ALTAIR

ONLY FORWARD

Altair® AcuSolve® 2025

Altair AcuSolve Programs Reference Manual

Updated: 11/15/2024

Contents

AcuSolve Programs Reference Manual

AcuSolve utility programs covering preparatory and post-processing as well as user-defined functions and utility scripts.

Most of the programs are utility programs used in special circumstances. Most users, however, need to know and run only a few of these programs, in particular AcuRun and AcuTrans. These programs may have a large number of options. Again, most users only deal with a few of these options. In the description section of each command, the options most used are given as examples.

Command Line Options and Configuration Files

2

Each program requires zero or more options. These options may be supplied on the command line or placed in one or more configuration files.

For example, to invoke AcuRun for problem channel and input file `channel.inp`, you can either issue the command:

```
acuRun -pb channel -inp channel.inp
```

or place the following lines in the configuration file `Acusim.cnf`:

```
problem= channel
input_file= channel.inp
```

and issue the command

```
acuRun
```

If both command line argument and configuration options are provided, the former takes precedence.

Each option has a (long) descriptive and a (short) abbreviated name. Both names are equivalent. For example,

```
acuRun -pb channel
```

is equivalent to

```
acuRun -problem channel
```

Similarly, in `Acusim.cnf`:

```
problem= channel
```

is equivalent to

```
pb= channel
```

The descriptive name is meant to be used in the configuration files, while the abbreviated name is meant for the command line argument.

The environment variable ACUSIM_CNF_FILES can be used to specify multiple configuration files. This variable accepts colon-separated (:) lists of files. For example if ACUSIM_CNF_FILES is set as:

```
setenv ACUSIM_CNF_FILES ./Acusim.cnf:~/Acusim.cnf:/ACUSIM/Acusim.cnf
```

the files `Acusim.cnf` are searched in the local, home, and finally the installed `/ACUSIM` directories for each option. Missing files are ignored. If the environment variable is not set, it is assumed to have the following value:

```
./Acusim.cnf:~/Acusim.cnf
```

Empty lines and lines starting with a hash (#) are ignored in configuration files. All unrecognized options are also ignored. Each recognized line consists of one option and its value; they are separated by an "=" and any (or no) amount of white space (tabs and blanks).

To get the list of executable options along with their current values, issue the command with the -h (or -help) option. This causes the executable to print a usage message and exit. For example,

```
acuRun -inp channel.inp -h
```

prints a message such as the following and exits:

```
acuRun:
acuRun: Usage:
acuRun:
acuRun: acuRun [options]
acuRun:
acuRun: Options:
acuRun:
acuRun: -h print usage and exit
acuRun: help= TRUE [command-line]
acuRun: -pb <str> problem name
acuRun: problem= channel [./Acusim.cnf]
acuRun: -inp <str> input file name (_auto, use <problem>.inp)
acuRun: input_file= channel.inp [command-line]
...
acuRun: -v <int> verbose level
acuRun: verbose= 1 [default]
acuRun:
acuRun: Configuration Files:
acuRun:
acuRun: ./Acusim.cnf:~/Acusim.cnf:/ACUSIM/Acusim.cnf
acuRun:
acuRun: Release: 1.7
acuRun:
```

Many options are shared by more than one executable. For example,

```
verbose= 1
```

is interpreted by all programs. To assign an option to a single executable, add the name of the executable plus a period "." to the beginning of the option's name. For example,

```
verbose= 1
acuRun.verbose= 0
```

The verbose value of 1 is used by all programs except AcuRun, which uses the value of 0.

An option may be assigned to a single computer architecture by adding its name plus a period (.) to the beginning of the option's name. For example,

```
verbose= 1
```

```
HP.verbose= 2
```

The executable and architecture names may be combined in either order:

```
HP.acuRun.verbose= 0
acuRun.SUN.verbose= 1
```

Each option has one of five types.

String (str)

This is a user-given string value. On the command line, this option must be followed by a value. For example:

```
acuRun -problem channel
```

The equivalent option in the `Acusim.cnf` file is

```
problem= channel
```

Note: The (upper/lower) case of the option is preserved.

If a string contains white spaces, it must be enclosed in a pair of double-quotes. For example,

```
acuTrans -osis "wall surface"
```

The double-quotes may be omitted in `Acusim.cnf`:

```
surface_integral_output_sets= wall surface
```

Enumerated (enum)

This is a set of values available for a given option. On the command line, this option must be followed by a value. For example,

```
acuTrans -to table
```

or equivalently the `Acusim.cnf` option

```
translate_to= table
```

causes AcuTrans to translate its output data to table format.

Boolean (bool)

This turns on or off a given option. To turn on an option, simply issue the command with the option as the argument. For example:

```
acuRun -echo
```

To turn off an option, issue the command with a no_ appended to the option. For example:

```
acuRun -no_echo
```

In the configuration file, assign 1, t, true, yes or on, in upper or lower case, to specify on; or assign 0, f, false, no or off, in upper or lower case, to specify off. For example the following configuration options are equivalent to the above command line options:

```
echo_input= yes
```

and

```
echo_input= FALSE
```

Integer (int)

This is an integer value option. On the command line, this option must be followed by a value. For example:

```
acuRun -np 4
```

or equivalently in the configuration file,

```
num_processors= 4
```

Real (real)

This is a floating point value option. On the command line, this option must be followed by a value. For example:

```
acuSurf -angle 89.9
```

or equivalently in the configuration file,

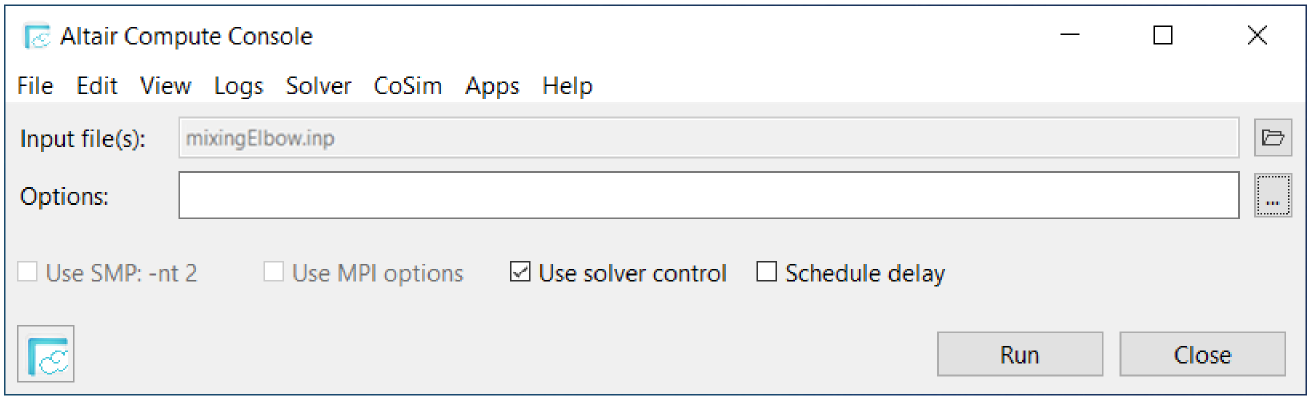
```
max_angle= 89.9
```

Run AcuSolve From Altair Compute Console

The Altair Compute Console provides an interactive graphical user interface (GUI) that allows you to select input files for AcuSolve runs.

It also allows you to define run options, activate multiple processor SMP and/or MPI runs. Any required environment variables for AcuSolve runs are pre-defined when the Altair Compute Console is used to submit jobs.

On Windows platforms, the Altair Compute Console can be started by choosing: `Start > Altair <Version> > Compute Console`.



Input file(s): mixingElbow.inp

Options:

Use SMP: -nt 2 [ ] Use MPI options [ ] Use solver control [✓] Schedule delay [ ]

Figure 1:

On Linux platforms, this utility can be started from your working directory as: `<install_dir>/altair/scripts/acc -gui mixingElbow.inp`.

For more information and to access the Altair Compute Console help, navigate to `Help > Console Manual` from the Altair Compute Console GUI.

Solver Programs 4

This chapter contains the solver programs.

This chapter covers the following:

Given an input file, the solver is run in three steps:

1. Run AcuPrep to parse the input file and prepare it for the solver;
2. Run AcuView to calculate radiation view factors, if necessary; and
3. Run AcuSolve to solve the problem.

These steps may be performed manually, or automatically using AcuRun. The latter is the preferred mode of operation, since AcuRun hides all the machine-dependent parallel environment setup steps.

AcuRun

Script to run AcuPrep and AcuSolve in scalar or parallel environments.

Syntax

`acuRun [options]`

Type

AcuSolve Solver Program Command

Description

AcuRun is a script that facilitates running a problem in the scalar and all supported parallel environments. This script is a one-step process that executes AcuPrep to preprocess the user input file, AcuView to calculate radiation view factors if needed, and AcuSolve to solve the problem.

For a general description of option specifications, see *Command Line Options and Configuration Files*.

Parameters

The following parameters are supported:

`help` *or* *`h`* *(boolean)*
If set, the program prints a usage message and exits. The usage message includes all available options, their current values, and the place where each option is set.

`problem` *or* *`pb`* *(string)*
The name of the problem is specified via this parameter. This name is used to generate and recognize internal files.

`problem_directory` *or* *`pdir`* *(string)*
The home directory of the problem; this is where the user input files reside. This option is needed by certain parallel processing packages. If *`problem_directory`* is set to ".", namely the Unix current directory, it is replaced by the full address of the current directory, as provided by the Unix getcwd command.

`working_directory` *or* *`dir`* *(string)*
All internal files are stored in this directory. This directory does not need to be on the same file system as the input files of the problem.

`file_format` *or* *`fmt`* *(enumerated) [binary]*
Format of the internal files:

ascii	ASCII files
binary	Binary files
hdf	Hierarchical data format (HDF5)

`process_modules` *or* *`do`* *(enumerated) [all]*
Specifies which modules to execute

prep	Execute only AcuPrep
view	Execute only AcuView
solve	Execute only AcuSolve. This requires that -acuPrep and AcuView (if necessary) have previously been executed.
prep-solve	Execute only AcuPrep and AcuSolve. This requires that AcuView has previously been executed.
all	Execute AcuPrep, AcuView, and AcuSolve

`install_smpd` or `ismpd` (boolean) [FALSE]
If set, this parameter installs the appropriate message passing daemon and then exits. Valid only for mp=impi, mp=mpi and mp=pmpi when running on Windows platforms. A message passing daemon is only necessary for mp=impi and mp=pmpi when running across multiple hosts. For mp=mpi, it is always necessary to have the message passing daemon service running before executing the simulation. See the *Altair HyperWorks Installation Guide* for more details.

`test_mpi` or `tmpi` (boolean) [FALSE]
If set, run a test of MPI only by executing the Pallas MPI benchmark.

`log_output` or `log` (boolean) [TRUE]
If set, the printed messages of AcuRun and the modules executed by it are redirected to the `.log` file -`problem.run.Log`; where run is the ID of the current run. Otherwise, the messages are sent to the screen and no logging is performed.

`append_log` or `append` (boolean) [FALSE]
If *`log_output`* is set, this parameter specifies whether to create a new `.log` file or append to an existing one.

`translate_output_to` or `tot` (enumerated) [none]
Automatically executes AcuTrans at the conclusion of the run to convert nodal output into the desired visualization format.

none	No translation is performed.
table	Output is translated to table format.
cgns	Output is translated to CGNS format.
ensight	Output is translated to EnSight format.
fieldview	Output is translated to FieldView format.
h3d	Output is translated to H3D format.

`copy_probe_data` or `toprobe` (boolean) [FALSE]
After a run is completed, create output data for 2D plotting in the directory `PROBE.DIR`, containing all files necessary for plotting, and allows for easy access to and lightweight transfer of data whether plotting data on a remote server or bringing data down to a local machine.

`copy_probe_data_directory` or `toprobedir` (string)
Copy probe data to the user-defined directory.

`acuoptistruct_type` or `aos` (enum) [none]
After a run is completed, create an OptiStruct input deck with type: none,ctnl,tnl,tl,snl,sl.

`input_file` or `inp` (string)
The input file name is specified via this option. If *`input_file`* is set to _auto, `problem.inp` is used.

`restart` or `rst` (boolean) [FALSE]
Create and run a simple restart file.

`fast_restart` or `frst` (boolean) [FALSE]
Perform a fast restart. Using this type of restart, the AcuPrep stage is skipped and the solver resumes from the last restart file that was written for the run that is being restarted. The number of processors must be held constant across the restart when the *`fast_restart`* parameter is used.

`fast_restart_run_id` or `frstrun` (integer) [0]
The run ID to use as the basis for the *`fast_restart`*. When this parameter is set to 0, the last available run ID is used.

`fast_restart_time_step_id` or `frstts` (integer) [FALSE]
The time step to use as the basis for the *`fast_restart`*. When this option is set to 0, the last available time step is used.

`echo_input` or `echo` (boolean) [TRUE]
Echo the input file to the file `problem.run.echo`; where run is the ID of the current run.

`echo_precedence` or `prec` (boolean) [FALSE]
Echo the resolved boundary condition precedence, if any, to the file `problem.bc_warning`.

`echo_help` or `eh` (boolean) [FALSE]
Echo the command help, that is, the results of acuRun -h, to the `.log` file.

`run_as_tets` or `tet` (boolean) [FALSE]
Run the problem as all tet mesh. All continuum fluid and solid elements are split into tetrahedron elements, and all thermal shell elements are converted to wedge elements. The surface facets are split into triangles. This option typically reduces the memory and CPU usage of AcuSolve.

`auto_wall_surface_output` or `awosf` (string) [TRUE]
Create surface output for *`auto_wall`* surfaces.

`user_libraries` or `libs` (string)
Comma-separated list of user libraries. These libraries contain the user-defined functions and are generated by acuMakeLib or acuMakeDll.

`viewfactor_file_name` or `vf` (string)
Name of the view factor file. If *`viewfactor_file_name`* is set to _auto, an internal name is used. This option is useful when you want to save the computed view factors for another simulation.

`message_passing_type` or `mp` (enumerated) [impi]
Type of message passing environment used for parallel processing:

none	Run in single-processor mode.
impi	Run in parallel on all platforms using Intel MPI.
pmpi	Run in parallel on all platforms using Platform MPI.
mpi	Run in parallel on all platforms using MPICH.
msmpi	Run in parallel on Windows using Microsoft MPI.
hpmpi	Run in parallel on all platforms using Platform MPI. **Note:** HP-MPI has been replaced by Platform MPI and this option is only included for backward compatibility. Identical to the setting mp=pmpi.
openmp	Run in parallel on all platforms using OpenMP.

Note: Select impi, msmpi or openmp for `message_passing_type` (mp) when you are running a simulation with UDF on 64-bit Windows.

`num_processors` or `np` *(integer) [1]*
This parameter specifies the total number of threads to be used. The number of physical processors used is `num_processors` divided by `num_threads`. If `message_passing_type` is set to none, `num_processors` is reset to 1, and vice versa.

`num_threads` or `nt` *(string) [_auto]*
This parameter specifies the number of threads per processor to be used. If num_threads>1, this option converts an MPI run to a hybrid MPI/OpenMP run. If this option is set to _auto, AcuRun automatically detects the number of cores on the machine and sets the thread count accordingly. This forces AcuSolve to use shared memory parallel message passing whenever possible.

`num_acuprep_threads` or `npt` *(string) [_auto]*
This parameter specifies the number of threads per processor to run AcuPrep.

`num_actual_solver_threads` or `nast` *(integer) [0]*
Number of threads the solver actually uses for each process. Deactivated by default. Activate to overwrite nThreads but use the number of processes and mesh partitioning style set by nProcs and nThreads.

`num_subdomains` or `nsd` *(integer) [1]*
This parameter specifies the number of subdomains into which the mesh is to be decomposed. If `num_subdomains` is less than `num_processors`, it is reset to `num_processors`. Hence, setting `num_subdomains` to 1 allows you to control both the number of subdomains and processors through the `num_processors` parameter.

`host_lists` or `hosts` *(string) [_auto]*
List of machines (hosts) separated by commas. This list may exceed `num_processors`; the extra hosts are ignored. This list may also contain fewer entries than `num_processors`. In this case, the hosts are populated in the order that they are listed. The process is then repeated, starting from

the beginning of the list, until each process has been assigned to a host. Multiple process may be assigned to a single host in one entry by using the following syntax: **`host:num_processes`**, where **`host`** is the host name, and **`num_processes`** is the number of processes to place on that machine. When this option is set to _auto, the local host name is used.

The following examples illustrate the different methods of assigning the execution hosts for a parallel run:

Example 1:

```
acuRun -np 6 -hosts node1,node2
```

Result: The node list is repeated to use the specified number of processors. The processes are assigned: node1, node2, node1, node2, node1, node2

Example 2:

```
acuRun -np 6 -hosts node1,node1,node2,node2,node1,node2
```

Result: The processes are assigned: node1, node1, node2, node2, node1, node2

Example 3:

```
acuRun -np 6 -hosts node1:2,node2:4
```

Result: The processes are assigned: node1, node1, node2, node2, node2, node2

`view_message_passing_type` or `vmp` (enumerated) [none]

Type of message passing environment used for parallel processing of view factors:

none	Run in single-processor mode.
impi	Run in parallel on all platforms using Intel MPI.
pmpi	Run in parallel on all platforms using Platform MPI.
mpi	Run in parallel on all platforms using MPICH.
msmpi	Run in parallel on Windows using Microsoft MPI.
hpmpi	Run in parallel on all platforms using Platform MPI. **Note:** HP-MPI has been replaced by Platform MPI and this option is only included for backward compatibility. Identical to the setting mp=pmpi.

`num_view_processors` or `vnp` (integer) [0]

This parameter specifies the number of processors to be used for view factor computation. If *`view_num_processors`* is 0, *`num_processors`* is assumed. If *`view_message_passing_type`* is set to none, the *`view_num_processors`* is reset to 1, and vice versa.

`view_host_lists` or `vhosts` (string) [_auto]

List of machines (hosts), separated by commas, to run the view factor computation. This list may exceed *`num_processors`*; the extra hosts are ignored. This list may also contain fewer entries than *num_processors*. In this case, the hosts are populated in the order that they are listed. The process is then repeated, starting from the beginning of the list, until each process has been assigned to a host. Multiple process may be assigned to a single host in one entry by using the following syntax: **host:num_processes**, where **host** is the host name, and **num_processes** is the number of processes to place on that machine. When this option is set to _auto, the *host_lists* parameter is used.

The following examples illustrate the different methods of assigning the execution hosts for a parallel run:

Example 1:

```
acuRun -np 6 -vhosts node1,node2
```

Result: The node list is repeated to use the specified number of processors. The processes are assigned: node1, node2, node1, node2, node1, node2

Example 2:

```
acuRun -np 6 -vhosts node1,node1,node2,node2,node1,node2
```

Result: The processes are assigned: node1, node1, node2, node2, node1, node2

Example 3:

```
acuRun -np 6 -vhosts node1:2,node2:4
```

Result: The processes are assigned: node1, node1, node2, node2, node2, node2

pbs (boolean) [FALSE]
Set parallel data, such as np and hosts, from PBS variable PBS_NODEFILE. Used when running AcuSolve through a PBS queue.

pbs_attach (boolean) [FALSE]
When toggled on, this parameter enables tighter integration with PBS schedulers. The AcuSolve processes are launched using the *pbs_attach* wrapper script that enables pausing, resuming and closer tracking of the batch processes.

lsf (boolean) [FALSE]
Set parallel data, such as np and hosts, from LSF variable LSB_HOSTS. Used when running AcuSolve through a LSF queue.

sge (boolean) [FALSE]
Set parallel data, such as np and hosts, from the $TMPDIR/machines SGE file. Used when running AcuSolve through a Sun Grid Engine queuing system.

nbs (boolean) [FALSE]
Set parallel data, such as np and hosts, from NBS var NBS_MACHINE_FILE.

ccs (boolean) [FALSE]
Set parallel data, such as np and hosts, from the CCP_NODES environment variable set by the CCS scheduler. Used when running AcuSolve through a Microsoft -Compute Cluster Server job scheduler.

`slurm` *(boolean) [FALSE]*
: Set parallel data, such as np and hosts, from output of the "scontrol show hostnames" command. Slurm Workload Manager, formerly known as Simple Linux Utility for Resource Management or SLURM, is an open source cluster management and job scheduling system for large and small Linux clusters.

`numactl` *(boolean) [FALSE]*
: Use the Linux operating system command numactl to bind processes to cores. This option is only valid when running on Linux platforms.

`run_with_gpu` *or* `gpu` *(boolean) [FALSE]*
: Enabling GPU acceleration of the solver GPU can speed up the linear solvers, the most run time-consuming part in steady-state simulations. Currently only single GPU device is supported.

`gpuid` *(integer) [0]*
: Choose specific GPU device when multiple GPUs are available. For a system with multiple Nvidia or AMD GPUs installed, use nvidia-smi or rocm-smi to list all of the available devices. `gpuid` specifies which device to use.

`gputype` *(enumerated) [cudasp]*
: Specify GPU type Nvidia (cudasp) or AMD Radeon (rocmsp): cudasp, rocmsp.

`run_with_gpu_mesh` *or* `meshgpu` *(boolean) [TRUE]*
: Solve mesh deformation equation using GPU. Used with the -gpu option.

`run_with_gpu_temp` *or* `tempgpu` *(boolean) [TRUE]*
: Solve thermal equation using GPU. Used with the -gpu option.

`run_with_gpu_flow` *or* `flowgpu` *(boolean) [TRUE]*
: Solve flow equation using GPU. Used with the -gpu option.

`run_with_gpu_sa` *or* `sagpu` *(boolean) [TRUE]*
: Solve Spalart-Allmaras turbulence model using GPU. Used with the -gpu option.

`acuprep_executable` *or* `acuprep` *(string)*
: Full path of the AcuSolve executable.

`acuview_executable` *or* `acuview` *(string)*
: Full path of the AcuSolve executable.

`acusolve_executable` *or* `acusolve` *(string)*
: Full path of the AcuSolve executable.

`acutrans_executable` *or* `acutrans` *(string)*
: Full path of the AcuTrans executable, used with the -translate_output_to option.

`acuoptistruct_executable` *or* `acuoptistruct` *(string)*
: Full path of the acuOptiStruct executable.

`mpirun_executable` *or* `mpirun` *(string)*
: Full path to mpirun, mpimon, dmpirun or prun executable used to launch an MPI job. If _auto, executable is determined internally. When using mp=pmpi, you can also set this option to _system, which will use the system level installation of Platform MPI.

mpirun_options or mpiopt (string)

Arguments to append to the mpirun command that is used to launch the parallel processes. When this option is set to _auto, AcuRun internally determines the options to append to the mpirun command.

`remote_shell` *or* `rsh` *(string)*

Remote shell executable for MPI launchers. Usually this is rsh (or remesh on HP-UX) but can be set to ssh if needed. When this option is set to _auto, executable is determined internally.

`automount_path_remove` *or* `arm` *(string) [_none]*

Automount path remove. If `automount_path_remove` is set to _none, this option is ignored. See below for details.

`automount_path_replace` *or* `arep` *(string) [_none]*

Automount path replacement. If `automount_path_replace` is set to _none, this option is ignored. See below for details.

`automount_path_name` *or* `apath` *(string) [_none]*

Automount path name. If `automount_path_name` is set to _none, this option is ignored. See below for details.

`lm_daemon` *or* `lmd` *(string)*

Full address of the network license manager daemon, acuLmd, that runs on the server host. When this option is set to _auto, the value is internally changed to `$ACUSIM_HOME/$ACUSIM_MACHINE/bin/acuLmd`. This option is only used when lm_service_type = classical.

`lmtype` *or* `lm_service_type` *(enumerated) [hwu]*

Type of the license manager service:

hwu	Altair Units licensing
token	Token based licensing
classical	Classical AcuSim style licensing

`lm_server_host` *or* `lmhost` *(string)*

Name of the server machine on which the network license manager runs. When this option is set to _auto,, the local host name is used. This option is only used when lm_service_type = classical.

`lm_port` *or* `lmport` *(integer) [41994]*

TCP port number that the network license manager uses for communication. This option is only used when lm_service_type = classical.

`lm_license_file` *or* `lmfile` *(string) [_auto]*

Full address of the license file. This file is read frequently by the network license manager. When this option is set to _auto, the value is internally changed to `$ACUSIM_HOME/$ACUSIM_MACHINE/license.dat`. This option is only used when lm_service_type = classical.

`lm_checkout` *or lmco (string) [_auto]*

Full address of the license checkout co-processor, AcuLmco. This process is spawned by the solver on the local machine. When this option is set to _auto, the value is internally changed to `$ACUSIM_HOME/$ACUSIM_MACHINE/bin/acuLmco`. This option is only used when lm_service_type = classical.

`lm_queue_time` or `lmqt` (integer) [3600]
The time, in seconds, to camp on the license queue before abandoning the wait. This option is only used when lm_service_type = classical.

`lm_type` or `lmt` (enumerated) [classical]
Type of license checkout, used with *`lm_service_type`*=classical:

classical	Classical
light	Light

`external_code_host` or `echost` (string) [_auto]
External code host name for establishing socket connection.

`external_code_port` or `ecport` (integer) [0]
External code port number for establishing socket connection.

`external_code_wait_time` or `ecwait` (real) [3600]
How long to wait for the external code to establish socket connection.

`external_code_echo` or `ececho` (boolean) [FALSE]
Echo messages received from external code for stand-alone debugging.

`line_buff` or `lbuff` (boolean) [FALSE]
Flush standard output after each line of output.

`print_environment_variables` or `printenv` (boolean) [FALSE]
Prints all the environment variables in AcuRun and those read by AcuSolve.

`mpitune` or `tune` (boolean) [FALSE]
When toggled on, this parameter will run Intel's mpitune application to optimize message passing speed. Only available when using mp=impi.

`verbose` or `v` (integer) [1]
Set the verbose level for printing information to the screen. Each higher verbose level prints more information. If *`verbose`* is set to 0, or less, only warning and error messages are printed. If *`verbose`* is set to 1, basic processing information is printed in addition to warning and error messages. This level is recommended. *`verbose`* levels greater than 1 provide additional information useful only for debugging.

Examples

Given the input file `channel.inp` of problem channel, the simplest way to solve this problem in scalar mode is:

```
acuRun -pb channel -np 1
```

or alternatively, place the options in the configuration file `Acusim.cnf` as

```
problem= channel
num_processors= 1
```

and invoke AcuRun as:

```
acuRun
```

In the above example, AcuRun executes AcuPrep to read and process the user input file, bypasses AcuView since there is no radiation data, and runs

AcuSolve in scalar mode.

To execute AcuPrep, AcuView, and AcuSolve separately and log the output messages, issue the following commands:

```
acuRun -do prep -log
acuRun -do view -log -append
acuRun -do solve -log -append
```

where the last two invocations append to the `.log` file created by the first invocation. For problems with a large number of radiation surfaces, AcuView can be very expensive to run. This step can be skipped on the second and successive runs, if the geometry has not changed, by executing:

```
acuRun -do prep-solve -log
```

To run the channel problem in parallel on 10 processors using MPI, issue the following command:

```
acuRun -pb channel -mp mpi -np 10
```

Here AcuRun runs AcuPrep to prepare the data, sets up all MPI related data and runs acuSolve-mpi via mpirun. The *`message_passing_type`* (mp) option is typically set by the system administrator in the system `Acusim.cnf` file. To run the channel problem in parallel on four processors with six threads per processor using a hybrid MPI/OpenMP strategy, issue the following command:

```
acuRun -pb channel -mp impi -np 6
```

Here AcuRun runs AcuPrep to prepare the data, sets the MPI related data and the OpenMP parallel environment, and runs AcuSolve.

To run AcuSolve in parallel across multiple Windows hosts, it is first necessary to install the message passing daemon onto each system. This is accomplished using the following command (executed on each host):

```
acuRun -pb channel -ismpd -mp impi
```

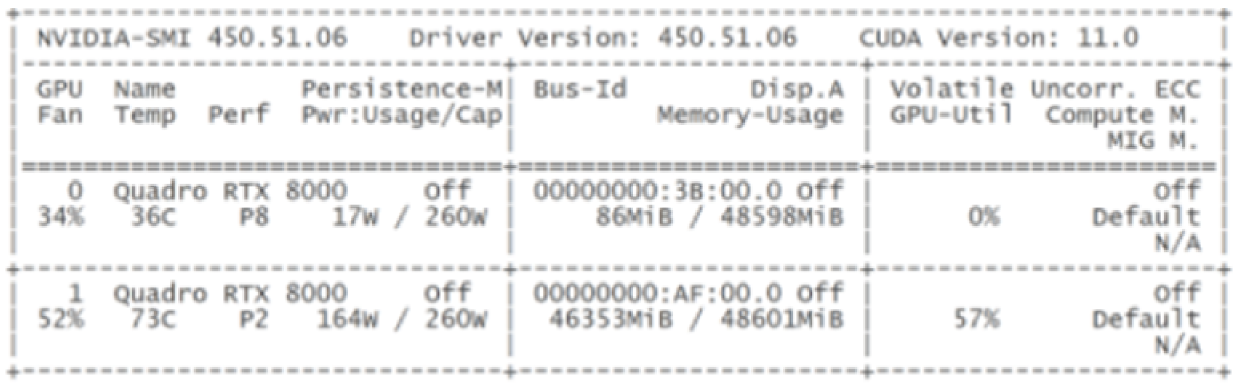
Since this command attempts to install a service on the system, it does require administrative privileges. This procedure only needs to be performed once on each machine. After the service has been installed, the Windows operating system will automatically restart it after the system is rebooted. Once the service is installed, the solver is launched as follows:

```
acuRun -pb channel -mp impi -np 2 -nt 1 -hosts winhost1,winhost2
```

When AcuRun launches the parallel job, it first checks for the presence of the appropriate message passing daemon and issues an error if the service is not found.

To list the GPU devices on your system, issue the command nvidia-smi.

For example, if you have the following list of gpu cards after typing "nvidia-smi"



```
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 450.51.06    Driver Version: 450.51.06    CUDA Version: 11.0     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|                               |                      |               MIG M. |
|===============================+======================+======================|
|   0  Quadro RTX 8000     Off  | 00000000:3B:00.0 Off |                  Off |
| 34%   36C    P8    17W / 260W |     86MiB / 48598MiB |      0%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+
|   1  Quadro RTX 8000     Off  | 00000000:AF:00.0 Off |                  Off |
| 52%   73C    P2   164W / 260W |  46353MiB / 48601MiB |     57%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+
```

Figure 2:

You can choose any of the GPU cards by typing

`acuRun -gpu -gpuid 0` (when you want to use the GPU card ID 0).

`acuRun -gpu -gpuid 1` (for GPU card ID 1)

By default, every equation that is supported by GPU will be solved on GPU if GPU acceleration is enabled by specifiying "-gpu". Alternatively, you can disable GPU acceleration of individual equations. For example, in an ALE simulation with mesh movement, specifying "-no_meshgpu" instructs the solver to solve the mesh movement equation on CPU. Those options are useful if the mesh size of the simulation reaches the limit of what the GPU RAM size can allow.

The *run_as_tet* parameter may be used to split the mesh into an all tetrahedron mesh. For example,

```
acuRun -pb channel -tet
```

This will typically reduce the memory and CPU usages of AcuSolve, without reducing the accuracy. For fluid and solid media, brick elements are split into five or six tets, wedges are split into three tets, pyramids are split into two tets, and tets are kept as is. For thermal shells, brick elements are split into two wedges and wedge elements are kept as is. All surface facets are split into corresponding triangles. The split is consistent and conformal, in the sense that a quad face common to two elements is split in the same direction, and two quad faces with pair-wise periodic nodes have the same diagonal split. Moreover, the algorithm attempts to reduce large angles. The user element and surface numbers are modified by multiplying the user-given numbers by 10 and adding the local tet number. This only effects user numbers passed to user-defined functions.

When running a problem with enclosure radiation, AcuRun detects this fact and runs AcuView by default. Again by default, the view factors are stored in the working directory under an internal name. The name and location of this file may be changed using the *viewfactor_file_name* parameter. This is useful when multiple problems are run with the same geometry, radiation surfaces and agglomeration. For example assume input files `test1.inp` and `test2.inp` have the same geometry, radiation surfaces, and agglomeration parameters. In this case, to save CPU time they may be executed as:

```
acuRun -inp test1.inp -vf test.vf -do all
acuRun -inp test2.inp -vf test.vf -do prep-solve
```

The *restart* parameter creates and runs the simplest restart input file:

```
RESTART{}
RUN{}
```

The *fast_restart* parameter is a special form of restart that bypasses AcuPrep entirely. When specifying this type of restart, the solver simply resumes from the run specified in the *fast_restart_run_id* and *fast_restart_time_step* with no modifications to the solver settings. The source run for the *fast_restart* must utilize the same number of processors as the current run. It should be noted that a limited number of modifications can be made when using the *fast_restart* parameter by using the AcuModSlv script to modify the solver directives file that is written into the working directory. This is an advanced option and is rarely used.

The parameters *acuprep_executable*, *acuview_executable* and *acusolve_executable* are advanced parameters and should rarely be changed. They are used to access executables other than the current one. If used, the version number of all modules must be the same.

Running AcuSolve and/or AcuView in distributed memory parallel requires that the problem and working directories be accessible by all involved machines. Moreover, these directories must be accessible by the same name. In general, *working_directory* does not pose any difficulty, since it is typically specified relative to the *problem_directory*. However, *problem_directory* requires special attention.

By default, *problem_directory* is set to the current Unix directory ".". AcuRun translates this address to the full Unix address of the current directory. It then performs three operations on this path name in order to overcome some potential auto mount difficulties: first, if the path name starts with the value of *automount_path_remove*, this starting value is removed. Second, if the path name starts with *from_path*, it is replaced by *to_path*, where the string "from_path,to_path" is the value of *automount_path_replace* parameter. Third, if the path name does not start with the value of *automount_path_name*, it is added to the start of the path name. These operations are performed only if *problem_directory* is not explicitly given as a command line option to AcuRun, and also if the option associated with each operation is not set to _none. If problem_directory is explicitly given as a command line option, you are responsible for taking care of all potential automount problems.

The parameters *lm_daemon*, *lm_server_host*, *lm_port*, *lm_license_file*, *lm_checkout*, and *lm_queue_time* are used for checking out a license from a network license manager when using classical Acusim licensing (*lm_service_type* = classical). These options are typically set once by the system administrator in the installed system configuration file, `Acusim.cnf`. Given the proper values, AcuSolve automatically starts the license manager if it is not already running. For a more detailed description of these parameters consult acuLmg. This executable is used to start, stop, and query the license manager. These options are ignored when *lm_service_type* = hwu.

There are four options that control how AcuSolve interacts with an external code for Direct Coupling Fluid Structural Interaction problems. The parameter *external_code_host* specifies a host name for establishing the socket connection and overwrites the *socket_host* parameter in the `EXTERNAL_CODE` command. Similarly, *external_code_host* specifies the port for the socket connection and overwrites *socket_port* in `EXTERNAL_CODE`. The maximum amount of time to wait for the external code connection is given by *external_code_wait*. The parameter *external_code_echo* echos the messages received by the external code into a `.ecd` file. This file may subsequently be given as the value of *fifo_receive_file* of the `EXTERNAL_CODE` command with fifo type to perform a stand-alone run of AcuSolve.

AcuRun duplicates many of the AcuPrep, AcuView, and AcuSolve options. However, there are some advanced options in these modules that are not covered by AcuRun; See AcuPrep, AcuView, and AcuSolve for complete lists of their options. These options may be specified in the configuration files before running AcuRun.

AcuSub

Script to submit an AcuSolve job to PBS/LSF/CCS queue.

Syntax

`acuSub [options]`

Type

AcuSolve Solver Program Command

Description

AcuSub is a script that creates a PBS/LSF/CCS submission file.

Parameters

The following parameters are supported:

`help` or `h` (boolean)
: If set, the program prints a usage message and exits. The usage message includes all available options, their current values, and the place where each option is set.

`problem` or `pb` (string)
: The name of the problem is specified via this parameter. This name is used to generate and recognize internal files.

`sched_problem` or `spb` (string)
: The name of the scheduler problem is specified via this parameter.

`problem_directory` or `pdir` (string)
: The home directory of the problem; this is where the user input files reside.

`working_directory` or `dir` (string)
: All internal files are stored in this directory. This directory does not need to be on the same file system as the input files of the problem.

`process_modules` or `do` (enumerated) [all]
: Specifies which modules to execute.

prep	Execute only AcuPrep.
view	Execute only AcuView.
solve	Execute only AcuSolve. This requires that -acuPrep and AcuView (if necessary) have previously been executed.
prep-solve	Execute only AcuPrep and AcuSolve. This requires that AcuView has previously been executed.
all	Execute AcuPrep, AcuView, and AcuSolve.
trace	Execute AcuTrace.

twowaytrace	Execute two-way coupling AcuTrace.

`input_file` or `inp` *(string)*
: The input file name is specified via this option. If `input_file` is set to _auto, `problem.inp` is used.

`trace_input_file` or `tin` *(string)*
: The trace input file name is specified via this option. If `trace_input_file` is set to _auto, `problem.tin` is used.

`restart` or `rst` *(boolean) [FALSE]*
: Create and run a simple restart file.

`fast_restart` or `frst` *(boolean) [FALSE]*
: Perform a fast restart. Using this type of restart, the AcuPrep stage is skipped and the solver resumes from the last restart file that was written for the run that is being restarted.

> **Note:** The number of processors must be held constant across the restart when the `fast_restart` parameter is used.

`fast_restart_run_id` or `frstrun` *(integer) [0]*
: The run ID to use as the basis for the `fast_restart`. When this parameter is set to 0, the last available run ID is used.

`fast_restart_time_step_id` or `frstts` *(integer) [FALSE]*
: The time step to use as the basis for the `fast_restart`. When this option is set to 0, the last available time step is used.

`num_processors` or `np` *(integer) [1]*
: This option specifies the number of processors.

`num_processors per node` or `ppn` *(integer) [1]*
: This option specifies the number of processors per node to be used.

`num_threads` or `nt` *(string) [_auto]*
: This option specifies the number of threads per processor to be used. If `num_threads`>1, this option converts an MPI run to a hybrid MPI/OpenMP run. If this option is set to _auto, acuSub automatically detects the number of cores on the machine and sets the thread count accordingly.

`run_with_gpu` or `gpu` *(boolean) [FALSE]*
: Submit a job to a GPU machine. Enabling GPU acceleration of the solver GPU can speed up the linear solvers, the most run time-consuming part in steady-state simulations. Currently only single GPU device is supported.

`queue` or `q` *(string) [_undefined]*
: Queue name to submit a job.

`requested_node` or `req_nodes` *(string) [_undefined]*
: Comma separated list of requested nodes (used only with ccs).

`job_template` or `template` *(string)[_undefined]*
: Job template to use when submitting a job (used with ccs).

`constraint` *(string) [_none]*
Constraint to request (used with slurm, pbspro, and ccs).

`parallel_environment` or `pe` *(string) [_undefined]*
Parallel environment to use (used with sge).

`wall_time` or `time` *(string) [_undefined]*
Total wall clock time for the job (minutes).

`dependency` or `dep` *(string) [_none]*
Job ID that must complete before this one begins (i.e. dependency).

`command` or `cmd` *(string) [_auto]*
Command to execute in the batch script in place of acuRun.

`additional_opts` or `opts` *(string) [_none]*
Additional options to add to the acuRun command.

`additional_trace_opts` or `traceopts` *(string) [_none]*
Additional options to add to the acuRunTrace command.

`preliminary_command` or `pre_cmd` *(string) [_none]*
Additional commands to execute before the acuRun command.

`post_command` or `post_cmd` *(string) [_none]*
Additional commands to execute after the acuRun command.

`additional_scheduler_command` or `sched_cmd` *(string) [none]*
Additional scheduler directives to add to the submission command.

`submit_scrtipt` or `sub` *(boolean) [TRUE]*
Submit a batch script after creating.

`shell` *(enumerated) [sh]*
Shell to use in batch script: sh, csh.

`path_replace` *(boolean) [FALSE]*
Flag to perform path replacement for network file systems.

`path_target` *(string) [_none]*
Comma separated list of strings to search for in all paths.

`path_replacement` *(string) [_none]*
Comma separated list of strings to replace `path_target` with.

`sched` *(enumerated) [pbspro]*
Type of batch scheduler: openpbs, pbspro, ccs, lsf, slurm, sge.

`pbs_attach` *(boolean) [FALSE]*
When toggled on, this parameter enables tighter integration with PBS schedulers. The AcuSolve processes are launched using the `pbs_attach` wrapper script that enables pausing, resuming and closer tracking of the batch processes (used only with pbspro).

`perl` *(string) [_auto]*
Path to the perl bin directory (used on Windows only).

`job` *(string) [_auto]*
: Path to the ccp job command (used on Windows only).

`verbose` *or* `v` *(integer) [1]*
: Set the verbose level for printing information to the screen. Each higher verbose level prints more information. If *`verbose`* is set to 0, or less, only warning and error messages are printed. If *`verbose`* is set to 1, basic processing information is printed in addition to warning and error messages. This level is recommended. *`verbose`* levels greater than 1 provide additional information useful only for debugging.

Examples

Below is an example of submitting a job using acuSub. The channel problem can be submitted in parallel on 32 processors (np) on two nodes, with each node having 16 processors per node (ppn):

```
acuSub -pb channel -np 32 -ppn 16
```

or alternatively, place the options in the configuration file `Acusim.cnf` as

```
problem = channel
num_processors = 32
ppn = 16
```

and invoke AcuSub as:

```
acuSub
```

AcuPrep

Prepares the user data to be run by AcuSolve.

Syntax

`acuPrep [options]`

Type

AcuSolve Solver Program Command

Description

Prepares the user data to be run by AcuSolve. The following options are supported; for a general description of option specifications, see Command Line Options and Configuration Files.

In the following option descriptions, the full name of each option is followed by its abbreviated name and its type.

`help` *or* *`h`* *(boolean) [FALSE]*

If set, the program prints a usage message and exits. The usage message includes all available options, their current values and the place where each option is set.

`problem` *or* *`pb`* *(string)*

The name of the problem is specified via this option. This name is used to generate and recognize internal files.

`input_file` *or* *`inp`* *(string)*

Name of the input file is specified via this option. If *`input_file`* is set to _auto, `problem.inp` is used.

`working_directory` *or* *`dir`* *(string)*

All internal files are stored in this directory. This directory does not need to be on the same file system as the input files of the problem.

`file_format` *or* *`fmt`* *(enumerated)*

Format of the internal files:

ascii	ASCII files
binary	Binary files
hdf	Hierarchical data format (HDF5)

`num_acuprep_threads` *or* *`npt`* *(integer) [8]*

This parameter specifies the number of threads per processor to run AcuPrep.

`num_subdomains` *or* *`nsd`* *(integer) [1]*

This option specifies the number of subdomains into which the mesh is to be decomposed. This number must be greater than or equal to the number of processors to be used to run the problem. The number of subdomains, and similarly the number of processors, may be changed at restart.

`num_solver_threads` or *`nst`* *(integer) [1]*

This option specifies the number of threads on which the solution will be run. When this value is greater than 1, a two level domain decomposition is performed. The first optimizes for the distributed memory message passing, and the second for the shared memory message passing.

`echo_input` or *`echo`* *(boolean) [TRUE]*

Echo the input file to the file `problem.run.echo`; where run is the ID of the current run.

`echo_precedence` or *`prec`* *(boolean) [FALSE]*

Echo the resolved boundary condition precedence (if any) to the file `problem.bc_warning`.

`run_as_tets` or *`tet`* *(boolean) [FALSE]*

The *`run_as_tets`* option may be used to split the mesh into an all tet mesh. All continuum fluid and solid elements are split into tetrahedron elements, and all thermal shell elements are converted to wedge elements. The surface facets are split into triangles. This option typically reduces the memory and CPU usage of AcuSolve.

`domain_decomposition` or *`ddc`* *(enumerated) [metis]*

Domain decomposition technique to be used for decomposing the mesh into subdomains. This advanced option should rarely be changed from its default value.

metis	Use METIS from University of Minnesota. METIS is based on technology described in the following reference: "*A Fast and Highly Quality Multilevel Scheme for Partitioning Irregular Graphs*". George Karypis and Vipin Kumar. SIAM Journal on Scientific Computing, Vol. 20, No. 1, pp. 359-392, 1999.
acuddc	Causes the domain decomposition to be performed as a separate process using AcuDdc.
restart	Use the domain decomposition from the restart files. This requires that the nodal coordinates, element connectivity, and number of subdomains remain unaltered from the restart run.

`acuddc_executable` or *`acuddc`* *(string)*

Full path to the AcuDdc executable. This advanced option should rarely be changed from its default value.

`max_ddc_agglomeration` or *`mda`* *(string) [_auto]*

If *`mda`* is larger than 1, a pre-agglomeration of elements into clusters of size *`mda`* is performed prior to domain decomposition (DDC). This reduces the DDC memory requirement, but with the potential cost of yielding an inferior decomposition. This option is particularly useful for very large problems when limited memory resources are available. When this option is set to _auto, the agglomeration size is automatically set to yield less than 100 million agglomerated element clusters for the purpose of domain decomposition.

`cache_size` or *`cache`* *(integer) [1024]*

Size of the computer cache in Kilobytes (KB). This is used to determine the optimal element block size. This advanced option should rarely be changed from its default value.

`auto_generate_ebc` or `ebc` *(boolean) [TRUE]*
: Assign `mass_flux` of type free for fluid boundary surfaces where no mass flux element boundary condition is assigned. This advanced option should rarely be changed from its default value.

`auto_reference_frame_interface` or `rfi` *(boolean) [TRUE]*
: Generate reference frame interfaces. This advanced option should rarely be changed from its default value.

`auto_set_back_flow_diffusion` or `bfd` *(boolean) [FALSE]*
: Set back flow diffusion for outflow Simple Boundary Conditions. This advanced option should rarely be changed from its default value.

`granular_no_slip` or `grannoslip`*(boolean) [FALSE]*
: Enforce no-slip condition for granular flow walls. This advanced option should rarely be changed from its default value.

`sort_nodes_and_elements` or `sort` *(boolean) [TRUE]*
: Sort the nodes and elements to localize data access. On cache-based machines, this option improves performance. This advanced option should rarely be changed from its default value.

`ignore_negative_jacobian` or `inj` *(boolean) [FALSE]*
: Ignore elements with negative jacobians. This advanced option should rarely be changed from its default value.

`line_buff` or `lbuff` *(boolean) [FALSE]*
: Flush standard output after each line of output.

`verbose` or `v` *(integer) [1]*
: Set the verbose level for printing information to the screen. Each higher verbose level prints more information. If `verbose` is set to 0 (or less), only warning and error messages are printed. If `verbose` is set to 1, basic processing information is printed in addition to warning and error messages. This level is recommended. `verbose` levels greater than 1 provide information useful only for debugging.

Examples

To run a problem, first write an input file, then execute AcuPrep to preprocess the input file into a format used by the solver, execute AcuView if necessary, and then run the solver by executing AcuSolve. Finally, translate the solution using AcuTrans. AcuPrep, AcuView and AcuSolve are not normally run by themselves, but rather the script AcuRun is used to run these programs.

Given the input file `channel.inp`, for example, the simplest form for running AcuPrep is

```
acuPrep -pb channel
```

or alternatively place the option in the configuration file `Acusim.cnf` as

```
problem= channel
```

and invoke AcuPrep as

```
acuPrep
```

The command

```
acuPrep -ddc acuddc
```

will create a file with a `.mts` extension, then launch (spawn) AcuDdc with this as an input file, wait for it to return, then read the `.ddc` file it produces and continue. This can help to reduce the amount of memory needed by AcuPrep for very large problems. If a `.ddc` file already exists and the nodal coordinates, element connectivity, and number of subdomains remain unaltered from when the file was created, then the domain decomposition step can be skipped by executing:

```
acuPrep -ddc restart
```

AcuDdc

Performs domain decomposition.

Syntax

`acuDdc [options]`

Type

AcuSolve Solver Program Command

Description

Normally domain decomposition (DDC) is performed automatically by AcuPrep. Advanced users can use AcuDdc to do this step manually. AcuPrep generates the input file needed by AcuDdc. The file name can be found in the `.log` file and then it is given in the ifile option. It has either a `.mts` extension, for ASCII files, or a `.mts.B` extension, for binary files. AcuDdc then writes its output to `ofile`, which should be the same file name but with a `.ddc` or `.ddc.B` extension. This file is read, and the DDC step is bypassed, by -acuPrep when it is restarted:

```
acuPrep -ddc restart
```

In the following, the full name of each option is followed by its abbreviated name and its type. For a general description of option specifications, see Command Line Options and Configuration Files. See below for more individual option details:

`help` or `h` (boolean)
: If set, the program prints a usage message and exits. The usage message includes all available options, their current values and the place where each option is set.

`input_file` or `ifile` (string)
: Name of the input file. This is a mandatory parameter.

`output_file` or `ofile` (string)
: Name of the output file. This is a mandatory parameter.

`domain_decomposition_technique` or `ddctech` (enumerated)
: Domain decomposition technique to be used for decomposing the mesh into subdomains:

metis	Use METIS from University of Minnesota. METIS is based on technology described in the following reference: "*A Fast and Highly Quality Multilevel Scheme for Partitioning Irregular Graphs*". George Karypis and Vipin Kumar. SIAM Journal on Scientific Computing, Vol. 20, No. 1, pp. 359-392, 1999.

`max_ddc_agglomeration` or `mda` (integer)
: Agglomerates elements into clusters of size *`mda`* prior to domain decomposition. This option may be necessary for extremely large problems.

`file_format` or `fmt` (enumerated)
: Format of the internal files:

ascii	ASCII files
binary	Binary files

verbose or v (integer)

Set the verbose level for printing information to the screen. Each higher verbose level prints more information. If *verbose* is set to 0, or less, only warning and error messages are printed. If *verbose* is set to 1, basic processing information is printed in addition to warning and error messages. This level is recommended. *verbose* levels greater than 1 provide information useful only for debugging.

Examples

To use AcuDdc to manually create an output file usable by AcuPrep:

```
acuDdc -ifile ACUSIM.DIR/channel.6.mts -ofile ACUSIM.DIR/channel.6.ddc \
-ddtech metis -mda 0 -fmt binary -v 1
```

The input and output file names (`ifile` and `ofile`) are mandatory parameters. An error will be issued if either one is not given.

The option *domain_decomposition_technique* is an advanced option and should rarely be changed from its default value.

The parameter *file_format* determines if the input and output files are saved in ASCII or binary format. The extension `.B` on the file name parameters is optional. It will be added or removed automatically based on the value of *file_format*.

If *mda* is larger than 1, a pre-agglomeration of elements into clusters of size *mda* is performed prior to domain decomposition. This reduces the *DDC* memory requirement, but with the potential cost of yielding an inferior decomposition. This option is particularly useful for extremely large problems with over 200 million nodes or elements.

AcuView

Calculate radiation view factors for AcuSolve.

Syntax

`acuView [options]`

Type

AcuSolve Solver Program Command

Description

Calculates radiation view factors for AcuSolve. As with AcuPrep, AcuView is not normally run by itself, but rather the script AcuRun is used to run it.

In the following, the full name of each option is followed by its abbreviated name and its type. For a general description of option specifications, see *Command Line Options and Configuration Files*. See below for more individual option details:

`help` or *`h`* *(boolean)*
: If set, the program prints a usage message and exits. The usage message includes all available options, their current values and the place where each option is set.

`problem` or *`pb`* *(string)*
: The name of the problem is specified via this option. This name is used to generate and recognize internal files.

`problem_directory` or *`pdir`* *(string)*
: The home directory of the problem. This is where the user input files reside. This option is needed by certain parallel processing packages. If *`problem_directory`* is set to ".", namely the Unix current directory, it is replaced by the full address of the current directory, as provided by the Unix getcwd command.

`working_directory` or *`dir`* *(string)*
: All internal files are stored in this directory. This directory does not need to be on the same file system as the input files of the problem.

`file_format` or *`fmt`* *(enumerated)*
: Format of the internal files:

ASCII	ASCII files
binary	Binary files

`num_processors` or *`np`* *(enumerated)*
: This option specifies the number of processors to be used.

`num_threads` or *`nt`* *(enumerated)*
: This option specifies the number of threads per processor to be used. This option is included here for compatibility with AcuSolve. It currently has no impact on the executable.

`viewfactor_file_name` or `vf` *(string)*

Name of the view factor file. If `viewfactor_file_name` is set to _auto, an internal name is used.

`max_smoothing_passes` or `msp` *(enumerated)*

Maximum number of iterations of the least-squares smoothing algorithm maintaining the reciprocity and row-sum properties. Multiple passes are needed only for negative view factors. Error issued if the algorithm fails.

`viewfactor_storage` or `storage` *(enumerated)*

Internal format of storing view factors:

simple	Store all entries
sparse	Store non-zeros in sparse column format
compact	Store non-zeros in sparse compact format

`lm_daemon` or `lmd` *(string)*

Full address of the network license manager daemon, AcuLmd, that runs on the server host. When this option is set to _auto, the value is internally changed to `$ACUSIM_HOME/$ACUSIM_MACHINE/bin/acuLmd`. This option is only used when `lm_service_type` = classical.

