

SANDIA REPORT

SAND2025-11623O

Printed September 11, 2025



Sandia
National
Laboratories

SIERRA Fire Module: Fuego User Manual – Version 5.26

Sierra Thermal Fluids Team

Prepared by
Sandia National Laboratories
Albuquerque, New Mexico 87185
Livermore, California 94550

Issued by Sandia National Laboratories, operated for the United States Department of Energy by National Technology & Engineering Solutions of Sandia, LLC.

NOTICE: This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government, nor any agency thereof, nor any of their employees, nor any of their contractors, subcontractors, or their employees, make any warranty, express or implied, or assume any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represent that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government, any agency thereof, or any of their contractors or subcontractors. The views and opinions expressed herein do not necessarily state or reflect those of the United States Government, any agency thereof, or any of their contractors.

Printed in the United States of America. This report has been reproduced directly from the best available copy.

Available to DOE and DOE contractors from

U.S. Department of Energy
Office of Scientific and Technical Information
P.O. Box 62
Oak Ridge, TN 37831

Telephone: (865) 576-8401
Facsimile: (865) 576-5728
E-Mail: reports@osti.gov
Online ordering: <http://www.osti.gov/scitech>

Available to the public from

U.S. Department of Commerce
National Technical Information Service
5301 Shawnee Road
Alexandria, VA 22312

Telephone: (800) 553-6847
Facsimile: (703) 605-6900
E-Mail: orders@ntis.gov
Online order: <https://classic.ntis.gov/help/order-methods>



Contents

1. Fuego Overview	9
2. Releases & Deprecations	11
2.1. Version 5.28 (pre-release)	11
2.2. Version 5.26 (latest - 2025/09/11)	11
2.3. Version 5.24 (2025/03/19)	12
2.4. Version 5.22 (2024/10/08)	12
2.5. Version 5.20 (2024/06/17)	13
2.6. Version 5.18 (2024/02/23)	13
3. Theory	14
3.1. Introduction	14
3.2. Math Models	16
3.3. Particles	148
3.4. Numerics	198
4. Simulation Setup	289
4.1. Input File Basics	289
4.2. Mesh	295
4.3. Linear Solvers	298
4.4. Material Properties	304
4.5. Solution Control	319
4.6. Time Control	327
4.7. Transfer	327
4.8. Units	332
4.9. Solution Options	333
4.10. Boundary Conditions	344
4.11. Initial Conditions	357
4.12. Fire Features	358
4.13. Post-Processing	371
4.14. User Customization	376
4.15. Output Reference	382
4.16. Restart Reference	390
4.17. IO Region	392
4.18. Particles	394
5. Tutorials & Examples	403
5.1. Sierra 100	403
6. Running & Troubleshooting	448
6.1. Running Fuego	448
6.2. Diagnostic Output	455
6.3. Troubleshooting	455

7. Command References	457
7.1. Domain	457
7.2. Transfer	526
7.3. Solvers	542
7.4. Procedure	593
7.5. Time Control	611
7.6. Solution Control	617
7.7. Fuego Region	638
7.8. Particle Region	790
7.9. Average Region	930
7.10. Input Output Region	998
7.11. Initial Conditions	1019
7.12. Boundary Conditions	1029
References	1211
Bibliography	1211

List of Figures

Fig. 2.1. Fuego Deprecation Timeline	14
Fig. 3.1. Ordinate Direction Definition	27
Fig. 3.2. Control volume definition for an exposed surface. The degree of freedom is noted by the dark circle; shading of the control volume is also indicated. Triangles and diamonds depict surface- and volume-based integration point locations.	63
Fig. 3.3. Model geometry for Magnussen's Eddy Dissipation Concept. The control volume is comprised of two zones; the properties of each zone are assumed to be adequately represented by a single set of values (i.e., lumped or perfectly stirred). The mass exchange between the zones is controlled by turbulent mixing.	67
Fig. 3.4. Assumed flame surface geometry. L is the integral turbulent length scale. The reaction zone thickness is characterized by the Kolmogorov dissipative turbulent length scale, η	69
Fig. 3.5. Pentagamma function and asymptotic expansion	120
Fig. 3.6. Representative mesh layout for 1-D composite fire boundary condition	127
Fig. 3.7. Mesh index definition for 1-D composite fire boundary condition	130
Fig. 3.8. The Fuego-computed particle velocity and trajectory are compared with predictions from (3.713) and (3.714) (left) and error (right). Parameters from Table 3.14 with $d_p = 3 \cdot 10^{-2}$	174
Fig. 3.9. The Fuego-computed particle velocity and trajectory are compared with predictions from (3.713) and (3.714) (left) and error (right). Parameters from Table 3.14 with $d_p = 3 \cdot 10^{-3}$	174
Fig. 3.10. The Fuego-computed particle velocity and trajectory are compared with predictions from (3.713) and (3.714) (left) and error (right). Parameters from Table 3.14 with $d_p = 3 \cdot 10^{-3}$	175

Fig. 3.11.	The mean square displacement of particles is shown as a function of time. Different curves show the statistical noise associated with different numbers of particles considered.	176
Fig. 3.12.	d^2 (left) and T_p (right) as a function of time for evaporating water droplets	179
Fig. 3.13.	The enthalpy over time in nonreacting channel flow with hot particles injected. The net input enthalpy includes the initial domain enthalpy and the enthalpy of all of the injected particles over time. Squares show the enthalpy associated with particles in the domain. Circles show the enthalpy of the particles that have left the domain. Diamonds show the enthalpy associated with fluid in the domain. Triangles show the excess enthalpy of fluid that has left the domain (difference between the outlet enthalpy and inlet that was accounted for in the net input category). The sum of the categories indicated by symbols is shown to agree with the net input enthalpy indicating overall conservation of enthalpy.	185
Fig. 3.14.	The mass deposition rate integrated over the outlet is shown as a function of time. The corresponding particle inlet mass flux is 0.1.	186
Fig. 3.15.	The momentum associated with the particle phase, the fluid phase, and the combined momentum shown. The right-hand panel shows the initial period in greater detail.	187
Fig. 3.16.	Condensed-phase conduction is approximated based on the difference between the film and mean droplet temperatures and on an estimated heat transfer coefficient that describes a boundary layer thickness. Over this boundary layer thickness, the temperature difference $T_f - T_d$ is presumed to act.	190
Fig. 3.17.	Terminal velocities for particles as a function of diameter and particle Reynolds numbers determined from (3.711)	190
Fig. 3.18.	Water droplet evaporation and condensation with initial temperatures set to the wet bulb temperature. Left plot exhibits the linear d^2 -law behavior while the right hand plot shows the droplet temperatures as constant (no heating).	191
Fig. 3.19.	Aluminum particle evaporation with and without combustion with initial temperatures set to the wet bulb temperature showing the linear d^2 -law behavior.	191
Fig. 3.20.	The general configuration for verification of two-way coupling. The domain should be of sufficient length that the particles equilibrate with the gas-phase flow.	192
Fig. 3.21.	Particle size (diameter) distribution for Lagrangian particle spray with and without diameter cutoffs set at 0.3 and 0.65	194
Fig. 3.22.	Lagrangian particle spray section of a Fuego input deck showing use of diameter cutoffs	194
Fig. 3.23.	Alumina absorption coefficient for standard models Brewster and Kanopka along with a user-specified model.....	195
Fig. 3.24.	For a Lagrangian particle spray, the number of particles contained within a parcel for three representative particle diameters using constant number, constant mass, and user defined number of particles per parcel. Circles represent parcels with the points inside representing the number of particles contained in the parcel. ...	197
Fig. 3.25.	Examples of particle insertion types that can be used with particle insert from file mechanism	198

Fig. 3.26.	Example of particle spread from a conical shaped particle spray nozzle at early times. This nonstandard spray form was generated through the particle creation from file data mechanism. Here particle temperatures are set to be a function of their position with the hottest particles leaving the nozzle near the circular base of the cone.	199
Fig. 3.27.	Same simulation as Figure 3.26 but at late time	199
Fig. 3.28.	Control Volume is Centered about Finite-Element Node	200
Fig. 3.29.	Cell-Peclet number blending function.	221
Fig. 3.30.	Linear profile skew upwind scheme: a) all nodes on the intersected element face are upwind of the subface, b) omit nodes on intersected element face that are downwind of the subface.	222
Fig. 3.31.	Boundary mass flux integration locations.	226
Fig. 3.32.	Integration locations for a wall boundary.	251
Fig. 3.33.	Quadrilateral element topology and numbering	260
Fig. 3.34.	Triangular element topology and numbering	262
Fig. 3.35.	Triangular natural coordinate system, shaded area corresponds to opposite node. .	263
Fig. 3.36.	Hexahedral element topology and numbering.....	264
Fig. 3.37.	Hexahedron subcontrol points numbering	267
Fig. 3.38.	Tetrahedral element topology and numbering.....	269
Fig. 3.39.	Tetrahedron subcontrol points numbering	270
Fig. 3.40.	Wedge subcontrol points numbering	273
Fig. 3.41.	Pyramid subcontrol points numbering	275
Fig. 3.42.	Control volume faces in a single element. Contributions to the the control volume centered about node 3.....	278
Fig. 3.43.	Flux integration points (X) determine nodal (•) contributions to the coefficient stencil: a) mid-face rule of CVFEM, b) <i>edge</i> -operator, c) <i>centroid</i> -operator (one-point integration).....	279
Fig. 3.44.	Integration point locations must be shifted out towards the element-edge with increasing element aspect ratio.	281
Fig. 3.45.	Triangular element geometry in defined by edge length and two vertex angles. ...	283
Fig. 3.46.	Limits of Edge-Length Ratio for Positive Coefficients in 3D CVFEM.	284
Fig. 3.47.	Control volume definition on an h-adapted mesh with hanging nodes. (Four-patch of parent elements with refinement in bottom-right element.)	287
Fig. 3.48.	Control volume definition on an h-adapted mesh with transition control volumes about the hanging nodes. (Four-patch of parent elements with refinement in bottom-right element.).....	287
Fig. 3.49.	Control volume definition on a heterogeneous h-adapted mesh with no hanging nodes. (Four-patch of parent elements with refinement in bottom-right element and splitting in adjacent parent elements.)	288
Fig. 4.1.	Enumerated blocks	296
Fig. 4.2.	Control volumes for flux across a single face f	339

List of Tables

Table 1.1.	Fuego image gallery	10
Table 3.1.	Constant parameters for $k - \epsilon$ turbulence models.....	51
Table 3.2.	Constant parameters for $k - \omega$ turbulence models.	52
Table 3.3.	Constant parameters for LES turbulence models.	52
Table 3.4.	Constant parameters for miscellaneous turbulence models. Default values may be changed using the $k - \epsilon$ model parameters input.	52
Table 3.5.	Wall details for supported equation sets.....	65
Table 3.6.	Mass injection details for supported equation sets.	65
Table 3.7.	Soot model parameters (Tesner et al.(1971); Holen, et al.(1994))	112
Table 3.8.	Parameters used in Leckner's gas phase emittance model.	122
Table 3.9.	Species-specific parameters used in (3.492)	122
Table 3.10.	Coefficients C_{ij} for calculating the scale total emittance of CO ₂ from (3.493) and (3.494), (valid for $T > 400K$).	122
Table 3.11.	Coefficients C_{ij} for calculating the scale total emittance of H ₂ O from (3.493) and (3.494), (valid for $T > 400K$).	122
Table 3.12.	Input parameters related to the particle evolution provided through the input file	166
Table 3.13.	Variables passed from the gas-phase Eulerian solver required to evolve the particles	166
Table 3.14.	Input parameters related to the particle trajectory verification	175
Table 3.15.	Input parameters related to the particle dispersion verification	177
Table 3.16.	Input parameters related to the particle heating and cooling verification	178
Table 3.17.	Input parameters related to the verification of droplet evaporation.	179
Table 3.18.	Particle initial conditions for verification of droplet evaporation.	180
Table 3.19.	Input parameters related to the verification of energy conservation without evaporation	185
Table 3.20.	Input parameters related to the verification of momentum conservation	187
Table 3.21.	Input parameters related to the verification of energy conservation with evaporation.	189
Table 3.22.	Nodal shape functions and derivatives for quadrilateral elements	260
Table 3.23.	Element variable values and differentials at control-volume faces for quadrilateral elements. Face-to-edge number mapping.....	261
Table 3.24.	Element variable values and differentials at sub-control volume centers for quadrilateral elements.	261
Table 3.25.	Nodal shape functions and derivatives for triangular elements.	262
Table 3.26.	Element variable values and differentials at control-volume faces for triangular elements	263
Table 3.27.	Element variable values and differentials at sub-control volume centers for triangular elements.....	263
Table 3.28.	Nodal shape functions and derivatives for hexahedral elements. Range is (-1,1).	264
Table 3.29.	Element variable values and differentials at control-volume faces for hexahedral elements. Face-to-edge number mapping.	266
Table 3.30.	Element variable values and differentials at sub-control volume centers for hexahedral elements.....	266

Table 3.31.	Subcontrol face definitions for exact surface area calculation on hexahedral elements.	267
Table 3.32.	Subcontrol volume definitions for exact volume calculation on hexahedral elements.	268
Table 3.33.	Nodal shape functions and derivatives for tetrahedral elements. Range is (0,1). .	268
Table 3.34.	Element variable values and differentials at control-volume faces for tetrahedral elements. Face-to-edge number mapping.	269
Table 3.35.	Element variable values and differentials at sub-control volume centers for tetrahedral elements.	270
Table 3.36.	Subcontrol face definitions for exact surface area calculation on tetrahedral elements.	271
Table 3.37.	Subcontrol volume definitions for exact volume calculation on tetrahedral elements.	271
Table 3.38.	Nodal shape functions and derivatives for wedge elements. Range is (0,1) and (-1,1).	271
Table 3.39.	Element variable values and differentials at control-volume faces for wedge elements. Face-to-edge number mapping.	272
Table 3.40.	Element variable values and differentials at sub-control volume centers for wedge elements.	272
Table 3.41.	Subcontrol face definitions for exact surface area calculation on wedge elements.	273
Table 3.42.	Subcontrol volume definitions for exact volume calculation on wedge elements.	273
Table 3.43.	Nodal shape functions and derivatives for pyramid elements. Range is (-1,1) and (0,1).	274
Table 3.44.	Element variable values and differentials at control-volume faces for pyramid elements. Face-to-edge number mapping.	274
Table 3.45.	Element variable values and differentials at sub-control volume centers for pyramid elements.	275
Table 3.46.	Subcontrol face definitions for exact surface area calculation on pyramid elements.	276
Table 3.47.	Subcontrol volume definitions for exact volume calculation on pyramid elements.	276
Table 4.1.	Spectral file format	363
Table 4.2.	Spectral file example	364

1 Fuego Overview

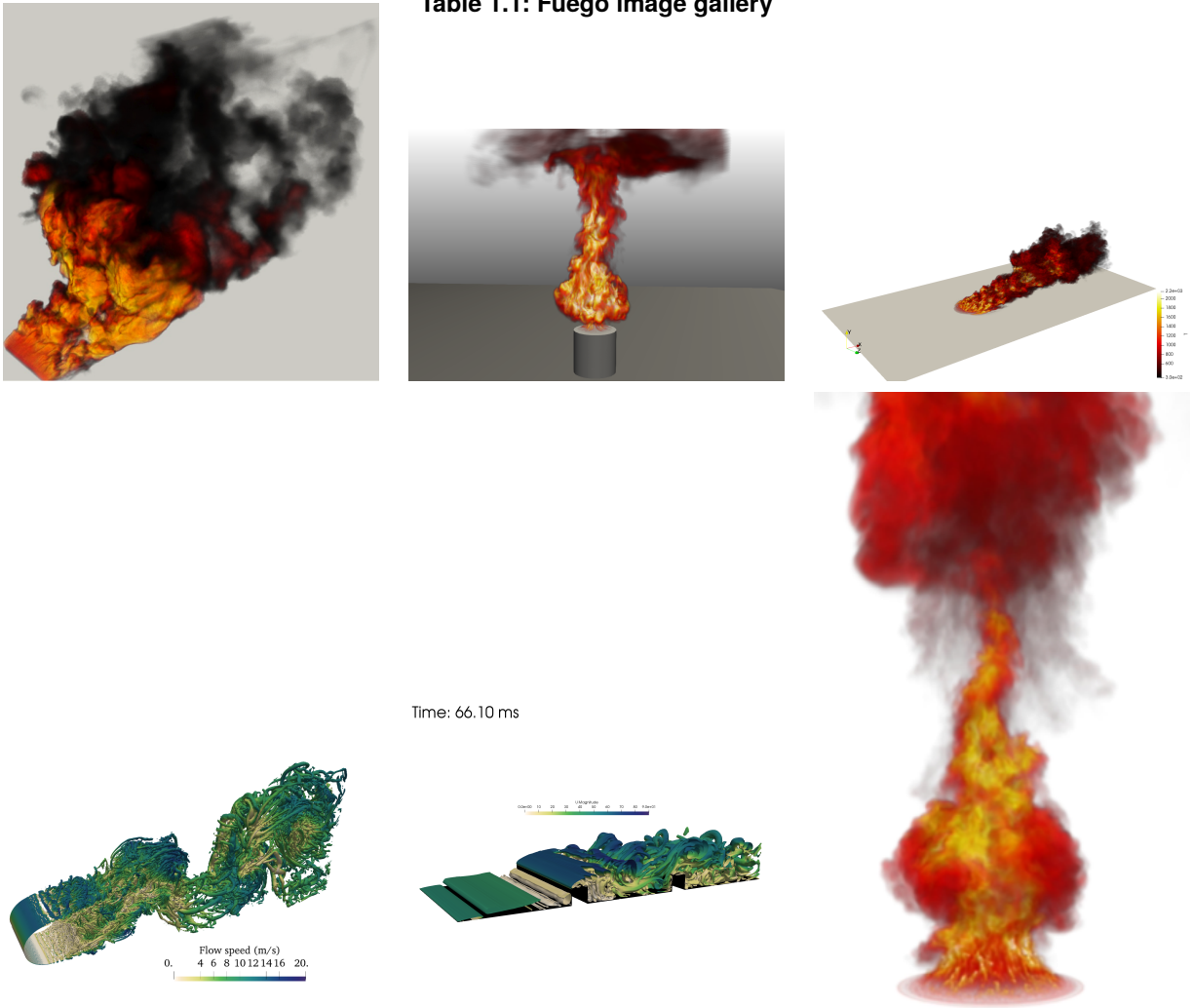


Sierra/Fuego is a low-Mach CFD solver primarily used for simulating:

- Fire environments (internal and external)
 - Eddy-dissipation-concept and flamelet combustion models
 - Coupling with Sierra/PMR for participating media radiation
 - Composite decomposition models
- Buoyancy driven flows
- Conjugate heat transfer (coupled with Sierra/Aria)
- Turbulent flow (including RANS, LES, and DES models)
- Multiphase liquid/gas flows using volume-of-fluid (VOF)
- Particle transport and coupling with fluid flow

Fire simulations are the primary Fuego application area, and are used to support weapon system qualification and safety assessments. Fuego is also used for a variety of non-fire applications including conjugate heat transfer, acoustic coupling with Sierra/SM and Sierra/SD, multi-phase flow problems, and general turbulent flow simulations.

Table 1.1: Fuego image gallery



Documents containing results obtained through use of this software should cite

```
@TECHREPORT{
  SAND2025-116230,
  author = "Sierra Thermal Fluids Team",
  title = "SIERRA Multimechanics Module: Fuego User Manual - Version 5.26",
  institution = "Sandia National Laboratories",
  year = "2025",
  type = "{SAND Report}",
  number = "2025-116230",
  address = "Albuquerque, NM and Livermore, CA"
}
```

2 Releases & Deprecations

This section provides a high-level overview of changes associated with each release. This overview includes new features, bug fixes, deprecations, etc.

2.1 Version 5.28 (pre-release)



Note: These are the running release notes for the daily version of Fuego, accumulated since the last release. Version 5.28 is set for release in Winter 2026.

Bug Fixes

New Features

Change in Behaviors

2.2 Version 5.26 (latest - 2025/09/11)

Bug Fixes

- Fixed issue with multi-block time stepping where if 1) a fixed time stepping block is followed by an adaptive time stepping block and 2) projection time step scaling is used, then the initial continuity preconditioner would be stale since we only were re-scaling the linear system for adaptive time stepping. Now we always rescale the linear system when projection time step scaling is used ensuring the preconditioner remains valid.
- Fixed issue with multi-block time stepping and steering files where upon restart, if the restart point is within a new time stepping block (e.g., fixed stepping transitioned to automatic), then the time step would be read as the CFL limit. Now we always override the steering file upon restart with appropriate values from the current time stepping block.

New Features

- Added the ability to use plugin utilities to add custom user-defined code
- Added the ability to define subroutines or other evaluators for temperature instead of using the default Newton solve
- Added the ability to define injected progress variable quantities on mass inject BCs
- Converted all mass inject BCs to be device-compatible so they can run on GPUs
- Added the ability to use a MPMD copy transfer. Activated via “USE MPMD RADIATION WITH COPY TRANSFER”, the copy transfer is a beta feature and

requires that Fuego / PMR use the same mesh. It avoids excessive memory usage that can occur when using the general interpolative transfer on high aspect ratio meshes decomposed across relatively few MPI ranks (e.g., GPU applications).

- Added the ability to use MueLu preconditioners in PMR solves

Change in Behaviors

2.3 Version 5.24 (2025/03/19)

Bug Fixes

- An issue using assemblies with large problems ($O(100M)$ elements) has been fixed
- An issue with the Fuego-PMR transfer exceeding the MPI 2GB limit has been fixed in STK
- Fixed a bug with tabular property caching on GPU runs

New Features

- The dynamic KSGS LES model has been ported to GPU-compatible algorithms
- Added new developer debugging tools (extra profiling and checking for NaN/Inf values in GPU algorithms)
- Integrate against new Trilinos version with Kokkos 4.5, leading to as much as a 2.8x speedup in MueLu solve times.

Change in Behaviors

- The “postprocess yplus” command on wall boundaries is no longer needed, y^+ is always postprocessed

2.4 Version 5.22 (2024/10/08)

Bug Fixes:

- Fixed inconsistent heat loss variable computation in some circumstances, for instance at walls with intermediate values of mixture fraction.

New Features

- Wall-modeled LES (WMLES) exchange-based approach for momentum and energy turbulent boundary conditions are fully supported. Exchange distance remains user specified.
- New “USE NEUMANN CONDITION FOR KSGS” line command that should be activated when using the WMLES k-sgs model to avoid conflation of “subgrid-scale kinetic energy” with “turbulent kinetic energy”.

Change in Behaviors

- Consistent WMLES usage of wall friction velocity when the “DETERMINE UTAU VIA NONLINEAR LAW OF THE WALL ITERATION” line command is activated.

2.5 Version 5.20 (2024/06/17)

New Features

- Updated the particle post-processors to work properly on the particle block for all types except `integral` and `integrated_flux`. Using `all_blocks` will result in an error since it mixes particle and non-particle blocks.
- New `high_aspect_continuity` preset solver.

Change in Behaviors

- By default, the continuity solve will now use the `high_aspect_continuity` preset solver.
- By default, the preset linear solvers will use the automatic convergence tolerance, see [Tolerance Settings](#).

2.6 Version 5.18 (2024/02/23)

New Features:

- Lagrangian particle thermophoresis: particle momentum source terms can now optionally include thermophoretic force contributions for Lagrangian particles

Bug Fixes:

- L2 norm postprocessor fixed to allow intermediate variables in the string function
- Reported mass flow rate in log file when using multiple open boundary conditions was incorrect
- Fixed error in non-unity Lewis laminar species transport model
- Fixed stress tensor trace scaling in 2D for various turbulence models
- Fixed bug in flux post-processor reduction for parallel-shared nodes

Upcoming Deprecations (will warn in 5.18)

- Using `Nalu` for participating media radiation is deprecated. Use `pmr` instead.
- Using `encore` post-processors is deprecated - use the `Begin Postprocess` blocks in the Fuego Region.
- Zero and Second order projection methods are deprecated.

- The Number of logged clipping events command no longer has any effect and will be removed in a future release.

A timeline of upcoming deprecations is provided in the figure below

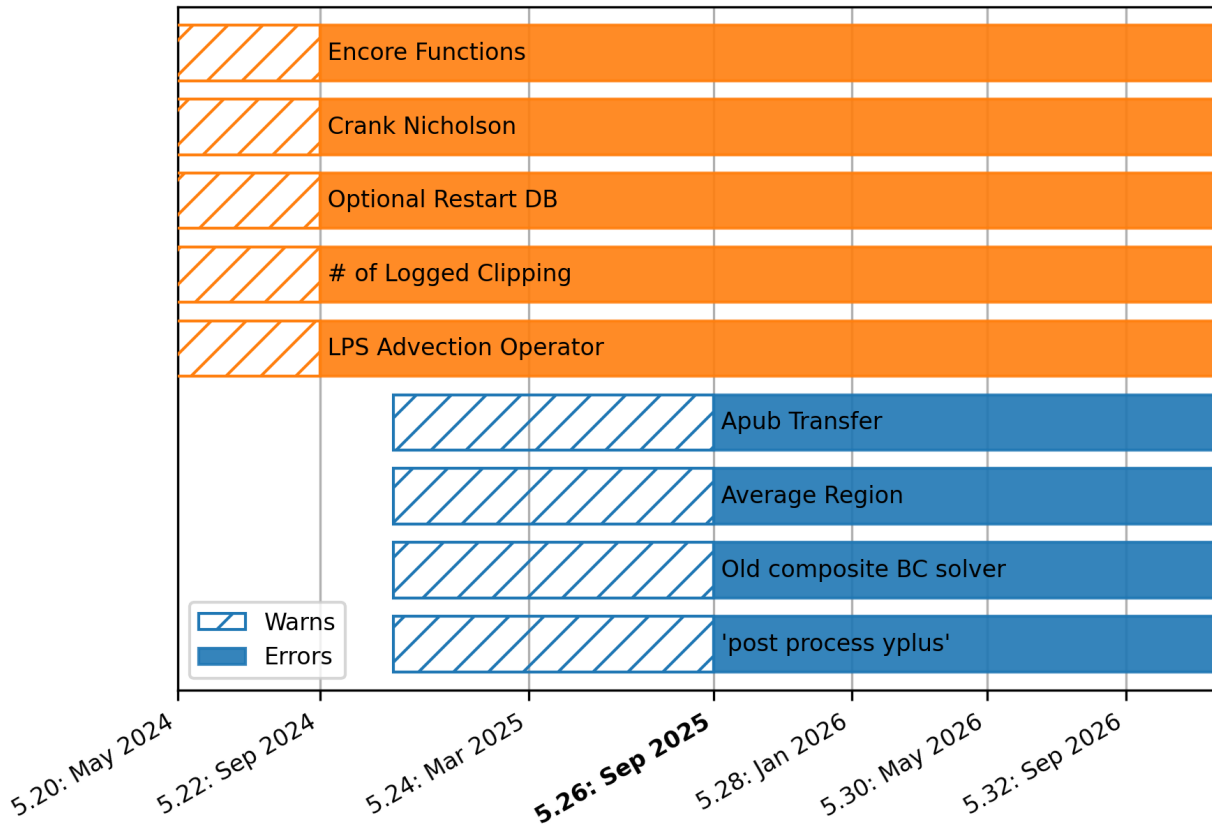


Fig. 2.1: Fuego Deprecation Timeline

3 Theory

3.1 Introduction

The SIERRA Low Mach Module: Fuego, henceforth referred to as Fuego, is the key element of the ASC fire environment simulation project. The fire environment simulation project is directed at characterizing both open large-scale pool fires and building enclosure fires. Fuego represents the turbulent, buoyantly-driven incompressible flow, heat transfer, mass transfer, combustion, soot, and absorption coefficient model portion of the simulation software. Using MPMD coupling, Scefir and Nalu handle the participating-media thermal radiation mechanics. This project is an integral part of the SIERRA multi-mechanics software development project. Fuego depends heavily upon the core architecture developments provided by SIERRA for massively parallel computing, solution adaptivity, and mechanics coupling on unstructured grids.

Abnormal Thermal Environments

Fuego is part of a suite of numerical simulation tools used to address abnormal thermal environments for nuclear weapon systems [1]. From manufacture to disassembly, a weapon will see three types of environments: normal, hostile, and abnormal. Abnormal environments result from natural phenomena, such as fires, floods, tornadoes, earthquakes, lightning strikes, meteor strikes, etc., and human phenomena, generally classified as “accidents”. In general, these phenomena can present thermal, mechanical, and electrical hazards to a weapon system. Nuclear weapon systems must respond to these abnormal environments in a deterministically safe manner.

Fire phenomena in the context of the abnormal thermal environment weapons response issue is part of a three stage process leading from an accident to the system response. For certain scenarios, these stages are uncoupled and may be sequential in time; in others, the stages are tightly coupled and concurrent in time.

The first stage is the initial accident or environmental scenario that is defined typically through probabilistic studies such as historic data involving accident frequencies of a given type, ignition probabilities, etc. These are used to define scenarios for deterministic simulation tools that determine the state of integrity of the weapon system and the distribution of fuel. The weapon integrity is determined by the mechanical, transient-dynamic environment it sees during an accident. For accident scenario description, Fuego is intended to handle the distribution of liquid fuels, although initial implementation will be somewhat limited due to the very broad possibilities (e.g., fuel pools, spills, sprays, porous flows) and complexity involved in two-phase flow.

The second stage is the actual buoyant, turbulent, reacting, flow that is the source of the thermal hazard for the weapon system. Fuego and MPMD-coupled Sierra/PMR are the primary tools that describe the fire phenomenology that links an accident description to thermal radiation and convection on a weapon system. Fire involves a very complex, coupled set of physical phenomena over a very broad range of time and length scales. The key features are the turbulent, buoyant flows involving combustion of the fuel and air, and the formation of soot which results in participating media radiation (Sierra/PMR), and a range of convection heat transfer conditions from free to forced convection (Fuego).

The third stage is the weapon thermal response. As with the fire itself, the response of the warhead to a fire is described by very complex, coupled set of physical phenomena. Simulation will require the coupling of several, separate effects codes for a complete description. Heat from the fire is conducted into the weapon and transmitted by surface-surface radiation. Materials such as foams decompose and result in pressurization. Conduction across engineered joints is pressure dependent as is the decomposition process. Materials such as aluminum can potentially melt and relocate. Energetic materials can decompose and react. Within this environment the engineered fail-safes in the weapon electrical system must operate with high reliability to ensure nuclear safety.

Because of the number of physical phenomena involved from the accident scenario to the weapon response for abnormal thermal environments, and the very disparate time and length scales over which these phenomena occur, it is necessary to have high-performance, massively-parallel,

computers to even consider addressing a problem of this scale and complexity. Further, the key to integrating this suite of tools is flexibility of coupling and a common database architecture. Thus it is intended that all the simulation requirements identified above will ride on a common software architecture (SIERRA) with broad coupling flexibilities.

The principal value of the suite of numerical simulation tools is not the description of the accident to response process, but the ability to evaluate prevention and mitigation design strategies. Preventative strategies are primarily applied via administrative controls. Examples include design and maintenance to minimize fuel levels, separation of fuels from air and ignition sources, and/or weapons separate from the combination. Mitigation strategies include suppression (either manually through fire-fighters or by automated fire suppression equipment), design of thermally activated fail-safes, and containment design. In general, multiple barriers exist between fire and health consequences to the general public for nuclear weapons.

Document Organization

This document contains theory and numerical details for the Fuego code. A discussion of the physical models and governing transport equations (math models) is given in *Math Models*. A discussion of the numerical methods that we use to solve the governing transport equations is given in *Numerics*.

The Einstein notation of repeated indices is used extensively throughout this document. The only exception is for equations involving chemical species where an explicit summation operator is used to imply summation over all chemical species.

3.2 Math Models

Fire simulation requires the solution of variable property, high Grashof number, turbulent, low Mach number flow including the effects of species and soot transport, radiation, and buoyancy.

Conservation laws include mass of the mixture, momentum, mass of the individual species, and energy. Length scales vary from molecular to convection dominated. For purposes of discussion, length scales are also categorized by the method of resolution.

The transport equations used to describe fire physics are based on two sets of approximations to the fundamental equations of fluid dynamics. Fast acoustic time scales are removed from the equations using low Mach number asymptotics, described in *Low Mach Number Equations*.

