eo -o qt.log qt.com
```

The last command line option creates the file `qt.com`, which contains the Abaqus job commands, and saves it in the current directory. The message `Job qt.com not submitted` is then written to the terminal. The job is not submitted to any queue.

Locking out modifications to environment file parameters

In the example below, all Abaqus jobs will run in batch mode by default, and execution of Abaqus/Aqua jobs is not allowed. The inclusion of the **admin** parameter prevents modification of these settings in lower-level environment files. If this parameter is part of the environment file in the installation directory, the values of **run_mode** and **no_aqua** will override any corresponding values in a user's local directory or command line. Therefore, this example constrains all jobs submitted at your site to run in batch mode.

```
run_mode = BATCH
no_aqua = ON
admin = ['run_mode','no_aqua']
```

The **no_aqua** parameter would typically be used to provide a “friendly” message to users if Abaqus/Aqua is not licensed at your site.

Customizing Abaqus/CAE and Abaqus/Viewer

This chapter describes user interface customization, printer configuration, and graphics card tuning for Abaqus/CAE and Abaqus/Viewer.

Platform dependencies sometimes exist for Abaqus/CAE and Abaqus/Viewer. These dependencies can change between releases; therefore, they are listed on the **Support** page at *www.3ds.com/simulia*, where the latest information is published.

In this section:

Customizing the user interface

To customize the Abaqus/CAE and Abaqus/Viewer user interface, you can specify general display properties on Windows platforms.
Settings on other platforms, such as Linux, may also affect the appearance of the user interface and some of its functions. You can also record all of your actions in the Abaqus/CAE or Abaqus/Viewer user interface in a file named `abaqus.guiLog`.

In this section:

Hardware acceleration (all platforms)

With some graphics devices Abaqus/CAE and Abaqus/Viewer may fail when hardware acceleration is turned on. It is possible to turn off hardware acceleration if completely necessary, although it is not recommended. Disabling hardware acceleration will severely degrade graphics performance in Abaqus/CAE and Abaqus/Viewer. You can disable hardware acceleration using one of the following methods:

Linux platforms

Start Abaqus/CAE or Abaqus/Viewer using the `-mesa` option:

```
abaqus cae -mesa
abaqus viewer -mesa
```

Windows platforms

There are two ways to disable hardware acceleration on Windows platforms:

- Add the parameter **abaqus_no_hardware_acceleration=**ON to the Abaqus environment file; or
- Create a system environment variable using the following command:

  ```
  set Abaqus_NO_HARDWARE_ACCELERATION=1
  ```

Common customizations on Windows platforms

The following procedures explain how to specify some commonly desired settings on Windows platforms:

The following procedures explain how to specify some commonly desired settings on Windows platforms:

Change the “start-in” location for any Abaqus shortcut

1. Use **Windows Explorer** to go to the directory where the Abaqus shortcuts are located. The shortcuts contained in this directory affect all users on the computer and may require special permission to change.
2. Click mouse button 3 on the shortcut (**AbaqusCAE**, **Abaqus Command**, or **Abaqus Viewer**) for which you wish to change the start-in location, and select **Properties**; then click the **Shortcut** tab.
3. In the text box labeled **Start in:**, set the full path to the directory you wish to use as the default location for saving the files created by that Abaqus product.

Stop Abaqus/CAE and Abaqus/Viewer windows from being erased when a dialog box is moved

Menu items may vary depending upon your Windows operating system. For Windows7 operating systems:

1. Select **Start->Control Panel**.
2. Type `Effects` in the search box and press **Return**.
3. Select **Adjust the appearance and performance of Windows**.
4. In the **Visual Effects** tab of the **Performance Options** dialog box, toggle off **Show window contents while dragging**.
5. Click **OK** to save your settings and to close the **Performance Options** dialog box.

Change the colors and fonts used in Abaqus/CAE and Abaqus/Viewer

You can change the colors and fonts displayed in Abaqus/CAE and Abaqus/Viewer by applying a new color scheme to your session. Color schemes determine the colors and text settings that Windows uses to display each component in an application, such as its menus, dialog boxes, and title bar. For example, when the **Windows Standard** color scheme is selected, Windows displays white text in Tahoma font against a blue background in the title bar; and displays black text in Tahoma font on a white background in application menus.

You can also customize a color scheme by editing the color or text settings for any individual item in the color scheme. This customization enables you to change more specific settings, like increasing the text size in the title bar without changing the text size in other areas of the application.

Changes to the color and font settings affect all applications, not just Abaqus programs. Menu items may vary depending upon your Windows operating system. For Windows7 operating systems:

1. Select **Start->Control Panel**, and type `Display` in the search box.
2. If desired, choose a new font size.
3. Select **Personalization**.
4. Choose a preset **Theme** from the list.
5. If desired, change the color or font settings for individual items in the selected color scheme:
   a. Type `Window` in the search box. Select **Change window colors and metrics** in the **Personalization** area of the search results.

      The **Window Color and Appearance** dialog box appears.

   b. Select the **Item** for which you want to change color and font display.

The dialog box displays the current color and, if applicable, the font settings for the selected item. Windows excludes font settings for items that do not display text, like the active window border.

c. Choose a new item color from the **Color 1** list. For items like the active title bar that allow a gradient between two colors, you can also adjust the second color in the gradient by choosing a new color from the **Color 2** list.

d. Adjust the text settings for the selected item from the options at the bottom of the dialog box. You can choose a new **Font** from the list, click the **Size** arrows to increase or decrease text size, choose a new text **Color** from the list, and toggle the bold or italic formatting for the text in this item.

e. Repeat the previous two steps to change color and text display for other individual items in a color scheme.

f. Click **OK** to close the **Window Color and Appearance** dialog box.

6. Click **OK** to save your settings and to close the **Display Properties** dialog box.

Change the default fonts used in Abaqus/CAE and Abaqus/Viewer

By default, Windows renders text in the Abaqus/CAE and Abaqus/Viewer viewport windows by referring to the fonts available in your system fonts directory. You can override this default behavior and use other fonts for your session by adding the parameter **hks_font_path** to your Abaqus environment file. Set this parameter to multiple, comma-delimited values if you want to set up multiple font directories for your session.

Display Chinese characters in Abaqus/CAE and Abaqus/Viewer

You can enable a localized Chinese version of Abaqus/CAE and Abaqus/Viewer. This localized version displays Chinese characters for text in menus, dialog boxes, the Model and Results Trees, and the message area; text in the viewport is not localized.

Menu items may vary depending upon your Windows operating system. For Windows7 operating systems:

1. Set the regional language settings for Windows to enable Chinese characters:
   a. Select **Start->Control Panel**.
   b. To install the Chinese language, select **Install or uninstall display languages** and follow the prompts.
2. Set the system environment variable `ABAQUS_USE_LOCALIZATION=1`.

   - To set the variable for a single session, enter the following command in a command prompt:

     ```
     set ABAQUS_USE_LOCALIZATION=1
     ```

     You must run Abaqus/CAE or Abaqus/Viewer from the same command prompt.

   - To set the environment variable permanently, save it in the Windows system properties. Environment variables can be saved using the **Advanced** tabbed page of the **System Properties** dialog box.

Linux settings that affect Abaqus/CAE and Abaqus/Viewer

Linux operating systems provide you with many options for customization. Because you can alter parts of the operating environment that are held constant on other platforms, your Linux settings may alter some basic interactions within Abaqus/CAE and Abaqus/Viewer. The exact settings available and the methods you use to change them vary according to the version of Linux that you have installed. Three possible settings and their effects are as follows:

Removal of window title bars

Your access to some dialog and toolbox functions may be limited if you have customized your system so that window title bars are not displayed. Without title bars, you may not be able to move a dialog box. Using the **[Esc]** key is the only means to close a dialog or toolbox that has no title bar or **Cancel** button.

Removal of window borders

Resizing a dialog box requires you to click and drag the border. If your Linux customizations include the removal of application window borders, you may not be able to resize dialog boxes in Abaqus/CAE and Abaqus/Viewer. Instead, use the scroll bars to access data that extend beyond the edges of a dialog box.

Displaying Japanese characters

If your locale setting is Japanese, Abaqus/CAE and Abaqus/Viewer can display Japanese text in the viewport. For example, text annotations and the state and title blocks are displayed in Japanese. To display the Japanese characters correctly, the Japanese TrueType fonts must be installed in the directory `/usr/lib/X11/fonts`.

Recording all user interface actions in a log file

You can record all of the actions you take in the Abaqus/CAE or Abaqus/Viewer user interface in a file named `abaqus.guiLog`. This setting can be toggled on for a single session, or you can select it as the default behavior.

Enabling user interface recording at runtime

Start Abaqus/CAE or Abaqus/Viewer using the `-guiRecord` option:

```
abaqus cae -guiRecord
abaqus viewer -guiRecord
```

Enabling user interface recording using `ABQ_CAE_GUIRECORD`

You can specify user interface recording by setting `ABQ_CAE_GUIRECORD` from a command line or, on Windows platforms, as a system environment variable.

- To set the variable for a single session, enter the following command in a command prompt:

  ```
  set ABQ_CAE_GUIRECORD=1
  ```

  You must run Abaqus/CAE or Abaqus/Viewer from the same command prompt.
- To enable user interface recording as the default behavior on Windows platforms, save the parameter `ABQ_CAE_GUIRECORD=ON` in the Windows system properties. Environment variables can be saved using the **Advanced** tabbed page of the **System Properties** dialog box (select **Start**->**Settings**->**Control Panel**, and double-click **System**).

When user interface recording is the default behavior, you can still disable it from the command line by starting Abaqus/CAE or Abaqus/Viewer using the `-guiNoRecord` option:

```
abaqus cae -guiNoRecord
abaqus viewer -guiNoRecord
```

Configuring printers

Print commands are not normally needed for Windows platforms. Abaqus/CAE and Abaqus/Viewer automatically detect and list any installed Windows printers.
If you encounter problems using the print tool in Abaqus/CAE or Abaqus/Viewer, check the installed Windows printers on your machine and remove any printers that are no longer valid.

This section describes how Abaqus uses print commands. Print commands on Linux platforms should be identical to those used for other applications.

When Abaqus/CAE or Abaqus/Viewer prints to a PostScript printer, it goes through the following steps:

Creating a temporary PostScript file

The temporary PostScript file contains all the PostScript code necessary to describe the page to be printed.

Retrieving the print command specified in the print dialog box

The command specified in the print dialog box can be any command that sends to the printer an unmodified copy of the file whose path is its last argument (some PostScript modifications such as the ones done by PostScript print managers are allowed).

To customize the default print command that is used by Abaqus/CAE or Abaqus/Viewer when it shows the print dialog box, add the following line to the `abaqus_v6.env` file in your home directory or in the site directory of the Abaqus release:

```
def onCaeStartup():
    session.printOptions.setValues(printCommand=
    'print_command_and_arguments_here')
```

Appending the name of the temporary file and invoking the resulting command

The name of the previously created temporary file is appended to the print command and the PostScript file is printed.

Deleting the temporary PostScript file

If the printer that you are using does not support print spooling, the temporary PostScript file may be deleted before the file is printed. To prevent the temporary PostScript files from being deleted, add the following line to the `abaqus_v6.env` file in your home directory or in the site directory of the Abaqus release:

```
def onCaeStartup():
    session.printOptions.setValues(deleteTemporaryFiles=False)
```

Tuning graphics cards

This section contains the information that you need to configure Abaqus/CAE and Abaqus/Viewer for a graphics adapter that is not yet qualified.

In this section:

Why is tuning necessary?

SIMULIA tunes and qualifies a limited set of graphics adapters prior to each release. Tuning parameters for these graphics adapters are included in Abaqus. However, new graphics adapters and new drivers for existing graphics adapters become available between releases. Tuning may enable you to take advantage of these new adapters and drivers without waiting for a new release of Abaqus/CAE or Abaqus/Viewer.

Abaqus/CAE and Abaqus/Viewer use OpenGL for high-speed graphics rendering. While the OpenGL standard has strict conformance tests, some features are implementation dependent and require tuning to function correctly. Tuning a new graphics adapter or driver ensures that Abaqus/CAE and Abaqus/Viewer graphics are rendered correctly and that maximum rendering performance is obtained for each system.

You can find the latest information on qualified graphics adapters on the **Support** page at *www.3ds.com/simulia*. If you read the information on this page and follow the tuning procedures described in this section, you should be able to render Abaqus/CAE and Abaqus/Viewer graphics correctly and with maximum performance. If you continue to experience problems, you should contact your local technical support office for assistance.

How can I tune the parameters?

Abaqus/CAE and Abaqus/Viewer provide the following two methods for tuning graphics parameters:

- Select **View->Graphics Options** from the main menu bar. Abaqus displays the **Graphics Options** dialog box shown in *Figure 1* from which you can select the desired settings. This approach allows you to select from only the most commonly used tuning parameters.

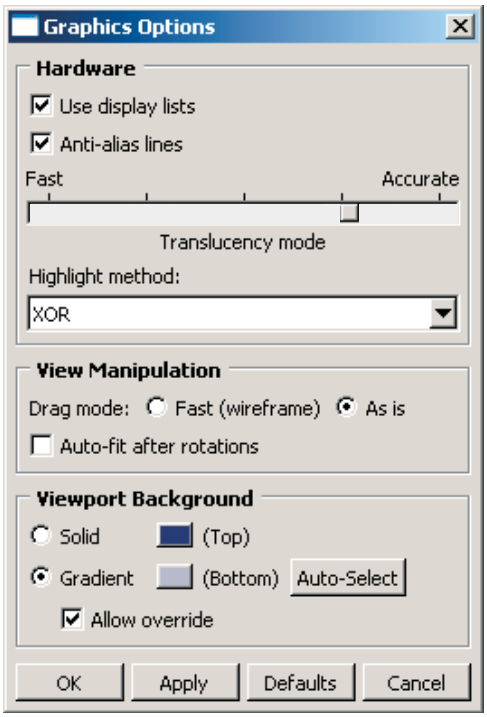


Figure 1: The Graphics Options dialog box.

  The settings in the **Graphics Options** dialog box are described in *Configuring graphics display options*.

- Use an Abaqus Scripting Interface command to select the desired settings. You can enter the command in the command line interface (CLI) and modify the values of the tuning parameters. This approach provides complete control of all the tuning parameters and is described in *Using the Abaqus Scripting Interface to tune the graphics parameters*. This section also describes how you can use an Abaqus Scripting Interface command to obtain information about the graphics card that you are using.

Table 1 shows the tuning parameters that are available from Abaqus/CAE and Abaqus/Viewer. The table also shows the standard value of each parameter and whether you can use the **Graphics Options** dialog box to modify it. Certain parameters can be modified only before starting an Abaqus/CAE or Abaqus/Viewer session (see *Making your graphics configuration permanent*, for details on modifying parameters on startup).

Table 1: Tuning parameters.

Parameter	Standard value	Modify using Graphics Options dialog box	Modify only on startup
displayLists	On	Yes	No
antiAlias	On	Yes	No
translucencyMode [2]	More accurate than fast	Yes	No
highlightMethod [3]	Hardware	Yes	No

Parameter	Standard value	Modify using Graphics Options dialog box	Modify only on startup
highlightMethodHint	Hardware	Yes	No
dragMode	As-is	Yes	No
autoFitAfterRotate	Off	Yes	No
backgroundColor	#333366	Yes	No
backgroundBottomColor	#acacc1	Yes	No
backgroundStyle	Gradient	Yes	No
backgroundOverride	On	Yes	No
doubleBuffering	On	No	No
polygonOffsetConstant	0.0 to 100.0	No	No
polygonOffsetSlope	0.0 to 100.0	No	No
printPolygonOffsetConstant	0.0 to 100.0	No	No
printPolygonOffsetSlope	0.0 to 100.0	No	No
textureMapping	On	No	No
printTextureMapping	On	No	No
vertexArrays	On	No	No
vertexArraysInDisplayLists	On	No	No
backfaceCulling	On	No	No
directRendering	On	No	Yes
accelerateOffScreen	Off	No	Yes
backingStore	On	No	No
hardwareAcceleration [4]	On	No	Yes
hardwareOverlay	None	No	Yes
hardwareOverlayAvailable [5]	None	No	N/A
shadersAvailable [5]	None	No	N/A
viewManipDisplayListThreshold [1]	40	No	No
contourRangeTexturePrecision	5.0×10^{-6}	No	No

[1]The threshold is only used in the Visualization module of Abaqus/CAE (Abaqus/Viewer) when display lists are enabled.

[2]Translucency mode settings for rendering of translucent objects range from 1 (optimized for performance) to 5 (optimized for accuracy). The default value is 4.

[3]The highlight method is indirectly set by setting the *highlightMethodHint* parameter. Abaqus uses this value to determine an appropriate setting for *highlightMethod*.

[4]Hardware acceleration is applicable only to Windows platforms.

[5]You cannot directly set the *hardwareOverlayAvailable* parameter or the *shadersAvailable* parameter. Abaqus automatically sets these parameters by detecting the available hardware on your system.

Using the Abaqus Scripting Interface to tune the graphics parameters

You can enter Abaqus Scripting Interface commands in the command line interface to tune your graphics parameters and to find out information about the graphics adapter installed on your system. This section explains how to use the Abaqus Scripting Interface to modify the graphics options; the Abaqus Scripting Interface is described in detail in the *Abaqus Scripting User's Guide*.

In general, you should use the default values for most of the parameters. However, Abaqus provides the capability to modify parameters to fix the following specific problems:

- The *hardwareAcceleration* parameter controls a number of different graphics tuning parameters and generally should not be modified. Hardware acceleration options are discussed in *Hardware acceleration (all platforms)*.
- The *hardwareOverlay* parameter is controlled by the *hardwareOverlayAvailable* parameter. If your system supports hardware overlay planes, Abaqus/CAE and Abaqus/Viewer will use them by default. If your system supports hardware overlay planes but viewports display a solid color and will not display a model, you may need to manually set *hardwareOverlay*=OFF.
- The *contourRangeTexturePrecision* parameter sets the tolerance used when computing the appropriate scale for transforming result (contour) values to texture values. When set too low, the “out of range” colors may be shown incorrectly for values near the range limits.
- Some graphics adapters do not support the use of textures to generate contour plots properly. If you experience problems displaying contour plots (for example, all contours appear gray or the system aborts), you need to set *textureMapping*=OFF to emulate texture mapping in software. Similarly, if you experience problems printing contour plots, you need to set *printTextureMapping*=OFF.
- Some graphics adapters do not fully support the use of vertex arrays to process information about vertices. Some specific problems indicate that vertex arrays are not fully supported: when you drag the radius of a circle in the Sketcher, the circle is not visible; when you display an *X*–*Y* plot, the axis labels are not visible; and some facets in the shaded display of a mesh are missing. If you experience any of these problems, set *vertexArraysInDisplayLists*=OFF. If this does not resolve the problem, suppress the use of vertex arrays altogether by setting *vertexArrays*=OFF.
- The *backfaceCulling* parameter controls the display of facets that are determined to be facing away from the viewer. If the front sides of elements appear to be missing in the display or if the display is incomplete, set *backfaceCulling*=OFF.
- You can disable direct rendering (set *directRendering*=OFF) for Linux systems that do not behave correctly when accessing the graphics hardware directly.
- You can disable hardware-accelerated off-screen rendering (set *accelerateOffScreen*=OFF) when you want printed images to be rendered without OpenGL hardware acceleration or if you experience problems with the Probe functionality in the Visualization module of Abaqus/CAE (Abaqus/Viewer).
- You can disable the backing store (set *backingStore*=OFF) when you want to conserve memory. When *accelerateOffScreen*=ON, the memory for the backing store is allocated from memory on the graphics card. When OFF, the memory for backing store is allocated from system memory. The backing store is generated by rendering the viewport to an off-screen area. Subsequent viewport refreshes are performed more quickly by copying the off-screen area to the viewport window. Even when *backingStore*=ON, the backing store will not be created if the viewport can be redrawn sufficiently quickly.
- The *translucencyMode* parameter determines whether Abaqus/CAE optimizes the rendering of translucent objects for performance, accuracy, or for a level in between. Lower values provide better performance, while higher values provide greater accuracy.
- The *polygonOffsetConstant* and *polygonOffsetSlope* parameters, which affect onscreen display, require manual tuning for each graphics adapter. On Linux systems the *printPolygonOffsetConstant* and *printPolygonOffsetSlope* parameters can generally be set equal to the same values as the corresponding onscreen display parameters. On

Windows systems the *printPolygonOffsetConstant* and *printPolygonOffsetSlope* parameters do not generally need to be adjusted.

- The *viewManipDisplayListThreshold* parameter can be lowered if there is an unacceptable delay when initiating view manipulation operations in the the Visualization module. Increasing this value may increase the delay for large models but should produce improved graphics performance during the view manipulation. If set high with a large model, the delay can be many seconds and in excessive cases may exceed system graphics memory and result in an empty display (no visible model) for the view manipulation.

You can tune the graphics parameters using the following Abaqus Scripting Interface objects:

- GraphicsOptions: The members of the GraphicsOptions object determine the current graphics settings. These settings can be modified during a session using the `setValues` method. The arguments to the `setValues` method are described in *GraphicsOptions object*.

You can view the current settings of the graphics parameters by entering the following command in the command line interface:

```
print(session.graphicsOptions)
```

The following output is typical:

```
({'accelerateOffScreen': OFF,
  'antiAlias': ON,
  'autoFitAfterRotate': OFF,
  'backfaceCulling': ON,
  'backgroundBottomColor': '#acacc1',
  'backgroundColor': '#333366',
  'backgroundOverride': ON,
  'backgroundStyle': GRADIENT,
  'backingStore': ON,
  'contourRangeTexturePrecision': 5.0e-06
  'directRendering': ON,
  'displayLists': ON,
  'doubleBuffering': ON,
  'dragMode': AS_IS,
  'graphicsDriver': OPEN_GL,
  'hardwareAcceleration': ON,
  'hardwareOverlay': OFF,
  'hardwareOverlayAvailable': False,
  'highlightMethod': SOFTWARE_OVERLAY,
  'highlightMethodHint': (HARDWARE_OVERLAY,
   SOFTWARE_OVERLAY, XOR, BLEND),
  'polygonOffsetConstant': 2.0,
  'polygonOffsetSlope': 0.75,
  'printPolygonOffsetConstant': 1.0,
  'printPolygonOffsetSlope': 0.75,
  'printTextureMapping': ON,
  'shadersAvailable': True,
  'stencil': False,
  'textureMapping': ON,
  'translucencyMode': 3,
  'vertexArrays': ON,
  'vertexArraysInDisplayLists': ON,
  'viewManipDisplayListThreshold': 40})
```

Note:

Some of the parameters listed above have been deprecated. For information on accessing deprecated parameters, see *BackwardCompatibility object*.

The following command uses the `setValues` method to modify some members of the GraphicsOptions object:

```
session.graphicsOptions.setValues(autoFitAfterRotate=ON,
    dragMode=AS_IS)
```

You can enter this command in the command line interface.

- GraphicsInfo: The members of the GraphicsInfo object provide information about the graphics adapter installed on your system. This information may be useful for determining how to tune the graphics adapter. The members are described in *GraphicsInfo object*. The members are for reference only and cannot be modified.

  You can view the graphics information by entering the following command in the command line interface:

  ```
  print session.graphicsInfo
  ```

  The following output is typical on Windows platforms:

  ```
  ({'glRenderer': 'Quadro FX 560/PCI/SSE2',
    'glVendor': 'NVIDIA Corporation',
    'glVersion': (2, 0, '.3'),
    'glxClientVendor': None,
    'glxClientVersion': (None, None, None),
    'glxServerVendor': None,
    'glxServerVersion': (5, 1, None)})
  ```

Tuning the polygonOffsetConstant and polygonOffsetSlope parameters

If display lists are enabled, you will not see the effect of tuning these parameters; therefore, you must toggle off **Use display lists** before attempting to tune your graphics adapter. Alternatively, you can enter the following command in the command line interface:

```
session.graphicsOptions.setValues(displayLists=OFF)
```

Setting drag mode to AS_IS is helpful for fine tuning the parameters. Rotating the view interactively will show you if minor adjustments are necessary.

```
session.graphicsOptions.setValues(dragMode=AS_IS)
```

It is recommended that you tune *polygonOffsetConstant* first, then tune *polygonOffsetSlope*. To tune these parameters, you should first display the part that is generated by the example script in *Creating a part*. To retrieve the script, use the following command in a command prompt window (operating system shell):