`lmtype` or `lm_service_type` *(enumerated)*

Type of the license manager service:

hwu	Altair Units licensing
token	Token based licensing
classical	Classical Acusim style licensing

`lm_server_host` or `lmhost` *(string)*

Name of the server machine on which the network license manager runs. When set to _auto, the local host name is used. This option is only used when `lm_service_type` = classical.

`lm_port` or `lmport` *(enumerated)*

TCP port number that the network license manager uses for communication. This option is only used when `lm_service_type` = classical.

`lm_license_file` or `lmfile` *(string)*

Full address of the license file. This file is read frequently by the network license manager. When this option is set to _auto, the value is internally changed to `$ACUSIM_HOME/$ACUSIM_MACHINE/license.dat`. This option is only used when `lm_service_type` = classical.

`lm_log_file` or `lmlog` *(string)*

Full path to the file in which to record licensing check-out and check-in events. When this value is set to _none, the messages are directed to the system log. This option is only used when `lm_service_type` = classical.

`lm_checkout` or `lmco` *(string)*

Full address of the license checkout co-processor, AcuLmco. This process is spawned by the solver on the local machine. When this option is set to _auto, the value is internally changed to

`$ACUSIM_HOME/$ACUSIM_MACHINE/bin/aculmco`. This option is only used when *`lm_service_type`* = classical.

`lm_queue_time` **or** *`lmqt`* *(enumerated)*
: The time, in seconds, to camp on the license queue before abandoning the wait. This option is only used when *`lm_service_type`* = classical.

`remote_shell` **or** *`rsh`* *(string)*
: Remote shell executable.

`line_buff` **or** *`lbuff`* *(boolean)*
: Flush standard output after each line of output.

`verbose` **or** *`v`* *(enumerated)*
: Set the verbose level for printing information to the screen. Each higher verbose level prints more information. If *`verbose`* is set to 0, or less, only warning and error messages are printed. If *`verbose`* is set to 1, basic processing information is printed in addition to warning and error messages. This level is recommended. *`verbose`* levels greater than 1 provide information useful only for debugging.

Examples

For example, given the input file `channel.inp` that was prepared by AcuPrep, the simplest form for running AcuView is

```
acuView -pb channel
```

or alternatively place the option in the configuration file `Acusim.cnf` as

```
problem= channel
```

and invoke AcuView as

```
acuView
```

AcuSolve

Solves problems that were prepared by AcuPrep and AcuView.

Syntax

`acuSolve [options]`

or

`acuSolve-mpi [options]`

Type

AcuSolve Solver Program Command

Description

Once an input file is processed by AcuPrep and AcuView, AcuSolve is executed to run the problem. These are not normally run by themselves, but rather the script AcuRun is used to run these programs.

The program acuSolve-mpi is the MPI-based parallel version of AcuSolve. This program is symbolically linked to AcuSolve.

In the following, the full name of each option is followed by its abbreviated name and its type. For a general description of option specifications, see *Command Line Options and Configuration Files*. See below for more individual option details:

`help` or `h` *(boolean)*
: If set, the program prints a usage message and exits. The usage message includes all available options, their current values and the place where each option is set.

`problem` or `pb` *(string)*
: The name of the problem is specified via this option. This name is used to generate and recognize internal files.

`problem_directory` or `pdir` *(string)*
: The home directory of the problem. This is where the user input files reside. This option is needed by certain parallel processing packages.

`working_directory` or `dir` *(string)*
: All internal files are stored in this directory. This directory does not need to be on the same file system as the input files of the problem.

`file_format` or `fmt` *(enumerated) [binary]*
: Format of the internal files:

ascii	ASCII files
binary	Binary files
hdf	Hierarchical data format (HDF5)

`output_file_format` or `ofmt` *(enumerated) [_auto]*
: Internal output file format.

_auto	Use same as *file_format*
ascii	ASCII files
binary	Binary files
hdf	Hierarchical data format (HDF5)

num_processors or *np* *(enumerated) [1]*
This option specifies the number of processors to be used. For shared-memory parallelization, such as parallel SGI machines, this option should be set to 1, and both the environment variable MP_SET_NUMTHREADS and the option num_threads set to the number of threads. The total number of threads is *num_processors* times num_threads.

num_threads or *nt* *(enumerated) [1]*
This option specifies the number of threads per processor to be used.

executable or *exec* *(string)*
Full address of this executable. The option is needed by certain parallel processing packages.

user_libraries or *libs* *(string)*
Comma-separated list of user libraries. These libraries contain the user-defined functions and are generated by AcuMakeLib or AcuMakeDll.

viewfactor_file_name or *vf* *(string) [_auto]*
View factor file name. If *viewfactor_file_name* is set to _auto, an internal name is used.

fast_restart or *frst* *(boolean) [FALSE]*
Perform fast restart.

fast_restart_run_id or *frstrun* *(integer) [0]*
Run ID for fast restart.

fast_restart_time_step or *frstts* *(integer) [0]*
Time step for fast restart.

projection_eigenvalue_type or *dfltype* *(enumerated) [galerkin]*
Method used to compute the eigenvalues when projection is active in the linear solver.

galerkin	Use galerkin method for eigenvalue computation
harmonic	Use harmonic method for eigenvalue computation
harmonic-transpose	Use harmonic-transpose method for eigenvalue computation
svd	Use svd method for eigenvalue computation
svd-harmonic	Use svd-harmonic method for eigenvalue computation

disperse_damping_factor or *dispdampfactor* *(real) [1.0]*
A damping factor for disperse multi-field and EDEM. This is an advanced option and should rarely be changed.

edem_set_time or *edemsettime* *(real) [0.0]*
Initial time setting for EDEM coupling.

`edem_liquid_molecular_weight` or `edemliqmolwei` *(real) [180.0]*
Molecular weight for EDEM liquid phase.

`edem_liquid_density` or `edemliqdens` *(real) [1000.0]*
Density for EDEM liquid phase.

`edem_liquid_boiling_temperature` or `edemliqboiltemp` *(real) [373.15]*
Boiling temperature for EDEM liquid phase.

`edem_liquid_latent_heat` or `edemliqlatentheat` *(real) [2.264705e6]*
Latent heat for EDEM liquid phase.

`edem_vapor_density` or `edemvapdens` *(real) [0.73]*
Density for EDEM vapor phase.

`edem_liquid_specific_heat` or `edemliqheat` *(real) [4183.0]*
Specific heat for EDEM liquid phase.

`edem_vapor_specific_heat` or `edemvapheat` *(real) [1900.0]*
Specific heat for EDEM vapor phase.

`display_design_topology` or `dispdesigntopo` *(boolean) [FALSE]*
Flag indicating whether or not to display topology velocity magnitude for topology optimization.

`compute_topology_darcy` or `comptopodarcy` *(boolean) [FALSE]*
Flag indicating whether or not to compute a darcy factor value for topology optimization.

`compute_topology_active_constraints` or `comptoactive` *(boolean) [FALSE]*
Flag indicating whether or not to solve only active constraints of topology optimization.

`topology_heat_reference_temperature` or `topoheatreftemp` *(real) [273.15]*
Topology heat reference temperature.

`topology_heat_brinkman` or `topoheatbrinkman` *(real) [1e-6]*
Topology heat Brinkman number.

`topology_heat_peclet` or `topoheatpeclet` *(real) [1000]*
Topology heat Peclet number.

`topology_ramp_mass_constraint` or `toporampmass` *(integer) [0]*
Topology mass constraints for ramp cases.

`discontinuity_capturing_type_topology_linear` or `dctai` *(boolean) [TRUE]*
Flag indicating whether or not to activate the topology linear discontinuity capturing type.

`compute_topology_darcy` or `comptopodarcy` *(boolean) [FALSE]*
Flag indicating whether or not to compute a darcy factor value for topology optimization.

`linear_solver_error_output` or `lesouterr` *(boolean) [FALSE]*
Flag indicating whether or not to output diagnostic information to a disk when an error is encountered in the linear solver. This is an advanced debugging option and is rarely used.

`gls_diffusion` or `diff` *(boolean) [TRUE]*
Activate the diffusion term in the least squares operator. This is an advanced option and should rarely be changed.

`gls_source` or `lssrc` *(boolean) [TRUE]*
: Include the source term in the weighting side of the least squares operator. This is an advanced option and should rarely be changed.

`porosity_gls_factor` or `plsfct` *(real) [1.0]*
: A factor multiplying the porosity GLS source term. This is an advanced option and should rarely be changed.

`gls_lhs_cross_factor` or `lcfct` *(real) [0.5]*
: A factor multiplying certain pressure-velocity coupling terms in the LHS. This is an advanced option and should rarely be changed.

`tau_factors` or `taufcts` *(string) [_none]*
: GLS tau coefficient factors in the format "inertia, convection, diffusion, source, tau_cont, tau_cont_inertia." If _none, the default tau factors are used. This is an advanced option and should rarely be changed.

`max_principal_factor` or `mpfct` *(real) [0.5]*
: Advective maximum principal factor. This is an advanced option and should rarely be changed.

`non_newtonian_lhs_factor` or `nnlfct` *(real) [0.5]*
: Factor scaling non-Newtonian linearization of LHS. This is an advanced option and should rarely be changed.

`enhanced_strain_type` or `est` *(integer) [1]*
: Enhanced strain type ID.

`multipoint_constraint_tolerance` or `mbctol` *(real) [0.001]*
: Tolerance for multi-point constraint boundary condition (mbc). This is an advanced option and should rarely be changed.

`temporary_memory_allocation_factor` or `tmpmfct` *(real) [1.0]*
: Memory allocation factor (> 1) for temporary data. This is an advanced option and should rarely be changed.

`ssi_l2_factors` or `ssil2fcts` *(string)*
: Coefficients used to optimize the quadrature weights in the surface-surface interface formulation. The format is "area, x, , reg, tol, iters." This is an advanced option and should rarely be changed.

`ssi_memory_allocation_factor` or `ssimfct` *(real) [1.0]*
: Memory over-allocation factor (> 1) for search. This is an advanced option and should rarely be changed.

`ssi_tau_factors` or `ssitaufcts` *(string)*
: SSI tau factors for momentum flux in the surface-surface interface formulation. The format is "mass,viscous." This is an advanced option and should rarely be changed.

`ssi_computation_type` or `ssicomptype` *(enumerated) [new]*
: Algorithm used to compute the flow across non-conformal interfaces.

legacy	Legacy non-conformal interface treatment. This option reproduces the behavior of releases 14.0 and older.

new	Improved algorithm released in version 2017. This option is now default.

`back_flow_diffusion_factor` or `bfdfct` *(real) [1.0]*
: Back flow diffusion factor at outflow boundaries. This is an advanced option and should rarely be changed.

`sst_filter_tolerance` or `ssttol` *(real) [0.0]*
: Residual tolerance above which the turbulence variables are smoothed to improve stability. This is an advanced option and should rarely be changed.

`number_turbulence_film_filter` or `ntff` *(integer) [0]*
: Number of filter pass for film coefficient (HTC) and reference temperature.

`number_heat_flux_filter` or `nhff` *(integer) [0]*
: Number of filter pass for surface heat flux in turbulence flows.

`wall_function_slip_velocity_factor` or `twfvfct` *(real) [0.0]*
: Slip velocity correction factor for turbulence wall functions.

`wall_function_spalding` or `twfsp` *(boolean) [TRUE]*
: Flag to specify the use of the Spalding function for the eddy viscosity wall function when using Spalart-Allmaras based turbulence models. This option was introduced in 13.0 and is active by default. To reproduce the behavior of the wall functions in versions prior to 13.0, this option should be turned off.

`mesh_poisson_ratio` or `alenu` *(real) [0.0]*
: ALE mesh movement Poisson ratio. This is an advanced option and should rarely be changed. Used only when `mesh_technology` = neohookean, ogden or foam.

`mesh_reference_push_factor` or `alefct` *(real) [0.0]*
: ALE mesh movement reference-position forward push factor. This is an advanced option and should rarely be changed. Used only when `mesh_technology` = neohookean, ogden or foam.

`mesh_volume_absorption_factor` or `alevaf` *(real) [0.0]*
: ALE mesh element volume absorption factor. This option controls the stiffness of volume elements when performing moving mesh simulations using the ALE approach. When this factor is set greater than 1.0, small elements are assigned a higher stiffness and large elements absorb more deformation. Internal testing suggests that a value of 2.0-2.5 works well for most applications. Used only when `mesh_technology` = neohookean, ogden or foam.

`mqm_mesh_reference_push_factor` or `mqmpush` *(real) [0.0]*
: ALE mesh movement reference-position forward push factor. This option controls whether the original coordinates are used as the reference coordinates for evaluating the quality of the element or whether the deformed coordinates from the previous step are used. To ensure that the mesh returns to its original state after the displacement of the boundaries is returned to zero, this option must be set to 0.0. In general, this also produces a better quality deformed mesh, but at the expense of a stiffer linear system. Used only when `mesh_technology`=metric.

`num_mmi_smoothing_passes` or `mmipass` *(integer) [1]*
: The number of mesh smoothing passes to apply to the nodal coordinates after the mesh is deformed using the interpolated mesh motion approach. Mesh smoothing improves the quality of

the mesh substantially and it is recommended that this option is always active. Used only when a `MESH_MOTION` command of *type*=interpolated_motion is present within the input file.

num_free_surface_smoothing_passes or *fspass* *(integer) [100]*
: Number of passes taken to smooth the free surface, which maintaining volume conservation. The smoothing is performed at every ALE nonlinear iteration. This is an advanced option and should rarely be changed.

free_surface_compact_support or *fscs* *(real) [1.0]*
: Extent to which the free surface mass flux is distributed in order to smooth the free surface. A value of 1 redistributes the mass flux fully within a free surface facet. A value beyond 1 will cross the surface facet boundary. This is an advanced option and should rarely be changed.

free_surface_regularization_factor or *fsreg* *(real)*
: Free surface smoothing regularization factor. This is an advanced option and should rarely be changed.

free_surface_gravity_factor or *fsgf* *(real) [0.1]*
: Free surface gravity factor in flow equation. This is an advanced option and should rarely be changed.

free_surface_viscous_factor or *fsvf* *(real) [0.5]*
: Free surface viscosity factor in flow equation. This is an advanced option and should rarely be changed.

non_reflecting_bc_running_average_field or *nrbcraf* *(boolean) [TRUE]*
: Enhance the effectiveness of the non-reflecting boundary condition through the use of running average field variables. The *running_average* in the `EQUATION` command needs to be on when *flow* = navier_stokes. It is not necessary for *flow* = compressible_navier_stokes.

les_bfgs_acceleration or *lesbfgs* *(boolean) [FALSE]*
: BFGS acceleration for the linear algebra. This is an advanced option and should rarely be changed.

radiation_linearization_level or *radlin* *(integer) [4]*
: Linearization level of enclosure radiation in the energy equation (0,...,4). This is an advanced option and should rarely be changed.

radiation_max_principal_factor or *rmpfct* *(real) [1.0]*
: Enclosure radiation maximum principal factor. This is an advanced option and should rarely be changed.

amg_memory_allocation_factor or *amgmfct* *(real) [1.0]*
: Algebraic multigrid memory allocation factor (> 1). This is an advanced option and should rarely be changed.

external_file_wait_time or *exft* *(real) [600]*
: How long to wait for the external file to appear.

external_output_host or *eohost* *(string) [_auto]*
: External output host name for establishing socket connection. Used with the `EXTERNAL_OUTPUT` input file command.

`external_output_port` or *`eoport`* *(integer) [0]*
: External output port number for establishing socket connection. Used with the `EXTERNAL_OUTPUT` input file command.

`external_output_wait_time` or *`eowait`* *(real) [3600.0]*
: How long to wait for the external output code to establish socket connection. Used with the `EXTERNAL_OUTPUT` input file command.

`particle_trace_host` or *`pthost`* *(string) [_auto]*
: Particle trace host name for establishing socket connection. Used with runtime coupling of AcuTrace.

`particle_trace_port` or *`ptport`* *(integer) [0]*
: Particle trace port number for establishing socket connection. Used with runtime coupling of AcuTrace.

`particle_trace_wait_time` or *`ptwait`* *(real) [3600.0]*
: How long to wait for the particle trace code to establish socket connection. Used with runtime coupling of AcuTrace.

`external_code_host` or *`echost`* *(string) [_auto]*
: External code host name for establishing socket connection.

`external_code_port` or *`ecport`* *(integer) [0]*
: External code port number for establishing socket connection.

`external_code_wait_time` or *`ecwait`* *(real) [3600.0]*
: How long to wait for the external code to establish socket connection.

`external_code_echo` or *`ececho`* *(boolean) [FALSE]*
: Echo messages received from external code for stand-alone debugging.

`external_code_force_type` or *`ecft`* *(enumerated) [css]*
: Type of external code force computation:

css	Convential sequential staggered
gss	Higher-order displacement extrapolations
exp	Exp

`external_code_force_offset` or *`ecfo`* *(enumerated) [none]*
: External code force computation offset option:

none	No offset
disp	Offset based on displacement
force	Offset based on force
Offset based on force	Offset based on both force and displacement

`external_code_heat_type` or *`echt`* *(enumerated) [css]*
: Type of external code heat flux computation:

css	Convential sequential staggered

lm_daemon or *lmd* *(string) [_auto]*

Full address of the network license manager daemon, AcuLmd, that runs on the server host. When this option is set to _auto, the value is internally changed to `$ACUSIM_HOME/$ACUSIM_MACHINE/bin/acuLmd`. This option is only used when *lm_service_type* = classical.

lm_service_type or lmtype (enumerated) [hwu]

Type of the license manager Type of the license manager service:

hwu	Altair Units licensing
token	Token based licensing
classical	Classical Acusim style licensing

lm_server_host or *lmhost* *(string)*

Name of the server machine on which the network license manager runs. When set to _auto, the local host name is used. This option is only used when *lm_service_type* = classical.

lm_port or *lmport* *(integer) [41994]*

TCP port number that the network license manager uses for communication. This option is only used when *lm_service_type* = classical.

lm_license_file or *lmfile* *(string) [_auto]*

Full address of the license file. This file is read frequently by the network license manager. When this option is set to _auto, the value is internally changed to `$ACUSIM_HOME/$ACUSIM_MACHINE/license.dat`. This option is only used when *lm_service_type* = classical.

lm_log_file or *lmlog* *(string)*

Full path to the file in which to record licensing check-out and check-in events. When this value is set to _none, the messages are directed to the system log. This option is only used when *lm_service_type* = classical.

lm_checkout or *lmco* *(string) [_auto]*

Full address of the license checkout co-processor, AcuLmco. This process is spawned by the solver on the local machine. When this option is set to _auto, the value is internally changed to `$ACUSIM_HOME/$ACUSIM_MACHINE/bin/acuLmco`. This option is only used when *lm_service_type* = classical.

lm_queue_time or *lmqt* *(integer) [3600]*

The time, in seconds, to camp on the license queue before abandoning the wait. This option is only used when *lm_service_type* = classical.

remote_shell or *rsh* *(string)*

Remote shell executable.

exchange_synchronized_send or *excssend* *(boolean) [FALSE]*

Flag to use synchoronized send communication.

exchange_sort_receive or *excsrecv* *(boolean) [TRUE]*

Flag to sort received messeages. Ensures repeatability of runs by enforcing the order of received messages. May increase exchange time, however.

`exchange_trace` or *`exctrace`* *(boolean) [FALSE]*
: Flag to ouput data for communication tracing from a parallel run. This is an advanced profiling option and should rarely be used.

`collect_subdomain_output` or *`csout`* *(boolean) [TRUE]*
: Collect the nodal and surface nodal output files, that is, `ACUSIM.DIR/*.osf.B`, across the subdomains to reduce the number of generated files. Newer versions of the post-processing tools must be used if this option is turned on.

`line_buff` or *`lbuff`* *(boolean) [FALSE]*
: Flush standard output after each line of output.

`sync_disk` or *`sync`* *(boolean) [FALSE]*
: Periodically sync the disks.

`print_environment_variables` or *`printenv`* *(boolean) [FALSE]*
: Print all environment variables.

`verbose` or *`v`* *(integer)*
: Set the verbose level for printing information to the screen. Each higher verbose level prints more information. If *`verbose`* is set to 0 (or less), only warning and error messages are printed. If *`verbose`* is set to 1, basic processing information is printed in addition to warning and error messages. This level is recommended. *`verbose`* levels greater than 1 provide information useful only for debugging.

Examples

Having a pre-processed user problem channel, AcuSolve may be executed (in its simplest form) as

```
acuSolve -pb channel
```

or alternatively place the option in the configuration file `Acusim.cnf` as:

```
problem= channel
```

and invoke AcuSolve as:

```
acuSolve
```

A fast restart may be performed by specifying -fast_restart. AcuSolve will start from a solution other than the latest if either -fast_restart_run_id or -fast_restart_time_step (or both) is specified. The previous run and history files will be extended instead of creating new ones.

The options lm_daemon, lm_server_host, lm_port, lm_license_file, lm_checkout, and lm_queue_time are used for checking out a license from a network license manager.

There are four options that control how AcuSolve interacts with an external code for Direct Coupling Fluid Structural Interaction problems. The option *`external_code_host`* specifies a host name for establishing the socket connection and overwrites the *`socket_host`* parameter in the `EXTERNAL_CODE` command. Similarly, *`external_code_port`* specifies the port for the socket connection and overwrites *`socket_port`* in `EXTERNAL_CODE`. The maximum amount of time to wait for the external code connection is given by *`external_code_wait`*. The option *`external_code_echo`* echos the messages received by the external code into a .ecd file. This file may subsequently be given as the value of *fifo_receive_file* of the EXTERNAL_CODE command with fifo type to perform a stand-alone run of AcuSolve.

The *wall_function_slip_velocity_factor* option allows fine tuning of the turbulence wall function. The wall function is unchanged when this parameter is 0. For a value of 1, the surface velocity is slipped such that the mass flux is consistent with the law of the wall. In between values may be used to better capture the velocity profile by setting the wall velocity to a linear interpolation of these limit cases. This is particularly useful for meshes with very large y+, such as 1000 or larger.

Turning on *non_reflecting_bc_running_average_field* specifies the use of running average fields to enhance the performance of non-reflective boundary conditions.

By default, AcuSolve writes a separate file for each subdomain and each nodal and surface output command. This may result in numerous files being written to ACUSIM.DIR. For example surface output produces files of the form ACUSIM.DIR/*.osf.B. If *collect_subdomain_output* is turned on, data for all subdomains are collected and written to a single "merged" file for each type of output command. This applies to the nodal data files of NODAL_OUTPUT, RESTART_OUTPUT, DERIVED_QUANTITY_OUTPUT, ERROR_ESTIMATOR_OUTPUT, RUNNING_AVERAGE_OUTPUT, and NODAL_RESIDUAL_OUTPUT commands, and the surface data files of SURFACE_OUTPUT and RADIATION_SURFACE. This option will result in an increase in elapsed time running AcuSolve because output is written to disk serially.

The parameter *file_format* determines if the input and output files are saved in ASCII, binary, or HDF5 format. The extension .B on the file name parameters is optional. It will be added or removed automatically based on the value of *file_format*. The HDF5 format reduces and organizes the storage of number of files. It also improves I/O efficiency and performance by avoiding opening and closing many small files. The file extension of a standalone HDF5 file is ".H" and the for the container file is ".C". A single container file can be opened for the output, say channel.1_0.ohd.C, which will contain all the output files for a single subdomain.

The disk may be periodically synced by turning on *sync_disk*. This is useful when another code is used to read data immediately after AcuSolve has finished writing it. Otherwise, on some systems data may be not fully available since writing does not properly sync the disk.

AcuSolve will print all the environment variable it sees before starting to run when *print_environment_variables* is turned on. This is primarily useful for debugging parallel processing problems using MPI.

All other options are advanced options and should rarely be changed from their default values.

AcuSig

Signal AcuSolve to perform certain actions while it is executing.

Syntax

`acuSig [options]`

Type

AcuSolve Solver Program Command

Description

AcuSig is used to send a signal to AcuSolve while the latter is executing in order to change its behavior.

In the following, the full name of each option is followed by its abbreviated name and its type. For a general description of option specifications, see *Command Line Options and Configuration Files*. See below for more individual option details:

`help` or *`h`* *(boolean)*
: If set, the program prints a usage message and exits. The usage message includes all available options, their current values, and the place where each option is set.

`problem` or *`pb`* *(string)*
: The name of the problem is specified via this option. This name is used to generate and recognize internal files.

`problem_directory` or *`pdir`* *(string)*
: The home directory of the problem. This is where the user input files reside. This option allows AcuSig to be executed from any directory and still affect the problem in the given directory. If *`problem_directory`* is set to ".", namely the Unix current directory, it is replaced by the full address of the current directory, as provided by the Unix getcwd command.

`working_directory` or *`dir`* *(string))*
: All internal files are stored in this directory. This directory does not need to be on the same file system as the input files of the problem.

`run_id` or *`run`* *(integer)*
: Run number the signal applies to. If 0, the latest run is used.

`stop` *(boolean)*
: Signal AcuSolve to stop at the end of the current time step.

`stop_time_step` or *`sts`* *(integer)*
: Signal AcuSolve to stop at the end of time step *`sts`*. -1 indicates do not stop.

`halt` *(boolean)*
: Signal AcuSolve to stop as soon as possible.

`output` or *`out`* *(boolean)*
: Signal AcuSolve to write nodal output to disk at the end of the current time step.

`output_time_steps` or *`ots`* *(string)*
: Comma separated list of time steps to be output.

output_times or *otv* *(string)*
Comma separated list of times to be output.

force_output_restart or *rst* *(boolean)*
Signal AcuSolve to write restart output to disk at the end of the current time step.

output_residual or *res* *(boolean)*
Signal AcuSolve to write nodal residual to disk at the end of the current time step.

output_projection_vecs or *pvout* *(boolean)*
Signal AcuSolve to output the nodal projection vectors at the end of the current time step.

not_converged or *nc* *(boolean)*
Signal AcuSolve the current time step is not converged.

do_not_amp_timeinc or *nati* *(boolean)*
Signal AcuSolve to not increase the time increment of this step.

reduce_timeinc or *rti* *(boolean)*
Signal AcuSolve to reduce the time increment of the next time step.

redo_step or *rts* *(boolean)*
Signal AcuSolve to redo the current time step with a smaller time increment if possible.

must_redo_step or *mrts* *(boolean)*
Signal AcuSolve to redo the current time step with a smaller time increment, or exit if not possible.

flush *(boolean)*
Signal AcuSolve to flush the `.log` file buffer.

line_buff or *lbuff* *(boolean)*
Signal AcuSolve to flush the `.log` file buffer for each line.

set_verbose or *sv* *(integer)*
Set the solver verbose level for printing information to the screen. Each higher verbose level prints more information. If *set_verbose* is set to 0, or less, only solver warning and error messages are printed. If *set_verbose* is set to 1, basic processing information is printed in addition to warning and error messages. This level is recommended. *set_verbose* levels greater than 1 provide additional information useful only for debugging.

user_global_data or *data* *(string)*
Send a name=value message to AcuSolve.

verbose or *v* *(integer)*
Set the verbose level for printing information to the screen. Each higher verbose level prints more information. If *verbose* is set to 0, or less, only warning and error messages are printed. If *verbose* is set to 1, basic processing information is printed in addition to warning and error messages. This level is recommended. *verbose* levels greater than 1 provide information useful only for debugging.

Examples

The following command will cause AcuSolve to terminate normally at the end of the current time step, including performing all the output expected at the end of a run:

```
acuSig -pb channel -stop
```

or alternatively if the configuration file `Acusim.cnf` contains

```
problem = channel
```

then AcuSig may be invoked as:

```
acuSig -stop
```

Normally, AcuSig is issued in the problem directory. However, AcuSig may be issued from any directory provided that problem directory is given on the command line. For example,

```
acuSig -stop -pdir $ACUSIM_HOME/HP/latest/examples/channel
```

To signal AcuSolve to stop as above but as soon as possible, execute:

```
acuSig -halt
```

To signal AcuSolve to write out nodal output at the end of the current step, execute:

```
acuSig -out
```

To signal AcuSolve to write out nodal output at the end of several specific time steps, execute:

```
acuSig -ots "18,20,22"
```

For the three commands below, AcuSolve will continue to execute normally after the current step. However, they all may be combined with -stop to cause AcuSolve to stop after the output is written.

To signal AcuSolve to write out nodal output at several specific times, actual output is at the end of the time step containing each time, execute:

```
acuSig -otv "3.5,4.0,4.5"
```

To signal AcuSolve to write out restart output at the end of the current step, execute:

```
acuSig -rst
```

To signal AcuSolve to write out nodal residual output at the end of the current step, execute:

```
acuSig -res
```

Normally the `.log` file is flushed at the end of every step. To flush it immediately, but just once, execute:

```
acuSig -flush
```

To cause the `.log` file buffer to be flushed for every line, execute:

```
acuSig -lbuff
```

To return to the normal behavior of flushing the `.log` file buffer at the end of every step:

```
acuSig -no_lbuff
```

To change the verbose level of the solver so that it will output basic processing information in addition to warning and error messages, execute:

```
acuSig -sv 1
```

A value for the user global data var 1 may be passed to a running AcuSolve job by issuing the command:

```
acuSig -data "var 1 = 332"
```

The value takes effect at the beginning of the next time step.

Preparatory Programs 5

This chapter contains auxiliary programs that help to prepare input files for running AcuSolve.

This chapter covers the following:

AcuImport

Imports a mesh or problem from an external format.

Syntax

`acuImport [options]`

Type

AcuSolve Preparatory Program

Description

AcuImport is a utility program that translates a problem and/or a mesh from a third party format and writes it on disk in a format suitable for running AcuRun.

In the following, the full name of each option is followed by its abbreviated name and its type. For a general description of option specifications, see *Command Line Options and Configuration Files*. See below for more individual option details:

`help` or `h` (boolean)
: If set, the program prints a usage message and exits. The usage message includes all available options, their current values, and the place where each option is set.

`problem` or `pb` (string)
: The name of the problem is specified via this option. This name is used to generate input file names.

`file_format` or `fmt` (enumerated)
: Export the mesh in this format:

ascii	ASCII text format.
binary	Binary format.

`import_from` or `from` (enumerated)
: Import from this format:

ANSYS	Import from ANSYS database.
Fluent	Import from Fluent/UNS `.cas` file.
ideas	Import from I-deas universal file.
starcd	Import from StarCD format.

`import_file_name` or `file` (string)
: The name of the file from which the problem and/or mesh is to be imported.

`mesh_directory` or `mdir` (string)
: The name of the directory to place the mesh files.

`split_as_tets` or `tet` *(boolean)*
: Convert the imported mesh to an all-tetrahedron AcuSolve mesh. Currently supported only with -from fluent.

`coordinates_scaling_factor` or `crdfct` *(real)*
: Scale the coordinates by this factor.

`line_buff` or `lbuff` *(boolean)*
: Flush standard output after each line of output.

`verbose` or `v` *(integer)*
: Set the verbose level for printing information to the screen. Each higher verbose level prints more information. If `verbose` is set to 0 (or less), only warning and error messages are printed. If `verbose` is set to 1, basic processing information is printed in addition to warning and error messages. This level is recommended. `verbose` levels greater than 1 provide information useful only for debugging.

Examples

To import the Fluent/UNS `.cas` file -`channel.cas`, issue the command:

```
acuImport -pb channel -from fluent -file channel.cas
```

Alternatively place the options in the configuration file `Acusim.cnf` as:

```
problem = channel
import_from = fluent
import_file_name = channel.cas
```

and issue the command:

```
acuImport
```

The above extracts the mesh data from the Fluent/UNS file `channel.cas`, and writes the input file -`channel.inp` plus a number of supporting geometry and boundary condition files. The input can then be edited to incorporate additional data, and subsequently run by AcuRun.

The imported coordinates may be scaled by the factor given by -coordinates_scaling_factor. For example, if the coordinates in a Fluent mesh are given in inches, but an AcuSolve mesh is needed in meters, the following command will do the conversion:

```
acuImport -pb channel -from fluent -file channel.cas -crdfct 0.0254
```

Input files from the following formats are imported by this program: Fluent, I-deas and ANSYS. Details of the first two are given below.

Fluent Format

For Fluent format, AcuImport imports a Fluent/UNS `.cas` file from Fluent Inc. The UNS data sets 10, 12, 13, 2010, 2012, and 2013 are read in their entirety, data set 39 is read to extract names, and data set 37 is read to extract some global parameters. These data sets are converted to coordinate file `problem.crd`; connectivity files `problem.etag.cnn`, where `etag` is the set identification; surface connectivity files `problem.etag.stag.ebc`, where `stag` is the surface set identification; and nodal boundary condition files `problem.etag.stag.nbc`, which are the nodes of the surface files. It also generates an input file `problem.inp`, where these files are referenced by their appropriate input commands. This input file must be edited to incorporate the appropriate boundary condition values, material models, and so on.

All Fluent meshes, including mixed element types, may be imported by AcuImport. Optionally, the imported mesh may be converted to an all-tetrahedron mesh by specifying -split_as_tets.

I-deas Format

For I-deas format, AcuImport imports an I-deas Universal file, from Structural Dynamics Research Corporation. The following data set are translated:

- Dataset 164: Units
- Dataset 790: Load Sets
- Dataset 791: Restraint Sets
- Dataset 2411: Nodes - Double Precision
- Dataset 2412: Elements
- Dataset 2420: Coordinate System

Datasets 164 and 2420 provide the coordinate transformation, which is used to transform the nodal coordinates and nodal boundary conditions to the global coordinate system. The coordinate system of dataset 2420 must be a Cartesian system, as specified by a 0 value for Record 3 Field 2 (coordinate system type).

Dataset 2411 provides the nodal coordinates which are transformed to global coordinate system and written to the `problem.crd` file.

Dataset 2412 provides the elements. The elements are separated into element sets based on element shape and material ID. The elements are transferred to files `problem.tag.mid.cnn`; where tag is one of the element shapes tet4, wedge6, or hex8, as determine from Record 1 Field 1 (fe descriptor ID). Only values 111, 112, and 115 are allowed. mid is the material ID given on Record 1 Field 4 (material property table number).

Dataset 791 provides the nodal boundary conditions. Record 1 Field 2 (restraint type) must be either 1 (nodal displacement) or 2 (nodal temperature). Type 1 translates into boundary conditions on a vector field, and type 2 translates into a boundary condition on a scalar field. The boundary condition variable is extracted from the first token of Record 2 (restraint set name). The (lower/upper) case of this token is ignored. For example, if Record 2 is "velocity at inflow", a velocity boundary condition is assumed. The entire record is used for the user given name of the `NODAL_BOUNDARY_CONDITION` command. For a scalar field, the boundary condition value is extracted from Record 4 Field 1 (temperature value). For a vector field, Fields 3, 4, and 5 of Record 3 (switches for physical dofs) specify whether or not a vector component (in the coordinate system given by dataset 2420) is specified or not. The boundary condition values are extracted from Record 4 Fields 1, 2, and 3 (displacement for dof 1, 2, and 3).

Dataset 790 provides the element boundary conditions, surface outputs, and turbulence wall surfaces. Record 1 Field 2 (load type) must be 2 (finite element face pressure). The boundary condition variable is extracted from the first token of Record 2 (load set name). The (upper/lower) case of this token is ignored. If this token is surface_output, then the set is translated to a `SURFACE_OUTPUT` command. If this token is turbulence_wall, the set is translated to a `TURBULENCE_WALL` command. Otherwise the set is translated to an `ELEMENT_BOUNDARY_CONDITION` command. In any of these cases, the user given name of the set is the entire string of Record 2. For example, if Record 2 is "pressure on the outflow", a pressure boundary condition is assumed. In all cases, surface files `problem.tag.ebc.set.ebc` are generated; where `tag` is either tri3 or quad4; `ebc` is a sequential count of sets; and `set` is extracted from Record 1 Field 1 (load set number). The boundary condition value is extracted from Record 4 Field 1 (pressure value on face). If Record 3 Field 1 (face pressure load label) is negative, and the set is translated to an element boundary condition, then the boundary condition of type free is assumed.

The directory where the resulting AcuSolve mesh files are placed can be changed. For example, to place these files in the directory `mesh.out` use the command:

```
acuImport -mdir mesh.out
```

AcuSurf

Extract external surfaces of a mesh, for imposing boundary conditions.

Syntax

`acuSurf [options]`

Type

AcuSolve Preparatory Program

Description

AcuSurf is a simple utility program that extracts and organizes all external surfaces of a mesh. The generated surface files may be used in the input file for element boundary conditions, surface output, and so on.

The surfaces are collected based on their parent connectivity file, surface shape, and neighboring angles. The collections are written in files `problem.srfindex`; where `index` is an index starting from 1 to the number of surface sets.

Note: Disconnected surfaces are stored in different files.

The algorithm used by AcuSurf is as follows. It first traverses through all element surfaces and collects only those surfaces with elements on one side. These surfaces are the external or boundary surfaces. All surfaces that have elements on both sides are internal surfaces, and hence are not considered.

The algorithm then traverses through the surfaces given in `surface_file_list` files. Each boundary surface found in these files is tagged as it is read.

It then traverses through the "partial" surfaces given in `partial_surface_file_list` files. These files must contain surfaces without the parent element numbers. Similar to the `surface_file_list` case, the boundary surfaces found in these files are tagged as they are read. In addition, the surfaces including the parent element numbers are written in the surface files `file.index` where `file` is the original file name; and `index` is the index of the parent connectivity file starting from 1 in the order specified in `connectivity_file_list`.

It then traverses through the surface node sets given in `surface_node_file_list` files. These files must contain the list of nodes on the boundary. For each file, the nodes of each boundary surface are checked against the node list. If all surface nodes are present, the surface is tagged as it is read and is written into the file `file.index_shape`, where `file` is the node set file name, `index` is the index of the parent connectivity file starting from 1 in the order specified in `connectivity_file_list`, and shape is either tri3, quad4, or tri6.

Finally, if *`auto_generate_surfaces`* flag is set, it randomly selects one surface from the non-tagged boundary surface, which starts a new surface set. It then recursively traverses through the neighbors of the selected surfaces. If a neighbor has the same surface shape, that is, 3-node triangle, 4-node quadrilateral, or 6-node triangle, comes from the same connectivity file, and its outward normal direction has an angle less than or equal to max_angle, the neighbor is added to the surface set. If no other surface qualifies to be added to the surface set, then a new non-tagged surface is chosen and the process is repeated.

Choice of the *max_angle* can significantly affect the collection process. A *max_angle* of 0 only admits elements that fall exactly on a plane, whereas a *max_angle* of 180 gives one surface set for each connected set of surfaces with the same shape and parent connectivity file.

This utility also determines whether or not the mesh is constructed by a two dimensional mesh extracted into a third dimension. If so, it determines the axis of extrusion. Any external surface that is perpendicular to this axis is ignored, unless -no_c2d is set, in which case no external surface is ignored. For convenience, AcuSurf also determines the extruded node pairs, which are output in the file `problem.pbc`. This file is suitable for the nodes parameter of the `PERIODIC_BOUNDARY_CONDITION` command.

In the following, the full name of each option is followed by its abbreviated name and its type. For a general description of option specifications, see *Command Line Options and Configuration Files*. See below for more individual option details:

help or *h* *(boolean)*
: If set, the program prints a usage message and exits. The usage message includes all available options, their current values, and the place where each option is set.

problem or *pb* *(string)*
: The name of the problem is specified via this option. This name is used to generate input file names and extracted surface file names.

coordinates_file or *crd* *(string)*
: Nodal coordinates file name. The file must contain four columns: node number and x, y, z coordinates. If *coordinates_file* is set to _auto, `problem.crd` and `problem.crd.B` are assumed.

connectivity_file_list or *cnn* *(string)*
: Comma-separated list of connectivity files. The shape of the elements is extracted from the number of columns in each file. The elements are assumed to be 4-node tetrahedra if the file has 5 columns; 5-node pyramids if 6 columns; 6-node wedges if 7 columns; 8-node bricks if 9 columns; and 10-node tetrahedra if 11 columns. If *connectivity_file_list* is set to _auto, all `problem*.cnn` and `problem*.cnn.B` files are used.

surface_file_list or *srf* *(string)*
: Comma-separated list of surface files. The shape of the surfaces is extracted from the number of columns in each file. The surfaces are assumed to be 3-node triangles if the file has 5 columns; 4-node quads if 6 columns; and 6-node triangles if 8 columns. The surfaces in these files are removed from the generated surface files. If *surface_file_list* is set to _none, no surface file is read. If *surface_file_list* is set to _auto, then `problem*.srf`, `problem*.srf.B`, `problem*.ebc`, and `problem*.ebc.B` are used.

partial_surface_file_list or *psrf* *(string)*
: Comma-separated list of surface files, without the parent element numbers. The shape of the surfaces is extracted from the number of columns in each file. The surfaces are assumed to be 3-node triangles if the file has 4 columns; 4-node quads if 5 columns; and 6-node triangles if 7 columns. The boundary surfaces of each file are written in files `file.index`, where `file` is the original file name; and `index` is the index of the parent connectivity file starting from 1 in the order specified in *connectivity_file_list*. The generated files contain the parent element

numbers, which are needed by all ACUSIM programs. All surfaces in these files are removed from the generated surface files. If *partial_surface_file_list* is set to _none, no surface file is read.

surface_node_file_list or *node* *(string)*
: Comma-separated list of surface node files. The boundary surfaces of each connectivity file are searched for the nodes in each of these files. If any boundary surface is found, a surface file is created with the name `file.index_shape`, where `file` is the node file name; `index` is the index of the parent connectivity file starting from 1 in the order specified in *connectivity_file_list*; and `shape` is either tri3, quad4, or tri6. All extracted surfaces are removed from the generated surface files. If *surface_node_file_list* is set to _none, no node file is read. If *surface_node_file_list* is set to _auto, then `problem*.nbc` and `problem*.nbc.B` are used.

auto_generate_surfaces or *asrf* *(boolean)*
: Output boundary surfaces that are not processed due to parameters *surface_file_list*, *partial_surface_file_list*, and *surface_node_file_list*.

max_angle or *angle* *(real)*
: This option specifies the maximum angle allowed between two adjacent surfaces in order to be considered as part of the same surface set. *max_angle* must be between 0 and 180 degrees. If *max_angle* is set to 0, surfaces that fall on a flat plane, and have the same shape and parent connectivity file are collected together. If *max_angle* is set to 180, all connected surfaces of the same shape and parent connectivity are collected together.

output_surface_nodes or *onod* *(boolean)*
: Generate a nodal list for every surface file. The nodes of the surfaces are written in `surface_file.nod`; where *surface_file* is the name of the surface file.

check_2d_mesh or *c2d* *(boolean)*
: Do not generate surfaces for the 2D planes of a 2D mesh.

verbose or *v* *(integer)*
: Set the verbose level for printing information to the screen. Each higher verbose level prints more information. If *verbose* is set to 0, or less, only warning and error messages are printed. If *verbose* is set to 1, basic processing information is printed in addition to warning and error messages. This level is recommended. *verbose* levels greater than 1 provide information useful only for debugging.

Examples

Consider the mesh of the channel problem. AcuSurf may be executed as:

```
acuSurf -pb channel -angle 10
```

or alternatively the options may be put into the configuration file `Acusim.cnf` as follows:

```
problem = channel
max_angle = 10
```

and AcuSurf invoked as

```
acuSurf
```

The above example produces four surface sets, stored in files `channel.srf1` through `channel.srf4`, and four node sets, stored in files `channel.srf1.nod` through `channel.srf4.nod`. Whereas,

```
acuSurf -pb channel -angle 91
```

produces a single surface file `channel.srf1` containing all the surfaces, plus its nodes in the – `channel.srf1.nod` file.

In another example, assume you have the inflow surface file `channel.inflow.srf`, then:

```
acuSurf -pb channel -srf channel.inflow.srf
```

produces the files `channel.srf1` through `channel.srf3` (plus their nodal files). These files do not contain any of the surfaces in `channel.inflow.srf`. The file `channel.inflow.srf.nod` is also generated.

In another example, assume you have the inflow surfaces without the parent element number in the partial file `channel.inflow.psrf`, then

```
acuSurf -pb channel -psrf channel.inflow.psrf
```

produces the files `channel.srf1` through `channel.srf3`, the file `channel.inflow.psrf.1`, which is the inflow surface file `channel.inflow.psrf` including the parent element number for the first connectivity file, plus the nodal files `channel.srf1.nod` through `channel.inflow.psrf.1.nod`.

In another example, assume you have a list of inflow nodes in the file `channel.inflow.nodes`, then

```
acuSurf -pb channel -node channel.inflow.nodes
```

produces the files `channel.srf` through `channel.srf3`, `channel.inflow.nodes.1_quad4` containing the inflow surfaces for the first connectivity file, plus the nodal files.

To prevent the nodal files to be generated, issue the command with -no_onod option, or place

```
output_surface_nodes= FALSE
```

in the `Acusim.cnf` configuration file.

AcuMesh2Tet

Convert a mesh into an all-tetrahedra mesh.

Syntax

`acuMesh2Tet [options]`

Type

AcuSolve Preparatory Program

Description

Since tetrahedra are much more efficient in terms of CPU and memory usage than other element topologies, and have nearly the same accuracy, it is often very advantageous to convert pyramids, wedges, hexes, and high-order tetrahedra into linear (4-node) tetrahedra. AcuMesh2Tet is a simple utility program that does this conversion for both interior and surface elements.

For each interior connectivity file `file`, AcuMesh2Tet produces a tetrahedra file `file.tet`. Similarly, for each file `srf_file` in the surface file list, a triangle surface file `srf_file.tri` is produced.

It is important to specify any periodic boundary condition files in *`periodic_bc_file_list`* if they exist. They are used to ensure that the resulting mesh is continuous across periodic surfaces.

In the following, the full name of each option is followed by its abbreviated name and its type. For a general description of option specifications, see *Command Line Options and Configuration Files*. See below for more individual option details:

`help` or *`h`* *(boolean)*
: If set, the program prints a usage message and exits. The usage message includes all available options, their current values, and the place where each option is set.

`problem` or *`pb`* *(string)*
: The name of the problem is specified via this option. This name is used to generate input file names and extracted surface file names.

`coordinates_file` or *`crd`* *(string)*
: Nodal coordinates file name. The file must contain four columns: node number and x, y, z coordinates. If *`coordinates_file`* is set to _auto, `problem.crd` and `problem.crd.B` are assumed.

`connectivity_file_list` or *`cnn`* *(string)*
: Comma-separated list of connectivity files. The shape of the elements is extracted from the number of columns in each file. The elements are assumed to be 4-node tetrahedra if the file has 5 columns; 5-node pyramids if 6 columns; 6-node wedges if 7 columns; 8-node bricks if 9 columns; and 10-node tetrahedra if 11 columns. If *`connectivity_file_list`* is set to _auto, all `problem*.cnn` and `problem*.cnn.B` files are used.

`periodic_bc_file_list` or *`pbc`* *(string)*
: Comma-separated list of periodic boundary condition files. If *`periodic_bc_file_list`* is set to _auto, all `problem*.pbc` and `problem*.pbc.B` files are used.

surface_file_list or *srf* *(string)*

Comma-separated list of surface files. The shape of the surfaces is extracted from the number of columns in each file. The surfaces are assumed to be 3-node triangles if the file has 5 columns; 4-node quads if 6 columns; and 6-node triangles if 8 columns. The surfaces in these files are removed from the generated surface files. If *surface_file_list* is set to _none, no surface file is read. If *surface_file_list* is set to _auto, then `problem*.srf`, `problem*.srf.B`, `problem*.ebc`, and `problem*.ebc.B` are used.

verbose or *v* *(integer)*

Set the verbose level for printing information to the screen. Each higher verbose level prints more information. If *verbose* is set to 0 (or less), only warning and error messages are printed. If *verbose* is set to 1, basic processing information is printed in addition to warning and error messages. This level is recommended. *verbose* levels greater than 1 provide information useful only for debugging.

Examples

Consider the mesh of the channel problem. AcuSurf may be executed as:

```
acuMesh2Tet -pb channel -srf channel.outflow.srf,channel.wall.srf
```

or alternatively the options may be put into the configuration file `Acusim.cnf` as follows:

```
problem= channel
surface_file_list= channel.outflow.srf,channel.wall.srf
```

and AcuMesh2Tet invoked as

```
acuMesh2Tet
```

The result will be one new interior connectivity file and two new surface connectivity files: – `channel.cnn.tet`, `channel.outflow.srf.tri` and `channel.wall.srf.tri`.

AcuProj

Project the solution fields of a solved problem to another mesh.

Syntax

`acuProj [options]`

Type

AcuSolve Preparatory Program

Description

AcuProj is a simple utility program that projects the solution from the restart files of a solved problem with one mesh to another mesh. The generated files are suitable for inclusion in the `NODAL_INITIAL_CONDITION` command of the new problem.

In the following, the full name of each option is followed by its abbreviated name and its type. For a general description of option specifications, see *Command Line Options and Configuration Files*. See below for more individual option details:

`help` or *`h`* *(boolean)*
: If set, the program prints a usage message and exits. The usage message includes all available options, their current values, and the place where each option is set.

`problem` or *`pb`* *(string)*
: The name of the problem is specified via this option. All generated file names start with this name.

`coordinates_file` or *`crd`* *(string)*
: The coordinates file for the new mesh. The file must contain four columns: node number and `x`, y, z coordinates. If *`coordinates_file`* is set to _auto, `problem.crd` and `problem.crd.B` are assumed.

`from_problem` or *`fpb`* *(string)*
: The name of the solved problem is specified via this option. If *`from_problem`* is set to _auto, the name of the problem option is used.

`from_working_directory` or *`fdir`* *(string)*
: The working directory of the solved problem. If a relative directory address is provided, that is, the directory does not start with an "/", it is considered relative to the *`from_problem_directory`*.

`from_problem_directory` or *`fpdir`* *(string)*
: The home directory of the solved problem.

`from_run_id` or *`run`* *(integer)*
: The run number of the solved problem from which the projection is requested. If *`from_run_id`* is set to 0, the last eligible run is assumed; see description below for more details.

`from_time_step` or *`ts`* *(integer)*
: The time step of the solved problem from which the projection is requested. If *`from_time_step`* is set to 0, the last available time step consistent with *`from_run_id`* is assumed; see description below for more details.

`search_technique` or `search` (enumerated)

Technique for evaluating the projection:

node	Choose the value of the closest node.
element	Perform element interpolation.

`line_buff` or `lbuff` (boolean)

Flush standard output after each line of output.

`verbose` or `v` (integer)

Set the verbose level for printing information to the screen. Each higher verbose level prints more information. If *`verbose`* is set to 0, or less, only warning and error messages are printed. If *`verbose`* is set to 1, basic processing information is printed in addition to warning and error messages. This level is recommended. *`verbose`* levels greater than 1 provide information useful only for debugging.

Examples

Suppose you have a solved channel problem in the problem directory channel1. Assume you have a new mesh in a sister directory channel2. To project the solution from channel1 to channel2, issue the following command from channel2:

```
acuProj -pb channel -crd channel.crd -fpb channel -fpdir ../channel1
```

or alternatively place the options in the configuration file `Acusim.cnf` as

```
problem                   = channel
coordinates_file          = channel.crd
from_problem              = channel
from_problem_directory    = ../channel1
```

and invoke AcuProj as

```
acuProj
```

The above command takes the latest run which has a restart file and projects its nodal fields to the nodes of the new mesh. If you assume that the channel problem was solved for the flow equation only, then the above command generates the files `channel.vel.nic` and `channel.pres.nic` for the nodal velocity and pressure fields, respectively. These files may be included in the new input file via the `NODAL_INITIAL_CONDITION` command as follows:

```
NODAL_INITIAL_CONDITION ( velocity ) {
nodal_values = Read( "channel.vel.nic" )
}
NODAL_INITIAL_CONDITION ( pressure ) {
nodal_values = Read( "channel.pres.nic" )
}
```

The following table lists the nodal fields that AcuProj projects from the restart file, if such fields exist. Also included in the table are the names of the resulting files.

Nodal Field	**File Name**
velocity	`problem.vel.nic`
pressure	`problem.pres.nic`
temperature	`problem.temp.nic`
species	`problem.spec_id.nic`
eddy viscosity	`problem.eddy.nic`
turbulence kinetic energy	`problem.tke.nic`
turbulence eddy frequency	`problem.tomega.nic`
viscoelastic stress	`problem.vest.nic`

For the species field, ID ranges from 1 to the number of species.

The options from_run_id and from_time_step specify the restart run and time step to extract the data. Either or both options may be set to 0, indicating the use of the latest data:

Table 1:

from_run_id	from_time_step	Approach
>0	>0	Use the restart data from the specified run and time step
0	>0	Use the restart data from highest run that has the specified restart time step
>0	0	Use the restart data from the last time step of the specified run
0	0	Use the restart data from the latest run and time step

The physical domains of the two meshes do not need to match. If a nodal point of the new mesh falls outside the domain of the solved mesh, the values on the boundary node of the solved mesh closest to the node under consideration are used.

Two projection methods are available: node and element. The node search technique assigns the values of the closest node from the solved problem, whereas the element search technique performs an element interpolation. The latter technique is more accurate, but potentially much more expensive.

AcuPbc

Creates a periodic boundary condition file from two surfaces.

Syntax

`acuPbc [options]`

Type

AcuSolve Preparatory Program

Description

AcuPbc is a simple utility program that creates a periodic boundary condition file suitable for the *`nodal_pairs`* parameter of the `PERIODIC_BOUNDARY_CONDITION` command. The two surfaces of the periodic condition are defined by two nodal boundary condition files. These surfaces must be the same geometrically up to a simple translation (periodic condition) or rotation (axisymmetric condition).