Turbulent transport at high Grashof numbers is modeled using a Reynolds averaging approach, described in *RANS Temporal Filtering*.

In what follows, we note that unless specifically stated otherwise all units in the equations and submodel expressions are cgs. For a more extensive treatment of units and unit conversions in Fuego, please see the “Units and Unit Conversions” section in the User’s Manual. The numerical methods we use to solve the transport equations are of the finite volume class. Therefore, we generally write the transport equations in the integral form.

Low Mach Number Equations

The low Mach number equations are a subset of the full compressible Navier-Stokes (and continuity and energy) equations, admitting large variations in gas density while remaining acoustically incompressible. The low Mach number equations are preferred over the full compressible equations for our problems of interest. We avoid resolving fast-moving acoustic signals which have no bearing on the transport processes. Derivations of the low Mach number equations are found in Rehm and Baum [2], Paolucci [3], Majda and Sethian [4], and Merkle and Choi [5]. The equations are derived from the compressible equations using a perturbation expansion in terms of the lower limit of the Mach number squared; hence the name. The asymptotic expansion leads to a splitting of pressure into a spatially constant thermodynamic pressure and a locally varying dynamic pressure. The dynamic pressure is decoupled from the thermodynamic state and cannot propagate acoustic waves. The thermodynamic pressure is used in the equation of state and to determine thermophysical properties. The thermodynamic pressure can vary in time and can be calculated using a global energy balance.

Asymptotic Expansion

The asymptotic expansion for the low Mach number equations begins with the full compressible equations in Cartesian coordinates. The equations are the minimum set required to propagate acoustic waves. The equations are written in divergence form using Einstein notation (summation over repeated indices):

$$\frac{\partial \rho}{\partial t} + \frac{\partial \rho u_j}{\partial x_j} = 0, \quad (3.1)$$

$$\frac{\partial \rho u_i}{\partial t} + \frac{\partial \rho u_j u_i}{\partial x_j} + \frac{\partial P}{\partial x_i} = \frac{\partial \tau_{ij}}{\partial x_j} + \rho g_i, \quad (3.2)$$

$$\frac{\partial \rho E}{\partial t} + \frac{\partial \rho u_j H}{\partial x_j} = -\frac{\partial q_j}{\partial x_j} + \frac{\partial u_i \tau_{ij}}{\partial x_j} + \rho u_i g_i. \quad (3.3)$$

The primitive variables are the velocity components, u_i , the pressure, P , and the temperature T . The viscous shear stress tensor is τ_{ij} , the heat conduction is q_i , the total enthalpy is H , the total internal energy is E , the density is ρ , and the gravity vector is g_i . The total internal energy and total enthalpy contain the kinetic energy contributions. The equations are closed using the following models and definitions:

$$P = \rho \frac{R}{W} T, \quad (3.4)$$

$$E = H - P/\rho, \quad (3.5)$$

$$H = h + \frac{1}{2} u_k u_k, \quad (3.6)$$

$$\tau_{ij} = \mu \left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i} \right) - \frac{2}{3} \mu \frac{\partial u_k}{\partial x_k} \delta_{ij}, \quad (3.7)$$

$$q_i = -k \frac{\partial T}{\partial x_i}. \quad (3.8)$$

The mean molecular weight of the gas is W , the molecular viscosity is μ , and the thermal conductivity is k . A Newtonian fluid is assumed along with the Stokes hypothesis for the stress tensor.

The equations are scaled so that the variables are all of order one. The velocities, lengths, and times are nondimensionalized by a characteristic velocity, U_∞ , and a length scale, L . The pressure, density, and temperature are nondimensionalized by P_∞ , ρ_∞ , and T_∞ . The enthalpy and energy are nondimensionalized by $C_{p,\infty}T_\infty$. Dimensionless variables are noted by overbars. The dimensionless equations are:

$$\frac{\partial \bar{\rho}}{\partial \bar{t}} + \frac{\partial \bar{\rho} \bar{u}_j}{\partial \bar{x}_j} = 0, \quad (3.9)$$

$$\begin{aligned} \frac{\partial \bar{\rho} \bar{u}_i}{\partial \bar{t}} + \frac{\partial \bar{\rho} \bar{u}_j \bar{u}_i}{\partial \bar{x}_j} + \frac{1}{\gamma \text{Ma}^2} \frac{\partial \bar{P}}{\partial \bar{x}_i} &= \frac{1}{\text{Re}} \frac{\partial \bar{\tau}_{ij}}{\partial \bar{x}_j} + \frac{1}{\text{Fr}_i} \bar{\rho} \\ \frac{\partial \bar{\rho} \bar{h}}{\partial \bar{t}} + \frac{\partial \bar{\rho} \bar{u}_j \bar{h}}{\partial \bar{x}_j} &= -\frac{1}{\text{Pr}} \frac{1}{\text{Re}} \frac{\partial \bar{q}_j}{\partial \bar{x}_j} + \frac{\gamma - 1}{\gamma} \frac{\partial \bar{P}}{\partial \bar{t}} \\ &+ \frac{\gamma - 1}{\gamma} \frac{\text{Ma}^2}{\text{Re}} \frac{\partial \bar{u}_i \bar{\tau}_{ij}}{\partial \bar{x}_j} + \bar{\rho} \bar{u}_i \frac{\gamma - 1}{\gamma} \frac{\text{Ma}^2}{\text{Fr}_i} \\ &- \frac{\gamma - 1}{2} \text{Ma}^2 \left(\frac{\partial \bar{\rho} \bar{u}_k \bar{u}_k}{\partial \bar{t}} + \frac{\partial \bar{\rho} \bar{u}_j \bar{u}_k \bar{u}_k}{\partial \bar{x}_j} \right). \end{aligned} \quad (3.10)$$

The groupings of characteristic scaling terms are:

$$\text{Re} = \frac{\rho_\infty U_\infty L}{\mu_\infty}, \quad \text{Reynolds number}, \quad (3.11)$$

$$\text{Pr} = \frac{C_{p,\infty} \mu_\infty}{k_\infty}, \quad \text{Prandtl number}, \quad (3.12)$$

$$\text{Fr}_i = \frac{u_\infty^2}{g_i L}, \quad \text{Froude number}, \quad g_i \neq 0, \quad (3.13)$$

$$\text{Ma} = \sqrt{\frac{u_\infty^2}{\gamma R T_\infty / W}}, \quad \text{Mach number}, \quad (3.14)$$

where γ is the ratio of specific heats.

For small Mach numbers, $\text{Ma} \ll 1$, the kinetic energy, viscous work, and gravity work terms can be neglected in the energy equation since those terms are scaled by the square of the Mach number. The inverse of Mach number squared remains in the momentum equations, suggesting singular behavior. In order to explore the singularity, the pressure, velocity and temperature are expanded as asymptotic series in terms of the parameter ϵ :

$$\bar{P} = \bar{P}_0 + \bar{P}_1 \epsilon + \bar{P}_2 \epsilon^2 \dots \quad (3.15)$$

$$\bar{u}_i = \bar{u}_{i,0} + \bar{u}_{i,1}\epsilon + \bar{u}_{i,2}\epsilon^2 \dots \quad (3.16)$$

$$\bar{T} = \bar{T}_0 + \bar{T}_1\epsilon + \bar{T}_2\epsilon^2 \dots \quad (3.17)$$

The zeroth-order terms are collected together in each of the equations. The form of the continuity equation stays the same. The gradient of the pressure in the zeroth-order momentum equations can become singular since it is divided by the characteristic Mach number squared. In order for the zeroth-order momentum equations to remain well-behaved, the spatial variation of the \bar{P}_0 term must be zero. If the magnitude of the expansion parameter is selected to be proportional to the square of the characteristic Mach number, $\epsilon = \gamma \text{Ma}^2$, then the \bar{P}_1 term can be included in the zeroth-order momentum equation.

$$\frac{1}{\gamma \text{Ma}^2} \frac{\partial \bar{P}}{\partial x_i} = \frac{\partial}{\partial x_i} \left(\frac{1}{\gamma \text{Ma}^2} \bar{P}_0 + \frac{\epsilon}{\gamma \text{Ma}^2} \bar{P}_1 + \dots \right) = \frac{\partial}{\partial x_i} \left(\bar{P}_1 + \epsilon \bar{P}_2 + \dots \right) \quad (3.18)$$

The form of the energy equation remains the same, less the kinetic energy, viscous work and gravity work terms. The P_0 term remains in the energy equation as a time derivative. The low Mach number equations are the zeroth-order equations in the expansion including the P_1 term in the momentum equations. The expansion results in two different types of pressure and they are considered to be split into a thermodynamic component and a dynamic component. The thermodynamic pressure is constant in space, but can change in time. The thermodynamic pressure is used in the equation of state. The dynamic pressure only arises as a gradient term in the momentum equation and acts to enforce continuity. The un-split dimensional pressure is

$$P = P_{th} + \gamma \text{Ma}^2 P_1, \quad (3.19)$$

where the dynamic pressure, $p = P - P_{th}$, is related to a pressure coefficient

$$\bar{P}_1 = \frac{P - P_{th}}{\rho_\infty u_\infty^2} P_{th}. \quad (3.20)$$

The resulting unscaled low Mach number equations are:

$$\frac{\partial \rho}{\partial t} + \frac{\partial \rho u_j}{\partial x_j} = 0, \quad (3.21)$$

$$\frac{\partial \rho u_i}{\partial t} + \frac{\partial \rho u_j u_i}{\partial x_j} + \frac{\partial p}{\partial x_i} = \frac{\partial \tau_{ij}}{\partial x_j} + (\rho - \rho_\circ) g_i, \quad (3.22)$$

$$\frac{\partial \rho h}{\partial t} + \frac{\partial \rho u_j h}{\partial x_j} = -\frac{\partial q_j}{\partial x_j} + \frac{\partial P_{th}}{\partial t}, \quad (3.23)$$

where the ideal gas law becomes

$$P_{th} = \rho \frac{R}{W} T. \quad (3.24)$$

The hydrostatic pressure gradient has been subtracted from the momentum equation, assuming an ambient density of ρ_\circ . The stress tensor and heat conduction remain the same as in the original equations.

Variable Thermodynamic Pressure

For a low Mach number set of equations, the time derivative of pressure can only be nonzero in a closed volume with energy addition or subtraction. Relaxing the low Mach number limit allows a time and spatially varying pressure to appear in the energy equation (see [Conservation of Energy](#)).

Laminar Flow Equations

Laminar transport equations are not used for fire problems, but they are important for other classes of problems such as manufacturing. The low Mach number approximation is assumed (see [Low Mach Number Equations](#)).

Conservation of Mass

The mass conservation equation of a mixture of gases is given by

$$\int \frac{\partial \rho}{\partial t} dV + \int \rho u_j n_j dS = 0, \quad (3.25)$$

where u_j is the mass average velocity of the mixture [6].

Conservation of Momentum

The conservation of momentum equations are given by

$$\int \frac{\partial \rho u_i}{\partial t} dV + \int \rho u_i u_j n_j dS + \int P n_i dS = \int \tau_{ij} n_j dS + \int (\rho - \rho_o) g_i dV, \quad (3.26)$$

where the viscous stress tensor is

$$\tau_{ij} = \mu \left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i} \right) - \frac{2}{3} \mu \frac{\partial u_k}{\partial x_k} \delta_{ij}. \quad (3.27)$$

The pressure, P , in the momentum equations deserves a special note as this quantity can represent either the dynamic, i.e., the second term in the Mach number expansion in the case of the low Mach number assumption, or the static pressure in the case of formally compressibility. In either case, as shown above the hydrostatic pressure gradient has been removed which gives rise to the far-field density, ρ_o , in the buoyancy body force. Optionally, we allow for the following sets of buoyancy models:

- Boussinesq buoyancy approximation where the density difference is approximated as

$$(\rho - \rho_o) \approx -\frac{\rho_o}{T_o} (T - T_o), \quad (3.28)$$

- Standard buoyant model in which case the pressure above does include the hydrostatic pressure and the buoyancy right-hand-side source term is,

$$\rho g_i, \quad (3.29)$$

- A Boussinesq approximation for a binary mixture in which case the right-hand-side contribution is:

$$\rho MW^{ref} \left(\frac{1}{MW_1} - \frac{1}{MW_2} \right) [Y_1 - Y^{ref}] g_i. \quad (3.30)$$

The user is referred to the Fuego user manual for exact line commands for each of these buoyancy options.

Note that zero pressure is almost always a convenient initial condition for a low Mach fluid flow. However, in cases without buoyancy, it can be anything, as the value only defines the additive constant for the pressure solve. However, one must ensure that the value matches for both initial and boundary condition specifications.

For buoyant flow, specifying zero pressure is convenient in tandem with the “differential” buoyancy option. This buoyancy term subtracts off the hydrostatic contribution such that the source term is written as

$$g (\rho - \rho_o) \quad (3.31)$$

One can see that using this term along with a zero pressure initial condition allows one to avoid specifying initial and boundary conditions as the hydrostatic pressure, i.e., as a function of height.

Conservation of Energy

The conservation of energy equation in terms of enthalpy (including a source term due to radiation absorption and emission) is

$$\begin{aligned} \int \frac{\partial \rho h}{\partial t} dV + \int \rho h u_j n_j dS = & - \int q_j n_j dS - \int \frac{\partial q_i^r}{\partial x_i} dV \\ & + \int \left(\frac{\partial P}{\partial t} + u_j \frac{\partial P}{\partial x_j} \right) dV + \int \tau_{ij} \frac{\partial u_i}{\partial x_j} dV, \end{aligned} \quad (3.32)$$

where the energy diffusion flux vector is given by

$$q_j = -\kappa \frac{\partial T}{\partial x_j} + \sum_{k=1}^K \rho h_k Y_k \hat{u}_{j,k}, \quad (3.33)$$

and $\hat{u}_{j,k}$ is the diffusion velocity of species k in the j direction.

This form of the energy equation is derived by starting with the energy equation and supplemental relationships of internal energy and total enthalpy provided in *Asymptotic Expansion*. The time term and convection term due to kinetic energy are expanded using the chain rule and simplified by enforcing the continuity equation. The remaining kinetic energy terms and gravitational force term are removed by dotting velocity with the momentum equation (to obtain the mechanical energy equation) and subtracting it from the energy equation. This procedure provides the full material derivative of pressure and the expanded viscous dissipation term.

The last two terms of (3.32) are only active when formal compressibility (in an acoustic sense) are important (see the Fuego user manual for the appropriate command lines to activate the low speed compressible and high speed compressible form in Fuego).

For a low Mach number flow, the time derivative of the pressure appearing above is substituted by the thermodynamic reference pressure, P_{th} , that can only be nonzero in a closed volume with energy addition or subtraction. However, the low Mach number approximation mandates that the thermodynamic pressure is always spatially uniform.

The enthalpy of the mixture, h , is a mass-average of the component enthalpies, h_k , given by

$$h = \sum_{k=1}^K h_k Y_k. \quad (3.34)$$

The energy diffusion flux vector includes a scaled gradient of temperature whereas the independent field to be solved in (3.32) is enthalpy. The form of the gradient of temperature is derived by first taking the gradient of (3.34) and using the chain rule,

$$\frac{\partial h}{\partial x_j} = \sum_{k=1}^K Y_k \frac{\partial h_k}{\partial x_j} + \sum_{k=1}^K h_k \frac{\partial Y_k}{\partial x_j}. \quad (3.35)$$

Given the thermodynamic definition of specific heat, the above equation is given by,

$$\frac{\partial h}{\partial x_j} = \sum_{k=1}^K Y_k C_{p,k} \frac{\partial T}{\partial x_j} + \sum_{k=1}^K h_k \frac{\partial Y_k}{\partial x_j} \quad (3.36)$$

$$\frac{\partial h}{\partial x_j} = C_p \frac{\partial T}{\partial x_j} + \sum_{k=1}^K h_k \frac{\partial Y_k}{\partial x_j}. \quad (3.37)$$

This equation is rearranged,

$$\frac{\partial T}{\partial x_j} = \frac{1}{C_p} \left(\frac{\partial h}{\partial x_j} - \sum_{k=1}^K h_k \frac{\partial Y_k}{\partial x_j} \right), \quad (3.38)$$

and substituted into the energy diffusion flux vector to obtain,

$$q_j = -\frac{\kappa}{C_p} \left(\frac{\partial h}{\partial x_j} - \sum_{k=1}^K h_k \frac{\partial Y_k}{\partial x_j} \right) + \sum_{k=1}^K \rho h_k Y_k \hat{u}_{j,k}. \quad (3.39)$$

Commonly, the last two terms in the above equation can be canceled when a simple diffusion model is assumed (see *Conservation of Species*, (3.45)) in the limit where the ratio of thermal and mass diffusion is equal (unity Lewis number, or equivalently speaking the Prandtl number equals the Schmidt number, i.e.,

$$Le^{unity} = \frac{Sc}{Pr} = \frac{\alpha}{D} = 1. \quad (3.40)$$

For completeness, the thermal diffusivity, Prandtl and Schmidt number are defined by,

$$\alpha = \frac{\kappa}{\rho C_p}, \quad (3.41)$$

$$Pr = \frac{C_p \mu}{\kappa} = \frac{\mu}{\rho \alpha}. \quad (3.42)$$

and

$$Sc = \frac{\mu}{\rho D_{ab}}. \quad (3.43)$$

Conservation of Species

The mass conservation equation for species k in a mixture of K gas phase species is

$$\int \frac{\partial \rho Y_k}{\partial t} dV + \int \rho Y_k u_j n_j dS = - \int \rho \hat{u}_{j,k} Y_k n_j dS + \int \dot{\omega}_k dV, \quad (3.44)$$

where $\dot{\omega}_k$ is the mass generation rate of species k per unit volume by homogeneous chemical reactions. We allow several approximations for the diffusion velocity, $\hat{u}_{j,k}$.

The simplest form is Fickian diffusion with the same value of mass diffusivity for all species,

$$\hat{u}_{i,k} = -D \frac{1}{Y_k} \frac{\partial Y_k}{\partial x_i}. \quad (3.45)$$

This form is used for the Reynolds-averaged form of the equations for turbulent flow.

A more accurate approximation uses a mixture-averaged diffusion coefficient, \bar{D}_k , for each species diffusion velocity,

$$\hat{u}_{i,k} = -\bar{D}_k \frac{1}{X_k} \frac{\partial X_k}{\partial x_i} = -\bar{D}_k \left(\frac{1}{Y_k} \frac{\partial Y_k}{\partial x_i} + \frac{1}{W} \frac{\partial W}{\partial x_i} \right). \quad (3.46)$$

However, the form above does not enforce the requirement that the sum over all species diffusional fluxes is equal to zero. To achieve this, we decompose the fluxes ($j_{i,k}$) into a modeled form and a correction to the modeled form in terms of the diffusional velocities

$$j_{i,k} = \rho Y_k \left(\hat{u}_{i,k}^m + \hat{u}_i^c \right) \quad (3.47)$$

where $\hat{u}_{i,k}^m$ is defined by (3.46). To define the correction velocity \hat{u}_i^c we sum over all species and apply the zero flux condition to get

$$\sum_{k=1}^K j_{i,k} = - \sum_{k=1}^K \frac{\rho W_k \bar{D}_k}{W} \frac{\partial X_k}{\partial x_i} + \rho \hat{u}_i^c \sum_{k=1}^K Y_k = 0 \quad (3.48)$$

which can be simplified to

$$\hat{u}_i^c = \sum_{k=1}^K \frac{\rho W_k \bar{D}_k}{W} \frac{\partial X_k}{\partial x_i} \quad (3.49)$$

Conservation of Momentum, Axisymmetric with Swirl

Axisymmetric flows, with or without swirl, are described by two-dimensional equations in cylindrical coordinates. All azimuthal derivatives are zero (i.e., $\partial/\partial\theta = 0$). The axial coordinate is x , the radial coordinate is r , and the azimuthal coordinate is θ . The radius is retained in the equations and the purpose will become more clear in the discussion of the discrete integral form. The axial velocity is u , the radial velocity is v , and the azimuthal velocity is w .

Axial-Momentum

$$\frac{\partial \rho u r}{\partial t} + \frac{\partial}{\partial x} (\rho u^2 r) + \frac{\partial}{\partial r} (\rho u v r) + r \frac{\partial p}{\partial x} = \frac{\partial}{\partial x} (r \tau_{xx}) + \frac{\partial}{\partial r} (r \tau_{xr}) + \rho r g_x \quad (3.50)$$

Radial-Momentum

$$\frac{\partial \rho v r}{\partial t} + \frac{\partial}{\partial x} (\rho u v r) + \frac{\partial}{\partial r} (\rho v^2 r) + r \frac{\partial p}{\partial r} - \rho w^2 = \frac{\partial}{\partial x} (r \tau_{rx}) + \frac{\partial}{\partial r} (r \tau_{rr}) - \tau_{\theta\theta} + \rho r g_r \quad (3.51)$$

Azimuthal-Momentum

$$\frac{\partial \rho w r}{\partial t} + \frac{\partial}{\partial x} (\rho u w r) + \frac{\partial}{\partial r} (\rho v w r) + \rho v w = \frac{\partial}{\partial x} (r \tau_{\theta x}) + \frac{1}{r} \frac{\partial}{\partial r} (r^2 \tau_{\theta r}) \quad (3.52)$$

The viscous stress terms for the cylindrical equations are

$$\tau_{xx} = \mu \left[2 \frac{\partial u}{\partial x} - \frac{2}{3} \left(\frac{\partial u}{\partial x} + \frac{\partial v}{\partial r} + \frac{v}{r} \right) \right] \quad (3.53)$$

$$\tau_{rx} = \mu \left[\frac{\partial v}{\partial x} + \frac{\partial u}{\partial r} \right] \quad (3.54)$$

$$\tau_{rr} = \mu \left[2 \frac{\partial v}{\partial r} - \frac{2}{3} \left(\frac{\partial u}{\partial x} + \frac{\partial v}{\partial r} + \frac{v}{r} \right) \right] \quad (3.55)$$

$$\tau_{\theta\theta} = \mu \left[2 \frac{v}{r} - \frac{2}{3} \left(\frac{\partial u}{\partial x} + \frac{\partial v}{\partial r} + \frac{v}{r} \right) \right] \quad (3.56)$$

$$\tau_{r\theta} = \mu r \frac{\partial}{\partial r} \left(\frac{w}{r} \right) \quad (3.57)$$

$$\tau_{x\theta} = \mu \frac{\partial w}{\partial x} \quad (3.58)$$

The azimuthal equation can be simplified by relating the swirl velocity to the angular velocity, $w = r\omega$. The momentum equation, written in terms of the angular velocity, is

$$\frac{\partial \rho \omega r}{\partial t} + \frac{\partial}{\partial x} (\rho u \omega r) + \frac{\partial}{\partial r} (\rho v \omega r) + 2\rho v \omega = \frac{\partial}{\partial x} \left(r \mu \frac{\partial \omega}{\partial x} \right) + \frac{\partial}{\partial r} \left(r \mu \frac{\partial \omega}{\partial r} \right) + 2\mu \frac{\partial \omega}{\partial r}. \quad (3.59)$$

The production term that is used in the turbulence model is

$$\Phi = 2 \left[\left(\frac{\partial u}{\partial x} \right)^2 + \left(\frac{\partial v}{\partial r} \right)^2 + \left(\frac{v}{r} \right)^2 \right] + \left(\frac{\partial u}{\partial r} + \frac{\partial v}{\partial x} \right)^2 - \frac{2}{3} \left(\frac{\partial u}{\partial x} + \frac{\partial v}{\partial r} + \frac{v}{r} \right)^2. \quad (3.60)$$

Laminar Flow Boundary Conditions

The laminar flow math models require boundary conditions for velocity, pressure, temperature and enthalpy variables, and mixture composition.

Inflow

There are three types of inflow boundary conditions. For velocity-specified inflow, Dirichlet conditions are applied to velocities in the momentum equations, temperature in the energy equation, and mass fractions in the species equations. The mass flow rate at the boundary is specified for the continuity equation. The pressure floats to a consistent value. Alternatively, a control volume balance is retained at the boundary nodes and the convection fluxes are specified.

For pressure-specified inflow, the outflow boundary condition is applied with the added condition that the flow must enter the domain normal to the mesh boundary. Transport equations are solved for the momentum, energy and species equations.

Outflow

The pressure is specified at integration points on the outflow boundary. The specified pressure is used in the surface integration procedure for approximation nodal gradients. The pressure gradients are used to construct an interpolation for the mass flow rate at the boundary. Transport equations are solved for the momentum, energy and species equations. Upwind extrapolation is used for the scalars if the flow is leaving the domain. The boundary values of velocity and specified far-field values of scalars are used if the flow is entering the domain.

Wall

It is assumed that there is no mass flow through the wall. The velocity is specified as a Dirichlet boundary condition in the momentum equations. The temperature is specified as a Dirichlet boundary condition in the energy if the wall is isothermal. We currently do not support heterogeneous chemical reactions at a surface, so there should be no boundary condition applied to the mass fractions.

Symmetry Plane

There is no mass flow rate through the symmetry plane and there is no transport of scalar variables. The normal stress (pressure and viscous) at the symmetry plane is applied in the momentum equations.

Radiation Transport Equation

For applications involving PMR, both the radiative heat flux and the divergence of the radiative heat flux are needed. The radiative heat flux vector provides the radiative flux to the boundary of the heat conduction region. The flux divergence provides one of the principal volumetric heat sources in the turbulent combustion region for fire applications.

Boltzmann Transport Equation

The spatial variation of the radiative intensity corresponding to a given direction and at a given wavelength within a radiatively participating material, $I(s)$, is governed by the Boltzmann transport equation. In general, the Boltzmann equation represents a balance between absorption, emission, out-scattering, and in-scattering of radiation at a point. For combustion applications, however, the steady form of the Boltzmann equation is appropriate since the transient term only becomes important on nanosecond time scales which is orders of magnitude shorter than the fastest chemical reaction [7].

Experimental data shows that the radiative properties for heavily sooting, fuel-rich hydrocarbon diffusion flames ($10^{-4}\%$ to $10^{-6}\%$ soot by volume) are dominated by the soot phase and to a lesser extent by the gas phase (Modest [8], pg. 425). Since soot emits and absorbs radiation in a relatively constant spectrum, it is common to ignore wavelength effects when modeling radiative transport in these environments. Additionally, scattering from soot particles commonly generated by hydrocarbon flames is several orders of magnitude smaller than the absorption effect and may be neglected [7]. With these assumptions in mind, the appropriate form of the Boltzmann radiative transport equation for heavily sooting hydrocarbon diffusion flames is

$$s_i \frac{\partial}{\partial x_i} I(s) + \mu_a I(s) = \frac{\mu_a \sigma T^4}{\pi}, \quad (3.61)$$

where μ_a is the absorption coefficient, $I(s)$ is the intensity along the direction s_i , and T is the temperature.

The flux divergence (on the right hand side of (3.32)) may be written as a difference between the radiative emission and mean incident radiation at a point,

$$\frac{\partial q_i^r}{\partial x_i} = \mu_a [4\sigma T^4 - G], \quad (3.62)$$

where G is the scalar flux. The quantity, $G/4\pi$, is often referred to as the mean incident intensity [9].

The scalar flux and radiative flux vector represent angular moments of the directional radiative intensity at a point [8],

$$G = \int_0^{2\pi} \int_0^\pi I(s) \sin \theta_{zn} d\theta_{zn} d\theta_{az}, \quad (3.63)$$

$$q_i^r = \int_0^{2\pi} \int_0^\pi I(s) s_i \sin \theta_{zn} d\theta_{zn} d\theta_{az}, \quad (3.64)$$

where θ_{zn} and θ_{az} are the zenith and azimuthal angles respectively as shown in *Ordinate Direction Definition*, with $\mathbf{s} = \sin \theta_{zn} \sin \theta_{az} \mathbf{i} + \cos \theta_{zn} \mathbf{j} + \sin \theta_{zn} \cos \theta_{az} \mathbf{k}$.

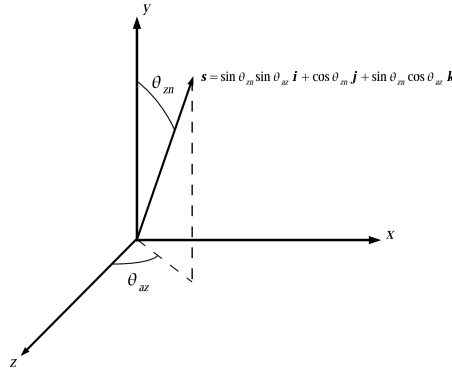


Fig. 3.1: Ordinate Direction Definition

Radiation Intensity Boundary Condition

The radiation intensity must be defined at all portions of the boundary along which $s_i n_i < 0$, where n_i is the outward directed unit normal vector at the surface. The intensity is applied as a Dirichlet condition which must be determined from the surface properties and temperature. The diffuse surface assumption provides reasonable accuracy for many engineering combustion applications. The intensity leaving a diffuse surface in all directions is given by

$$I(s) = \frac{1}{\pi} \left[\tau \sigma T_\infty^4 + \epsilon \sigma T_w^4 + (1 - \epsilon - \tau) q_j^{r,inc} n_j \right], \quad (3.65)$$

where ϵ is the total normal emissivity of the surface, τ is the transmissivity of the surface, T_w is the temperature of the boundary, T_∞ is the environmental temperature and $q_j^{r,inc}$ is the incident radiation, or irradiation for direction j . Recall that the relationship given by Kirchhoff's Law that relates emissivity, transmissivity and reflectivity, ρ , is

$$\rho + \tau + \epsilon = 1. \quad (3.66)$$

where it is implied that $\alpha = \epsilon$.

Turbulence Modeling Overview

Turbulent reacting flows involve a very large range of length and time scales, requiring massive computational resources to directly resolve all of the physical processes for even the most simple problem. To be able to solve complex problems of interest in a reasonable amount of time, modeling approximations must be made. A filtered form of the time-dependent Navier-Stokes, energy, and species mass conservation equations presented in *Laminar Flow Equations* are used, and closure models are applied to the new terms that arise due to the filtering operation. Temporal filtering is used in the Reynolds-Averaged Navier-Stokes (RANS) method, and spatial filtering is used in the Large Eddy Simulation (LES) method. The form of the models are dependent on the type of filtering performed, and will be discussed for both the RANS and LES approaches in the following sections.

The length scales between the smallest control volume dimension and the largest mesh dimension are defined as being “resolved”, and the transport equations are used to solve the physics in this range. The effects of the resolved turbulent scales may be modeled for RANS closures or they may be directly solved for LES closures. Turbulence length scales can extend down many orders of magnitude beyond the smallest finite volume dimension to the Kolmogorov scales, and these subgrid scales must be modeled in either closure approach.

The output of the closure models is expressed as a source term in the conservation equations for the mean flow and as effective properties in the radiative transport equation. Hence, the output of the closure models can be interpreted as being cell-averaged values for the control volume for the appropriate time scale. For the RANS formulation used here, the time scale is long relative to the turbulence time scales (i.e., long time average). For LES, the time scale is the local advection time. For the current suite of models, the momentum closure model is of the lumped-parameter type; that is, it assumes homogeneity of the subgrid turbulence. The remaining closures, species and energy, are of the zone-model type; that is, they assume heterogeneity of the species and energy subgrid. Two zones (one combusting, one not) are used in the current zone models.

For length scales above the length scale of the mesh, the physics is modified via boundary and initial conditions. Momentum boundary conditions include specified velocity (wind, and mass sources), or constant pressure (inflow/outflow). Species boundary conditions include a mass source for the fuel (pool model). Thermal boundary conditions include flux and temperature conditions. The following sections provide details of the math models for conservation laws and fire physics models used in SIERRA/Fuego.

RANS Temporal Filtering

In many typical engineering applications, only time averages of physical quantities are of interest. Often, details of the turbulent fluctuations are of little concern. RANS formulations address this need by solving a temporally-filtered form of the transport equations, directly yielding the time-averaged variables of interest. For this reason, RANS approaches represent a relatively low-cost solution method at the expense of additional modeling complexity.

An independent variable ϕ can be temporally filtered to obtain its mean $\bar{\phi}$ with the mathematical form (Tennekes and Lumley [10])

$$\overline{\phi(\mathbf{x})} = \lim_{\tau \rightarrow \infty} \frac{1}{\tau} \int_{t_o}^{t_o+\tau} \phi(\mathbf{x}, t) dt. \quad (3.67)$$

The original variable can be represented as the sum of its mean and fluctuating component, $\phi = \bar{\phi} + \phi'$, with the properties that $\overline{\bar{\phi}} = \bar{\phi}$ and $\overline{\phi'} = 0$. This is called the Reynolds decomposition of a variable.