```
abaqus fetch job=modelAExample
```

Select **File->Run Script** from the main menu bar, select the example script from the **Run Script** dialog box that appears, and click **OK**. The example script creates a new viewport; however, Abaqus should display only one viewport while you are trying to tune the graphics parameters. Select any old viewports and delete them by selecting **Viewport->Delete Current** from the main menu bar.

To tune the polygonOffsetConstant parameter:

1. From the **Views** toolbar, apply the bottom view  .
2. In the command line interface, enter the following commands:

   ```
   session.graphicsOptions.setValues(polygonOffsetSlope=0.0)
   session.graphicsOptions.setValues(polygonOffsetConstant=0.0)
   ```

3. Display the bottom view again to refresh the display.
4. Examine the model for visible lines. If all lines are not visible, repeat Step 2, increasing the value of the polygon offset constant by a small increment; for example, `0.1`. The normal range for this parameter is

between `0.5` and `1.5`, and two decimal places usually provide sufficient precision. You should attempt to find a value as small as possible that produces a correct display. Values that are too large may cause the lines to appear to float above the part. The following figures illustrate the lines that should be visible in your model.

Figure 1 illustrates the model with an incorrect value for *polygonOffsetConstant*; some lines are missing between the shaded areas.

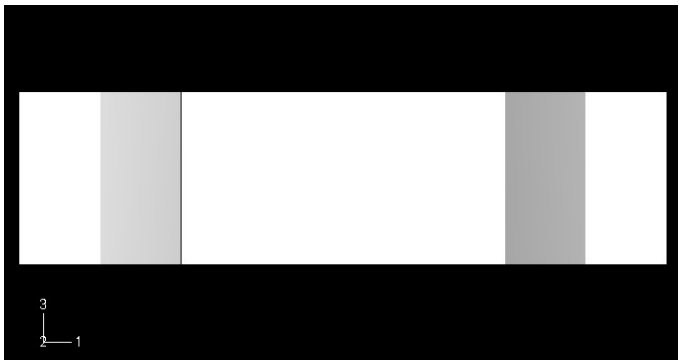


Figure 1: Incorrect value for *polygonOffsetConstant*.

Figure 2 illustrates the model with a correct value for *polygonOffsetConstant*; all the shaded areas are separated by lines.

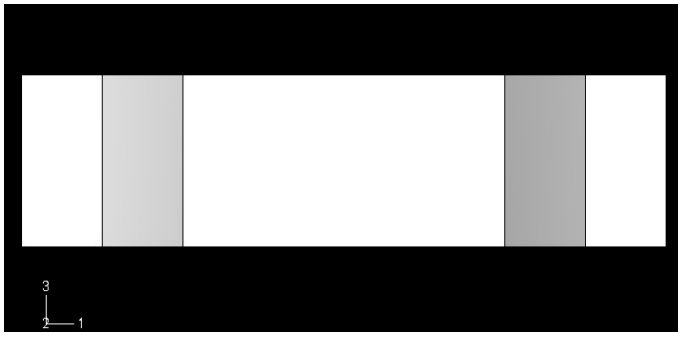


Figure 2: Correct value for *polygonOffsetConstant*.

After you have derived a value for *polygonOffsetConstant*, you can tune *polygonOffsetSlope*.

To tune the polygonOffsetSlope parameter:

1. From the **Views** toolbar, apply the isometric view  . This view shows edges at a 45° angle on at least one axis.
2. In the command line interface, enter the following command:

   ```
   session.graphicsOptions.setValues(polygonOffsetSlope=1.0)
   ```

3. Examine the model for visible lines. If all lines are not visible, repeat Step 2, increasing the value of the polygon offset slope by a small increment; for example, `0.1` or `0.05`. The normal range for this parameter is between `1.0` and `2.0`, and two decimal places usually provide sufficient precision. If the *polygonOffsetConstant* value is too low, it may force the *polygonOffsetSlope* to be high. High values of *polygonOffsetSlope* may cause the edges of hidden polygons to show through where they meet visible polygons. In this case, raise the *polygonOffsetConstant* value to get an acceptable value for *polygonOffsetSlope*.

Figure 3 illustrates the model with an incorrect value for *polygonOffsetSlope*; some line segments are missing between the shaded areas, there is a line missing inside the hole, and some lines appear dashed when they should appear solid.

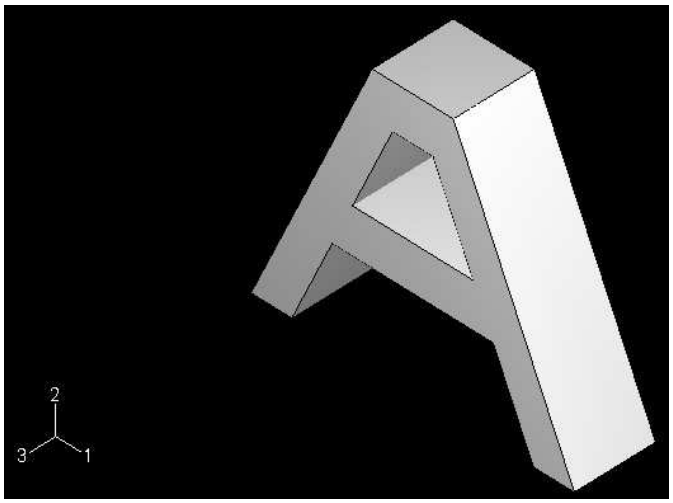


Figure 3: Incorrect value for *polygonOffsetSlope*.

Figure 4 illustrates the model with a correct value for *polygonOffsetSlope*; all the shaded areas are separated by solid lines.

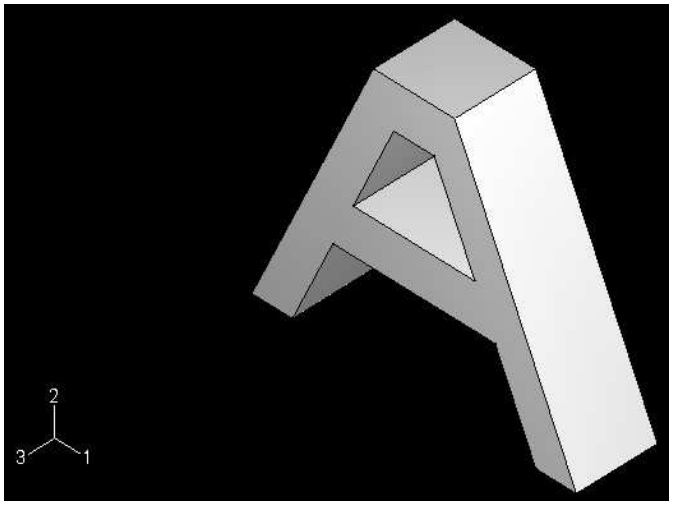


Figure 4: Correct value for *polygonOffsetSlope*.

Test the tuned values of *polygonOffsetConstant* and *polygonOffsetSlope* on several models to make sure the values are satisfactory. When you have finished tuning the graphics parameters, you should return your settings for display lists and drag mode to the original values.

When you are satisfied with the parameter settings, you should modify the environment file as described in *Making your graphics configuration permanent*.

Making your graphics configuration permanent

Once you are satisfied with the values you have specified for the tuning parameters, you can make the changes permanent by including an `onCaeGraphicsStartup` function in your environment file (`custom_v6.env` or `abaqus_v6.env`). To avoid conflicts with other graphics settings, you should add the customized `onCaeGraphicsStartup` function only to the environment file in your home directory (see *Using the Abaqus environment files*, for details on environment file location and execution).

The members of the DefaultGraphicsOptions object determine the default graphics settings that are enabled when you start a session and when you click **Defaults** in the **Graphics Options** dialog box. You can view the default graphics settings by entering the following command in the command line interface:

```
print(session.defaultGraphicsOptions)
```

You use the `setValues` method in the environment file (`abaqus_v6.env`) to modify the members of the DefaultGraphicsOptions object. To set your default graphics options in the environment file, you must use the `session.defaultGraphicsOptions` object instead of the `session.graphicsOptions` object that you modified from the command line interface. The following example environment file configures your Abaqus/CAE and Abaqus/Viewer graphics settings:

```
def onCaeGraphicsStartup():
    session.defaultGraphicsOptions.setValues(
        polygonOffsetConstant=1.0,
        polygonOffsetSlope=1.2)
```

Installation subdirectories

The SIMULIA products are installed into the same directory.

The following subdirectories will be created:

install_dir/*os*/CAEresources/	Configuration files.
install_dir/*os*/code/bin/	SIMULIA product executables, third-party executables, and libraries.
install_dir/*os*/code/command/	SIMULIA product command procedures.
install_dir/*os*/code/include/	Links to header files used for building postprocessing programs with the **abaqus make** utility.
install_dir/*os*/SMA/samples/	Input files and directories associated with the installed products, including installation verification problems, timing test problems, files for the Introduction to Abaqus seminar, files for the Getting Started tutorial guides, files for application briefs, files used with the **abaqus findkeyword** utility, and files from the Example Problems, Benchmarks, and Verification Guides.
install_dir/*os*/SMA/site/	Site-specific files: the Abaqus environment files (abaqus_v6.env and custom_v6.env), a sample environment file (abaqusinc.env), include files (*file_name*.inc), a data file containing information used in the verification procedure (chksum.dat), and the platform computer tables.
install_dir/*os*/tools/SMApy	Python interpreter(s).
install_dir/*os*/code/python2.7/lib/	Abaqus Python modules.

Accessing remote file systems for installation and execution

This appendix describes accessing remote file systems for Abaqus installation and execution.

In this section:

Running Abaqus remotely on Linux

It is possible to run Abaqus on NFS-mounted file systems or to export the display from a remote computer; however, users may experience performance and reliability problems with these alternatives.

In this section:

NFS-mounted file systems

There are several scenarios for running Abaqus on NFS-mounted file systems; the impact of using NFS-mounted file systems varies with the method of NFS use. The most common scenarios for running Abaqus on NFS-mounted file systems are as follows:

- Abaqus is installed on a remote file system. The CPU, save directory (see *Job variables*), and scratch directory are local (on the computer where Abaqus will be run).

  In this scenario Abaqus executables and shared libraries are loaded into local memory as they are needed across the network from the file system where Abaqus is installed. Processing and output occur locally. If there is sufficient local memory to prevent frequent paging of the code, network traffic will be relatively light. When local memory is insufficient to prevent paging, performance will suffer and reliability may be impacted.
- Abaqus is installed on a remote file system. The save directory and/or scratch directory is/are not local to the CPU where Abaqus is executed.

  In this scenario program files and data used during execution are written across the network. Potentially large volumes of data will be transferred across the network, and performance may be impacted adversely. Abaqus performance may be extremely slow, and other users of the network may be affected because the network could become saturated. In addition, Abaqus does not trap file errors arising from NFS, so a failure in accessing NFS-mounted files, even temporary, will cause the Abaqus job to fail. The use of Abaqus in this configuration should be avoided whenever possible. If this configuration is required for an Abaqus/Standard job, the user should move the save directory to the NFS-mounted file system before moving the scratch directory.

Exporting the display

This configuration is relevant only for Abaqus/CAE and Abaqus/Viewer. In this configuration Abaqus is installed on a remote file system and the CPU, save directory, and scratch directory are located on the same remote computer, which is accessed through remote login. All processing occurs on the remote computer, and only output messages or graphics are exported to the local computer and display. This method is known to cause performance problems. Minor incompatibilities between OpenGL and GLX libraries can introduce significant graphics problems and, in some cases, can prohibit Abaqus/CAE or Abaqus/Viewer from running. The use of Abaqus in this configuration is not supported and should be avoided whenever possible.

Using a network ODB connector

Users can create a network ODB connector to access an output database on a remote computer (see *Accessing an output database on a remote computer*). Abaqus/CAE or Abaqus/Viewer can start the server on the remote system and assign port numbers if the following are true:

- The user name on the remote host is the same as the user name on the local system.
- The remote shell command (`rsh`) or the secure shell command (`ssh`) is configured so that it does not prompt the user for a password.

Abaqus checks the security of the connection by passing a key back and forth between the server and the client. For information about how the key is generated, see the Dassault Systèmes Knowledge Base. If a file called `.abaqus_net_passwd` is present in the user's home directory on the remote system, Abaqus uses the password in the file for authentication instead of the key generated by Abaqus. Abaqus checks that the user is the only account with permission to read and write to the password file. In addition, the user must update the file after 30 days, and the password must be at least eight characters long. These files are described in *Network Output Database File Connector*.

If users experience problems establishing communication or if the user names are different, they can start the network ODB server manually from the command line using the **abaqus networkDBConnector** execution procedure on the remote computer. Abaqus uses the password in `.abaqus_net_passwd` to authenticate the connection between the client and the server if the user starts the network ODB server manually. If `.abaqus_net_passwd` does not exist in the user's directory on the remote system, the user cannot start the network ODB server from the command line.

You can disable network odb connectors by removing `dmbwtr` and `dmbwtrd` from the *install_dir*`/`*os*`/code/bin/SMAExternal/dmbwtr/` directory.

Verification procedure

The verification procedure checks the installation of all licensed Abaqus products and reports on the success or failure of verification for each product.
The verification procedure runs automatically after the Abaqus installer has finished, but only a subset of the products are verified. The verification procedure can also be run as an Abaqus command option.

The procedure runs verification problems for each licensed product and compares the results to reference values. The command line options are not affected by license type; that is, the verification procedure attempts to verify all products named in the command. Before the verification procedure is run, licensing requirements are checked for the selected product(s). If a teaching academic license is detected, the verification procedure that is run during installation checks only Abaqus/Standard, Abaqus/Explicit, and Abaqus/CAE.

Product verification is skipped if the product is not licensed. The command line option `-noLic` can be used to bypass these checks. The verification problems for all Abaqus products are extracted automatically from the disk during the installation.

To run the verification procedure from the command line:

Run the procedure by typing the following command:

```
abaqus verify [-adams -all -ams -tosca -cae -catiav4
-cPerf -dcatiav5 -design -docUrl -exp -foundation
-help -install -ioPerf -log -make -moldflow -noGui
-noLic -parallel -param -parasolid -proe
-retainFiles -scripting -std -swi -user_exp -user_std
-verbose -viewer]
```

Common options

`-all`	Verify all licensed products. All other verification options except `log` and `noLic` are ignored.
`-help`	Print summary of `verify` usage.
`-install`	Verify Abaqus parametric studies and Abaqus/CAE.
`-log`	Direct all output to a file named `verify.log` in the current working directory.

Product options

`-ams`	Verify Abaqus/AMS.
`-tosca`	Verify Tosca for Abaqus.
`-cae`	Verify Abaqus/CAE.
`-design`	Verify Abaqus/Design.
`-exp`	Verify Abaqus/Explicit.
`-foundation`	Verify Abaqus/Foundation.
`-param`	Verify parametric studies in Abaqus.
`-std`	Verify Abaqus/Standard.
`-user_exp`	Verify Abaqus/Explicit with user subroutines.
`-user_std`	Verify Abaqus/Standard with user subroutines.

`-viewer`	Verify Abaqus/Viewer.

Translator options

`-adams`	Verify Abaqus Interface for MSC.ADAMS.
`-catiav4`	Verify Geometry Translator for CATIA V4.
`-dcatiav5`	Verify Direct Geometry Import for CATIA V5 (Direct Geometry Import is a component of the CATIA V5 Associative Interface). Verifies geometry import capability in Abaqus/CAE; does not verify installation or functionality of the CATIA V5 Associative Interface plug-in.
`-moldflow`	Verify Abaqus Interface for Moldflow.
`-parasolid`	Verify Geometry Translator for Parasolid.
`-proe`	Verify Geometry Translator for Pro/ENGINEER (the Geometry Translator is a component of the Pro/ENGINEER Associative Interface). Verifies geometry import capability in Abaqus/CAE; does not verify installation or functionality of the Pro/ENGINEER Associative Interface plug-in.
`-swi`	Verify Geometry Translator for SolidWorks (the Geometry Translator is a component of the SolidWorks Associative Interface). Verifies geometry import capability in Abaqus/CAE; does not verify installation or functionality of the SolidWorks Associative Interface plug-in.

Additional options

`-cPerf`	Verify Abaqus/CAE performance.
`-docUrl`	Verify Abaqus HTML documentation URL.
`-ioPerf`	Verify I/O performance.
`-make`	Verify the **abaqus make** utility.
`-noGui`	Verify the `-noGUI` option for Abaqus/CAE and Abaqus/Viewer.
`-noLic`	Requires `-all`, `-install`, or a list of product options. Run the verification procedure for the specified products but bypass all licensing checks. The procedure attempts to verify all selected products regardless of licensing.
`-parallel`	Verify Abaqus analysis jobs using parallelization.
`-scripting`	Verify the Abaqus scripting interface.
`-retainFiles`	Retain all verification files in the `verify` directory (by default, the files are deleted after a successful verification).
`-verbose`	Include additional details for debugging purposes.

If a product selected for verification is not licensed, an Abaqus warning is displayed for the selected product and the verification of other selected products continues.

Reviewing and resolving verification procedure failures

If the verification procedure finishes successfully, all files and the `verify` directory are removed (unless you use the `retainFiles` option). Error diagnostics for all products that fail verification remain in the `verify` directory. It is very important that you review these error messages.

The `verify` directory and results can be found in the *install_dir*`/InstallDate/logs/.../tmp/` directory after the media component Abaqus/CAE has been installed.

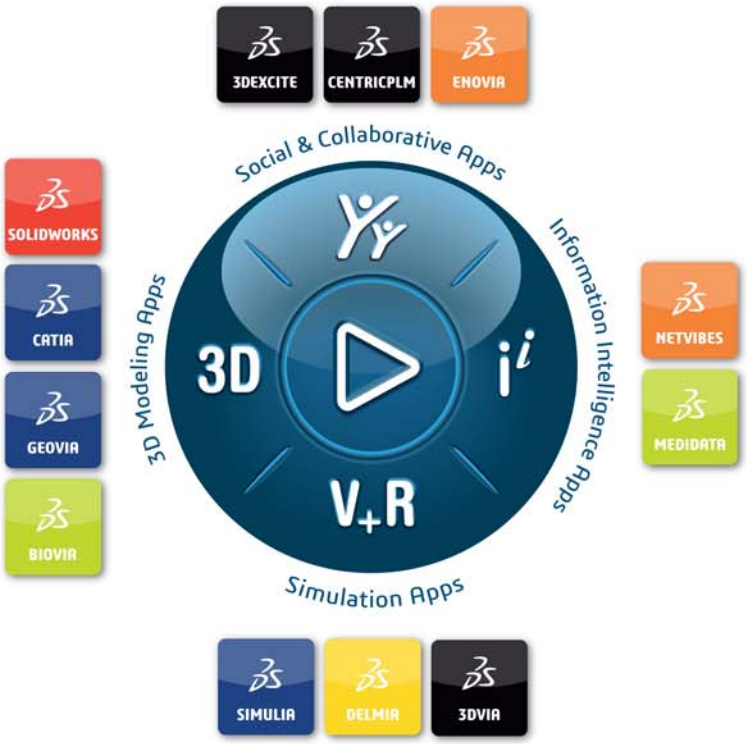
The following suggestions may help you to correct common installation errors that cause the verification procedure to fail:

- Make sure that the license file was installed properly. If there are problems with the license file, error messages will be written to standard output.

- Make sure that you have not tried to execute Abaqus products for which you are not licensed.
- Make sure that the operating system and compiler level are consistent with those specified for this release in the Program Directory. (See *http://media.3ds.com/support/progdir*. Choose SIMULIAAbaqus as the product **Line** and Abaqus2025 as the **Level**, then choose **Prerequisites** in the left-hand pane.)

If the error messages and these suggestions are insufficient to verify the installation, check for information about installation problems, resolutions, and verification in the Dassault Systèmes Knowledge Base.

If you are still unable to resolve the problem, contact your local office or representative for help. An overview of support options is available in the Dassault Systèmes Knowledge Base.



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