> **Note:** It is not sufficient for the nodes to satisfy this condition; the surface elements must satisfy it as well.

In the following, the full name of each option is followed by its abbreviated name and its type. For a general description of option specifications, see *Command Line Options and Configuration Files*. See below for more individual option details:

`help` *or* *`h`* *(boolean)*
: If set, the program prints a usage message and exits. The usage message includes all available options, their current values, and the place where each option is set.

`problem` *or* *`pb`* *(string)*
: The name of the problem is specified via this option. The periodic boundary condition file name is generated by appending `.pbc` to this name.

`coordinates_file` *or* *`crd`* *(string)*
: The coordinates file for the problem. The file must contain four columns: node number and x, y, z coordinates. If *`coordinates_file`* is set to _auto, `problem.crd` and `problem.crd.B` are assumed.

`nodal_bc_1_file` *or* *`nbc1`* *(string)*
: The nodal boundary condition file associated with the first surface.

`nodal_bc_2_file` *or* *`nbc2`* *(string)*
: The nodal boundary condition file associated with the second surface.

`periodic_type` *or* *`ptype`* *(enumerated)*
: The type of the periodicity:

periodic	Periodic boundary condition.
axisymmetric	Axisymmetric boundary condition.

`rotation_axis` or `axis` *(string)*

Array of six comma-separated numbers defining the axis of rotation. Used with axisymmetric periodic type.

`verbose` or `v` *(integer)*

Set the verbose level for printing information to the screen. Each higher verbose level prints more information. If `verbose` is set to 0 (or less), only warning and error messages are printed. If `verbose` is set to 1, basic processing information is printed in addition to warning and error messages. This level is recommended. `verbose` levels greater than 1 provide information useful only for debugging.

Examples

Suppose a periodic boundary condition between the inflow and outflow surfaces of a channel problem is needed. The command would be:

```
acuPbc -pb channel -crd channel.crd -nbc1 channel.inflow.nbc \
-nbc2 channel.outflow.nbc -ptype periodic
```

or alternatively place the options in the configuration file `Acusim.cnf` as

```
problem= channel
coordinates_file= channel.crd
nodal_bc_1_file= channel.inflow.nbc
nodal_bc_2_file= channel.outflow.nbc
periodic_type= periodic
```

and invoke AcuPbc as

```
acuPbc
```

The output is a new file named `channel.pbc`.

For an axisymmetric problem the option rotation_axis is also required. This option is a string that consists of six comma-separated numbers. The first three numbers are the coordinates of one point on the axis of rotation and the second three are the coordinates of another point on this axis. For example, a periodic boundary condition file for a fan problem with an axis of rotation along the z-axis can be created with the command:

```
acuPbc -pb fan -crd fan.crd -nbc1 fan.srf1.nbc -nbc2 fan.srf2.nbc \
-ptype axisymmetric -axis 0,0,0,0,0,1
```

This produces two files: `fan.pbc` and `fan.pbc.nbc`. The latter file contains all nodes which are common to both nodal boundary condition files. Normally this file contains all the nodes on the axis, which usually require special nodal boundary conditions. It is created for all periodic types, but is not useful for periodic types other than axisymmetric.

AcuPev

Project the eigenvectors of a solid mesh solution to the fluid nodes.

Syntax

`acuPev [options]`

Type

AcuSolve Preparatory Program

Description

AcuPev is a utility program that projects a solution from a solid mesh eigenvalue problem to a set of given fluid nodes. The results are written to disk in a format suitable for performing fluid-structure interaction (FSI) simulations in AcuSolve. Two formats are supported: raw and nike. The first uses separate files for coordinates, connectivity, and each eigen mode. The second reads a Nike3D solution file, which contains all the necessary data.

In the following, the full name of each option is followed by its abbreviated name and its type. For a general description of option specifications, see *Command Line Options and Configuration Files*. See below for more individual option details:

`help` or *`h`* *(boolean)*

If set, the program prints a usage message and exits. The usage message includes all available options, their current values, and the place where each option is set.

`problem` or *`pb`* *(string)*

The name of the problem. This is used to generate file names.

`project_from` or *`from`* *(enumerated)*

Project from this format:

raw	Project from raw file. Requires *`solid_coordinates_file`*, *`solid_connectivity_file_list`*, and *`solid_mode_file_list`*.
nike	Project from Nike3D file. Requires input_file_name.

`input_file_name` or *`file`* *(string)*

The name of the file from which the solid mesh and eigenvalue solution is to be imported. Used with nike format.

`solid_coordinates_file` or *`scrd`* *(string)*

Nodal coordinates file name of the solid mesh. The file must contain four columns: node number and x, y, z coordinates. Used with raw format.

`solid_connectivity_file_list` or *`scnn`* *(string)*

Comma-separated list of connectivity files of the solid mesh. The shape of the elements is extracted from the number of columns in each file. The elements are assumed to be either 4-node tetrahedra or quads. See *`solid_connectivity_main_dimension`* if the file has 5 columns; 5-node pyramids if 6 columns; 6-node wedges if 7 columns; 8-node bricks if 9 columns; and 10-node tetrahedra if 11 columns. Used with raw format.

solid_connectivity_main_dimension or *scnndim* *(enumerated)*
Dominant dimensionality of the solid mesh connectivity:

2	Two dimensional; 4-node connectivity files are quads.
3	Three dimensional; 4-node connectivity files are test.

solid_mode_file_list or *smode* *(string)*
List of mode (eigenvalue/vector) files from the solid mesh solution. Used with raw format.

coordinates_file or *crd* *(string)*
Nodal coordinates file name of the fluid mesh. The file must contain four columns: node number and x, y, z coordinates. If *coordinates_file* is set to _auto, `problem.crd` and `problem.crd.B` are assumed.

node_file_list or *node* *(string)*
Comma-separated list of node files of the fluid mesh onto which the eigenvectors are projected.

surface_file_list or *srf* *(string)*
Comma-separated list of surface files of the fluid mesh onto which the eigenvectors are projected.

contact_constraints_file or *ccf* *(string)*
Contact constraints file name. The file must contain one line for each constraint and 10 columns: a point on the flexible body, x, y, z coordinates, a point on the rigid plane, x, y, z coordinates, the normal to the rigid plane, x, y, z components, and the restitution coefficient. If _none, no contact is assumed. If not _none, the file `contact_file.cmd` is produced that contains the data required for contact by the `FLEXIBLE_BODY` command.

generate_input_commands or *ginp* *(boolean)*
If set, a `FLEXIBLE_BODY` command is printed out which may be placed in the input file with little modification.

line_buff or *lbuff* *(boolean)*
Flush standard output after each line of output.

verbose or *v* *(integer)*
Set the verbose level for printing information to the screen. Each higher verbose level prints more information. If *verbose* is set to 0, or less, only warning and error messages are printed. If *verbose* is set to 1, basic processing information is printed in addition to warning and error messages. This level is recommended. *verbose* levels greater than 1 provide information useful only for debugging.

Examples

To use the raw format, issue the command:

```
acuPev -pb plate -from raw -scrd solid.crd -scnn solid.cnn \
-smode mode01.dat,mode02.dat -crd fluid.crd -node surface.nod \
-ccf contact.dat -ginp TRUE
```

Alternatively, place the options in the configuration file `Acusim.cnf` as:

```
problem= plate
project_from= raw
solid_coordinates_file= solid.crd
solid_connectivity_file_list= solid.cnn
solid_mode_file_list= mode01.dat,mode02.dat
coordinates_file= fluid.crd
node_file_list= surface.nod
contact_constraints_file= contact.dat
generate_input_commands= TRUE
```

and issue the command:

```
acuPev
```

One file from either *node_file_list* or *surface_file_list* is required. More may be supplied. The nodes are extracted from all the surface files and combined with the nodes from all the node files. Then the eigenvectors from each mode in *solid_mode_file_list* are projected to the coordinates corresponding to the combined nodes. The output is three files: `problem.crd.xmd`, `problem.crd.ymd`, and `problem.crd.zmd`. These are the x, y, and z components of the projected eigenvectors. The first column in each file is the fluid node number. The rest of the columns are the eigenvectors, one for each mode in *solid_mode_file_list*, in the same order. These files may be passed directly to the *nodal_modes* parameter of the `NODAL_BOUNDARY_CONDITION` command. AcuPev also sends lines like:

```
acuPev: Eigenvalue 1 vec 12/linf = 2.50000e+00 ...
acuPev: Eigenvalue 2 vec 12/linf = 4.00000e+00 ...
```

to standard output. These are the eigenvalues that are used in the stiffness parameter of the `FLEXIBLE_BODY` command. If *contact_constraints_file* is not _none, then the contact data from this file is read and a file named `contact_file.cmd` is produced that may be passed directly to the *contact_constraints* parameter of the `FLEXIBLE_BODY` command. Setting *generate_input_commands* to TRUE generates a `FLEXIBLE_BODY` command that is sent to standard output. This command usually requires little modification to be used in the input file.

The format of both the solid and fluid coordinates files are the same as in the `COORDINATES` command. The format of the files in *solid_connectivity_files* is the same as in `ELEMENT_SET`; here the topology is determined by the number of columns. *solid_connectivity_main_dimension* is used to distinguish between quadrilaterals and tetrahedra. The format of the surface files is the same as in the various surface commands: elementID, surfaceID, and several columns for the surface connectivity. AcuPev ignores the first two columns and uses the connectivity as above. Each of the mode files has the following format. The first line contains two columns: index eigenvalue. All the subsequent lines contain four columns: nodeId x-eigenvector y-eigenvector z-eigenvector. *index* must match the order in *solid_mode_file_list*. That is, *index* = 1 for the first file given, and so on. *nodeId* and the number of rows for the eigenvectors must match that of *solid_coordinates_file*. The file referred to by *contact_constraints_file* consists of one line for each constraint and 10 columns: a point on the flexible body, x, y, z coordinates, a point on the rigid plane, x, y, z coordinates, the normal to the rigid plane, x, y, z components, and the restitution coefficient. See the `FLEXIBLE_BODY` command for a description of these quantities.

The macro acuAnsys2Pev.mac is available from within ANSYS to translate an ANSYS solution to raw format files.

The stand-alone Python script file `acuNastran2Pev.py` is available to extract the needed data from MSC-Nastran eigenvalue solution files and translate them to raw format files.

A Nike3D solution file may also be used. For example,

```
acuPev -pb plate -from nike -file nike_plate -crd fluid.crd -node surface.nod \
-ccf contact.dat -ginp TRUE
```

This produces the same output files as above.

AcuSif

Split a set of nodes into multiple sets and create new geometry files.

Syntax

`acuSif [options]`

Type

AcuSolve Preparatory Program

Description

AcuSif is a utility program that splits a set of nodes into multiple sets and creates new geometry files. These files are saved to a directory specified by *`target_directory`*.

All the nodes specified by *`split_node_file`* are not necessarily split. AcuSif takes every node in this "candidate" list and finds all the elements from *`connectivity_file_list`* connected to it. This subset of elements are connected to each other by the surfaces they share. The subset is then further divided into groups where each group consists of elements connected to each other, but not by surfaces that are entirely made up of candidate nodes. For every group, except the first group, a duplicate node, of the node considered, is introduced and used for that group.

This algorithm can give unexpected results, especially at corners. For example, consider a tetrahedral mesh and a candidate list of nodes that forms a right circular cylinder within that mesh. The goal is to split all the elements inside the cylinder from those outside. However, in the corners there likely will be many tetrahedra that consist completely of nodes from the candidate list. These will be split off from the others. In this case the solution is to eliminate the corners from the candidate list by performing the split in two passes: in the first pass just the nodes from the bottom and top of the cylinder are used for the candidate list, and in the second pass the circular wall nodes are used.

Alternatively, *`split_surface_file`* instead of *`split_node_file`* may be used. The two parameters are mutually exclusive. In this case a list of surfaces in the usual format is provided in the specified file. The algorithm is essentially the same as above, but is more robust since it does not suffer from the corner problem. However, if multiple surfaces are to be split off, usually because each surface can be associated with only one element set, then this executable must be executed multiple times, once per surface.

In the following, the full name of each option is followed by its abbreviated name and its type. For a general description of option specifications, see *Command Line Options and Configuration Files*. See below for more individual option details:

`help` or *`h`* *(boolean)*
: If set, the program prints a usage message and exits. The usage message includes all available options, their current values, and the place where each option is set.

`problem` or *`pb`* *(string)*
: The name of the problem. This is used to generate file names.

`coordinates_file` *or* `crd` *(string)*

Nodal coordinates file name. The file must contain four columns: node number and x, y, z coordinates. If `coordinates_file` is set to _auto, `problem.crd` and `problem.crd.B` are assumed.

`connectivity_file_list` *or* `cnn` *(string)*

Comma-separated list of connectivity files. The shape of the elements is extracted from the number of columns in each file. The elements are assumed to be 4-node tetrahedra if the file has 5 columns; 5-node pyramids if 6 columns; 6-node wedges if 7 columns; 8-node bricks if 9 columns; and 10-node tetrahedra if 11 columns. If _auto, all files of the form `problem*.cnn` and `problem*.cnn.B` are assumed.

`surface_file_list` *or* `srf` *(string)*

Comma-separated list of surface connectivity files. If _none, no surface files are split; if _auto, all files of the form `problem*.ebc`, `problem*.ebc.B`, `problem*.srf` and `problem*.srf.B` are assumed.

`surface_node_file_list` *or* `node` *(string)*

Comma-separated list of surface node files. If _none, no surface node files are split; if _auto, all files of the form `problem*.nbc` and `problem*.nbc.B` are assumed.

`split_node_file` *or* `snodes` *(string)*

Candidate split node file. Single column file of nodes. If _none, no split node file is used.

`split_surface_file` *or* `ssrf` *(string)*

Candidate split surface file. File format is elementId, surfaceId, node1,..., nodeN, where N is the number of nodes in the corresponding element face. If _none, no split surface file is used.

`output_period_nodes` *or* `opbc` *(boolean)*

If set, split nodes are output as a periodic boundary condition file named `problem.sif.pbc`.

`output_split_surfaces` *or* `osrf` *(boolean)*

If set, split surfaces are output as surface connectivity files with names of the form `problem.sif.*.srf`.

`output_shell_elements` *or* `oshl` *(boolean)*

If set, split surfaces are output as shell element files with names of the form `problem.sif.*.cnn`.

`external_surface_shell` *or* `eshl` *(boolean)*

If set, external surfaces are output as shell element sets.

`target_directory` *or* `tdir` *(string)*

Name of the directory to place new files.

`verbose` *or* `v` *(integer)*

Set the verbose level for printing information to the screen. Each higher verbose level prints more information. If `verbose` is set to 0, or less, only warning and error messages are printed. If `verbose` is set to 1, basic processing information is printed in addition to warning and error messages. This level is recommended. `verbose` levels greater than 1 provide information useful only for debugging.

Examples

```
acuSif -pb channel -crd channel.crd -cnn channel.cnn -srf channel_wall.srf \
-node channel_wall.nbc -snodes split.nod
```

Alternatively place the options in the configuration file `Acusim.cnf` as:

```
problem= channel
coordinates_file= channel.crd
connectivity_file_list= channel.cnn
surface_file_list= channel_wall.srf
surface_node_file_list= channel_wall.nbc
split_node_file= split.nod
```

and issue the command:

```
acuSif
```

If *`output_period_nodes`* is set, the pairs of nodes that came from a split node are saved into a file.

If *`output_split_surfaces`* is set, then the new surfaces created by the split are output. One surface is created for each side of the split. Different surface files are also created for each element set that is involved.

If *`output_shell_elements`* is set, then the pairs of new surfaces created by the split are used to create new shell elements and are then output into a file.

AcuTopoBlock

Generates nodal boundary conditions that are used as blockages for topology optimization run.

Syntax

`acuTopoBlock`

Type

AcuSolve Preparatory Program

Description

AcuTopoBlock is a utility program that uses the results from a topology optimization run to generate nodal boundary conditions that may be used to create solid blockage for a subsequent run. The utility is useful when the model has multiple inlets and outlets and the aim is to create separate fluid volumes that do not mix.

Parameters

The following parameters are supported:

`help` or `h` *(boolean)*
: If set, the program prints a usage message and exits. The usage message includes all available options, their current values, and the place where each option is set.

`problem` or `pb` *(string) [_undefined]*
: The name of the problem is specified via this option. This name is used to generate input file names and extracted surface file names.

`file` *(string) [_undefined]*
: The nodal boundary condition file associated with `block_file` = _undefined [default].

`design_level` *(real) [0.5]*
: A nodal boundary condition file is created to form a volume with `design_level`.

`reverse` *(boolean) [FALSE]*
: A nodal boundary condition file is created to block outside of the `design_level` (larger than design level).

`working_directory` or `dir` *(string) [ACUSIM.DIR]*
: All internal files are stored in this directory. This directory does not need to be on the same file system as the user-supplied input files.

`run_id` or `run` *(integer) [0]*
: Number of the run in which the translation is requested. If `run_id` is set to 0, the last run in the working directory is assumed.

`case_id` or `case` *(integer) [0]*
: Number of the case in which the translation is requested. If `case` is set to 0, the last case in the working directory is assumed. If the option `levelset_cut` = TRUE is used, the last case is always selected.

`levelset_cut` *(boolean) [FALSE]*

If `levelset_cut` is TRUE, levelset is used to generate the `.nbc` file for solid blockage instead of design topology.

`levelset_level` *(real) [0]*

This is an offset parameter to create a thickness of solid between the different fluid volumes if the levelset solution option is used (when `levelset_cut` = TRUE).

`verbose` *or* `v` *(integer)*

Set the verbose level for printing information to the screen. Each higher verbose level prints more information. If `verbose` is set to 0 (or less), only warning and error messages are printed. If `verbose` is set to 1, basic processing information is printed in addition to warning and error messages. This level is recommended. `verbose` levels greater than 1 provide information useful only for `debugging.verbose`= 1 [default].

Guidelines

AcuTopoBlock can create the `.nbc` file based on the computed design value in the simulation, or the levelset solution. If the levelset solution option is used, there is an offset parameter that may be used to create a "thickness" of the solid between the different fluid volumes.

Note: It is possible to repeat the process above if more than two volumes are computed. Multiple nodal boundary conditions files may be used.

To use level set instead of design topology:

`levelset_cut` = TRUE

The levelset offset is set by

`levelset_level` = 0.0

A possible workflow is as follows:

1. Prepare a model with multiple inlets and outlets. Decide which volume to create first. Set the inlet flow boundary condition and outlet boundary condition for the first target volume, and make the inlet and outlet walls for the remaining inlets and outlets.
2. Run the first topology flow optimization run to completion.
3. Use AcuTopoBlock to generate the nbc for the second run.
4. Set the inlet flow boundary condition and outlet boundary condition for the second target volume, and make the inlet and outlet walls for the remaining inlets and outlets. Include the `.nbc` file created by acuTopoBlock in a nodal boundary condition command for `design_topology` setting a constant value of 1.
5. Run the second topology flow optimization run to completion.

Post-Processing Programs

6

This chapter contains utility programs for post-processing or extracting data for post-processing solver results.

This chapter covers the following:

AcuTrans

Translates the AcuSolve solution output to other formats.

Syntax

`acuTrans [options]`

Type

AcuSolve Post-Processing Program

Description

The results of AcuSolve are stored using an internal format in a number of files in the directory specified by the *`working_directory`* option (`ACUSIM.DIR` by defaults) and in binary by default. AcuTrans is used to gather and translate these results into various formats more suited for post-processing or visualizing by third party products.

In the following, the full name of each option is followed by its abbreviated name and its type. For a general description of option specifications, see *Command Line Options and Configuration Files*. See below for more individual option details:

`help` or *`h`* *(boolean)*
: If set, the program prints a usage message and exits. The usage message includes all available options, their current values and the place where each option is set.

`problem` or *`pb`* *(string)*
: The name of the problem is specified via this option. This name is used to build internal file names and to generate output files. All generated output files start with the problem name.

`working_directory` or *`dir`* *(string)*
: All internal files are stored in this directory. This directory does not need to be on the same file system as the input files of the problem.

`run_id` or *`run`* *(integer)*
: Number of the run in which the translation is requested. If *`run_id`* is set to 0, the last run in the working directory is assumed.

`translate_to` or *`to`* *(enumerated)*
: Translate the output to this format:

info	Prints information about the available time steps and variables.
table	Translate into a two dimensional array of data
stats	Print statistics on each output data
cgns	Translate to CGNS data base
ensight	Translate to EnSight 6.0 .case file
fieldview	Translate to FieldView unstructured binary file (2.4 or 2.7)

h3d	Translate to H3D format
ideas	Translate to I-deas Universal file
spectrum	Translate to Spectrum Visualizer compressed ASCII file
actran	Translate to ActranLA files

`fieldview_options` or `fvopt` *(string)*
Set FieldView translation options:

classical	Translate to combined mesh/solution unstructured binary file format 2.4
split	Translate to split file mesh/solution unstructured binary file format 2.7 to reduce disk file size

`fieldview_region` or `fvr` *(enumerated)*
Set FieldView region:

single	Create a single region
medium	Create one region per medium, that is, solid, fluid, shell
element_set	Create one region per element set
subdomain	Create one region per original subdomain

`ensight_options` or `ensopt` *(string)*
Set EnSight translation options:

6	Translate to EnSight 6
gold	Translate to EnSight Gold

`h3d_options` or `h3dopt` *(string)*
Set H3D translation options:

multi	Create separate files for the mesh and solutions at various time-steps
single	Create a single file

`time_steps` or `ts` *(string)*
Comma-separated list of time steps to be translated. The comma-separated fields have the general range format **`beg:end:inc`**, where **`:end:inc`** and **`:inc`** are optional. **`beg`** specifies the first time step in the range. It may be either a given time step, as specified by a number, the letter F (or f) requesting the first available time step, or the letter L (or l) requesting the last available time step. end is the last time step in the range. It may be either a time step number or L (or l) requesting the last available time step. If **`end`** is missing, the range is assumed to simply represent a single time step, that is, end=beg and inc=1. inc is the increment that ranges from **`beg`** to **`end`**. It may be either a number or the letter A (or a) requesting all available time steps in the range. If **`:inc`** is missing, it is assumed to be one. The range may also be specified by the

single letter A (or a), requesting all available time steps. This is equivalent to F:L:A. *time_steps* is used only for nodal data and is ignored for time series data. Examples of *time_steps* option include:

```
acuTrans -ts 35              # step 35
acuTrans -ts 35,33,37         # steps 33, 35, and 37
acuTrans -ts 33:37:2          # steps 33, 35, and 37
acuTrans -ts 35,33:37:2,37    # steps 33, 35, and 37
acuTrans -ts 33:37            # all steps from 33 to 37
acuTrans -ts 33:37:A          # available steps from 33 to 37
acuTrans -ts F:L:A            # all available steps
acuTrans -ts A                # all available steps
```

optimization_case_time_steps or *cts* *(string)*
Comma-separated list of optimization case time steps to be translated. The comma-separated fields have the general range format **beg:end:inc**, where **:end:inc** and **:inc** are optional. **beg** specifies the first time step in the range. It may be either a given time step, as specified by a number, the letter F, or f, requesting the first available time step, or the letter L (or l) requesting the last available time step. **end** is the last time step in the range. It may be either a time step number or L (or l) requesting the last available time step. If **end** is missing, the range is assumed to simply represent a single time step, that is, end=beg and inc=1. **inc** is the increment that ranges from **beg** to **end**. It may be either a number or the letter A (or a) requesting all available time steps in the range. If **:inc** is missing, it is assumed to be one. The range may also be specified by the single letter A (or a), requesting all available time steps. This is equivalent to F:L:A.

optimization_case or *case* *(string)*
Comma-separated list of optimization cases to be translated.

additional_runs or *runs* *(string)*
Comma-separated list of runs to be translated. These are in addition to the main run given by *run_id*. Additional runs not compatible with the main are ignored. Combining four runs may be accomplished by:

```
acuTrans -run 1 -runs "2,3,4"
```

ignore_missing_steps or *imts* *(boolean)*
If set, missing requested time steps are ignored. Otherwise, if the requested time step does not exist, the command issues an error message and exits.

ignore_missing_cases or *imcs* *(boolean)*
If set, missing requested optimization cases are ignored. Otherwise, if the requested optimization case does not exist, the command issues an error message and exits.

ignore_missing_runs or *imruns* *(boolean)*
If set, missing runs are ignored. Otherwise, if the requested run does not exist, the command issues an error message and exits. This option is used with the -runs command line option.

ignore_missing_variables or *imv* *(boolean)*
If set, missing requested variables are ignored. Otherwise, if the requested variable does not exist, the command issues an error message and exits.

ignore_zeros or *iz* *(boolean)*
If set, ignore zeros when computing statistics, that is, when using -to stats. This option is usually used for variables that are defined only on surfaces. See *extended_nodal_output*.

mesh_output or *mesh* *(boolean)*
If set, the problem mesh is translated. This option is valid with translation formats ideas, spectrum, EnSight, H3D, FieldView and actran. The CGNS formats always receive the mesh. When converting to FieldView and *fvopt* is set to single, the mesh is always translated.

remove_duplicate_surfaces or *rmds* *(boolean)*
If set, all surfaces are checked for uniqueness before being written into the CGNS, FieldView, EnSight, or H3D file. This prevents surfaces that are utilized multiple times in the input file for SIMPLE_BOUNDARY_CONDITION, SURFACE_OUTPUT, or ELEMENT_BOUNDARY_CONDITION commands from appearing multiple times in output files. This option reduces the size of output files and eliminates redundant boundary definitions. The option is on by default. You can recover pre-V12.0 behavior in AcuTrans by turning this option off.

mesh_movement or *ale* *(boolean)*
If set, output deformed coordinates for *mesh_output*. Otherwise, output the reference coordinates.

deformed_crd_type or *defcrd* *(enumerated)*
This option controls which displacement field is written to the *mesh_displacement* variable in all output types.

midstep	Write the mesh displacement field that corresponds to the displacement at the mid point of each time step. This option is useful when investigating the flow across non-conformal interfaces. The mid point of each time step is where the solution across the non-conformal interface is actually satisfied and should be visualized in order to see continuous contours across it. However, the boundary displacements for prescribed motions will also be written at the mid step and may not agree with the expectations based on input settings.
endstep	Write the mesh displacement field that corresponds to the displacement at the end point of each time step. This option produces a displacement field that aligns with the requested boundary displacements at each step, but does not correspond to the mesh displacement field that satisfies the governing equations. To visualize the mesh on which the governing equations are satisfied, use midstep. In practice, there is little use in this if there are no non-conformal interfaces present in the simulation and visualizing the solution on the mesh with the expected boundary displacements is preferred.

nodal_output or *out* *(boolean)*
If set, the nodal output, as specified by the NODAL_OUTPUT command in the input file, is translated. This option is valid with all translation formats.

nodal_output_vars or *outv* *(string)*

Comma-separated list of *nodal_output* variables to be translated. The list may include:

Table 2:

Variable (abbr)	Fields	Description
node_id (node)	1	User-given node number
coordinates (crd)	3	Nodal coordinates
density (dens)	1	Density, only for *flow* = compressible_navier_stokes.
velocity (vel)	3	Velocity vector
phasic velocity (phasic_vel)	3	Phasic velocity vector of the first field, when *multi_field* = eulerian_eulerian or algebraic_eulerian.
mach_number (mach)	1	Mach number, only for *flow* = compressible_navier_stokes.
pressure (pres)	1	Pressure
temperature (temp)	1	Temperature
relative_humidity	1	Relative humidity
dewpoint_temperature	1	Dewpoint temperature
humidity_film_thickness	1	Humidity film thickness
species (spec)	nSpecs	Species
incident_radiation (incident_rad)	1	Incident radiation
field	nFields	Field values. For *multi_field*= levelset, *multi_field* = algebraic_eulerian and eulerian_eulerian, the field values correspond to volume fractions, and are named as *volume_fraction-"fieldname"*. For *multi_field* = advective_diffusive, the field values correspond to mass fraction.
levelset (levelset)	1	Levelset
eddy_viscosity (eddy)	1	Turbulence eddy viscosity

Variable (abbr)	Fields	Description
kinetic_energy (tke)	1	Turbulence kinetic energy
eddy_frequency (tomega)	1	Turbulence eddy frequency
sqrt_eddy_period (tg)	1	Inverse of the square root of eddy frequency
dissipation_rate (teps)	1	Turbulence dissipation rate
intermittency (tintc)	1	Turbulence intermittency
transition_re_theta (treth)	1	Critical momentum thickness Reynolds Number
surface_y_plus (yp)	1	y+ on turbulence walls
turbulence_y_plus (turb_yp)	1	Turbulence y+
surface_film_coefficient (film)	1	Convective heat transfer coefficient on turbulence walls
wall_shear_stress (wall_shear)	3	Wall shear stress on turbulence walls.
surface_heat_flux (osf_heat)	1	Surface heat flux
viscoelastic_stress (vest)	6	Viscoelastic stresses
mesh_displacement (mesh_disp)	3	Mesh displacement vector
mesh_velocity (mesh_vel)	3	Mesh velocity vector
electric_potential (elecP)	1	Electric potential (voltage)
joule_heat_density	1	Joule heat density (W/m^3)
current_density	3	Electric current flowing per unit cross-sectional area (Amp/m^2)

where *nSpecs* is the number of species as given in the `EQUATION` command in the input file. The problem must contain the requested variable in order for it to be translated. For example, the parameter *turbulence* in the `EQUATION` command must be set to a value other than none in order for *eddy_viscosity* to be available. The list of variables is sorted in the order given in the above table. If *nodal_output_vars* is set to _all, all available variables are translated. The *surface_y_plus*, *surface_film_coefficient* and *wall_shear_stress* are non-zero only on surface nodes given by `TURBULENCE_WALL`, or alternatively by `SIMPLE_BOUNDARY_CONDITION` of type wall. The *surface_film_coefficient* is computed even if there is no temperature equation. However, all relevant fluid material models must include specific heat and conductivity models. It should also be noted that *eddy_frequency* and *sqrt_eddy_period* only appear when using the k-omega turbulence models. The *dissipation_rate* variable only appears when using the k-epsilon based turbulence models, and *intermittency* and *transition_re_theta* only appear when using the turbulence transition models.

extended_nodal_output or extout (boolean)

Extended nodal output flag. If set, adds to the *nodal_output* variable list available variables from *running_average_output*, *time_average_output*, *derived_quantity_output*, *surface_output*, *radiation_surface*, *solar_radiation_surface*, *output_nodal_residual*, *output_error_estimator*, and *time_average_error_estimator*. The nodal projections of miscellaneous element quantities and gradients of available field variables are also added to the list. Those variables defined only on a subset of the nodes, such as surfaces, are set to zero on the rest of the nodes. The list may include:

Table 3:

Variable (abbr)	**Fields**	**Description**
grad_velocity (grad_vel)	9	Gradient of velocity vector
grad_phasic_velocity (grad_phasic_vel)	9	Gradient of phasic velocity vector of the first field, when *multi_field* = eulerian_eulerian or algebraic_eulerian.
grad_pressure (grad_pres)	3	Gradient of pressure
grad_temperature (grad_temp)	3	Gradient of temperature
grad_field	3*nFields	Gradient of field values. For *multi_field* = levelset, *multi_field*= algebraic_eulerian, and *multi_field* = eulerian_eulerian, the grad field values correspond to grad volume fractions, and are named as *grad_volume_fraction-"fieldname"*.
grad_species (grad_spec)	3*nSpecs	Gradient of species
grad_eddy_viscosity (grad_eddy)	3	Gradient of turbulence eddy viscosity
grad_kinetic_energy (grad_tke)	3	Gradient of turbulence kinetic energy
grad_eddy_frequency (grad_tomega)	3	Gradient of turbulence eddy frequency
grad_sqrt_eddy_period (grad_tg)	3	Gradient of inverse of the square root of eddy frequency

Variable (abbr)	Fields	Description
grad_dissipation_rate (grad_teps)	3	Gradient of inverse of the turbulence dissipation rate
grad_intermittency (grad_tintc)	3	Gradient of turbulence intermittency
grad_transition_re_theta (grad_treth)	3	Gradient of critical momentum thickness Reynolds Number
grad_viscoelastic_stress (grad_vest)	18	Gradient of viscoelastic stress
grad_mesh_displacement(grad_mesh_disp)	9	Gradient of mesh displacement vector
grad_mesh_velocity (grad_mesh_vel)	9	Gradient of mesh velocity vector
volume (vol)	1	Nodal volume
strain_rate_invariant_2 (strain_i2)	1	Second invariant of the strain rate tensor
vorticity (vort)	3	Vorticity
cfl_number (cfl)	1	Element-integrated CFL number
density (dens)	1	Density
viscosity (visc)	1	Viscosity
material_viscosity (mat_visc)	1	Molecular viscosity
gravity (grav)	3	Gravity
specific_heat (cp)	1	Specific heat
conductivity (cond)	6	Conductivity
heat_source (heat_src)	1	Heat source
diffusivity (diff)	nSpecs	Species diffusivity
species_source (spec_src)	nSpecs	Species source
turbulence_y (turb_y)	1	Turbulence distance to wall
turbulence_y_plus (turb_yp)	1	Turbulence y+
des_length (deslen)	1	Length scale for DES turbulence model
field_diffusivity	1	Field diffusivity

Variable (abbr)	Fields	Description
field_source	1	Field source
material_conductivity	1	Material conductivity
total_pressure (tot_pres)	1	Total pressure
running_ave_velocity (ora_vel)	3	Running average velocity
running_ave_pressure (ora_pres)	1	Running average pressure
running_ave_temperature (ora_temp)	1	Running average temperature
running_ave_field (ora_field)	nFields	Running average field values. For *multi_field* = levelset and *multi_field* = algebraic_eulerian, the field values correspond to running average volume fractions, and are named as *running_ave_volume_fraction-"fieldn*
running_ave_species (ora_spec)	nSpecs	Running average species
running_ave_eddy_viscosity (ora_eddy)	1	Running average eddy viscosity
running_ave_kinetic_energy (ora_tke)	1	Running average turbulence kinetic energy
running_ave_eddy_frequency (ora_tomega)	1	Running average turbulence eddy frequency
running_ave_sqrt_eddy_period (ora_tg)	1	Running average of inverse of the square root of eddy frequency
running_ave_dissipation_rate (ora_teps)	1	Running average turbulence dissipation rate
running_ave_intermittency (ora_tintc)	1	Running average of the turbulence intermittency
running_ave_transition_re_theta (ora_treth)	1	Running average of the critical momentum thickness Reynolds number
running_ave_mesh_displacement (ora_mesh_disp)	3	Running average mesh displacement
running_ave_viscoelastic_stress (ora_vest)	6	Running average viscoelastic stress
residual_velocity (onr_vel)	3	Residual of momentum equations

Variable (abbr)	Fields	Description
residual_pressure (onr_pres)	1	Residual of continuity equation
residual_temperature (onr_temp)	1	Residual of temperature equation
residual_species (onr_spec)	nSpecs	Residual of species equations
residual_eddy_viscosity (onr_eddy)	1	Residual of turbulence eddy viscosity equation
residual_kinetic_energy (onr_tke)	1	Residual of turbulence kinetic energy equation
residual_eddy_frequency (onr_tomega)	1	Residual of turbulence eddy frequency equation
residual_sqrt_eddy_period (onr_tg)	1	Residual of inverse of the square root of eddy frequency
residual_dissipation_rate (onr_teps)	1	Residual of turbulence dissipation rate equation
residual_intermittency (onr_tintc)	1	Residual of the turbulence intermittency
residual_transition_re_theta (onr_treth)	1	Residual of the critical momentum thickness Reynolds number
residual_mesh_displacement (onr_mesh_disp)	3	Residual of mesh displacement equations
residual_viscoelastic_stress (onr_vest)	6	Residual of viscoelastic stress
error_estimator_volume (oee_vol)	1	Volume
error_estimator_covariant_metric (oee_covar)	6	Covariant metric
error_estimator_velocity (oee_vel)	3	Error estimate of momentum equations
error_estimator_pressure (oee_pres)	1	Error estimate of continuity equation
error_estimator_temperature (oee_temp)	1	Error estimate of temperature equation
error_estimator_species (oee_spec)	nSpecs	Error estimate of species equations
error_estimator_eddy_viscosity (oee_eddy)	1	Error estimate of turbulence equation

Variable (abbr)	Fields	Description
error_estimator_viscoelastic_stress (oee_vest)	6	Error estimate of viscoelastic equations
error_estimator_tau_velocity (oee_tau_vel)	1	Error estimate of least-squares metric for continuity equation
error_estimator_tau_pressure (oee_tau_pres)	1	Error estimate of least-squares metric for momentum equations
error_estimator_tau_temperature (oee_tau_temp)	1	Error estimate of least-squares metric for heat equation
error_estimator_tau_species (oee_tau_spec)	nSpecs	Error estimate of least-squares metric for species equations
error_estimator_tau_eddy_viscosity (oee_tau_eddy)	1	Error estimate of least-squares metric for turbulence equations
error_estimator_tau_viscoelastic_stress (oee_tau_vest)	6	Error estimate of least-squares metric for viscoelastic equations
time_ave_error_volume (oae_vol)	1	Time-averaged volume
time_ave_error_covariant_metric (oae_covar)	6	Time-averaged covariant metric
time_ave_error_velocity (oae_vel)	3	Time-averaged error estimate of momentum equations
time_ave_error_pressure (oae_pres)	1	Time-averaged error estimate of continuity equation
time_ave_error_temperature (oae_temp)	1	Time-averaged error estimate of temperature equation
time_ave_error_species (oae_spec)	nSpecs	Time-averaged error estimate of species equations
time_ave_error_eddy_viscosity (oae_eddy)	1	Time-averaged error estimate of turbulence equation
time_ave_error_viscoelastic_stress (oae_vist)	6	Time-averaged error estimate of viscoelastic stress
time_ave_error_tau_velocity (oae_tau_vel)	1	Time-averaged error estimate of least-squares metric for continuity equation

Variable (abbr)	Fields	Description
time_ave_error_tau_pressure (oae_tau_pres)	1	Time-averaged error estimate of least-squares metric for momentum equations
time_ave_error_tau_temperature (oae_tau_temp)	1	Time-averaged error estimate of least-squares metric for heat equation
time_ave_error_tau_species (oae_tau_spec)	nSpecs	Time-averaged error estimate of least-squares metric for species equations
time_ave_error_tau_eddy_viscosity (oae_tau_eddy)	1	Time-averaged error estimate of least-squares metric for turbulence equation
time_ave_error_tau_viscoelastic_stress (oae_tau_vest)	6	Time-averaged error estimate of least-squares metric for viscoelastic equations
time_ave_velocity (ota_vel)	3	Time-averaged velocity
time_ave_velocity_square (ota_vel_sqr)	6	Time-averaged velocity square
time_ave_velocity_regular (ota_vel_reg)	3	Time-averaged non-conservative velocity
time_ave_pressure (ota_pres)	1	Time-averaged pressure
time_ave_pressure_square (ota_pres_sqr)	1	Time-averaged square of pressure
time_ave_stress (ota_stress)	6	Time-averaged Cauchy stress
surface_area (osf_area)	1	Surface area
surface_mass_flux (osf_mass)	1	Surface mass flux
surface_momentum_flux (osf_mom)	3	Surface momentum flux
surface_traction (osf_trac)	3	Surface traction
surface_moment (osf_moment)	3	Surface moment
surface_convective_temperature_flux (osf_conv_temp)	1	Surface convective temperature flux
surface_convective_species_flux (osf_conv_spec)	nSpecs	Surface convective species flux

Variable (abbr)	**Fields**	**Description**
surface_species_flux (osf_spec_flux)	nSpecs	Surface species flux
radiation_area (orf_area)	1	Radiation area
radiation_heat_flux (orf_heat)	1	Radiation heat flux
radiation_mean_radiant_temperature (orf_mr_temp)	1	Radiation mean radiant temperature
solar_area (oqf_area)	1	Solar area
solar_heat_flux (oqf_heat)	1	Solar heat flux

where *nSpecs* is the number of species as given in the EQUATION command in the input file. The output fields for *time_ave_velocity_square* and *time_ave_stress* are xx, yy, zz, xy, yz, and zx. The output fields for gradient variables are, for example, ux, uy, uz, vx,....

running_average_output or *ora* *(boolean)*
If set, the nodal running average field output, as specified by the RUNNING_AVERAGE_OUTPUT command in the input file, is translated. This option is valid with translation formats table and stats.

running_average_output_vars or *orav* *(string)*
Comma-separated list of *running_average_output* variables to be translated. The list may include:

Table 4:

Variable (abbr)	**Fields**	**Description**
node_id (node)	1	User-given node number
coordinates (crd)	3	Nodal coordinates
velocity (vel)	3	Velocity vector
pressure (pres)	1	Pressure
temperature (temp)	1	Temperature
relative_humidity	1	Relative humidity
dewpoint_temperature	1	Dewpoint temperature
humidity_film_thickness	1	Humidity film thickness
field	nFields	Fields. For *multi_field* = levelset and *multi_field* = algebraic_eulerian, the field

Variable (abbr)	Fields	Description
		values correspond to volume fractions, and are named as *volume_fraction-"fieldname"*. For *multi_field* = advective_diffusive, the field values correspond to mass fraction.
species (spec)	nSpecs	Species
eddy_viscosity (eddy)	1	Turbulence eddy viscosity
kinetic_energy (tke)	1	Turbulence kinetic energy
eddy_frequency (tomega)	1	Turbulence eddy frequency
sqrt_eddy_frequency (sqrt_tg)	1	Inverse of the square root of eddy frequency
dissipation_rate (teps)	1	Turbulence dissipation rate
intermittency (tintc)	1	Turbulence intermittency
transition_re_theta (treth)	1	Critical momentum thickness Reynolds number
mesh_displacement (mesh_disp)	3	Mesh displacement vector
viscoelastic_stress (vest)	6	Viscoelastic stresses

where *nSpecs* is the number of species as given in the EQUATION command in the input file. The problem must contain the requested variable in order for it to be translated. For example, the parameter *turbulence* in the EQUATION command must be set to a value other than none in order for *eddy_viscosity* to be available. The list of variables is sorted in the order given in the above table. If *running_average_output_vars* is set to _all, all of the available variables are translated.

time_average_output or *ota (boolean)*
If set, the time averaged nodal output, as requested by the TIME_AVERAGE_OUTPUT command in the input file, is translated. This option is valid with translation formats table and stats.

time_average_output_vars or otav (string)
Comma-separated list of *time_average_output* variables to be translated. The list may include:

Table 5:

Variable (abbr)	Fields	Description
node_id (node)	1	User-given node number

Variable (abbr)	Fields	Description
coordinates (crd)	3	Nodal coordinates
velocity (vel)	3	Time-averaged velocity vector
velocity_square (vel_sqr)	6	Time-averaged velocity square
velocity_regular (vel_reg)	3	Non-conservative time-averaged velocity
pressure (pres)	1	Time-averaged pressure
pressure_square (pres_sqr)	1	Time-averaged pressure square
stress	6	Time-averaged stress

The output fields for *velocity_square* and stress are xx, yy, zz, xy, yz and zx. The list of variables is sorted in the order given in the above table. If *time_average_output_vars* is set to _all, all available variables are translated.

derived_quantity_output or *odq* *(boolean)*
If set, the derived quantity nodal output, as requested by the `DERIVED_QUANTITY_OUTPUT` command in the input file, is translated. This option is valid with all translation formats except Actran.

derived_quantity_output_vars or *odqv* *(string)*
Comma-separated list of *derived_quantity_output* variables to be translated. The list may include:

Table 6:

Variable (abbr)	Fields	Description
node_id (node)	1	User-given node number
coordinates (crd)	3	Nodal coordinates
density (dens)	1	Density
viscosity (visc)	1	Viscosity
gravity (grav)	3	Gravity vector
specific_heat (cp)	1	Specific heat
conductivity (cond)	6	Conductivity

Variable (abbr)	**Fields**	**Description**
material_conductivity (mat_cond)	1	Material conductivity
heat_source (heat_src)	1	Heat source
diffusivity (diff)	nSpecs	Diffusivity for species equations
species_source (spec_src)	nSpecs	Species source
field_diffusivity (field_diff)	1	Field diffusivity
field_source (field_src)	1	Field source
turbulence_y (turb_y)	1	Turbulence distance to wall
des_length (deslen)	1	Length scale for Detached Eddy Simulation (DES) models
turbulence_intensity (turb_intens)	1	*turbulence_intensity* only for two-equation turbulence models
material_viscosity (mat_visc)	1	Molecular viscosity
cfl_number (cfl)	1	CFL number only for *flow* = compressible_navier_stokes.
total_temperature (tot_temp)	1	Total temperature only for *flow* = compressible_navier_stokes.
total_pressure (tot_pres)	1	Total pressure

where *nSpecs* is the number of species as given in the EQUATION command in the input file. The problem must contain the requested variable in order for it to be translated. For example, the parameter turbulence in the EQUATION command must be set to a value other than none in order for *turbulence_y* to be available. The list of variables is sorted in the order given in the above table. If *derived_quantity_output_vars* is set to _all, all of the available variables are translated.

surface_output or *osf* (boolean)
If set, the nodal values of surface output, as requested by the SURFACE_OUTPUT command in the input file, is translated. This option is valid with translation formats table, stats, and spectrum.

surface_output_sets or *osfs* (string)
Comma-separated list of *surface_output* sets. These are the user-given names specified as the user-given name of the SURFACE_OUTPUT commands in the input file. If *surface_output_sets* is set to _all, all output sets are translated.

`surface_output_vars` or `osfv` *(string)*

Comma-separated list of `surface_output` variables to be translated. The list may include:

Table 7:

Variable (abbr)	Fields	Description
`node_id (node)`	1	User-given node number
`coordinates (crd)`	3	Nodal coordinates
`area`	1	Nodal area
`mass_flux (mass)`	1	Mass flux
`momentum_flux (mom)`	3	Momentum flux
`traction (trac)`	3	Traction
`moment`	3	Moment of momentum
`convective_temperature_flux (conv_temp)`	1	Advection of temperature
`heat_flux (heat)`	1	Diffusive heat flux
`field_flux (field)`	nFields	Field flux. For `multi_field` = levelset, `multi_field` = algebraic_eulerian and `multi_field` = eulerian_eulerian, `field_flux` is named as `field_flux_volume_fraction-"fieldname"`. For `multi_field` = advective_diffusive, `field_flux` is named as `field_flux_mass_fraction-“fieldname”`.
`convective_field_flux (conv_field)`	nFields	Convective field flux. For `multi_field` = levelset, `multi_field` = algebraic_eulerian and `multi_field` =

Variable (abbr)	**Fields**	**Description**
		eulerian_eulerian, the *convective_field_flux* is named as *convective_flux_volume_fraction-"fieldna* For *multi_field* = advective_diffusive, the *convective_field_flux* is named as *convective_flux_mass_fraction-"fieldname*
convective_species_flux (conv_spec)	nSpecs	Advection of species
species_flux (spec_flux)	nSpecs	Diffusive species flux

where *nSpecs* is the number of species given by the EQUATION command in the input file. The list of variables is sorted in the order given in the above table. If *surface_output_vars* is set to _all, all of the available variables are translated.

surface_integral_output or *osi* *(boolean)*
If set, the integrated values of surface output, as requested by the SURFACE_OUTPUT command in the input file, are translated. This option is valid with translation formats table and stats.

surface_integral_output_sets or *osis* *(string)*
Comma-separated list of *surface_output* sets. These are the user-given names specified as the user-given name of the SURFACE_OUTPUT commands in the input file. If *surface_integral_output_sets* is set to _all, all output sets are translated.

surface_integral_output_vars or *osiv* *(string)*
Comma-separated list of *surface_integral_output* variables to be translated. The list may include:

Table 8:

Variable (abbr)	**Fields**	**Description**
time_step (step)	1	Time step
time	1	Run time
area	1	Area
mass_flux (mass)	1	Integrated mass flux
momentum_flux (mom)	3	Integrated momentum flux
traction (trac)	3	Integrated Cauchy traction

Variable (abbr)	Fields	Description
moment	3	Integrated moment of momentum
convective_temperature_flux (conv_temp)	1	Integrated advection of temperature
heat_flux (heat)	1	Integrated diffusive heat flux
convective_species_flux (conv_spec)	nSpecs	Integrated advection of species
species_flux (spec_flux)	nSpecs	Integrated diffusive species flux
velocity (vel)	3	Integrated velocity
velocity_magnitude (vel_mag)	1	Integrated velocity magnitude
pressure (pres)	1	Integrated pressure
total_pressure (tot_pres)	1	Integrated total pressure
temperature (temp)	1	Integrated temperature
total_temperature (tot_temp)	1	Integrated total temperature
incident_radiation	1	Integrated incident radiation
relative_humidity	1	Integrated relative humidity
dewpoint_temperature	1	Integrated dewpoint temperature
humidity_film_thickness	1	Humidity film thickness
field_flux	nFields	Area averaged field flux values. For multi_field = levelset, multi_field = algebraic_eulerian and multi_field = eulerian_eulerian, field_flux is named as field_flux_volume_fraction-"f For multi_field = advective_diffusive, field_flux is named as field_flux_mass_fraction-“fie

Variable (abbr)	Fields	Description
convective_field_flux	nFields	Area averaged convective field flux values. For *multi_field* = levelset, *multi_field* = algebraic_eulerian and *multi_field* = eulerian_eulerian, the *convective_field_flux* is named as *convective_flux_volume_fracti* For *multi_field* = advective_diffusive, the *convective_field_flux* is named as *convective_flux_mass_fraction*
field	nFields	Area averaged field values. For *multi_field* = levelset, *multi_field* = algebraic_eulerian and *multi_field* = eulerian_eulerian, the field values correspond to volume fractions, and are named as *volume_fraction-"fieldname"*. For *multi_field* = advective_diffusive, the field values correspond to mass fraction, and are named as *mass_fraction-"fieldname"*.
species (spec)	nSpecs	Integrated species
eddy_viscosity (eddy)	1	Integrated turbulence eddy viscosity
kinetic_energy (tke)	1	Integrated turbulence kinetic energy
eddy_frequency (tomega)	1	Integrated turbulence eddy frequency
sqrt_eddy_period (tg)	1	Inverse of the square root of eddy frequency
dissipation_rate (teps)	1	Integrated turbulence dissipation rate

Variable (abbr)	Fields	Description
intermittency (tintc)	1	Integrated turbulence intermittency
transition_re_theta (treth)	1	Integrated critical momentum thickness
surface_y_plus (yp)	1	Integrated y+ on turbulence walls
surface_film_coefficient (film)	1	Integrated conv. heat transfer coef. on turb. walls
wall_shear_stress (wall_shear)	3	Integrated wall shear stress
viscoelastic_stress (vest)	6	Integrated viscoelastic stress
ave_density (ave_dens)	1	Integrated average density
mesh_displacement (mesh_disp)	3	Integrated mesh displacement
mesh_velocity (mesh_vel)	3	Integrated mesh velocity
total_current	1	Integrated total current
mass_ave_velocity (mass_vel)	3	Mass averaged velocity
mass_ave_pressure (mass_pres)	1	Mass averaged pressure
mass_ave_total_pressure (mass_tot_pres)	1	Mass averaged total pressure
mass_ave_temperature (mass_temp)	1	Mass averaged temperature
mass_ave _total_temperature (mass_tot_temp)	1	Mass averaged total temperature
mass_ave_field (mass_field)	nFields	Mass average field values. For *multi_field* = levelset, *multi_field* = algebraic_eulerian and *multi_field* = eulerian_eulerian, the field values correspond to mass averaged volume fractions, and are named as *mass_ave_volume_fraction-"fie* For *multi_field* = advective_diffusive, the field values correspond to mass

Variable (abbr)	Fields	Description
		fraction, and are named as *mass_ave_mass_fraction-"field*
mass_ave_species(mass_spec)	nSpecs	Mass averaged species
mass_ave_eddy_viscosity(mass_eddy)	1	Mass averaged eddy viscosity
mass_ave_kinetic_energy(mass_tke)	1	Mass averaged turbulence kinetic energy
mass_ave_eddy_frequency(mass_tomega)	1	Mass averaged turbulence eddy frequency
mass_ave_sqrt_eddy_period (mass_tg)	1	Mass averaged inverse of the square root of eddy frequency
mass_ave_dissipation_rate (mass_teps)	1	Mass averaged turbulence dissipation rate
mass_ave_intermittency (mass_tintc)	1	Mass averaged turbulence intermittency
mass_ave_transition_re_theta (mass_treth)	1	Mass averaged critical momentum thickness Reynolds number
mass_ave_viscoelastic_stress (mass_vest)	6	Mass averaged viscoelastic stress
mass_flux_ave_velocity (massf_vel)	3	Mass flux averaged velocity
mass_flux_ave_pressure (massf_pres)	1	Mass flux
mass_flux_ave_total_pressure (massf_tot_pres)	1	Mass flux averaged total pressure
mass_flux _ave_total_temperature (massf_tot_temp)	1	Mass flux averaged total temperature
mass_flux_ave_temperature (massf_temp)	1	Mass flux averaged temperature
mass_flux_ave_field (massf_field)	nFields	Mass flux averaged field values. For *multi_field* = levelset, *multi_field* = algebraic_eulerian and *multi_field* = eulerian_eulerian, the field values correspond to volume

Variable (abbr)	Fields	Description
		fractions, and are named as *mass_flux_ave_volume_fraction* For *multi_field* = advective_diffusive, the field values correspond to mass fractions, and are named as *mass_flux_ave_mass_fraction-"*
mass_flux_ave_species (massf_spec)	nSpecs	Mass flux averaged species
mass_flux_ave_eddy_viscosity (massf_eddy)	1	Mass flux averaged eddy viscosity
mass_flux_ave_kinetic_energy (massf_tke)	1	Mass flux averaged turbulence kinetic energy
mass_flux_ave_eddy_frequency (massf_tomega)	1	Mass flux averaged turbulence eddy frequency
mass_flux_ave_sqrt_eddy_period (massf_tg)	1	Mass flux averaged inverse of the square root of eddy frequency
mass_flux_ave_dissipation_rate (massf_teps)	1	Mass flux averaged turbulence dissipation rate
mass_flux_ave_intermittency (massf_tintc)	1	Mass flux averaged turbulence intermittency
mass_flux_ave_transition_re_theta (massf_treth)	1	Mass flux averaged critical momentum thickness Reynolds number
mass_flux_ave_viscoelastic_stress (massf_vest)	6	Mass flux averaged viscoelastic stress
bulk_temperature	1	Bulk temperature
partial_volume	1	Integrated

where *nSpecs* is the number of species given by the EQUATION command in the input file. The list of variables is sorted in the order given in the above table. If *surface_integral_output_vars* is set to _all, all of the available variables are translated.

surface_statistics_output or *oss* (boolean)
If set, the surface output statistics, as requested by the SURFACE_OUTPUT command in the input file, are translated. This option is valid with translation formats table and stats.