In combustion problems, the overall exothermic process can result in large localized temperature increases and a correspondingly large density decrease in open systems where the molecular weight change from reactants to products is small. Allowing for turbulent fluctuations of density, the above temporal averaging procedure gives rise to additional terms involving time averages of products of density and other variable (e.g., velocity) fluctuations. An alternative approach to applying the Reynolds decomposition strictly to all independent variables is to consider a mass-weighted decomposition known as Favre averaging (Libby and Williams [11], p. 15; Kuo [12], p. 419). This simplifies all of the transport equations and eases modeling. A Favre-averaged variable $\tilde{\phi}$ is defined in terms of Reynolds averages as

$$\tilde{\phi} \equiv \frac{\overline{\rho\phi}}{\bar{\rho}}. \quad (3.68)$$

A variable can then be decomposed into its Favre-mean and fluctuating component as

$$\phi = \tilde{\phi} + \phi'', \quad (3.69)$$

where $\overline{\rho\phi''} = 0$. Note that $\overline{\phi''} \neq 0$. The relation between time averaged and Favre-averaged quantities is

$$\tilde{\phi} = \bar{\phi} \left(1 + \frac{\overline{\rho'\phi'}}{\bar{\rho}\bar{\phi}} \right). \quad (3.70)$$

Favre averaging is used for all turbulent transport equations solved in SIERRA/Fuego.

For the RANS formulation used here, the laminar conservation equations of *Laminar Flow Equations* are first temporally filtered, revealing additional terms that can be simplified by substituting the Favre decomposition, resulting in the Favre-filtered equations that will be presented in *Turbulent Flow Equations, Favre-Averaged*. This procedure results in new terms in

the equations that consist of time averages of products of fluctuating quantities, called Reynolds stresses. These moments must be modeled to close the system of equations.

The length of the time filter is typically much larger than the time scales of a turbulent flow, meaning that all time scales from the largest turbulence scale down to the minimum Kolmogorov scale are represented by these Reynolds stresses. In a strict sense, there can be no time dependence of a mean (time-averaged) quantity. However, if there are variations in mean quantities that occur on time intervals long compared to the averaging interval, then the transient terms for the mean quantities may be justified and required. For this reason, unsteady RANS simulations are possible with the present formulation. The available RANS turbulence closure models are discussed in *Turbulence Closure Models*.

LES Spatial Filtering

Unlike the RANS approach which models most or all of the turbulent fluctuations, LES directly solves for all resolved turbulent length scales and only models the smallest scales below the grid size. In this way, a majority of the problem-dependent, energy-containing turbulent structure is directly solved in a model-free fashion. The subgrid scales are closer to being isotropic than the resolved scales, and they generally act to dissipate turbulent kinetic energy cascaded down from the larger scales in momentum-driven turbulent flows. Modeling of these small scales is generally more straightforward than RANS approaches, and overall solutions are usually more tolerant to LES modeling errors because the subgrid scales comprise such a small portion of the overall turbulent structure. While LES is generally accepted to be much more accurate than RANS approaches for complex turbulent flows, it is also significantly more expensive than equivalent RANS simulations due to the finer grid resolution required. Additionally, since LES results in a full unsteady solution, the simulation must be run for a long time to gather any desired time-averaged statistics. The trade-off between accuracy and cost must be weighed before choosing one method over the other.

The separation of turbulent length scales required for LES is obtained by using a spatial filter rather than the RANS temporal filter. This filter has the mathematical form

$$\overline{\phi(\mathbf{x}, t)} \equiv \int_{-\infty}^{+\infty} \phi(\mathbf{x}', t) G(\mathbf{x}' - \mathbf{x}) d\mathbf{x}', \quad (3.71)$$

which is a convolution integral over physical space \mathbf{x} with the spatially-varying filter function G . The filter function has the normalization property $\int_{-\infty}^{+\infty} G(\mathbf{x}) d\mathbf{x} = 1$, and it has a characteristic length scale Δ so that it filters out turbulent length scales smaller than this size. In the present formulation, a simple “box filter” is used for the filter function,

$$G(\mathbf{x}' - \mathbf{x}) = 1/V(\mathbf{x}' - \mathbf{x}) \in \mathcal{V} \quad (3.72)$$

$$= 0 \quad \text{otherwise} \quad (3.73)$$

, where V is the volume of control volume \mathcal{V} whose central node is located at \mathbf{x} . This is essentially an unweighted average over the control volume. The length scale of this filter is

approximated by $\Delta = V^{\frac{1}{3}}$. This is typically called the grid filter, as it filters out scales smaller than the computational grid size.

Similar to the RANS temporal filter, a variable can be represented in terms of its filtered and subgrid fluctuating components as

$$\phi = \bar{\phi} + \phi'. \quad (3.74)$$

For most forms of the filter function $G(\mathbf{x})$, repeated applications of the grid filter to a variable do not yield the same result. In other words, $\bar{\bar{\phi}} \neq \bar{\phi}$ and therefore $\overline{\phi'} \neq 0$, unlike with the RANS temporal averages.

As with the RANS formulation, modeling is much simplified in the presence of large density variations if a Favre-filtered approach is used. A Favre-filtered variable $\tilde{\phi}$ is defined as

$$\tilde{\phi} \equiv \frac{\overline{\rho\phi}}{\bar{\rho}} \quad (3.75)$$

and a variable can be decomposed in terms of its Favre-filtered and subgrid fluctuating component as

$$\phi = \tilde{\phi} + \phi''. \quad (3.76)$$

Again, note that the useful identities for the Favre-filtered RANS variables do not apply, so that $\tilde{\tilde{\phi}} \neq \tilde{\phi}$ and $\overline{\phi''} \neq 0$. The Favre-filtered approach is used for all LES models in SIERRA/Fuego.

Turbulent Flow Equations, Favre-Averaged

The Favre-averaged turbulent transport equations are derived from the laminar equations of *Laminar Flow Equations* by passing the equations through either the RANS temporal filter of (3.67) or the LES spatial filter of (3.71). The mathematical form of the equations are essentially identical between the two filtering methods, so only a single set of equations will be presented. Care should be taken to interpret the filters as either temporal or spatial, depending on the closure models selected. While it is the Favre-averaged form of the equations that are solved, a comparison of the simple Reynolds-averaged and the Favre-averaged form is given at the end of this page for reference.

The approach most commonly used in turbulence modeling is called the Boussinesq eddy viscosity approximation, which relates the turbulent stress tensor to the filtered strain rate tensor through a modeled turbulent eddy viscosity. This general modeling approach has shown remarkable success for a broad range of problems (Wilcox [13]), and is the approach used in SIERRA/Fuego. A similar approach is used for scalar transport, where the scalar flux vector is related to scalar gradients through a modeled diffusion coefficient.

The following subsections describe the turbulent transport equations expressed in terms of a turbulent eddy viscosity or turbulent diffusion coefficient through the Boussinesq approximation. The treatment of these coefficients is dependent upon which of the many closure models are selected, and will be described in *Turbulence Closure Models*.

Conservation of Mass

The integral form of the Favre-filtered continuity equation used for turbulent transport is

$$\int \frac{\partial \bar{\rho}}{\partial t} dV + \int \bar{\rho} \tilde{u}_j n_j dS = 0. \quad (3.77)$$

This equation is in closed form, and no additional modeling is required.

Conservation of Momentum

The integral form of the Favre-filtered momentum equations used for turbulent transport are

$$\int \frac{\partial \bar{\rho} \tilde{u}_i}{\partial t} dV + \int \bar{\rho} \tilde{u}_i \tilde{u}_j n_j dS + \int \bar{p} n_i dS = \int \bar{\tau}_{ij} n_j dS + \int \tau_{u_i u_j} n_j dS + \int (\bar{\rho} - \rho_o) g_i dV, \quad (3.78)$$

where the turbulent stress $\tau_{u_i u_j}$ is defined as

$$\tau_{u_i u_j} \equiv -\bar{\rho}(\widetilde{u_i u_j} - \tilde{u}_i \tilde{u}_j). \quad (3.79)$$

RANS Modeling

For RANS simulations, $\tau_{u_i u_j}$ represents the Reynolds stress tensor and can be reduced to the form $\tau_{u_i u_j} = -\overline{\rho u_i'' u_j''}$ by substitution of the Favre decomposition $u_i \equiv \tilde{u}_i + u_i''$ of each variable and simplifying. The deviatoric (trace-free) part of the stress tensor is defined as

$$\tau_{u_i u_j}^D \equiv \tau_{u_i u_j} - \frac{1}{3} \tau_{u_k u_k} \delta_{ij} \quad (3.80)$$

$$\tau_{u_i u_j}^D = \tau_{u_i u_j} + \frac{2}{3} \bar{\rho} \tilde{k} \delta_{ij} \quad (3.81)$$

where the turbulent kinetic energy is defined as $\tilde{k} \equiv \frac{1}{2} \widetilde{u_k'' u_k''}$. The deviatoric part of the Reynolds stress tensor is modeled by the Boussinesq approximation which relates the Reynolds stresses to the filtered strain rate tensor through a modeled turbulent viscosity μ_t , resulting in

$$\tau_{u_i u_j}^D = \mu_t \left(\frac{\partial \tilde{u}_i}{\partial x_j} + \frac{\partial \tilde{u}_j}{\partial x_i} \right) - \frac{2}{3} \mu_t \frac{\partial \tilde{u}_k}{\partial x_k} \delta_{ij} \quad (3.82)$$

$$\tau_{u_i u_j}^D = 2\mu_t \left(\tilde{S}_{ij} - \frac{1}{3} \tilde{S}_{kk} \delta_{ij} \right), \quad (3.83)$$

where the filtered strain rate tensor is defined by

$$\tilde{S}_{ij} \equiv \frac{1}{2} \left(\frac{\partial \tilde{u}_i}{\partial x_j} + \frac{\partial \tilde{u}_j}{\partial x_i} \right). \quad (3.84)$$

Substituting this into (3.81) yields the modeled form of the full Reynolds stress tensor (Kuo [12], p. 445)

$$\tau_{u_i u_j} = 2\mu_t \left(\tilde{S}_{ij} - \frac{1}{3} \tilde{S}_{kk} \delta_{ij} \right) - \frac{2}{3} \bar{\rho} \tilde{k} \delta_{ij}. \quad (3.85)$$

The Favre-filtered momentum equations then become

$$\begin{aligned} \int \frac{\partial \bar{\rho} \tilde{u}_i}{\partial t} dV + \int \bar{\rho} \tilde{u}_i \tilde{u}_j n_j dS + \int \left(\bar{p} + \frac{2}{3} \bar{\rho} \tilde{k} \right) n_i dS = \\ \int 2(\mu + \mu_t) \left(\tilde{S}_{ij} - \frac{1}{3} \tilde{S}_{kk} \delta_{ij} \right) n_j dS + \int (\bar{\rho} - \rho_o) g_i dV, \end{aligned} \quad (3.87)$$

where RANS closure models for the turbulent viscosity μ_t are presented in *Turbulence Closure Models*.

LES Modeling

For LES, $\tau_{u_i u_j}$ in (3.78) represents the subgrid stress tensor. The deviatoric part of the subgrid stress tensor is defined as

$$\tau_{u_i u_j}^D \equiv \tau_{u_i u_j} - \frac{1}{3} \tau_{u_k u_k} \delta_{ij} \quad (3.88)$$

$$\tau_{u_i u_j}^D = \tau_{u_i u_j} + \frac{2}{3} \bar{\rho} q^2 \delta_{ij}, \quad (3.89)$$

where the subgrid turbulent kinetic energy is defined as $q^2 \equiv \frac{1}{2} (\widetilde{u_k u_k} - \tilde{u}_k \tilde{u}_k)$. The deviatoric part of the subgrid stress tensor is then modeled similar to RANS closures as (Moin, et al. [14])

$$\tau_{u_i u_j}^D = 2\mu_t \left(\tilde{S}_{ij} - \frac{1}{3} \tilde{S}_{kk} \delta_{ij} \right). \quad (3.90)$$

Substituting this into (3.89) yields the modeled form of the full subgrid stress tensor

$$\tau_{u_i u_j} = 2\mu_t \left(\tilde{S}_{ij} - \frac{1}{3} \tilde{S}_{kk} \delta_{ij} \right) - \frac{2}{3} \bar{\rho} q^2 \delta_{ij}. \quad (3.91)$$

For low Mach-number flows, a vast majority of the turbulent kinetic energy is contained at resolved scales (Erlebacher, et al. [15]). For this reason, the subgrid turbulent kinetic energy q^2 will not be directly treated and will instead be included in the pressure as an additional normal stress. The Favre-filtered momentum equations then become

$$\begin{aligned} \int \frac{\partial \bar{\rho} \tilde{u}_i}{\partial t} dV + \int \bar{\rho} \tilde{u}_i \tilde{u}_j n_j dS + \int \left(\bar{p} + \frac{2}{3} \bar{\rho} q^2 \right) n_i dS = \\ \int 2(\mu + \mu_t) \left(\tilde{S}_{ij} - \frac{1}{3} \tilde{S}_{kk} \delta_{ij} \right) n_j dS + \int (\bar{\rho} - \rho_o) g_i dV, \end{aligned} \quad (3.93)$$

where LES closure models for the subgrid turbulent eddy viscosity μ_t are presented in *Turbulence Closure Models*.

Conservation of Energy

The integral form of the Favre-filtered energy equation used for turbulent transport is

$$\int \frac{\partial \bar{\rho} \tilde{h}}{\partial t} dV + \int \bar{\rho} \tilde{h} \tilde{u}_j n_j dS = - \int \bar{q}_j n_j dS - \int \tau_{hu_j} n_j dS - \int \frac{\partial \bar{q}_i^r}{\partial x_i} dV \quad (3.94)$$

$$+ \int \left(\frac{\partial P}{\partial t} + \tilde{u}_j \frac{\partial P}{\partial x_j} \right) dV + \int \overline{\tau_{ij} \frac{\partial u_i}{\partial x_j}} dV. \quad (3.95)$$

The simple Fickian diffusion velocity approximation, (3.45), is assumed, so that the mean diffusive heat flux vector \bar{q}_j is

$$\bar{q}_j = - \left[\frac{\mu}{\text{Pr}} \frac{\partial h}{\partial x_j} - \frac{\mu}{\text{Pr}} \sum_{k=1}^K h_k \frac{\partial Y_k}{\partial x_j} \right] - \frac{\mu}{\text{Sc}} \sum_{k=1}^K h_k \frac{\partial Y_k}{\partial x_j}. \quad (3.96)$$

If $\text{Sc} = \text{Pr}$, i.e., unity Lewis number ($\text{Le} = 1$), then the diffusive heat flux vector simplifies to $\bar{q}_j = -\frac{\mu}{\text{Pr}} \frac{\partial \tilde{h}}{\partial x_j}$. The viscous dissipation term is closed by

$$\begin{aligned} \overline{\tau_{ij} \frac{\partial u_i}{\partial x_j}} &= \left((\mu + \mu_t) \left(\frac{\partial \tilde{u}_i}{\partial x_j} + \frac{\partial \tilde{u}_j}{\partial x_i} \right) - \frac{2}{3} \left(\bar{\rho} \tilde{k} + \mu_t \frac{\partial \tilde{u}_k}{\partial x_k} \right) \delta_{ij} \right) \frac{\partial \tilde{u}_i}{\partial x_j} \\ &= \left[2\mu \tilde{S}_{ij} + 2\mu_t \left(\tilde{S}_{ij} - \frac{1}{3} \tilde{S}_{kk} \delta_{ij} \right) - \frac{2}{3} \bar{\rho} \tilde{k} \delta_{ij} \right] \frac{\partial \tilde{u}_i}{\partial x_j}. \end{aligned} \quad (3.97)$$

The turbulent diffusive flux vector τ_{hu_j} in (3.95) is defined as

$$\tau_{hu_j} \equiv \bar{\rho} \left(\widetilde{hu_j} - \tilde{h} \tilde{u}_j \right). \quad (3.98)$$

For RANS simulations, τ_{hu_j} represents the turbulent energy diffusive flux vector and is simplified to the form $\tau_{hu_j} = \overline{\rho h'' u_j''}$ by substitution of the Favre decomposition of each variable. It is then modeled by

$$\tau_{hu_j} = \overline{\rho h'' u_j''} = -\frac{\mu_t}{\text{Pr}_t} \frac{\partial \tilde{h}}{\partial x_j}, \quad (3.99)$$

where Pr_t is the turbulent Prandtl number and μ_t is the modeled turbulent eddy viscosity from momentum closure. For LES, τ_{hu_j} represents the subgrid turbulent energy diffusive flux vector, and is modeled in the same way as

$$\tau_{hu_j} = -\frac{\mu_t}{\text{Pr}_t} \frac{\partial \tilde{h}}{\partial x_j}, \quad (3.100)$$

where Pr_t is the subgrid turbulent Prandtl number and μ_t is the modeled subgrid turbulent eddy viscosity from momentum closure.

The resulting filtered and modeled turbulent energy equation for both RANS and LES is given in Libby and Williams [11], p. 25, as

$$\int \frac{\partial \bar{\rho} \tilde{h}}{\partial t} dV + \int \bar{\rho} \tilde{h} \tilde{u}_j n_j dS = \int \left(\frac{\mu}{\text{Pr}} + \frac{\mu_t}{\text{Pr}_t} \right) \frac{\partial \tilde{h}}{\partial x_j} n_j dS - \int \frac{\partial \bar{q}_i^r}{\partial x_i} dV + \int \left(\frac{\partial P}{\partial t} + \tilde{u}_j \frac{\partial P}{\partial x_j} \right) dV + \int \overline{\tau_{ij} \frac{\partial u_j}{\partial x_i}} dV. \quad (3.101)$$

This equation is also given in Gran et al. [16] (without the transient and radiation source terms and the additional term for laminar transport). The turbulent Prandtl number must have the same value as the turbulent Schmidt number for species transport to maintain unity Lewis number.

Conservation of Species

The integral form of the Favre-filtered species equation used for turbulent transport is

$$\int \frac{\partial \bar{\rho} \tilde{Y}_k}{\partial t} dV + \int \bar{\rho} \tilde{Y}_k \tilde{u}_j n_j dS = - \int \tau_{Y_k u_j} n_j dS - \int \overline{\rho Y_k \hat{u}_{j,k}} n_j dS + \int \bar{\omega}_k dV, \quad (3.102)$$

where the form of diffusion velocities (see (3.45)) assumes the Fickian approximation with a constant value of diffusion velocity for consistency with the turbulent form of the energy equation, (3.95).

The turbulent diffusive flux vector $\tau_{Y_k u_j}$ is defined as

$$\tau_{Y_k u_j} \equiv \bar{\rho} \left(\widetilde{Y_k u_j} - \tilde{Y}_k \tilde{u}_j \right). \quad (3.103)$$

For RANS simulations, $\tau_{Y_k u_j}$ represents the turbulent species diffusive flux vector and is simplified to the form $\tau_{Y_k u_j} = \overline{\rho Y_k'' u_j''}$ by substitution of the Favre decomposition of each variable. It is then modeled as

$$\tau_{Y_k u_j} = \overline{\rho Y_k'' u_j''} = - \frac{\mu_t}{\text{Sc}_t} \frac{\partial \tilde{Y}_k}{\partial x_i}, \quad (3.104)$$

where Sc_t is the turbulent Schmidt number for all species and μ_t is the modeled turbulent eddy viscosity from momentum closure. For LES, $\tau_{Y_k u_j}$ represents the subgrid turbulent species diffusive flux vector, and is modeled identically as

$$\tau_{Y_k u_j} = - \frac{\mu_t}{\text{Sc}_t} \frac{\partial \tilde{Y}_k}{\partial x_i}, \quad (3.105)$$

where Sc_t is the subgrid turbulent Schmidt number for all species and μ_t is the subgrid modeled turbulent eddy viscosity from momentum closure.

The Favre-filtered and modeled turbulent species transport equation for both RANS and LES then becomes (Gran et al. [16])

$$\int \frac{\partial \bar{\rho} \tilde{Y}_k}{\partial t} dV + \int \bar{\rho} \tilde{Y}_k \tilde{u}_j n_j dS = \int \left(\frac{\mu}{\text{Sc}} + \frac{\mu_t}{\text{Sc}_t} \right) \frac{\partial \tilde{Y}_k}{\partial x_j} n_j dS + \int \bar{\omega}_k dV. \quad (3.106)$$

If transporting both energy and species equations, the laminar Prandtl number must be equal to the laminar Schmidt number and the turbulent Prandtl number must be equal to the turbulent Schmidt number to maintain unity Lewis number. Although there is a species conservation equation for each species in a mixture of K species, only $K - 1$ species equations need to be solved since the mass fractions sum to unity and

$$\tilde{Y}_k = 1 - \sum_{j \neq k}^K \tilde{Y}_j. \quad (3.107)$$

Radiation Transport

The Favre-averaged energy equation, (3.101), requires the time-averaged radiative flux divergence. From (3.62), the time-averaged radiative flux divergence is given by

$$\overline{\frac{\partial q_i^r}{\partial x_i}} = 4\sigma \overline{\mu_a T^4} - \overline{\mu_a G}. \quad (3.108)$$

For optically thin turbulent eddies, which is the case for many combustion applications, fluctuations in the absorption coefficient and the scalar flux are weakly correlated [7] so (3.108) may be simplified to

$$\overline{\frac{\partial q_i^r}{\partial x_i}} = 4\sigma \overline{\mu_a T^4} - \bar{\mu}_a \bar{G}. \quad (3.109)$$

The time averaged scalar flux is obtained from the time averaged Boltzmann radiative transport equation

$$s_i \frac{\partial}{\partial x_i} \overline{I(s)} + \bar{\mu}_a \overline{I(s)} = \frac{\overline{\mu_a \sigma T^4}}{\pi}, \quad (3.110)$$

where the correlation between the turbulent fluctuations in the absorption coefficient and the intensity is assumed small to simplify the absorption term.

Both (3.109) and (3.110) include the time averaged emission term, $\overline{\alpha T^4}$, which may significantly increase the radiative emission from a turbulent flame above what would be estimated from the mean temperature and absorption coefficient values. The details of the closure used for this term are discussed in the turbulent combustion model section.

Comparison Between Time-Averaging and Favre-Averaging

The time-averaged (Reynolds-averaged) and Favre-averaged transport equations are given in the following section.

Conservation of Mass

The continuity equation:

- Time averaged:

$$\int \frac{\partial \bar{\rho}}{\partial t} dV + \int \bar{\rho} \bar{u}_j n_j dS + \int \overline{\rho' u'_j} n_j dS = 0 \quad (3.111)$$

- Favre averaged:

$$\int \frac{\partial \bar{\rho}}{\partial t} dV + \int \bar{\rho} \tilde{u}_j n_j dS = 0 \quad (3.112)$$

Conservation of Momentum

The momentum transport equations:

- Time averaged:

$$\int \frac{\partial \bar{\rho} \bar{u}_i}{\partial t} dV + \int \frac{\partial \bar{\rho' u'_i}}{\partial t} dV + \int \bar{\rho} \bar{u}_i \bar{u}_j n_j dS + \int \bar{\rho} \overline{u'_i u'_j} n_j dS + \int \bar{u}_j \overline{\rho' u'_i} n_j dS + \int \bar{u}_i \overline{\rho' u'_j} n_j dS + \int \overline{\rho' u'_i u'_j} n_j dS = 0 \quad (3.113)$$

- Favre averaged:

$$\int \frac{\partial \bar{\rho} \tilde{u}_i}{\partial t} dV + \int \bar{\rho} \tilde{u}_i \tilde{u}_j n_j dS + \int \bar{\rho} n_i dS = \int \bar{\tau}_{ij} n_j dS - \int \overline{\rho u''_i u''_j} n_j dS + \int \bar{\rho} g_i dV \quad (3.114)$$

Conservation of Energy

The energy transport equation, assume Lewis number is one:

- Favre averaged:

$$\int \frac{\partial \bar{\rho} \tilde{h}}{\partial t} dV + \int \bar{\rho} \tilde{h} \tilde{u}_j n_j dS = \int \frac{\kappa}{C_p} \frac{\partial \bar{h}}{\partial x_j} n_j dS - \int \overline{\rho h'' u''_j} n_j dS - \int \frac{\partial \bar{q}_i^r}{\partial x_i} dV \quad (3.115)$$

Conservation of Species

The species transport equation:

- Time averaged:

$$\int \frac{\partial \bar{\rho} \bar{Y}_k}{\partial t} dV + \int \frac{\partial \bar{\rho}' Y'_k}{\partial t} dV + \int \bar{\rho} \bar{Y}_k \bar{u}_j n_j dS + \int \bar{\rho} Y'_k u'_j n_j dS + \int \bar{u}_j \bar{\rho}' Y'_k n_j dS + \int \bar{Y}_k \bar{\rho}' u'_j n_j dS \quad (3.116)$$

- Favre averaged:

$$\int \frac{\partial \bar{\rho} \tilde{Y}_k}{\partial t} dV + \int \bar{\rho} \tilde{Y}_k \tilde{u}_j n_j dS = - \int \bar{\rho} Y''_k u''_j n_j dS + \int \bar{\rho} Y_k \hat{u}_{j,k} n_j dS + \int \bar{\omega}_k dV \quad (3.117)$$

Turbulence Closure Models

The Favre-filtered turbulent flow equations of the previous section have been modeled in terms of μ_t , the turbulent eddy viscosity for RANS simulations and the subgrid turbulent eddy viscosity for LES. Evaluation of this eddy viscosity is dependent upon the closure model selected. All models supported by SIERRA/Fuego are described below.

Standard $k - \epsilon$ RANS Model

The standard $k - \epsilon$ closure model is a two-equation type of model, where transport equations for the turbulent kinetic energy and the turbulent dissipation rate are solved to obtain length-scale and time-scale estimates for the local turbulence field, to be used for modeling the turbulent eddy viscosity μ_t . The turbulent kinetic energy, k , and the dissipation rate of turbulent kinetic energy, ϵ , are given by (Gran et al. [16])

$$\int \frac{\partial \bar{\rho} k}{\partial t} dV + \int \bar{\rho} k \tilde{u}_j n_j dS = \int \frac{\mu_t}{\sigma_k} \frac{\partial k}{\partial x_j} n_j dS + \int (P_k - \bar{\rho} \epsilon) dV \quad (3.118)$$

$$\int \frac{\partial \bar{\rho} \epsilon}{\partial t} dV + \int \bar{\rho} \epsilon \tilde{u}_j n_j dS = \int \frac{\mu_t}{\sigma_\epsilon} \frac{\partial \epsilon}{\partial x_j} n_j dS + \int \frac{\epsilon}{k} (C_{\epsilon 1} P_k - C_{\epsilon 2} \bar{\rho} \epsilon) dV, \quad (3.119)$$

respectively, where the turbulence production rate, P_k , is defined as

$$P_k \equiv -\overline{\rho u''_i u''_j} \frac{\partial \tilde{u}_i}{\partial x_j}, \quad (3.120)$$

and is modeled using the same Boussinesq approximation as in (3.85),

$$\begin{aligned} P_k &= \mu_t \left(\frac{\partial \tilde{u}_i}{\partial x_j} + \frac{\partial \tilde{u}_j}{\partial x_i} \right) \frac{\partial \tilde{u}_i}{\partial x_j} - \frac{2}{3} \left(\bar{\rho} k + \mu_t \frac{\partial \tilde{u}_k}{\partial x_k} \right) \frac{\partial \tilde{u}_m}{\partial x_m} \\ &= \left[2\mu_t \left(\tilde{S}_{ij} - \frac{1}{3} \tilde{S}_{kk} \delta_{ij} \right) - \frac{2}{3} \bar{\rho} \tilde{k} \delta_{ij} \right] \frac{\partial \tilde{u}_i}{\partial x_j}. \end{aligned} \quad (3.121)$$

The turbulent eddy viscosity is then given by the Prandtl-Kolmogorov relationship,

$$\mu_t = C_\mu \bar{\rho} k \tau. \quad (3.122)$$

where $\tau = \min(\frac{k}{\epsilon}, \Delta t_f)$. The filter time, Δt_f is provided by the temporally filtered Navier Stokes model (Tieszen et al. [17]). The parameters $C_{\epsilon 1}$, $C_{\epsilon 2}$, σ_k , and σ_ϵ are adjustable constants.

Frequently, although not formally justified in high Reynolds flows, the diffusion coefficient for the turbulent kinetic energy and turbulence dissipation, (3.118) and (3.119), may include the molecular viscosity. This option is supported within Fuego by entering the following command line in the Fuego region block, `include molecular viscosity in k-e diffusion term`.

Low Reynolds Number $k - \epsilon$ RANS Model

In the case of the low Reynolds number turbulent flows, the standard $k - \epsilon$ transport equations can be modified to contain additional damping functions to improve their accuracy. The low Reynolds number model of Launder and Sharma [18] is used, along with recent modification proposed by Mathur and He [19]. In this formulation, (3.118), includes an additional right-hand-side source term,

$$S_k^{lr} = -2\mu \left(\frac{\partial \sqrt{k}}{\partial x_j} \right)^2. \quad (3.123)$$

Moreover, the dissipation rate equation, which is now solved for $\tilde{\epsilon}$, i.e., the transformed dissipation rate that is numerically zero at the wall, also includes a new source term,

$$S_{\tilde{\epsilon}}^{lr} = 2 \frac{\mu \mu^T}{\rho} \left(\frac{\partial S}{\partial x_k} \right)^2, \quad (3.124)$$

where the strain-rate magnitude is defined by $S = \sqrt{2S_{ij}S_{ij}}$.

The constants in the dissipation rate equation are modified by damping coefficients, $C_{\epsilon 1} = f_1 C_{\epsilon 1}$ and $C_{\epsilon 2} = f_2 C_{\epsilon 2}$, where f_1 is unity and $f_2 = 1 - 0.3e^{-Re_t^2}$.

In this formulation, the eddy viscosity relationship defined by,

$$\mu_t = f_\mu C_\mu \bar{\rho} k \tau, \quad (3.125)$$

where the dampening function is $f_\mu = e^{-\frac{A_\mu}{(1+Re_t/50)^2}}$ and τ is now a function of $\tilde{\epsilon}$. The constant A_μ is 3.4 with $Re_t = \frac{\rho k^2}{\mu \tilde{\epsilon}}$.

The physical turbulent dissipation rate is related to the numerical turbulent dissipation rate by,

$$\epsilon = \tilde{\epsilon} + 2 \frac{\mu}{\rho} \left(\frac{\partial \sqrt{k}}{\partial x_j} \right)^2. \quad (3.126)$$

Wall functions for momentum and other turbulence quantities and transported variables are not used with this model.

RNG $k - \epsilon$ RANS Model

The RNG $k - \epsilon$ model was derived using a rigorous statistical decomposition of the velocity field called renormalization group (RNG) theory. This model has several significant benefits over the standard $k - \epsilon$ model, including improved accuracy for rapidly strained flows, swirling flows, and low Reynolds number flows, without additional modifications. Additionally, values for the model constants are derived analytically rather than being evaluated empirically. Papageorgakis and Assanis [20] describe the version of the RNG $k - \epsilon$ model as implemented here.

The same turbulent kinetic energy equation as in the standard $k - \epsilon$ model, (3.118), is used for the RNG $k - \epsilon$ equation. The turbulent kinetic energy dissipation rate equation is the same as (3.119), with the addition of a single source term on the right-hand-side of the equation,

$$S_{\epsilon}^{\text{RNG}} = -\frac{C_{\mu}\eta^3(1 - \eta/\eta_o)\epsilon^2}{1 + \beta\eta^3} \frac{1}{k}, \quad (3.127)$$

where C_{μ} , β , and η_o are model constants, and

$$\eta = (2\tilde{S}_{ij}\tilde{S}_{ij})^{\frac{1}{2}} \frac{k}{\epsilon}. \quad (3.128)$$

As with the standard $k - \epsilon$ model, the turbulent eddy viscosity is then given by the Prandtl-Kolmogorov relationship,

$$\mu_t = C_{\mu}\bar{\rho}k\tau. \quad (3.129)$$

$v^2 - f$ RANS Model

Durbin [21] introduced a method for handling the wall region without using either wall functions or damping functions. In his method a fine grid is required near the wall (e.g., the first grid point is typically within one dimensionless unit of distance from the wall where the coordinate normal to the wall is nondimensionalized with the inner scale for a turbulent boundary layer, $y^+ = yu_{\tau}/\nu < 1$ at the first grid point, where u_{τ} is the friction velocity, $\sqrt{\tau_w/\bar{\rho}}$). The model employs two transport equations in addition to slightly modified k and ϵ equations to account for the nonhomogeneous region near the wall. The eddy viscosity is formulated using the component of turbulent kinetic energy normal to the wall for velocity scaling (instead of using \sqrt{k} as in the standard $k - \epsilon$ model).