`surface_statistics_output_sets` or `osss` *(string)*

Comma-separated list of `surface_output` sets. These are the user-given names specified as the user-given name of the `SURFACE_OUTPUT` commands in the input file. If `surface_statistics_output_sets` is set to _all, all output sets are translated.

`surface_statistics_output_vars` or `ossv` *(string)*

Comma-separated list of `surface_statistics_output` variables to be translated. The list may include:

Table 9:

Variable (abbr)	**Fields**	**Description**
time_step (step)	1	Time step
time	1	Run time
min_velocity (min_vel)	3	Minimum of nodal velocity on the surface
min_pressure (min_pres)	1	Minimum of nodal pressure on the surface
min_total_pressure (min_tot_pres)	1	Minimum of nodal total pressure on the surface
min_temperature (min_temp)	1	Minimum of nodal temperature on the surface
min _total_temperature (min_tot_temp)	1	Minimum of nodal total temperature on the surface
min_incident_radiation	1	Minimum of nodal incident radiation on the surface
min_relative_humidity	1	Minimum of nodal relative humidity on the surface
min_dewpoint_temperature	1	Minimum of nodal dewpoint temperature on the surface
min_humidity_film_thickness	1	Minimum of nodal humidity film thickness on the surface
min_field (min_field)	nFields	Mimimum of nodal field values. For *multi_field* = levelset and *multi_field* = algebraic_eulerian, the field values correspond

Variable (abbr)	Fields	Description
		to volume fractions, and are named as *volume_fraction-"fieldname"* For *multi_field* = advective_diffusive, the field values correspond to mass fraction.
min_species (min_species)	nSpecs	Minimum of nodal species on the surface
min_eddy_viscosity (min_species)	1	Minimum of nodal turbulence eddy viscosity on the surface
min_kinetic_energy (min_tke)	1	Minimum of nodal turbulence kinetic energy on the surface
min_eddy_frequency (min_tomega)	1	Minimum of nodal turbulence eddy frequency on the surface
min_sqrt_eddy_period (min_tg)	1	Minimum of nodal inverse of square root of eddy frequency on the surface
min_dissipation_rate (min_teps)	1	Minimum of nodal turbulence dissipation rate
min_intermittency (min_tintc)	1	Minimum of nodal turbulence intermittency on the surface
min_transition_re_theta (min_treth)	1	Minimum of nodal critical momentum thickness Reynolds number on the surface
min_mesh_displacement (min_mesh_disp)	3	Minimum of nodal mesh displacement on the surface
min_mesh_velocity (min_mesh_vel)	3	Minimum of nodal mesh velocities on the surface

Variable (abbr)	Fields	Description
min_surface_y_plus (min_yp)	1	Minimum of nodal y plus on the surface
min_surface_film_coefficient (min_film)	1	Minimum of nodal surface film coefficient on the surface
min_wall_shear_stress (min_wall_shear_stress)	3	Minimum of nodal wall shear stress on the surface
min_density (min_dens)	1	Minimum of nodal density on the surface
min_viscoelastic_stress (min_vest)	6	Minimum of nodal viscoelastic stress on the surface
max_velocity (max_vel)	3	Maximum of nodal velocity on the surface
max_pressure (max_pres)	1	Maximum of nodal pressure on the surface
max_total_pressure (max_tot_pres)	1	Maximum of nodal total pressure on the surface
max_temperature (max_temp)	1	Maximum of nodal temperature on the surface
max _total_temperature (max_tot_temp)	1	Maximum of nodal total temperature on the surface
max_incident_radiation	1	Maximum of nodal incident radiation on the surface
max_relative_humidity	1	Maximum of nodal relative humidity on the surface
max_dewpoint_temperature	1	Maximum of nodal dewpoint temperature on the surface
max_humidity_film_thickness	1	Maximum of nodal humidity film thickness on the surface
max_field (max_field)	nFields	Maximum of nodal field values. For *multi_field* =

Variable (abbr)	Fields	Description
		levelset and *multi_field* =algebraic_eulerian, the field values correspond to volume fractions, and are named as *volume_fraction-"fieldname"* For *multi_field* = advective_diffusive, the field values correspond to mass fraction.
max_species (max_species)	nSpecs	Maximum of nodal species on the surface
max_eddy_viscosity (max_species)	1	Maximum of nodal turbulence eddy viscosity on the surface
max_kinetic_energy (max_tke)	1	Maximum of nodal turbulence kinetic energy on the surface
max_eddy_frequency (max_tomega)	1	Maximum of nodal turbulence eddy frequency on the surface
max_sqrt_eddy_period (max_tg)	1	Maximum of nodal inverse of square root of eddy frequency on the surface
max_dissipation_rate (max_teps)	1	Maximum of nodal turbulence dissipation rate
max_intermittency (max_tintc)	1	Maximum of nodal turbulence intermittency on the surface
max_transition_re_theta (min_treth)	1	Maximum of nodal critical momentum thickness Reynolds number on the surface
max_mesh_displacement (max_mesh_disp)	3	Maximum of nodal mesh displacement on the surface

Variable (abbr)	Fields	Description
max_mesh_velocity (max_mesh_vel)	3	Maximum of nodal mesh velocities on the surface
max_surface_y_plus (max_yp)	1	Maximum of nodal y plus on the surface
max_surface_film_coefficient (max_film)	1	Maximum of nodal surface film coefficient on the surface
max_wall_shear_stress (max_wall_shear)	3	Maximum of nodal wall shear stress
max_density (max_dens)	1	Maximum of nodal density on the surface
max_viscoelastic_stress (max_vest)	6	Maximum of nodal viscoelastic stress on the surface
std_velocity (std_vel)	3	Standard deviation of nodal velocity on the surface
std_pressure (std_pres)	1	Standard deviation of nodal pressure on the surface
std_total_pressure (std_tot_pres)	1	Standard deviation of nodal total pressure on the surface
std_temperature (std_temp)	1	Standard deviation of nodal temperature on the surface
std _total_temperature (std_tot_temp)	1	Standard deviation of nodal total temperature on the surface
std_incident_radiation	1	Standard deviation of nodal incident radiation on the surface
std_relative_humidity	1	Standard deviation of nodal relative humidity on the surface

Variable (abbr)	**Fields**	**Description**
std_dewpoint_temperature	1	Standard deviation of nodal dewpoint temperature on the surface
std_humidity_film_thickness	1	Standard deviation of nodal humidity film thickness on the surface
std_field (std_field)	nFields	Standard deviation of nodal field values. For *multi_field* = levelset and *multi_field* = algebraic_eulerian, the field values correspond to volume fractions, and are named as *volume_fraction-"fieldname"* For *multi_field* = advective_diffusive, the field values correspond to mass fraction.
std_species (std_species)	nSpecs	Standard deviation of nodal species on the surface
std_eddy_viscosity (std_species)	1	Standard deviation of nodal turbulence eddy viscosity on the surface
std_kinetic_energy (std_tke)	1	Standard deviation of nodal turbulence kinetic energy on the surface
std_eddy_frequency (std_tomega)	1	Standard deviation of nodal turbulence eddy frequency on the surface
std_sqrt_eddy_period (std_tg)	1	Standard deviation of nodal inverse of square root of eddy frequency on the surface
std_dissipation_rate (std_teps)	1	Standard deviation of nodal turbulence dissipation rate

Variable (abbr)	Fields	Description
std_intermittency (std_tintc)	1	Standard deviation of nodal turbulence intermittency on the surface
std_transition_re_theta (std_treth)	1	Standard deviation of nodal critical momentum thickness Reynolds number on the surface
std_mesh_displacement (std_mesh_disp)	3	Standard deviation of nodal mesh displacement on the surface
std_mesh_velocity (std_mesh_vel)	3	Standard deviation of nodal mesh velocities on the surface
std_surface_y_plus (std_yp)	1	Standard deviation of nodal y plus on the surface
std_surface_film_coefficient (std_film)	1	Standard deviation of nodal surface film coefficient on the surface
std_wall_shear_stress (std_wall_shear)	3	Standard deviation of nodal wall shear stress on the surface
std_density (std_dens)	1	Standard deviation of nodal density on the surface
std_viscoelastic_stress (std_vest)	6	Standard deviation of nodal viscoelastic stress on the surface
uni_velocity (uni_vel)	3	Uniformity index of velocity on the surface
uni_pressure (uni_pres)	1	Uniformity index of pressure on the surface
uni_total_pressure (uni_tot_pres)	1	Uniformity index of total pressure on the surface

Variable (abbr)	Fields	Description
uni_temperature (uni_temp)	1	Uniformity index of temperature on the surface
uni _total_temperature (uni_tot_temp)	1	Uniformity index of total temperature on the surface
uni_incident_radiation	1	Uniformity index of nodal incident radiation on the surface
uni_relative_humidity	1	Uniformity index of relative humidity on the surface
uni_dewpoint_temperature	1	Uniformity index of dewpoint temperature on the surface
uni_humidity_film_thickness	1	Uniformity index of humidity film thickness on the surface
uni_field (uni_field)	nFields	Uniformity index of nodal field values. For *multi_field* = levelset and *multi_field* = algebraic_eulerian, the field values correspond to volume fractions, and are named as *volume_fraction-"fieldname"* For *multi_field* = advective_diffusive, the field values correspond to mass fraction.
uni_species (uni_species)	nSpecs	Uniformity index of species on the surface
uni_kinetic_energy (uni_tke)	1	Uniformity index of turbulence kinetic energy on the surface
uni_eddy_frequency (uni_tomega)	1	Uniformity index of turbulence eddy frequency on the surface

Variable (abbr)	Fields	Description
uni_sqrt_eddy_period (uni_tg)	1	Uniformity index of inverse of square root of eddy frequency on the surface
uni_dissipation_rate (uni_teps)	1	Uniformity index of turbulence dissipation rate on the surface
uni_intermittency (uni_tintc)	1	Uniformity index of nodal turbulence intermittency on the surface
uni_transition_re_theta (uni_treth)	1	Uniformity index of nodal critical momentum thickness Reynolds number on the surface
uni_mesh_displacement (uni_mesh_disp)	3	Uniformity index of mesh displacement on the surface
uni_mesh_velocity (uni_mesh_vel)	3	Uniformity index of mesh velocity on the surface
uni_surface_y_plus (uni_yp)	1	Uniformity index of y plus on the surface
uni_surface_film_coefficient (uni_film)	1	Uniformity index of surface film coefficient on the surface
uni_wall_shear_stress (uni_wall_shear)	3	Uniformity index of wall shear stress on the surface
uni_density (uni_dens)	1	Uniformity index of density on the surface
uni_viscoelastic_stress (uni_vest)	6	Uniformity index of viscoelastic stress on the surface

where *nSpecs* is the number of species given by the `EQUATION` command in the input file. The list of variables is sorted in the order given in the above table. If *surface_statistics_output_vars* is set to _all, all of the available variables are translated.

output_radiation_surface or *orf* *(boolean)*

If set, the nodal values of radiation surfaces, as requested by the RADIATION_SURFACE command in the input file, is translated. At least one of the two nodal output parameters of the RADIATION_SURFACE command must be set in order for any data be translated. Nodal surface output is available only for radiation surfaces of type wall. This option is valid with translation formats table and stats.

output_radiation_surface_sets or *orfs* *(string)*

Comma-separated list of RADIATION_SURFACE sets. These are the user-given names specified as the user-given name of the RADIATION_SURFACE commands in the input file. If *output_radiation_surface_sets* is set to _all, all output sets are translated.

output_radiation_surface_vars or *orfv* *(string)*

Comma-separated list of *output_radiation_surface* variables to be translated. The list may include:

Table 10:

Variable (abbr)	**Fields**	**Description**
node_id (node)	1	User-given node number
coordinates (crd)	3	Nodal coordinates
area	1	Nodal area
heat_flux (heat)	1	Radiation heat flux
mean_radiant_temperature (mr_temp)	1	Mean radiant temperature

The list of variables is sorted in the order given in the above table. If *output_radiation_surface_vars* is set to _all, all of the available variables are translated.

output_radiation_integral or *ori* *(boolean)*

If set, the integrated values of radiation surface, as requested by the RADIATION_SURFACE command in the input file, are translated. At least one of the four output parameters of the RADIATION_SURFACE command must be set in order for any data be translated. This option is valid with translation formats table and stats.

output_radiation_integral_sets or *oris* *(string)*

Comma-separated list of *output_radiation_integral* sets. These are the user-given names specified as the user-given name of the RADIATION_SURFACE commands in the input file. If *output_radiation_integral_sets* is set to _all, all output sets are translated.

output_radiation_integral_vars or *oriv* *(string)*

Comma-separated list of *output_radiation_integral* variables to be translated. The list may include:

Table 11:

Variable (abbr)	Fields	Description
time_step (step)	1	Time step
time	1	Run time
area	1	Integrated area
heat_flux (heat)	1	Integrated radiation heat flux
temperature (temp)	1	Integrated temperature (fourth-power weighting)
mean_radiant_temperature (mr_temp)	1	Integrated mean radiant temperature (fourth-power weighting)

The list of variables is sorted in the order given in the above table. If *output_radiation_integral_vars* is set to _all, all of the available variables are translated.

output_solar_radiation_surface or *oqf (boolean)*
If set, the nodal values of solar radiation surfaces, as defined by the SOLAR_RADIATION_SURFACE command in the input file, is translated. There are no output parameters in the SOLAR_RADIATION_SURFACE. Instead solar radiation fluxes of all solved time steps are available for translation. This option is valid with translation formats table and stats.

output_solar_radiation_surface_sets or *oqfs (string)*
Comma-separated list of SOLAR_RADIATION_SURFACE sets. These are the user-given names specified as the user-given name of the SOLAR_RADIATION_SURFACE commands in the input file. If *output_solar_radiation_sets* is set to _all, all output sets are translated.

output_solar_radiation_surface_vars or *oqfv (string)*
Comma-separated list of *output_solar_radiation_surface* variables to be translated. The list may include:

Table 12:

Variable (abbr)	Fields	Description
node_id (node)	1	User-given node number
coordinates (crd)	3	Nodal coordinates
area	1	Nodal area
heat_flux (heat)	1	Radiation heat flux

The list of variables is sorted in the order given in the above table. If *output_solar_radiation_surface_vars* is set to _all, all of the available variables are translated.

output_solar_radiation_integral or *oqi* *(boolean)*
If set, the integrated values of radiation surface, as defined by the SOLAR_RADIATION_SURFACE command in the input file, are translated. There are no output parameters in the SOLAR_RADIATION_SURFACE. Instead solar radiation fluxes of all solved time steps are available for translation. This option is valid with translation formats table and stats.

output_solar_radiation_integral_sets or *oqis* *(string)*
Comma-separated list of *output_solar_radiation_integral* sets. These are the user-given names specified as the user-given name of the SOLAR_RADIATION_SURFACE commands in the input file. If*output_solar_radiation_integral_sets* is set to _all, all output sets are translated.

output_solar_radiation_integral_vars or *oqiv* *(string)*
Comma-separated list of *output_solar_radiation_integral* variables to be translated. The list may include:

Table 13:

Variable (abbr)	**Fields**	**Description**
time_step (step)	1	Time step
time	1	Run time
area	1	Integrated area
heat_flux (heat)	1	Integrated radiation heat flux

The list of variables is sorted in the order given in the above table. If *output_solar_radiation_integral_vars* is set to _all, all of the available variables are translated.

element_integral_output or *oei* *(boolean)*
If set, the integrated values of element output, as requested by the ELEMENT_OUTPUT command in the input file, is translated. This option is valid with translation formats table and stats.

element_integral_output_sets or *oeis* *(string)*
Comma-separated list of element_output output sets. These are the user-given names specified as the user-given name of the ELEMENT_OUTPUT commands in the input file. If *element_integral_output_sets* is set to _all, all output sets are translated.

element_integral_output_vars or *oeiv* *(string)*
Comma-separated list of *element_integral_output* variables to be translated. The list may include:

Table 14:

Variable (abbr)	Fields	Description
time_step (node)	1	Time step
time	1	Run time
volume (vol)	1	Volume
total_mass (mass)	1	Total mass
velocity (vel)	3	Integrated velocity
acceleration (accel)	3	Integrated acceleration
pressure (pres)	1	Integrated pressure
total_pressure (tot_pres)	1	Integrated total pressure
temperature (temp)	1	Integrated temperature
total_ temperature (tot_temp)	1	Integrated total temperature
viscoelastic_stress (vest)	6	Integrated viscoelastic stresses
species (spec)	nSpecs	Integrated species
field	nFields	Integrated field values. For *multi_field* = levelset and *multi_field* = algebraic_eulerian, the field values correspond to volume fractions, and are named as *volume_fraction-"fieldname"*.
eddy_viscosity (eddy)	1	Integrated turbulence eddy viscosity
kinetic_energy (tke)	1	Integrated turbulence kinetic energy
velocity scale (tvel)	1	Integrated velocity scale
dissipation_rate (teps)	1	Integrated turbulence dissipation rate
eddy_frequency (tomega)	1	Integrated turbulence eddy frequency
eddy_time (ttau)	1	Integrated eddy time
sqrt_eddy_period (tg)	1	Integrated inverse of square root of eddy frequency
intermittency (tintc)	1	Integrated turbulence intermittency

Variable (abbr)	Fields	Description
transition_re_theta (treth)	1	Integrated critical momentum thickness Reynolds number
mesh_displacement (mesh_disp)	3	Integrated mesh displacement
mesh_velocity (mesh_vel)	3	Integrated mesh velocity
grad_velocity (grad_vel)	9	Integrated gradient of velocity
grad_pressure (grad_pres)	3	Integrated gradient of pressure
grad_temperature (grad_temp)	3	Integrated gradient of temperature
grad_species (grad_spec)	3*nSpecs	Integrated gradient of species
grad_field (grad_field)	1	Integrated gradient of field
stress	6	Integrated Cauchy stress
heat_flux (heat)	3	Integrated diffusive heat flux
species_flux (spec_flux)	3*nSpecs	Integrated diffusive species flux
user_output (udf)	nUserOuts	Integrated user output
center_of_gravity (cg)	3	Integrated center of gravity
relative_humidity (rel_hum)	1	Integrated relative humidity
dewpoint_temperature (dewpoint_temp)	1	Integrated dewpoint temperature
electric_potential	1	Integrated electric potential
mass_ave_momentum (aveMom)	3	Integrated mass-averaged momentum
mass_ave_pressure (mass_pres)	1	Integrated mass averaged pressure
mass_ave_total_pressure (mass_tot_pres)	1	Integrated mass averaged total pressure
mass_ave_temperature	1	Integrated mass averaged temperature
mass_ave_total_temperature (mass_tot_temp)	1	Integrated mass averaged total temperature
mass_ave_enthalpy (mass_enpy)	1	Integrated mass averaged enthalpy
mass_ave_viscoelastic_stress (mass_vest)	6	Integrated mass averaged viscoelastic stresses

Variable (abbr)	Fields	Description
mass_ave_species	nSpecs	Integrated mass averaged species
mass_ave_field (mass_field)	nFields	Integrated mass averaged field values. For *multi_field* = levelset and *multi_field* = algebraic_eulerian, the field values correspond to mass averaged volume fractions, and are named as *mass_ave_volume_fraction-"fieldname"*.
mass_ave_eddy_viscosity (mass_eddy)	1	Integrated mass averaged turbulence eddy viscosity
mass_ave_kinetic_energy (mass_tke)	1	Integrated mass averaged turbulence kinetic energy
mass_ave_dissipation_rate (mass_teps)	1	Integrated mass averaged turbulence dissipation rate
mass_ave_eddy_frequency (mass_tomega)	1	Integrated mass averaged eddy frequency
mass_ave_eddy_time (mass_ttau)	1	Integrated mass averaged eddy time
mass_ave_sqrt_eddy_period (mss_tg)	1	Integrated mass averaged inverse of square root of eddy frequency
mass_ave_transition_re_theta (mass_treth)	1	Integrated mass averaged critical momentum thickness Reynolds number
mass_ave_intermittency (mass_tintc)	1	Integrated mass averaged turbulence intermittency
mass_ave_field (mass_field)	nFields	Integrated mass averaged field
mass_ave_relative_humidity (ave_rel_hum)	1	Integrated mass averaged relative humidity
mass_ave_dewpoint_temperature (ave_dewpoint_temp)	1	Integrated mass averaged dewpont temperature

where *nSpecs* is the number of species given by the `EQUATION` command and *nUsers* is the number of user variables given by the `ELEMENT_OUTPUT` command in the input file. The output fields for stress are xx, yy, zz, xy, yz, and zx. The output fields for gradient variables are, for example, ux, uy, uz, vx,.... The list of variables is sorted in the order given in the above table. If *element_integral_output_vars* is set to _all, all of the available variables are translated.

element_statistics_output or *oes* (*boolean*)
: If set, the element output statistics, as requested by the `ELEMENT_OUTPUT` command in the input file, are translated. This option is valid with translation formats table and stats.

element_statistics_output_sets or *oess* (*string*)
: Comma-separated list of *element_output* sets. These are the user-given names specified as the user-given name of the `ELEMENT_OUTPUT` commands in the input file. If *element_statistics_output_sets* is set to _all, all output sets are translated.

element_statistics_output_vars or *oesv* (*string*)
: Comma-separated list of *element_statistics_output* variables to be translated. The list may include:

Table 15:

Variable (abbr)	Fields	Description
time_step (step)	1	Time step
time	1	Run time
min_velocity (min_vel)	3	Minimum velocity in the element set
min_acceleration (min_accel)	3	Minimum acceleration in the element set
min_pressure (min_pres)	1	Minimum pressure in the element set
min_total_pressure (min_tot_pres)	1	Minimum total pressure in the element set
min_temperature (min_temp)	1	Minimum temperature in the element set
min_total_temperature (min_tot_temp)	1	Minimum total temperature in the element set
min_relative_humidity (min_rel_hum)	1	Minimum relative humidity in the element set
min_dewpoint_temperature (min_dewpoint_temp)	1	Minimum dewpoint temperature in the element set
min_field (min_field)	nFields	Mimimum in the element set. For *multi_field* = levelset and *multi_field* = algebraic_eulerian, the field values correspond to volume fractions, and are named as

Variable (abbr)	Fields	Description
		volume_fraction-"fieldname". For *multi_field* = advective_diffusive, the field values correspond to mass fraction.
min_species (min_spec)	nSpecs	Minimum species in the element set
min_eddy_viscosity (min_eddy)	1	Minimum turbulence eddy viscosity in the element set
min_dissipation_rate (min_teps)	1	Minimum turbulence dissipation rate in the element set
min_kinetic_energy (min_tke)	1	Minimum turbulence kinetic energy in the element set
min_eddy_frequency (min_tomega)	1	Minimum turbulence eddy frequency in the element set
min_eddy_time (min_ttau)	1	Minimum turbulence eddy time in the element set
min_sqrt_eddy_period (min_tg)	1	Minimum of inverse of square root of eddy frequency in the element set
min_intermittency (min_tintc)	1	Minimum turbulence intermittency in the element set
min_transition_re_theta (min_treth)	1	Minimum critical momentum thicnkess Reynolds number in the element set
min_mesh_displacement (min_mesh_disp)	3	Minimum mesh displacement in the element set
min_mesh_velocity (min_mesh_vel)	3	Minimum mesh velocity in the element set
min_viscoelastic_stress (min_vest)	6	Minimum viscoelastic stress in the element set
max_velocity (max_vel)	3	Maximum velocity in the element set

Variable (abbr)	Fields	Description
max_acceleration (max_accel)	3	Maximum acceleration in the element set
max_pressure (max_pres)	1	Maximum pressure in the element set
max_total_pressure (max_tot_pres)	1	Maximum total pressure in the element set
max_temperature (max_temp)	1	Maximum temperature in the element set
max_total_temperature (max_tot_temp)	1	Maximum total temperature in the element set
max_relative_humidity (max_rel_hum)	1	Maximum relative humidity in the element set
max_dewpoint_temperature (max_dewpoint_temp)	1	Maximum dewpoint temperature in the element set
max_field (max_field)	nFields	Maximum in the element set. For *multi_field* = levelset and *multi_field* = algebraic_eulerian, the field values correspond to volume fractions, and are named as *volume_fraction-"fieldname"*. For *multi_field* = advective_diffusive, the field values correspond to mass fraction.
max_species (max_spec)	nSpecs	Maximum species in the element set
max_eddy_viscosity (max_eddy)	1	Maximum turbulence eddy viscosity in the element set
max_kinetic_energy (max_tke)	1	Maximum turbulence kinetic energy in the element set
max_eddy_frequency (max_tomega)	1	Maximum turbulence eddy frequency in the element set
max_eddy_time (max_ttau)	1	Maximum turbulence eddy time in the element set

Variable (abbr)	Fields	Description
max_sqrt_eddy_period (max_tg)	1	Maximum of inverse of square root of eddy frequency in the element set
max_dissipation_rate (max_teps)	1	Maximum turbulence dissipation rate in the element set
max_intermittency (max_tintc)	1	Maximum turbulence intermittency in the element set set
max_transition_re_theta (max_treth)	1	Maximum critical momentum thicnkess Reynolds number in the element set
max_mesh_displacement (max_mesh_disp)	3	Maximum mesh displacement in the element set
max_mesh_velocity (max_mesh_vel)	3	Maximum mesh velocity in the element set
max_viscoelastic_stress (max_vest)	6	Maximum viscoelastic stress in the element set
std_velocity (std_vel)	3	Std deviation velocity in the element set
std_acceleration (std_accel)	3	Std deviation acceleration in the element set
std_pressure (std_pres)	1	Std deviation pressure in the element set
std_total_pressure (std_tot_pres)	1	Std deviation total pressure in the element set
std_temperature (std_temp)	1	Std deviation temperature in the element set
std_total_temperature (std_tot_temp)	1	Std deviation total temperature in the element set
std_relative_humidity (std_rel_hum)	1	Standard deviation of relative humidity in the element set
std_dewpoint_temperature (std_dewpoint_temp)	1	Standard deviation of dewpoint temperature in the element set

Variable (abbr)	Fields	Description
std_field (std_field)	nFields	Standard deviation of field values in the element set. For *multi_field* = levelset and *multi_field* = algebraic_eulerian, the field values correspond to volume fractions, and are named as *volume_fraction-"fieldname"*. For *multi_field* = advective_diffusive, the field values correspond to mass fraction.
std_species (std_spec)	nSpecs	Standard deviation of species in the element set
std_eddy_viscosity (std_eddy)	1	Std deviation turbulence eddy viscosity in the element set
std_kinetic_energy (std_tke)	1	Std deviation turbulence kinetic energy in the element set
std_eddy_frequency (std_tomega)	1	Std deviation turbulence eddy frequency in the element set
std_eddy_time (std_ttau)	1	Std deviation turbulence eddy time in the element set
std_sqrt_eddy_period (std_tg)	1	Standard deviation of inverse of square root of eddy frequency in the element set
std_dissipation_rate (std_teps)	1	Std deviation turbulence dissipation rate in the element set
std_intermittency (std_tintc)	1	Standard deviation of turbulence intermittency in the element set
std_transition_re_theta (std_treth)	1	Standard deviation of critical momentum thickness Reynolds number in the element set

Variable (abbr)	Fields	Description
std_mesh_displacement (std_mesh_disp)	3	Standard deviation of mesh displacement in the element set
std_mesh_velocity (std_mesh_vel)	3	Standard deviation of mesh velocities in the element set
std_viscoelastic_stress (std_vest)	6	Standard deviation of viscoelastic stress in the element set

where *nSpecs* is the number of species given by the `EQUATION` command in the input file. The list of variables is sorted in the order given in the table above. If *element_statistics_output_vars* is set to _all, all of the available variables are translated.

time_history_output or *oth* (boolean)
If set, the time history output, as requested by the `TIME_HISTORY_OUTPUT` command in the input file, is translated. This option is valid with translation formats table and stats.

time_history_output_sets or *oths* (string)
Comma-separated list of *time_history_output* sets. These are the user-given names specified as the user-given name of the `TIME_HISTORY_OUTPUT` commands in the input file. If *time_history_output_sets* is set to _all, all output sets are translated.

time_history_output_vars or *othv* (string)
Comma-separated list of *time_history_output* variables to be translated. The list may include:

Table 16:

Variable (abbr)	Fields	Description
time_step (step)	1	Time step
time	1	Run time
velocity (vel)	3	Velocity
pressure (pres)	1	Pressure
temperature (temp)	1	Temperature
relative_humidity	1	Relative humidity
dewpoint_temperature	1	Dewpoint temperature
humidity_film_thickness	1	Humidity film thickness

Variable (abbr)	Fields	Description
field	nFields	Field values. For *multi_field* = levelset and *multi_field* = algebraic_eulerian, the field values correspond to volume fractions, and are named as *volume_fraction-"fieldname"*. For *multi_field* = advective_diffusive, the field values correspond to mass fraction.
species (spec)	nSpecs	Species
eddy_viscosity (eddy)	1	Turbulence eddy viscosity
kinetic_energy (tke)	1	Turbulence kinetic energy
eddy_frequency (tomega)	1	Turbulence eddy frequency
sqrt_eddy_period (tg)	1	Inverse of square root of eddy frequency
dissipation_rate (teps)	1	Turbulence dissipation rate
intermittency (tintc)	1	Turbulence intermittency
transition_re_theta (treth)	1	Critical momentum thickness Reynolds number
mesh_displacement (mesh_disp)	3	Mesh displacement vector
mesh_velocity (mesh_vel)	3	Mesh velocity vector
viscoelastic_stress (vest)	6	Viscoelastic stress

where *nSpecs* is the number of species as given by the EQUATION command in the input file. The list of variables is sorted in the order given in the above table. If *time_history_output_vars* is set to _all, all of the available variables are translated.

time_history_output_nodes or *othn* *(string)*
Comma-separated list of *time_history_output* nodes to be translated. The format of this option is the same as the *time_steps* options given above.

fan_component_output or *ofc* *(boolean)*
If set, the integrated values of fan component, as specified by the FAN_COMPONENT command in the input file, are translated. This option is valid with translation formats table and stats.

fan_component_output_set or *ofcs* *(string)*
Comma-separated list of fan_component sets. These are the user-given names specified as the user-given name of the `FAN_COMPONENT` commands in the input file. If *fan_component_output_set* is set to _all, all fan components are translated.

fan_component_output_vars or *ofcv* *(string)*
Comma-separated list of fan_component variables to be translated. The list may include:

Table 17:

Variable (abbr)	**Fields**	**Description**
time_step (step)	1	Time step
time	1	Run time
area	1	Integrated area
mass_flux (mass)	1	Integrated mass flux

The list of variables is sorted in the order given in the above table. If *fan_component_output_vars* is set to _all, all of the available variables are translated.

heat_exchanger_component_output or *ohc* *(boolean)*
If set, the integrated values of heat exchanger component, as specified by the `HEAT_EXCHANGER_COMPONENT` command in the input file, are translated. This option is valid with translation formats table and stats.

heat_exchanger_component_output_set or *ohcs* *(string)*
Comma-separated list of heat_exchanger_component sets. These are the user-given names specified as the user-given name of the `HEAT_EXCHANGER_COMPONENT` commands in the input file. If *heat_exchanger_component_output_set* is set to _all, all heat exchanger components are translated.

heat_exchanger_component_output_vars or *ohcv* *(string)*
Comma-separated list of *heat_exchanger_component* variables to be translated. The list may include:

Table 18:

Variable (abbr)	**Fields**	**Description**
time_step (step)	1	Time step
time	1	Run time
area	1	Integrated area
mass_flux (mass)	1	Integrated mass flux
air_temperature (temp)	1	Area average of temperature

Variable (abbr)	Fields	Description
coolant_temperature (cool_temp)	1	Coolant top water temperature
coolant_heat (cool_heat)	1	Coolant heat reject
upstream_temperature	1	Upstream temperature

The list of variables is sorted in the order given in the above table. If *heat_exchanger_component_output_vars* is set to _all, all of the available variables are translated.

aero_acoustic_output or *oaa* *(boolean)*
If set, the computational aero-acoustic data at the sample points, as specified by the CAA_OUTPUT command in the input file, is translated. This option is valid with translation formats table, stats, and -actran.

aero_acoustic_output_sets or *oaas* *(string)*
Comma-separated list of aero_acoustic_output sets to be translated. Each item in the list corresponds to the qualifier of a CAA_OUTPUT command in the input file.

aero_acoustic_output_vars or *oaav* *(string)*
Comma-separated list of *aero_acoustic_output* variables to be translated. See the CAA_OUTPUT command for definitions of the variables. The list may include:

Table 19:

Variable (abbr)	Fields	Description
node_id (node)	1	User-given sample point number
coordinates (crd)	3	Sample point coordinates
velocity (vel)	3	Velocity
pressure (pres)	1	Pressure
momentum_stress (mom_stress)	6	Momentum stress tensor
total_stress (tot_stress)	6	Total stress tensor
lighthill_stress	6	Lighthill stress tensor
reduced_lighthill_stress	6	Reduced Lighthill stress tensor
div_momentum_stress (div_mom_stress)	3	Divergence of momentum stress
div_total_stress (div_tot_stress)	3	Divergence of total stress
div_lighthill_stress	3	Divergence of Lighthill stress

Variable (abbr)	Fields	Description
div_reduced_lighthill_stress	3	Divergence of reduced Lighthill stress
normal_momentum_flux (norm_mom_flux)	1	Normal component of momentum flux
normal_momentum_flux_rate (norm_mom_flux_rate)	1	Normal component of momentum flux rate
normal_div_total_stress (norm_div_tot_stress)	1	Normal component of divergence of total stress
fwh_monopole	1	fwh monopole
fwh_dipole	1	fwh dipole
ei_na	1	Nodal element volume
ei_na_velocity (ei_na_vel)	3	Element-integrated velocity
ei_na_pressure (ei_na_pres)	1	Element-integrated pressure
ei_na_momentum_stress (ei_na_mom_stress)	6	Element-integrated momentum stress tensor
ei_na_total_stress (ei_na_tot_stress)	6	Element-integrated total stress tensor
ei_na_lighthill_stress	6	Element-integrated Lighthill stress tensor
ei_na_reduced_lighthill_stress	6	Element-integrated reduced Lighthill stress tensor
ei_na_div_momentum_stress (ei_na_div_mom_stress)	3	Element-integrated divergence of momentum stress
ei_na_div_total_stress (ei_na_div_tot_stress)	3	Element-integrated divergence of total stress
ei_na_div_lighthill_stress	3	Element-integrated divergence of Lighthill stress
ei_na_div_reduced_lighthill_stress	3	Element-integrated divergence of reduced Lighthill stress
ei_gna_div_momentum_stress (ei_gna_div_mom_stress)	1	Element-integrated gradient of shape function contracted with divergence of momentum stress

Variable (abbr)	Fields	Description
ei_gna_div_total_stress (ei_gna_div_tot_stress)	1	Element-integrated gradient of shape function contracted with divergence of total stress
ei_gna_div_lighthill_stress	1	Element-integrated gradient of shape function contracted with divergence of Lighthill stress
ei_gna_div_reduced_lighthill_stress	1	Element-integrated gradient of shape function contracted with divergence of reduced Lighthill stress
si_na	1	Nodal surface area
si_na_normal (si_na_norm)	3	Surface-integrated normal
si_na_normal_pressure (si_na_norm_pres)	3	Surface-integrated normal pressure
si_na_normal_momentum_flux (si_na_norm_mom_flux)	1	Surface-integrated normal momentum flux
si_na_normal_momentum_flux_rate (si_na_norm_mom_flux_rate)	1	Surface-integrated normal momentum flux rate
si_na_normal_div_total_stress (si_na_norm_div_tot_stress)	1	Surface-integrated normal divergence of total stress
si_na_fwh_monopole	1	Surface-integrated FWH monopole
si_na_fwh_dipole	1	Surface-integrated FWH dipole
si_na_pressure	1	Surface-integrated FWH pressure

The output fields for stress quantities are xx, yy, zz, xy, yz, and zx. The list of variables is sorted in the order given in the above table. If *aero_acoustic_output_vars* is set to _all, all of the available variables are translated.

output_nodal_residual or *onr* *(boolean)*
If set, the nodal residual, as requested by the NODAL_RESIDUAL_OUTPUT command in the input file, is translated. This option is valid with translation formats table, stats, FieldView and H3D.

output_nodal_residual_vars or *onrv* *(string)*
Comma-separated list of *output_nodal_residual* variables to be translated. The list may include:

Table 20:

Variable (abbr)	Fields	Description
node_id (node)	1	User-given node number
coordinates (crd)	3	Nodal coordinates
velocity (vel)	3	Momentum equation residual
pressure (pres)	1	Continuity equation residual
temperature (temp)	1	Heat equation residual
species (spec)	nSpecs	Species equation residual
eddy_viscosity (eddy)	1	Turbulence equation residual
kinetic_energy (tke)	1	Kinetic energy residual
eddy_frequency (tomega)	1	Eddy frequency residual
sqrt_eddy_period (tg)	1	Inverse of square root of eddy frequency residual
dissipation_rate (teps)	1	Turbulence dissipation rate
mesh_displacement (mesh_disp)	3	Mesh displacement equation residual
viscoelastic_stress (vest)	6	Viscoelastic stress residual

where *nSpecs* is the number of species given by the EQUATION command in the input file.

output_error_estimator or *oee (boolean)*
If set, the error estimate, as requested by the ERROR_ESTIMATOR_OUTPUT command in the input file, is translated. This option is valid with translation formats table, stats, and FieldView.

output_error_estimator_vars or *oeev (string)*
Comma-separated list of *output_error_estimator* variables to be translated. The list may include:

Table 21:

Variable (abbr)	Fields	Description
node_id (node)	1	User-given node number
coordinates (crd)	3	Nodal coordinates

Variable (abbr)	Fields	Description
volume (vol)	1	Volume
covariant_metric (covar)	6	Covariant metric
velocity (vel)	3	Error estimate of momentum equation
pressure (pres)	1	Error estimate of continuity equation
temperature (temp)	1	Error estimate of heat equation
species (spec)	nSpecs	Error estimate of species equations
eddy_viscosity (eddy)	1	Error estimate of turbulence equation
viscoelastic_stress (vest)	6	Error estimate of viscoelastic stress
tau_velocity (tau_vel)	1	Error estimate of least-squares metric for continuity equation
tau_pressure (tau_pres)	1	Error estimate of least-squares metric for momentum equations
tau_temperature (tau_temp)	1	Error estimate of least-squares metric for heat equation
tau_species (tau_spec)	nSpecs	Error estimate of least-squares metric for species equations
tau_eddy_viscosity (tau_eddy)	1	Error estimate of least-squares metric for turbulence equations
tau_viscoelastic_stress (tau_vest)	6	Error estimate of least-squares metric for viscoelastic equations

where *nSpecs* is the number of species given by the `EQUATION` command in the input file.

time_average_error_estimator or *oae (boolean)*
If set, the aaa estimate, as requested by the `ERROR_ESTIMATOR_OUTPUT` command in the input file, is translated. This option is valid with translation formats table, stats, and FieldView.

time_average_error_estimator_vars or *oaev (string)*
Comma-separated list of *time_average_error_estimator* variables to be translated. The list may include:

Table 22:

Variable (abbr)	Fields	Description
node_id (node)	1	User-given node number
coordinate (crd)	3	Nodal coordinates
volume (vol)	1	Time-averaged volume
covariant_metric (covar)	6	Time-averaged covariant metric
velocity (vel)	3	Time-averaged error estimate of momentum equation
pressure (pres)	1	Time-averaged error estimate of continuity equation
temperature (temp)	1	Time-averaged error estimate of heat equation
species (spec)	nSpecs	Time-averaged error estimate of species equations
eddy_viscosity (eddy)	1	Time-averaged error estimate of turbulence equation
viscoelastic_stress (vest)	6	Time-averaged error estimate of viscoelastic equations
tau_velocity (tau_vel)	1	Time-averaged error estimate of least-squares metric for continuity equation
tau_pressure (tau_pres)	1	Time-averaged error estimate of least-squares metric for momentum equations
tau_temperature (tau_temp)	1	Time-averaged error estimate of least-squares metric for heat equation
tau_species (tau_spec)	nSpecs	Time-averaged error estimate of least-squares metric for species equations
tau_eddy_viscosity (tau_eddy)	1	Time-averaged error estimate of least-squares metric for turbulence equations

Variable (abbr)	Fields	Description
tau_viscoleastic_stress (tau_vest)	6	Time-averaged error estimate of least-squares metric for viscoelastic equations

where *nSpecs* is the number of species given by the `EQUATION` command in the input file.

moment_center or *mc* (string)
: Comma-separated list of coordinates that represent the point about which moments should be computed for surface nodal outputs and surface integrated outputs. This option provides the ability to transform the moments computed by AcuSolve from the global origin to any location in space.

moment_center_mesh_motion or *mcmm* (string)
: Name of the mesh motion command used to transform the *moment_center* for moving mesh applications.

flexible_body_output or *ofb* (boolean)
: If set, the values describing the state of a flexible body, as specified by the `FLEXIBLE_BODY` command in the input file, are translated. This option is valid with the translation formats table and stats.

flexible_body_output_sets or *ofbs* (string)
: Comma-separated list of flexible body names. These are the (string) specified as the user-given name of the `FLEXIBLE_BODY` commands in the input file. If *flexible_body_output_sets* is set to _all, all flexible bodies are translated.

flexible_body_output_vars or *ofbv* (string)
: Comma-separated list of `FLEXIBLE_BODY` variables to be translated. The list may include:

Table 23:

Variable (abbr)	Fields	Description
time_step (step)	1	Time step
time	1	Run time
displacement (disp)	1	Modal displacement of the flexible body
velocity (vel)	1	Modal velocity of the flexible body
internal_force	1	Modal forces applied to the flexible body that were extracted from the fluid loading

Variable (abbr)	Fields	Description
external_force	1	Modal forces applied to the flexible body as an external force

The list of variables is sorted in the order given in the table above. If *flexible_body_output_vars* is set to _all, all of the available variables are translated. When the *translate_to* option is set to table, the results from each mode specified for the flexible body are written to a separate file.

line_buff or *lbuff* *(boolean)*
Flush standard output after each line of output.

verbose or *v* *(integer)*
Set the verbose level for printing information to the screen. Each higher verbose level prints more information. If *verbose* is set to 0 (or less), only warning and error messages are printed. If *verbose* is set to 1, basic processing information is printed in addition to warning and error messages. This level is recommended. *verbose* levels greater than 1 provide information useful only for debugging.

Examples

To translate nodal output results of the channel problem in order to visualize with FieldView, issue the command:

```
acuTrans -pb channel -to fieldview -out -ts 5
```

which creates the file `channel_step000005.fv` suitable for visualization by FieldView. Alternatively place the option(s) in the configuration file `Acusim.cnf` as follows:

```
problem= channel
translate_to= fieldview
nodal_output= TRUE
time_steps= 5
```

and issue the command:

```
acuTrans
```

Due to restrictions in the Unix shell, any option value containing spaces must be enclosed in a pair of single or double quotes. If a list of strings is requested, such as in the case of *surface_output_sets* option, enclose the names in pairs of single or double quotes, namely:

```
acuTrans -osf -osfs "surface one","surface two"
```

or alternatively:

```
acuTrans -osf -osfs "surface one,surface two"
```

This precludes having commas in the surface names.

AcuSolve is capable of generating 19 types of results. Each type may be translated separately. The following table lists the solver commands needed to request AcuSolve to generate the output and the AcuTrans options for translating them:

Solver Command	AcuTrans Option
NODAL_OUTPUT	nodal_output
RUNNING_AVERAGE_OUTPUT	running_average_output
DERIVED_QUANTITY_OUTPUT	derived_quantity_output
TIME_AVERAGE_OUTPUT	time_average_output
SURFACE_OUTPUT	surface_output
SURFACE_OUTPUT	surface_integral_output
SURFACE_OUTPUT	surface_statistics_output
RADIATION_SURFACE	output_radiation_surface
RADIATION_SURFACE	output_radiation_integral
SOLAR_RADIATION_SURFACE	output_solar_radiation_surface
SOLAR_RADIATION_SURFACE	output_solar_radiation_integral
ELEMENT_OUTPUT	element_integral_output
TIME_HISTORY_OUTPUT	time_history_output
FAN_COMPONENT	fan_component_output
HEAT_EXCHANGER_COMPONENT	heat_exchanger_component_output
CAA_OUTPUT	aero_acoustic_output
NODAL_RESIDUAL_OUTPUT	output_nodal_residual

ERROR_ESTIMATOR_OUTPUT

output_error_estimator time_average_error_estimator

FLEXIBLE_BODY

flexible_body_output

These results may be translated into one of the following formats: Table, Stats, CGNS, FieldView, H3D, I-deas, Spectrum, Ensight, and Actran. Not all result types may be translated into all formats. The following two tables list the supported combinations:

Table 24:

AcuTrans Option	Table	Stats	CGNS	FieldView
mesh_output	no	no	yes*	yes*
nodal_output	yes	yes	yes	yes
running_average_output	yes	yes	no	no
derived_quantity_output	yes	yes	yes	yes
time_average_output	yes	yes	no	no
surface_output	yes	yes	no	no
surface_integral_output	yes	yes	no	no
surface_statistics_output	yes	yes	no	no
output_radiation_surface	yes	yes	no	no
output_radiation_integral	yes	yes	no	no
output_solar_radiation_surface	yes	yes	no	no
output_solar_radiation_integral	yes	yes	no	no
element_integral_output	yes	yes	no	no
time_history_output	yes	yes	no	no
fan_component_output	yes	yes	no	no
heat_exchanger_component_output	yes	yes	no	no
aero_acoustic_output	yes	yes	no	no
output_nodal_residual	yes	yes	no	yes
output_error_estimator	yes	yes	no	yes

AcuTrans Option	Table	Stats	CGNS	FieldView
time_average_error_estimator	yes	yes	no	yes

Table 25:

AcuTrans Option	H3D	I-deas	Spectrum	EnSight	Actran
mesh_output	yes	yes	yes	yes	yes
nodal_output	yes	yes	yes	yes	no
running_average_output		no	no	no	no
derived_quantity_output	yes	yes	yes	yes	no
time_average_output	no	no	no	no	no
surface_output	no	no	yes	no	no
surface_integral_output	no	no	no	no	no
surface_statistics_output	yes	yes	no	no	no
output_radiation_surface	no	no	no	no	no
output_radiation_integral	no	no	no	no	no
output_solar_radiation_surface	no	no	no	no	no
output_solar_radiation_integral	no	no	no	no	no
element_integral_output	no	no	no	no	no
time_history_output	no	no	no	no	no
fan_component_output	no	no	no	no	no
heat_exchanger_component_output	no	no	no	no	no
aero_acoustic_output	no	no	no	no	yes
output_nodal_residual	no	no	no	no	no
output_error_estimator	no	no	no	no	no
time_average_error_estimator	no	no	no	no	no

yes*: always output the mesh when `nodal_output` is specified.

Info Format

For info format, AcuTrans prints information about all the available time steps and variables.

Table Format

For table format, AcuTrans generates a number of output files, each containing a two-dimensional array of data. In the following problem is given by problem and step is the time step.

```
nodal output:
```

The nodal output of a given time step is written as a two dimensional table in an output file named `problem_stepstep.out`. The rows in the table correspond to the coordinates node numbers, written in the same order given by the `COORDINATE` command in the input file. The columns correspond to the translated data, in the order given in the *`nodal_output_vars`* table. Each data type occupies one or more columns, as given in the Fields column of the above table. The following example:

```
acuTrans -pb channel -out -outv pres,node,vel -ts 10
```

creates the file `channel_step10.out`, which has 5 columns corresponding to the node_id, x_velocity, y_velocity, z_velocity, and pressure. If the extended nodal output flag is set, the nodal projections of gradients of available field variables and available variables from *`running_average_output`*, *`time_average_output`*, *`derived_quantity_output`*, *`surface_output`*, *`output_radiation_surface`*, *`output_solar_radiation_surface`*, *`output_nodal_residual`*, *`output_error_estimator`*, and *`time_average_error_estimator`* are added to the *`nodal_output`* variable list, in the order given in the *`extended_nodal_output`* table. Those variables defined only on a subset of the nodes, such as surfaces, are set to zero on the rest of the nodes. The full list of available variables for a particular problem can be obtained with the command

```
acuTrans -pb channel -out -extout -to stats
```

For example, solar heat flux can be added to the nodal output as follows:

```
acuTrans -pb channel -out -extout -outv pres,node,vel,oqf_heat -ts 10
```

```
running_average_output
```

The running average field output of a given time step is written as a two dimensional table in an output file named `problem_stepstep.ora`. The rows in the table correspond to the coordinates node numbers, written in the same order given by the `COORDINATE` command in the input file. The columns correspond to the translated data, in the order given in the *`running_average_output_vars`* table. Each data type occupies one or more columns, as given in the Fields column of the above table. The following example,

```
acuTrans -pb channel -ora -orav pres,node,vel -ts 10
```

creates the file `channel_step10.ora`, which has 5 columns corresponding to node_id, x_velocity, y_velocity, z_velocity, and pressure.

```
time_average_output
```

The time averaged output of a given time step is written as a two dimensional table in an output file named `problem_stepstep.ota`. The rows of the table correspond to the coordinate node numbers, written in the same order given by the `COORDINATE` command in the input file. The columns correspond to the translated data, in the order given in the *`time_average_output_vars`* table. Each data type occupies one or more columns, as given in the Fields column of the above table.

```
derived_quantity_output
```

The derived quantity output of a given time step is written as a two dimensional table in an output file named `problem_stepstep.odq`. The rows of the table correspond to the coordinate node numbers, written in the same order given by the `COORDINATE` command in the input file. The columns correspond to the translated data, in the order given in the *`derived_quantity_output_vars`* table. Each data type occupies one or more columns, as given in the Fields column of the above table.

```
surface_output
```

The nodal values of a given surface output set and a given time step are written as a two dimensional table in an output file named `problem_srfsrf_stepstep.osf`, where `srf` is a surface ID, starting from one, corresponding to each `SURFACE_OUTPUT` command in the order given in the input file. The rows of the tables correspond to all the nodes of the surface, numerically sorted. The columns correspond to the translated data, in the order given in the *`surface_output_vars`* table. Each data type occupies one or more columns, as given in the Fields column of the above table. For example, assume that "wall surface" is the second `SURFACE_OUTPUT` command in the input file, then

```
acuTrans -pb channel -osf -osfs "wall surface" -ts 10
```

creates the file `channel_srf2_step10.osf` containing all the output data available for surface output. The definitions of the surface output variables are given by the integrands of the *`surface_integral_output`* table below.

```
surface_integral_output
```

The integrated results of a given surface output set are written as a two-dimensional table in an output file named `problem_srf.osi`, where `srf` is a surface ID, starting from one, corresponding to each `SURFACE_OUTPUT` command in the order given in the input file. The rows of the tables correspond to all the available time steps. The columns correspond to the translated data, in the order given in the *`surface_integral_output_vars`* table. Each data type occupies one or more columns, as given in the Fields column of the above table. For example, assume that "wall -surface" is the second `SURFACE_OUTPUT` command in the input file, then

```
acuTrans -pb channel -osi -osis "wall surface" -osiv step,trac
```

creates the file `channel_srf2.osi` containing time step numbers and the surface integral of traction, for a total of four columns. The definitions of the surface integral output variables are given by:

Table 26:

Variable	**Definition**	**SI units**
area	$\int dA$	m^2
ave density	$\int \rho \, dA / \int dA$	kg/m^3
bulk temperature	$\int H\rho (u \cdot n) dA / \int C_p \rho (u \cdot n) dA$	K
convective_field_flux (conv_field)	$\int \rho F_i (u \cdot n) dA$	kg/sec
convective species flux	$\int \rho \, \varphi_i (u \cdot n) \, dA$	kg/sec
convective temperature flux	$\int \rho \, H (u \cdot n) dA$	W
dewpoint_temperature (dewpoint_temp)	$\int T_{dew} dA / \int dA$	K
dissipation rate	$\int \varepsilon \, dA / \int dA$	m^2/sec^3
eddy frequency	$\int \omega \, dA / \int dA$	1/sec
eddy viscosity	$\int \mu_t \, dA / \int dA$	m^2/sec
field	$\int F_i \, dA / \int dA$	non-dimensional
field_flux (field)	$\int F_i \cdot n dA$	kg/sec
heat flux	$\int q \cdot n \, dA, \quad q = \kappa_{TOT} \nabla T$	W
humidity_film_thickness	$\int HFT dA / \int dA$ where HFT is the humidity film thickness	m
incident_radiation (incident_rad)	$G = \sum_{i=1}^{N} w_i I(r)$ where I is the intensity in direction I, N is the number of ordinates, w is the weight given based on the number of ordinates.	W/m^2

Variable	Definition	SI units
intermittency	$\int \gamma \, dA / \int dA$	non-dimensional
kinetic energy	$\int k \, dA / \int dA$	m^2/sec^2
total_current	$\int I \, dA / \int dA$	Amp
mass average dissipation rate	$\int \varepsilon \rho \, dA / \int \rho \, dA$	m^2/sec^3
mass average eddy frequency	$\int \omega \rho \, dA / \int \rho \, dA$	1/sec
mass average eddy viscosity	$\int \mu_t \rho \, dA / \int \rho \, dA$	m^2/sec
mass average field	$\int F_i \rho \, dA / \int \rho \, dA$	non-dimensional
mass average intermittency	$\int \gamma \rho \, dA / \int \rho \, dA$	non-dimensional
mass average kinetic energy	$\int k \rho \, dA / \int \rho \, dA$	m^2/sec^2
mass average pressure	$\int p \rho \, dA / \int \rho \, dA$	N/m^2
mass average species	$\int \varphi_i \rho \, dA / \int \rho \, dA$	non-dimensional
mass average sqrt eddy period	$\int \sqrt{\omega} \, \rho \, dA / \int \rho \, dA$	(sec)$^{0.5}$
mass average temperature	$\int T \rho \, dA / \int \rho \, dA$	K
mass average total pressure	$\int p_{TOT} \rho \, dA / \int \rho \, dA$	N/m^2
mass average total temperature	$\int T_{TOT} \rho \, dA / \int \rho \, dA$	K
mass average transition re theta	$\int \mathrm{Re}_\theta \rho \, dA / \int \rho \, dA$	non-dimensional
mass average velocity	$\int u \rho \, dA / \int \rho \, dA$	m/sec
mass average viscoelastic stress	$\int \sigma \rho \, dA / \int \rho \, dA$	N/m^2
mass flux	$\int \rho (u \cdot n) dA$	kg/sec
mass flux average dissipation rate	$\int \varepsilon \rho (u \cdot n) dA / \int \rho (u \cdot n) dA$	m^2/sec^3

Variable	Definition	SI units
mass flux average eddy frequency	$\int \omega\, \rho(u \cdot n) dA / \int \rho(u \cdot n) dA$	1/sec
mass flux average eddy viscosity	$\int \mu_t\, \rho(u \cdot n) dA / \int \rho(u \cdot n) dA$	m^2/sec
mass flux average field	$\int F_i\, \rho(u \cdot n) dA / \int \rho(u \cdot n) dA$	non-dimensional
mass flux average intermittency	$\int \gamma\, \rho(u \cdot n) dA / \int \rho(u \cdot n) dA$	non-dimensional
mass flux average kinetic energy	$\int k\, \rho(u \cdot n) dA / \int \rho(u \cdot n) dA$	m^2/sec^2
mass flux average pressure	$\int p\, \rho(u \cdot n) dA / \int \rho(u \cdot n) dA$	N/m^2
mass flux average species	$\int \varphi\, \rho(u \cdot n) dA / \int \rho(u \cdot n) dA$	non-dimensional
mass flux average sqrt eddy period	$\int \sqrt{\omega}\, \rho(u \cdot n) dA / \int \rho(u \cdot n) dA$	$(sec)^{0.5}$
mass flux average temperature	$\int T\, \rho(u \cdot n) dA / \int \rho(u \cdot n) dA$	K
mass flux average total pressure	$\int p_{TOT}\, \rho(u \cdot n) dA / \int \rho(u \cdot n) dA$	N/m^2
mass flux average total temperature	$\int T_{TOT}\, \rho(u \cdot n) dA / \int \rho(u \cdot n) dA$	K
mass flux average transition re theta	$\int \mathrm{Re}_\theta\, \rho(u \cdot n) dA / \int \rho(u \cdot n) dA$	non-dimensional
mass flux average velocity	$\int u\, \rho(u \cdot n) dA / \int \rho(u \cdot n) dA$	m/sec
mass flux average viscoelastic stress	$\int \sigma\, \rho(u \cdot n) dA / \int \rho(u \cdot n) dA$	N/m^2
mesh displacement	$\int d_{ALE}\, dA / \int dA$	m
mesh velocity	$\int u_{ALE}\, dA / \int dA$	m/sec
moment	$\int r \times [(-pI + \tau) \cdot n] dA$	N m
momentum flux	$\int \rho(u \cdot n)\, u\, dA$	kg/(m sec^2)
partial volume	$\int \frac{1}{3}(x \cdot n)\, dA$	m^3

Variable	**Definition**	**SI units**
pressure	$\int p\, dA / \int dA$	N/m^2
relative_humidity (rel_hum)	$\int RH\, dA / \int dA$ where RH is relative humidity	non-dimensional
species	$\int \varphi_i\, dA / \int dA$	non-dimensional
species flux	$\int \varphi_i \cdot n\, dA$	m^2
sqrt eddy period	$\int \sqrt{\omega}\, dA / \int dA$	(sec)$^{0.5}$
surface film coefficient	$\int h\, dA / \int dA$	W/m^{2}K
surface y plus	$\int y^+ dA / \int dA,\ y^+ = y\frac{\rho}{\mu}\sqrt{\tau_w/\rho},\ \tau_w =$	non-dimensional
temperature	$\int T\, dA / \int dA$	K
time	$\int dt$	sec
time_step (step)		non-dimensional
total pressure	$\int p_{TOT}\, dA / \int dA$ $p_{TOT} = p + \frac{1}{2}\rho u^2$ for incompressible flow $p_{TOT} = p\left(1 + \frac{\gamma - 1}{2}M^2\right)^{\gamma/(\gamma-1)}$ for compressible flow	N/m^2
total temperature	$\int T_{TOT}\, dA / \int dA$ $T_{TOT} = T + \frac{1}{2}\frac{u^2}{C_p}$ for incompressible flow $T_{TOT} = T\left(1 + \frac{\gamma - 1}{2}M^2\right)$ for compressible flow	K
traction	$\int (-pI + \tau) \cdot n\, dA$ $\tau = 2\mu_{TOT}\nabla_s u$	N

Variable	Definition	SI units
transition re theta	$\int \mathrm{Re}_\theta dA / \int dA$	non-dimensional
velocity	$\int u\, dA / \int dA$	m/sec
viscoelastic stress	$\int \sigma\, dA / \int dA$	N/m^2
wall shear stress	$\int \tau_w\, dA / \int dA$	N/m^2

where the integrals are over the area defined by the corresponding SURFACE_OUTPUT command, n is the outward-pointing unit normal to the surface, that is, the mass flux at an inflow boundary is negative, ρ is the density, u is the velocity, p is the pressure, τ is the diffusive stress tensor, $\mu_{TOT} = \mu + \mu_t$ is the total viscosity (molecular plus turbulent), r is the radius vector, H is the enthalpy, C_p is the specific heat, q is the heat flux vector, $\kappa_{TOT} = \kappa + \kappa_t$ is the total conductivity (molecular plus turbulent), φ_i, i=1, ...,*nSpecs* are the species variables, ψ_i is the diffusive flux for species i, y^+ is the turbulence y^+ value at the first node away from the surface, τ_w is the wall shear stress, u_w is the normalized velocity vector at the first point off the wall, t is the unit vector parallel to the flow, h is the surface film coefficient extracted from the self-similarity solution near the edge of the boundary layer, d_{ALE} is the mesh displacement, u_{ALE} is the mesh velocity, k is the turbulent kinetic energy, ω is the turbulent eddy frequency, σ is the viscoelastic stress tensor and x is the coordinate vector.

By default, the moment output variable is computed about the global origin (0,0,0). This moment can be transformed to any desired location by using the -moment_center option of AcuTrans. For example, to evaluate the moment on a surface named "wall surface" about the point 1,0,0, the following command is used:

```
acuTrans -pb channel -osi -osis "wall surface" -osiv moment -mc 1.0,0.0,0.0
```

For moving mesh applications where the body of interest is moving, it is possible to compute the moment about a non-stationary point. For example, it may be desirable to compute the moment about the center of gravity of a rigid body. To accomplish this, the initial center point of the body is specified using the -moment option and the motion of the point is determined by referring to the MESH_MOTION command that controls the position of the body using the -moment_center_mesh_motion option. For example, to compute the moment about the center of gravity of an object undergoing rigid body motion, the following command is used:

```
acuTrans -pb movingMesh -osi -osis "rigid walls" -osiv moment -mc 0.5,0.5,0.0 -mcmm
 "my rigid body"
surface_statistics_output
```

The statistics of a given surface output set are written as a two dimensional table in an output file named `problem_srf.oss`, where `srf` is a surface ID, starting from one, corresponding to each SURFACE_OUTPUT command in the order given in the input file. The rows of the tables correspond to all the available time steps. The columns correspond to the translated data, in the order given in the `surface_statistics_output_vars` table. Each data type occupies one or more columns, as given in the Fields column of the above table. For example, assume that "`wall -surface`" is the second `SURFACE_OUTPUT` command in the input file, then

```
acuTrans -pb channel -oss -osss "wall surface" -ossv step,velocity
```

creates the file `channel_srf2.oss` containing time step numbers and the surface statistics of velocity, for a total of ten columns. The definitions of the surface statistic output variables are given by:

Variable	Definition
min_velocity	min(u_j)j=1,...,num. surface nodes
min_pressure	min(p_j)j=1,...,num. surface nodes
min_total_pressure	min(P_{TOTj})j=1,...,num. surface nodes
min_temperature	min(T_j)j=1,...,num. surface nodes
min_total_temperature	$\min(T_{TOT\ j})_{j=1,...num}$ surface nodes
min_species	min(φ_j)j=1,...,num. surface nodes
minimum_field	min(F_j)j=1,...,num. surface nodes
min_eddy_viscosity	min(μ_{tj})j=1,...,num. surface nodes
min_kinetic_energy	min(k_j)j=1,...,num. surface nodes
min_eddy_frequency	$\min\left(\frac{1}{\sqrt{\omega_j}}\right)$j=1,...,num. surface nodes
min_sqrt_eddy_period	$\min\left(\frac{1}{\sqrt{\omega_j}}\right)$j=1,...,num. surface nodes
min_dissipation_rate	min(ε_j)j=1,...,num. surface nodes
min_intermittency	min(γ_j)j=1,...,num. surface nodes
min_transition_re_theta	min($Re_{\theta j}$j=1,...,num. surface nodes
min_mesh_displacement	min(d_{ALEj})j=1,...,num. surface nodes
min_mesh_velocity	min(u_{ALEj})j=1,...,num. surface nodes
min_surface_y_plus	min(y^+j)j=1,...,num. surface nodes
min_surface_film_coefficient	min(h_j)j=1,...,num. surface nodes
min_wall_shear_stress	min($\tau_w u_{wj}$)j=1,...,num. surface nodes

Variable	Definition
min_density	min(ρ_j)j=1,...,num. surface nodes
min_viscoelastic_stress	min(σ_j)j=1,...,num. surface nodes
max_velocity	max(u_j)j=1,...,num. surface nodes
max_pressure	max(p_j)j=1,...,num. surface nodes
max_total_pressure	max(P_{TOTj})j=1,...,num. surface nodes
max_temperature	max(T_j)j=1,...,num. surface nodes
max_total_temperature	$\max(T_{TOT\ j})_{j=1,...num}$ surface nodes
max_species	max(φ_j)j=1,...,num. surface nodes
max_field	max(F_j)j=1,...,num. surface nodes
max_eddy_viscosity	max(μ_{tj})j=1,...,num. surface nodes
max_kinetic_energy	max(k_j)j=1,...,num. surface nodes
max_eddy_frequency	max(ω_j)j=1,...,num. surface nodes
max_sqrt_eddy_period	$\max\left(\frac{1}{\sqrt{\omega_j}}\right)$j=1,...,num. surface nodes
max_dissipation_rate	max(ε_j)j=1,...,num. surface nodes
max_intermittency	max(γ_j)j=1,...,num. surface nodes
max_transition_re_theta	$max(\mathrm{Re}_{\theta\ j})$ j=1,...,num. surface nodes
max_mesh_displacement	$max(d_{ALE\ j})$j=1,...,num. surface nodes
max_mesh_velocity	$max(u_{ALE\ j})$j=1,...,num. surface nodes
max_surface_y_plus	$max(y^{+}{}_j)$j=1,...,num. surface nodes
max_surface_film_coefficient	$max(h_j)$j=1,...,num. surface nodes
max_wall_shear_stress	$max(\tau_w u_{w\ j})$ j=1,...,num. surface nodes
max_density	$max(\rho_j)$j=1,...,num. surface nodes

Variable	Definition
max_viscoelastic_stress	$max(\sigma_j)$ j=1,...,num. surface nodes
std_velocity	$std(u_j)$ j=1,...,num. surface nodes
std_pressure	$std(p_j)$ j=1,...,num. surface nodes
std_total_pressure	$std(p_{TOT\,j})$ j=1,...,num. surface nodes
std_temperature	$std(T_j)$ j=1,...,num. surface nodes
std_total_temperature	$std(T_{TOT\ j})_{j=1,...num}$ surface nodes
std_species	$std(\varphi_j)$ j=1,...,num. surface nodes
std_field	$std(F_j)$ j=1,...,num. surface nodes
std_eddy_viscosity	$std(\mu_{t\,j})$ j=1,...,num. surface nodes
std_kinetic_energy	$std(k_j)$ j=1,...,num. surface nodes
std_eddy_frequency	$std(\omega_j)$ j=1,...,num. surface nodes
std_sqrt_eddy_period	$std(\frac{1}{\sqrt{\omega}_j})$ j=1,...,num. surface nodes
std_dissipation_rate	$std(\varepsilon_j)$ j=1,...,num. surface nodes
std_intermittency	$std(\gamma_j)$ j=1,...,num. surface nodes
std_transition_re_theta	$std(\mathrm{Re}_{\theta\,j})$ j=1,...,num. surface nodes
std_mesh_displacement	$std(d_{ALE\,j})$ j=1,...,num. surface nodes
std_mesh_velocity	$std(u_{ALE\,j})$ j=1,...,num. surface nodes
std_surface_y_plus	$std(y^{+}{}_j)$ j=1,...,num. surface nodes

Variable	**Definition**
std_surface_film_coefficient	$std(h_j)$ j=1,...,num. surface nodes
std_wall_shear_stress	$std(\tau_w u_{wj})$ j=1,...,num. surface nodes
std_density	$std(\rho_j)$ j=1,...,num. surface nodes
std_viscoelastic_stress	$std(\sigma_j)$ j=1,...,num. surface nodes
uni_velocity	$uni(u)$
uni_pressure	$uni(p)$
uni_total_pressure	$uni(p_{\mathrm{TOT}})$
uni_temperature	$uni(T)$
uni_total_temperature	$uni(T_{TOT})$
uni_species	$uni(\varphi)$
uni_field	$uni(F_j)$ j=1,...,num. surface nodes
uni_eddy_viscosity	$uni(\mu_t)$
uni_kinetic_energy	$uni(k)$
uni_eddy_frequency	$uni(\omega)$
uni_sqrt_eddy_period	$uni(\frac{1}{\sqrt{\omega}})$
uni_dissipation_rate	$uni(\varepsilon)$
uni_intermittency	$uni(\gamma)$
uni_transition_re_theta	$uni(\mathrm{Re}_\theta)$
uni_mesh_displacement	$uni(d_{\mathrm{ALE}})$

Variable	**Definition**
uni_mesh_velocity	$uni(u_{\mathrm{ALE}})$
uni_surface_y_plus	$uni(y^+)$
uni_surface_film_coefficient	$uni(h)$
uni_wall_shear_stress	$uni(\tau_w \cdot u_w)$
uni_density	$uni(\rho)$
uni_viscoelastic_stress	$uni(\sigma)$

where the statistics are computed over the nodes defined by the corresponding `SURFACE_OUTPUT` command.

The uniformity index of a variable x is defined as follows:

$$uni(x) = 1 - \frac{1}{2}\frac{\overline{\Delta x}}{|\bar{x}|} \qquad (1)$$

where

$$\bar{x} = \frac{\int x dA}{\int dA}$$

and

$$\overline{\Delta x} = \frac{\int |x - \bar{x}| dA}{\int dA}$$

```
output_radiation_surface
```

The nodal values of a given radiation surface set and a given time step are written as a two dimensional table in an output file named `problem_srfsrf_stepstep.orf`, where `srf` is a surface ID, starting from one, corresponding to each `RADIATION_SURFACE` command in the order given in the input file. The rows of the tables correspond to all the nodes of the surface, numerically sorted. The columns correspond to the translated data, in the order given in the *`output_radiation_surface_vars`* table. Each data type occupies one or more columns, as given in the Fields column of the above table. For example, assume that "wall surface" is the second `RADIATION_SURFACE` command in the input file, then

```
acuTrans -pb channel -orf -orfs "wall surface" -ts 10
```

creates the file `channel_srf2_step10.orf` containing all the output data available for that radiation surface.

```
output_radiation_integral
```

The integrated results of a given radiation surface set are written as a two dimensional table in an output file named `problem_srfsrf.ori`, where `srf` is a surface ID, starting from one, corresponding to each `RADIATION_SURFACE` command in the order given in the input file. The rows of the tables correspond to all the available time steps. The columns correspond to the translated data, in the order given in the *`output_radiation_integral_vars`* table. Each data type occupies one or more columns, as given in the Fields column of the above table. For example, assume that "wall surface" is the second `RADIATION_SURFACE` command in the input file, then

```
acuTrans -pb channel -ori -oris "wall surface" -oriv step,heat
```

creates the file `channel_srf2.ori` containing time step numbers and the surface integral of radiation part of the heat flux, for a total of two columns.

```
output_solar_radiation_surface
```

The nodal values of a given solar radiation surface set and a given time step are written as a two dimensional table in an output file named `problem_srfsrf_stepstep.oqf`, where `srf` is a surface ID, starting from one, corresponding to each `SOLAR_RADIATION_SURFACE` command in the order given in the input file. The rows of the tables correspond to all the nodes of the surface, numerically sorted. The columns correspond to the translated data, in the order given in the *`output_solar_radiation_surface_vars`* table. Each data type occupies one or more columns, as given in the Fields column of the above table. For example, assume that "wall surface" is the second `SOLAR_RADIATION_SURFACE` command in the input file, then

```
acuTrans -pb channel -oqf -oqfs "wall surface" -ts 10
```

creates the file `channel_srf2_step10.oqf` containing all the output data available for that solar radiation surface.

```
output_solar_radiation_integral
```

The integrated results of a given solar radiation surface set are written as a two dimensional table in an output file named `problem_srfsrf.oqi`, where `srf` is a surface ID, starting from one, corresponding to each `SOLAR_RADIATION_SURFACE` command in the order given in the input file. The rows of the tables correspond to all the available time steps. The columns correspond to the translated data, in the order given in the *`output_solar_radiation_integral_vars`* table. Each data type occupies one or more columns, as given in the Fields column of the above table. For example, assume that "wall surface" is the second `SOLAR_RADIATION_SURFACE` command in the input file, then

```
acuTrans -pb channel -oqi -oqis "wall surface" -oqiv step,heat
```

creates the file `channel_srf2.oqi` containing time step numbers and the surface integral of solar radiation part of the heat flux, for a total of two columns.

```
element_integral_output
```

The integrated results of a given element output set are written as a two dimensional table in an output file named `problem_elemelem.oei`, where `elem` is an element output ID, starting from one, corresponding to each `ELEMENT_OUTPUT` command in the order given in the input file. The rows of the tables correspond to all the available time steps. The columns correspond to the translated data, in the order given in the *`element_integral_output_vars`* table. Each data type occupies one or more columns, as given in the Fields column of the above table. For example, assume that "sample one" is the second `ELEMENT_OUTPUT` command in the input file, then

```
acuTrans -pb channel -oei -oeis "sample one" -oeiv time,temp
```

creates the file `channel_elem2.oei` containing run time and the spatially averaged temperature, for a total of two columns.

```
time_history_output
```

The nodal time history of a given time history output set and node is written as a two dimensional table in an output file named `problem_setset_nodenode.oth`, where `set` is a set ID, starting from one, corresponding to each `TIME_HISTORY_OUTPUT` command in the order given in the input file and node is the node number of the set. The rows of the tables correspond to all the available time steps. The columns correspond to the translated data, in the order given in the *`time_history_output_vars`* table. Each data type occupies one or more columns, as given in the Fields column of the above table. For example, assume that nodes 1234 and 5678 appear in the second `TIME_HISTORY_OUTPUT` command in the input file, then

```
acuTrans -pb channel -oth -othn 1234,5678 -othv time,vel
```

creates the files `channel_set2_node1234.oth` and `channel_set2_node5678.oth` containing both the run time and nodal velocities of nodes 1234 and 5678, respectively.

```
fan_component_output
```

The integrated results of a given fan component set are written as a two dimensional table in an output file named `problem_fanfan.ofc`, where the second `fan` is a fan ID, starting from one, corresponding to each `FAN_COMPONENT` command in the order given in the input file. The rows of the tables correspond to all the available time steps. The columns correspond to the translated data, in the order given in the *`fan_component_output_vars`* table. Each data type occupies one or more columns, as given in the Fields column of the above table. For example, assume that "engine fan" is the second `FAN_COMPONENT` command in the input file, then

```
acuTrans -pb vehicle -ofc -ofcs "engine fan" -ofcv step,mass
```

creates the file `vehicle_fan2.ofc` containing time step numbers and the mass flux across the fan surface, for a total of two columns.

```
heat_exchanger_component_output
```

The integrated results of a given heat exchanger component set are given as a two dimensional table in an output file named `problem_hechec.ohc`, where `hec` is a heat exchanger component ID, starting from one, corresponding to each `HEAT_EXCHANGER_COMPONENT` command in the order given in the input file. The rows of the tables correspond to all the available time steps. The columns correspond to the translated data, in the order given in the *heat_exchanger_component_output* table. Each data type occupies one or more columns, as given in the Fields column of the above table. For example, assume that "radiator" is the second `HEAT_EXCHANGER_COMPONENT` command in the input file, then

```
acuTrans -pb vehicle -ohc -ohcs radiator -ohcv step,temp,cool_temp
```

creates the file `vehicle_hec2.ohc` containing time step numbers, area-averaged inlet air temperature, and coolant top water temperature, for a total of three columns.

```
aero_acoustic_output
```

The computational aero-acoustic data of a given CAA output set and a given time step are written as a two dimensional table in an output file named `problem_setset_stepstep.oaa`, where `set` is a set ID, starting from one, corresponding to each `CAA_OUTPUT` command in the order given in the input file. The rows in the table correspond to the sample points, written in the same order given by the corresponding `CAA_OUTPUT` command in the input file. The columns correspond to the translated data, in the order given in the *aero_acoustic_output_vars* table. Each data type occupies one or more columns, as given in the Fields column of the above table. For example, assume that "CAA 2" is the second `CAA_OUTPUT` command in the input file, then

```
acuTrans -pb channel -oaa -oaas "CAA 2" -oaav crd,lighthill_stress -ts 10
```

creates the file `channel_set2_step10.oaa` containing the sample point coordinates and Lighthill stress at step 10 with a total of nine columns. It is not necessary for each set to have the same variables available. For example, a common strategy requires the divergence of the lighthill stress in the volume and the normal component of the divergence of the total stress on the permeable surfaces.

```
output_nodal_residual
```

The nodal residual output of a given time step is written as a two dimensional table in an output file named `problem_stepstep.onr`. The rows in the table correspond to the coordinates node numbers, written in the same order given by the `COORDINATE` command in the input file. The columns correspond to the translated data, in the order given in the *output_nodal_residual_vars* table. Each data type occupies one or more columns, as given in the Fields column of the above table. The following example,

```
acuTrans -pb channel -onr -onrv vel,temp -ts 10
```

creates the file `channel_step10.onr`, which has a total of four columns containing the momentum and heat equation residuals.

```
output_error_estimator
```

The error estimator output of a given time step is written as a two dimensional table in an output file named `problem_stepstep.oee`. The rows in the table correspond to the coordinates node numbers, written in the same order given by the `COORDINATE` command in the input file. The columns correspond to the translated data, in the order given in the *`output_error_estimator_vars`* table. Each data type occupies one or more columns, as given in the Fields column of the above table. The following example,

```
acuTrans -pb channel -oee -oeev vel,temp -ts 10
```

creates the file `channel_step10.oee`, which has a total of four columns containing an error estimate for the momentum and heat equations.

```
time_average_error_estimator
```

The time-averaged error estimator output of a given time step is written as a two dimensional table in an output file `named problem_stepstep.oae`. The rows in the table correspond to the coordinates node numbers, written in the same order given by the `COORDINATE` command in the input file. The columns correspond to the translated data, in the order given in the *`time_average_error_estimator_vars`* table. Each data type occupies one or more columns, as given in the Fields column of the above table. The following example,

```
acuTrans -pb channel -oae -oaev vel,temp -ts 10
```

creates the file `channel_step10.oae`, which has a total of four columns containing a time-averaged error estimate for the momentum and heat equations.

Stats format

For stats format, AcuTrans computes and prints the minimum, maximum and average of each data field. For *`nodal_output`*, *`running_average_output`*,*`time_average_output`*, *`derived_quantity_output`*, *`surface_output`*, *`output_radiation_surface`*, *`output_solar_radiation_surface`*, *`aero_acoustic_output`*, *`output_nodal_residual`*, *`output_error_estimator`*, and *`time_average_error_estimator`* the statistics are gathered over the number of nodes; and for *`surface_integral_output`*, *`surface_statistics_output`*, *`output_radiation_integral`*,*`output_solar_radiation_integral`*, *`element_integral_output`*,*`time_history_output`*, *`fan_component_output`*, and *`heat_exchanger_component_output`* the statistics are gathered over the available time steps. For example, the channel problem

```
acuTrans -verbose 1 -osf -to stats
```

outputs the following to the screen:

```
acuTrans:  Problem = channel
acuTrans:  Run = 1
acuTrans:  Translation format = stats
acuTrans:  Translate to stats = surface output
```

```
acuTrans:  Surface name = wall surface
acuTrans:  Surface ID = 0
acuTrans:  Surface output vars = node_id,mass_flux,momentum_flux,traction
acuTrans:  Process time step = 5
acuTrans:  Variable
acuTrans:  mass_flux
acuTrans:  x_momentum_flux
acuTrans:  y_momentum_flux
acuTrans:  z_momentum_flux
acuTrans:  x_traction
acuTrans:  y_traction
acuTrans:  z_traction
```

```
acuTrans: Total CPU/Elapse time = 2.000000e-02 3.431511e-02 Sec
acuTrans: Total Memory Usage = 3.222275e-02 Mbytes
```

When computing statistics of variables not defined everywhere, usually those associated with *extended_nodal_output*, see *nodal_output* above, it is often useful to also specify *ignore_zeros*. The statistics will then be taken only over those nodes with non-zero values.

CGNS Format

For CGNS format, AcuTrans translates the output data to the (CFD General Notation System) CGNS database file format using version 3.2.1 of the CGNS library. See www.cgns.org. The mesh and *nodal_output* results of the specified time steps are written in a file named `problem.cgns`. For example, the channel problem

```
acuTrans -out -to cgns
```

creates the database file `channel.cgns`.

FieldView Format

For FieldView format, AcuTrans translates the output data to a FieldView unstructured binary file, formats 2.4 or 2.7. FieldView is a visualization package distributed by Intelligent Light. See www.ilight.com. If *fieldview_options* option is set to classical, the *nodal_output* results of each time step along with the nodal coordinates, element connectivity and all surface definitions are written to files named `problem_stepstep.fv`. The surface definitions of each `ELEMENT_BOUNDARY_CONDITION` command are written under the FieldView boundary surface name "EBC: name", where name is the name of the set. Similarly the surface definitions of each `SIMPLE_BOUNDARY_CONDITION`, `PARTICLE_SURFACE`, `SURFACE_OUTPUT`, and `TURBULENCE_WALL` command are written under the FieldView boundary surface name "CBC: name", "PSF: name", "OSF: name", and "TWS: name", respectively. A FieldView region file named `problem_stepstep.fv.fvreg` is also created for each step that is translated. For example, the channel problem

```
acuTrans -out -to fieldview
```

creates the file `channel_step000005.fv`, which contains the nodal coordinates, element connectivity, boundary surfaces "EBC: outflow" and "OSF: wall surface", and the nodal values of velocity and pressure. It also creates the file `channel_step000005.fv.fvreg`.

When *fieldview_options* is set to split, FieldView split file format, version 2.7, is used. Here the mesh is written to a file named `problem_mesh.fv`, while the nodal solution of each time step is written to the file `problem_step.fv`. A FieldView region file named `problem_mesh.fv.fvreg` is also created. For example, the channel problem

```
acuTrans -out -to fieldview -fvopt split
```

creates the files `channel_mesh.fv`, `channel_step000005.fv`, and `channel_mesh.fv.fvreg`. This option eliminates duplications in mesh output. However, you need FieldView 8.2 or later to read these files.

The FieldView region file contains information that helps FieldView to decompose the information in the `.fv` files. It is not required in general but it is necessary for parallel visualization. The parameter *`fieldview_region`* controls how regions are created. A value of single creates a single region, medium creates one region per medium, fluid, solid, shell, and none, element_set creates one region per element set, and domain creates one region per original subdomain.

For simulations that include mesh motion, there are three options for how the position of the mesh is written to disk. The first option is to write the undeformed coordinates. This enables visualization of the solution with the nodes in the original position:

```
acuTrans -out -to fieldview -no_ale
```

The second option is to write the deformed position of the nodes at each time step, and have their location correspond to the position at the end of the time step. This is the result that AcuTrans will produce based on the default settings:

```
acuTrans -out -to fieldview -ale -deformed_crd_type endstep
```

The third option is to write the deformed position of the nodes at each time step, and have their location correspond to the position at the mid point of the time step. This option is useful for investigating the continuity of the flow across non-conformal interfaces.

```
acuTrans -out -to fieldview -ale -deformed_crd_type midstep
```

H3D Format

For H3D format, AcuTrans translates the output data to H3D file format. H3D, or Hyper3D, is a compressed binary file format designed to efficiently store Finite Element model and corresponding result data. H3D files can be visualized by HyperView, a full-featured post-processor, and HyperView Player, a free post-processor viewer.

If the *`h3d_options`* option is set to multi, the nodal coordinates, element connectivity, and all surface definitions are written to a file named `problem.h3d`. The *`nodal_output`* results of each time step are written to separate files named `problem_step.h3d`. For example, the channel problem

```
acuTrans -out -to h3d
```

creates the file `channel.h3d`, which contains the nodal coordinates, element connectivity, and boundary surfaces.In addition, it creates the file `channel_0005.h3d`, which contains the nodal values of velocity and pressure.

When *h3d_options* is set to single, H3D single file format is used. Here both model and result data are written in a single file named `problem.h3d`. For example, the channel problem

```
acuTrans -out -to h3d -h3dopt single
```

creates the file `channel.h3d`, which contains the nodal coordinates, element connectivity, boundary surfaces, and the nodal values of velocity and pressure.

I-deas Format

For I-deas format, AcuTrans translates the output data to the I-deas Universal file format. This is a file format created by Structural Dynamics Research Corporation. The *nodal_output* results of each time step are written in a file named `problem_resstep.unv`. The nodal coordinates and element connectivity of the problem are written in a file named `problem_mesh.unv`, if option *mesh_output* is set. For example, the channel problem

```
acuTrans -mesh -out -to ideas
```

creates the files `channel_res5.unv` and `channel_mesh.unv`.

Spectrum Format

For spectrum format, AcuTrans translates the output data to the Spectrum Visualizer compressed ASCII file format. Spectrum Visualizer is a visualization package created by Centric Engineering Systems, Inc., now part of ANSYS, Inc. The *nodal_output* and *surface_output* results of each time step are written in a file named `problem_resstep.vis.Z`. The nodal coordinates, element connectivity, and surface definitions of the problem are written in a file named `problem_mesh.vis.Z`, if option *mesh_output* is set. The surface definitions are written as two dimensional elements and their corresponding surface output is written as the projected nodal values of these elements. For example, the channel problem

```
acuTrans -mesh -out -osf -to spectrum
```

creates the files `channel_res5.vis.Z` and `channel_mesh.vis.Z`.

EnSight Format

For EnSight format, AcuTrans translates the output data to the EnSight file format. This option creates a number of files. The primary one is `problem.case`, which references the other files. These other files are all contained in the directory `ENSIGHT.DIR`. The *nodal_output* results of each time step are written in files named `problem.var.step`. The nodal coordinates and element connectivity of the problem are written in a file named `problem_mesh.geo`, if option *mesh_output* is set. For example, the channel problem

```
acuTrans -mesh -out -to ensight
```

creates the files `channel.case`, `channel_mesh.geo`, `channel.velocity.000001`, and `channel.pressure.000001`. The file `channel.case` looks like

```
FORMAT
type: ensight
GEOMETRY
model: channel_mesh.geo
```

```
VARIABLE
vector per node: velocity channel.velocity.******
scalar per node: pressure channel.pressure.******
TIME
time set: 1
number of steps: 1
filename numbers:
1
time values:
0.1
```

Actran Format

For Actran format, AcuTrans translates aero-acoustic output data to the HDF5 file format. Only *aero_acoustic_output* is supported. For example:

```
acuTrans -oaa -oaas "CAA 2" -to actran -mesh \
-oaav vel,pres,norm_div_total_stress,div_total_stress,div_lighthill_stress
```

creates the files `coordiantes.hdf` and `res_step.hdf` which are HDF5 files to be read by ActranLA from Free Field Technologies S.A. to perform acoustic propagation calculations. The variables listed in this example are the only ones used by ActranLA so they are the only ones translated. All others are ignored.

AcuRunTrace

Generate particle tracing from the solver solution.

Syntax

`acuRunTrace [options]`

Type

AcuSolve Post-Processing and Co-Processing Program

Description

AcuRunTrace computes particle traces for flows computed by AcuSolve. These include unsteady as well as steady flows, flows with mesh motion as well as without, and flows computed on meshes with interface surfaces. AcuRunTrace can also compute traces for flows on meshes with multiple reference frames in either of two ways: by converting flow velocities to the local reference frame or by treating the boundaries between stationary and rotating reference frames as interface surfaces and the flow as pseudotransient.

AcuRunTrace is actually a script that executes the program AcuTrace in scalar or parallel environments. The program AcuTraceClassic is an older version of AcuTrace with less functionality. Most of the inputs to AcuTrace are provided in a trace input file. The AcuTrace Command Reference Manual describes these inputs. The command line options of AcuRunTrace control the run environment of AcuTrace as well as high level options, such as the name of the trace input file. The program AcuTrace does not ever need to be invoked directly.

There are four types of output that can be generated by AcuRunTrace: trace, time cut, Poincare plane, and interpolate. Output is controlled by commands in the trace input file. Particle traces are computed by AcuRunTrace as a series of segments using fifth-order time-discontinuous Galerkin (TDG) with error control. Trace output records the endpoints of these segments.

Poincare plane output is written only when a particle path crosses through and inside of the Poincare section rectangles. The particle path is interpolated between the segment endpoints that are on either side of the rectangle. Three points define a rectangle as follows: the first two points define one edge of the rectangle, and the third point is projected to the closest plane that is normal to this edge and passes through one of the first two points. The fourth point is constructed to finish the rectangle.

Time cut output is produced for all particles at the time cuts. The particle path is interpolated in time between the segment endpoint steps on either side of the time cut.

Interpolate output does not require any particle tracing. It records the initial seed values of the particle. Interpolate output is a legacy format provided for backward capability.

The other three types of output can be converted to other formats, such as EnSight and FieldView, by the utility AcuTransTrace.

In the following, the full name of each AcuRunTrace option is followed by its abbreviated name and its type. For a general description of option specifications, see *Command Line Options and Configuration Files*. See below for individual option details:

`help` or `h` (boolean)
: If set, the program prints a usage message and exits. The usage message includes all available options, their current values and the source of their current values.

`problem` or `pb` (string)
: The name of the problem is specified via this option. This name is used to build internal file names and to generate output files.

`file_format` or `fmt` (enumerated) [=bin_rec]
: Format of the trace output files.

bin_rec	Binary record format.
ascii	ASCII text format.
binary	Raw binary format.

`trace_input_file` or `tin` (string)
: Name of the trace input file is specified via this option. If *`trace_input_file`* is set to _auto, `name.tin` is used, where `name` is the string supplied to the problem. See the *AcuTrace Command Reference Manual* for trace input file details.

`working_directory` or `dir` (string)
: All internal files are stored in this directory. This directory does not need to be on the same file system as the user-supplied input files.

`problem_directory` or `pdir` (string)
: The home directory of the problem. This is where the user input files reside.

`log_output` or `log` (boolean)
: If set, the printed messages of AcuRunTrace are redirected to the `.log` file `problem.run.tlog`, where `problem` is the string supplied to the problem option and run is the ID of the current trace run. Otherwise, the messages are sent to the screen and no logging is performed.

`append_log` or `append` (boolean)
: If *`log_output`* is set, this parameter specifies whether to create a new log file or append to an existing one.

`echo_help` or `eh` (boolean)
: Echo the command help, that is, the results of acuRunTrace -h, to the `.log` file.

`message_passing_type` or `mp` (enumerated)
: Type of message passing environment used for the parallel processing of a particle tracer:

none	Run in single-processor mode.
impi	Run in parallel on all platforms using Intel MPI.
pmpi	Run in parallel on all platforms using Platform MPI.
mpi	Run in parallel on all platforms using MPICH.
msmpi	Run in parallel on Windows using Microsoft MPI.
hpmpi	Run in parallel on all platforms using Platform MPI.

> **Note:** HP-MPI has been replaced by Platform MPI and this option is only included for backward compatibility. Identical to setting mp=pmpi.

openmp	Run in parallel on all platforms using OpenMP.

num_processors or *np* *(integer)*

This option specifies the total number of threads to be used. The number of physical processors used is *num_processors* divided by *num_threads*. If *message_passing_type* is set to none, *num_processors* is reset to 1.

num_threads or *nt* *(string)*

This option specifies the number of threads per processor to be used. If *num_threads*>1, this option converts an MPI run to a hybrid MPI/OpenMP run. If this option is set to _auto, AcuRunTrace automatically detects the number of cores on the machine and sets the thread count accordingly. This forces AcuTrace to use shared memory parallel message passing whenever possible.

host_lists or *hosts* *(string)*

List of machines (hosts) separated by commas. This list may exceed *num_processors*; the extra hosts are ignored. This list may also contain fewer entries than *num_processors*. In this case, the hosts are populated in the order that they are listed. The process is then repeated, starting from the beginning of the list, until each process has been assigned to a host. Multiple process may be assigned to a single host in one entry by using the following syntax: **host:num_processes**, where **host** is the host name, and **num_processes** is the number of processes to place on that machine. When this option is set to _auto, the local host name is used.

The following examples illustrate the different methods of assigning the execution hosts for a parallel run:

Example 1:

```
acuRunTrace -np 6 -hosts node1,node2
```

Result: The node list is repeated to use the specified number of processors. The processes are assigned: node1, node2, node1, node2, node1, node2

Example 2:

```
acuRunTrace -np 6 -hosts node1,node1,node2,node2,node1,node2
```

Result: The processes are assigned: node1, node1, node2, node2, node1, node2

Example 3:

```
acuRunTrace -np 6 -hosts node1:2,node2:4
```

Result: The processes are assigned: node1, node1, node2, node2, node2, node2

`pbs` *(boolean)*
: Set parallel data, such as np and hosts, from PBS variable PBS_NODEFILE. Used when running AcuTrace through a PBS queue.

`lsf` *(boolean)*
: Set parallel data, such as np and hosts, from LSF variable LSB_HOSTS. Used when running AcuTrace through a LSF queue.

`sge` *(boolean)*
: Set parallel data, such as np and hosts, from SGE file `$TMPDIR/machines`.

`nbs` *(boolean)*
: Set parallel data, such as np and hosts, from NBS var NBS_MACHINE_FILE.

`ccs` *(boolean)*
: Set parallel data, such as np and hosts, from CCS var CCP_NODES.

`slurm` *(boolean)*
: Set parallel data, such as np and hosts, from output of srun hostname -s command.

`acutrace` or `acutrace_executable` *(string)*
: Full path of the AcuTrace executable.

`acu_port` or `acuport` *(integer)*
: Port for establishing connection to AcuSolve.

`acu_wait` or `acuwait` *(integer)*
: The maximum time to wait when trying to establish a connection to AcuSolve.

`flush_freq` or `flush` *(integer)*
: Output flush frequency. Output is flushed after every `flush_freq` segments of the current particle are computed.

`user_libraries` or `libs` *(string)*
: Comma-separated list of user libraries. These libraries contain the user-defined functions and are generated by AcuMakeLib or AcuMakeDll.

`mpirun_executable` or `mpirun` *(string)*
: Full path to mpirun, mpimon, dmpirun or prun executable used to launch an MPI job. If _auto, executable is determined internally.

`mpirun_options` or `mpiopt` *(string)*
: Optional arguments to append to the mpirun command that is used to launch the parallel processes. When this option is set to _auto, AcuRun internally determines the options to append to the mpirun command.

`remote_shell` or `rsh` *(string)*
: Remote shell executable for MPI launchers. Usually this is rsh but can be set to ssh if needed. If _auto, executable is determined internally.

`memv` *(boolean)*
: Print memory allocations.

`automount_path_remove` or `arm` *(string)*
: Automount path remove. If `automount_path_remove` is set to _none, this option is ignored.

automount_path_replace or *arep* *(string)*

Automount path replacement. If *automount_path_replace* is set to _none, this option is ignored.

automount_path_name or *apath* *(string)*

Automount path name. If *automount_path_name* is set to _none, this option is ignored.

lm_daemon or *lmd* *(string)*

Full address of the network license manager daemon, AcuLmd, that runs on the server host. When this option is set to _auto, the value is internally changed to `$ACUSIM_HOME/$ACUSIM_MACHINE/bin/acuLmd`. This option is only used when *lm_service_type* = classical.

lm_service_type or *lmtype* *(enumerated)*

Type of the license manager service:

hwu	Altair Units licensing.
token	Token based licensing.
classical	Classical Acusim style licensing.

lm_server_host or *lmhost* *(string)*

Name of the server machine on which the network license manager runs. When set to _auto, the local host name is used. This option is only used when *lm_service_type* = classical.

lm_port or lmport *(integer)*

TCP port number that the network license manager uses for communication. This option is only used when *lm_service_type* = classical.

lm_license_file or *lmfile* *(string)*

Full address of the license file. This file is read frequently by the network license manager. When this option is set to _auto, the value is internally changed to `$ACUSIM_HOME/$ACUSIM_MACHINE/license.dat`. This option is only used when *lm_service_type* = classical.

lm_checkout or *lmco* *(string)*

Full address of the license checkout co-processor, acuLmco. This process is spawned by the solver on the local machine. When this option is set to _auto, the value is internally changed to `$ACUSIM_HOME/$ACUSIM_MACHINE/bin/acuLmco`.

lm_queue_time or *lmqt* *(integer)*

The time, in seconds, to camp on the license queue before abandoning the wait. This option is only used when *lm_service_type* = classical.

line_buff or *lbuff* *(boolean)*

Flush standard output after each line of output.

print_environment_variables or *printenv* *(boolean)*

Prints all the environment variables in AcuRunTrace and those read by AcuTrace.

verbose or *v* *(integer)*

Set the verbose level for printing information to the screen. Each higher verbose level prints more information. If *verbose* is set to 0, or less, only warning and error messages are printed. If *verbose* is set to 1, basic processing information is printed in addition to warning and error messages. This level is recommended. `verbose` levels of 2 and 3 provide useful information on the progress of the trace. Levels greater than 3 provide information useful only for debugging.

Examples

Given the solution of the steady-state channel problem and a set of coordinates stored in the file `channel.seed`, the particle traces from these points can be computed by first creating the file `channel.tin`:

```
EQUATION {
particle = massless
}
FLOW_FIELD {
problem = "channel"
}
AUTO_SOLUTION_STRATEGY {
}
TRACE_OUTPUT {
active = on
}
PARTICLE_SEED( "seeds" ) {
seed_coordinates = Read("channel.seed")
}
RUN {
}
```

and then executing the command

```
acuRunTrace -tin channel.tin
```

To run the channel problem in parallel on four processors using PMPI, issue the following command:

```
acuRunTrace -tin channel.tin -mp pmpi -np 10
```

Here AcuRunTrace sets the PMPI parallel environment and runs AcuTrace.

To run the channel problem in parallel on a 10-processor shared-memory platform, issue the following command:

```
acuRunTrace -tin channel.tin -mp openmp -np 10
```

Here AcuRunTrace sets the OpenMP parallel environment, and runs AcuTrace.

To run the channel problem in parallel on three processors with two threads per processor using a hybrid MPI/

OpenMP strategy, issue the following command:

```
acuRunTrace -tin channel.tin -mp mpi -np 6 -nt 2
```

Here AcuRunTrace sets up all the MPI related data and the OpenMP parallel environment, and runs AcuTrace.

The options acutrace_executable and mpirun_executable should rarely be changed. They are used to access executables other than the current ones.

Running AcuTrace in distributed memory parallel requires that the problem and working directories be accessible by all involved machines. Moreover, these directories must be accessible by the same name. In general, *`working_directory`* does not pose any difficulty, since it is typically specified relative to the *`problem_directory`*. However, *`problem_directory`* requires special attention.

By default, *`problem_directory`* is set to the current Unix directory ".". AcuRunTrace translates this address to the full Unix address of the current directory. It then performs three operations on this path name in order to overcome some potential automount difficulties. First, if the path name starts with the value of *`automount_path_remove`*, this starting value is removed. Second, if the path name starts with *`from_path`*, it is replaced by *`to_path`*, where the string "from_path,to_path" is the value of *`automount_path_replace`* option. Third, if the path name does not start with the value of *`automount_path_name`*, it is added to the start of the path name. These operations are performed only if *`problem_directory`* is not explicitly given as a command line option to AcuRunTrace, and also if the option associated with each operation is not set to _none. If *`problem_directory`* is explicitly given as a command line option, you are responsible for taking care of all potential automount problems.

The options lm_daemon, lm_server_type, lm_server_host, lm_port, lm_license_file, lm_checkout, and lm_queue_time are used for checking out a license from a network license manager. These options are typically set once by the system administrator in the installed system configuration file, `Acusim.cnf`. Given the proper values, AcuTrace automatically starts the license manager if it is not already running. For a more detailed description of these parameters consult AcuLmg. This executable is used to start, stop, and query the license manager.

The file format for output is specified by the *`file_format`* parameter. The available formats are

- bin_rec
- binary
- ASCII

The bin_rec format is the default. The other two formats are provided for backward compatibility with an older version of AcuTrace. Trace, timecut, and Poincare output written in the bin_rec format can be converted to a number of useful formats, such as FieldView and EnSight, by the AcuTransTrace command. It is recommended that bin_rec format be used when any of these three output types are active. `INTERPOLATE_OUTPUT` is a legacy output type provided for backward capability.

Generally, AcuTrace runs as a post-processor to AcuSolve. However, it is possible to run the two codes concurrently in a coupled manner. There are two AcuRunTrace options that control this co-processing mode. *`acu_port`* specifies the port for the socket connection and overwrites *`coupling_socket_port`* in the `FLOW_FIELD` command in the trace input file. The maximum amount of time to wait for the AcuSolve connection is given by *`acu_wait`*.

AcuRunFwh

Run AcuFwh to propagate the acoustic source computed in AcuSolve to far field observer locations.

Syntax

`acuRunFwh [options]`

Type

AcuSolve Post-Processing Program

Description

AcuRunFwh is used to launch the program AcuFwh, which propagates the acoustic sources computed by AcuSolve to far field microphone locations based on the Ffowcs-Williams-Hawking approach. AcuFwh acts as a post-processor to an AcuSolve run and relies on data written to disk during the simulation. The `FWH_OUTPUT` and `FWH_SURFACE_OUTPUT` AcuSolve input file commands control what data is written to disk.

In the following, the full name of each option is followed by its abbreviated name and its type. See below for more individual option details:

`help` or `h` (boolean)

If set, the program prints a usage message and exits. The usage message includes all available options, their current values, and the place where each option is set.

`problem` or `pb` (string)

The name of the problem is specified via this option. This name is used to build internal file names and to generate output files. All generated output files start with the problem name.

`working_directory` or `dir` (string)

All internal files are stored in this directory. This directory does not need to be on the same file system as the user-supplied input files.

`problem_directory` or `pdir` (string)

The home directory of the problem. This is where the user input files reside.

`run_id` or `run` (integer)

Number of the run in which the translation is requested. If *`run_id`* is set to 0, the last run in the working directory is assumed.

`time_steps` or `ts` (string)

Comma-separated list of time steps to be translated. The comma-separated fields have the general range format **`beg:end:inc`**, where **`:end:inc`** and **`:inc`** are optional. **`beg`** specifies the first time step in the range. It may be either a given time step, as specified by a number, the letter F (or f) requesting the first available time step, or the letter L (or l) requesting the last available time step. end is the last time step in the range. It may be either a time step number or L (or l) requesting the last available time step. If **`end`** is missing, the range is assumed to simply represent a single time step (that is, **`end=beg`** and **`inc=1`**). **`inc`** is the increment that ranges from **`beg`** to **`end`**. It may be either a number or the letter A (or a) requesting all available time steps in the range. If **`:inc`** is missing, it is assumed to be one. The range may also be specified by the single letter A (or a), requesting all available time steps. This is equivalent to **`F:L:A`**.

`ignore_missing_steps` or *`imts`* *(boolean)*
: If set, missing requested time steps are ignored. Otherwise, if the requested time step does not exist, the program issues an error message and exits.

`fwh_output_sets` or *`ofss`* *(string)*
: Comma-separated list of fwh surface output sets to be propagated by AcuFwh. Each item in the list corresponds to the qualifier of a `FWH_SURFACE_OUTPUT` command in the input file.

`time_increment` or *`dt`* *(real)*
: Time increment which is used for acoustic calculations. If zero, the AcuSolve time increment is used.

`sound speed` or *`c`* *(real)*
: The speed of sound that is used for acoustic calculations. The default value is 343.2.

`microphone_file_name` or *`mf`* *(string)*
: The name of the ascii file containing the microphone coordinates and IDs. Each line in the file represents one microphone. The file should have four columns: the first column represents the microphone ID and the other three columns represent the microphone coordinates. The default is `<problem>.mic`.

`symmetry_type` or *`smt`* *(enumerated)*
: Type of symmetry to use.

none	No symmetry is considered.
planar	Planar symmetry is applied.
axisymmetric	Axisymmetry is applied.

If none (default), no symmetry is considered and the acoustic problem is solved in an infinite domain. The values one and two indicate planar symmetry and axisymmetry, respectively. When planar symmetry is used, the sound source can be mirrored with respect to one, two, or three orthogonal planes. Axisymmetry, on the other hand, replicates the sound source about an axis.

`num_symmetry_planes` or *`nsp`* *(integer)*
: This option is used with planar symmetry type and represents the number of symmetry planes than can be one, two, or three. If two or three planes are used, they should be orthogonal.

`symmetry_center` or *`sc`* *(string)*
: This option is used with planar symmetry type, and is the comma-separated coordinates of a reference point on the symmetry plane(s). If more than one plane is used, the reference point should belong to all of the planes.

`symmetry_direction_1` or *`sn1`* *(string)*
: This option is used with planar symmetry type, and is the comma-separated elements of a vector normal to the first plane of symmetry. The default is 1,0,0.

`symmetry_direction_2` or *`sn2`* *(string)*
: This option is used with planar symmetry type, and is the comma-separated elements of a vector normal to the second plane of symmetry. The default is 0,1,0.

`symmetry_direction_3` or `sn3` *(string)*
: This option is used with planar symmetry type, and is the comma-separated elements of a vector normal to the third plane of symmetry. The default is 0,0,1.

`num_axisymmetric_cuts` or `nac` *(integer)*
: This option is used with axisymmetry and indicates the number of axisymmetric replications of the sound source. The default is zero.

`axisymmetric_axis` or `asa` *(string)*
: This option is used with axisymmetry and is the comma-separated coordinates and direction of the axisymmetry axis.

`angular_velocity` *(string)*
: Angular velocity of rotating surfaces, specified as a comma separated list. Used with `fixed_flow` = true.

`rotation_center` *(string)*
: Coordinates of rotation center, specified as a comma separated list. Used with `fixed_flow` = true.

`fixed_flow` *(boolean)*
: This boolean option indicates the rotation of a steady solution to generate the acoustic signal. This option is turned off by default.