The turbulent kinetic energy, k , is given by (3.118) while the dissipation rate of turbulent kinetic energy, ϵ , is given by

$$\int \frac{\partial \bar{\rho}\epsilon}{\partial t} dV + \int \bar{\rho}\epsilon \tilde{u}_j n_j dS = \int \frac{\mu_t}{\sigma_{\epsilon}} \frac{\partial \epsilon}{\partial x_j} n_j dS + \int \frac{1}{T} (C'_{\epsilon 1} P_k - C_{\epsilon 2} \bar{\rho}\epsilon) dV. \quad (3.130)$$

The time scale, T , is the usual time scale k/ϵ , away from the wall region; however, near the wall, if k/ϵ becomes smaller than the Kolmogorov time scale $\sqrt{\nu/\epsilon}$, then the latter is used for T . This is

formally stated by

$$T = \min \left[T_1, \frac{\alpha}{2\sqrt{3}} \frac{k}{\bar{v}^2 C_\mu \sqrt{\tilde{S}^2}} \right] \quad (3.131)$$

$$T_1 = \max \left[\frac{k}{\epsilon}, 6\sqrt{\frac{\nu}{\epsilon}} \right], \quad (3.132)$$

where

$$\tilde{S}^2 = \tilde{S}_{ij} \tilde{S}_{ij} = \frac{1}{4} \left(\frac{\partial \tilde{u}_i}{\partial x_j} + \frac{\partial \tilde{u}_j}{\partial x_i} \right) \left(\frac{\partial \tilde{u}_i}{\partial x_j} + \frac{\partial \tilde{u}_j}{\partial x_i} \right) \quad (3.133)$$

and the modified constant, C'_{ϵ_1} , is given by

$$C'_{\epsilon_1} = C_{\epsilon_1} \left(1 + 0.045 \sqrt{k/\bar{v}^2} \right). \quad (3.134)$$

The model includes a transport equation for \bar{v}^2 ,

$$\frac{\partial \bar{\rho} \bar{v}^2}{\partial t} + \frac{\partial \bar{\rho} \tilde{u}_j \bar{v}^2}{\partial x_j} = \frac{\partial}{\partial x_j} \left[(\mu + \mu_t) \frac{\partial \bar{v}^2}{\partial x_j} \right] + \bar{\rho} k f - \frac{\bar{\rho} N \bar{v}^2}{T_1}. \quad (3.135)$$

An elliptic relaxation model equation is formulated to solve for the variable f in the above equation. The purpose of the elliptic relaxation model is to account for nonlocal effects such as wall blocking; the equation is given by

$$f - L^2 \frac{\partial}{\partial x_j} \left(\frac{\partial f}{\partial x_j} \right) = C_1 \frac{(2/3 - \bar{v}^2/k)}{T_1} + C_2 2\nu_t \frac{\tilde{S}^2}{k} + (N-1) \frac{\bar{v}^2/k}{T_1}. \quad (3.136)$$

Finally, the turbulent eddy viscosity is given by

$$\mu_t = C_\mu \bar{\rho} \bar{v}^2 \tau. \quad (3.137)$$

$k - \omega$ RANS Model

The $k - \omega$ turbulence model and its variants are similar in structure to the $k - \epsilon$ models. However, instead of computing the turbulent dissipation rate directly, the $k - \omega$ model models the transport the reciprocal of a turbulent timescale referred to as the turbulent frequency. This quantity, ω , can be related to the turbulent dissipation by

$$\epsilon = \beta^* k \omega. \quad (3.138)$$

The the transport equations are given by the 2006 model, (Wilcox [22]),

$$\int \frac{\partial \bar{\rho} k}{\partial t} dV + \int \bar{\rho} k \tilde{u}_j n_j dS = \int (\mu + \sigma_k \frac{\bar{\rho} k}{\omega}) \frac{\partial k}{\partial x_j} n_j dV + \int (P_k^\omega - \beta^* \bar{\rho} k \omega) dV, \quad (3.139)$$

$$\int \frac{\partial \bar{\rho} \omega}{\partial t} dV + \int \bar{\rho} \omega \tilde{u}_j n_j dS = \int (\mu + \sigma_\omega \frac{\bar{\rho} k}{\omega}) \frac{\partial \omega}{\partial x_j} n_j dV + \int \left(\gamma \frac{\omega}{k} P_k^\omega - \beta \bar{\rho} \omega^2 + \frac{\bar{\rho} \sigma_d}{\omega} \frac{\partial k}{\partial x_j} \frac{\partial \omega}{\partial x_j} \right) dV. \quad (3.140)$$

The user is to note the above standard for writing the effective diffusive flux coefficient. The model also has a number of adjustable parameters: $\beta_0 = 0.0708$, $\beta^* = 0.09$, $\gamma = \frac{13}{25}$, $C_{lim} = \frac{7}{8}$, $\sigma_k = 0.6$, and $\sigma_\omega = 0.5$. The constant β is given by,

$$\beta = \beta_0 f_\beta \quad (3.141)$$

where

$$f_\beta = \frac{1 + 85 \chi_\omega}{1 + 100 \chi_\omega} \quad (3.142)$$

The value of χ_ω is as follows:

$$\chi_\omega = \left| \frac{\Omega_{ij} \Omega_{jk} S_{ki}}{(\beta^* \omega)^3} \right| \quad (3.143)$$

The production term is the same as in $k - \epsilon$. Typically limiters are used to prevent it from exceeding the dissipation rate by too large an amount. Although the 2006 description does not speak of production limiters, other sources that use the 2006 model do, i.e.

$$P_k^\omega = \max(P_k, 10 \bar{\rho} k \omega). \quad (3.144)$$

The value of 10 is expected to be a user specified quantity (see input file manual for more details). In general, this term is defaulted to a very high number.

The eddy viscosity is

$$\mu_T = \bar{\rho} \frac{k}{\hat{\omega}}. \quad (3.145)$$

where $\hat{\omega}$ is,

$$\hat{\omega} = \max(\omega, C_{lim} \sqrt{\frac{2 S_{ij} S_{ij}}{\beta^*}}). \quad (3.146)$$

Shear Stress Transport (SST)

It has been observed that standard 1998 $k - \omega$ models display a strong sensitivity to the free stream value of ω . To remedy, this, an alternative set of transport equations have been used that are based on smoothly blending the $k - \omega$ model near a wall with $k - \epsilon$ away from the wall (see Mentor [23]). Because of the relationship between ω and ϵ , the transport equations for turbulent kinetic energy and dissipation can be transformed into equations involving k and ω . Aside from constants, the transport equation for k is unchanged. However, an additional cross-diffusion term is present in the ω equation. Blending is introduced by using smoothing which is a function of the

distance from the wall, $F(y)$. The transport equations for the Mentor 2003 model ([23]) are provided by the following:

$$\int \frac{\partial \bar{\rho} k}{\partial t} dV + \int \bar{\rho} k \tilde{u}_j n_j dS = \int (\mu + \hat{\sigma}_k \mu_t) \frac{\partial k}{\partial x_j} n_j + \int (P_k^\omega - \beta^* \bar{\rho} k \omega) dV, \quad (3.147)$$

$$\int \frac{\partial \bar{\rho} \omega}{\partial t} dV + \int \bar{\rho} \omega \tilde{u}_j n_j dS = \int (\mu + \hat{\sigma}_\omega \mu_t) \frac{\partial \omega}{\partial x_j} n_j + \int 2(1 - F) \frac{\bar{\rho} \sigma_{\omega 2}}{\omega} \frac{\partial k}{\partial x_j} \frac{\partial \omega}{\partial x_j} dV + \int \left(\frac{\hat{\gamma}}{\nu_t} P_k^\omega - \hat{\beta} \bar{\rho} \omega^2 \right) dV. \quad (3.148)$$

The model coefficients, $\hat{\sigma}_k$, $\hat{\sigma}_\omega$, $\hat{\gamma}$ and $\hat{\beta}$ must also be blended, which is represented by

$$\hat{\phi} = F \phi_1 + (1 - F) \phi_2. \quad (3.149)$$

where $\sigma_{k1} = 0.85$, $\sigma_{k2} = 1.0$, $\sigma_{\omega 1} = 0.5$, $\sigma_{\omega 2} = 0.856$, $\gamma_1 = \frac{5}{9}$, $\gamma_2 = 0.44$, $\beta_1 = 0.075$ and $\beta_2 = 0.0828$.

The blending function is given by

$$F = \tanh(A_1^4), \quad (3.150)$$

where

$$A_1 = \min \left(\max \left(\frac{\sqrt{k}}{\beta^* \omega y}, \frac{500 \mu}{\bar{\rho} y^2 \omega} \right), \frac{4 \bar{\rho} \sigma_{\omega 2} k}{CD_{k\omega} y^2} \right). \quad (3.151)$$

The final parameter is

$$CD_{k\omega} = \max \left(2 \bar{\rho} \sigma_{\omega 2} \frac{1}{\omega} \frac{\partial k}{\partial x_j} \frac{\partial \omega}{\partial x_j}, 10^{-10} \right). \quad (3.152)$$

In the 2003 SST model description, the production term is expected to be limited:

$$P_k^\omega = \max (P_k, 10 \bar{\rho} k \omega). \quad (3.153)$$

The value of 10 is expected to be a user specified quantity (see input file manual for more details). In general, this term is defaulted to a very high number.

An important component of the SST model is the different expression used for the eddy viscosity,

$$\mu_t = \frac{a_1 \bar{\rho} k}{\max (a_1 \omega, S F_2)}, \quad (3.154)$$

where F_2 is another blending function given by

$$F_2 = \tanh(A_2^2). \quad (3.155)$$

The final parameter is

$$A_2 = \max \left(\frac{2 \sqrt{k}}{\beta^* \omega y}, \frac{500 \mu}{\bar{\rho} \omega y^2} \right). \quad (3.156)$$

Standard Smagorinsky LES Model

The standard Smagorinsky LES closure model approximates the subgrid turbulent eddy viscosity using a mixing length-type model, where the LES grid filter size Δ provides a natural length scale. The subgrid eddy viscosity is modeled simply as (Smagorinsky [24])

$$\mu_t = \rho (C_s \Delta)^2 |\tilde{S}|, \quad (3.157)$$

where the strain rate tensor magnitude is defined as $|\tilde{S}| \equiv (2\tilde{S}_{ij}\tilde{S}_{ij})^{\frac{1}{2}}$. The constant coefficient C_s typically varies between 0.1 and 0.24 and should be carefully tuned to match the problem being solved (Rogallo and Moin [25]). It is assigned a value of 0.17 here.

Although this model is desirable due to its simplicity and efficiency, care should be taken in its application. It is known to predict subgrid turbulent eddy viscosity proportional to the shear rate in the flow, independent of the local turbulence intensity. Non-zero subgrid turbulent eddy viscosity is even predicted in completely laminar regions of the flow, sometimes even preventing a natural transition to turbulence. Therefore, this model should only be used when this behavior will not adversely affect results.

Dynamic Smagorinsky LES Model

As mentioned in the previous section, the standard Smagorinsky model requires careful tuning of the constant model coefficient for the particular problem being simulated, and it is often overly-dissipative due to its inability to adapt to the local turbulent environment. Germano et al. [26] developed an improvement over the standard Smagorinsky model, where the coefficient C_s is dynamically calculated based on the local turbulence field. A generalization of this method for variable-density flow is used here (Moin et al. [14]).

Similar to the standard Smagorinsky LES closure model, the subgrid eddy viscosity is modeled by the mixing length approximation

$$\mu_t = C_R \bar{\rho} \Delta^2 |\tilde{S}|, \quad (3.158)$$

where the strain rate tensor magnitude is defined as $|\tilde{S}| \equiv (2\tilde{S}_{ij}\tilde{S}_{ij})^{\frac{1}{2}}$. The coefficient C_R is dynamically evaluated by taking advantage of scale similarity in the inertial range of the turbulence spectrum, near the minimum resolved scales. This is done by introducing a “test filter” which is identical to the grid filter defined in (3.71) except for having a larger filter size denoted by $\hat{\Delta}$. The test filter of variable ϕ is denoted by $\hat{\phi}$.

The previously-defined subgrid stress tensor can be rewritten as

$$\begin{aligned} \tau_{u_i u_j} &\equiv -(\bar{\rho} \widetilde{u_i u_j} - \bar{\rho} \tilde{u}_i \tilde{u}_j) \\ &= -\left(\overline{\rho u_i u_j} - \frac{\overline{\rho u_i} \overline{\rho u_j}}{\bar{\rho}} \right) \end{aligned} \quad (3.159)$$

and an analogous larger-scale “subtest” stress $T_{u_i u_j}$ can be analogously defined as

$$T_{u_i u_j} \equiv - \left(\widehat{\overline{\rho u_i u_j}} - \frac{\widehat{\overline{\rho u_i}} \widehat{\overline{\rho u_j}}}{\hat{\rho}} \right), \quad (3.160)$$

where the $\hat{(\cdot)}$ notation denotes resolved quantities that have been passed through the test filter. These two stresses can be related to each other through the algebraic identity of Germano [27],

$$L_{u_i u_j} \equiv T_{u_i u_j} - \widehat{\tau_{u_i u_j}} \quad (3.161)$$

$$L_{u_i u_j} = - \left(\widehat{\overline{\rho \tilde{u}_i \tilde{u}_j}} - \frac{\widehat{\overline{\rho \tilde{u}_i}} \widehat{\overline{\rho \tilde{u}_j}}}{\hat{\rho}} \right). \quad (3.162)$$

Note that the right-hand side of (3.162) is completely computable in terms of resolved quantities.

By modeling the two stresses in (3.161) and equating them to (3.162), the model coefficient C_R can be dynamically evaluated. The subtest stress is modeled analogously to the subgrid stress, as

$$\tau_{u_i u_j} \approx 2C_R \bar{\rho} \Delta^2 |\tilde{S}| \left(\tilde{S}_{ij} - \frac{1}{3} \tilde{S}_{kk} \delta_{ij} \right) \quad (3.163)$$

$$T_{u_i u_j} \approx 2C_R \hat{\rho} \hat{\Delta}^2 |\hat{S}| \left(\widehat{\tilde{S}_{ij}} - \frac{1}{3} \widehat{\tilde{S}_{kk} \delta_{ij}} \right), \quad (3.164)$$

where C_R is assumed to be the same at both scales. The test-filtered strain rate tensor is defined similar to $|\tilde{S}|$ as

$$|\hat{S}| \equiv \left(2 \widehat{\tilde{S}_{ij} \tilde{S}_{ij}} \right)^{\frac{1}{2}}. \quad (3.165)$$

Notice that when the modeled forms of $\tau_{u_i u_j}$ and $T_{u_i u_j}$ are substituted into (3.161), C_R appears inside a test filtering operation. Formally solving this system of equations for C_R requires the expensive proposition of solving an additional set of coupled integro-differential equations (Ghosal et al. [28]). Alternatively, it is common practice to remove C_R from the test filter with the assumption that it is varying slowly over distances on the order of the test filter size. This greatly simplifies calculations, although it yields a system of over-determined equations for this single constant. The square of the error involved in this approximation is $Q = (L_{ij} - C_R M_{ij})^2$, where

$$L_{u_i u_j} = - \left(\widehat{\overline{\rho \tilde{u}_i \tilde{u}_j}} - \frac{\widehat{\overline{\rho \tilde{u}_i}} \widehat{\overline{\rho \tilde{u}_j}}}{\hat{\rho}} \right) \quad (3.166)$$

$$M_{u_i u_j} = 2 \hat{\rho} \hat{\Delta}^2 |\hat{S}| \left(\widehat{\tilde{S}_{ij}} - \frac{1}{3} \widehat{\tilde{S}_{kk} \delta_{ij}} \right) - 2 \bar{\rho} \Delta^2 |\tilde{S}| \left(\tilde{S}_{ij} - \frac{1}{3} \tilde{S}_{kk} \delta_{ij} \right). \quad (3.167)$$

Minimizing this error in a least-squares fashion yields an expression for the modeled Smagorinsky coefficient (Lilly [29]),

$$C_R = \frac{L_{u_i u_j} M_{u_i u_j}}{M_{u_i u_j} M_{u_i u_j}}, \quad (3.168)$$

that can be used directly in (3.158) for the subgrid turbulent eddy viscosity.

Due to the above simplifications, the model constant C_R can sometimes fluctuate wildly to both large positive and negative values. These fluctuations can possibly lead to numerical instability, so they must be controlled. A common solution, and one that is taken here, is to pass the numerator and denominator of (3.168) through the test filter, yielding

$$C_R = \frac{\overline{L_{u_i u_j} M_{u_i u_j}}}{\overline{M_{u_i u_j} M_{u_i u_j}}}. \quad (3.169)$$

This can be crudely justified by recognizing that C_R was already assumed to vary slowly over distances equal to the test filter size, so that this filtering operation is simply enforcing that assumption.

This form of the dynamic Smagorinsky closure model allows energy backscatter, which is an intermittent transfer of turbulent kinetic energy from small scales to larger scales rather than the typical cascade from large to small scales. While backscatter can occur in real turbulent flows, the predicted negative eddy viscosities of the dynamic Smagorinsky model are more often attributable to model errors than to a real physical backscatter process. This can easily destabilize a simulation, so negative eddy viscosity is disallowed in the present formulation.

The only free parameter in the dynamic Smagorinsky closure model is the ratio between the test and grid filter sizes, $\alpha = \hat{\Delta}/\Delta$. Solutions are fairly insensitive to the choice of α , although values of around $\alpha = 2$ are usually considered optimal (Germano et al. [26]). This ratio is dictated by the box filter formulation used in Fuego and the mesh topology selected by the user. The test filter volume for a particular CVFEM node is defined as the volume of all surrounding finite elements that contain that node. (See *Numerics* for more information about the CVFEM formulation.) On uniform hexahedral and uniform quadrilateral meshes, the test filter ratio will have a value of 2.0. The ratio will be around 1.59 for uniform tetrahedral meshes and around 1.73 for uniform triangular meshes, which are still reasonable values.

Subgrid-Scale Kinetic Energy One-Equation LES Model

The subgrid scale kinetic energy one-equation turbulence model, or $Ksgs$ model, represents a simple LES closure model. The transport equation for subgrid turbulent kinetic energy is given by

$$\int \frac{\partial \bar{\rho} k^{sgs}}{\partial t} dV + \int \bar{\rho} k^{sgs} \tilde{u}_j n_j dS = \int \frac{\mu_t}{\sigma_k} \frac{\partial k^{sgs}}{\partial x_j} n_j dS + \int \left(P_k^{sgs} - D_k^{sgs} \right) dV. \quad (3.170)$$

The production of subgrid turbulent kinetic energy, P_k^{sgs} , is modeled by (3.121) while the dissipation of turbulent kinetic energy, D_k^{sgs} , is given by

$$D_k^{sgs} = \bar{\rho} C_\epsilon \frac{k^{sgs \frac{3}{2}}}{\Delta}, \quad (3.171)$$

where the grid filter length, Δ , is given in terms of the grid cell volume by

$$\Delta = V^{\frac{1}{3}}. \quad (3.172)$$

The subgrid turbulent eddy viscosity is then provided by

$$\mu_t = \bar{\rho} C_{\mu\epsilon} \Delta k^{\text{sgs}\frac{1}{2}}, \quad (3.173)$$

where the values of C_ϵ and $C_{\mu\epsilon}$ are 0.845 and 0.0856, respectively.

Wall-Resolved LES Approach

The baseline one-equation model is designed to support applications where wall boundary layers are not resolved, i.e., wall-modeled LES (WMLES). For simulations that include wall-resolved boundary layer meshing approaches, i.e., near wall normalized wall unit, y^+ less than unity, additional transport terms and blending functions must be added. These additional contributions ensure that quantities such as turbulent viscosity and normalized turbulence kinetic energy scale cubically and quadratically with y^+ . The approach followed in Fuego to support the wall-resolved LES (WRLES) use case is to support the model developed by Inagaki [30].

The one-equation form for the Inagaki model is very similar to the baseline model with only a few select changes that allow proper scaling of the subgrid-scale turbulence kinetic energy near a wall boundary. In this model, the dissipation rate, D_k^{sgs} now includes an added dissipation term,

$$D_k^{\text{sgs}} = \bar{\rho} \epsilon_{\text{sgs}} = \bar{\rho} C_\epsilon \frac{k_{\text{sgs}}^{\frac{3}{2}}}{\Delta} + 2\mu \frac{\partial \sqrt{k_{\text{sgs}}}}{\partial x_j} \frac{\partial \sqrt{k_{\text{sgs}}}}{\partial x_j}, \quad (3.174)$$

while the turbulent viscosity is modified via a blending coefficient f_μ ,

$$\mu_t = f_\mu \bar{\rho} C_{\mu\epsilon} \Delta k^{\text{sgs}\frac{1}{2}}. \quad (3.175)$$

The functional form for f_μ provided in Inagaki [30] is,

$$f_\mu = 1 - \exp \left[- \left(\left(\frac{y'_\epsilon}{A_o} \right)^{\frac{2}{3}} \right)^{B_o} \right] \quad (3.176)$$

where the model constants A_o and B_o are 20 and 2, the former of which closely replicates Van Driest scaling while the B_o value of 2 provides the functional form of $f_\mu \propto y^2$ (here, y is the minimum distance to the wall, a value that is internally computed within Fuego).

The functions y'_ϵ and u'_ϵ are,

$$y'_\epsilon = \frac{y u'_\epsilon}{\nu} \quad (3.177)$$

and

$$u'_\epsilon = (\nu \epsilon_{\text{sgs}})^{\frac{1}{4}} \sqrt{C_l \frac{y}{\Delta}}, \quad (3.178)$$

that include the static model constant C_l , which is specified to be a value of 4. As implied by (3.174), ϵ_{sgs} is the modified dissipation rate that includes the augmented dissipation by gradients of $\sqrt{k_{sgs}}$,

$$\epsilon_{sgs} = C_\epsilon \frac{k_{sgs}^{\frac{3}{2}}}{\Delta} + 2\nu \frac{\partial \sqrt{k_{sgs}}}{\partial x_j} \frac{\partial \sqrt{k_{sgs}}}{\partial x_j}. \quad (3.179)$$

The model constants for C_ϵ and C_{μ_ϵ} in Inagaki are provided as 0.835 and 0.054, respectively. Validation test cases demonstrate a somewhat weak dependence on the precise values of the model constants as with the diffusive flux vector scaling, σ^k , which was provided as 0.54. Note that this model is not designed to be activated in concert with the dynamic procedure to follow.

Dynamic Subgrid-Scale Kinetic Energy One-Equation LES Model

Similar to the dynamic Smagorinsky model in *Dynamic Smagorinsky LES Model*, a dynamic version is developed for the subgrid kinetic energy model. The standard version with fixed coefficients over-predicts turbulent viscosity while the dynamic version is known to offer a better predictability. In Fuego, C_ϵ and C_{μ_ϵ} are calculated dynamically which is considered to be a standard approach for the dynamic $Ksgs$ model [31]. The concept of “test filter” is identical to that of the dynamic Smagorinsky model in *Dynamic Smagorinsky LES Model*. Subgrid-scale kinetic energy for grid-filter and test-filter levels are

$$k^{sgs} = \frac{1}{2} (\overline{u_k u_k} - \tilde{u}_k \tilde{u}_k), \quad (3.180)$$

$$k^{test} = \frac{1}{2} (\overline{\tilde{u}_k \tilde{u}_k} - \hat{u}_k \hat{u}_k). \quad (3.181)$$

Exact form of the dissipation D_k^{sgs} in (3.170) is

$$D_k^{sgs} = 2\bar{\mu} \left[\overline{S_{ij}^* D_{ij}^*} - \tilde{S}_{ij}^* \tilde{D}_{ij}^* \right] \quad (3.182)$$

where $S_{ij}^* = S_{ij} - \frac{1}{3} S_{kk} \delta_{ij}$, $D_{ij}^* = D_{ij} - \frac{1}{3} D_{kk} \delta_{ij}$, and $D_{ij} = \partial u_i / \partial x_j$. Meanwhile, k^{test} dissipates by both molecular and turbulent viscosities of the grid-filtered level since the quantity is fully resolved in the test-filter level [31].

$$D_k^{test} = 2(\bar{\mu} + \mu_t) \left[\overline{\tilde{S}_{ij}^* \tilde{D}_{ij}^*} - \hat{S}_{ij}^* \hat{D}_{ij}^* \right] \quad (3.183)$$

Using scale similarity, (3.171) applies to the test-filter level as

$$D_k^{test} = C_\epsilon \hat{\rho} \frac{k^{test \frac{3}{2}}}{\hat{\Delta}}, \quad (3.184)$$

and therefore, C_ϵ is calculated by

$$C_\epsilon = \frac{2(\bar{\mu} + \mu_t) \left[\overline{\tilde{S}_{ij}^* \tilde{D}_{ij}^*} - \hat{S}_{ij}^* \hat{D}_{ij}^* \right]}{\hat{\rho} k^{test \frac{3}{2}} / \hat{\Delta}} \quad (3.185)$$

The other coefficient, $C_{\mu\epsilon}$, is computed similarly to the (3.168) as

$$C_{\mu\epsilon} = \frac{L_{u_i u_j} M_{u_i u_j}}{M_{u_i u_j} M_{u_i u_j}}, \quad (3.186)$$

where $L_{u_i u_j}$ is defined identically to (3.166) and $M_{u_i u_j}$ is simplified by

$$M_{u_i u_j} = 2\hat{\rho}\hat{\Delta}k^{\text{test}\frac{1}{2}}\widehat{\tilde{S}_{ij}^*}. \quad (3.187)$$

Note that dynamic subgrid kinetic energy model does not require an additional filtering as in (3.169). The method will predict negative values of turbulent viscosity, similar to dynamic smagorinsky. Numerical issues can arise if model predicts an overly negative value of the turbulent viscosity for an overly long timescale. Because of this, clipping is suggested [31]

$$\mu_{t,\text{clip}} = \max(\mu_{t,\text{model}}, -\tau_{\text{clip}}\bar{\mu}) \quad (3.188)$$

with a value of τ_{clip} of unity, such that the effective viscosity remains non-negative. In Fuego, we allow this value to be user-defined with a default value of zero. A sufficiently large value of τ_{clip} will effectively result in using the directly modeled turbulent viscosity.

Buoyancy Models for the Production Rate

There are three supported models that augment the production of turbulent kinetic energy via buoyancy contributions, buoyant vorticity generation [32], Rodi's [33], and de Ris' [34] buoyancy term.

The buoyant vorticity generation model has been developed and validated by Sandia National Laboratories group 9132 for use in large scale buoyant plumes. The model attempts to augment the production of turbulent kinetic energy by adding a source term, G_B to both the turbulent kinetic energy and dissipation rate equation that is related to the baroclinic torque,

$$G_B = \frac{C_{bvg}(\mu + \mu_t) \left\| \frac{\partial \rho}{\partial x_j} \times \frac{\partial P}{\partial x_j} \right\|}{\rho^2}. \quad (3.189)$$

of the model.

The buoyancy model of Rodi is given by

$$G_B = \beta \frac{\mu_t}{Pr_t} \frac{\partial T}{\partial x_j} g_j. \quad (3.190)$$

De Ris' buoyancy model offers two versions - flaming

$$C_{deris}(\rho_\infty - \rho_f)\tilde{g}k^{0.5} \quad (3.191)$$

and non-flaming.

$$C_{deris}\Delta k^{0.5}(|\nabla \rho \times \vec{g}| - \nabla \rho \cdot \vec{g}) \quad (3.192)$$

Default value of the user-defined coefficient C_{deriv} is 0.01. Note that ambient and flame density, rather than local density, matters on the flaming version.

In each model, derivatives are evaluated at the subcontrol volume center while the property values are lumped.

The right hand side of the turbulent kinetic energy equation for all model is $rhs+ = \int G_B dV$. For the dissipation rate equation, the source term is $rhs+ = \int C_{\epsilon 3} \frac{1}{T} G_B dV$ for the buoyant vorticity generation model while it is $rhs+ = \int C'_{\epsilon 1} C_{\epsilon 4} \frac{1}{T} G_B dV$ otherwise. Recall that the inverse time scale is determined by the turbulence model of choice, i.e., $\frac{\epsilon}{k}$ for the standard $k - \epsilon$ model and provided in (3.131) for the $v^2 - f$ model.

Note that the use of the buoyancy models has not been evaluated with the $v^2 - f$ model.

Turbulence closure model constants

For each of the aforementioned turbulence closure models, there are several constant coefficients which may be modified by the user in the input deck. [Table 3.1](#), [Table 3.2](#), [Table 3.3](#), and [Table 3.4](#) list these parameters, their mapping to input deck names, and default values. Each of these default values may be modified by the user by specifying the respective **Turbulence Model Parameter** line in the **Global Constants** block under the **Sierra** domain.

Table 3.1: Constant parameters for $k - \epsilon$ turbulence models.

Turbulence Model	Symbol	User Input Name	Default Value
Standard $k - \epsilon$	C_μ	Cmu	0.09
	$C_{\epsilon 1}$	Ceps_1	1.44
	$C_{\epsilon 2}$	Ceps_2	1.92
	C_χ	Cchi	2.0
	σ_k	Sigma_K	1.0
	σ_ϵ	Sigma_E	1.3
Low Reynolds $k - \epsilon$	C_μ	Cmu	0.09
	$C_{\epsilon 1}$	Ceps_1	1.44
	$C_{\epsilon 2}$	Ceps_2	1.92
	σ_k	Sigma_K	1.0
	σ_ϵ	Sigma_E	1.3
	A_μ	Amu	3.4
RNG $k - \epsilon$	C_μ	Cmu	0.0837
	$C_{\epsilon 1}$	Ceps_1	1.42
	$C_{\epsilon 2}$	Ceps_2	1.68
	σ_k	Sigma_K	0.7194
	σ_ϵ	Sigma_E	0.7194
$\nu^2 - f$	C_μ	Cmu	0.22
	$C_{\epsilon 1}$	Ceps_1	1.4
	$C_{\epsilon 2}$	Ceps_2	1.9
	σ_k	Sigma_K	1.0
	σ_ϵ	Sigma_E	1.0
	C_1	CF_1	0.4
	C_2	CF_2	0.3
	α	Alpha	0.6
	C_T	Nseg	6.0
	C_L	CL	0.23
	C_η	Ceta	70.0

Table 3.2: Constant parameters for $k - \omega$ turbulence models.

Turbulence Model	Symbol	User Input Name	Default Value
$k - \omega$	β_0	Beta_Zero	0.0708
	β^*	Beta_Star	0.09
	σ_k	Sigma_K	3/5
	σ_ω	Sigma_W	0.5
	γ	Gamma	13/25
	C_{lim}	Clim	7/8
SST	A_1	A_One	0.31
	β_1	Beta_One	0.075
	β_2	Beta_Two	0.0828
	β^*	Beta_Star	0.09
	γ_1	Gamma_One	5/9
	γ_2	Gamma_Two	0.44
	σ_{k1}	Sigma_K_One	0.85
	σ_{k2}	Sigma_K_Two	1.0
	$\sigma_{\omega 1}$	Sigma_W_One	0.5
	$\sigma_{\omega 2}$	Sigma_W_Two	0.856

Table 3.3: Constant parameters for LES turbulence models.

Turbulence Model	Symbol	User Input Name	Default Value
One-equation	C_v	Cv	0.5
	C_ϵ	Ceps	0.845
	$C_{\mu\epsilon}$	Cmueps	0.0856
Standard Smagorinsky	C_v	Cv	0.5
	C_s	Cs	0.17
Dynamic Smagorinsky	C_s	Cs	0.17

Table 3.4: Constant parameters for miscellaneous turbulence models. Default values may be changed using the $k - \epsilon$ model parameters input.