`num_steady_rotations` *(integer)*
: Sets the number of rotations of a steady solution. Used with `fixed_flow` = true.

`log_output` or `log` *(boolean)*
: If set, the printed messages of AcuRunFwh are redirected to the log file `problem.run.flog`, where `problem` is the string supplied to the problem option and `run` is the ID of the current trace run. Otherwise, the messages are sent to the screen and no logging is performed.

`append_log` or `append` *(boolean)*
: If `log_output` is set, this parameter specifies whether to create a new `.log` file or append to an existing one.

`echo_help` or `eh` *(boolean)*
: Echo the command help (that is, the results of acuRunFwh -h) to the `.log` file.

`message_passing_type` or `mp` *(enumerated)*
: Type of message passing environment used for the parallel processing of the acoustic propagation:

none	Run in single-processor mode.
impi	Run in parallel on all platforms using Intel MPI.
pmpi	Run in parallel on all platforms using Platform MPI.
mpi	Run in parallel on all platforms using MPICH.
msmpi	Run in parallel on Windows using Microsoft MPI.
hpmpi	Run in parallel on all platforms using Platform MPI.

> **Note:** HP-MPI has been replaced by Platform MPI and this option is only included for backward compatibility. Identical to setting mp=pmpi.

openmp	Run in parallel on all platforms using OpenMP.

num_processors or *np* *(integer)*
: This option specifies the total number of threads to be used. The number of physical processors used is *num_processors* divided by *num_threads*. If *message_passing_type* is set to none, *num_processors* is reset to 1.

num_threads or *nt* *(integer)*
: This option specifies the number of threads per processor to be used. If *num_threads* > 1, it converts an MPI run to a hybrid MPI/OpenMP run.

host_lists or *hosts* *(string)*
: List of machines (hosts) separated by commas. This list may exceed *num_processors*; the extra hosts are ignored. This list may also contain fewer entries than *num_processors*. In this case, the hosts are populated in the order that they are listed. The process is then repeated, starting from the beginning of the list, until each process has been assigned to a host. Multiple process may be assigned to a single host in one entry by using the following syntax: **host:num_processes**, where **host** is the host name, and **num_processes** is the number of processes to place on that machine. When this option is set to _auto, the local host name is used.

The following examples illustrate the different methods of assigning the execution hosts for a parallel run:

Example 1:

```
acuRunFwh -np 6 -hosts node1,node2
```

Result: The node list is repeated to use the specified number of processors. The processes are assigned: node1, node2, node1, node2, node1, node2

Example 2:

```
acuRunFwh -np 6 -hosts node1,node1,node2,node2,node1,node2
```

Result: The processes are assigned: node1, node1, node2, node2, node1, node2

Example 3:

```
acuRunFwh -np 6 -hosts node1:2,node2:4
```

Result: The processes are assigned: node1, node1, node2, node2, node2, node2

pbs *(boolean)*
: Set parallel data (such as np and hosts) from PBS variable PBS_NODEFILE. Used when running AcuFwh through a PBS queue.

`lsf` *(boolean)*
: Set parallel data (such as np and hosts) from LSF variable LSB_HOSTS. Used when running AcuFwh through a LSF queue.

`sge` *(boolean)*
: Set parallel data (such as np and hosts) from SGE file `$TMPDIR/machines`.

`nbs` *(boolean)*
: Set parallel data (such as np and hosts) from NBS var NBS_MACHINE_FILE.

`ccs` *(boolean)*
: Set parallel data (such as np and hosts) from CCS var CCP_NODES.

`slurm` *(boolean)*
: Set parallel data (such as np and hosts) from output of srun hostname -s command.

`acufwh_executable` or `acufwh` *(string)*
: Full path of the AcuFwh executable.

`mpirun_executable` or `mpirun` *(string)*
: Full path to mpirun, mpimon, dmpirun or prun executable used to launch an MPI job. If _auto, executable is determined internally.

`mpirun_options` or `mpiopt` *(string)*
: Optional arguments to append to the mpirun command that is used to launch the parallel processes. When this option is set to _auto, AcuRun internally determines the options to append to the mpirun command.

`remote_shell` or `rsh` *(string)*
: Remote shell executable for MPI launchers. Usually this is rsh but can be set to ssh if needed. If _auto, executable is determined internally.

`automount_path_remove` or `arm` *(string)*
: Automount path remove. If `automount_path_remove` is set to _none, this option is ignored.

`automount_path_replace` or `arep` *(string)*
: Automount path replacement. If `automount_path_replace` is set to _none, this option is ignored.

`automount_path_name` or `apath` *(string)*
: Automount path name. If `automount_path_name` is set to _none, this option is ignored.

`lm_daemon` or `lmd` *(string)*
: Full address of the network license manager daemon, AcuLmd, that runs on the server host. When this option is set to _auto, the value is internally changed to `$ACUSIM_HOME/$ACUSIM_MACHINE/bin/acuLmd`. This option is only used when `lm_service_type` = classical.

`lm_service_type` or `lmtype` *(enumerated)*
: Type of the license manager service:

hwu	Altair Units licensing
token	Token based licensing
classical	Classical Acusim style licensing

`lm_server_host` or `lmhost` *(string)*
: Name of the server machine on which the network license manager runs. When set to _auto, the local host name is used. This option is only used when `lm_service_type` = classical.

`lm_port` or `lmport` *(integer)*
: TCP port number that the network license manager uses for communication. This option is only used when `lm_service_type` = classical.

`lm_license_file` or `lmfile` *(string)*
: Full address of the license file. This file is read frequently by the network license manager. When this option is set to _auto, the value is internally changed to `$ACUSIM_HOME/$ACUSIM_MACHINE/license.dat`. This option is only used when `lm_service_type` = classical.

`lm_checkout` or `lmco` *(string)*
: Full address of the license checkout co-processor, AcuLmco. This process is spawned by the solver on the local machine. When this option is set to _auto, the value is internally changed to `$ACUSIM_HOME/$ACUSIM_MACHINE/bin/acuLmco`. This option is only used when `lm_service_type` = classical.

`lm_queue_time` or `lmqt` *(integer)*
: The time, in seconds, to camp on the license queue before abandoning the wait. This option is only used when `lm_service_type` = classical.

`line_buff` or `lbuff` *(boolean)*
: Flush standard output after each line of output.

`print_environment_variables` or `printenv` *(boolean)*
: Prints all the environment variables in AcuRunFwh and those read by AcuFwh.

`verbose` or `v` *(integer)*
: Set the verbose level for printing information to the screen. Each higher verbose level prints more information. If `verbose` is set to 0 (or less), only warning and error messages are printed. If `verbose` is set to 1, basic processing information is printed in addition to warning and error messages. This level is recommended. `verbose` levels of 2 and 3 provide useful information on the progress of the trace. Levels greater than 3 provide information useful only for debugging.

Examples

AcuFwh relies on the definition of an `FWH_OUTPUT` command and one or more `FWH_SURFACE_OUTPUT` commands from the AcuSolve input file. The `FWH_OUTPUT` command defines the output properties and constants for the acoustic simulation, whereas the `FWH_SURFACE_OUTPUT` command defines the surfaces that are used for integration of the acoustic quantities. For example:

```
FWH_OUTPUT{
reference_density                    = 1.225
reference_pressure                   = 0.0
reference_sound_speed                = 340
source_output_frequency              = 1
source_output_time_interval          = 0
num_source_saved_states              = 0
}
```

defines the Ffowcs-Williams-Hawkings output properties, and

```
FWH_SURFACE_OUTPUT( "FWH_Mirror" ) {
coordinates                         = Read( "MESH.DIR/FWH_Surf_Mirror.crd" )
partial_surfaces                    = Read( "MESH.DIR/FWH_Surf_Mirror.ebc")
shape                               = three_node_triangle
quadrature                          = full
}
```

defines the FWH surface output, where `FWH_Surf_Mirror.crd` and `FWH_Surf_Mirror.ebc` are the coordinate and connectivity files corresponding to the acoustic surface. The connectivity file that is used with the `FWH_SURFACE_OUTPUT` command is for the "partial surfaces", and hence contains only the surface element and node IDs, while the full connectivity file contains the parent volume element IDs as well. As an example, if a line in the full connectivity file of a surface contains the following:

770169 28458 0 18716 18

where the numbers represent parent element ID, surface ID, and the three node IDs, respectively, then the same line in the partial surface connectivity file contains:

28458 0 18716 18

Note: The parent element ID is not present.

The FWH data is written to disk within the working directory of the AcuSolve run as the simulation progresses. Refer to the *AcuSolve Commands Reference Manual* for more details on the `FWH_OUTPUT` and `FWH_SURFACE_OUTPUT` commands.

AcuFwh reads the data that is written to disk, then propagates the acoustic source signal to the far field. For example,

```
AcuRunFwh -np 2 -nt 2 -mp openmp -mf microphones.mic -smt 1 -nsp 1 -sn1 0,1,0 -sc
 0,0,0 -ofss FWH_Mirror
```

processes the `FWH_SURFACE_OUTPUT` called "FWH_Mirror" on two processors using openMP and propagates the results to the microphone locations listed in the file `microphones.mic`. Additionally, the results are mirrored with respect to a plane with a reference point coordinates of 0,0,0 and a normal vector of 0,1,0.

AcuOdb

Translates AcuSolve data to Abaqus CAE in ODB format.

Syntax

`acuOdb [options]`

Type

AcuSolve Post-Processing Program

Description

The results of AcuSolve are stored using an internal format in a number of files in the directory specified by the working_directory option, ACUSIM.DIR by default, and in binary by default. AcuOdb is used to gather and translate these results into Abaqus CAE in ODB format for post-processing or visualizing by Abaqus products. This program is similar to AcuTrans, and the options have the same meaning.

The *`nodal_output_vars`* and *`extended_nodal_output`* parameters are the same as in AcuTrans; see that command for more details.

In the following, the full name of each option is followed by its abbreviated name and its type. For a general description of option specifications, see *Command Line Options and Configuration Files*. See below for more individual option details:

`help` *or* *`h`* *(boolean)*
: If set, the program prints a usage message and exits. The usage message includes all available options, their current values and the source of their current values.

`problem` *or* *`pb`* *(string)*
: The name of the problem is specified via this option. This name is used to build internal file names and to generate output files. All generated output files start with the problem name.

`working_directory` *or* *`dir`* *(string)*
: All internal files are stored in this directory. This directory does not need to be on the same file system as the user-supplied input files.

`run_id` *or* *`run`* *(integer)*
: Number of the run in which the translation is requested. If *`run_id`* is set to 0, the last run in the working directory is assumed.

`time_steps` *or* *`ts`* *(string)*
: Comma-separated list of time steps to be translated. The comma-separated fields have the general range format **`beg:end:inc`**, where **`:end:inc`** and **`:inc`** are optional. **`beg`** specifies the first time step in the range. It may be either a given time step, as specified by a number, the letter F (or f) requesting the first available time step, or the letter L (or l) requesting the last available time step. **`end`** is the last time step in the range. It may be either a time step number or L (or l) requesting the last available time step. If **`end`** is missing, the range is assumed to simply represent a single time step, that is, **`end=beg`** and **`inc=1`**. **`inc`** is the increment that ranges from **`beg`** to **`end`**. It may be either a number or the letter A (or a) requesting all available time steps in the range. If **`:inc`** is missing, it is assumed to be one. The range may also be specified by the

single letter A (or a), requesting all available time steps. This is equivalent to F:L:A. `time_steps` is used only for nodal data and is ignored for time series data. Examples of `time_steps` option include:

```
acuOdb -ts 35          # step 35
acuOdb -ts 35,33,37              # steps 33, 35, and 37
acuOdb -ts 33:37:2               # steps 33, 35, and 37
acuOdb -ts 35,33:37:2,37         # steps 33, 35, and 37
acuOdb -ts 33:37                 # all steps from 33 to 37
acuOdb -ts 33:37:A               # available steps from 33 to 37
acuOdb -ts F:L:A                 # all available steps
acuOdb -ts A                     # all available steps
```

`ignore_missing_steps` or `imts` *(boolean)*
If set, missing requested time steps are ignored. Otherwise, if the requested time step does not exist, the command issues an error message and exits.

`ignore_missing_variables` or `imv` *(boolean)*
If set, missing requested variables are ignored. Otherwise, if the requested variable does not exist, the command issues an error message and exits.

`mesh_output` or `mesh` *(boolean)*
If set, the problem mesh is translated.

`nodal_output_vars` or `outv` *(string)*
Comma-separated list of nodal output variables to be translated. The list may include:

Table 27:

Variable (abbr)	Fields	Description
node_id (node)	1	User-given node number
coordinates (crd)	3	Nodal coordinates
velocity (vel)	3	Velocity vector
pressure (pres)	1	Pressure
temperature (temp)	1	Temperature
incident_radiation	1	Incident radiation
species (spec)	nSpecs	Species
field	nFields	Field values. For *multi_field* = levelset and *multi_field*= algebraic_eulerian, the field values correspond to volume fractions, and are named as *volume_fraction-"fieldname"*.
eddy_viscosity (eddy)	1	Turbulence eddy viscosity

Variable (abbr)	Fields	Description
kinetic_energy (tke)	1	Turbulence kinetic energy
eddy_frequency (tomega)	1	Turbulence eddy frequency
sqrt_eddy_period (tg)	1	Inverse of square root of eddy frequency
dissipation_rate (teps)	1	Turbulence dissipation rate
intermittency (tintc)	1	Turbulence intermittency
transition_re_theta (treth)	1	Critical momentum thickness Reynolds number
surface_y_plus (yp)	1	y+ on turbulence walls
surface_film_coefficient (film)	1	Convective heat transfer coef. on turbulence walls
wall_shear_stress (wall_shear)	3	Wall shear stress on turbulence walls
mesh_displacement (mesh_disp)	3	Mesh displacement vector
mesh_velocity (mesh_vel)	3	Mesh velocity vector

where *nSpecs* is the number of species as given in the `EQUATION` command in the input file. The problem must contain the requested variable in order for it to be translated. For example, the parameter *turbulence* in the `EQUATION` command must be set to a value other than none in order for *eddy_viscosity* to be available. The list of variables is sorted in the order given in the above table. If *nodal_output_vars* is set to _all, all available variables are translated. The *surface_y_plus* and *surface_film_coefficient* are non-zero only on surface nodes given by `TURBULENCE_WALL`, or alternatively by `SIMPLE_BOUNDARY_CONDITION` of type wall. The *surface_film_coefficient* is computed even if there is no temperature equation. However, all relevant fluid material models must include specific heat and conductivity models.

extended_nodal_output or extout (boolean)

Extended nodal output flag. If set, adds to the *nodal_output* variable list available variables from *running_average_output*, *time_average_output*, *derived_quantity_output*, *surface_output*, *radiation_surface*, *solar_radiation_surface*, *output_nodal_residual*, *output_error_estimator*, and *time_average_error_estimator*. The nodal projections of miscellaneous element quantities and gradients of available field variables are also added to the list. Those variables defined only on a subset of the nodes, such as surfaces, are set to zero on the rest of the nodes. The list may include:

Table 28:

Variable (abbr)	Fields	Description
grad_velocity (grad_vel)	9	Gradient of velocity vector
grad_pressure (grad_pres)	3	Gradient of pressure
grad_temperature (grad_temp)	3	Gradient of temperature
grad_field	3*nFields	Gradient of field values. For *multi_field* = levelset and *multi_field* = algebraic_eulerian, the grad field values correspond to grad volume fractions, and are named as *grad_volume_fraction-"fieldname"*.
grad_species (grad_spec)	3*nSpecs	Gradient of species
grad_eddy_viscosity (grad_eddy)	3	Gradient of turbulence eddy viscosity
grad_kinetic_energy (grad_tke)	3	Gradient of turbulent kinetic energy
grad_eddy_frequency (grad_tomega)	3	Gradient of turbulent eddy frequency
grad_sqrt_eddy_period (grad_tg)	3	Gradient of inverse of square root of eddy frequency
grad_dissipation_rate (grad_teps)	3	Gradient of turbulence dissipation rate
grad_intermittency (grad_tintc)	3	Gradient of turbulence intermittency
grad_transition_re_theta (grad_treth)	3	Gradient of critical momentum thickness Reynolds Number
grad_viscoelastic_stress (grad_vest)	18	Gradient of viscoelastic stress
grad_mesh_displacement (grad_mesh_disp)	9	Gradient of mesh displacement vector
grad_mesh_velocity (grad_mesh_vel)	9	Gradient of mesh velocity vector
volume (vol)	1	Nodal volume

Variable (abbr)	Fields	Description
strain_rate_invariant_2 (strain_i2)	1	Second invariant of the strain rate tensor
velocity_magnitude (vmag)	1	Magnitude of the velocity vector
vorticity (vort)	3	Vorticity
cfl_number (cfl)	1	Element-integrated CFL number
density (dens)	1	Density
viscosity (visc)	1	Viscosity
material_viscosity (mat_visc)	1	Molecular viscosity
gravity (grav)	3	Gravity
specific_heat (cp)	1	Specific heat
conductivity (cond)	1	Conductivity
field_diffusivity	1	Field diffusivity
field_source	1	Field source
material_conductivity	1	Material conductivity
total_pressure (tot_pres)	1	Total pressure
heat_source (heat_src)	1	Heat source
turbulence_y (turb_y)	1	Turbulence distance to wall
turbulence_y_plus (turb_yp)	1	Turbulence y+
des_length (deslen)	1	Length scale for DES turbulence model
running_ave_velocity (ora_vel)	3	Running average velocity
running_ave_pressure (ora_pres)	1	Running average pressure
running_ave_temperature (ora_temp)	1	Running average temperature
running_ave_field (ora_field)	nFields	Running average field values. For *multi_field* = levelset and *multi_field* = algebraic_eulerian, the field values correspond to running average volume

Variable (abbr)	Fields	Description
		fractions, and are named as *running_ave_volume_fraction-"fiel*
running_ave_species (ora_spec)	nSpecs	Running average species
running_ave_eddy_viscosity (ora_eddy)	1	Running average eddy viscosity
running_ave_kinetic_energy (ora_tke)	1	Running average turbulence kinetic energy
running_ave_eddy_frequency (ora_tomega)	1	Running average turbulence eddy frequency
running_ave_sqrt_eddy_period (ora_tg)	1	Running average inverse of square root of eddy frequency
running_ave_dissipation_rate (ora_teps)	1	Running average turbulence dissipation rate
running_ave_intermittency (ora_tintc)	1	Running average of the turbulence intermittency
running_ave_transition_re_theta (ora_treth)	1	Running average of the critical momentum thickness Reynolds number
running_ave_mesh_displacement (ora_mesh_disp)	3	Running average mesh displacement
running_ave_viscoelastic_stress (ora_vest)	6	Running average viscoelastic stress
residual_velocity (onr_vel)	3	Residual of momentum equations
residual_pressure (onr_pres)	1	Residual of continuity equation
residual_temperature (onr_temp)	1	Residual of temperature equation
residual_eddy_viscosity (onr_eddy)	1	Residual of turbulence equation
residual_kinetic_energy (onr_tke)	1	Residual of turbulence kinetic energy equation
residual_eddy_frequency (onr_tomega)	1	Residual of turbulence eddy frequency equation
residual_sqrt_eddy_period (onr_tg)	1	Residual inverse of square root of eddy frequency

Variable (abbr)	Fields	Description
residual_dissipation_rate (onr_teps)	1	Residual of turbulence dissipation rate
residual_intermittency (onr_tintc)	1	Residual of the turbulence intermittency
residual_transition_re_theta (onr_treth)	1	Residual of the critical momentum thickness Reynolds number
residual_mesh_displacement (onr_mesh_disp)	3	Residual of mesh displacement equations
error_estimator_volume (oee_vol)	1	Volume
error_estimator_velocity (oee_vel)	3	Error estimate of momentum equations
error_estimator_pressure (oee_pres)	1	Error estimate of continuity equation
error_estimator_temperature (oee_temp)	1	Error estimate of temperature equation
error_estimator_eddy_viscosity (oee_eddy)	1	Error estimate of turbulence equation
error_estimator_tau_velocity (oee_tau_vel)	1	Error estimate of least-squares metric for continuity equation
error_estimator_tau_pressure (oee_tau_pres)	1	Error estimate of least-squares metric for momentum equations
error_estimator_tau_temperature (oee_tau_temp)	1	Error estimate of least-squares metric for heat equation
error_estimator_tau_eddy_viscosity (oee_tau_eddy)	1	Error estimate of least-squares metric for turbulence equations
time_ave_error_volume (oae_vol)	1	Time-averaged volume
time_ave_error_velocity (oae_vel)	3	Time-averaged error estimate of momentum equations
time_ave_error_pressure (oae_pres)	1	Time-averaged error estimate of continuity equation
time_ave_error_temperature (oae_temp)	1	Time-averaged error estimate of temperature equation

Variable (abbr)	Fields	Description
time_ave_error_eddy_viscosity (oae_eddy)	1	Time-averaged error estimate of turbulence equation
time_ave_error_tau_velocity (oae_tau_vel)	1	Time-averaged error estimate of least-squares metric for continuity equation
time_ave_error_tau_pressure (oae_tau_pres)	1	Time-averaged error estimate of least-squares metric for momentum equations
time_ave_error_tau_temperature (oae_tau_temp)	1	Time-averaged error estimate of least-squares metric for heat equation
time_ave_error_tau_eddy_viscosity (oae_tau_eddy)	1	Time-averaged error estimate of least-squares metric for turbulence equation
time_ave_velocity (ota_vel)	3	Time-averaged velocity
time_ave_velocity_regular (ota_vel_reg)	3	Time-averaged non-conservative velocity
time_ave_pressure (ota_pres)	1	Time-averaged pressure
time_ave_pressure_square (ota_pres_sqr)	1	Time-averaged square of pressure
surface_area (osf_area)	1	Surface area
surface_mass_flux (osf_mass)	1	Surface mass flux
surface_momentum_flux (osf_mom)	3	Surface momentum flux
surface_traction (osf_trac)	3	Surface traction
surface_moment (osf_moment)	3	Surface moment
surface_convective_temperature_flux (osf_conv_temp)	1	Surface convective temperature flux
surface_heat_flux (osf_heat)	1	Surface heat flux
radiation_area (orf_area)	1	Radiation area
radiation_heat_flux (orf_heat)	1	Radiation heat flux

Variable (abbr)	Fields	Description
radiation_mean_radiant_temperature (orf_mr_temp)	1	Radiation mean radiant temperature
solar_area (oqf_area)	1	Solar area
solar_heat_flux (oqf_heat)	1	Solar heat flux

where *nSpecs* is the number of species as given in the `EQUATION` command in the input file. The output fields for *time_ave_velocity_square* and *time_ave_stress* are xx, yy, zz, xy, yz, and zx. The output fields for gradient variables are, for example, ux, uy, uz, vx,....

verbose or v (integer)

Set the verbose level for printing information to the screen. Each higher verbose level prints more information. If *verbose* is set to 0, or less, only warning and error messages are printed. If *verbose* is set to 1, basic processing information is printed in addition to warning and error messages. This level is recommended. *verbose* levels greater than 1 provide information useful only for debugging.

Examples

To translate nodal output results of the channel problem to the ODB format, issue the command:

```
acuOdb -pb channel -ts 5
```

which creates suitable ODB files. Alternatively place the option(s) in the configuration file `Acusim.cnf` as follows:

```
problem= channel
time_steps= 5
```

and issue the command:

```
acuOdb
```

AcuTransTrace

Translate acuRunTrace solution binary record output to other formats.

Syntax

`acuTransTrace [options]`

Type

acuRunTrace Post-Processing Program

Description

The results of AcuRunTrace are stored using an internal format in a number of files in the directory specified by the working directory option, `ACUSIM.DIR` by default. These files are written in binary record format by default. AcuTransTrace can gather and translate results in binary record format into various formats more suited for post-processing or visualizing by third party products. For example, to translate the trace output results of the channel problem in order to visualize with FieldView, issue the command:

```
acuTransTrace -pb channel -to fieldview -fvopt streamline,steady
```

which creates the file `channel.trace.fvp` for visualization by FieldView. Alternatively place the option(s) in the configuration file `Acusim.cnf` as follows:

```
problem                 = channel
translate_to            = fieldview
fieldview_options       = streamline,steady
```

and issue the command:

```
acuTransTrace
```

AcuRunTrace generates three types of results: `TRACE_OUTPUT`, `TIME_CUT_OUTPUT` and `POINCARE_OUTPUT`. The corresponding AcuTransTrace options for translating them are streamline, time_cut, and poincare. The term streamline is used here to refer to both true streamline output for steady AcuSolve flows and pathline output for unsteady flows. The results may be translated into one of the following formats: ensight, fieldview, endstats, and poincaretable. In addition, query reports which AcuRunTrace result types are available. The following table lists the supported combinations:

Table 29:

AcuTransTrace option	EnSight	FieldView	Endstats	Poincare Table	Query
streamlin	no	yes	yes	no	yes
time_cut	yes	yes	yes	no	yes

AcuTransTrace option	EnSight	FieldView	Endstats	Poincare Table	Query
poincare	yes	yes	yes	yes	yes

For all formats except query, AcuTransTrace reports the output variables found in the AcuRunTrace results, for example:

```
acuTransTrace:                Outputs are:
acuTransTrace:            velocity_magnitude
acuTransTrace:                    element_id
acuTransTrace:                element_set_id
acuTransTrace:                        marker
acuTransTrace:                      x_length
acuTransTrace:                      y_length
acuTransTrace:                      z_length
acuTransTrace:                    x_velocity
acuTransTrace:                    y_velocity
acuTransTrace:                    z_velocity
```

Note: Particle position is required by all the translation formats and therefore is not listed. Particle time is only listed for the poincaretable and Poincare EnSight formats; it is required by all the other formats and therefore not listed for those.

For a general description of option specifications, see *Command Line Options and Configuration Files*. See below for individual option details.

`help` or `h` *(boolean)*
: If set, the program prints a usage message and exits. The usage message includes all available options, their current values and the source of their current values.

`problem` or `pb` *(string)*
: The name of the problem is specified via this option. This name is used to build internal file names and to generate output files.

`working_directory` or `dir` *(string)*
: All internal files are stored in this directory. This directory does not need to be on the same file system as the user-supplied input files.

`run_id` or `run` *(integer)*
: Number of the AcuSolve run used by the trace run. If `run_id` is set to 0, this option is ignored. This parameter needs to explicitly set only in rare cases.

`trace_run_id` or `trun` *(integer)*
: Number of the AcuRunTrace run to translate. If `trace_run_id` is set to 0, the last trace run in the working directory is assumed.

`translate_to` or `to` *(enumerated)*
: Translate the AcuRunTrace output to this format:

query	Print information about available AcuRunTrace result types.

ensight	Translate to EnSight Gold format.
fieldview	Translate to FieldView format.
endstats	Report the end point statistics, that is, the reasons the trace of each particle ended.
poincaretable	Translate Poincare results into an AcuTraceClassic style Poincare table.

`num_threads` or `nt` *(integer)*
Number of threads to use.

> **Note:** MPI is not available, so there is no *`np`* option. This option is available only with *`translate_to`* = ensight.

`fieldview_options` or `fvopt` *(string)*
A comma separated list of FieldView translation options. The list may include:

streamline	Use streamline (pathline) results if available.
time_cut	Use timecut results if available.
poincare	Use Poincare results if available.
pseudotc	Use pseudo timecut values for time with Poincare results.
ascii	Convert to FieldView ASCII Particle Path format. This format is for use with steady-state flows and is a synonym for steady.
binary	Convert to FieldView BINARY Particle Path format. This format is for use with transient flows and is a synonym for unsteady.

`ensight_options` or `ensopt` *(string)*
A comma separated list of EnSight translation options. The list may include:

time_cut	Use timecut results if available.
poincare	Use Poincare results if available.
pseudotc	Use pseudo timecut value for time with Poincare results.
binary	Write binary EnSight files.
ascii	Write ASCII EnSight files.
nomesh	Do not write the AcuSolve mesh file. This can be a run time savings if the mesh file exists and is current.

`end_stat_options` or `endopt` *(string)*
A comma separated list of end point statistics translation options. The list may include:

streamline	Use streamline (pathline) results if available.

time_cut	Use timecut results if available.
poincare	Use Poincare results if available.

`poincaretable_options` or `ptbopt` (string)

A comma separated list of Poincare table translation options. The list may include:

ordered	Sort the table by seed number and Poincare plane number. This can take a while.
unordered	Do not sort the table.

`verbose` or `v` (integer)

Set the verbose level for printing information to the screen. Each higher verbose level prints more information. If *`verbose`* is set to 0, or less, only warning and error messages are printed. If *`verbose`* is set to 1, basic processing information, including the output variables found, is printed in addition to warning and error messages. This level is recommended.

Examples

For query format, AcuTransTrace simply reports the available AcuRunTrace result types and which AcuSolve run was used. For example, if the output of

```
acuTransTrace -to query
```

is

```
acuTransTrace: -----------------------------------------------------------------
acuTransTrace:                        Date = Tue Aug 30 20:54:35 2011
acuTransTrace:                     Problem = mixer
acuTransTrace:                    Flow Run = 1
acuTransTrace:                   Trace Run = 8
acuTransTrace:        Streamline trace type = Available
acuTransTrace:           Timecut trace type = Available
acuTransTrace:          Poincare trace type = Available
acuTransTrace:                    Hostname = COOT
acuTransTrace:                    Platform = Whistler 5.1 i686
acuTransTrace:                     Machine = WIN
acuTransTrace:                     Release = 1.8a
acuTransTrace:                Release date = Aug 30 2011
acuTransTrace: -----------------------------------------------------------------
```

then the last AcuRunTrace trace run ID is 8 and all three possible result types are available for translation. If the output of

```
acuTransTrace -to query -trun 1
```

includes

```
acuTransTrace:        Streamline trace type = Available
acuTransTrace:           Timecut trace type = Available
acuTransTrace:          Poincare trace type = Unvailable
```

then streamline and timecut results are available for *`trace_run_id`* = 1, but Poincare results are not.

For fieldview format, AcuTransTrace translates the output to either FieldView ASCII Particle Path or Binary Particle Path format. The ASCII format is specified by the steady option, while binary is specified by the unsteady option. For either option, the file is named `<problem>.trace.fvp`. streamline (pathline) and poincare output can be converted to ASCII format only. The conversion of poincare output uses the time values associated with the plane hits. If pseudotc is set, Poincare plane numbers are converted to time values, and these times are used instead of the actual times. In the case of multiple plane hits, the pseudo time cut value is given by

```
(hit_number - 1) * number_of_planes + plane_number
```

To convert trace output to FieldView format, run

```
acuTransTrace -to fieldview -fvopt steady,streamline
```

Alternatively, the ascii option can be used as a synonym for steady:

```
acuTransTrace -to fieldview -fvopt ascii,streamline
```

To convert time cut output to FieldView format, run

```
acuTransTrace -to fieldview -fvopt steady,time_cut
```

or

```
acuTransTrace -to fieldview -fvopt unsteady,time_cut
```

which is also equivalent to

```
acuTransTrace -to fieldview -fvopt binary,time_cut
```

To convert Poincare output to FieldView format, run

```
acuTransTrace -to fieldview -fvopt poincare
```

or

```
acuTransTrace -to fieldview -fvopt poincare,pseudotc
```

For ensight format, AcuTransTrace translates the output to EnSight Gold format. This option creates a number of files. The primary one is `problem.trace.case`, which references the other files. These other files are all contained in the directory `ENSIGHT.DIR`. With the exception of particle position, the particle and flow outputs are written in files named `problem.varname.nnnnn`, where varname is the variable name and nnnnn is the time cut or Poincare plane number; particle position outputs are written in files named `problem.mgeo.nnnnn`. For example, if AcuRunTrace wrote time cut output for two time cuts for *`particle_coordinates`* and *`particle_time`*,

```
acuTransTrace -to ensight -ensopt ascii,time_cut
```

creates the file `mixer.trace.case` and the ASCII files `mixer_mesh.geo`, `mixer.mgeo.00001`, `mixer.mgeo.00002`, `mixer.time.00001`, and `mixer.time.00002`. The file `mixer.trace.case` looks like:

```
#===============================================
# EnSight Gold Case file for particle
#
# Problem        : mixer
#
# Generated by   : acuTransTrace
# Release        : 1.8a
# Release date   : Aug 30 2011
# Date           : Tue Aug 30 18:35:21 2011
#
#================================================
FORMAT
type: ensight gold
GEOMETRY
model: ENSIGHT.DIR/mixer.mesh.geo
measured: ENSIGHT.DIR/mixer.mgeo.*****
VARIABLE
scalar per measured node: time ENSIGHT.DIR/mixer.time.*****
TIME
time set: 1
number of steps: 2
filename numbers: 1 2
time values: 0 0.0001
```

Time cut or Poincare output can be converted to EnSight format. Since a particle may hit a Poincare plane multiple times, a given seed can appear multiple times in a single EnSight file. If the pseudotc option is used, however, Poincare plane numbers converted to time values, and so a seed does only appear once in a single EnSight file. In the case of multiple plane hits, the pseudo time cut value is again given by the expression described above in the FieldView discussion.

To convert Poincare output to EnSight format, then, use either

```
acuTransTrace -to ensight -ensopt poincare,binary (or ascii)
```

or

```
acuTransTrace -to ensight -ensopt poincare,pseudotc,binary (or ascii)
```

For endstats format, AcuTransTrace creates a text file `problem.trace.end`. The file consists of a table, one line per particle, listing the possible reasons why the particle trace ended.

For example, if the AcuRunTrace run for problem mixer produced Poincare output, then

```
acuTransTrace  -to endstats -endopt poincare
```

produces the file `mixer.trace.end`. All end stats files have 19 columns as follows:

Table 30:

Column number	**Column heading**	**Description**
1	Seed	Global seed number
2	X	X coordinate of particle position
3	Y	Y coordinate of particle position

Column number	Column heading	Description
4	Z	Z coordinate of particle position
5	Time	Particle time
6	Speed	Magnitude of particle velocity
7	Lx	X coordinate of particle stretch vector, or 1 if no stretch equation
8	Ly	Y coordinate of particle stretch vector, or 0 if no stretch equation
9	Lz	Z coordinate of particle stretch vector, or 0 if no stretch equation
10	ElmId	Element set ID number (1-based)
11	ElemId	Element ID (1-based)
12	Out	1 if particle left domain through an outflow face, 0 otherwise
13	Wall	1 if particle stopped at a wall surface; 0 otherwise
14	Node	1 if particle stopped at a wall node; 0 otherwise
15	Ssi	1 if particle stopped at an interface surface; 0 otherwise
16	Rff	1 if particle stopped at a reference frame boundary; 0 otherwise
17	Time	1 if particle time exceeded maximum trace time; 0 otherwise
18	Tcut	1 if particle hit the last time cut and only time cut output requested; else 0
19	Segs	1 if number of particle segments exceeds maximum; 0 otherwise

For columns 12 to 19, more than one column can contain a "1" because the table lists all the possible reasons why the particle trace ended.

Note: One is not a possible value for Rff if the AcuRunTrace `FLOW_FIELD` type is pseudotransient.

For poincaretable format, AcuTransTrace creates a text file `problem.pps`. The file consists of a table, each line containing a single Poincare plane/particle pair. A Poincare plane/particle pair can appear multiple times. The first five columns of the table are always:

- global particle number
- Poincare plane number
- x,y,z coordinates of the intersection of the particle path line with the plane

The contents of the remaining columns are reported by AcuTransTrace. For example, if

```
acuTransTrace -to poincaretable
```

reports

```
acuTransTrace: ----------------------------------------------------------------
acuTransTrace:                 Outputs are:
acuTransTrace:                         time
acuTransTrace:                   element_id
acuTransTrace:               element_set_id
acuTransTrace:                   x_velocity
acuTransTrace:                   y_velocity
acuTransTrace:                   z_velocity
acuTransTrace: ----------------------------------------------------------------
```

then columns 6-11 of `problem.pps` are time, the element set and element IDs, and the particle velocity.

AcuFv2H3D

Translate a FieldView binary data file to a H3D format file.

Syntax

`acuFv2H3D[options]`

Type

AcuSolve Post-Processing Program

Description

In the following, the full name of each option is followed by its abbreviated name and its type. See the description below for more details.

`help` or *`h`* *(boolean)*
: If set, the program prints a usage message and exits. The usage message includes all available options, their current values and the source of their current values.

`problem` or *`pb`* *(string)*
: The name of the problem is specified via this option.

`working_directory` or *`dir`* *(string)*
: All fieldview files are stored in this directory.

`run_id` or *`run`* *(integer)*
: The run number in which the translation is requested. If *`run_id`* is set to 0, the last run in the working directory is assumed.

Examples

AcuFv2H3D is used to translate fieldview data files of the channel problem to `.h3d` data files by issuing the following command. The converted `.h3d` files can be imported into HyperMesh CFD and SimLab for visualization.

```
acuFv2H3D -pb channel
```

AcuTherm

Translates the AcuSolve solution to an input file for TAITherm.

Syntax

`acuTherm [options]`

Type

AcuSolve Post-Processing Program

Description

In the following, the full name of each option is followed by its abbreviated name and its type. See the description below for more details.

`help` or *`h`* *(boolean)*
: If set, the program prints a usage message and exits. The usage message includes all available options, their current values and the source of their current values.

`problem` or *`pb`* *(string)*
: The name of the problem is specified via this option. This name is used to build internal file names and to generate output files.

`working_directory` or *`dir`* *(string)*
: All internal files are stored in this directory. This directory does not need to be on the same file system as the user-supplied input files.

`run_id` or *`run`* *(integer)*
: The run number in which the translation is requested. If *`run_id`* is set to 0, the last run in the working directory is assumed; see the description below for more details.

`translate_to` or *`to`* *(enumerated)*
: Translate the output to this format:

taitherm	Translate into a Patran file for TAITherm
jmag	Translate into a CSV format for JMAG
abaqus	Translate into a CSV format for Abaqus or general
csv	Translate into a CSV format for TAITherm or general

`time_steps` or *`ts`* *(string)*
: Comma-separated list of time steps to be translated. The comma-separated fields have the general range format beg:end:inc, where :end:inc and :inc are optional. beg specifies the first time step in the range. It may be either a given time step, as specified by a number, or the letter L (or l) requesting the last time step, or the letter A (or a) requesting all available time steps in the range.

`ignore_missing_steps` or *`imts`* *(boolean)*
: If set, missing requested time steps are ignored. Otherwise, if the requested time step does not exist, the command issues an error message and exits.

`surface_output_sets` or `osfs` *(string)*
: Comma-separated list of surface_output sets. These are the user-given names specified as the user given name of the `SURFACE_OUTPUT` commands in the input file. If `surface_output_sets` is set to _auto, all output sets are used.

`min_film_coefficient` or `mfc` *(real)*
: Minimum acceptable film coefficient (default, 0). This value provides a lower bound on the `surface_film_coefficient`. Any values in the AcuSolve solution database that fall below this value are clipped. This option can be used to eliminate high reference temperatures that result from low film coefficients in regions of stagnant flow.

`num_temperature_filter_passes` or `ntfp` *(real)*
: Number of passes for smoothing the temperature field.

`num_reference_temperature_filter_passes` or `nrfp` *(real)*
: Number of passes for smoothing the reference temperature.

`num_film_filter_passes` or `nffp` *(real)*
: Number of passes for smoothing the film coefficient.

`num_heat_filter_passes` or `nhfp` *(real)*
: Number of passes for smoothing the heat flux field.

`film_calculation_type` or `fct` *(enumerated)*
: Heat transfer coefficient (HTC) calculation methods using acuTherm: direct, user_ref_temp, turb_wall.

  direct
  : Compute the surface film coefficient by evaluating the nodal temperature at the surface as well as at some point at a specified distance away from the surface. This option utilizes `thermal_layer_edge_y_plus` to determine where in the domain to evaluate the reference temperature. This option requires that `NODAL_OUTPUT`, `DERIVED_QUANTITY_OUTPUT` and surface nodal output be available for all time steps of interest at each surface of interest.

  user_ref_temp
  : Compute the surface film coefficient by using a user specified constant reference temperature. The reference temperature is provided by `user_reference_temperature`. This option requires that `NODAL_OUTPUT` and surface nodal output be available for all time steps of interest at each surface of interest.

  turb_wall
  : Utilize the "Thermal Law of the Wall" to determine the surface film coefficient. This option requires that `NODAL_OUTPUT` and surface nodal output be available for all time steps of interest at each surface of interest.

`thermal_layer_edge_y_plus` or `tlyp` *(real)*
: Distance away from the wall (in non-dimensional coordinates) where the solution is sampled to determine the local reference temperature. Used with `film_calculation_type`=direct.

`user_reference_temperature` or `urt` *(real)*
: Constant reference temperature to use for evaluation of the film coefficient. Used with `film_calculation_type`=user_ref_temp.

`limit_reference_temperature` or *`lrt`* *(boolean)*

Flag specifying whether the reference temperature should be limited to the min and max values present in the nodal solution. Used with *`film_calculation_type`* = turb_wall.

`verbose` or *`v`* *(integer)*

Set the verbose level for printing information to the screen. Each higher verbose level prints more information. If verbose is set to zero (or less), only warning and error messages are printed. If verbose is set to one, basic processing information is printed in addition to warning and error messages. This level is recommended. Verbose levels greater than one provide additional information useful only for debugging.

AcuTherm translates the solution of a problem run by AcuSolve into either a Patran Neutral file suitable for running TAITherm, or a comma separated value (CSV) file suitable for use with JMAG, Abaqus or for other thermal associated applications.

For example, given the solution of the problem underhood, with surface output sets "surface 1" and "surface 2", AcuTherm may be used to translate the output to a Patran Neutral file `underhood.ntl`, by issuing the command:

```
acuTherm -pb underhood -osfs surface 1, surface 2 -to taitherm
```

or alternatively by placing the options in the configuration file `Acusim.cnf` as

```
problem = underhood
surface_output_sets = surface 1, surface 2
to = taitherm
```

and invoking AcuTherm as:

```
acuTherm
```

The created Patran Neutral file, `underhood.ntl`, contains:

- Card 25: Title Card
- Card 26: Summary Data
- Card 01: Nodal Data
- Card 02: Element Data
- Card 17: Convection Coefficients
- Card 18: Element Flow Data

The Nodal Data cards include only the nodes on the surfaces. The Element Data cards include all the output surfaces. The Convection Coefficients and Element Flow Data cards have the format defined by TAITherm. AcuTherm requires nodal output as well as surface nodal output for all surfaces involved. The former may be requested through the `NODAL_OUTPUT` command in the input file, while the latter is specified through the parameters *`nodal_output_frequency`* or *`nodal_output_time_interval`* of the `SURFACE_OUTPUT` command. For the case of film_calculation_type=direct, AcuTherm also requires that `DERIVED_QUANTITY_OUTPUT` be present. The options run_id and time_step specify the output run and time step for extracting the data. Either or both options may be set to 0, indicating the use of the latest data.

AcuSolve Thermal Wall Function - function of local surface y+ and local Prandtl number

$$T^{+} = \mathrm{Pr}y^{+}e^{-L} + (2.12ln(1+y^{+}) + \beta)e^{-1/L} \quad L = [0.01(\mathrm{Pr}y^{+})^{4}]/[1+5Pr^{3}y^{+}] \quad \beta = (3.85\mathrm{Pr}^{1/3} - 1.3)^{2} + 2.1 \tag{2}$$

The local Prandtl Number (Pr) is computed as a function of local material viscosity (μ), specific heat (C_p) and conductivity (k). $\mathrm{Pr} = \mu * C_p / k$.

AcuTherm provides three options (film_calculation_type) to compute the Heat Transfer Coefficient (HTC).

film_calculation_type (fct) = user_ref_temp

AcuTherm calculates the heat transfer coefficient by using a user-specified constant reference temperature. The reference temperature is provided by *user_reference_temperature*. This option requires that NODAL_OUTPUT and surface nodal output be available for all time steps of interest at each surface of interest.

$$htc_1 = Q / (T_{ref} - T_{wall}) \tag{3}$$

Q = local surface heat flux (in AcuSolve, if the solid surface is losing heat, then the sign of Q is positive for the solid surface and it will be negative for the fluid surface at the same location.)

T_{wall} = local wall temperature

T_{ref} = user-specified reference temperature (must be smaller than min[T_{wall}])

Use htc1 when comparing CFD with the experimental htc value computed using the same expression above. This is recommended when there is a well-defined reference freestream/bulk temperature.

film_calculation_type (fct) = direct

AcuTherm calculates the heat transfer coefficient by evaluating the nodal temperature at the surface as well as at some point at a specified distance away from the surface. This option utilizes thermal_layer_edge_y_plus to determine where in the domain to evaluate the reference temperature. This option requires that NODAL_OUTPUT, DERIVED_QUANTITY_OUTPUT and surface nodal output be available for all time steps of interest at each surface of interest.

$$htc_2 = Q / (T_{ref\_local} - T_{wall}) \tag{4}$$

Q = local surface heat flux (in AcuSolve, if the solid surface is losing heat, then the sign of Q is positive for the solid surface and it will be negative for the fluid surface at the same location.)

T_{wall} = local wall temperature

T_{ref_local} = local ref. temperature computed by AcuTherm at user-specified thermal layer edge yplus

thermal_layer_edge_yplus = default value is 100.0

Use htc2 when there is a well-defined reference freestream temperature in the streamwise direction.

film_calculation_type (fct) = turb_wall

AcuTherm utilizes the "Thermal Law of the Wall" to determine the surface film coefficient. This option requires that `NODAL_OUTPUT` and surface nodal output be available for all time steps of interest at each surface of interest.

$htc_3 = (\rho * C_p * U_\tau) / T^+(y^+)$

ρ = local wall density

C_p = local wall specific heat

$U_\tau = sqrt(\tau_{wall} / \rho)$ = local friction velocity

$T^+(y^+)$ = local non-dimensional temperature at user-specified y+ (recommended y+ = 100)

For example: Pr = 0.72 (air), T+(y+=100) = 13.7

AcuLiftDrag

Extract the coefficient of lift and the coefficient of drag for one or more surfaces.

Syntax

`acuLiftDrag [options]`

Type

AcuSolve Post-Processing Program

Description

AcuLiftDrag is a post-processing utility that uses the AcuSolve database of surface integrated traction forces to compute the normalized lift and drag coefficients for one or more surfaces within the solution database.

The selected surfaces, determined by the osis flag, are queried and selected during the execution of the application. The surface integrated traction forces from those surfaces are then summed to provide the total force for each vector component. That vector is then decomposed into the relative lift and drag components according to the *`angle_of_attack`* flag and *`drag_direction`*/*`lift_direction`* column specifier flags. The following relationship is used to determine the decomposed lift and drag magnitude, before normalization.

$$lift = F_l * \cos\alpha - F_d * \sin\alpha \tag{5}$$

$$drag = F_d * \cos\alpha + F_l * \sin\alpha \tag{6}$$

Where, F_l is the force in the cardinal direction perpendicular to the inflow at zero angle of attack, F_d is the force in the cardinal direction parallel to the inflow at zero angle of attack and α is the angle of attack in degrees of the surface in question.

Once the summed lift and drag components of the force vector are found, they are each normalized according to the following relationship.

$$\begin{gathered} C_l = \frac{lift}{0.5 * chord * span * \rho * U_\infty^2} \\ C_d = \frac{drag}{0.5 * chord * span * \rho * U_\infty^2} \end{gathered} \tag{7}$$

Where, the chord and span are defined by the *`wing_chord`* flag and *`wing_span`* flag. These quantities are arbitrarily used to define the total surface area of the body in question. Lastly, ρ is the fluid density and U_∞ is the freestream velocity, defined by density and *`reference_velocity`*, respectively.

In the following, the full name of each option is followed by its abbreviated name and its type. For a general description of option specifications, see *Command Line Options and Configuration Files*. See below for more individual option details:

`help` or `h` *(boolean)*
: If set, the program prints a usage message and exits. The usage message includes all available options, their current values, and the place where each option is set.

`problem` or `pb` *(string)*
: The name of the problem is specified via this option. This name is used to generate input file names and extracted surface file names.

`working_directory` or `dir` *(string)*
: All internal files are stored in this directory. This directory does not need to be on the same file system as the user-supplied input files.

`run_id` or `run` *(integer)*
: Number of the run in which the translation is requested. If `run_id` is set to 0, the last run in the working directory is assumed.

`surface_integral_output_sets` or `osis` *(string)*
: Comma-separated list of surface_outputs to sum (support Unix style wildcard expressions). These are the user-given names specified as the user-given name of the `SURFACE_OUTPUT` commands in the input file. If `surface_integral_output_sets` is set to _all, all output sets are included into the force computation.

`wing_chord` or `chord` *(real)*
: This option specifies the reference chord of the surface in question. If the term chord does not apply to the simulation, this option can be considered one dimension of the frontal area. The value is used to compute the reference area with chord*span, which is then used to normalize the force values into force coefficients.

`wing_span` or `span` *(real)*
: This option specifies the reference span of the surface in question. If the term span does not apply to the simulation, this option can be considered one dimension of the frontal area. The value is used to compute the reference area with chord*span, which is then used to normalize the force values into force coefficients.

`angle_of_attack` or `aoa` *(real)*
: This option specifies the angle of attack of the incoming fluid or surface in question. The value is used to decompose the cartesian traction vector quantities into fluid aligned lift and drag components.

`lift_direction` or `lift_dir` *(enumerated)*
: This option specifies the direction of lift at zero angle of attack. You can specify values of x, y, or z referring to the cartesian coordinate system associated with the inflow direction.

`drag_direction` or `drag_dir` *(enumerated)*
: This option specifies the direction of drag at zero angle of attack. You can specify values of x, y, or z referring to the cartesian coordinate system associated with the inflow direction.

`reference_density` or `ref_rho` *(real)*
: This option specifies the reference density used to normalize the force values into force coefficients.

`reference_velocity` or `ref_vel` *(real)*
: This option specifies the reference velocity used to normalize the force values into force coefficients. It should be specified as the magnitude of the inflow velocity.

`x_variable` or `x_vars` *(enumerated)*
: This option specifies the x axis variable for the exported data: step, time.

`query` or `q` *(boolean)*
: If this option is set to TRUE, the application will run in query mode. Query mode will simply print a list of surfaces that are available for force computation. The list of surfaces is queried from the list of `surface_output` sets available in the solution database.

`verbose` or `v` *(integer)*
: Set the verbose level for printing information to the screen. Each higher verbose level prints more information. If `verbose` is set to 0 (or less), only warning and error messages are printed. If `verbose` is set to 1, basic processing information is printed in addition to warning and error messages. This level is recommended. `verbose` levels greater than 1 provide information useful only for debugging.

Examples

Consider the computation of the lift and drag force coefficients of a simple two-dimension airfoil:

```
acuLiftDrag -osis "airfoil surface" -reference_velocity 14.5387 -wing_chord 1.0 -
wing_span 50.0
```

or alternatively the options may be put into the configuration file `Acusim.cnf` as follows and then type `acuLiftDrag` in the AcuSolve cmd prompt (Windows) or the Linux command line to execute:

```
acuLiftDrag.pb=airfoil
acuLiftDrag.osis="airfoil surface"
acuLiftDrag.density=1.0
acuLiftDrag.wing_span=50
acuLiftDrag.aoa=0
acuLiftDrag.reference_velocity=14.5387755102
```

The following will be printed to the standard output of the terminal:

```
acuLiftDrag:
acuLiftDrag:  Opening the AcuSolve solution data base
acuLiftDrag:  Problem <airfoil> directory <ACUSIM.DIR> runId <1>
acuLiftDrag:
acuLiftDrag:  Opened run id <1>
acuLiftDrag:
acuLiftDrag:  Processing surface output variable:  traction
acuLiftDrag:    Processing surface output:  airfoil surface
acuLiftDrag:    Writing file: airfoil.liftDrag.dat
acuLiftDrag:
acuLiftDrag:    C_l - final: -0.00181443659229
acuLiftDrag:      C_l - min: -0.0247666844194
acuLiftDrag:      C_l - max: 0.016792946081
acuLiftDrag:      C_l - ave: 0.000865548912408
acuLiftDrag:      C_l'- rms: 8.97872604896e-159
acuLiftDrag:
acuLiftDrag:    C_d - final: 0.00599087791896
acuLiftDrag:      C_d - min: 0.00575298069594
acuLiftDrag:      C_d - max: 0.515670407393
```

```
acuLiftDrag:      C_d - ave: 0.011099596032
acuLiftDrag:      C_d'- rms: 0.0381130336459
acuLiftDrag:
acuLiftDrag:  Total elapsed time <0.7324> seconds
acuLiftDrag:
```

The above example produces a single output file containing the simulated series of lift and drag coefficients called `airfoil.lifDrag.dat`.

The file contains the following columns:

timestep lift coefficient drag coefficient

100 -2.5207876546848220e-03 6.0041848600869774e-03

For a steady state simulation, only the last value is likely of interest to you, as it would be the converged solution for a Reynolds Averaged Navier-Stokes simulation. For a transient simulation the entire file may be useful to you, as this represents the time dependent lift/drag coefficients as calculated by AcuSolve.