Model	Symbol	User Input Name	Default Value
Buoyant vorticity generation	C_{BVG}	Cbvg	0.35
	$C_{\epsilon 3}$	Ceps_3	0.0
Rodi's source term	$C_{\epsilon 4}$	C_eps4	0.0
EDC laminar limit	$C_{\gamma, lam}$	Cgammalam	2.0
	$C_{\tau, lam}$	Ctaulam	0.02
	$C_{lam, trans}$	Clamtrans	40.0

Wall Boundary Conditions for Turbulence Models

Resolution of Boundary Layer; Momentum

The wall velocity boundary condition is the typical no-slip boundary; a specified value is expected.

Resolution of Boundary Layer; Turbulence Quantities

The resolution of the boundary layer is expected when the low Reynolds number or v^2 - f model is in use.

For the v^2 - f model, the wall turbulent kinetic energy and normal fluctuating stress component are each zero while the dissipation rate is given by

$$\epsilon_w = 2\nu \frac{\partial k^{1/2}}{\partial x_j} . \quad (3.193)$$

For the low Reynolds number, the wall turbulent kinetic energy is again zero while the dissipation rate, here considered to be the isotropic dissipation rate, is given as zero.

Resolution of Boundary Layer; Enthalpy

The wall value of enthalpy is computed based on the specified temperature and either reference or local mass fractions. In the case of a heat flux boundary condition, the wall node value is computed based on the control volume balance.

Wall Functions for Turbulent Flow Boundary Conditions

Resolution of the near-wall turbulent boundary layer can require extensive mesh points. Adjacent to the wall exists an extremely thin viscous sublayer where these forces dominate and are relatively insensitive to free stream parameters. Following the viscous sublayer is a buffer layer, the so-called “log-layer” and, ultimately, the turbulent core. The Van Driest hypothesis of turbulent flow near solid boundaries can be used to derive the appropriate form of this log-law zone. In general, the use of wall functions eliminates the need to resolve the near wall layers by prescribing the wall shear stress and resulting force based on the law of the wall (Launder and Spalding [35]).

The primary assumptions of the law of the wall are

- Local equilibrium of turbulent kinetic energy production and dissipation,
- Constant shear stress within the log-law region,
- Couette flow (pure shear flow).

Wall Functions; Momentum

The wall shear stress enters the discretization of the momentum equations by the term

$$\int \tau_{ij} n_j dS = -F_{wi}. \quad (3.194)$$

Wall functions are used to prescribe the value of the wall shear stress rather than resolving the boundary layer within the near-wall domain. The fundamental momentum law of the wall formulation, assuming fully-developed turbulent flow near a no-slip wall, can be written as (Launder and Spalding [35])

$$u^+ = \frac{u_{\parallel}}{u_{\tau}} = \frac{1}{\kappa} \ln(Ey^+), \quad (3.195)$$

where u^+ is defined by the the near-wall parallel velocity, u_{\parallel} , normalized by the wall friction velocity, u_{τ} . The wall friction velocity is related to the turbulent kinetic energy by

$$u_{\tau} = C_{\mu}^{1/4} k^{1/2}. \quad (3.196)$$

by assuming that the production and dissipation of turbulence is in local equilibrium. Moreover, y^+ is defined as the normalized perpendicular distance from the point in question to the wall,

$$y^+ = \frac{\rho Y_p}{\mu} \left(\frac{\tau_w}{\rho} \right)^{1/2} = \frac{\rho Y_p u_{\tau}}{\mu} \quad (3.197)$$

The classical law of the wall is as follows:

$$u^+ = \frac{1}{\kappa} \ln(y^+) + C \quad (3.198)$$

where κ is the von Karman constant and C is the dimensionless integration constant that varies based on authorship and surface roughness. The above expression can be re-written as

$$u^+ = \frac{1}{\kappa} \ln(y^+) + \frac{1}{\kappa} \ln(\exp(\kappa C)) \quad (3.199)$$

or

$$u^+ = \frac{1}{\kappa} (\ln(y^+) + \ln(\exp(\kappa C))) \quad (3.200)$$

$$u^+ = \frac{1}{\kappa} \ln(Ey^+) \quad (3.201)$$

where E is referred to in the text as the dimensionless wall roughness parameter and is described by

$$E = \exp(\kappa C) \quad (3.202)$$

In Fuego, κ is set to the value of 0.42 while the value of E is set to 9.8 for smooth walls. White [36] suggests values of $\kappa = 0.41$ and $E = 7.768$. The viscous sublayer is assumed to extend to a value of $y^+ = 11.63$.

The wall shear stress, τ_w , can be expressed as

$$\tau_w = \rho u_\tau^2 = \rho u_\tau \frac{u_\parallel}{u^+} = \frac{\rho \kappa u_\tau}{\ln(E y^+)} u_\parallel = \lambda_w u_\parallel, \quad (3.203)$$

where λ_w is simply the grouping of the factors from the law of the wall. For values of y^+ less than 11.63, the wall shear stress is given by

$$\tau_w = \mu \frac{u_\parallel}{Y_p}. \quad (3.204)$$

The force imparted by the wall, for the i_{th} component of velocity, can be written as

$$F_{w,i} = -\lambda_w A_w u_{i\parallel}, \quad (3.205)$$

where A_w is the total area over which the shear stress acts.

The use of a general, non-orthogonal mesh adds a slight complexity to specifying the force imparted on the fluid by the wall. As shown in (3.205), the velocity component parallel to the wall must be determined. Use of the unit normal vector, n_j , provides an easy way to determine the parallel velocity component by the following standard vector projection,

$$\Pi_{ij} = [\delta_{ij} - n_i n_j]. \quad (3.206)$$

Carrying out the projection of a general velocity, which is not necessarily parallel to the wall, yields the velocity vector parallel to the wall,

$$u_{i\parallel} = \Pi_{ij} u_j = u_i \left(1 - n_i^2\right) - \sum_{j=1; j \neq i}^n u_j n_i n_j. \quad (3.207)$$

Note that the component that acts on the particular i^{th} component of velocity,

$$-\lambda_w A_w (1 - n_i n_i) u_{i\parallel}, \quad (3.208)$$

provides a form that can be potentially treated implicitly; i.e., in a way to augment the diagonal dominance of the central coefficient of the i^{th} component of velocity. The use of residual form adds a slight complexity to this implicit formulation only in that appropriate right-hand-side source terms must be added.

Finally, while the default wall-modeling approach in Fuego is to use the near-wall element's representation of u_\parallel , in addition to a length scale associated with this first element, an exchange-based approach of [37] is also supported. In this modeling paradigm, a user-supplied projected distance off the wall (in the wall normal direction) is provided to compute the so-called “exchange-velocity” that is used for u_\parallel . Generally, this distance is roughly three times the normal distance of the near wall element and requires a parallel search to seek the owning element from which u_\parallel is computed (using the underlying elemental interpolation function). In addition to reported increased accuracy of this model, the advantage is that the specified projected distance can be held constant during a mesh refinement study. Further modeling enhancements such as those outlined in Larsson et al. [37], e.g., the ODE wall-stress model, are also possible.

Wall Functions; Turbulent Kinetic Energy

The near wall turbulent kinetic energy can be obtained by two different procedures. The most common approach is to solve a transport equation for the near wall value of turbulent kinetic energy with a modified production and dissipation term on the right hand side of the turbulent kinetic energy equation, (3.118). As will be shown below, the form of the near wall production and dissipation term are determined based on equilibrium arguments, i.e., $P_k = \rho\epsilon$.

Another common approach is to assign the value of turbulent kinetic energy that strictly results in the equality $P_k = \rho\epsilon$. In this formulation, it is assumed that the convection and diffusive flux is zero across the control volume.

Both procedures, which formally do not address the role of buoyancy production, begin with the determination of the near wall value of the production of turbulent kinetic energy. The turbulent kinetic energy production term is consistent with the law of the wall formulation and can be expressed as

$$P_{kw} = \tau_w \frac{\partial u_{\parallel}}{\partial y}. \quad (3.209)$$

The parallel velocity, u_{\parallel} , can be related to the wall shear stress by

$$\tau_w \frac{u^+}{y^+} = \mu \frac{u_{\parallel}}{Y_p}. \quad (3.210)$$

Taking the derivative of both sides of (3.210), and substituting this relationship into (3.209) yields,

$$P_{kw} = \frac{\tau_w^2}{\mu} \frac{\partial u^+}{\partial y^+}. \quad (3.211)$$

Applying the derivative of the law of the wall formulation, (3.195), provides the functional form of $\partial u^+ / \partial y^+$,

$$\frac{\partial u^+}{\partial y^+} = \frac{\partial}{\partial y^+} \left[\frac{1}{\kappa} \ln(Ey^+) \right] = \frac{1}{\kappa y^+}. \quad (3.212)$$

Substituting (3.212) within (3.211) yields a commonly used form of the near wall production term,

$$P_{kw} = \frac{\tau_w^2}{\rho \kappa u_{\tau} Y_p}. \quad (3.213)$$

Assuming local equilibrium, $P_k = \rho\epsilon$, and using (3.213) and (3.196) provides the form of the near wall turbulence dissipation,

$$\epsilon = \frac{u_{\tau}^3}{\kappa Y_p} = \frac{C_{\mu}^{3/4} k^{3/2}}{\kappa Y_p}, \quad (3.214)$$

while the form of the wall shear stress is given by,

$$\tau_w = \rho C_\mu^{1/2} k \quad (3.215)$$

Under the above assumptions, the near wall value for turbulent kinetic energy, in the absence of convection, diffusion, or accumulation is given by,

$$k = \frac{u_\tau^2}{C_\mu^{1/2}}. \quad (3.216)$$

If the second method (Dirichlet condition on near wall turbulent kinetic energy) is to be used, the value of the wall friction velocity, u_τ , can be obtained in an iterative manner (Sondak and Pletcher [38]) by use of (3.195). This method has been used and shown to be satisfactory (Elkaim [39]) and strictly enforces the assumptions of the law of the wall that have already been outlined.

In the method that elects to solve a near wall turbulent kinetic energy transport equation, the production and dissipation terms in the turbulent kinetic energy transport equation are [potentially] given by (3.213) and

$$-\rho\epsilon = -\rho \frac{C_\mu^{3/4} k^{3/2}}{\kappa Y_p}, \quad (3.217)$$

Unfortunately, there does not seem to be one universal description of the near wall turbulent kinetic energy production term and dissipation term, (3.213) and (3.217), respectively. For example, in the law of the wall formulation, given by Launder and Spalding [40], the near wall production term is given by,

$$P_{kw} = \tau_w \frac{u_\parallel}{y_p}. \quad (3.218)$$

In this formulation, the wall shear stress is given by the law of the wall formulation, (3.203), providing the value of y^+ is greater than 11.63 (otherwise, it is given by the laminar shear stress, (3.204)). The dissipation term, $-\rho\epsilon$ is given by

$$-\rho\epsilon = -\rho \frac{C_\mu^{3/4} k^{3/2}}{\kappa Y_p} \ln E y^+. \quad (3.219)$$

Note that in the absence of convection, diffusion or accumulation, the above two forms of the near wall production and dissipation source terms revert to (3.216). Therefore, if the modeled flow is consistent with the law of the wall formulations, all methods should yield similar limiting behavior. Under conditions of non equilibrium, i.e., a separated flow, or values of y^+ within the viscous sublayer, some models may perform better. However, it is important to note that if the flow to be simulated includes separation and reattachment, or the computation mesh is such that y^+ is within the viscous sublayer, the law of the wall formulation can provide nonsensical results.

In Fuego, there are currently two general supported methods from which to choose when applying the near wall turbulent kinetic energy boundary condition. The first method, which can be activated by the command line `omit near wall turbulent ke transport equation`, is the form of (3.216) that enforces a Dirichlet condition. The second method is to solve a full control volume balance for the near wall turbulent kinetic, with convection and diffusion terms, with a modified production and dissipation term given by either

- Equations (3.213) and (3.214).
- Equations (3.218) and (3.219)

The use of (3.213) and (3.214) can be activated by the command line (within the wall boundary condition block) `use equilibrium production model` which is based on the ability to express the wall shear stress consistent with the assumptions of full equilibrium between production and dissipation, (3.215). In all cases that do not set a Dirichlet condition for the turbulent kinetic energy, the assembled buoyancy source terms are not removed.

The above derivation is applicable to both a RANS and wall-modeled LES paradigm. However, when running an LES-based model, i.e., either the one-equation subgrid scale turbulent kinetic energy or the Smagorinsky model, care must be taken for the interpretation and usage of Fuego’s “turbulent kinetic energy” field. Specifically, in the case of the Smagorinsky model, this variable will have been post-processed and should not be used to compute the wall shear stress, i.e., (3.215) for the momentum system. Rather, a non-linear iteration over the law of the wall equation should be used to compute this quantity, see (3.195). For the one-equation subgrid scale model, it is advised that the wall value should not be assigned a Dirichlet condition based on (3.216) as this derivation represents the kinetic energy and not the subgrid scale kinetic energy. A Neumann-based approach is advised in this case.

$k - \omega$ SST Wall Functions; Turbulent Kinetic Energy

When a Dirichlet condition is not set for turbulent kinetic energy, the approach in modifying the near wall production and dissipation terms is followed.

In this approach, the equation for k is solved near the wall to remove the assumptions of log layer flow one level. However, we invoke the log layer assumption to write,

$$P_k = \frac{\tau_w^2}{\rho \kappa u_\tau Y_p}. \quad (3.220)$$

Balancing production and dissipation in the $k - \omega$ model allows us to write,

$$P_k = \rho \frac{u_\tau^3}{\kappa Y_p} = \rho \frac{(\beta')^{3/4} k^{3/2}}{\kappa Y_p}. \quad (3.221)$$

The dissipation rate is also modified accordingly such that the production equality with dissipation is retained. An alternative method is to use the approximation of of Launder and Spaulding which prescribes production as,

$$P_k = \tau_w \frac{u_{||}}{Y_p}. \quad (3.222)$$

In practice, this formulation seems to be less stable since the production and dissipation terms are now in near-equilibrium.

Wall Functions; Turbulence Dissipation Transport

Consistently within the literature, the near wall turbulence dissipation is assigned the Dirichlet value given by (3.214). Frequently, this expression is lagged by one sub-iteration in an effort to maintain consistency between the Dirichlet wall condition and the freezing of the ϵ/k ratio of the turbulence dissipation equation, (3.119).

Wall Functions; Turbulent Frequency Transport

Low Reynolds Number Treatment

The low Reynolds approach for $k - \omega$ uses a sequence of Dirichlet conditions similar to what is used for $k - \epsilon$. However, unlike the latter, $k - \omega$ requires no extra damping terms near the wall. When the wall is resolved, exact Dirichlet conditions are known for both the velocity and k :

$$\vec{u} = 0, \quad k = 0. \quad (3.223)$$

A Dirichlet condition is also used on ω . While the $k - \epsilon$ model is rendered less stable because k appears in this boundary condition, the ω equation depends only on the near-wall grid spacing. The boundary condition is

$$\omega = \frac{6\nu}{\beta y^2}, \quad (3.224)$$

which is valid for $y^+ < 3$. Above, β depends on the model type. If SST is in use, $\beta = \beta_1$ while if the Wilcox model is in use, $\beta = \beta_0$.

High Reynolds Number Treatment

The high Reynolds approach is also quite similar to the $k - \epsilon$ model except ω is handled differently.

Automatic Wall Functions

Because ω has analytic solutions in both the log layer and viscous sub-layer, an automatic treatment is developed that blends those two solutions to provide Dirichlet conditions for all y . Let ω_h be the high Reynolds number formulation and ω_l be the low Reynolds version. Then the Dirichlet condition on ω is

$$\omega = \omega_l \sqrt{1 + \left(\frac{\omega_h}{\omega_l} \right)^2}. \quad (3.225)$$

However, u_τ for the high Reynolds ω value is computed based on the parallel velocity: The velocity equation is augmented by a traction force based on the friction velocity u_τ . This quantity

may be solved for iteratively using the law of the wall. A Dirichlet condition is also used for k , assuming it is in the log region, which is similar to the $k - \epsilon$ model:

$$k = \frac{u_\tau^2}{\sqrt{\beta^*}}. \quad (3.226)$$

In the case of ω , an analytic expression is known in the log layer:

$$\omega = \frac{u_\tau}{\sqrt{\beta^*} \kappa y}, \quad (3.227)$$

which is independent of k . Note that some implementations use a predefined constant instead of $\sqrt{\beta^*}$, although the standard values are consistent with these expressions. Because all these expressions require y to be in the log layer, they should absolutely not be used unless it can be guaranteed that $y^+ > 10$, and $y^+ > 25$ is preferable.

$$u_\tau = \sqrt{\nu \left| \frac{u_{||}}{y} \right|}. \quad (3.228)$$

The automatic wall function approach is obtained by removing the `omit near wall turbulent ke` equation line command and activating either the SST or KW turbulence models.

Wall Functions; Enthalpy Transport

For non-adiabatic boundaries, heat loss to the wall must be considered. The use of the Reynolds analogy provides a functional form of the energy transport similar to the that of the logarithmic law-of-the-wall momentum formulation. The thermal boundary layer is modeled either as a linear profile ($y^+ < 11.63$) where the thermal boundary layer is dominated by conduction or a logarithmic profile where the effects of turbulence dominate over thermal conduction, Versteeg and Malalasekera [41].

The law-of-the-wall used in Fuego has the following form,

$$q_w = \frac{\rho C_p (T_w - T_p) u_\tau}{T^+}, \quad (3.229)$$

where

$$T^+ = \sigma_T [u^+ + P], \quad (3.230)$$

and T_p is the sampled temperature. This sampling is either based on the local temperature obtained from the boundary face or, for cases where the exchange-based model is activated, the interpolated temperature at the location of the exchange.

The role of T^+ is to account for the fact that the thickness of the thermal conduction layer is [practically] of a different size than that of the viscous sublayer (momentum).

In the above equation, P is the universal “P function” (Jayatilleke [42]) and can be expressed as a function of the molecular and turbulent Prandtl number,

$$P = 9.24 \left[\left(\frac{\sigma}{\sigma_T} \right)^{0.75} - 1 \right] \left(1 + 0.28 \exp \left[-0.007 \frac{\sigma}{\sigma_T} \right] \right), \quad (3.231)$$

where σ_T and σ represent the turbulent and molecular Prandtl number, respectively.

Therefore, it is seen that the so-called “P function” is the parameter that functionally changes the thickness of the thermal conduction layer from that of the viscous sublayer. For example, if one were to model a high-Prandtl number fluid such as common vegetable oil, one would note that the thickness of the viscous sublayer is far greater than that of the thermal sublayer. However, for low-Prandtl number fluids, the opposite is true. The subsequent value of T^+ ensures this functionality.

In the case of a user defined heat flux at a wall boundary condition, the full quantity is assembled as a right-hand-side source term. As a post processing step, (3.229) (in temperature form) is rearranged to provide the wall temperature. In practice, the heat flux boundary condition block is to be defined on an already defined wall boundary condition block (without temperature specification). In this manner, multiple boundary conditions are “painted” on a particular sideset.

Wall Functions; Scalar Transport

Wall functions for use in a convective diffusive problem, e.g., diffusional transport of fuel (through multicomponent evaporation) from a jet fuel pool, are not currently supported.

Inlet Conditions for Turbulence Quantities

Turbulent Kinetic Energy

The inlet turbulent kinetic energy must be specified for any simulation that involves a velocity-specified inlet. If actual values of the inlet turbulent kinetic energy are not available, then a suitable value based on basic definitions is used. In general, the kinetic energy associated with the turbulent flow is defined by,

$$k = \frac{1}{2} \left(\overline{u'^2} + \overline{v'^2} + \overline{w'^2} \right). \quad (3.232)$$

The turbulence intensity T_i , is related to the kinetic energy by,

$$T_i = \frac{\left(\frac{2}{3} k \right)^{1/2}}{U_{ref}}. \quad (3.233)$$

Rearranging (3.233) for the turbulent kinetic energy yields a working form for the specification of inlet turbulent kinetic energy based on a reference velocity, U_{ref} ,

$$k = \frac{3}{2} (U_{ref} T_{in})^2. \quad (3.234)$$

The value of U_{ref} can typically be taken to be the magnitude of the velocity.

Turbulence Dissipation Rate

As with the turbulent kinetic energy inlet condition for specified velocity, the inlet value of the turbulence dissipation rate must also be specified. If values are known, for instance based on experimental data, then the available data should be used. Otherwise, the following assumed form of the turbulence dissipation rate is used,

$$\epsilon = C_\mu^{3/4} \frac{k^{3/2}}{l}, \quad (3.235)$$

where $l = 0.07L$; L represents a characteristic length scale of an inlet eddy and k represents the inlet turbulent kinetic energy as determined above.

Mass Injection Boundary Condition

Incorporation of boundary layer development over an inflow surface can be achieved by the usage of a “mass injection” surface definition. For example, a quiescent pool fire notes entrainment over the pool surface due to vertical acceleration of the fire. Moreover, in crosswind applications that involve the release of a traceable species, a developing boundary layer interacts with the inlet. Fuego supports the ability to combine an inflow boundary that, in some cases, includes the presence of a wall. As will be discussed below, managing a “blowing” boundary condition from a wall is supported by the mass injection modeling approach. This boundary condition also provides the foundational implementation for the complementary fuel boundary condition that determines the mass flux of fuel given an incident heat loading. For more information regarding the pool boundary condition, please refer to *Fuel Boundary Condition Submodel*.

To begin the discussion of the mass injection boundary condition, Figure 3.2 outlines a typical exposed surface control volume for CVFEM. In this image, the near wall control volume is shaded with the degree-of-freedom (DOF) noted by the solid circular symbol. Triangles and diamonds depict surface- and volume-based integration points. The solid circle degree-of-freedom includes two elements, while the opposing node includes an inner element patch that is void of boundary contributions. The control volume shaded includes two boundary integration points. For a typical inflow, the known state entering the domain can be enforced either via a strong-form, i.e., Dirichlet, or a weak-form where advection and diffusion contributions are assembled to the control volume balance. In fact, the control volume approach allows for the natural integration-by-parts form for advection and diffusion. When using a Dirichlet approach, the specified value that is provided by the user prevails for the nodal DOF, and all other surface and volume contributions that may have been assembled during the iteration over elements are neglected.

The mass injection formulation provides the advective contribution, i.e., $\dot{m}\phi^{spec}$, where \dot{m} is the mass flow rate and ϕ^{spec} is the specified value that enters the domain, to the boundary node control volume. In the formulation, the mass flow rate is determined by the user-specified mass flux (units *mass/area/time*) and the integration point area. In Figure 3.2 two integration point contributions are made to the equation system. A diffusive contribution may also be added to the transport equation, if appropriate. For example, the momentum equation may compute a wall shear stress through a turbulent wall model and will have both an advective and diffusive contribution. Since the mass injection boundary condition is a mixture between an inflow and wall boundary condition, in some cases a Dirichlet condition due to the wall model specification parameters may prevail.

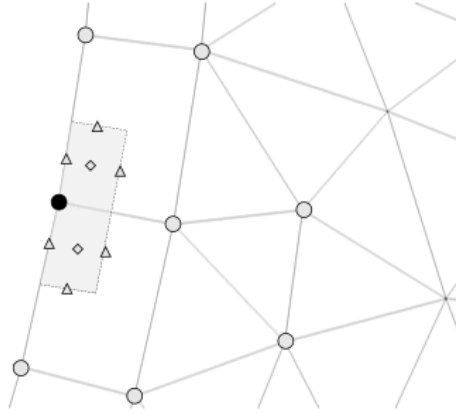


Fig. 3.2: Control volume definition for an exposed surface. The degree of freedom is noted by the dark circle; shading of the control volume is also indicated. Triangles and diamonds depict surface- and volume-based integration point locations.

A theoretical contributions due to the mass injection boundary condition are provided in the enumerated list below, while a condensed table that compares and contrasts the standard wall and mass injection boundary condition is presented in Table 3.5 and Table 3.6.

1. The continuity equation, which solves for the pressure DOF, assembles the computed integration point mass flow rate based on the user-specified mass flux and the integration point area.
2. The momentum equation, which solves for the velocity DOF, assembles $\dot{m}u_i$. In this expression, u_i is computed based on $\dot{m}/(\rho|A|)n_i$, where $|A|$ is the integration point area magnitude and n_i is the unit normal area component at the boundary integration point. The density used in this expression is based on user-specified conditions and follows the underlying material model evaluation. For example, if a mass injection temperature is specified, and the user activated an ideal gas law (uniform species), then density is computed based on the reference molecular weight, reference pressure, and wall injection temperature. The integrated viscous stress contribution is determined by the wall modeling approach. For a laminar-based configuration, the viscous stress tensor is integrated over the boundary surface and assembled to the wall-node equation system. For turbulent applications, the chosen turbulence modeling choices outlined in [Wall Functions](#);

Momentum dictate the wall shear stress closure. The effect of blowing on the wall friction velocity by default is assumed to be small and the standard wall boundary approach for the calculation of this variable is followed.

3. The transport equations including mixture fraction, mass fraction, soot-model transported equations, and user-defined progress variables assemble $\dot{m}\phi^{pec}$ to wall-node equation system, where ϕ^{sec} is the user-specified value. A zero diffusional flux is implied.
4. The enthalpy and conserved enthalpy equation set assembles $\dot{m}h^{MI}$ to wall-node equation system, where h^{MI} is the derived enthalpy that is computed consistently with the property model and injection temperature specification. For example, h^{MI} may be computed based on the injection temperature and species that the user specifies through a Cantera material property interface. For the diffusional contribution, the enthalpy supports a weak flux boundary condition for modeled turbulent wall treatments, and omits any contribution when a resolved laminar model set is active.
5. The turbulent kinetic energy equation, which solves for the modeled kinetic energy, assumes by default that the blowing contribution is small and is, therefore, neglected. The wall modeling approach outlined in *Wall Functions; Momentum* dictates the near-wall form for this DOF. For resolved modeling approaches, the user-specified value is used to enforce a Dirichlet condition. Models that require a normal distance to a wall include the mass injection surface in the determination of this variable. For the non-resolved model suite (in absence of a Dirichlet condition specification by the user), zero diffusion across the boundary surface is assumed.
6. The turbulence dissipation equation, which solves for the modeled turbulence dissipation, assumes by default that the blowing contribution is small and is, therefore, neglected. Like the turbulent kinetic energy and momentum equation, the wall modeling approach outlined *Wall Functions; Momentum* dictates the near-wall form for this DOF.
7. The elliptic relaxation equation, which is part of the $v^2 - f$ model suite, applies a user-defined Dirichlet value at the wall.
8. The velocity-square equation, \bar{v}^2 , which is part of the $v^2 - f$ model suite, also assumes by default that the blowing contribution is small and is, therefore, neglected. A weak boundary condition applying zero diffusion is implied.

Table 3.5: Wall details for supported equation sets.

Equation	Wall	
	Advection	Diffusion
Continuity	Weak	n/a
Momentum	Zero	Strong (if wall-resolved), weak otherwise
Mixture-fraction	Zero	Weak (zero flux)
Mass-fraction	Zero	Weak (zero flux)
Soot-based	Zero	Weak (zero flux)
Progress variables	Zero	Weak (zero flux)
Enthalpy	Zero	Weak (if turbulent), Strong (if wall-resolved)
Conserved Enthalpy	Zero	Weak (zero-flux)
Turbulent KE	Zero	Zero (Weak or Strong)
Turbulence dissipation	Zero	Zero (Strong)
Elliptic Relaxation	Zero	Zero (Strong)
Velocity-square	Zero	Zero (Strong)

Table 3.6: Mass injection details for supported equation sets.

Equation	Mass Injection	
	Advection	Diffusion
Continuity	Weak	n/a
Momentum	Weak, $\dot{m}u_i$	Weak
Mixture-fraction	$\dot{m}\phi^{spec}$	Weak (zero flux)
Mass-fraction	$\dot{m}\phi^{spec}$	Weak (zero flux)
Soot-based	$\dot{m}\phi^{spec}$	Weak (zero flux)
Progress variables	$\dot{m}\phi^{spec}$	Weak (zero flux)
Enthalpy	Weak, $\dot{m}h^{MI}$	Weak (if wall temperature is specified)
Conserved Enthalpy	Weak, $\dot{m}h^{MI}$	Weak (zero-flux)
Turbulent KE	Zero	Weak of Strong
Turbulence dissipation	Zero	Weak or Strong
Elliptic Relaxation	Zero	Zero (Strong)
Velocity-square	Zero	Zero (Strong)

Mass Blowing Correction for Turbulent Applications

In cases where mass injection is sufficiently high, the wall friction velocity must be modified. Fuego supports the following modified law of the wall form in such cases,

$$u^+ = \frac{u_{||}}{u_\tau} = \frac{1}{\kappa} \ln(Ey^+) + Kv^+ \quad (3.236)$$

The model constant K is generally approximately ten, while the normalized blowing velocity is defined by $\frac{v_\perp}{u_\tau}$. For more information, the reader is referred to [43]. Note that model constant E

appearing above is generally modified to 8.166 for blowing walls and that stability of the nonlinear solve is reduced at high values of v^+ .

EDC Turbulent Combustion Model

The combustion submodel is Magnussen's Eddy Dissipation Concept (EDC) and development details can be found in Magnussen, et al. [44], Magnussen [45], Byggstøl and Magnussen [46], Magnussen [47], Lilleheie, et al. [48], and Gran and Magnussen [49].

Model Characteristics

The underlying assumption in the EDC model is that combustion in turbulent flows is controlled by turbulent mixing. The combustion model is an algebraic zone-type model and is influenced by local cell (control volume) values only. The model derivation assumes that the minimum cell dimension is large relative to the thickness of a flame (reaction zone) structure. This thickness varies with strain-rate, but the cell size should not be less than a few millimeters. The equations are not valid for laminar or near-laminar flow, but are based on fully developed turbulence arguments. The turbulent combustion model uses information from three sources: 1) thermochemistry, 2) species and state information from the cell values, and 3) turbulence kinetic energy and dissipation. From these data, the model creates source/sink terms for species equations and the energy equation (via radiative transport).

The model function is to provide an integral effect of combustion processes occurring within the control volume for the duration of a time-step. In this manner, reaction zone structures are not resolved, but the aggregate effect of turbulent combustion is modeled. To model the integral effect, two homogeneous zones are defined within each control volume for which there is combustion, as shown in Figure 3.3. The zones are termed the reaction zone (fine structures) and the surrounding zone. The size and mass exchange rate between these zones are influenced by the local turbulence properties and are the principal means by which turbulent fluctuations are accounted for within the model. The assumption that each zone is homogeneous is equivalent to assuming that the mixing within each zone is instantaneous. Since combustion occurs within (but is not limited to) the reaction zone, the assumptions for combustion correspond to those for a perfectly stirred reactor (PSR). Slower reactions can also occur in the surroundings, in which case, the assumptions for reaction in the surroundings are also consistent with PSR assumptions.

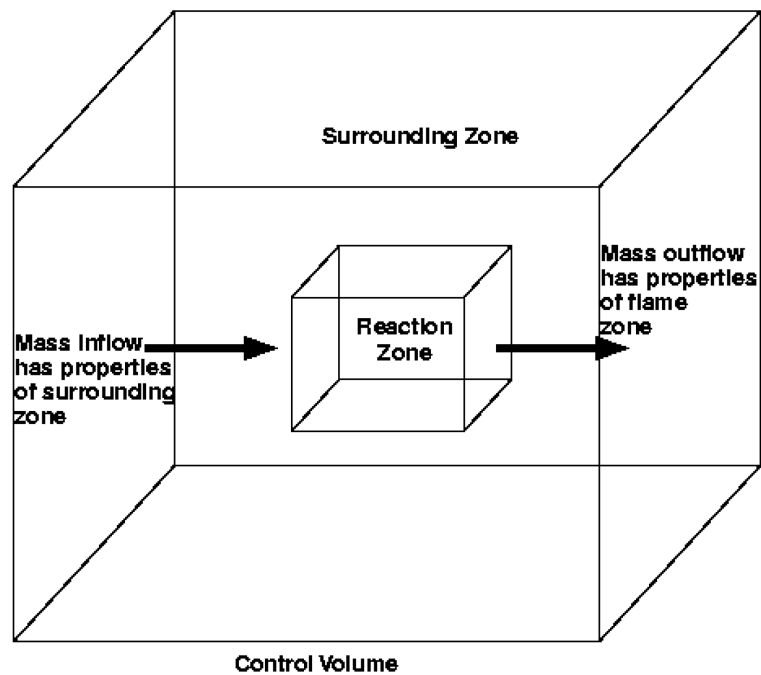


Fig. 3.3: Model geometry for Magnussen's Eddy Dissipation Concept. The control volume is comprised of two zones; the properties of each zone are assumed to be adequately represented by a single set of values (i.e., lumped or perfectly stirred). The mass exchange between the zones is controlled by turbulent mixing.

Physical Interpretation

Magnussen's EDC model is derived to be a general combustion model for premixed to non-premixed scalar fields and for high to moderate turbulence levels. It is not intended to be used for laminar combustion. Magnussen's physical interpretation of combustion is based on the concept that chemical reaction occurs in regions of the flow in which the dissipation of turbulent energy takes place, i.e., fine structure regions. These regions are concentrated in isolated volumes and represent a small fraction of the flow. The regions have characteristic dimensions that are of the Kolmogorov length scale in one or two dimensions, but not the third.

Fires are buoyant flows. Turbulent fires tend to be large, having base diameters above a meter. The turbulent length scales are large and the flow velocities are relatively slow, on the order of meters to tens of meters per second. (Still photographs of reaction zone structure within large fires can be found in Tieszen, et al. [50]). Therefore, turbulence levels tend to be moderate. Near the base of a fire, the combustion zone can be characterized as a continuous wrinkled flame sheet that appears to wrap around larger turbulent structures. The basic combustion mode is that of a strained diffusion flame with large surface area due to the turbulence. At higher elevations in the fire, turbulence levels increase and the character may change. Premixed combustion is possible as unburned products in the smoke are re-entrained into the fire. While Magnussen's model was originally derived in terms of high turbulence levels resulting in fine structure regions (i.e., localized regions of high vorticity at dissipation scales), the model is appropriate for moderate turbulent intensities that occur in fires.

Figure 3.4 shows the physical geometry from which the combustion model will be derived for fires. Turbulence controls the reaction and surrounding volume fractions and fuel mass transport per unit volume. In general, turbulent momentum exchange processes result in scalar stirring at all length scales down to molecular mixing processes which are diffusion controlled. Without length scale information below the grid scale of the computation, it is impossible to correctly represent the interactions between all the relevant physical processes at their relevant length scales.

Magnussen's EDC model attempts to represent the mixing processes that are most important to the overall heat release from combustion. It is based on the assumption that the overall heat release rate is controlled by the mass transport into the reaction zone. Therefore, considerable effort is made to model turbulent momentum processes that affect mass transport into the reaction zone. In the surrounding gases, turbulent mixing occurs with (in all likelihood) a similar vigor, however, its effect on the combustion rate is considered less important since the turbulence is not directly contributing to mass transport into the reaction zone. For this reason, there are two different levels of mixing assumptions made within the model.

With respect to Figure 3.3, the turbulence level in each control volume is taken into account in the consideration of the mass exchange between the reaction zone and the surrounding zone. However, within each zone, it is assumed that the properties are instantaneously homogeneous and uniform, i.e., perfectly stirred. This perfectly stirred assumption obviously over-predicts mixing within each zone for any real level of turbulence, and only begins to approximate reality at the highest levels of turbulence. On the other hand, the perfectly stirred assumption allows point calculations to be made in each zone for conveniently determining thermochemical properties.

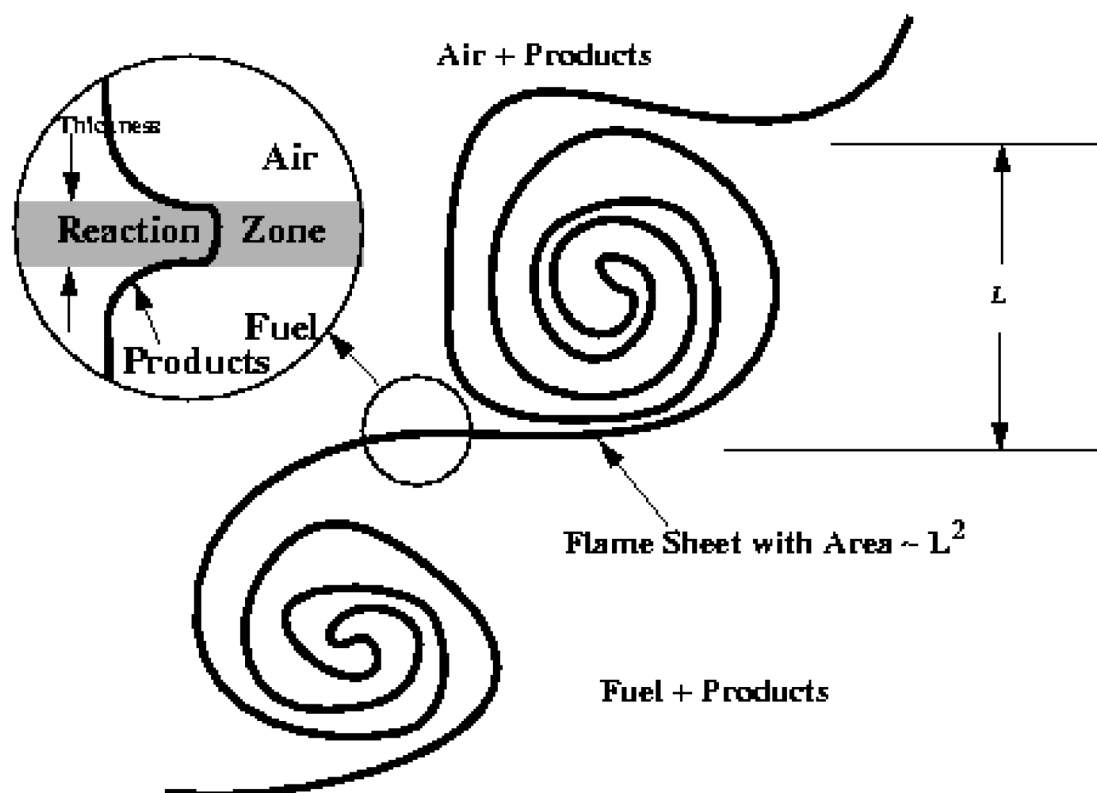


Fig. 3.4: Assumed flame surface geometry. L is the integral turbulent length scale. The reaction zone thickness is characterized by the Kolmogorov dissipative turbulent length scale, η .

Without this assumption, it would be necessary to specify the gradients within each zone and integrate the specified gradients throughout the cell to obtain cell averaged property information. The approach here is to assume that over-predicting mixing within each zone via the perfectly stirred assumption has only a secondary effect on heat release rates within each cell.

Thermochemistry

Within the current strategy, chemical reaction can occur in both zones. However, in the simplest case, no reaction occurs within the surroundings due to the low temperature and unmixedness; all reaction occurs within the reaction zone. The notion of zones, perfect stirring within the zones, and type of chemistry involved are all independent assumptions, but have interrelated consequences. For example, finite-rate chemistry involving hundreds or thousands of species could be considered within the zones. From the perfectly stirred assumption within each zone, the finite-rate chemistry would be calculated as if it were occurring in a perfectly stirred reactor. In a real diffusion reaction, there are spatial variations in species concentrations for real turbulence levels so that the various chemical pathways, as well as heat, mass, and momentum transport, in a real strained diffusion flame can be quantitatively different than those calculated on the basis of perfect stirring. This effect is probably the strongest disadvantage of the perfectly stirred assumption. Only in the limit of infinitely-fast turbulent mixing does perfect stirring actually exist. In practice, the computation of detailed, finite-rate chemistry concurrently with a three-dimensional fluid mechanics calculation is expensive. Except in the limit where the turbulent strain rate is high enough that finite rate chemistry is warranted, it is adequate to use simpler descriptions of the chemistry. In the case of high strain rates, precalculation of the chemistry is usually done and the results tabulated in a look-up table to determine extinction limits.

For the current implementation, it is assumed that the chemistry can be represented as irreversible, “infinitely-fast” reactions that occur within each reactor. In classical combustion studies, the concept of “infinitely-fast” reactions is not usually invoked in the context of a perfectly stirred reactor. In the context of the current model, the meaning of an “infinitely-fast” reaction in the flame zone (a perfectly stirred reactor) is that the reactant stream entering the reaction zone is converted to products instantly as it enters the zone, and then the products are mixed instantly throughout the zone. The zone then reflects the thermodynamic properties of the combustion products at the adiabatic flame temperature for a given composition while the surrounding zone has the properties of reactants (and possibly previously combusted products) near the cell temperature.

In general, if the turbulent mass exchange rate between the zones (i.e. strain-rate) is sufficiently high that infinitely-fast chemistry assumptions do not apply, then finite-rate reactions within the perfectly stirred reactor can be used. Residence time scales that warrant finite-rate considerations tend to be at the sub-millisecond level. In the current implementation, the case of high turbulence levels leading to blow-out of a reactor is treated as a limits test. The test method is discussed in *Limits Testing*.

In principle, it is not necessary to assume irreversible chemistry within each zone. At long time scales (i.e., low turbulence levels), chemical equilibrium will result. The use of irreversible

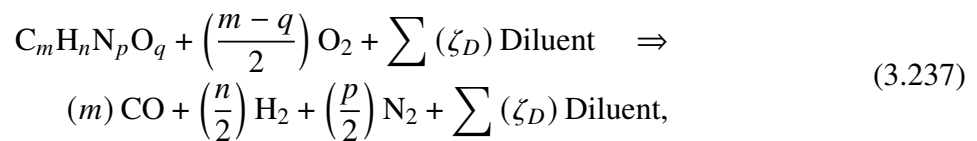
chemistry avoids the need to calculate the equilibrium state of the forward and reverse reactions for every combusting cell at every time step. For the current implementation, the time savings is deemed to be worth the cost in accuracy.

Regardless of the assumptions about chemistry employed in modeling the reaction zone, the actual reaction zones in a fire will very likely be similar to strained diffusion flames (wrinkled flame sheets wrapped into vortical structures). Perhaps higher in a fire with the re-entrainment of smoke, partially premixed combustion can occur. For diffusion reactions, combustion occurs within a region encompassing a stoichiometric surface between fuel and air. Therefore, the reaction zone is modeled as occurring with stoichiometric reactions. The reactants being transported into the reaction zone via turbulent mixing come from the surroundings zone and thus have the composition of the surroundings. There will be a limiting amount of one reactant if the combustion is to occur at off-stoichiometric conditions. The excess of the other reactant, prior products, and inerts do not participate in chemical reactions, but are transported in and out of the combustion zone by turbulent mixing. However, their presence affects the zone properties (for example, through their heat capacity).

Combustion products are transported into the surroundings at the same rate as the reactants are transported into the reaction zone (conservation of mass). However, the perfect stirring assumption for properties means that these products have uniform properties. In a diffusion reaction, products mix with fuel on one side of the reaction zone and air on the other. On the fuel side of the reaction zone, significant amounts of CO and soot can result from interaction between the inflowing fuel and outflowing products. The formation of CO is important not only from a toxic pollutant perspective but its formation results in significantly less heat release and lower temperatures. Given the limits of a two-zone model with perfect mixing within each zone, there is no simple way to model both stoichiometric combustion and the formation of CO on the fuel side of the reaction. In the current formulation, an *ad-hoc* approach is used in which combustion in the reaction zone is assumed to occur in sequential steps, each of which is irreversible and infinitely fast. The first step is stoichiometric oxidation of the fuel species to CO and H₂ products. The second step is the oxidation of CO and H₂ to CO₂ and H₂O provided there is excess O₂ in the reactant stream. If the overall stoichiometry in the control volume is fuel rich, significant amounts of CO and H₂ will be formed, while if it is lean only CO₂ and H₂O will be formed.

Chemical Mechanism

For an arbitrary CHNO fuel, the stoichiometric, irreversible reaction to CO and H₂ products is given by

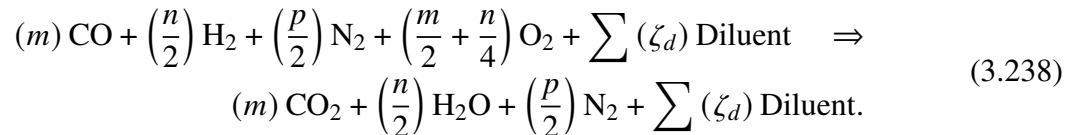


where m , n , p , and q are the numbers of carbon, hydrogen, nitrogen, and oxygen atoms within the fuel molecule, respectively, and the terms in parentheses are the stoichiometric coefficients.

The summation term for diluents includes all other species present in the reaction stream including nitrogen in air, combustion products in the surroundings from previous combustion processes, etc. . . Diluents, including the combustion products, are assumed to have no effect on the chemical reaction itself. However, diluents do have an effect on the temperature rise through their specific heats and the presence of products is used as an ignition criteria for the combustion model.

The assumption that combustion products act like diluents (i.e., have no effect on the reaction) is obviously a simplification. Product species include CO, H₂, CO₂, and H₂O. The presence of CO and H₂ in the reactant stream would affect equilibrium results; however, irreversible reactions have already been assumed in the model so the presence of these species does not represent an additional simplification. On the other hand, the presence of large amounts of CO₂ and H₂O in the reactant stream may reduce the amount of O₂ consumed for a given amount of fuel due to partial oxidation of the products via the oxygen in the CO₂ and H₂O in an overall fuel rich environment. However, this effect is partially compensated since the extra O₂ would be consumed by the second reaction.

The second reaction is the subsequent oxidation of CO and H₂ to CO₂ and H₂O. This reaction oxidizes both the CO and H₂ produced by the first reaction and any CO and H₂ that passed through the first reaction as products (i.e., diluent). The reaction is given by



In the current implementation, soot is considered to be a trace species. As such, its mass and energetics are not considered part of the above chemical reactions. Soot has its own production terms and is considered to oxidize in proportion to the fuel oxidation in the first reaction. See the soot model in *Soot Generation Model for Multicomponent Combustion* for details.

Species Consumption/Production Limits

The reactants being transported into the reaction zone come from the surroundings and therefore have the same composition as the surroundings. As such, the reaction can only proceed within the limits of available fuel and oxygen from the reactant stream. For example, if there is insufficient oxygen in the reactant stream, then all of the oxygen will be consumed by Reaction 1, ((3.237)), and the excess fuel will be passed with products from Reaction 1 to Reaction 2, ((3.238)).

Reaction 2 will not take place because all the oxygen was consumed in Reaction 1 (i.e., in both reactions, oxygen is limiting). If there is insufficient fuel in Reaction 1, then all the fuel will be consumed and excess oxygen will be passed to Reaction 2. Depending on the ratio of oxygen to CO and H₂, all the secondary fuels may be consumed or all the oxygen may be consumed.

To find the limiting mass, it is convenient to define an equivalence ratio. Equivalence ratios are normally defined in terms of molar ratios, but mass ratios yield the same result [51] and are

preferred here since mass fractions are used in the transport equations.

$$\Phi = \frac{\left(\frac{Y_{fuel}}{Y_{oxy}} \right)_{mix}}{\left(\frac{Y_{fuel}}{Y_{oxy}} \right)_{stoic}} \quad (3.239)$$

The numerator is the ratio of the actual mass of fuel to oxygen in the reactant stream,

$$\left(\frac{Y_{fuel}}{Y_{oxy}} \right)_{mix} = \frac{\text{mass Fuel}}{\text{mass Oxygen}} \Big|_{mix}. \quad (3.240)$$

The denominator is determined for each reaction. Generically, the first and second reactions have the following form

$$\sum (\zeta_{fuel}) \text{Fuel} + \zeta_{O_2} O_2 + \sum (\zeta_D) \text{Diluent} \rightarrow \sum (\zeta_{prod}) \text{Product} + \sum (\zeta_D) \text{Diluent}, \quad (3.241)$$

where ζ are stoichiometric coefficients on a molar basis. The stoichiometric fuel to oxygen mass ratio is

$$\left. \frac{Y_{fuel}}{Y_{oxy}} \right|_{stoic} = \frac{\sum W_{fuel} (\zeta_{fuel})}{W_{O_2} (\zeta_{O_2})}, \quad (3.242)$$

where W is a molecular weight. Specifically for the first reaction, the stoichiometric mass ratio of $C_m H_n N_p O_q$ to O_2 is

$$\left. \frac{Y_{fuel}}{Y_{oxy}} \right|_{stoic} = \frac{(12m + n + 14p + 16q)}{32 \left(\frac{m - q}{2} \right)}. \quad (3.243)$$

Therefore, the equivalence ratio for the first reaction which is based on carbon monoxide and hydrogen products is given by

$$\Phi_1 = \left(\frac{Y_{fuel}}{Y_{oxy}} \right) \frac{16(m - q)}{(12m + n + 14p + 16q)}, \quad (3.244)$$

and similarly, the equivalence ratio for the second reaction which is based on carbon dioxide and water products is given by

$$\Phi_2 = \left(\frac{Y_{co} + Y_{h2}}{Y_{oxy}} \right) \frac{16 \left(m + \frac{n}{2} \right)}{(28m + n)}. \quad (3.245)$$

If either equivalence ratio is greater than unity, then the mass of oxygen will be completely consumed by its reaction. If either equivalence ratio is less than unity, then the mass of fuel will be completely consumed by its reaction. If either equivalence ratio is unity, then the mass of fuel and oxygen will both be completely consumed by that reaction. Note that $C_m H_n N_p O_q$ is not a fuel in the second reaction because if there is any of this fuel left, all the oxygen was consumed in the first reaction. Therefore, under these conditions the second reaction cannot proceed due to lack of

oxygen. Also note that the expression for Φ_2 does not identify which secondary fuel, CO or H₂, is limiting.

In order to determine the limiting reactant mass in a multi-fuel (or multi-oxidant) system, a more general approach based on equivalence ratios is required. Consider the reaction

$\zeta_A A + \zeta_B B \rightarrow \zeta_C C + \zeta_D D$ where ζ are stoichiometric coefficients. The stoichiometric mass ratio of reactant B to A is

$$\left. \frac{Y_B}{Y_A} \right|_{stoic} = \left. \frac{\text{mass}_B}{\text{mass}_A} \right|_{stoic} = \frac{W_B \zeta_B}{W_A \zeta_A}. \quad (3.246)$$

Further, Y_A and Y_B are the mass fractions of A and B in the mixture and

$$\frac{Y_B}{Y_A} = \left. \frac{\text{mass } B}{\text{mass } A} \right|_{mix}. \quad (3.247)$$

The ratio of these quantities is an equivalence ratio; i.e., if

$$\frac{Y_B}{Y_A} > \frac{W_B \zeta_B}{W_A \zeta_A}, \quad (3.248)$$

then A is the limiting reactant, else B is the limiting reactant. However, this inequality can be usefully rearranged. If

$$\frac{Y_A}{W_A \zeta_A} < \frac{Y_B}{W_B \zeta_B}, \quad (3.249)$$

then A is the limiting reactant. The same procedure can be shown to apply to reactions where there are more than two reactants; i.e., if

$$\frac{Y_A}{W_A \zeta_A} < \frac{Y_B}{W_B \zeta_B} < \dots < \frac{Y_n}{W_n \zeta_n}, \quad (3.250)$$

then A is the limiting reactant of n reactants. Therefore,

$$\text{First Reactant Depleted} = \min_n \left(\frac{Y_n}{W_n \zeta_n} \right). \quad (3.251)$$

Note that the units of $Y_n/W_n \zeta_n$ are $[(\text{mass } n)_{mix}/(\text{mass } n)_{stoic}]/(\text{mass})_{mix}$. Also note that diluents are not reactants and they are not depleted by the reaction. The $\min()$ function should only be applied to fuels and oxygen, not to all species.

To determine the change in mass fraction, ΔY_k^m , of reactant species k due to reaction m , multiply the limiting mass expression by the stoichiometric mass of species k :

$$\Delta Y_k^m = -W_k \zeta_k^m \min_n \left(\frac{Y_n}{W_n \zeta_n} \right)^m. \quad (3.252)$$

This expression has units of $[(\text{mass } k)_{stoic}/(\text{mass } n)_{stoic}] \times [(\text{mass } n)_{mix}/(\text{mass})_{mix}]$. Since n is the limiting reactant, the expression within the second set of square brackets is the change in mass fraction of species n due to reaction m ; this is because the limiting species n is completely used up

in the reaction (i.e., the mass fraction of species n goes to zero). The expression within the first set of square brackets modifies the change in mass fraction of species n to yield the change in mass fraction of species k due to reaction m . The change in mass fraction of product species k in reaction m is similar but without the minus sign in the above expression.

Since the reactions are given priority, the “products” of Reaction 1 are the “reactants” of Reaction 2. The new mass fractions in the reactant stream for Reaction 2 are given by

$$(Y_k)_{\text{Reaction 2 reactants}} = (Y_k)_{\text{surr}} \pm \Delta Y_k^{\text{Reaction 1}}. \quad (3.253)$$

As noted above, the sign of the second term, $\pm \Delta Y_k^{\text{Reaction 1}}$, is positive for products and negative for reactants. Similarly, the product composition from Reaction 2 is given by

$$(Y_k)_{\text{Reaction 2 products}} = (Y_k)_{\text{Reaction 2 reactants}} \pm \Delta Y_k^{\text{Reaction 2}}. \quad (3.254)$$

Here again the positive sign on the second term is used for products and negative sign is used for reactants. Since the reactions are assumed to occur infinitely fast, the product composition for Reaction 2 is the composition of the reaction zone,

$$(Y_k)_{\text{flame}} = (Y_k)_{\text{Reaction 2 products}}. \quad (3.255)$$

Conservation Laws

For convenience we restate the Favre-averaged species mass conservation equation, (3.106),

$$\int \frac{\partial \bar{\rho} \tilde{Y}_k}{\partial t} dV + \int \bar{\rho} \tilde{Y}_k \tilde{u}_j n_j dS = \int \left(\frac{\mu}{Sc} + \frac{\mu_t}{Sc_t} \right) \frac{\partial \tilde{Y}_k}{\partial x_j} n_j dS + \int \bar{\omega}_k dV, \quad (3.256)$$

where $\bar{\rho}$ is the time averaged density of the mixture, \tilde{Y}_k is the Favre-averaged mass fraction of species k , \tilde{u}_i is the Favre-averaged velocity of the mixture, μ_t is the turbulent eddy viscosity, Sc_t is the turbulent Schmidt number, and $\bar{\omega}_k$ is the time-averaged mass production rate of species k per unit volume of the mixture. This equation is solved on a mesh, one control volume of which is shown in Figure 3.3. Within the control volume, the species k mass consumption/production rate, $\dot{m}_{k,consumed/produced} = \bar{\omega}_k V_{cell}$, is determined by the EDC model, assuming that the mass transfer process into and out of the reaction zone from the surroundings (cf. Figure 3.3) can be represented as a steady process,

$$(\dot{m}_k)_{consumed/produced} = (\dot{m}_k)_{flame} - (\dot{m}_k)_{surr}. \quad (3.257)$$

The mixture mass flow rate between the surroundings and the reaction zone is also assumed to be steady,

$$(\dot{m})_{flame} = (\dot{m})_{surr}. \quad (3.258)$$

Combining these two expressions yields

$$\begin{aligned} (\dot{m}_k)_{consumed/produced} &= \left[\frac{(\dot{m}_k)_{flame}}{(\dot{m})_{flame}} - \frac{(\dot{m}_k)_{surr}}{(\dot{m})_{surr}} \right] (\dot{m})_{flame} \\ &= [(Y_k)_{flame} - (Y_k)_{surr}] (\dot{m})_{flame}. \end{aligned} \quad (3.259)$$

It is convenient to normalize this equation with the mass of the control volume, or

$$\frac{(\dot{m}_k)_{consumed/produced}}{M_{cell}} = [(Y_k)_{flame} - (Y_k)_{surr}] \frac{(\dot{m})_{flame}}{M_{cell}}. \quad (3.260)$$

The term in the brackets is a function of thermochemistry only and is specified by the chemical processes derived in the previous section. The second term, the normalized mass transfer rate, is a function of the turbulent mass exchange rate between the reaction zone and its surroundings. The derivation of this term is the subject of the next subsection.

Effect Of Turbulence On Combustion Rates

Magnussen derived the effect of turbulence on combustion rates in terms of high turbulence levels. The derivation here will be for moderate turbulence levels for the flame geometry shown in [Figure 3.4](#). The derivation herein does not include proportionality constants. Rather, dimensional reasoning is used to establish the relationship between reaction zone surface area, volume, and mass transfer rates with respect to the prevailing turbulence levels. Constants of proportionality, taken from Magnussen's original derivation, are added at the end.

Characteristic scales are needed for the mass transfer velocity into the reaction zone, the reaction zone surface area, and the reaction zone thickness. The mass transfer velocity into the reaction zone is a velocity appropriate to diffusional length scales that are being modified by the local strain field induced by the turbulent flow,

$$\text{Mass Transfer Velocity} \propto \nu. \quad (3.261)$$

An appropriate diffusional velocity is the Kolmogorov velocity, ν , which is characteristic of dissipative length scales (i.e., those in which the local strain field is being dissipated by diffusional effects). From Kolmogorov's definition, ν is given by

$$\nu \equiv (\nu\epsilon)^{1/4}, \quad (3.262)$$

where ν is the molecular mixture kinematic viscosity (evaluated at the surrounding temperature), and ϵ is the rate of kinetic energy dissipation.

The reaction zone is characterized as a continuous flame sheet, highly wrinkled and wrapped around large eddies. The volume of a large eddy is characterized by

$$\text{Volume}_{eddy} \propto L^3, \quad (3.263)$$

where L is the characteristic integral length scale of the turbulence. The reaction zone area is assumed to be proportional to both momentum and scalar influences. While all length scales of the turbulent cascade contribute to wrinkling and stretching the flame, it is assumed that large changes in surface area are associated with large length-scale fluctuations. Therefore, it is assumed that the square of the integral length scale is the most appropriate turbulent length scale for characterizing the reaction zone area.

Species concentrations also affect reaction zone area. Obviously, if no fuel is present, no reaction zone will be present regardless of level of turbulence present. The species influence are denoted by a function, χ , the rationale of which will be described later. Based on these arguments,

$$\text{Area}_{flame} \propto \chi L^2. \quad (3.264)$$

To obtain property values for each zone in [Figure 3.3](#), it is necessary to define the volume fractions of the reaction zone and surrounding zones. The reaction zone volume fraction is based on a reaction zone area and a reaction zone thickness. Since the reaction zone is a strain modified diffusional zone, its thickness is best modeled with a diffusional length scale that is characteristic of the turbulence-induced strain field. Thus the reaction zone thickness is proportional to the Kolmogorov scale, η ,

$$\text{Thickness}_{flame} \propto \eta. \quad (3.265)$$

Kolmogorov's definition of the diffusive length scale is

$$\eta \equiv \left(\frac{\nu^3}{\epsilon} \right)^{1/4}. \quad (3.266)$$

Since this is a characteristic scale analysis, the molecular mixture viscosity is evaluated at the surrounding temperature. The actual reaction zone thickness will be larger due to the volumetric expansion (i.e., lower density) in the reaction zone.

Based on these characteristic scales from the assumed reaction zone geometry in [Figure 3.4](#), expressions can be obtained for the mass transfer rate per total mass. The mass exchange rate into the reaction zone per unit eddy mass is given by

$$\frac{\dot{m}_{flame}}{M_{cell}} = \left(\frac{\dot{m}_{flame}}{M_{eddy}} \right) \left(\frac{M_{eddy}}{M_{cell}} \right). \quad (3.267)$$

The first term on the right hand side is given by

$$\frac{\dot{m}_{flame}}{M_{eddy}} = \frac{(\text{SurroundingDensity}) (\text{FlameArea}) (\text{MassTransferVelocity})}{(\text{EddyDensity}) (\text{EddyVolume})}. \quad (3.268)$$

The interpretation of the second term on the right hand side depends upon filtering used (i.e., averaging over scales). For LES, the length scale of the eddy being modeled is proportional to the length scale of the grid. In this case, the size of the eddy and the grid are the same. Therefore, the second term is unity. In RANS modeling, the eddy is much larger than the grid, as is the reaction zone surface being modeled. For RANS, it is assumed that averaged over a sufficient number of eddies, the mass exchange rate into the reaction zone per unit eddy (first term) is uniformly distributed (i.e., independent of length scale) up to the integral length scales. In this case the second term is irrelevant and is assigned a value of unity. For example, for an integral scale eddy with a length scale ten times the grid, the mass transfer into the reaction zone (averaged over many eddies) would be ten times the value for an eddy with a length scale that is just the size of the grid.

Conservation of mass requires that the mass exchange rate into and out of the reaction zone be identical so the properties can be evaluated at the thermodynamic state of either the reactant stream (surroundings) or the product stream (reaction zone). For convenience, they will be defined in terms of the reactant stream temperature and mass fractions. Using the characteristic length and velocity scale arguments given above yields

$$\frac{\dot{m}_{flame}}{M_{cell}} \propto \frac{(\rho_{surr}) (L^2 \chi) (v)}{(\rho_{cell}) (L^3)} = \chi \frac{v}{L} \frac{\rho_{surr}}{\rho_{cell}}. \quad (3.269)$$

The standard integral scale estimate [10] of the rate of energy supply to diffusive scale eddies is

$$\epsilon \propto \frac{\text{TurbulentKineticEnergy}}{\text{EddyRollOverTime}} \propto \frac{u'^2}{L/u'} = \frac{u'^3}{L}. \quad (3.270)$$

The turbulence kinetic energy is given as

$$k = \frac{3}{2} u'^2. \quad (3.271)$$

Substituting and rearranging gives

$$L \propto \frac{k^{3/2}}{\epsilon}. \quad (3.272)$$

Ignoring the constant of proportionality and substituting the results into the definition for the Kolmogorov velocity gives

$$v \propto L \left(\frac{\epsilon}{k} \right) \left(\frac{\nu \epsilon}{k^2} \right)^{1/4}. \quad (3.273)$$

Substituting gives the mass exchange rate into the reaction zone per control volume in terms of standard turbulence parameters,

$$\frac{\dot{m}_{flame}}{M_{cell}} \propto \left(\frac{\nu \epsilon}{k^2} \right)^{1/4} \left(\frac{\epsilon}{k} \right) \chi \frac{\rho_{surr}}{\rho_{cell}}. \quad (3.274)$$

The function χ is a scalar correction to take into account species effects on the reaction zone area. The function is bounded between (0,1) with 1 representing optimal species concentrations which will maximize the reaction zone area and 0 representing prohibitive species concentrations which would prevent reaction zone formation. Two scalar properties are important, the reactant concentrations and the product concentration (which acts as an ignition source since ignition is not assumed). Therefore, the limiter is written as the product of two terms,

$$\chi = \chi_1 \chi_2. \quad (3.275)$$

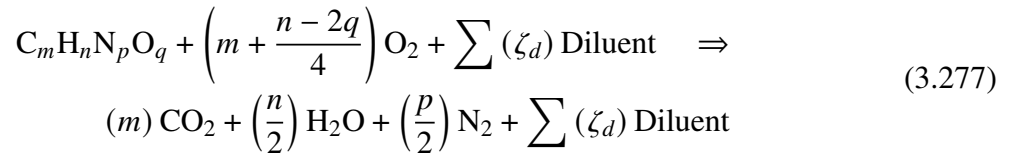
The function χ_1 is intended to take into account the effect of the reactant mass fractions on the reaction zone surface area. Since the reaction zone surface occurs at stoichiometric concentrations of fuel and oxygen in a diffusion flame, stoichiometric concentrations of reactants in a control volume will result in the largest reaction zone area (controlled by the turbulence levels). In this

case, χ_1 is unity. On the other hand, if either fuel or oxygen is zero within a control volume, then χ_1 is zero. Between these extremes, a functional form is assumed which has the correct limiting properties. The function is given by

$$\chi_1 = \frac{1}{\left(\frac{\hat{Y}_{oxy} + \hat{Y}_{prod}}{\hat{Y}_{min} + \hat{Y}_{prod}} \right) \left(\frac{\hat{Y}_{fuel} + \hat{Y}_{prod}}{\hat{Y}_{min} + \hat{Y}_{prod}} \right)}, \quad (3.276)$$

where the normalized mass fractions are defined below.