To simply query the solution database, use the following command:

```
acuLiftDrag -q
acuLiftDrag:
acuLiftDrag:  Opening the AcuSolve solution data base
acuLiftDrag:  Problem <airfoil> directory <ACUSIM.DIR> runId <1>
acuLiftDrag:
acuLiftDrag:  Opened run id <1>
acuLiftDrag:
acuLiftDrag:  Surface output name:                  +z slip
acuLiftDrag:       Number of surface nodes:         0
acuLiftDrag:       Number of integrated output steps: 300
acuLiftDrag:  Surface output name:                  -z slip
acuLiftDrag:       Number of surface nodes:         0
acuLiftDrag:       Number of integrated output steps: 300
acuLiftDrag:  Surface output name:                  airfoil surface
acuLiftDrag:       Number of surface nodes:         2616
acuLiftDrag:       Number of integrated output steps: 300
acuLiftDrag:  Surface output name:                  farfield
acuLiftDrag:       Number of surface nodes:         84
acuLiftDrag:       Number of integrated output steps: 300
```

AcuHeatBalance

Compute the total surface integrated heat flux, surface integrated convective temperature flux and integrated radiation heat flux from an AcuSolve solution database.

Syntax

`acuHeatBalance [options]`

Type

AcuSolve Post-Processing Program

Description

AcuHeatBalance is a post-processing utility that uses the AcuSolve solution database to compute the absolute and relative energy balance within the simulation. The energy balance can be computed by summing the surface integrated heat flux values obtained on the walls with the convective temperature flux obtained on the inflow/outflow. The application also reports the integrated temperature, radiation heat flux and mass flux values for the requested element sets.

The primary input for acuHeatBalance is provided with the -cnns flag, referring to the parent element connectivity name, or element set name. By default, the cnns flag is set to _all, which will query all the element sets associated with the simulation to determine the surfaces available and their energy related outputs through osi. The selected surface integrated heat flux on all boundary conditions of type wall is summed to provide the total heat loss/gain due to conduction and radiation. The selected surface integrated convective temperature flux on all boundary conditions of type inflow or outflow is summed to provide the total heat loss/gain due to convection. The quantities are summed to provide an indication of how well the total heat transfer is balanced within the system. For an ideal simulation, the sum of heat flux plus the sum of the inflow/outflow convective temperature flux should be equal to zero. The following relationship is used to determine the percent balance within the system.

$$\%\,Balance = \frac{\sum_{i=1}^{N_w} \dot{Q}_i + \left(\sum_{i=1}^{N_{inlets}} \dot{Q}_i + \sum_{i=1}^{N_{outlets}} \dot{Q}_i\right)}{\sum_{i=1}^{N_{inlets}} \dot{Q}_i + \sum_{i=1}^{N_{outlets}} \dot{Q}_i} * 100.0 \tag{8}$$

Where, $\dot{Q}_i$ is the heat flux and convective temperature flux for the walls and inlets/outlets, respectively. *Nw* is the total number of surfaces defined with the wall boundary condition (fluid and solid mediums if requested), *Ninlets* is the total number of surfaces defined with the inlet boundary condition, and *Noutlets* is the total number of surfaces defined with the outlet boundary condition.

For simulations where both fluid and solid element sets are present, the sum of heat flux on the walls ($\sum_{i=1}^{N_W} \dot{Q}_i$) will contain contributions from both mediums. This may give the indication that the balance is not close to zero. In this case, only the contribution from the fluid element set should be considered by specifying the -cnns flag appropriately.

AcuHeatBalance can only be used on problems were the steady state solution for the Reynolds Averaged Navier-Stokes is specified. The final value of the steady state run is used for the balance and is written to the output files.

In the following, the full name of each option is followed by its abbreviated name and its type. For a general description of option specifications, see *Command Line Options and Configuration Files*. See below for more individual option details:

help or *h* *(boolean)*
: If set, the program prints a usage message and exits. The usage message includes all available options, their current values, and the place where each option is set.

problem or *pb* *(string)*
: The name of the problem is specified via this option. This name is used to generate input file names and extracted surface file names.

run_id or *run* *(integer)*
: Number of the run in which the translation is requested. If *run_id* is set to 0, the last run in the working directory is assumed.

element_sets or *cnns* *(string)*
: Comma-separated list of *element_sets*. These are the user-given names specified as the user-given name of the ELEMENT_SET commands in the input file. If the *element_sets* option is set to _all, all output sets are included into the heat balance computation (default).

periodic or *p* *(boolean)*
: If this option is set to TRUE, the application will allow the balance computation to include the contribution of convective temperature flux from surfaces that are specified with the periodic boundary condition.

query or *q* *(boolean)*
: If this option is set to TRUE, the application will run in query mode. Query mode will simply print a list of element sets and associated surfaces that are available for heat balance computation. The list of surfaces is queried from the list of *surface_output* sets available in the solution database.

verbose or *v* *(integer)*
: Set the verbose level for printing information to the screen. Each higher verbose level prints more information. If *verbose* is set to 0, or less, only warning and error messages are printed. If *verbose* is set to 1, basic processing information is printed in addition to warning and error messages. This level is recommended. *verbose* levels greater than 1 provide information useful only for debugging.

Examples

Consider the computation of the heat transfer within a two-dimensional annulus, where no inlet or outlet surfaces are considered:

```
acuHeatBalance
```

The following will be printed to the standard output of the terminal, as an example:

```
acuHeatBalance:
acuHeatBalance:  Opening the AcuSolve solution data base
acuHeatBalance:  Problem <annulus_heat> directory <ACUSIM.DIR> runId <1>
acuHeatBalance:
acuHeatBalance:        Successfully Opened Problem:     annulus_heat
acuHeatBalance:        Successfully Opened Run Id:           1
acuHeatBalance:        Found Requested Element Sets from cnns string: 1
```

```
acuHeatBalance:         Element Set: Fluid     Surface Set: ID
acuHeatBalance:         Element Set: Fluid     Surface Set: Max_Z
acuHeatBalance:         Element Set: Fluid     Surface Set: Min_Z
acuHeatBalance:         Element Set: Fluid     Surface Set: OD
acuHeatBalance:         Element Set: Fluid     Surface Set: Symmetry
acuHeatBalance:
acuHeatBalance:   Writing to file:  annulus_heat.Individual.Balance.dat
acuHeatBalance:   Writing to file:  annulus_heat.Total.Balance.dat
acuHeatBalance:   Conduction Only
acuHeatBalance:   Walls Heat Flux Sum = -0.000120977209668 W
acuHeatBalance:
acuHeatBalance:   Program complete
```

Running AcuHeatBalance produces two output files containing the simulated contribution of heat flux from the requested surface sets called `<annulus_heat>.Individual.Balance.dat` and `<annulus_heat>.Total.Balance.dat`.

The file called `<annulus_heat>.Individual.Balance.dat` contains the following columns (area removed for brevity), reporting the individual surface output quantities for each surface:

Table 31:

ElementSet	SurfaceSet	SBCType	Temperature	Heat Flux	Convective Temperature Flux	Mass Flux
Fluid	ID	wall	373.0	-0.15	0.0	0.0
Fluid	Max_Z	symmetry	0.0	0.0	0.0	0.0
Fluid	Min_Z	symmetry	0.0	0.0	0.0	0.0
Fluid	OD	wall	327.0	0.15	0.0	0.0

The file called `<annulus_heat>.Total.Balance.dat` contains the following columns, reporting the sum of all the requested surfaces:

Table 32:

Temperature Avg	Heat Flux Total	Conv Temp Flux Total	Mass Flux Total
350	-1.2E-04	0	0

Consider the computation of the heat transfer within a two-dimensional channel, where the convective temperature flux from the inlet to outlet drives the wall heat flux:

```
acuHeatBalance -cnns Fluid
acuHeatBalance:
acuHeatBalance:  Opening the AcuSolve solution data base
acuHeatBalance:  Problem <channel_laminar_heat> directory <ACUSIM.DIR> runId <1>
acuHeatBalance:
acuHeatBalance:        Successfully Opened Problem:     channel_laminar_heat
acuHeatBalance:        Successfully Opened Run Id:           1
acuHeatBalance:        Found Requested Element Sets from cnns string: 1
```

```
acuHeatBalance:         Element Set: Fluid      Surface Set: Inlet
acuHeatBalance:         Element Set: Fluid      Surface Set: Outlet
acuHeatBalance:         Element Set: Fluid      Surface Set: Symm_MaxZ
acuHeatBalance:         Element Set: Fluid      Surface Set: Symm_MinY
acuHeatBalance:         Element Set: Fluid      Surface Set: Symm_MinZ
acuHeatBalance:         Element Set: Fluid      Surface Set: Wall
acuHeatBalance:
acuHeatBalance:    Writing to file:  channel_laminar_heat.Individual.Balance.dat
acuHeatBalance:    Writing to file:  channel_laminar_heat.Total.Balance.dat
acuHeatBalance:    Convective Temperature Flux Inlet + Outlet=51.2592664092 W
acuHeatBalance:    Walls Heat Flux Sum = -52.0100414952 W
acuHeatBalance:    Balance: Convective Flux + Walls Heat Flux = -0.750775086036 W
acuHeatBalance:    Percent Difference = 1.46466217453 %
acuHeatBalance:    Program complete
```

In this case, the file called `channel_laminar_heat.Individual.Balance.dat` contains the following columns, reporting the individual surfaces:

Table 33:

ElementSet	Surface Set	SBC Type	Temperature	Heat Flux	Convective Temperature Flux	MassFlux
Fluid	Inlet	inflow	294.5	0.8	-1717.7	-0.0059
Fluid	Outlet	outflow	303.4	0.0	1768.9	0.0059
Fluid	Symm_MaxZ	slip	0.0	0.0	0.0	0.0000
Fluid	Symm_MinY	slip	293.1	0.0	0.0	0.0000
Fluid	Symm_MinZ	slip	0.0	0.0	0.0	0.0000
Fluid	Wall	wall	348.2	-52.0	0.0	0.0000

The file called `channel_laminar_heat.Total.Balance.dat` contains the following columns, reporting the integrated sum of all the requested surfaces:

Table 34:

Temperature Avg	Heat Flux Total	Conv Temp Flux Total	Mass Flux Total
348.15	-52.0	51.2	0.0

To simply query the solution database, use the following command:

```
acuHeatBalance -q

acuHeatBalance:
acuHeatBalance:   Opening the AcuSolve solution data base
acuHeatBalance:   Problem <annulus_heat> directory <ACUSIM.DIR> runId <0>
acuHeatBalance:
acuHeatBalance:         Successfully Opened Problem:     annulus_heat
```

```
acuHeatBalance:         Successfully Open Run Id:            1
acuHeatBalance:         Total Number of Element Sets:       1
acuHeatBalance:         Element Set Name:                   Fluid
acuHeatBalance:
acuHeatBalance:         Total Number of Surface Sets:       5
acuHeatBalance:         Surface Set Name:                   ID
acuHeatBalance:         Surface Set Name:                   Max_Z
acuHeatBalance:         Surface Set Name:                   Min_Z
acuHeatBalance:         Surface Set Name:                   OD
acuHeatBalance:         Surface Set Name:                   Symmetry
acuHeatBalance:
```

The application will report the amount of heat transfer on a specific surface due to radiation when the *`radiation_output`* command is active on a given surface set.

The following example demonstrates the use of the tool on a database containing radiaition output with no inlets or outlets. The application provides the same previously discussed sum, which includes contributions from radiation. Surfaces contributing to the radiation heat transfer are added to the ouput files individually and are summed for clarity. The sum of a radiation problem should be nearly zero.

```
acuHeatBalance -q

acuHeatBalance:
acuHeatBalance:  Opening the AcuSolve solution data base
acuHeatBalance:  Problem <sphere_radiation> directory <ACUSIM.DIR> runId <0>
acuHeatBalance:
acuHeatBalance:         Successfully Opened Problem:     sphere_radiation
acuHeatBalance:         Successfully Opened Run Id:           1
acuHeatBalance:         Found Requested Element Sets from cnns string: 3
acuHeatBalance:         Element Set: Inner     Surface Set: Inner_Inner_r1
acuHeatBalance:         Element Set: Inner     Surface Set: Inner_Inner_ri
acuHeatBalance:         Element Set: Radiating     Surface Set: Inner_Radiating_r1
acuHeatBalance:         Element Set: Outer     Surface Set: Outer_Outer_r2
acuHeatBalance:         Element Set: Outer     Surface Set: Outer_Outer_ro
acuHeatBalance:         Element Set: Radiating     Surface Set: Outer_Radiating_r2
acuHeatBalance:         Found Radiation Output in Database
acuHeatBalance:         Element Set: Radiating     Radiation Surface Set:
 Inner_Radiating_r1
acuHeatBalance:         Element Set: Radiating     Radiation Surface Set:
 Outer_Radiating_r2
acuHeatBalance:
acuHeatBalance:   Writing to file:  sphere_radiation.Individual.Balance.dat
acuHeatBalance:   Writing to file:  sphere_radiation.Total.Balance.dat
acuHeatBalance:   Conduction + Radiation Only
acuHeatBalance:   Radiative Heat Flux Sum = -3.15662873618e-09 W
acuHeatBalance:   Walls Heat Flux Sum = 0.00224294477922 W
acuHeatBalance:
acuHeatBalance:   Program complete
```

Table 35:

Element Set	Surface Set	SBC Type	Temperature	Heat Flux	Convective Temperature Flux	Mas	ORI Tem	ORI HeatFlux
Inner	Inner_r1	wall	578	-140	0.0	0.0		

Element Set	Surface Set	SBC Type	Temperature	Heat Flux	Convective Temperature Flux	Mas	ORI Tem	ORI HeatFlux
Inner	Inner_ri	wall	300	140	0.0	0.0		
Radiating	Rad_r1	wall	578	0.0	0.0	0.0		
Outer	Outer_r2	wall	1035	140	0.0	0.0		
Outer	Outer_ro	wall	1300	-140	0.0	0.0		
Radiating	Rad_r2	wall	1035	0.0	0.0	0.0		
Radiating	Rad_r1						578	-140
Radiating	Rad_r2						1035	140

AcuOptiStruct

AcuOptiStruct is an AcuSolve post-processing program used to generate an OptiStruct input deck for linear and non-linear thermal stress analysis for conjugate heat transfer problems using the AcuSolve database. In the current version, pressure loads (PLOAD), temperature loads (TEMP) and displacement and rotation single point constraints (SPC) are supported. This utility can be used for both steady and transient problems.

Syntax

`acuOptiStruct [options]`

Type

AcuSolve Post-Processing Program

Description

In the following, the full name of each option is followed by its abbreviated name and its type. For a general description of option specifications, see *Command Line Options and Configuration Files*. See below for more individual option details:

`help` or *`h`* *(boolean)*

If set, the program prints a usage message and exits. The usage message includes all available options, their current values, and the place where each option is set.

`problem` or *`pb`* *(string)*

The name of the problem is specified via this option. This name is used to generate input file names and extracted surface file names.

`working_directory` or *`dir`* *(string)*

All internal files are stored in this directory. This directory does not need to be on the same file system as the user-supplied input files.

`run_id` or *`run`* *(integer)*

Number of the run for which the translation is requested. If *`run_id`* is set to 0, the last run in the working directory is assumed.

`output_format` or *`ofmt`* *(enumerated)*

standard	Writes the OptiStruct input deck in the standard format.
long	Writes the OptiStruct input deck in the long format.

`solid_names` or *`solids`* *(string)*

Comma-separated list of solid names.

`young_modulus` or *`young`* *(string)*

Comma-separated list of values of Young’s modulus for all the solids in the order in which they appear in the solids option.

`poisson_ratio` or *`poisson`* *(string)*

Comma-separated list of values of Poisson’s ratio for all the solids in the order in which they appear in the solids option.

`thermal_expansion` or `thermal` (string)
: Comma-separated list of values of thermal expansion coefficient for all the solids in the order in which they appear in the solids option.

`density` or `den` (string)
: Comma-separated list of values of density for all the solids in the order in which they appear in the solids option.

`reference_temperature` or `reftemp` (string)
: Reference temperature for thermal loading specified in Kelvin.

`spc_surfaces` or `spcsurfs` (string)
: Comma-separated list of names of different surfaces where single point constraint (SPC) needs to be applied.

`spc_surfaces_dof` or `spcsurfsdof` (string)
: Comma-separated list of degrees of freedom for different surfaces in the order in which they appear in the spcsurfs option.

`spc_surfaces_dof_values` or `spcsurfsdofvals` (string)
: Comma-separated list of values of degrees of freedom for different surfaces in the order in which they appear in the spcsurfsdof option.

`spc_volumes` or `spcvols` (string)
: Comma-separated list of names of different volume sets where a single point constraint (SPC) needs to be applied.

`spc_volumes_dof` or `spcvolsdof` (string)
: Comma-separated list of degrees of freedom for different volume sets in the order in which they appear in the spcvols option.

`spc_volumes_dof_values` or `spcvolsdofvals` (string)
: Comma-separated list of values of degrees of freedom for different volume sets in the order in which they appear in the spcvolsdof option.

`time_steps` or `ts` (string)
: Comma-separated list of time steps to be translated. The comma-separated fields have the general range format beg:end:inc, where :end:inc and :inc are optional. beg specifies the first time step in the range. It may be either a given time step, as specified by a number, the letter F (or f) requesting the first available time step, or the letter L (or l) requesting the last available time step. end is the last time step in the range. It may be either a time step number or L (or l) requesting the last available time step. If end is missing, the range is assumed to simply represent a single time step, that is, end=beg and inc=1. inc is the increment that ranges from beg to end. It may be either a number or the letter A (or a) requesting all available time steps in the range. If :inc is missing, it is assumed to be one. The range may also be specified by the single letter A (or a), requesting all available time steps. This is equivalent to F:L:A. `time_steps` is used only for nodal data and is ignored for time series data. Examples of `time_steps` include:

```
acuOptiStruct -ts 35            # step 35

acuOptiStruct -ts 35, 33, 37    # steps 33, 35, and 37

acuOptiStruct -ts 33:37:2       # steps 33, 35, and 37
```

acuOptiStruct -ts 35,33:37:2,37	# steps 33, 35, and 37
acuOptiStruct -ts 33:37	`# all steps from 33 to 37`
acuOptiStruct -ts L	# last available time step
acuOptiStruct -ts F:L:A	# all available steps
acuOptiStruct -ts A	# all available steps

`ignore_missing_steps` or `imts` *(boolean)*
If set, missing requested time steps are ignored. Otherwise, if the requested time step does not exist, the command issues an error message and exits.

`analysis_type` or `type` *(enumerated)*
The type of thermal stress analysis for which the OptiStruct input deck needs to be created.

ctnl	The analysis type will be set to continuous transient non-linear analysis.
tnl	The analysis type will be set to transient non-linear analysis.
tl	The analysis type will be set to transient linear analysis.
snl	The analysis type will be set to steady non-linear analysis.
sl	The analysis type will be set to steady linear analysis.

`line_buff` or `lbuff` *(boolean)*
Flush standard output after each line of output.

`verbose` or `v` *(integer)*
Set the `verbose` level for printing information to the screen. Each higher `verbose` level prints more information. If `verbose` is set to 0, or less, only warning and error messages are printed. If `verbose` is set to 1, basic processing information is printed in addition to warning and error messages. This level is recommended. `verbose` levels greater than one provide information useful only for debugging.

Examples

acuOptiStruct -pb Manifold -solids Solid -spcsurfs Flange_bolts,Outlet_end -spcsurfsdof 123456,123456 -spcsurfsdofvals 0,0 -type sl -ts L

In the example shown above, the acuOptiStruct program is used to generate an input deck for a linear steady state analysis for the problem named ‘Manifold’. The OS input deck is generated for the solid named ‘Solid’ and the surfaces, Flange_bolts and Outlet_end are constrained in all the degrees of freedom (both translation and rotation in X, Y and Z directions with a value of 0). The data from the last time step is used in this example. Since the Young’s modulus, Poisson’s ratio and thermal expansion coefficient are not specified explicitly, the default values will be used.

For the same set of constraints as in the previous example, but a transient non-linear analysis, the following command should be given:

acuOptiStruct -pb Manifold -solids Solid -spcsurfs Flange_bolts,Outlet_end -spcsurfsdof 123456,123456 -spcsurfsdofvals 0,0 -type tnl -ts A

In this case, a transient non-linear analysis input deck will be created with nodal data at every available time step.

For problems involving multiple solids with different material properties, the Young's modulus, Poisson's ratio, thermal expansion coefficient and density values should be specified, and the order should be the same as how they appear in the solids option:

acuOptiStruct -pb brake_disc -solids solidName1,solidName2 -young 180e09,200e09 -poisson 0.30,0.265 -thermal 1.7e-5,1.8e-5 -den 7200, 7800 -spcsurfs Flange_bolts,Outlet_end -spcsurfsdof 123456,123456 -spcsurfsdofvals 0,0, -type sl -ts L

In the above example, the OptiStruct deck will be generated for the two solids ‵solidName1′ and ‵solidName2′ with the following material properties:

Table 36:

	Young's modulus (GPa)	**Poisson's ratio**	**Thermal expansion coefficient (1/K)**	**Density (kg/m^3)**
solidName1	180	0.3	0.000017	7200
solidName2	200	0.265	0.000018	7800

AcuInterp

Interpolate an AcuSolve solution onto a user requested set of three-dimensional set of point locations.

Syntax

`acuInterp [options]`

Type

AcuSolve Post-Processing Program

Description

AcuInterp is a post-processing utility that interpolates from the AcuSolve solution database onto a user requested set of point locations. The point locations are defined in three-dimensional space by providing the *`point_files`* attribute or *`coordinate_start`*, *`coordinate_end`*, and *`number_interpolation`* attributes to the application. Each of the quantities available in the solution database may be requested using the variables attribute with a comma separated list of variables to interpolate or expressions to interpolate, for example, v1=du_dy+x. The file format is modifiable given the user specified set of options for the *`header flag`*, *`fields`*, and *`output_file_format`* attributes.

In the following, the full name of each option is followed by its abbreviated name and its type. For a general description of option specifications, see *Command Line Options and Configuration Files*. See below for more individual option details:

`help` or *`h`* *(boolean)*
: If set, the program prints a usage message and exits. The usage message includes all available options, their current values, and the place where each option is set.

`problem` or *`pb`* *(string)*
: The name of the problem is specified via this option. This name is used to generate input file names and extracted surface file names.

`run_id` or *`run`* *(integer)*
: Number of the run in which the extraction is requested. If *`run_id`* is set to 0, the last run in the working directory is assumed.

`variables` or *`vars`* *(string)*
: Comma-separated list of variables to interpolate or expressions to interpolate from the solution database. These variables are the AcuSolve specific names for the variable quantities that are available in the solution database directly or when using expression syntax, a set of appreciated variables as described in the section below. The list of variables and expression abbreviations can be written to standard out using the query attribute.

`time_steps` or *`ts`* *(string)*
: Specifies the time-steps to interpolate the data from. The attribute accepts integers and AcuSolve specific references for the first, last and increment step (F:L:10).

`point_files` or *`files`* *(string)*
: Comma-separated list to specify the files used to interpolate from, when provided by you. Defines the set of x, y, z point locations to interpolate data from. The *`point_files`* attribute should

reference file names that each contain rows of point locations with format, row index, x, y, z (using any delimiter).

`coordinate_start` or `crds` (string)
: Specifies the location of the starting coordinate of the linear interpolation when no user input is provided for the *`point_files`* attribute.

`coordinate_end` or `crde` (string)
: Specifies the location of the ending coordinate of the linear interpolation when no user input is provided for the *`point_files`* attribute.

`number_interpolation` or `ninterp` (integer)
: Specifies the number of linear interpolation points to be used between *`coordinate_start`* and *`coordinate_end`*.

`ale_flag` or `ale` (boolean)
: If this option is set to TRUE, the output locations will be updated dynamically based on the time-step selected in the *`time_steps`* attribute.

`header` (boolean)
: If this option is set to TRUE, the output files will contain a header to describe each of the columns in the interpolated dataset.

`output_fields` or `fields` (string)
: Comma-separated list to specify the output fields that are printed to the output files. Specified by combining the abbreviated values id,crd,data into a comma-separated list, for example, crd,data or id,crd,data.

`output_file_format` or `ofmt` (enumerated)
: Specifies the output format of the files written to disk. ASCII for text readable space delimited file, binary for compressed binary format.

`query` or `q` (boolean)
: If this option is set to TRUE, the application will run in query mode. Query mode will simply print a list of variables that are available for the interpolation. The information returned is queried from the list of quantities available in the solution database and is problem specific based on the output data requested.

`verbose` or `v` (integer)
: Set the verbose level for printing information to the screen. Each higher verbose level prints more information. If *`verbose`* is set to 0 (or less), only warning and error messages are printed. If *`verbose`* is set to 1, basic processing information is printed in addition to warning and error messages. This level is recommended. *`verbose`* levels greater than 1 provide information useful only for debugging.

Examples

Consider the interpolation of the velocity along a vertical line in the interior of a nozzle at a fixed streamwise location:

```
acuInterp -vars velocity -crds 0.127,0.0,0.0 -crde 0.127,0.0515,0 -ts 5000
```

or alternatively the options may be put into the configuration file `Acusim.cnf` as follows and then type `acuInterp` in the AcuSolve cmd prompt (Windows) or the Linux command line to execute:

```
acuInterp.pb=diffuser_compressible
acuInterp.vars=velocity
acuInterp.crds=0.127,0.0,0.0
acuInterp.crde=0.127,0.0515,0.0
acuInterp.ts=5000.0
```

The following will be printed to the standard output of the terminal:

```
acuInterp:
acuInterp:  Creating 100 linear interpolation points in 0.127,0.0,0.0 to
 0.127,0.0515,0
acuInterp:
acuInterp:  Opening the AcuSolve solution data base
acuInterp:  Problem <diffuser_compressible> directory <ACUSIM.DIR> runId <1>
acuInterp:
acuInterp:  Opened run id <1>
acuInterp:
acuInterp:  Extracting coordinates and building node map
acuInterp:  Extracting element connectivity
acuInterp:  Building projection object
acuInterp:  Processing variables <velocity>
acuInterp:
acuInterp:  Processing step <5000>
acuInterp:  Writing file: interp.dat
```

The above example produces a single output file containing data interpolated from time-step 5000 to the linearly interpolated points created from the starting and ending coordinates. The output file will have the name `diffuser_compressible_step_000005000.interp.dat`.

The output file contains the following columns:

X	Y	Z	X-Vel	Y-Vel	Z-Vel
1.27e-01	4.16e-03	0.00e+00	2.98e+02	2.64e+01	0.00e+00

For a steady state simulation, only the last value is likely of interest to you, as it would be the converged solution for a Reynolds Averaged Navier-Stokes simulation. This time-step is output by default. Additional time-steps, if specified in the `NODAL_OUTPUT` command, can be requested with the -ts option as described above.

Next, consider the interpolation of the pressure, normalized by the inlet pressure along the bottom wall of a nozzle:

```
acuInterp -vars p=p/135000.0 -files diffuser_compressible.dat -ts
    1000:2000:500
```

or alternatively the options may be put into the configuration file `Acusim.cnf` as follows:

```
acuInterp.pb=diffuser_compressible
acuInterp.vars=p=p/135000.0
acuInterp.files=diffuser_compressible.dat
acuInterp.ts=1000:2000:500
```

The following will be printed to the standard output of the terminal:

```
acuInterp:
acuInterp:  Reading points from file <diffuser_compressible.dat>
acuInterp:
acuInterp:  Opening the AcuSolve solution data base
acuInterp:  Problem <diffuser_compressible> directory <ACUSIM.DIR> runId <1>
acuInterp:
acuInterp:  Opened run id <1>
acuInterp:
acuInterp:  Extracting coordinates and building node map
acuInterp:  Extracting element connectivity
acuInterp:  Building projection object
acuInterp:  Processing variables <var=p/135000.0>
acuInterp:
acuInterp:  Processing step <1000>
acuInterp:  Evaluating user expression:  var
acuInterp:      Equation                        :  p/135000.0
acuInterp:  Writing file: diffuser_compressible.interp_step_00001000.dat
acuInterp:
acuInterp:  Processing step <1500>
acuInterp:  Evaluating user expression:  var
acuInterp:      Equation                        :  p/135000.0
acuInterp:  Writing file: diffuser_compressible.interp_step_00001500.dat
acuInterp:
acuInterp:  Processing step <2000>
acuInterp:  Evaluating user expression:  var
acuInterp:      Equation                        :  p/135000.0
acuInterp:  Writing file: diffuser_compressible.interp_step_00002000.dat
```

The above example produces three output files containing data interpolated from time-step 1000, 1500 and 2000 to the point location file as specified.

The output file contains the following columns:

X	Y	Z	var=p/135000.0
1.7775e-01	0.000e+00	0.000e+00	8.65e-01

To simply query the solution database, use the following command:

```
acuInterp -q
```

The following will be printed to the standard output of the terminal:

```
acuInterp:
acuInterp:  Opening the AcuSolve solution data base
acuInterp:  Problem <diffuser_compressible> directory <ACUSIM.DIR> runId <1>
acuInterp:
acuInterp:  Opened run id <1>
acuInterp:
acuInterp:  Available nodal output variables:
acuInterp:    velocity
acuInterp:    pressure
acuInterp:    temperature
```

```
acuInterp:    eddy_viscosity
acuInterp:    surface_y_plus
acuInterp:    surface_film_coefficient
acuInterp:    wall_shear_stress
acuInterp:    density
acuInterp:    mach_number
acuInterp:
acuInterp:  Available extended output variables:
acuInterp:    velocity
acuInterp:    pressure
acuInterp:    temperature
acuInterp:    eddy_viscosity
acuInterp:    surface_y_plus
acuInterp:    surface_film_coefficient
acuInterp:    wall_shear_stress
acuInterp:    density
acuInterp:    mach_number
acuInterp:    grad_velocity
acuInterp:    grad_pressure
acuInterp:    grad_temperature
acuInterp:    grad_eddy_viscosity
acuInterp:    grad_surface_y_plus
acuInterp:    grad_surface_film_coefficient
acuInterp:    grad_wall_shear_stress
acuInterp:    grad_density
acuInterp:    grad_mach_number
acuInterp:    volume
acuInterp:    strain_rate_invariant_2
acuInterp:    velocity_magnitude
acuInterp:    vorticity
acuInterp:    running_ave_velocity
acuInterp:    running_ave_pressure
acuInterp:    running_ave_temperature
acuInterp:    running_ave_eddy_viscosity
acuInterp:    viscosity
acuInterp:    gravity
acuInterp:    specific_heat
acuInterp:    conductivity
acuInterp:    heat_source
acuInterp:    turbulence_y
acuInterp:    turbulence_y_plus
acuInterp:    material_viscosity
acuInterp:    material_conductivity
acuInterp:    total_pressure
acuInterp:    total_temperature
acuInterp:    cfl_number
acuInterp:
acuInterp:  Available derived quantity output variables:
acuInterp:    viscosity
acuInterp:    gravity
acuInterp:    specific_heat
acuInterp:    conductivity
acuInterp:    heat_source
acuInterp:    turbulence_y
acuInterp:    turbulence_y_plus
acuInterp:    material_viscosity
acuInterp:    material_conductivity
acuInterp:    total_pressure
acuInterp:    total_temperature
acuInterp:    cfl_number
acuInterp:
acuInterp:  Available running average output variables:
acuInterp:    runing_ave_running_ave_velocity
```

```
acuInterp:    runing_ave_running_ave_pressure
acuInterp:    runing_ave_running_ave_temperature
acuInterp:    runing_ave_running_ave_eddy_viscosity
acuInterp:
acuInterp:  Available time average output variables:
acuInterp:
acuInterp:  Expression syntax variables:
acuInterp:  x                  = crd
acuInterp:  y                  = crd
acuInterp:  z                  = crd
acuInterp:  u                  = velocity
acuInterp:  v                  = velocity
acuInterp:  w                  = velocity
acuInterp:  time_ave_u         = time_ave_velocity
acuInterp:  time_ave_v         = time_ave_velocity
acuInterp:  time_ave_w         = time_ave_velocity
acuInterp:  time_ave_uu_square = time_ave_velocity_square
acuInterp:  time_ave_vv_square = time_ave_velocity_square
acuInterp:  time_ave_ww_square = time_ave_velocity_square
acuInterp:  time_ave_uv_square = time_ave_velocity_square
acuInterp:  time_ave_vw_square = time_ave_velocity_square
acuInterp:  time_ave_wx_square = time_ave_velocity_square
acuInterp:  running_ave_u      = running_ave_velocity
acuInterp:  running_ave_v      = running_ave_velocity
acuInterp:  running_ave_w      = running_ave_velocity
acuInterp:  u_mag              = velocity_magnitude
acuInterp:  du_dx              = grad_velocity
acuInterp:  du_dy              = grad_velocity
acuInterp:  du_dz              = grad_velocity
acuInterp:  dv_dx              = grad_velocity
acuInterp:  dv_dy              = grad_velocity
acuInterp:  dv_dz              = grad_velocity
acuInterp:  dw_dx              = grad_velocity
acuInterp:  dw_dy              = grad_velocity
acuInterp:  dw_dz              = grad_velocity
acuInterp:  x_vort             = vorticity
acuInterp:  y_vort             = vorticity
acuInterp:  z_vort             = vorticity
acuInterp:  vort_mag           = vort_mag
acuInterp:  vort_re_num        = vort_re_num
acuInterp:  d                  = turbulence_y
acuInterp:  p                  = pressure
acuInterp:  running_ave_p      = running_ave_pressure
acuInterp:  time_ave_p         = time_ave_pressure
acuInterp:  dp_dx              = grad_pressure
acuInterp:  dp_dy              = grad_pressure
acuInterp:  dp_dz              = grad_pressure
acuInterp:  T                  = temperature
acuInterp:  dT_dx              = grad_temperature
acuInterp:  dT_dy              = grad_temperature
acuInterp:  dT_dz              = grad_temperature
acuInterp:  tau_x              = wall_shear_stress
acuInterp:  tau_y              = wall_shear_stress
acuInterp:  tau_z              = wall_shear_stress
acuInterp:  x_disp             = mesh_displacement
acuInterp:  y_disp             = mesh_displacement
acuInterp:  z_disp             = mesh_displacement
acuInterp:  x_mesh_vel         = mesh_velocity
acuInterp:  y_mesh_vel         = mesh_velocity
acuInterp:  z_mesh_vel         = mesh_velocity
acuInterp:  rho                = density
acuInterp:  mu_total           = viscosity
acuInterp:  mu                 = material_viscosity
```

```
acuInterp:  mu_t                = eddy_viscosity
acuInterp:  dmu_t_dx            = grad_eddy_viscosity
acuInterp:  dmu_t_dy            = grad_eddy_viscosity
acuInterp:  dmu_t_dz            = grad_eddy_viscosity
acuInterp:  k                   = kinetic_energy
acuInterp:  dk_dx               = grad_kinetic_energy
acuInterp:  dk_dy               = grad_kinetic_energy
acuInterp:  dk_dz               = grad_kinetic_energy
acuInterp:  omega               = eddy_frequency
acuInterp:  domega_dx           = grad_eddy_frequency
acuInterp:  domega_dy           = grad_eddy_frequency
acuInterp:  domega_dz           = grad_eddy_frequency
acuInterp:  epsilon             = dissipation_rate
acuInterp:  depsilon_dx         = grad_dissipation_rate
acuInterp:  depsilon_dy         = grad_dissipation_rate
acuInterp:  depsilon_dz         = grad_dissipation_rate
acuInterp:  field_1             = field
acuInterp:  field_2             = field
acuInterp:  field_3             = field
acuInterp:  field_4             = field
acuInterp:  q                   = surface_heat_flux
acuInterp:  area                = area
acuInterp:  time_ave_pressure   = time_ave_pressure
acuInterp:  incident_radiation = incident_radiation
acuInterp:
```

AcuGetCpCf

Extract the coefficient of pressure (C_p) and the coefficient of friction (C_f) from an AcuSolve solution database.

Syntax

`acuGetCpCf [options]`

Type

AcuSolve Post-Processing Program

Description

AcuGetCpCf is a post-processing utility that uses the AcuSolve solution database of nodal pressure and shear stress to compute the coefficient of pressure (C_p) and the coefficient of friction (C_f) on a specified set of points. The C_p and C_f values are computed on a curve that is projected onto a user requested surface(s) or explicitly defined by you in three-dimensional space. The selected surfaces, determined by the `osis` flag, are queried for, and obtained during the execution of the application. The nodal values of pressure (shear stress) from those surfaces are extracted from the user specified `point_type` attribute, specified as `auto` by default. When `point_type`=auto, you must select the `radial_locations` and `cutplane_normal_direction` attributes to define the location and cut plane direction corresponding to the requested surface definition. The quantities obtained at the requested locations are then normalized according to the calculated dynamic pressure, as shown in the following relationship.

$$C_p = \frac{p - p_\infty}{0.5\rho_\infty U_\infty^2} \tag{9}$$

$$C_f = \frac{\tau_{wall}}{0.5\rho_\infty U_\infty^2} \tag{10}$$

Where, C_p is the coefficient of pressure, p is the local pressure, p_∞ is the freestream pressure, ρ_∞ is the freestream density and U_∞ is the freestream velocity, defined by the nodal pressure, `reference_pressure`, `reference_density` and `reference_velocity`, respectively.

C_f is the coefficient of friction, computed from the magnitude of wall shear stress, τ_{wall} and the same normalization parameters.

In the following, the full name of each option is followed by its abbreviated name and its type. For a general description of option specifications, see *Command Line Options and Configuration Files*. See below for more individual option details:

`help` *or* `h` *(boolean)*
: If set, the program prints a usage message and exits. The usage message includes all available options, their current values, and the place where each option is set.

problem or *pb* *(string)*
: The name of the problem is specified via this option. This name is used to generate input file names and extracted surface file names.

working_directory or *dir* *(string)*
: All internal files are stored in this directory. This directory does not need to be on the same file system as the input files of the problem.

run_id or *run* *(integer)*
: Number of the run in which the extraction is requested. If *run_id* is set to 0, the last run in the working directory is assumed.

time_step or *ts* *(integer)*
: Time step from which to extract selected data.

surface_integral_output_sets or *osis* *(string)*
: Comma-separated list of *surface_output* sets. These are the user-given names specified as the user-given name of the `SURFACE_OUTPUT` commands in the input file. If *surface_integral_output_sets* is set to _all, all output sets are included into the coefficient computation.

curve_type or *type* *(enumerated)*
: Specifies the type of data to generate during the normalization computation. You can specify either cp or cf to compute the coefficient of pressure and coefficient of friction, respectively.

point_type or *pttype* *(enumerated)*
: Specifies the type of data extraction method used to obtain nodal data. You can specify either auto or file. Selecting auto will automatically project the data from the requested surfaces onto a three-dimensional curve. When file is selected, the application requires an explicitly defined set of x, y, z point locations to project the data to.

points_file or *pts* *(string)*
: Specifies the file used when *point_type*=file and defines the set of x, y, z point locations to extract data from. The *points_file* contains rows of point locations with format, row index, x, y, z (using any delimiter).

radial_locations or *rad_locs* *(string)*
: Specifies the location(s) of the cut plane along the *cutplane_normal_direction* when *point_type*=auto. Comma separated string is accepted to run the extraction at multiple cut planes.

cutplane_normal_direction or *cut_dir* *(enumerated)*
: Specifies the cut plane normal direction when *point_type*=auto. Used in conjunction with the *radial_location* to define the projection direction and cut location.

gauge_pressure or *gauge_pres* *(real)*
: Specifies the gauge pressure conversion from relative to absolute pressure if required. Used when *absolute_pressure_offset* is not equal to 0.0 when the AcuSolve solution was run.

reference_density or *ref_rho* *(real)*
: Specifies the reference density used to normalize the pressure (friction) values into coefficients. It should be specified as the freestream density.

`reference_velocity` or *`ref_vel`* *(real)*
: Specifies the reference velocity used to normalize the pressure (friction) values into coefficients. It should be specified as the magnitude of the freestream velocity.

`reference_pressure` or *`ref_pres`**(real)*
: Specifies the reference pressure used to offset the pressure values for the coefficient computation. It should be specified as the freestream pressure.

`reference_viscosity` or *`ref_vis`* *(real)*
: Specifies the reference dynamic viscosity for the Reynolds number calculation.

`specific_heat_ratio` or *`gamma`* *(real)*
: Specifies the ratio of specific heats for the Mach number calculation.

`normalize_chord` or *`nc`* *(boolean)*
: If this option is set to TRUE, the output location will be normalized by the local chord length.

`chord_scale_fac` or *`csf`* *(real)*
: Specifies the scaling factor to apply to the coordinate locations. Used prior to scaling by the local chord length when *`normalize_chord`*=true.

`cp_scale_fac` or *`cpsf`* *(real)*
: Specifies the scaling factor to apply to the coefficients after they are calculated according to the provided reference quantities.

`inviscid_flow` or *`inviscid`* *(boolean)*
: If this option is set to True, the cp calculation will be normalized by the *`reference_velocity`* alone and not include effects of surface velocity if non-zero.

`query` or *`q`* *(boolean)*
: If this option is set to TRUE, the application will run in query mode. Query mode will simply print a list of surfaces that are available for coefficient computation. The list of surfaces is queried from the list of *`surface_output`* sets available in the solution database.

`header` *(boolean)*
: If this option is set to TRUE, the application will write the variable names for each data column in the output file.

`output_file_format` or *`ofmt`* *(enumerated)*
: Specifies the output format of the files written to disk. Ascii for text readable space delimited file, binary for compressed binary format.

`output_coordinate` or *`out_crd`* *(boolean)*
: If this option is set to TRUE, the application will also output the x-,y-,z-coordinate in addition to x/c,Cp.

`verbose` or *`v`* *(integer)*
: Set the verbose level for printing information to the screen. Each higher verbose level prints more information. If *`verbose`* is set to 0 (or less), only warning and error messages are printed. If *`verbose`* is set to 1, basic processing information is printed in addition to warning and error messages. This level is recommended. *`verbose`* levels greater than 1 provide information useful only for debugging.

Examples

Consider the computation of the pressure coefficient of a three-dimensional wing at multiple spanwise locations:

```
acuGetCpCf -pb wing -osis 'Wall - Output' -reference_pressure 94760.7 -
reference_velocity 291.7 -reference_density 1.1 -rad_locs 0.23926,0.526372,0.777595
-cut_dir y -type cp
```

or alternatively the options may be put into the configuration file `Acusim.cnf` as follows and then type `acuGetCpCf` in the AcuSolve cmd prompt (Windows) or the Linux command line to execute:

```
acuGetCpCf.pb=wing
acuGetCpCf.osis=”Wall - Output”
acuGetCpCf.reference_pressure=94760.7
acuGetCpCf.reference_velocity=291.7
acuGetCpCf.reference_density=1.1
acuGetCpCf.rad_locs=0.23926,0.526372,0.777595
acuGetCpCf.cut_dir=y
acuGetCpCf.type=cp
```

The following will be printed to the standard output of the terminal:

```
acuGetCpCf:
acuGetCpCf:  Opening the AcuSolve solution data base
acuGetCpCf:  Problem <onera> directory <ACUSIM.DIR> runId <0>
acuGetCpCf:
acuGetCpCf:  Opened run id <1>
acuGetCpCf:
acuGetCpCf:  Extracting <pressure> field from step <1500>
acuGetCpCf:
acuGetCpCf:  Projecting a circle of radius: 1.34e+00
acuGetCpCf:
acuGetCpCf:  Cut plane: x-z
acuGetCpCf:
acuGetCpCf:  Building projection object
acuGetCpCf:
acuGetCpCf:  Processing location:  1
acuGetCpCf:     Cut coordinate:            y = 0.23926
acuGetCpCf:     Local chord:               0.740
acuGetCpCf:     Rotational velocity:       0.0
acuGetCpCf:     Total velocity:            291.668
acuGetCpCf:     Reference dynamic pressure: 4.68e+04
acuGetCpCf:     Cp min/max:                -0.70275/0.90013
acuGetCpCf:     Chord Reynolds number:     1.33e+07
acuGetCpCf:     Writing file:              onera.cp.1.dat
acuGetCpCf:
acuGetCpCf:  Processing location:  2
acuGetCpCf:     Cut coordinate:            y = 0.526372
acuGetCpCf:     Local chord:               0.654
acuGetCpCf:     Rotational velocity:       0.0
acuGetCpCf:     Total velocity:            291.668
acuGetCpCf:     Reference dynamic pressure: 4.68e+04
acuGetCpCf:     Cp min/max:                -0.8154/0.8739
acuGetCpCf:     Chord Reynolds number:     1.18e+07
acuGetCpCf:     Writing file:              onera.cp.2.dat
acuGetCpCf:
acuGetCpCf:  Processing location:  3
acuGetCpCf:     Cut coordinate:            y = 0.777595
acuGetCpCf:     Local chord:               0.580
acuGetCpCf:     Rotational velocity:       0.0
```

```
acuGetCpCf:     Total velocity:             291.668
acuGetCpCf:     Reference dynamic pressure: 4.68e+04
acuGetCpCf:     Cp min/max:                 -0.8926/0.85577
acuGetCpCf:     Chord Reynolds number:      1.04e+07
acuGetCpCf:     Writing file:               onera.cp.3.dat
acuGetCpCf:
acuGetCpCf:  Freestream Mach number:        0.84
acuGetCpCf:  Theoretical compressible Cp:
acuGetCpCf:     At stagnation point (max):   1.18899
acuGetCpCf:     At sonic point (Mach = 1):  -0.32728
```

The above example produces three output files containing the last time-step from the solution with normalized x/C and pressure coefficient called `onera.cp.[1-3].dat`.

The output file contains the following columns:

x/C location	Cp
1.0000000000000000e+00	1.4607121703357254e-01

For a steady state simulation, only the last value is likely of interest to you, as it would be the converged solution for a Reynolds Averaged Navier-Stokes simulation. This time-step is output by default. Additional time-steps, if specified in the `NODAL_OUTPUT` command, can be requested with the -ts option as described above.

To simply query the solution database, use the following command:

```
acuGetCpCf -q
```

The following will be printed to the standard output of the terminal:

```
acuGetCpCf:
acuGetCpCf:  Opening the AcuSolve solution data base
acuGetCpCf:  Problem <onera> directory <ACUSIM.DIR> runId <0>
acuGetCpCf:
acuGetCpCf:  Opened run id <1>
acuGetCpCf:
acuGetCpCf:  Surface output name:                   Far field - Output
acuGetCpCf:       Number of surface nodes:          4713
acuGetCpCf:       Number of integrated output steps: 1500
acuGetCpCf:  Surface output name:                   Symmetry - Output
acuGetCpCf:       Number of surface nodes:          33631
acuGetCpCf:       Number of integrated output steps: 1500
acuGetCpCf:  Surface output name:                   Wall - Output
acuGetCpCf:       Number of surface nodes:          121282
acuGetCpCf:       Number of integrated output steps: 1500
```

Consider the computation of the pressure coefficient of a three-dimensional wing

```
acuGetCpCf -reference_pressure  94760.7 -reference_velocity  291.7 -
reference_density 1.1 -type cp -output_coordinate -normalize_chord -pttype file
-pts point_locations.txt

acuGetCpCf:
acuGetCpCf:  Opening the AcuSolve solution data base
acuGetCpCf:  Problem <onera> directory <ACUSIM.DIR> runId <0>
```

```
acuGetCpCf:
acuGetCpCf:  Opened run id <1>
acuGetCpCf:
acuGetCpCf:  Extracting <wall_shear_stress> field from step <1500>
acuGetCpCf:
acuGetCpCf:  Projecting a circle of radius: 1.34e+00
acuGetCpCf:
acuGetCpCf:  Reading file:  point_locations.txt
acuGetCpCf:  Building projection object
acuGetCpCf:
acuGetCpCf:  Processing location:  1
acuGetCpCf:     Local chord:                0.833
acuGetCpCf:     Rotational velocity:        0.0
acuGetCpCf:     Total velocity:             291.667
acuGetCpCf:     Reference dynamic pressure: 4.68e+04
acuGetCpCf:     Cf-x min/max:               2.96307690806e-06/0.00304619492264
acuGetCpCf:     Cf-y min/max:               -0.000470032309125/0.00153358625221
acuGetCpCf:     Cf-z min/max:               -0.00158707581029/0.0021541080237
acuGetCpCf:     Cf-mag min/max:             7.05953432918e-06/0.0033957476328
acuGetCpCf:     Chord Reynolds number:      1.33e+07
acuGetCpCf:     Writing file:               onera.cf.dat
acuGetCpCf:
acuGetCpCf:  Freestream Mach number:        0.84
acuGetCpCf:  Theoretical compressible Cp:
acuGetCpCf:     At stagnation point (max):   1.18899
acuGetCpCf:     At sonic point (Mach = 1):  -0.32725
```

The output file contains the following columns:

x/C	Cf(x)	Cf(y)	Cf(z)	Cf(mag)	X	Y	Z
1.00e+00	2.85e-04	1.12e-04	2.33e-06	3.07e-04	8.77e-01	2.39e-01	-4.84e-10

AcuGetNodeSubset

Extract a subset of AcuSolve nodal output to disk.

Syntax

`acuGetNodeSubset [options]`

Type

AcuSolve Post-Processing Program

Description

AcuGetNodeSubset is a post-processing utility used to extract a subset of nodal output from the AcuSolve solution database at user-given node locations.

In the following, the full name of each option is followed by its abbreviated name and its type. For a general description of option specifications, see *Command Line Options and Configuration Files*. See below for more individual option details:

`help` *or* *`h`* *(boolean)*
: If set, the program prints a usage message and exits. The usage message includes all available options, their current values, and the place where each option is set.

`problem` *or* *`pb`* *(string)*
: The name of the problem is specified via this option. This name is used to generate input file names and extracted surface file names.

`run_id` *or* *`run`* *(integer)*
: Number of the run in which the extraction is requested. If *`run_id`* is set to 0, the last run in the working directory is assumed.

`element_sets` *or* *`cnns`* *(string)*
: Comma-separated list of element sets to extract nodal data from. These are the names specified as the user-given name of the `ELEMENT_SET` commands in the input file. If *`element_sets`* is set to _all, all output sets are used to extract the nodal data.

`surfaces` *or* *`osis`* *(string)*
: Comma-separated list of *`surface_output`* sets. These are the names specified as the user-given name of the `SURFACE_OUTPUT` commands in the input file. If *`surface_integral_output_sets`* is set to _all, all output sets are used to extract the nodal data.

`node_file` *or* *`files`* *(string)*
: Comma-separated list of node files containing node IDs to extract the dataset from.

`time_steps` *or* *`ts`* *(string)*
: Comma-separated list of time steps to be translated. The comma-separated fields have the general range format **`beg:end:inc`**, where **`:end:inc`** and **`:inc`** are optional. **`beg`** specifies the first time step in the range. It may be either a given time step, as specified by a number, the letter F (or f) requesting the first available time step. **`end`** is the last time step in the range. It may be either a time step number or the letter L (or l) requesting the last available time step. If **`end`** is missing, the range is assumed to simply represent a single time step (that is, end = beg and inc

= 1). **inc** is the increment that ranges from **beg** to **end**. It may be either a number or the letter A (or a) requesting all available time steps in the range. If **:inc** is missing, it is assumed to be one. The range may also be specified by the single letter A (or a), requesting all available time steps. This is equivalent to F:L:A.

nodal_output_vars or *outv* *(string)*
Comma-separated list of *nodal_output* variables to be translated. The list may include:

Table 37:

Variable (abbr)	**Fields**	**Description**
density (dens)	1	Density, only for *flow* = compressible_navier_stokes.
velocity (vel)	3	Velocity vector
mach_number (mach)	1	Mach number, only for *flow* = compressible_navier_stokes.
pressure (pres)	1	Pressure
temperature (temp)	1	Temperature
relative_humidity	1	Relative humidity
dewpoint_temperature	1	Dewpoint temperature
humidity_film_thickness	1	Humidity film thickness
species (spec)	nSpecs	Species
incident_radiation (incident_rad)	1	Incident radiation
field	nFields	Field values. For *multi_field* = levelset and *multi_field* = algebraic_eulerian, the field values correspond to volume fractions, and are named as volume_fraction-"fieldname". For *multi_field* = advective_diffusive, the field values correspond to mass fraction.
levelset (levelset)	1	Levelset
eddy_viscosity (eddy)	1	Turbulence eddy viscosity
kinetic_energy (tke)	1	Turbulence kinetic energy

Variable (abbr)	**Fields**	**Description**
eddy_frequency (tomega)	1	Turbulence eddy frequency
sqrt_eddy_period (tg)	1	Inverse of the square root of eddy frequency
dissipation_rate (teps)	1	Turbulence dissipation rate
intermittency (tintc)	1	Turbulence intermittency
transition_re_theta (treth)	1	Critical momentum thickness Reynolds Number
surface_y_plus (yp)	1	y+ on turbulence walls
surface_film_coefficient (film)	1	Convective heat transfer coefficient on turbulence walls.
wall_shear_stress (wall_shear)	3	Wall shear stress on turbulence walls.
viscoelastic_stress (vest)	6	Viscoelastic stresses
mesh_displacement (mesh_disp)	3	Mesh displacement vector
mesh_velocity (mesh_vel)	3	Mesh velocity vector

where *nSpecs* is the number of species as given in the `EQUATION` command in the input file. The problem must contain the requested variable in order for it to be translated. For example, the parameter *turbulence* in the `EQUATION` command must be set to a value other than none in order for *eddy_viscosity* to be available. The list of variables is sorted in the order given in the above table. If *nodal_output_vars* is set to _all, all available variables are translated. The *surface_y_plus*, *surface_film_coefficient* and *wall_shear_stress* are non-zero only on surface nodes given by `TURBULENCE_WALL`, or alternatively by `SIMPLE_BOUNDARY_CONDITION` of type wall. The *surface_film_coefficient* is computed even if there is no temperature equation. However, all relevant fluid material models must include specific heat and conductivity models. It should also be noted that *eddy_frequency* and *sqrt_eddy_period* only appear when using the k-omega based turbulence models. The *dissipation_rate* variable only appears when using the k-epsilon based turbulence models, and *intermittency* and *transition_re_theta* only appear when using the turbulence transition models.

output_fields or *fields* *(string)*

Comma-separated list to specify the output fields that are printed to the output files. Specified by combining the abbreviated values id,crd,time,data into a comma-separated list, for example, crd,data or id,crd,data or id,crd,time,data.