Overall reaction stoichiometry is determined from the sum of Reactions 1 and 2 in the chemical reaction section ((3.237) and (3.238)). The overall reaction is



For the overall reaction, the mass ratio of oxygen to the mass ratio of fuel for stoichiometric reaction to CO_2 and H_2O is given by

$$S = \left(\frac{Y_{oxy}}{Y_{fuel}} \right)_{Optimal} = \left(m + \frac{n}{4} - \frac{q}{2} \right) \left(\frac{32}{12m + n + 14p + 16q} \right). \quad (3.278)$$

Since mass is conserved in the reaction, $1 + S$ kilograms of product (CO_2 and H_2O) are produced for every kilogram of fuel consumed for a fuel/oxygen reaction. Note the mass of diluents, such as the nitrogen in the air does not change, as a result of the reaction. It is useful to produce normalized mass fractions based on the masses involved in the stoichiometric reaction.

$$\hat{Y}_{oxy} = \frac{Y_{oxy}}{S} \quad \hat{Y}_{prod} = \frac{Y_{co2} + Y_{h2o}}{1 + S} \quad \hat{Y}_{fuel} = \frac{Y_{fuel}}{1} \quad (3.279)$$

Note that the sum of these terms does not equal unity but one minus the mass fraction of diluent in the mixture.

The actual reaction may involve the secondary fuels, so a more general expression is required for the stoichiometric mass ratio of oxygen to fuel (and is used in the Vulcan code).

$$S = \frac{SO2FU \cdot Y_{fuel} + SO2CO \cdot Y_{co} + SO2H2 \cdot Y_{h2}}{Y_{fuel} + Y_{co} + Y_{h2}}, \quad (3.280)$$

$$SO2FU = \left(m + \frac{n}{4} - \frac{q}{2} \right) \left(\frac{32}{12m + n + 14p + 16q} \right), \quad (3.281)$$

$$SO2CO = \frac{32}{2 \cdot 28}, \quad (3.282)$$

$$SO2H2 = \frac{32}{2 \cdot 2}. \quad (3.283)$$

The product mass fractions are adjusted for the mass of nitrogen that accompanies the oxygen in air – the nitrogen is treated as a product species. The normalized mass fractions are

$$\hat{Y}_{oxy} = \frac{Y_{oxy}}{S} \quad \hat{Y}_{prod} = \frac{3.39Y_{co2} + 3.92Y_{h2o}}{1 + 4.29S} \quad \hat{Y}_{fuel} = \frac{Y_{fuel}}{1} \quad (3.284)$$

where

$$Y_{prod}|_{co2} = Y_{co2} \left(1 + 3.76 \frac{MW_{n2}}{MW_{co2}} \right), Y_{prod}|_{h2o} = Y_{h2o} \left(1 + 1.88 \frac{MW_{n2}}{MW_{h2o}} \right), \quad (3.285)$$

The molar ratio of nitrogen to oxygen in air is 3.76 and the mass ratio is 3.29. The production mass fraction, Y_{prod} , can be computed directly from the CO_2 and H_2O mass fractions as long as the only source of product species in the flow field comes directly from combustion. If there is injection of product species into the domain from a diluent stream or from an ambient concentration, then a transport equation should be solved for the product mass fraction (see *Combustion Products Transport Equation*).

Since combustion always occurs at a stoichiometric surface in a diffusion flame, there is a limiting reactant mass fraction in a fuel/oxygen mixture within the control volume unless the ratio is stoichiometric. The limiting reactant mass fraction is given by

$$\hat{Y}_{min} = \min(\hat{Y}_{fuel}, \hat{Y}_{oxy}). \quad (3.286)$$

The function χ_1 can be seen to approach the correct limits most clearly if the mass fraction of products, Y_{prod} , is set to zero. If the mixture is fuel lean, $\hat{Y}_{min} = \hat{Y}_{fuel}$ and χ_1 is equal to the fuel to oxygen ratio which decreases to zero as the fuel mass fraction is decreased. If the mixture is fuel rich, $\hat{Y}_{min} = \hat{Y}_{oxy}$ and χ_1 is equal to the oxygen to fuel ratio which decreases to zero as the fuel mass fraction is increased. At stoichiometric, χ_1 is unity.

The function χ_2 is intended to take into account the existence of reaction zone surface as a precondition for reaction zone surface propagation. A stoichiometric surface without reaction can exist in a flow field if there is no ignition source. An external source is required for ignition. However, once ignited, reaction zone propagation can be interpreted as new flame surface being ignited by existing adjacent reaction zone surface. A good indicator of existing flame surface is the presence of hot combustion products within the control volume and this fact is used to create the function χ_2 .

The value of χ_2 is zero if no combustion products are present. If the product mass fraction was uniformly distributed, then the probability of ignition would increase with the ratio of product mass fraction to reactant mass fraction. However, the combustion products are not uniformly distributed but concentrated around the reaction zones, thereby increasing the probability of propagation of reaction zone surface for a given product mass fraction. The assumed functional form of χ_2 that has these characteristics is

$$\chi_2 = \frac{\left(\frac{\text{ExistingProductMassFraction}}{\text{MaxFlameVolume}} \right)}{\left(\frac{\text{MaxPossibleProductMassFraction}}{\text{CharacteristicProductVolume}} \right)} \quad (3.287)$$

$$= \left(\frac{\text{CharacteristicProductVolume}}{\text{MaxFlameVolume}} \right) \left(\frac{\text{ExistingProductMassFraction}}{\text{MaxPossibleProductMassFraction}} \right).$$

The maximum volume of the reaction zone is the thickness times its area,

$$\gamma = \frac{(\text{Area} \cdot \text{Thickness})_{\text{flame}}}{\text{Volume}_{\text{eddy}}} \propto \frac{L^2 \eta}{L^3} = \frac{\eta}{L}. \quad (3.288)$$

Using the definition for the Kolmogorov length scale and substituting the turbulence kinetic energy for length scale, L , gives

$$\gamma \propto \left(\frac{\nu \epsilon}{k^2} \right)^{3/4}, \quad (3.289)$$

which is the maximum reaction zone volume per eddy volume. The value of $\gamma^{1/3}$ is bounded by one since the length scale ratio of the flame volume to eddy volume cannot be larger than one.

The characteristic product volume can be defined by assuming the majority of combustion products are held up within a distance corresponding to the Taylor microscale from the reaction zone surface,

$$\gamma_\lambda = \frac{(\text{Area} \cdot \text{Thickness})_{\text{prod}}}{\text{Volume}_{\text{eddy}}} \propto \frac{L^2 \lambda}{L^3} = \frac{\lambda}{L}. \quad (3.290)$$

Note that this assumption is used only to establish an ignition probability. For actual property evaluation, it is assumed that the combustion products are well mixed with the surroundings. Taking the ratio of the volumes gives,

$$\frac{\gamma_\lambda}{\gamma} \propto \frac{\lambda}{\eta}. \quad (3.291)$$

Using the standard definition of this ratio (Tennekes and Lumley [10]) gives,

$$\frac{\lambda}{\eta} = \text{Re}_L^{1/4}. \quad (3.292)$$

The Reynolds number can be defined in terms of turbulent kinetic energy and its dissipation by,

$$\text{Re}_L = \frac{k^2}{\nu \epsilon}. \quad (3.293)$$

Substituting gives,

$$\left(\frac{\text{CharacteristicProductVolume}}{\text{MaximumFlameVolume}} \right) = \left(\frac{1}{\gamma^{1/3}} \right). \quad (3.294)$$

The existing product mass fraction is given by Y_{prod} . The maximum possible product mass fraction is the sum of the existing products and the products that could be formed if all available reactants were to burn. Since combustion takes place at a stoichiometric surface, the limiting reactant mass fraction is given by Y_{min} . Therefore, χ_2 becomes

$$\chi_2 = \left(\frac{1}{\gamma^{1/3}} \right) \left(\frac{\hat{Y}_{\text{prod}}}{\hat{Y}_{\text{prod}} + \hat{Y}_{\text{min}}} \right). \quad (3.295)$$

Functionally, χ_2 can exceed unity but the product $\chi_1\chi_2$ is limited to the range (0,1). The function χ is now completely described in terms of species and turbulence properties.

Combining all previous results gives the following result for species consumption/production,

$$\frac{(\dot{m}_k)_{consumed/produced}}{M_{cell}} \propto [(Y_k)_{flame} - (Y_k)_{surr}] \left(\left(\frac{\nu\epsilon}{k^2} \right)^{1/4} \left(\frac{\epsilon}{k} \right) \chi \left(\frac{\rho_{surr}}{\rho_{cell}} \right) \right), \quad (3.296)$$

where χ is defined above in terms of χ_1, χ_2 .

The above derivation is intended to provide a physical interpretation to Magnussen's EDC model for large fires typified by medium turbulence levels with diffusive combustion. Proportionality constants are needed to close the model. As always, constants can be tweaked for a given flow to produce the best result for that flow. However, we will use the constants as derived for more general flows (Ertesv{aa}g and Magnussen [52]). With these constants, the model equations match those from the KAMELEON-II-FIRE code (Holen et al. [53]).

Using these constants, the maximum reaction zone volume fraction is given by

$$\gamma = 9.7 \left(\frac{\nu\epsilon}{k^2} \right)^{3/4}. \quad (3.297)$$

Taking into account species limitations, the flame volume fraction is given by

$$\gamma\chi = 9.7 \left(\frac{\nu\epsilon}{k^2} \right)^{3/4} \chi. \quad (3.298)$$

The reaction rate of fuel is given by

$$\frac{(\dot{m}_k)_{consumed/produced}}{M_{cell}} \propto [(Y_k)_{flame} - (Y_k)_{surr}] \left(23.6 \left(\frac{\nu\epsilon}{k^2} \right)^{1/4} \left(\frac{\epsilon}{k} \right) \chi \left(\frac{\rho_{surr}}{\rho_{cell}} \right) \chi_3 \right). \quad (3.299)$$

The additional scalar function, χ_3 , at the end of (3.299) is multiplier on the combustion rate that Magnussen found necessary to maintain the mass transfer rate when the product concentration is high in premixed flames. Its necessity suggests that perhaps alternate scalings should be examined, but for consistency with the published model, it is implemented here as

$$\chi_3 = \min \left[\frac{\hat{Y}_{prod} + \hat{Y}_{min}}{\hat{Y}_{min}}, \frac{1}{\gamma^{1/3}} \right]. \quad (3.300)$$

Average Control Volume Properties

The volume and mass exchange process between the two zones is assumed to be constant over a time step. Consequently, cell averaged properties for the mean flow equations are a volume weighted sum of the properties in the two zones. Therefore, all control volume properties are given by

$$\phi_{cell} = \frac{\phi_{flame} V_{flame} + \phi_{surr} V_{surr}}{V_{cell}}. \quad (3.301)$$

The maximum volume fraction of the reaction zone, γ , was determined previously from momentum considerations. The actual volume fraction is the maximum volume fraction times the scalar function, χ . The surroundings is the volume fraction that remains after the reaction zone volume has been removed. Therefore,

$$\phi_{cell} = \phi_{flame} (\gamma\chi) + \phi_{surr} (1 - \gamma\chi). \quad (3.302)$$

Volume averaged properties given by (3.302) are desired. However, the estimates used to obtain $\gamma\chi$ (i.e., (3.297)) are based on uniform cell temperatures. Clearly, the flame zone will be hotter than the surroundings, so the volume fraction occupied by the flame will be larger than given by (3.302) (and the surroundings fraction smaller).

A first order non-isothermal estimate is made to account for flame volume fraction. This estimate assumes that the non-homogeneous density field does not affect the local turbulence field (or alternately, that dilatation cancels the baroclinic generation) such that isothermal, isotropic, homogeneous turbulence estimates for the turbulent kinetic energy, k , and its dissipation, ϵ , hold. (This assumption is made in virtually all models by the necessity that the fundamental research to quantify the actual coupling has not been done.)

A mass balance then gives a first order estimate for the actual flame volume fraction at the flame temperature. The actual flame volume at the flame temperature is given by its isothermal estimate times the cell mean density (used to obtain the isothermal estimate) divided by the actual flame density.

Thus, (3.302) becomes,

$$\phi_{cell} = \phi_{flame} (\gamma\chi) \frac{\rho_{cell}}{\rho_{flame}} + \phi_{surr} (1 - \gamma\chi) \frac{\rho_{cell}}{\rho_{surr}}. \quad (3.303)$$

Where the mean density is given by

$$\rho_{cell} = \left[\frac{(\gamma\chi)}{\rho_{flame}} + \frac{(1 - \gamma\chi)}{\rho_{surr}} \right]^{-1}. \quad (3.304)$$

An interpretation of (3.303) and (3.304) is that $\gamma\chi$, is intended as a non-isothermal volume estimate, which the mass weighted isothermal volume estimate happens to be the best available estimator until turbulence coupling in reacting flows can be elucidated. All cell averaged properties are given by (3.304). (3.302) is intended for clarification only.

Limits Testing

Parameters in the EDC model take on limiting values in the presence of piloting conditions and extinction conditions. The limits are discussing in the following subsections.

Ignition Criteria

Ignition will not occur in the above mechanism unless products are formed. An external ignition source (or pilot flame) is simulated by setting χ to be greater than zero (the product mass fraction is set to 0.2 times the maximum products that could be formed by the existing fuel in the current implementation) in a cell with fuel and oxygen present. This can be done on a cell by cell basis to represent point ignition sources, or in the whole domain if global ignition is required. If a pilot flame is to be simulated, the cells associated with it have χ set to be greater than zero for the duration of the calculation. If a transient ignition (e.g., spark) is to be simulated, the cells initially have χ set to be greater than zero. However, after a minimum temperature is reached within a cell, T_{ign} (K) (a user input), χ is no longer specified but calculated from the species concentrations and turbulence levels as derived previously.

Extinction Criteria

Extinction occurs when $\chi = 0$. This occurs automatically when the fuel and/or air is consumed. Local extinction can also be caused within a cell due to high turbulence levels. At high turbulence levels, the reaction zone can be appropriately modeled as a perfectly stirred reactor (PSR). A PSR blows out when the residence time is less than a minimum value for a given composition. The residence time, τ_{res} , in the reaction zone volume is given by

$$\tau_{res} = \frac{\text{Volume}_{flame}}{\text{VolumeFlowRate}_{flame}} = \frac{\left(\frac{\text{Volume}_{flame}}{\text{Volume}_{cell}} \right)}{\frac{\left(\frac{\dot{m}_{flame}}{\rho_{surr}} \right)}{\left(\frac{M_{cell}}{\rho_{cell}} \right)}}. \quad (3.305)$$

Simplifying gives

$$\tau_{res} = \frac{\chi \gamma}{\left(\frac{\dot{m}_{flame}}{M_{cell}} \right) \frac{\rho_{cell}}{\rho_{surr}}}. \quad (3.306)$$

Substituting prior relations gives

$$\tau_{res} \propto \frac{\chi \left(\frac{\nu \epsilon}{k^2} \right)^{3/4}}{\chi \left(\frac{\nu \epsilon}{k^2} \right)^{1/4} \left(\frac{\epsilon}{k} \right)}. \quad (3.307)$$

Simplifying and substituting Magnussen's constant of proportionality gives

$$\tau_{res} = \frac{1}{2.43} \left(\frac{\nu}{\epsilon} \right)^{1/2}. \quad (3.308)$$

Comparison of the calculated residence time with a user input minimum residence time (based on pre-calculation using a PSR and appropriate chemistry) determines whether or not combustion is allowed to continue. If so, heat release is calculated as derived herein, and finite-rate effects are not considered. However, if the calculated residence time is below the minimum value, χ is set equal to zero which causes combustion to cease within a cell.

Laminar Values

As currently formulated, the model assumes the flow is fully turbulent and does not model laminar combustion. Minimum values for the reaction zone volume, γ , and mass transport into the reaction zone per mass in the cell, \dot{m}_{flame}/M_{cell} , are required in conditions with low turbulence levels to prevent singularities.

Scalar Limits

The mass fractions of fuel, air, and products must remain bounded (0,1). This requires that the consumption rate for the species with the limiting concentration times the time step must be less than or equal to the mass of species.

Cell Value Information Used By Model

The combustion model requires inputs from the transport equations for cell averaged variables at the start of a time step. These variables include pressure, P_{th} (dynes/cm²), species mass fractions Y_i , density, ρ_{cell} (g/cm³), mixture molecular weight, W_{mix} , (g/mole), turbulent kinetic energy, k (cm²/sec²), dissipation of the turbulent kinetic energy, ϵ (cm²/sec³), mixture kinematic viscosity, ν (cm²/sec), individual (i.e., chemical plus sensible) enthalpies, h_i (ergs/g), and mixture enthalpy, h_{cell} (ergs/g).

Model Outputs

The two outputs of the combustion model are the species consumption rates and property estimates.

Species Consumption Estimates

Noting the general relation between cell averaged values and surrounding values, (3.302), the surrounding and cell mass fractions can be related to give

$$[(Y_k)_{flame} - (Y_k)_{surr}] = \frac{[(Y_k)_{flame} - (Y_k)_{cell}]}{(1 - \gamma\chi)}. \quad (3.309)$$

Substituting this result and the definition of τ_{res} into the species consumption/production rate gives the source term in the species transport equation, (3.256),

$$\bar{\omega}_k = \frac{[(Y_k)_{flame} - (Y_k)_{cell}]}{\tau_{res}} \left(\frac{\gamma\chi}{1 - \gamma\chi} \right) \chi_3, \quad (3.310)$$

for the species mass production/consumption rate in a control volume. The subscript k is understood to be for each species, $C_m H_n N_p O_q$, O_2 , N_2 , CO , H_2 , CO_2 , H_2O , and any diluents in the system.

Property Estimates

It is important in turbulent processes that nonlinear fluctuating quantities be appropriately represented. Properties for which nonlinear fluctuations are important include the radiative emissive power (proportional to the fourth power of temperature) and density.

To get the radiative emissive power, it is first necessary to get the temperature within each zone. This is accomplished by iterative estimate based on the species mass fractions within each zone. Since total (chemical plus sensible) enthalpy is used for each species, the total enthalpy per unit mass in the control volume does not change between the reaction zone or the surrounding zone. The partitioning of chemical and sensible enthalpy is different for the reaction and surrounding zone, but the specific total enthalpy is equal to the cell value defined at the beginning of each time step. (Note: this is not a statement of the energy equation, it is only a statement of property values within each zone and the cell. Obviously, the enthalpy does vary after radiation transport is allowed to occur and species are allowed to advect between cells at the end of the time step as governed by the energy equation).

The reaction zone temperature, T_{flame} , is obtained from iterative solution of

$$h_{flame} = \sum Y_k h_k(T) \Big|_{flame}, \quad (3.311)$$

and the surrounding temperature, T_{surr} , is obtained from iterative solution of

$$h_{surr} = \sum Y_k h_k(T) \Big|_{surr}. \quad (3.312)$$

The average emissive power is given by

$$\overline{\sigma \alpha T^4}_{rad} = \sigma \bar{\alpha} \left(T_{flame}^4 (\gamma\chi) \frac{\bar{\rho}}{\rho_{flame}} + T_{surr}^4 (1 - \gamma\chi) \frac{\bar{\rho}}{\rho_{surr}} \right) \quad (\text{ergs/cm}^2\text{-s}). \quad (3.313)$$

An important assumption implied by the form of (3.313) is that the turbulent fluctuations between the temperature and absorption coefficient are weakly correlated [7]. (Note that the intent of the averaging form above is to volume-weight the emissive power from the flame and surrounding zones. This form implies that $\gamma\chi$ should be viewed as a mass fraction rather than a volume fraction as discussed for (3.301).)

The density of each zone can be calculated according to the perfect gas law. For the reaction zone volume, the density is

$$\rho_{flame} = \frac{P_{th}W_{flame}}{RT_{flame}}, \quad (3.314)$$

where R is the universal gas constant and P_{th} is the constant thermodynamic pressure. For the surroundings, the density is

$$\rho_{surr} = \frac{P_{th}W_{surr}}{RT_{surr}}. \quad (3.315)$$

The soot model uses the temperatures, densities, and mass fractions of reaction zone and surroundings according to the above estimates.

Combustion Products Transport Equation

The product mass fraction represents the products formed by combustion (CO_2 and H_2O for hydrocarbon fuels, and H_2O for hydrogen fuel). If any of the product species are injected into the domain through either an initial condition or boundary condition to simulate a diluent stream or ambient concentration, their influence must be removed in order for the χ_2 reaction limiter to function properly. A transport equation similar to (3.256) is used where the reaction rate is given by

$$\dot{\omega}_{prod} = 3.392\dot{\omega}_{co2} + 3.924\dot{\omega}_{h2o}. \quad (3.316)$$

This transported product mass fraction can only be formed due to reaction within the domain and cannot be injected through either initial or boundary conditions. Therefore, the only boundary conditions that are required are at an outflow so that products may exit the domain, and a zero value at any surfaces where a species Dirichlet condition is applied. All of these cases are handled automatically so that nothing needs to be specified by the user.

Note that a pilot stream will be unable to ignite a flame when using this model. It will be treated as an inert diluent stream, so that the normal ignition model will be required to ignite the flame. This model in its current form should not be used for piloted flames.

Also note that if the only source of products in the simulation is combustion, then the product mass fraction can be computed directly from the local species mass fractions and solving this transport equation is unnecessary.

Chemical Equilibrium Models

The EDC combustion model uses a two-step chemical reaction, where the fuel species is consumed by the reaction in (3.237) to form CO and H₂, and then these intermediate species are consumed by the reaction in (3.238) to form CO₂ and H₂O. If oxygen is present in excess, then none of the intermediate species will remain and only CO₂ and H₂O will be produced. In reality, these reactions would not proceed to completion, but instead would reach an equilibrium where some of the intermediate species can persist. This can lead to a significantly different mixture composition and even a different mixture temperature than what the standard EDC model would predict, especially at higher temperatures.

Fuego includes two optional models that can include the effects of two independent chemical equilibrium reactions into the standard EDC model, to better predict high-temperature combustion species and temperatures.

CO₂ Dissociation Model

At high temperatures, the equilibrium reaction



becomes active to dissociate CO₂ species back into CO and O₂, which has the effect to cool the gas mixture. Including the effects of this dissociation reaction will help to control nonphysically-high temperatures that might result otherwise.

This model will adjust the EDC-reacted mixture $(Y_k)_{\text{flame}}$ in (3.310) to include the effects of equilibrium reaction (3.317). This equilibrium can be modeled by

$$K_p = \exp\left(\frac{-\Delta G_T^o}{R_u T}\right), \quad (3.318)$$

where R_u is the ideal gas constant, T is the temperature at which the equilibrium is being calculated, K_p is the equilibrium constant for this dissociation reaction, and ΔG_T^o is the standard-state Gibbs function change for this reaction. The equilibrium constant K_p for (3.317) is defined as

$$K_p = \frac{\left(\frac{P_{\text{CO}}}{P^o}\right) \left(\frac{P_{\text{O}_2}}{P^o}\right)^{\frac{1}{2}}}{\left(\frac{P_{\text{CO}_2}}{P^o}\right)}, \quad (3.319)$$

where P_{CO} , P_{O_2} , and P_{CO_2} are the partial pressures of CO, O₂, and CO₂, respectively, and P^o is the reference pressure taken as 1 atm. The standard-state Gibbs function change for this reaction can be evaluated in terms of the Gibbs function of formation for each species at temperature T ,

$$\Delta G_T^o = \left(\bar{g}_{f,\text{CO}}^o + \frac{1}{2} \bar{g}_{f,\text{O}_2}^o - \bar{g}_{f,\text{CO}_2}^o \right)_{T_{\text{ref}}=T}. \quad (3.320)$$

The partial pressure of species k can be computed by $P_k = X_k P$, where P is the static pressure of the mixture and X_k is the mole fraction of species k , defined as $X_k = n_k / n_{\text{tot}}$ with the total number of moles of all species being defined as $n_{\text{tot}} = \sum_i n_i$. After making these substitutions and simplifying, the equilibrium equation that needs to be solved, written in terms of moles of each species in a fixed-mass volume, is

$$\frac{n_{\text{CO}} n_{\text{O}_2}^{1/2}}{n_{\text{CO}_2} n_{\text{tot}}^{1/2}} \left(\frac{P}{P^o} \right)^{\frac{1}{2}} = \exp \left(\frac{-\Delta G_T^o}{R_u T} \right). \quad (3.321)$$

Additional equations may be written to enforce conservation of C and O atoms within the reaction volume,

$$N_C = n_{\text{CO}} + n_{\text{CO}_2} \quad (3.322)$$

$$N_O = 2 n_{\text{CO}_2} + n_{\text{CO}} + 2 n_{\text{O}_2}, \quad (3.323)$$

where N_C and N_O are the fixed number of moles of carbon and oxygen atoms, respectively, during the equilibrium reaction. (3.321), (3.322), and (3.323) represent a system of three equations that can be solved for the three unknowns n_{CO_2} , n_{CO} , and n_{O_2} at the equilibrium state.

The numerical solution procedure begins by approximating the number of moles of each species from the reacted mixture mass fraction vector Y_i as $n_i = Y_i / W_i$, on a per-unit-mass-of-mixture basis. Eliminating n_{CO_2} and n_{O_2} from (3.321) yields a nonlinear equation that can be solved directly for n_{CO} from the fixed atom balances at a fixed temperature T and pressure P ,

$$n_{\text{CO}}^2 (N_O - 2N_C + n_{\text{CO}}) - \left(\frac{P^o}{P} \right) \exp \left(\frac{-2\Delta G_T^o}{R_u T} \right) (N_C - n_{\text{CO}})^2 \left(N_O + n_{\text{CO}} + 2 \sum_j^{N_{\text{inert}}} n_j \right) = 0, \quad (3.324)$$

where $\sum_j^{N_{\text{inert}}} n_j$ represents the summation of the number of moles of all species present in the mixture that do not participate directly in the equilibrium reaction, {it i.e.} all species except for CO_2 , CO , and O_2 .

A standard Newton's method may be used to iteratively solve (3.324),

$$n_{\text{CO}}^{n+1} = n_{\text{CO}}^n - \frac{f(n_{\text{CO}})}{f'(n_{\text{CO}})}, \quad (3.325)$$

where the function $f(n_{\text{CO}})$ is (3.324) and the derivative function $f'(n_{\text{CO}})$ is

$$f'(n_{\text{CO}}) = 2 n_{\text{CO}} (N_O - 2N_C + n_{\text{CO}}) + n_{\text{CO}}^2 - \left(\frac{P^o}{P} \right) \exp \left(\frac{-2\Delta G_T^o}{R_u T} \right) \left[(N_C - n_{\text{CO}})^2 - 2 (N_C - n_{\text{CO}}) \left(N_O + n_{\text{CO}} + 2 \sum_j^{N_{\text{inert}}} n_j \right) \right]. \quad (3.326)$$

Once this equation is solved for n_{CO} , then the following equations may be used to evaluate the remaining equilibrium species moles,

$$n_{\text{CO}_2} = N_C - n_{\text{CO}} \quad (3.327)$$

$$n_{O_2} = \frac{1}{2} (N_O - 2N_C + n_{CO}) . \quad (3.328)$$

With the new molar mixture defined for the equilibrium species, the mass fraction vector may be reconstructed by $Y_i = n_i W_i$. This new mixture composition will result in a different temperature since the enthalpy is fixed. After the new temperature is evaluated, this entire procedure may be repeated iteratively until the mixture temperature converges to within a specified tolerance.

H₂ Dissociation Model

Similar to the CO₂ dissociation model described in *\mathrm{CO_2} Dissociation Model*, at high temperatures the equilibrium reaction



becomes active to dissociate H₂ species into H atoms, which has the effect to cool the gas mixture. Including the effects of this dissociation reaction in addition to the CO₂ dissociation reaction will help to control nonphysically-high temperatures that might result otherwise.

This model will adjust the EDC-reacted mixture $(Y_k)_{\text{flame}}$ in (3.310) to include the effects of equilibrium reaction (3.329). This equilibrium can be modeled by (3.318), with the equilibrium constant defined as

$$K_p = \frac{\left(\frac{P_H}{P^o}\right)^2}{\left(\frac{P_{H_2}}{P^o}\right)}, \quad (3.330)$$

where P_H and P_{H_2} are the partial pressures of H and H₂, respectively. The standard-state Gibbs function change for this reaction can be evaluated in terms of the Gibbs function of formation for each species at temperature T ,

$$\Delta G_T^o = \left(2 \bar{g}_{f,H}^o - \bar{g}_{f,H_2}^o\right)_{T_{\text{ref}}=T} . \quad (3.331)$$

Simplifying this equilibrium expression and writing it in terms of the number of moles of each species in a fixed-mass volume results in the equilibrium equation

$$\frac{n_H^2}{n_{H_2} n_{\text{tot}}} \left(\frac{P}{P^o}\right) = \exp\left(\frac{-\Delta G_T^o}{R_u T}\right). \quad (3.332)$$

An additional equation may be written to enforce conservation of H atoms within the reaction volume,

$$N_H = n_H + 2 n_{H_2}, \quad (3.333)$$

where N_H is the fixed number of moles of hydrogen atoms during the equilibrium reaction. (3.332) and (3.333) represent a system of two equations that can be solved for the two unknowns n_{H_2} and n_H at the equilibrium state.

Similar to the CO₂ dissociation model, the numerical solution procedure begins by approximating the number of moles of each species from the reacted mixture mass fraction vector Y_i as $n_i = Y_i/W_i$, on a per-unit-mass-of-mixture basis. Eliminating n_{H_2} from (3.332) yields a nonlinear equation that can be solved directly for n_H from the fixed atom balance at a fixed temperature T and pressure P ,

$$n_H^2 - \frac{1}{4} \left(\frac{P^o}{P} \right) \exp \left(\frac{-\Delta G_T^o}{R_u T} \right) (N_H - n_H) \left(N_H + n_H + 2 \sum_j^{N_{\text{inert}}} n_j \right) = 0, \quad (3.334)$$

where $\sum_j^{N_{\text{inert}}} n_j$ represents the summation of the number of moles of all species present in the mixture that do not participate directly in the equilibrium reaction, {i.e.} all species except for H₂ and H.

A standard Newton's method may be used to iteratively solve (3.334),

$$n_H^{n+1} = n_H^n - \frac{f(n_H)}{f'(n_H)}, \quad (3.335)$$

where the function $f(n_H)$ is (3.334) and the derivative function $f'(n_H)$ is

$$f'(n_H) = 2 n_H - \frac{1}{4} \left(\frac{P^o}{P} \right) \exp \left(\frac{-\Delta G_T^o}{R_u T} \right) \left[(N_H - n_H) - \left(N_H + n_H + 2 \sum_j^{N_{\text{inert}}} n_j \right) \right]. \quad (3.336)$$

Once this equation is solved for n_H , then the following equations may be used to evaluate the remaining equilibrium species moles,

$$n_{H_2} = \frac{1}{2} (N_H - n_H) \quad (3.337)$$

With the new molar mixture defined for the equilibrium species, the mass fraction vector may be reconstructed by $Y_i = n_i W_i$. This new mixture composition will result in a different temperature since the enthalpy is fixed. After the new temperature is evaluated, this entire procedure may be repeated iteratively until the mixture temperature converges to within a specified tolerance.

Laminar Flamelet Turbulent Combustion Model

Laminar flamelet models for non-premixed turbulent combustion treat turbulent flames as an ensemble of laminar diffusion flames. [54] Nonequilibrium chemistry effects may be included in the model by accounting for localized fluid strain, resulting in what is classically called the Strained Laminar flamelet Model (SLFM). Nonadiabatic effects may also be included by accounting for losses to the surroundings in the ensemble of flamelets.