`output_file_format` or `ofmt` *(enumerated)*

Specifies the output format of the files written to disk. ASCII for text readable space delimited file, binary for compressed binary format.

`verbose` or `v` *(integer)*

Set the verbose level for printing information to the screen. Each higher verbose level prints more information. If `verbose` is set to 0 (or less), only warning and error messages are printed. If `verbose` is set to 1, basic processing information is printed in addition to warning and error messages. This level is recommended. `verbose` levels greater than 1 provide information useful only for debugging.

Examples

Consider the extraction of the thermal solution from a single element set at multiple time points:

```
acuGetNodeSubset -cnns M1S1P1 outv temperature -ts 100:500:100
```

or alternatively the options may be put into the configuration file `Acusim.cnf` as follows and then type `acuGetNodeSubset` in the AcuSolve cmd prompt (Windows) or the Linux command line to execute:

```
acuGetNodeSubset.pb=battery acuGetNodeSubset.cnns=M1S1P1
acuGetNodeSubset.outv=temperature acuGetNodeSubset.ts=100:500:100
```

The following will be printed to the standard output of the terminal:

```
acuGetNodeSubset:
acuGetNodeSubset:  Opening the AcuSolve solution data base
acuGetNodeSubset:  Problem <battery> directory <ACUSIM.DIR> runId <0>
acuGetNodeSubset:
acuGetNodeSubset:  Opened run id <1>
acuGetNodeSubset:
acuGetNodeSubset:  Processing element set <M1S1P1>
acuGetNodeSubset:
acuGetNodeSubset:  Compiling the unique node list
acuGetNodeSubset:
acuGetNodeSubset:  Processing step <100>
acuGetNodeSubset:  Writing data to file <battery.subset_step000100.out>
acuGetNodeSubset:  Processing step <200>
acuGetNodeSubset:  Writing data to file <battery.subset_step000200.out>
acuGetNodeSubset:  Processing step <300>
acuGetNodeSubset:  Writing data to file <battery.subset_step000300.out>
acuGetNodeSubset:  Processing step <400>
acuGetNodeSubset:  Writing data to file <battery.subset_step000400.out>
acuGetNodeSubset:  Processing step <500>
acuGetNodeSubset:  Writing data to file <battery.subset_step000500.out>
acuGetNodeSubset:
acuGetNodeSubset:  Total elapsed time <96.8489789963> seconds
acuGetNodeSubset:
```

The above example produces five output files containing the simulated series of nodal (spatial dependent) temperatures contained for the element set with the name “M1S1P1”.

The file contains the following columns with example output quantities:

CoordinateX	CoordinateY	CoordinateZ	Temperature

-1.5499e-01	-1.225e-01	-2.396e-01	3.00586e+02

Consider the extraction of the thermal solution from a single surface set at a single time point:

```
acuGetNodeSubset -osis Cells_M1S8P7 -outv temperature -ts 100
```

or alternatively the options may be put into the configuration file `Acusim.cnf` as follows:

```
acuGetNodeSubset.pb=battery
acuGetNodeSubset.osis=Cells_M1S8P7
acuGetNodeSubset.outv=temperature
acuGetNodeSubset.ts=100
```

The following will be printed to the standard output of the terminal:

```
acuGetNodeSubset:
acuGetNodeSubset:  Opening the AcuSolve solution data base
acuGetNodeSubset:  Problem <battery> directory <ACUSIM.DIR> runId <0>
acuGetNodeSubset:
acuGetNodeSubset:  Opened run id <1>
acuGetNodeSubset:
acuGetNodeSubset:  Processing surface output <Cells_M1S8P7>
acuGetNodeSubset:    Surface output <Cells_M1S8P7> matches <Cells_M1S8P7>
acuGetNodeSubset:
acuGetNodeSubset:  Compiling the unique node list
acuGetNodeSubset:
acuGetNodeSubset:  Processing step <100>
acuGetNodeSubset:  Writing data to file <battery.subset_step000100.out>
acuGetNodeSubset:
acuGetNodeSubset:  Total elapsed time <21.0808241367> seconds
```

The above example produces five output files containing the simulated series of nodal (spatial dependent) temperatures contained for the element set with the name “M1S1P1”.

AcuPlotData

Plotting utility for basic two-dimensional plot creation from an ascii text data file.

Syntax

`acuPlotData [options]`

Type

AcuSolve Post-Processing Program

Description

AcuPlotData is a post-processing utility that uses generic ascii text data files as input to generate two-dimensional visualizations based on the functionality available in the Python library, Matplotlib. AcuPlotData is provided to AcuSolve users for the purpose of generating high-quality visualizations of AcuSolve outputs leveraging automated input and image processing. You are required to provide an ascii text file defined with the *`files`* attribute. Further control of the image development process is achieved using the input commands that are able to customize the visualization for user specific requirements.

In the following, the full name of each option is followed by its abbreviated name and its type. For a general description of option specifications, see *Command Line Options and Configuration Files*. See below for more individual option details:

`help` or *`h`* *(boolean)*

    If set, the program prints a usage message and exits. The usage message includes all available options, their current values, and the place where each option is set.

`problem` or *`pb`* *(string)*

    The name of the problem is specified via this option. This name is used to generate input file names and extracted surface file names.

`files` *(string)*

    Comma-separated list to specify the files used to generate plots from. Defines the files that will be plotted, one or more files are required at input. The *`files`* attribute should reference file names that each contain rows of data with format, row index, x, y, z (using any delimiter). Quantities are not restricted to any specific number of columns or data type (real and integer are accepted).

`x_column` or *`x_col`* *(int)*

    Specifies the *`x_column`* (abscissa axis) of the data file to be plotted as the independent variable. Accepts a single column for all *`y_columns`* provided by you.

`y_columns` or *`y_cols`* *(string)*

    Comma-separated list to specify the y-column places to be plotted. Each integer in the list specifies the *`y_column`* (ordinate axis) of the data file to be plotted as the dependent variable(s). Accepts one or more columns separated by commas for all *`y_columns`* provided by you.

`x_scale_factor` or *`x_scale`* *(real)*

    Specifies the x-scale factor to apply to plotted data. Multiplies the *`x_column`* data by *`x_scale_factor`*.

y_scale_factors or *y_scale* *(str)*
: Comma-separated list to specify the y-scale factor(s) to apply to plotted data. Multiplies the *y_columns* data by *y_scale_factor* once separated from the list. Only provide a single value when one value for *y_columns* is given.

offset *(real)*
: Specifies the percentage of initial data points to skip when plotting the data file.

fft *(boolean)*
: If this option is set to True, time dependent data will be transformed using a Fast-Fourier transformation. If using, the first column in the data file must be time, as this is used to compute the time-step size and subsequent quantities in relation to the FFT.

stats *(boolean)*
: If this option is set to True, basic statistical quantities will be computed from the user provided *y_colunms* fields. The application will compute the maximum, minimum, mean, and root-mean-square (rms) of data.

type or *plot_type* *(enumerated)*
: Specifies the format of the vertical and horizontal axes. When *plot_type*=standard, each axis is plotted with a linear range of points. *semilogx*, *semilogy* and *loglog* will specify that the horizontal, vertical or both axes are plotted using a logarithmic distribution of the data, respectively.

x_axis_range or *x_range* *(string)*
: Comma-separated string to specify the minimum and maximum range of the horizontal axis. When not issued, the plot will automatically select the minimum and maximum viewable region. You can also input "tight" to automatically select the viewable range, with tighter overlap on both sides of the plot.

y_axis_range or *y_range* *(string)*
: Comma-separated string to specify the minimum and maximum range of the vertical axis. When not issued, the plot will automatically select the minimum and maximum viewable region. You can also input "tight" to automatically select the viewable range, with tighter overlap on both sides of the plot.

x_axis_invert or *x_invert* *(boolean)*
: If this option is set to True, the horizontal (x) axis will be inverted.

y_axis_invert or *y_invert* *(boolean)*
: If this option is set to True, the vertical (y) axis will be inverted.

x_tick_range or *x_ticks* *(string)*
: Comma-separated string to specify the ticks on the horizontal axis. Provided as x-minimum, x-maximum, increment (no spaces). When not issued, the plot will automatically select the horizontal tick locations.

y_tick_range or *y_ticks* *(string)*
: Comma-separated string to specify the ticks on the vertical axis. Provided as y-minimum, y-maximum, increment (no spaces). When not issued, the plot will automatically select the vertical tick locations.

`marker_size` or `ms` *(string)*
: Comma-separated string to specify the size of each of the markers in points provided by the `y_columns` attribute.

`marker_edge_width` or `mew` *(real)*
: Specifies the edge width size of each of the markers provided by the `y_columns` attribute.

`marker_face_color` or `mfc` *(string)*
: Comma-separated string to specify the color of each marker face provided by the `y_columns` attribute. Set to `w` for non-filled markers.

`marker_edge_color` or `mec` *(string)*
: Comma-separated string to specify the color of each marker edge provided by the `y_columns` attribute. Set to `w` for no marker edge.

`draw_legend` or `dl` *(boolean)*
: If this option is set to True, a legend representing each of the data columns provided by the `y_columns` attribute will be added to the plot.

`legend` or `leg` *(string)*
: Comma-separated string to specify legend contents, with each entry added as a separate line in the legend.

`legend_size` or `leg_size` *(int)*
: Font size for legend text.

`legend_location` or `leg_loc` *(enumerated)*
: Specifies the location of the legend if `draw_legend`=true. The following options for the location are available; best, center, lower_left, center_right, upper_left, center_left, upper_right, lower_right, upper_center, lower_center.

`line_format` or `lfmt` *(string)*
: Comma-separated string to specify the line formats assigned to the data provided by the `y_columns` attribute. The following options for the format of the line are available; _auto, (markers=- -- + o . , s v x < >, colors = b g r c m y k w 0.75. Markers may be combined with other markers and colors to define the line/marker configuration (--o for dashed dotted line).

`line_width` or `lw` *(string)*
: Comma-separated string to specify the line and point width(s) assigned to the data provided by the `y_columns` attribute.

`title_string` or `title` *(string)*
: Specifies the name of the title on the plot.

`title_size` or `tsize` *(integer)*
: Specifies the size of the title on the plot.

`x_label` *(string)*
: Specifies the label for the horizontal axis of the plot.

`y_label` *(string)*
: Specifies the label for the vertical axis of the plot.

`label_size` or `lsize` *(integer)*
: Specifies the size of the labels on the plot.

`tick_label_size` or `tlsize` *(integer)*
: Specifies the size of the tick labels on the plot.

`axis_line_width` or `alw` *(integer)*
: Specifies the size of the axis width on the plot.

`draw_grid` or `grid` *(boolean)*
: If this option is set to True, a vertical and horizontal background grid will be drawn at the tick locations assigned in `x_tick_range` and `y_tick_range`.

`font` *(enumerated)*
: Specifies the type of font used for the plot text. Available fonts are serif, sans-serif, cursive, fantasy and monospace.

`tex` or `latex_flag` *(boolean)*
: If this option is set to True, the application will attempt to use LaTex to render strings.

`note_string` or `note` *(string)*
: If this option is specified, the application will write the user requested `note_string` on the plot.

`note_size` or `nsize` *(integer)*
: Specifies the size of the text written when `note_string`="string".

`note_loc` or `nloc` *(string)*
: Specifies the location where the user requested `note_string` will be written on the plot. The location is specified in screen coordinates.

`plot_to_file` or `ptf` *(boolean)*
: If this option is set to True, the plot will be saved to a file. The file name will be created dynamically based on the `problem_name` and `output_format` attributes.

`output_format` or `ofmt` *(string)*
: Specifies the image file type extension when `plot_to_file`=true. The default is pdf.

`plot_to_screen` or `pts` *(boolean)*
: If this option is set to True, the plot will be displayed during the application execution.

`num_time_steps` or `steps` *(integer)*
: Specifies the number of timesteps when making time-dependent plots for animations.

`ignore_missing_files` or `imf` *(boolean)*
: If this option is set to True, the application will ignore any missing files that are provided by the `files` attribute.

`verbose` or `v` *(integer)*
: Set the verbose level for printing information to the screen. Each higher verbose level prints more information. If `verbose` is set to 0 (or less), only warning and error messages are printed. If `verbose` is set to 1, basic processing information is printed in addition to warning and error messages. This level is recommended. `verbose` levels greater than 1 provide information useful only for debugging.

Examples

In the following example, the pressure along the bottom wall of a nozzle is plotted from three different data sources. The x-axis is scaled to represent the non-dimensional length using the *x_scale* attribute. Additional options are provided to further configure the plot that is outputted.

```
acuPlotData -files diffuser.interp.dat,pb-c4.dat,pbot-exp.dat -y_cols 4 -x_scale
22.7272 -y_label "p/p_stag" -x_label x/H -title "Diffuser - Lower Surface Pressure"
-legend "AcuSolve,NASA Reference,Experiment" -pb diffuser_lower_pressure -lfmt
or,ob,ow -ms 1,5,5 -mec r,b,k -lw 2,2,2
```

or alternatively the options may be put into the configuration file `Acusim.cnf` as follows and then type `acuPlotData` in the AcuSolve cmd prompt (Windows) or the Linux command line to execute:

```
acuPlotData.pb=diffuser_lower_pressure
acuPlotData.files=diffuser.interp.dat,pb-c4.dat,pbot-exp.dat
acuPlotData.y_cols=4
acuPlotData.ptf=on
acuPlotData.x_scale=22.7272
acuPlotData.y_label="p/p_stag"
acuPlotData.x_label=x/H
acuPlotData.title="Diffuser - Lower Surface Pressure"
acuPlotData.legend="AcuSolve,NASA Reference,Experiment"
acuPlotData.lfmt=or,ob,ow
acuPlotData.ms=1,5,5
acuPlotData.mec=r,b,k
acuPlotData.lw=2,2,2
```

The following will be printed to the standard output of the terminal:

```
acuPlotData:  Plotting frame <1>
acuPlotData:  Evaluating argument <diffuser.interp.dat>
acuPlotData:  Found <1> files
acuPlotData:  Processing file <diffuser.interp.dat>
acuPlotData:  Plotting <500> points
acuPlotData:  Evaluating argument <pb-c4.dat>
acuPlotData:  Found <1> files
acuPlotData:  Processing file <pb-c4.dat>
acuPlotData:  Plotting <26> points
acuPlotData:  Evaluating argument <pbot-exp.dat>
acuPlotData:  Found <1> files
acuPlotData:  Processing file <pbot-exp.dat>
acuPlotData:  Plotting <36> points
acuPlotData:  Printing plot to file <diffuser_lower_pressure.png>
```

The above example produces an image file with extension `.png`, an example is shown below.

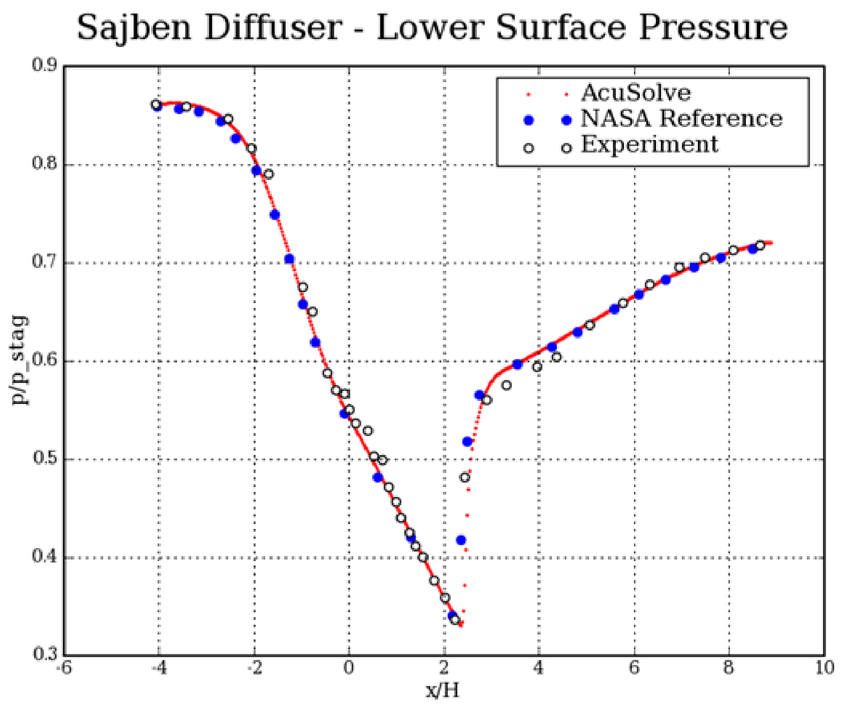


Figure 3:

In the following example, the skin friction coefficient along the wall is plotted from five different data sources. Additional options are provided to further configure the plot that is outputted.

```
acuPlotData -files exp.dat,sa.dat,sst.dat,ko.dat,rke.dat -y_cols 4 -y_label "Cf" -
x_label “Horizontal Location (m)” -title "Skin Friction Coefficient in an
Axisymmetric Diffuser" -legend "Experimental,AcuSolve-SA,AcuSolve-SST,AcuSolve-K-
Omega,AcuSolve-RKE" -lfmt ok,-r,-b,-g,-c -ms 2,1,1,1,1 -lw 2,2,2,2,2
```

or alternatively the options may be put into the configuration file `Acusim.cnf` as follows:

```
acuPlotData.pb=dr
acuPlotData.files= exp.dat,sa.dat,sst.dat,ko.dat,rke.dat
acuPlotData.y_cols=4
acuPlotData.ptf=on
acuPlotData.y_label="Cf"
acuPlotData.x_label=”Horizontal Location (m)”
acuPlotData.title="Skin Friction Coefficient in an Axisymmetric Diffuser"
acuPlotData.legend=Experimental,AcuSolve-SA,AcuSolve-SST,AcuSolve-K-Omega,AcuSolve-
RKE
acuPlotData.lfmt= ok,-r,-b,-g,-c
acuPlotData.ms=2,1,1,1,1
acuPlotData.lw=2,2,2,2,2
```

The above example produces an image file with extension `.png`, an example is shown below.

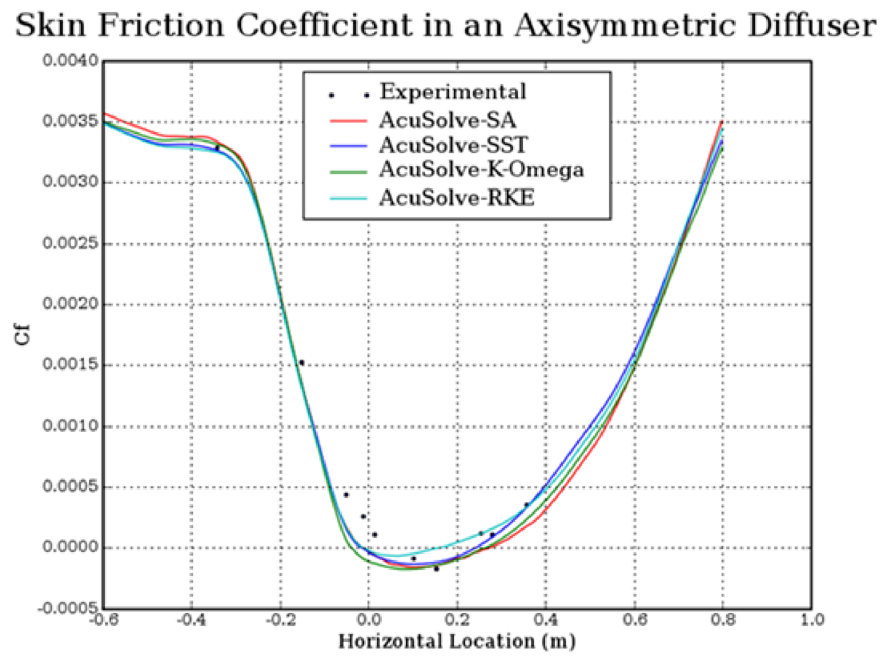

Skin Friction Coefficient in an Axisymmetric Diffuser
Experimental
AcuSolve-SA
AcuSolve-SST
AcuSolve-K-Omega
AcuSolve-RKE
Cf
Horizontal Location (m)
0.0040
0.0035
0.0030
0.0025
0.0020
0.0015
0.0010
0.0005
0.0000
-0.0005
-0.6
-0.4
-0.2
0.0
0.2
0.4
0.6
0.8
1.0

Figure 4:

User-Defined Function Programs

7

This chapter contains utility programs for preparing user-defined functions.

This chapter covers the following:

AcuMakeLib

Make a dynamic shared library from user codes, implementing user-defined functions. This program runs on Linux platforms only.

Syntax

`acuMakeLib [options]`

Type

AcuSolve User-Defined Functions Program

Description

AcuMakeLib is a utility program that builds a dynamic shared library from a set of user source codes containing user-defined functions. One or more libraries may then be loaded into AcuSolve. AcuMakeLib is for Linux platforms only.

AcuMakeLib relies on files stored in ACUSIM installed directories. As such, it requires the environment variables ACUSIM_HOME and ACUSIM_MACHINE. The `.acusim` file must be sourced for these variables to be defined.

AcuMakeLib is a Perl 5 script that first generates a Makefile, then executes it using the Unix make utility. AcuMakeLib is written in Perl 5. If execution of this script generates an error message, such as:

```
acuMakeLib - command not found
```

then Perl may not be on your system.

In the following, the full name of each option is followed by its abbreviated name and its type. For a general description of option specifications, see *Command Line Options and Configuration Files*. See below for more individual option details:

`help` or *`h`* *(boolean)*
: If set, the program prints a usage message and exits. The usage message includes all available options, their current values and the source of their current values.

`user_library` or *`lib`* *(string)*
: The name of the resulting dynamic shared library.

`user_source_files` or *`src`* *(string)*
: Comma-separated list of user source files. The comma-separated fields may contain shell file name generation characters. All files must have extension "`.c`" (for C codes) or "`.f, .for, .f77, .f90, .f95`" (for Fortran codes). If *`user_source_files`* is set to _auto, all files with extension "`.c`" and "`.f, .for, .f77, .f90, .f95`" in the current directory are assumed.

`optimization` or *`opt`* *(enumerated)*
: Compiler optimization level:

O	Default system optimization
O1	Compiler optimization O1

O2	Compiler optimization O2
O3	Compiler optimization O3
debug	Compile with debug option

`c_compiler` or `cc` (string)
: Specify C compiler command. If the default _auto option is selected, the script searches for Intel's icc compiler at first, then the gcc compiler later in the system. If you want to use a specific compiler, the compiler command can be specified by this option.

`f_compiler` or `fc` (string)
: Specify Fortran compiler command. If the default _auto option is selected, the script searches for Intel's ifort compiler at first, then the gfortran compiler later in the system. If you want to use a specific compiler, the compiler command can be specified by this option.

`c_compiler_flags` or `cflags` (string)
: These flags are added to the built-in C compiler flags. The default C compiler is gcc.

`fortran_compiler_flags` or `fflags` (string)
: These flags are added to the built-in Fortran compiler flags. The default Fortran compiler is gfortran.

`linker_flags` or `lflags` (string)
: These flags are added to the built-in linker flags.

`make_clean` or `clean` (boolean)
: Clean the directory before compiling.

`verbose` or `v` (integer)
: Set the verbose level for printing information to the screen. Each higher verbose level prints more information. If *`verbose`* is set to 0 (or less), only warning and error messages are printed. If *`verbose`* is set to 1, basic processing information is printed in addition to warning and error messages. This level is recommended. Verbose levels greater than 1 provide information useful only for debugging.

Examples

To build a dynamic shared library, named `libusr.so`, from C files `usr*.c` and Fortran files `fusr*.f` using GNU Compiler Collection, issue the command

```
acuMakeLib -lib libusr.so -src "usr*.c,fusr*.f" -cc gcc -fc gfortran
```

or alternatively place the options in the configuration file `Acusim.cnf` as follows:

```
user_library= libusr.so
user_source_files= usr*.c,fusr*.f
```

and invoke AcuMakeLib as:

```
acuMakeLib
```

Note: On the command line usr*.c,fusr*.f was enclosed in a pair of double quotes so that the shell would not expand it. Instead, the expansion was done in AcuMakeLib.

The Unix shell file name generation characters "*", "?", and "[]" are accepted and expanded by AcuMakeLib.

AcuMakeDll

Make a dynamic linked library from user codes, implementing user-defined functions. This program runs on Windows platforms only.

Syntax

`acuMakeDll [options]`

Type

AcuSolve User-Defined Functions Program

Description

AcuMakeDll is a utility program that builds a dynamic linked library from a set of user source codes containing user-defined functions. One or more libraries may then be loaded into AcuSolve. AcuMakeDll is for Windows platforms only.

AcuMakeDll is written in Perl 5. If execution of this script generates an error message, such as

```
acuMakeDll - command not found
```

then Perl may not be on your system.

AcuMakeDll relies on files stored in ACUSIM installed directories. As such, it requires the environment variables ACUSIM_HOME and ACUSIM_MACHINE.

AcuMakeDll supports Intel C++ Compiler (version 8 to 16), Microsoft Visual C++Compiler (version 8 to 14) and Intel Fortran Compiler (version 8 to 16). Other compilers may require extending AcuMakeDll.

AcuMakeDll looks for available C++ and Fortran compilers in your system, and their build environments are invoked at the run time. If your system has both Intel and Microsoft compilers, AcuMakeDll chooses the former. Version eight of both Intel C++ and Fortran Compilers support only 32-bit compilation. Search paths used in AcuMakeDll are the following default installation directories:

Intel C++ Compiler Search Path

Version 19 (2019)	%ProgramFiles(x86)%\IntelSWTools \compilers_and_libraries_2019\windows\bin\iclvars.bat
Version 18 (2018)	%ProgramFiles(x86)%\IntelSWTools \compilers_and_libraries_2018\windows\bin\iclvars.bat
Version 17 (2017)	%ProgramFiles(x86)%\IntelSWTools \compilers_and_libraries_2017\windows\bin\iclvars.bat
Version 16 (2016)	%ProgramFiles(x86)%\IntelSWTools \compilers_and_libraries_2016\windows\bin\iclvars.bat
Version 15 (2015)	%ProgramFiles(x86)%\Intel\Composer XE 2015\bin\iclvars.bat
Version 14 (2013 SP1)	%ProgramFiles(x86)%\Intel\Composer XE 2015\bin\iclvars.bat

Version 13 (2013)	%ProgramFiles(x86)%\Intel\Composer XE 2013\bin\iclvars.bat
Version 12.1 (2011)	%ProgramFiles(x86)%\Intel\Composer XE 2011 SP1\bin \iclvars.bat
Version 12.0 (2011)	%ProgramFiles(x86)%\Intel\ComposerXE-2011\bin\iclvars.bat
Version 11.1	%ProgramFiles(x86)%\Intel\Compiler\11.1\xxx\bin\iclvars.bat
Version 11.0	%ProgramFiles(x86)%\Intel\Compiler\11.0\xxx\cpp\bin \iclvars.bat
Version 10.1	%ProgramFiles(x86)%\Intel\Compiler\C++\10.1.xxx\{EM64T, IA32}\Bin\iclvars.bat
Version 10.0	%ProgramFiles(x86)%\Intel\Compiler\C++\10.0.xxx\{EM64T, IA32}\Bin\iclvars.bat
Version 9.1	%ProgramFiles(x86)%\Intel\Compiler\C++\9.1\{EM64T, IA32}\Bin\iclvars.bat
Version 9.0	%ProgramFiles(x86)%\Intel\Compiler\C++\9.0\{EM64T, IA32}\Bin\iclvars.bat
Version 8.0, 8.1	%ProgramFiles(x86)%\Intel\CPP\Compiler80\Ia32\Bin\iclvars.bat

where xxx is the three-digit build number.

Microsoft Visual C++ Search Path for the Community Version

Version 19	%ProgramFiles(x86)%\Microsoft Visual Studio\2019\Community \VC\Auxiliary\Build\vcvarsall.bat

Microsoft Visual C++ Search Path

Version 19	%ProgramFiles(x86)%\Microsoft Visual Studio\2019\Enterprise \VC\Auxiliary\Build\vcvarsall.bat
Version 19	%ProgramFiles(x86)%\Microsoft Visual Studio\2019\Professional \VC\Auxiliary\Build\vcvarsall.bat
Version 8--16	%ProgramFiles(x86)%\Microsoft Visual Studio xxx\VC \vcvarsall.bat where xxx is the version number: {16.0, 15.0, 14.0, 13.0, 12.0, 11.0, 10.0, 9.0, 8.0}

> **Note:** Intel 2017, 2018, 2019 do not support the VS2010 option. Using the VS2015 option instead.

Intel Fortran Compiler Search Path

Version 16 (2016)	%ProgramFiles(x86)%\IntelSWTools\compilers_and_libraries \windows\bin\ifortvars.bat
Version 15 (2015)	%ProgramFiles(x86)%\Intel\Composer XE 2015\bin\ifortvars.bat
Version 14 (2013 SP1)	%ProgramFiles(x86)%\Intel\Composer XE 2013 SP1\bin \ifortvars.bat
Version 13 (2013)	%ProgramFiles(x86)%\Intel\Composer XE 2013\bin\ifortvars.bat
Version 12.1 (2011)	%ProgramFiles(x86)%\Intel\Composer XE 2011 SP1\bin \ifortvars.bat
Version 12.0 (2011)	%ProgramFiles(x86)%\Intel\ComposerXE-2011\bin\ifortvars.bat
Version 11.1	%ProgramFiles(x86)%\Intel\Compiler\11.1\xxx\bin\ifortvars.bat
Version 11.1	%ProgramFiles(x86)%\Intel\Compiler\11.0\xxx\Fortran\Bin \ifortvars.bat
Version 10.1	%ProgramFiles(x86)%\Intel\Compiler\Fortran\10.1.xxx\{em64t, IA32}\bin\ifortvars.bat
Version 10.0	%ProgramFiles(x86)%\Intel\Compiler\Fortran\10.0.xxx\{em64t, IA32}\bin\ifortvars.bat
Version 9.1	%ProgramFiles(x86)%\Intel\Compiler\Fortran\9.1\{em64t, IA32}\bin\ifortvars.bat
Version 9.0	%ProgramFiles(x86)%\Intel\Compiler\Fortran\9.0\{em64t, IA32}\bin\ifortvars.bat
Version 8.1, 8.0	%ProgramFiles(x86)%\Intel\Fortran\compiler80\IA32\BIN \ifortvars.bat

where xxx is the three-digit build number.

If the Intel or Microsoft compiler is installed outside of the above search paths, you can still use -cbat and -fbat options to directly specify the full paths of `iclvars.bat` and `ifortvars.bat` files. You also need to provide necessary arguments for these batch files.

If your system does not have a C/C++ compiler, you may download and install a free compiler provided by Microsoft:

For Windows 10: Microsoft Visual Studio Express 2015 for Windows (contains 64-bit compiler)

www.visualstudio.com/downloads/download-visual-studio-vs#d-express-windows-10

For Windows 7 and 8: Microsoft Visual Studio Express 2015 for Windows

www.visualstudio.com/downloads/download-visual-studio-vs#d-express-windows-desktop

If your system has compilers other than Intel or Microsoft, you may start off from their Command Prompt. At first, execute the following command from the AcuSolve command prompt:

```
echo %ACUSIM_ROOT%\bin\acusim.bat
```

then copy and paste the output of the above command into the compiler's Command Prompt. This will bring up AcuSolve's environment variables. Once the compiler's Command Prompt has AcuSolve environment variables such as %ACUSIM_HOME% and %ACUSIM_MACHINE%, you can invoke AcuMakeDll to create a dynamic link library.

In the following, the full name of each option is followed by its abbreviated name and its type. For a general description of option specifications, see *Command Line Options and Configuration Files*. See below for more individual option details:

`help` *or* `h` *(boolean)*
: If set, the program prints a usage message and exits. The usage message includes all available options, their current values and the place where each option is set.

`user_library` *or* `lib` *(string)*
: The name of the resulting dynamic linked library.

`user_source_files` *or* `src` *(string)*
: Comma-separated list of user source files. The comma-separated fields may contain shell file name generation characters. All files must have extension "`.c`" (for C codes) or "`.f, .for, .f77, f90, f95`" (for Fortran codes). If `user_source_files` is set to _auto, all files with extension "`.c`" and "`.f, .for, .f77, f90, f95`" in the current directory are assumed.

`c_setup_path_arg` *or* `cbat` *(string)*
: Specify the full path to the `.bat` file plus the argument to setup the C build environment. The specified path should contain the `.bat` file name: `iclvars.bat`, `ipsxe-comp vars.bat` or `vcvarsall.bat`. An example usage of this option is:

-cbat "C:\Program Files (x86)\IntelSWTools\compilers_and_libraries\windows\bin\iclvars.bat x86_amd64"

-cbat "C:\Program Files (x86)\Microsoft Visual Studio\2019\Enterprise\VC\Auxiliary\Build\vcvarsall.bat x86_amd64"

-cbat "C:\Program Files (x86)\Microsoft Visual Studio\2019\Professional\VC\Auxiliary\Build\vcvarsall.bat x86_amd64"

`fortran_setup_path_arg` *or* `fbat` *(string)*
: Specify the full path to the `.bat` file plus the argument to setup the Fortran build environment. The specified path should contain the `.bat` file name: `ifortvars.bat` or `ipsxe-comp-vars.bat`. An example usage of this option is:

-fbat "C:\Program Files (x86)\IntelSWTools\compilers_and_libraries\windows\bin\ifortvars.bat intel64 vs2010"

`optimization` *or* `opt` *(enumerated)*
: Compiler optimization level:

O	Default system optimization

debug	Compile with debug option

`c_compiler_flags` or `cflags` (string)
These flags are added to the built-in C compiler flags.

`fortran_compiler_flags` or `fflags` (string)
These flags are added to the built-in Fortran compiler flags.

`linker_flags` or `lflags` (string)
These flags are added to the built-in linker flags.

`message_passing_type` or `mp` (enumerated)
Message passing type to link the DLL against:

none	No message passing environment
openmp	Link against openmp message passing environment
mpi	Link against mpi message passing environment
pmpi	Link against Platform mpi message passing environment
impi	Link against Intel mpi message passing environment
msmpi	Link against Microsoft mpi message passing environment

`make_clean` or `clean` (boolean)
If set, the program performs a clean make before compiling the user specified source code.

`verbose` or `v` (integer)
Set the verbose level for printing information to the screen. Each higher verbose level prints more information. If *`verbose`* is set to 0 (or less), only warning and error messages are printed. If *`verbose`* is set to 1, basic processing information is printed in addition to warning and error messages. This level is recommended. *`verbose`* levels greater than 1 provide information useful only for debugging.

Examples

To build a dynamic linked library, named `libusr.dll`, from C files `usr*.c` and Fortran files `fusr*.f`, issue the command

```
acuMakeDll -lib libusr.dll -src "usr*.c,fusr*.f" -mp pmpi
```

or alternatively place the options in the configuration file `Acusim.cnf` as follows:

```
user_library= libusr.dll
user_source_files= usr*.c,fusr*.f
```

and invoke AcuMakeDll as:

```
acuMakeDll
```

Note: On the command line usr*.c,fusr*.f was enclosed in a pair of double quotes so that the shell would not expand it. Instead, the expansion was done in AcuMakeDll.

The Windows shell file name generation characters "*", "?", and "[]" are accepted and expanded by AcuMakeDll.

Note: It is necessary to link against the message passing environment libraries when building a UDF that will be run in parallel.

License Manager Programs 8

This chapter contains programs associated with the classical Acusim license manager.

This chapter covers the following:

These programs are typically used by the system administer. However, you might be interested in

```
acuLmg -query
```

which provides the status of the license manager as well as the list of checked out and queued licenses.

Note: These programs are only used when *lm_service_type* = classical.

AcuLmg

Network license manager start, stop, and status query.

Syntax

`aculmg [options]`

Type

AcuSolve License Manager Program

Description

AcuLmg is used to start, stop, query, and test the network license manager. Even though the network license manager daemon (AcuLmd) runs only on the license server, AcuLmg may be issued from any machine on the network.

The option lm_checkout specifies the address of the license co-processor AcuLmco. This executable is launched on the local host by the module requesting a license, such as AcuLmg, issued with checkout_license option, and AcuSolve.

> **Note:** AcuLmco automatically starts the daemon AcuLmd if it is not already running.

The amount of time, in seconds, to camp in the license queue while waiting for a license is set by the option lm_queue_time.

Commands that need to communicate with the license server usually do so with rsh, or remesh on HP-UX. But if only ssh is supported, then the *`remote_shell`* parameter should be used:

```
acuLmg -start -rsh "ssh"
```

The daemon acuLmd logs its messages into the system `.log` file. Consult the `system file /etc/syslog.conf` for the location of the system `.log` file on your system.

In the following, the full name of each option is followed by its abbreviated name and its type. For a general description of option specifications, see *Command Line Options and Configuration Files*. See below for more individual option details:

`help` or *`h`* *(boolean)*
: If set, the program prints a usage message and exits. The usage message includes all available options, their current values and the source of their current values.

`start_daemon` or *`start`* *(boolean)*
: Start the network license manager daemon, AcuLmd, on the server machine. If the daemon is already running the program does not start a new one.

`stop_daemon` or *`stop`* *(boolean)*
: Stop the network license manager daemon, AcuLmd. If a daemon is servicing a license, after it is stopped via this command, it is automatically restarted again by the job(s) being serviced.

`restart_daemon` or `restart` *(boolean)*
Restart the network license manager daemon, AcuLmd, on the server machine. If the daemon is already running the program does not restart a new one. May be used in both stopped and started states.

`query_license` or `query` *(boolean)*
Query the status of the network license manager.

`report` or `stats` *(boolean)*
Report the usage statistics of the licenses.

`checkout_code` or `code` *(enumerated)*
Name of the code being queried, reported on, or checked out:

AcuSolve	AcuSolve
AcuView	acuView
AcuTrace	AcuTrace
AcuFwh	AcuFwh
AcuMeshSim	AcuMeshSim
AcuCatia2Acis	AcuCatia2Acis (deprecated)
AcuToken	AcuToken (deprecated)
LesLib	Linear algebra module\
_all	All licensed programs

`checkout_license` or `co` *(boolean)*
Checkout a license. This option simulates checking out a license for a period specified by checkout_time option. The purpose of this option is to test the license manager, to ensure proper operation.

`checkout_time` or `cot` *(integer)*
The time, in seconds, to wait before checking in the license which was checked out by the checkout_license option.

`checkout_num_processors` or `np` *(integer)*
Number of processors to checkout.

`lm_daemon` or `lmd` *(string)*
Full address of the network license manager daemon, AcuLmd, that runs on the server host. When this option is set to _auto, the value is internally changed to `$ACUSIM_HOME/$ACUSIM_MACHINE/bin/acuLmd`.

`lmtype` or `lm_service_type` *(enumerated)*
Type of the license manager service:

hwu	Altair Units licensing

token	Token based licensing
classical	Classical Acusim style licensing

`lm_server_host` or `lmhost` *(string)*
: Name of the server machine on which the network license manager runs. When this option is set to _auto, the local host name is used.

`lm_port` or `lmport` *(integer)*
: TCP port number that the network license manager uses for communication.

`lm_license_file` or `lmfile` *(string)*
: Full address of the license file. This file is read frequently by the network license manager. When this option is set to _auto, the value is internally changed to `$ACUSIM_HOME/$ACUSIM_MACHINE/license.dat`.

`lm_log_file` or `lmlog` *(string)*
: Full path to the file in which to record licensing check-out and check-in events. When this value is set to _none, the messages are directed to the system log.

`lm_restricted_access` or `lmra` *(boolean)*
: Restrict access to the license manager start/stop/restart operations.

`lm_checkout` or `lmco` *(string)*
: Full address of the license checkout co-processor, AcuLmco. This process is spawned by AcuLmg on the local machine. When this option is set to _auto, the value is internally changed to `$ACUSIM_HOME/$ACUSIM_MACHINE/bin/acuLmco`.

`lm_queue_time` or `lmqt` *(integer)*
: The time, in seconds, to camp on the license queue before abandoning the wait. Up to a maximum of seven days is allowed.

`remote_shell` or `rsh` *(string)*
: Remote shell executable for starting AcuLmd on a remote machine. Usually this is rsh (or remesh on HP-UX) but can be set to ssh if needed. If _auto, executable is determined internally.

`verbose` or `v` *(integer)*
: Set the verbose level for printing information to the screen. Each higher verbose level prints more information. If verbose is set to 0 (or less), only warning and error messages are printed. If verbose is set to 1, basic processing information is printed in addition to warning and error messages. This level is recommended. verbose levels greater than 1 provide information useful only for debugging.

Examples

To start the network license manager issue the command

```
acuLmg -start
```

This starts the daemon on the license server if it is not already running. Otherwise it simply returns with a message indicating that the daemon has already started.

To stop the network license manager issue the command

```
acuLmg -stop
```

This will stop the daemon which is running on the license server.

Note: Any process that has already checked out a license will attempt to restart the daemon.

To restart the network license manager issue the command

```
acuLmg -restart
```

This may be used in either stopped or started states. In the latter it simply returns with a message indicating that the daemon has already started.

To query the status of the network license manager issue the command

```
acuLmg -query
```

This will produce a message such as:

```
acuLmg: License server: pelican
acuLmg: Product: acuSolve
acuLmg: Number of licenses: 10
acuLmg: Last expiration date: Jul-05-2007
acuLmg: Licenses checked out (2):
acuLmg: (13273) 4p Sun Apr 29 23:23:13 2007 jaiman@oriole
acuLmg: (1308) 16p Sun Apr 29 23:23:28 2007 farzin@tern
acuLmg: Licenses queued (0):
acuLmg: None
```

indicating that five jobs are running and none are waiting on the queue.

To report statistics on license usage, issue the command:

```
acuLmg -stats
```

This will produce a message like:

```
acuLmg: License server: pelican
acuLmg: Product: acuSolve
acuLmg: Number of licenses: 10
acuLmg: Last expiration date: Jul-05-2007
acuLmg: Daemon start time: Tue Apr 10 15:10:34 2007
acuLmg: Current server time: Sun Apr 29 23:24:43 2007
acuLmg: Ave. lic. checked out: 0.3814
acuLmg: Max. lic. checked out: 5
acuLmg: Ave. lic. elapsed time: 1068 Sec
acuLmg: Max. lic. elapsed time: 1.204e+05 Sec
acuLmg: Ave. lic. processors: 3.443
acuLmg: Max. lic. processors: 16
acuLmg: Ave. lic. queued: 0
acuLmg: Max. lic. queued: 0
acuLmg: Ave. lic. queue time: 0 Sec
acuLmg: Max. lic. queue time: 0 Sec
```

To simulate checking out a license, issue the command:

```
acuLmg -co -cot 1000
```

This checks out a license and sleeps for 1000 seconds.

The above examples assume that the license manager options are already placed in the (system) configuration file `Acusim.cnf` by the system administer. This file will contain options such as:

```
lm_daemon= /acusim/SGI/latest/bin/acuLmd
lm_server_host= eagle
lm_port= 41994
lm_license_file= /acusim/license.dat
SGI.lm_checkout= /acusim/SGI/latest/bin/acuLmco
HAL.lm_checkout= /acusim/HAL/latest/bin/acuLmco
HP.lm_checkout= /acusim/HP/latest/bin/acuLmco
lm_queue_time= 3600
```

In this example, the network license daemon, acuLmd, runs on the SGI server host called eagle. The SGI executable is located at `/acusim/SGI/latest/bin/acuLmd`. The TCP port used for communication between all licensing modules is 41994. The license keys are placed in the file `/acusim/license.dat`. This file may contain any number of license keys for any number of network license daemons. Only the applicable entries are processed; all others are ignored. The above file may look like the following:

Table 38:

# key	code	no.lic	expiration	uid	hostid	host
bqPC3K2KxhMNZOFegr	AcuSolve	5	Jan-01-2012	4753	0x690a13bf	eagle
bqPC3K2KxhMNZOFegr	AcuSolve	3	Jun-01-2013	4753	0x690a13bf	eagle
n0BGLUJ5/ imf88k06M1SDQ	AcuSolve	2	Jan-01-2011	6435	0x73c43ad	hawk

where, for the network license server eagle with host ID 0x690a13bf, there are 5 simultaneous network licenses valid until January 1, 2012, and 3 simultaneous network licenses valid until June 1, 2013.

The executable acuLmi may be used to get the host ID of a machine. Simply run this command on the server with no argument, such as:

```
acuLmi
```

This produces results similar to the following:

```
machine: SGI64
host: eagle
hostid: 0x690a13bf
```

AcuLmd

Network license manager daemon.

Syntax

`acuLmd [options]`

Type

AcuSolve License Manager Program

Description

AcuLmd is the network license manager daemon. This program runs in the background and checks out licenses to the solver. This program is typically not invoked by itself, but rather the program acuLmg is used to start and stop AcuLmd.

In the following, the full name of each option is followed by its abbreviated name and its type. For a general description of option specifications, see *Command Line Options and Configuration Files*. See below for more individual option details:

`help` or `h` *(boolean)*
: If set, the program prints a usage message and exits. The usage message includes all available options, their current values and the source of their current values.

`lm_port` or `lmport` *(integer)*
: TCP port number that the network license manager uses for communication.

`lm_license_file` or `lmfile` *(string)*
: Full address of the license file. This file is read frequently. When this option is set to _auto, the value is internally changed to `$ACUSIM_HOME/$ACUSIM_MACHINE/license.dat`.

`lm_log_file` or `lmlog` *(string)*
: Full path to the file in which to record licensing check-out and check-in events. When this value is set to _none, the messages are directed to the system log.

`lm_restricted_access` or `lmra` *(boolean)*
: Restrict access to the license manager start/stop/restart operations.

`lm_uninstall_service` or `lmrm` *(boolean)*
: Uninstall the license manager service. Used on Windows platforms only.

Examples

To manually start the daemon from the license server machine, issue the command

```
acuLmd
```

If a daemon is already running, AcuLmd immediately returns. Otherwise, it spawns a second copy of AcuLmd to the background.

The daemon acuLmd logs its messages into the system `.log` file by default. Consult the system file `/etc/syslog.conf` for the location of the system `.log` file on your system. This behavior can be overridden by setting the lm_log_file option to redirect the output to a user specified file.

AcuLmi

Extracts host information for issuing a license.

Syntax

`acuLmi`

Type

AcuSolve License Manager Program

Description

AcuLmi provides the host ID of a machine, which is used for issuing a license.

Examples

Simply run this command on the server with no argument:

```
acuLmi
```

This produces results similar to the following:

```
machine: SGI64
host: eagle
hostid: 0x690a13bf
```

The string hostid on Windows machines is extracted from the disk drive (of the C: partition). Prior to version 1.7c the MAC address of a NIC card was used.

AcuLmf

Updates a license file.

Syntax

`acuLmf [options]`

Type

AcuSolve License Manager Program

Description

In the following, the full name of each option is followed by its abbreviated name and its type. For a general description of option specifications, see *Command Line Options and Configuration Files*. See below for more individual option details:

`help` or `h` (boolean)

If set, the program prints a usage message and exits. The usage message includes all available options, their current values, and the place where each option is set.

`lm_license_file` or `lmfile` (string)

Full address of the license file. When this option is set to _auto, the value is internally changed to `$ACUSIM_HOME/$ACUSIM_MACHINE/license.dat`.

`input_file` or `infile` (string)

Input license file name. Extra lines and characters will be removed and the license information properly formatted.

`verbose` or `v` (integer)

Set the verbose level for printing information to the screen. Each higher verbose level prints more information. If verbose is set to 0 (or less), only warning and error messages are printed. If verbose is set to 1, basic processing information is printed in addition to warning and error messages. This level is recommended. verbose levels greater than 1 provide information useful only for debugging.

Examples

This script is provided to facilitate updating the license file. For example,

```
acuLmf -lmfile /Acusim/license.dat -infile /tmp/license.txt
```

reads the license information from `/tmp/license.txt` and places it in the license file `/Acusim/license.dat`. The former file does not need to be a properly formatted license file; it can be the entire email from AcuSim that contains the license information. Specifying -lmfile is rarely necessary since normally the correct value is specified in the installed `Acusim.cnf` file. The license information may also be read in from standard input by specifying -infile _stdin. This is the default, so all of the following are equivalent:

```
acuLmf -infile /tmp/license.txt
acuLmf -infile _stdin < /tmp/license.txt
acuLmf < /tmp/license.txt
```

```
acuLmf
```

<user types or pastes in content of /tmp/license .txt>

Utility Scripts 9

AcuSolve contains a number of command line utility scripts that can be used for pre-processing the model, post-processing the model, running the model, and for monitoring the solution as it progresses.

These scripts are shipped with the distribution and are located in the /script and /bin directories of the installation. Each script has a full help listing that can be accessed using the "-h" command line option. The list below provides a brief summary of each script. Additional details can be found in the MANIFEST `.txt` file that is shipped with the distribution.

acuDdc	External domain decomposition for AcuPrep.
acuDmg	AcuSolve directory management tool for deleting output files.
acuDplace	Script to determine optimum process placement for Altix.
acuEnSight6To5	Convert an Ensight6 file to Ensight5.
acuEnv	Script to set AcuSolve's environment variables.
acuExtrudeSurf	Extrude a surface mesh to form a volume.
acuFmtArray	ASCII/binary conversion of AcuSolve files.
acuFv2H3D	Convert a FieldView file to H3D.
acuFwh	Ffowcs-Williams-Hawking solver.
acuGetData	Get AcuSolve results data directly from the AcuSolve output database.
acuGetCpCf	Script to calculate pressure and friction coefficients.
acuGetNodeSubset	Script to extract a subset of nodal output to disk.
acuHeatBalance	Compute the energy balance from an AcuSolve solution database.
acuImport	Import CFD files to AcuSolve.
acuInterp	Interpolate AcuSolve solution to requested points.
acuLiftDrag	Script to calculate lift and drag coefficients for one or more surfaces.
acuLmco	License manager co-processor for Linux based platforms.
acuLmd	License manager daemon.
acuLmf	Install license in the license file.
acuLmg	License manager user tool.

acuLmi	License machine info.
acuMakeDll	User-defined function make utility for AcuSolve (Windows).
acuMakeLib	User-defined function make utility for AcuSolve (Linux).
acuMesh2Tet	Convert a mesh to all tet mesh.
acuModSlv	Script to modify solver commands for utilization with the fast restart option.
acuOdb	Translate AcuSolve results to Abaqus ODB.
acuOptiStruct	Script to generate an OptiStruct input deck for thermal stress analysis.
acuOut	A GUI-based output translator to convert AcuSolve results into different formats.
acuPbc	Extract the build periodic BC set.
acuPerl	Windows `.bat` file to run AcuSolve perl scripts.
acuPev	Project EigenVectors for `FLEXIBLE_BODY`.
acuPlotData	Plot data from text files using matplotlib.
acuPmb	Pallas MPI Benchmark V2.2.
acuPorous	Simple curve-fit to get Darcy/Forch. coefficients to create porosity model inputs.
acuPrep	Prep the user data to be run by AcuSolve.
acuProbe	Plot time series solution and convergence.
acuProj	Project the solution to a new mesh.
acuPython	Run a Python script using AcuSolve's Python environment.
acuReport	Generate a report from a solution database.
acuRun	Script to run AcuPrep, AcuView and AcuSolve.
acuRunFwh	Script to acuFwh.
acuRunTrace	Compute steady particle traces.
acuScs	Extract simply connected sub meshes.
acuSetNodeValue	Set nodal values based on a user equation.
acuSflux	Compute flux table for `SOLAR_RADIATION`.

acuSif	Split internal faces of a mesh.
acuSig	Signal to AcuSolve (to output, stop, and so on.)
acusim	Set ACUSIM environment.
acuSolve	Main solver.
acuSolve-cuda-sp	NVIDIA GPU version of AcuSolve.
acuSolve-rocm-sp	AMD GPU version of AcuSolve.
acuSpec	Compute power spectrum using FFT or MEM.
acuSub	Submit an AcuSolve job to PBS/LSF/CCS queue.
acuSurf	Extract external surfaces of a mesh for imposing boundary conditions.
acuTherm	Translates the AcuSolve solution to an input file for TAITherm.
acuTrace	New particle tracer (run through acuRunTrace).
acuTrace2Ensight	Convert AcuTrace results to EnSight.
acuTrans	Translate AcuSolve results to other formats.
acuTransTrace	Translate AcuTrace results to other formats.
acuView	Calculate radiation view factors.

Intellectual Property Rights Notice

Technical Support

Altair's support resources include engaging learning materials, vibrant community forums, intuitive online help resources, how-to guides, and a user-friendly support portal.

Visit Customer Support to learn more about our support offerings.

Index