The fundamental assumption is that the chemical time scales of the important reactions are fast enough that they occur only in a thin layer around stoichiometry, thinner (ideally) than the smallest

scales of the turbulence. Defining a small quantity $\epsilon = \ell_{\text{reaction zone}}/\ell_{\text{mixing layer}} \ll 1$, we can examine the governing equations in that thin region using a multiscale asymptotic expansion as

$$Y_i = Y_i(\zeta, \tau, x, t) + \epsilon Y_i^1(\zeta, \tau, x, t) + \dots, \quad \zeta = \frac{Z(x, t) - Z_{\text{st}}}{\epsilon} \text{ and } \tau = t/\epsilon^2. \quad (3.338)$$

Collecting the dominant terms, making some standard simplifications, and assuming that the chemical reaction scales as ϵ^{-2} , the state of the gas depends on the flow scale Z and $\chi = 2D|\nabla Z|^2$:

$$\rho \frac{\partial Y_i}{\partial t} - \frac{\rho \chi}{2} \frac{1}{\text{Le}_i} \frac{\partial^2 Y_i}{\partial Z^2} - \dot{\omega}_i(\vec{\Phi}) = 0 \quad (3.339)$$

$$\rho \frac{\partial T}{\partial t} - \frac{\rho \chi}{2} \left(\frac{\partial^2 T}{\partial Z^2} + \frac{1}{c_p} \frac{\partial c_p}{\partial Z} \frac{\partial T}{\partial Z} \right) - \dot{\omega}_T(\vec{\Phi}) = 0 \quad (3.340)$$

$$T(Z = 0, t) = T_{\text{ox}}, T(Z = 1, t) = T_{\text{fuel}}, Y_i(Z = 0, t) = Y_{i,\text{ox}}, Y_i(Z = 1, t) = Y_{i,\text{fuel}}, \quad (3.341)$$

$$\rho = \rho(\vec{\Phi}) \text{ and } c_p = c_p(\vec{\Phi}), \quad (3.342)$$

where $\vec{\Phi}$ is the state vector $\vec{\Phi} = (P_{\text{th}}, T, Y_0, Y_1, \dots, Y_N)$. The approximation allows us to resolve the chemical scales in the phase space of the mixture fraction instead of on a three-dimensional grid, granting dramatic computational savings. If we make the additional assumption that the chemistry is quasi-steady on the scale of the flow, then the chemical structure in mixture fraction space can be pre-computed offline from the simulation for a range of flow parameters χ and tabulated (using `fuego_tabular_props`). During the flow simulation, the solution of the flamelet simulation can be queried to determine required flow properties, e.g. $\rho = \rho(Z, \chi)$. Note that the flamelet formulation in Eq. [Laminar Flamelet Turbulent Combustion Model](#) is specifically for a “two-stream” problem, with constant Lewis numbers, where the boundary and initial conditions of the simulation can be completely described by linear combinations of two constant state vectors. Additional “streams” and boundary heat losses will require additional transport equations to be solved.

This section summarizes the basic formulation and implementation details of both the adiabatic and nonadiabatic flamelet model and SLF model, including both the property table generation procedure in `fuego_tabular_props` and the usage of the property table in `fuego` to evaluate turbulent filtered quantities of interest for both adiabatic and nonadiabatic configurations.

Adiabatic Property Table Generation

Laminar Flamelet Generation

Unstrained flamelet libraries, where nonequilibrium chemistry effects may be neglected with respect to fluid strain rates, can be generated directly with the `fuego_tabular_props` application. These libraries should be used either in laminar flow or in turbulent flow where the turbulence/chemistry interactions may be neglected.

Equilibrium chemistry, Burke-Schumann chemistry, or nonreacting flow scenarios are supported in configurations where there are two or more streams that may be mixed and potentially reacted. The stream composition is parameterized by the mixture fraction vector Z_m , where each of the M component represents the fraction of mass that originated in that stream, where there are N streams and $M = N - 1$. The mixture fraction for the final stream may be evaluated as $Z_N = 1 - \sum_{m=1}^M Z_m$.

The resulting flamelet data can then be assembled into a sequence of multi-dimensional tables of dependent variable ϕ as a function of the mixture fraction vector, $\phi(Z_m)$, and can be used directly for laminar simulations. Adding turbulence interactions, nonequilibrium effects, and nonadiabatic effects will increase the dimensionality of this lookup table and require additional processing. See the following sections for more information.

Strained Laminar Flamelet Libraries

Strained laminar flamelet data may be generated for use in Fuego with the Spitfire code. This data is two-dimensional in nature, determined by the mixture fraction and a reference scalar dissipation rate χ_o at a reference mixture fraction Z_o . The instantaneous laminar scalar dissipation rate is defined as

$$\chi = 2D \frac{\partial Z}{\partial x_i} \frac{\partial Z}{\partial x_i}, \quad (3.343)$$

with D being the molecular mass diffusion coefficient. The reference value χ_o is arbitrary, although typical choices include the stoichiometric value $\chi_{st} = \chi(Z_{st})$ or the maximum value $\chi_{max} = \chi(Z = 0.5)$. Stoichiometric values are used in Fuego. Spitfire will then assemble it into a sequence of multi-dimensional tables of dependent variable ϕ as a function of the mixture fraction vector and reference scalar dissipation rate, $\phi(Z_m, \chi_o)$. These tables may be then be used to evaluate properties in Fuego.

Aksit-Moss SNL soot model

A modified Aksit-Moss soot model [55] is available, solving for soot moles per mass (N_s) and soot mass fraction (Y_s)

$$\int \frac{\partial \bar{\rho} \tilde{Z}}{\partial t} dV + \int \bar{\rho} \tilde{u}_j \tilde{Z} n_j dS = \int \left(\frac{\bar{\mu}}{Sc} \frac{\partial \tilde{Z}}{\partial x_j} - \tau_{Z,j}^{sgs} \right) n_j dS + \int \dot{S}_Z dV \quad (3.344)$$

$$\int \frac{\partial \bar{\rho} \tilde{h}}{\partial t} dV + \int \bar{\rho} \tilde{u}_j \tilde{h} n_j dS = \int \left(\frac{\bar{\mu}}{Pr} \frac{\partial \tilde{h}}{\partial x_j} - \tau_{h,j}^{sgs} \right) n_j dS - \int \dot{S}_h dV \quad (3.345)$$

$$\int \frac{\partial \bar{\rho} \tilde{h}_c}{\partial t} dV + \int \bar{\rho} \tilde{u}_j \tilde{h}_c n_j dS = \int \left(\frac{\bar{\mu}}{Pr} \frac{\partial \tilde{h}_c}{\partial x_j} - \tau_{h_c,j}^{sgs} \right) n_j dS \quad (3.346)$$

$$\int \frac{\partial \bar{\rho} \bar{N}_s}{\partial t} dV + \int \bar{\rho} \bar{u}_j \bar{N}_s n_j dS = \int \left(\frac{\bar{\mu}}{Sc_s} \frac{\partial \bar{N}_s}{\partial x_j} - \tau_{N_s, j}^{sgs} \right) n_j dS + \int \dot{S}_{N_s} dV \quad (3.347)$$

$$\int \frac{\partial \bar{\rho} \bar{Y}_s}{\partial t} dV + \int \bar{\rho} \bar{u}_j \bar{Y}_s n_j dS = \int \left(\frac{\bar{\mu}}{Sc_s} \frac{\partial \bar{Y}_s}{\partial x_j} - \tau_{Y_s, j}^{sgs} \right) n_j dS + \int \dot{S}_{Y_s} dV \quad (3.348)$$

with W_s denoting the molecular weight of soot and γ_L a leaning coefficient representing the rate at which mixture fraction is leaned per soot mass fraction rate. Source terms are defined as

$$\dot{S}_{N_s} = \dot{S}_{\text{nucleation}} - \dot{S}_{\text{coagulation}} \bar{Y}_s^{1/6} \bar{N}_s^{11/6} \quad (3.349)$$

$$\dot{S}_{Y_s} = W_s \dot{S}_{\text{nucleation}} + [\dot{S}_{\text{surface growth}} - \dot{S}_{\text{O}_2 \text{ oxid.}} - \dot{S}_{\text{OH oxid.}}] \bar{Y}_s^{2/3} \bar{N}_s^{1/3} \quad (3.350)$$

$$\dot{S}_Z = -\gamma_L \dot{S}_{Y_s} \quad (3.351)$$

$$\dot{S}_h = 4\sigma \left(\bar{q}_s^r Y_s + \bar{q}_{\text{gas}}^r \right) - (\bar{\alpha}_{\text{soot}} Y_s + \bar{\alpha}_{\text{gas}}) \bar{G}. \quad (3.352)$$

Each of the rates, such as $\dot{S}_{\text{nucleation}}$, must be provided as a tabulated source term to the model, *e.g.* from activating the “Aksit-Moss SNL” soot model in SpitFire.

Turbulent Averaging

In turbulent simulations, a filtered form of the governing equations are solved to reduce the resolution requirements to an affordable level. Temporal filtering is used in Reynolds Averaged Navier-Stokes (RANS) models and spatial filtering is used in Large Eddy Simulation (LES) models. Both types of filtering are represented with the notation $\bar{\phi}$, and are handled similarly in the present work. Density-weighted, or Favre filtering greatly simplifies the treatment of variable-density flow. A Favre-filtered quantity is represented by $\tilde{\phi} \equiv \bar{\rho \phi} / \bar{\rho}$.

For use in turbulent simulations, a Favre-filtered version of the variables in the property table must be calculated. This is performed by convoluting the property variable with the joint PDF of the independent variable sub-filter fluctuations, and is mathematically expressed as

$$\tilde{\phi}(\tilde{Z}_m, \widetilde{Z''^2}, \tilde{\chi}) = \int_0^\infty \int_0^1 \phi(Z_m, \chi_o) p_{Z\chi}(Z_i, \chi; \tilde{Z}_m, \widetilde{Z''^2}, \tilde{\chi}) dZ_i d\chi, \quad (3.353)$$

where $p_{Z\chi}(Z_i, \chi; \tilde{Z}_m, \widetilde{Z''^2}, \tilde{\chi})$ is the joint PDF of sub-filter fluctuations of the dependent variable ϕ in Z_i - χ space, parameterized by the filtered mixture fractions \tilde{Z}_m and the variance $\widetilde{Z''^2}$ of mixture fraction component Z_i , and the filtered scalar dissipation rate $\tilde{\chi}$. The reference scalar dissipation rate has the functionality $\chi_o(\tilde{Z}_i, \widetilde{Z''^2}, \tilde{\chi})$, which will be discussed in the following section. Variance of only a single component of mixture fraction, Z_i , is considered at present for simplicity, although extensions to include additional components are possible. Statistical independence will be assumed between Z_i and χ fluctuations, so that

$$\tilde{\phi}(\tilde{Z}_m, \widetilde{Z''^2}, \tilde{\chi}) = \int_0^\infty \int_0^1 \phi(Z_m, \chi_o) p_Z(Z_i; \tilde{Z}_m, \widetilde{Z''^2}) p_\chi(\chi; \tilde{\chi}) dZ_i d\chi. \quad (3.354)$$

For the present work, $p_Z(Z_i; \tilde{Z}_m, \widetilde{Z''^2})$ will be modeled as either a beta PDF or a clipped Gaussian PDF and $p_\chi(\chi; \tilde{\chi})$ will be modeled as the delta function $\delta(\chi - \tilde{\chi})$.

Property Table Implementation

The convolution integral in (3.354) would be prohibitively expensive to evaluate each time a value for $\tilde{\phi}$ is needed by a turbulent reacting simulation. Therefore, this integral will be pre-calculated so that each property table query will only involve an interpolation from a table of values.

Storing the final $\tilde{\phi}(\tilde{Z}_m, \widetilde{Z''^2}, \tilde{\chi})$ values directly is undesirable since the range of possible $\tilde{\chi}$ values for each flamelet is different, resulting in a non-orthogonal table. Instead, the values $\tilde{\phi}_T(\tilde{Z}_m, \widetilde{Z''^2}, \chi_o)$ are stored in an orthogonal table that is indexed by \tilde{Z}_m , $\widetilde{Z''^2}$, and $\chi_o(\tilde{Z}_m, \widetilde{Z''^2}, \tilde{\chi})$. These tabulated values are calculated by

$$\tilde{\phi}_T(\tilde{Z}_m, \widetilde{Z''^2}, \chi_o) = \int_0^1 \phi(Z_m, \chi_o) p_Z(Z_i; \tilde{Z}_m, \widetilde{Z''^2}) dZ_i. \quad (3.355)$$

The reference scalar dissipation rate χ_o needed for lookup in the table for $\tilde{\phi}_T(\tilde{Z}_m, \widetilde{Z''^2}, \chi_o)$ can be evaluated from the local filtered scalar dissipation rate $\tilde{\chi}$ through laminar flamelet theory. The instantaneous scalar dissipation rate χ can be approximated by

$$\begin{aligned} \chi &= \chi_{\max} \exp\left(-2[\text{erfc}^{-1}(2Z)]^2\right) \\ &= \chi_{\max} F_\chi(Z), \end{aligned} \quad (3.356)$$

where χ_{\max} is the maximum scalar dissipation rate found in the counterflow diffusion flame, which occurs at the stagnation point where $Z = 0.5$. (Note that this expression has not yet been extended to multiple mixture fractions, so that this treatment is only applicable for two-stream problems.) The value of χ at any reference location in the flamelet can be similarly approximated, so that $\chi_o = \chi_{\max} F_\chi(Z_o)$. Combining these models by equating the unknown χ_{\max} yields a closed-form expression linking the scalar dissipation rate at any location to the reference value on the flamelet with the same characteristic χ_{\max} ,

$$\chi = \chi_o \frac{F_\chi(Z)}{F_\chi(Z_o)}. \quad (3.357)$$

Applying the filtering operation in (3.354) to both sides of (3.357) for a single-mixture fraction configuration yields

$$\begin{aligned} \tilde{\chi} &= \int_0^\infty \int_0^1 \chi_o \frac{F_\chi(Z)}{F_\chi(Z_o)} p_Z(Z; \tilde{Z}, \widetilde{Z''^2}) p_\chi(\chi; \tilde{\chi}) dZ d\chi \\ &= \frac{\chi_o}{F_\chi(Z_o)} \int_0^\infty p_\chi(\chi; \tilde{\chi}) d\chi \int_0^1 F_\chi(Z) p_Z(Z; \tilde{Z}, \widetilde{Z''^2}) dZ \\ &= \frac{\chi_o}{F_\chi(Z_o)} \int_0^1 F_\chi(Z) p_Z(Z; \tilde{Z}, \widetilde{Z''^2}) dZ, \end{aligned} \quad (3.358)$$

so that the filtered reference scalar dissipation rate can be calculated from the filtered quantities provided by the turbulent flame simulation as

$$\chi_o(\tilde{Z}, \widetilde{Z''^2}, \tilde{\chi}) = \frac{\tilde{\chi} F_\chi(Z_o)}{\int_0^1 F_\chi(Z) p_Z(Z; \tilde{Z}, \widetilde{Z''^2}) dZ}. \quad (3.359)$$

To decrease computational cost, the integral in the denominator can be interpolated from pre-calculated values in a two-dimensional table as a function of \tilde{Z} and $\widetilde{Z''^2}$.

To summarize, the turbulent reacting simulation will query the property table for the variable $\tilde{\phi}(\tilde{Z}_m, \widetilde{Z''^2}, \tilde{\chi})$. Internally, (3.359) will be used to calculate χ_o as a function of the provided filtered independent variables. This value will then be used along with the provided independent variables to interpolate a value for $\tilde{\phi}_T(\tilde{Z}_m, \widetilde{Z''^2}, \chi_o)$ from the stored table that was pre-calculated with (3.355). This interpolated value will then be returned to the main simulation as the requested value for $\tilde{\phi}(\tilde{Z}_m, \widetilde{Z''^2}, \tilde{\chi})$.

If turbulence/chemistry interactions are to be neglected in the simulation, the delta function $\delta(Z - \tilde{Z})$ may be used for $p_Z(Z; \tilde{Z})$ in (3.359) so that the reference scalar dissipation rate can be computed simply as

$$\chi_o(\tilde{Z}, \tilde{\chi}) = \frac{\tilde{\chi} F(Z_o)}{F(\tilde{Z})}. \quad (3.360)$$

Once the multidimensional property table has been generated, it can be imported into `fuego` and queried for the dependent variables as a function of the independent variables \tilde{Z}_m , $\widetilde{Z''^2}$, and $\tilde{\chi}$. Models are required for each of these independent variables used by the flamelet property table. *Filtered Scalar Dissipation Rate* to *Filtered Scalar Variance* present models for each of these quantities for each of the supported turbulence closure models.

Nonadiabatic Property Table Generation

When including the effects of radiative or convective heat losses in a flamelet simulation, additional parameterizations beyond those in the previous section are required. These are the “conserved enthalpy”, h^* and heat loss parameter γ , where the heat loss parameter is defined as $\gamma = h - h^*$. The conserved enthalpy is identical to the traditional enthalpy except that its transport equation omits all source terms (typically due to radiative losses).

This formulation is used as a way to parameterize losses in a manner that is consistent with the opposed diffusion flame burner simulations used to generate the flamelet libraries. In these burner simulations, the inflowing pure stream states are fixed and cannot experience any heat losses; Losses only occur in the interior of the burner, and are represented by γ variation. A range of inflowing pure stream states may also be computed, and are parameterized through h^* variation. In this way, the full range of possible states may be tabulated and retrieved in a fire simulation through values of h and h^* , which are both straightforward to compute.

For turbulent simulations, the Favre-filtered property variable $\tilde{\phi}$ is evaluated as

$$\tilde{\phi}(\tilde{Z}_m, \widetilde{Z''^2}, \tilde{\chi}, \tilde{\gamma}, \tilde{h}^*) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} \int_0^{\infty} \int_0^1 \phi(Z_m, \chi_o, \gamma_o, h_o^*) p_{Z\chi\gamma h^*}(Z_m, \chi, \gamma, h^*; \quad (3.361)$$

$$\tilde{Z}_m, \widetilde{Z''^2}, \tilde{\chi}, \tilde{\gamma}, \tilde{h}^*) dZ_m d\chi d\gamma dh^*, \quad (3.362)$$

where γ_o and h_o^* are reference values of the heat loss parameter and the conserved enthalpy, respectively, to be defined in the following sections. Statistical independence will be assumed between fluctuations of each Z_m component, χ , γ , and h^* , so that

$$\tilde{\phi}(\tilde{Z}_m, \widetilde{Z''^2}, \tilde{\chi}, \tilde{\gamma}, \tilde{h}^*) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} \int_0^{\infty} \int_0^1 \int_0^1 \phi(Z_m, \chi_o, \gamma_o, h_o^*) p_{Z_i}(Z_i; \tilde{Z}_i, \widetilde{Z''^2}) p_{Z_m}(Z_m; \tilde{Z}_m) \quad (3.363)$$

$$p_{\chi}(\chi; \tilde{\chi}) p_{\gamma}(\gamma; \tilde{\gamma}) p_{h^*}(h^*; \tilde{h}^*) dZ_i dZ_{m \neq i} d\chi d\gamma dh^*. \quad (3.364)$$

For the present work, $p_{Z_i}(Z_i; \tilde{Z}_i, \widetilde{Z''^2})$ will be modeled as either a beta PDF or a clipped Gaussian PDF, and $p_{Z_m}(Z_m; \tilde{Z}_m)$, $p_{\chi}(\chi; \tilde{\chi})$, $p_{\gamma}(\gamma; \tilde{\gamma})$, and $p_{h^*}(h^*; \tilde{h}^*)$ will be modeled as the delta functions $\delta(Z_m - \tilde{Z}_m)$, $\delta(\gamma - \tilde{\gamma})$, and $\delta(\chi - \tilde{\chi})$, $\delta(\gamma - \tilde{\gamma})$, and $\delta(h^* - \tilde{h}^*)$, respectively.

The convolution integral in (3.364) would be prohibitively expensive to evaluate each time a value for $\tilde{\phi}$ is needed by a turbulent reacting simulation. Therefore, this integral will be pre-calculated so that each property table query will only involve an interpolation from a table of values.

Storing the final $\tilde{\phi}(\tilde{Z}_m, \widetilde{Z''^2}, \tilde{\chi}, \tilde{\gamma}, \tilde{h}^*)$ values directly is undesirable since the range of possible $\tilde{\chi}$, $\tilde{\gamma}$, and \tilde{h}^* values for each flamelet is different, resulting in a non-orthogonal table. Instead, the values $\tilde{\phi}_T(\tilde{Z}_m, \widetilde{Z''^2}, \chi_o, \gamma_o, h_o^*)$ are stored in an orthogonal table that is indexed by \tilde{Z}_m , $\widetilde{Z''^2}$, $\chi_o(\tilde{Z}_i, \widetilde{Z''^2}, \tilde{\chi})$, $\gamma_o(\tilde{Z}_i, \widetilde{Z''^2}, \tilde{\gamma})$, and $h_o^*(\tilde{Z}_i, \tilde{h}^*)$. These tabulated values are calculated by

$$\tilde{\phi}_T(\tilde{Z}_m, \widetilde{Z''^2}, \chi_o, \gamma_o, h_o^*) = \int_0^1 \int_0^1 \phi(Z_m, \chi_o, \gamma_o, h_o^*) p_{Z_i}(Z_i; \tilde{Z}_i, \widetilde{Z''^2}) p_{Z_m}(Z_m; \tilde{Z}_m) dZ_i dZ_{m \neq i}. \quad (3.365)$$

The required reference values of γ_o and h_o^* are described in the following sections.

Boundary heat loss

The addition of a temperature boundary condition on the wall requires a modification of the flamelet formulation of Eq. *Laminar Flamelet Turbulent Combustion Model*. The equation for a normalized temperature variable, $\theta = T/T_{\text{ox}} - 1$, is

$$\partial_t (\rho\theta) + \nabla \cdot (\rho \mathbf{u} \theta) - \nabla \cdot (\lambda \nabla \theta) = L_D \quad (3.366)$$

$$\theta = \dot{\omega}_T(\theta, \vec{Y}) \text{ in } \Omega \quad (3.367)$$

$$\theta(x \in \partial\Omega_{\text{fuel}}) = \theta_{\text{fuel}}, \quad (3.368)$$

$$\theta(x \in \partial\Omega_{\text{ox}}) = 0, \quad (3.369)$$

$$\theta(x \in \partial\Omega_{\text{wall}}) = \theta_{\text{wall}}, \quad (3.370)$$

which now has an extra boundary term $\theta(x \in \partial\Omega_{\text{wall}}) = \theta_{\text{wall}}$. The extra boundary condition remains when we apply the flamelet approximation, leaving $\theta(Z = Z_{\text{wall}}) = \theta_{\text{wall}}$. The value of the mixture fraction at the wall, however, is undecided: we only know that $\nabla Z \cdot n = 0$. During the simulation, the value of mixture fraction directly evaluated at the wall can be determined dynamically and the value of temperature can be computed. Away from the wall, however, one in principle would need to follow the ζ coordinate from the flamelet transformation until it intersects the wall. However, given that $\nabla Z \cdot n = 0$, the gradient trajectory in principle is tangential to the wall. Although the flamelet equation itself is well-posed, the asymptotic derivation of the flamelet model in the very near region to a nonadiabatic wall and the equations need to be modified in some fashion to account.

Flamelets can readily be described when they are adiabatic; in the limit of unity Lewis numbers and adiabatic systems the enthalpy is a linear function of the mixture fraction. The existence of radiative transport and wall heat transfer introduces deviations from this linear relationship between h and Z . Heat losses at the predominant boundary temperature are a common scenario. Defining a reference boundary temperature at $T_{\text{fr}}(Z) = (1 - Z)T_{\text{ox}} + ZT_{\text{fuel}}$, then this case a simplified flamelet temperature equation with heat losses could be written as

$$\rho \partial_t T - \frac{1}{2} \rho \chi \partial_Z^2 T = \dot{\omega}_T(\vec{\Phi}) - H_T(T - T_{\text{fr}}) \quad (3.371)$$

where H_T represents a heat transfer coefficient that will be further discussed below. This gives a heat loss term that is linear in T . Alternately, the heat loss can be written specifically for radiative-style losses, $\dot{q}_{\text{losses}} = \sigma(\vec{\Phi})(T^4 - T_{\text{fr}}^4)$. Regardless, with heat loss expressed in terms of T_{fr} the flamelet enthalpy is no longer linear in Z but instead takes on a roughly inverted triangular form with an extrema at the peak temperature, roughly Z_{st} . This has led us to express the difference between the adiabatic enthalpy, defined as $h_c = h_{\text{ox}}(1 - Z) + h_{\text{fuel}}Z$, and the actual flamelet computed enthalpy, h , as $\gamma = h - h_c$. The introduction of γ is done strictly as an expedient for generation of flamelet libraries. By assuming a triangular form (or any particular assumed form) we can stretch the table entries into a square format by tabulating as a function of the stoichiometric value of γ_{st} . This does require the use of the assumed form for γ for converting from the local Z value of $\gamma = h(Z) - h_c(Z)$ to γ_{st} . Comparison with DNS and unsteady flamelets for laminar flames shows good agreement with this type of enthalpy defect model for radiation in unity Lewis number flames (which is an appropriate assumption for turbulent, hydrocarbon fires) [56]. We make an assumption of path independence for the solution of at particular integrated heat loss, but in reality the solution will depend somewhat on the value of H_T and the form of the added heat loss term.

The compensation for boundary heat loss can be extended to a full range of temperature (below T_{fr}) in the flamelet libraries by not only including an integrated heat loss rate from the flamelets, but also a translation of the flamelet in enthalpy space. This translation is simply denoted h^* , where the conserved enthalpy line is shifted as $h_c = h_{\text{ox}}(1 - Z) + h_{\text{fuel}}Z + h^*$. This allows a full

description of wall boundary heat loss. Having two heat loss parameterizations, however, makes the lookup procedure non-unique, requiring a method for deciding which point on (γ, h^*) to use for the flamelet lookup. We prefer γ and use γ as much as possible. When γ is insufficient, which would be the case for overly cold or hot walls (wall temperatures outside of the range of temperatures spanned by the solutions of Eq. (3.371)). At the wall boundaries, the conserved enthalpy is defined to not be affected by heat loss while the true enthalpy is, providing γ at the wall.

Property Table Heat Loss Parameterization

For nonadiabatic flamelet library generation and tabulation, a functional form for the heat loss parameter γ in terms of reference quantities is required, similar in concept to the form of χ in (3.356). The value of γ must be zero in each of the pure streams, and should have a maximum value near the stoichiometric flame sheet since this quantity typically represents radiative losses to the environment. A piecewise linear functional form is selected for simplicity. For a single mixture fraction, this form is simply

$$\gamma = \gamma_o F_\gamma(Z, Z_o), \quad (3.372)$$

where γ_o is a reference heat loss at reference state Z_o (selected to be the stoichiometric condition Z_{st}) and the nondimensional function $F_\gamma(Z, Z_o)$ is defined as

$$F_\gamma(Z, Z_o) = \begin{cases} \frac{Z}{Z_o} Z \leq Z_o & : \\ \frac{1-Z}{1-Z_o} Z > Z_o & : \end{cases} \quad (3.373)$$

For multiple mixture fractions, γ is calculated by

$$\gamma = \gamma_o F_\gamma(Z_m, Z_{o,km}, \gamma_{o,k}^{\max}), \quad (3.374)$$

where γ_o is the maximum-magnitude reference heat loss in the vector $\gamma_{o,k}^{\max}$, which contains the reference heat loss parameters corresponding to maximum thermal losses for the K stoichiometric mixture fractions that can be defined between stream pairs, $Z_{o,km}$. The multiple stoichiometric mixture fractions are necessary because a single unique stoichiometric mixture fraction does not exist when using multiple mixture fractions.

The functional form for F_γ is quite complex for multiple mixture fractions, and will only be described briefly here. In general, for a three-stream problem, there are two independent mixture fractions and the realizable mixture fraction space is the triangle where the two mixture fractions sum to a value less than or equal to unity. The value of F_γ must be zero at the “corners” of this space, where the coordinates are $(0, 0)$, $(0, 1)$, and $(1, 0)$. The multiple stoichiometric mixture fractions between stream pairs will define points along the boundaries of this realizable mixture fraction space that represent local maxima in the heat loss distribution along that boundary. Straight lines may be used to connect these points in mixture fraction space, forming a “ridge” in the multidimensional F_γ distribution. When definable, a linear fit is used between this ridge and a corner where γ is zero. When not uniquely definable, linear fits are used between the ridge and

the adjacent boundary value along rays extended from the opposite corner of the state space. Note that the values $\gamma_{o,k}$ are required for the calculation of F_γ so that the final function may be normalized to a unity maximum value with appropriate relative scaling between the boundary heat loss values. Note that no more than a three-stream configuration is currently supported by `fuego_tabular_props`.

Applying the filtering operation in (3.364) to both sides of (3.374) yields

$$\begin{aligned}
\tilde{\gamma} &= \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} \int_0^{\infty} \int_0^1 \int_0^1 \gamma_o F_\gamma(Z_m, Z_{o,km}, \gamma_{o,k}^{\max}) p_{Z_i}(Z_i; \tilde{Z}_i, \widetilde{Z''^2}) p_{Z_m}(Z_m; \tilde{Z}_m) p_\chi(\chi; \tilde{\chi}) p_\gamma(\gamma; \tilde{\gamma}) p_{h^*}(h^*; \tilde{h}^*) dZ_i dZ_m \\
&= \gamma_o \int_{-\infty}^{\infty} p_{h^*}(h^*; \tilde{h}^*) dh^* \int_{-\infty}^{\infty} p_\gamma(\gamma; \tilde{\gamma}) d\gamma \int_0^{\infty} p_\chi(\chi; \tilde{\chi}) d\chi \int_0^1 \int_0^1 F_\gamma(Z_m, Z_{o,km}, \gamma_{o,k}^{\max}) p_{Z_i}(Z_i; \tilde{Z}_i, \widetilde{Z''^2}) p_{Z_m}(Z_m; \tilde{Z}_m) dZ_i dZ_m \\
&= \gamma_o \int_0^1 \int_0^1 F_\gamma(Z_m, Z_{o,km}, \gamma_{o,k}^{\max}) p_{Z_i}(Z_i; \tilde{Z}_i, \widetilde{Z''^2}) p_{Z_m}(Z_m; \tilde{Z}_m) dZ_i dZ_{m \neq i},
\end{aligned} \tag{3.375}$$

so that the filtered heat loss parameter can be calculated from the filtered quantities provided by the turbulent flame simulation as

$$\gamma_o(\tilde{Z}_m, \widetilde{Z''^2}, \tilde{\gamma}) = \frac{\tilde{\gamma}}{\int_0^1 \int_0^1 F_\gamma(Z_m, Z_{o,km}, \gamma_{o,k}^{\max}) p_{Z_i}(Z_i; \tilde{Z}_i, \widetilde{Z''^2}) p_{Z_m}(Z_m; \tilde{Z}_m) dZ_i dZ_{m \neq i}}. \tag{3.376}$$

To decrease computational cost, the integral in the denominator can be interpolated from pre-calculated values in a multi-dimensional table as a function of \tilde{Z}_m and $\widetilde{Z''^2}$. (3.376) can be used during a simulation to convert filtered independent variables to the reference heat loss parameter required to perform table lookups to retrieve $\tilde{\phi}_T$.

If turbulence/chemistry interactions are to be neglected in the simulation, the delta function $\delta(Z_i - \tilde{Z}_i)$ may be used for $p_{Z_i}(Z_i; \tilde{Z}_i, \widetilde{Z''^2})$ in (3.376) so that the reference heat loss parameter can be computed simply as

$$\gamma_o(\tilde{Z}_m, \tilde{\gamma}) = \frac{\tilde{\gamma}}{F_\gamma(\tilde{Z}_m, Z_{o,km}, \gamma_{o,k}^{\max})}. \tag{3.377}$$

Property Table Conserved Enthalpy Parameterization

For nonadiabatic flamelet library generation and tabulation, a functional form for the conserved enthalpy h^* in terms of reference quantities is required. The value of h^* should vary linearly within the range provided for each of the pure streams as a function of a reference heat loss parameter h_o^* , with an appropriate stream-weighted blending for all other compositions.

The stream-weighted mixture properties are computed with an augmented mixture fraction vector Z'_n in terms of Z_m ,

$$Z'_n = \left[Z_1, Z_2, \dots, Z_M, 1 - \sum_{m=1}^M Z_m \right], \quad (3.378)$$

where the last component is simply the last implied mixture fraction to recover a unity sum. A reference augmented mixture fraction is defined as the centroid of the realizable mixture fraction space with each component being identical and equal to

$$Z'_{o,n} = \frac{1}{N}. \quad (3.379)$$

From these definitions, minimum and maximum reference conserved enthalpy values may be computed as

$$h_{o,\min}^* = \sum_{n=1}^N h_{\text{stream},\min,n}^* Z'_{o,n} \quad (3.380)$$

$$h_{o,\max}^* = \sum_{n=1}^N h_{\text{stream},\max,n}^* Z'_{o,n}, \quad (3.381)$$

where $h_{\text{stream},\min,n}^*$ and $h_{\text{stream},\max,n}^*$ are vectors of the minimum and maximum conserved enthalpy in pure stream n , respectively. The conserved enthalpy can then be modeled as

$$h^* = h_{\min,Z}^* + \left(h_o^* - h_{o,\min}^* \right) a_Z, \quad (3.382)$$

where the mixture-weighted minimum conserved enthalpy is

$$h_{\min,Z}^* = \sum_{n=1}^N h_{\text{stream},\min,n}^* Z'_n \quad (3.383)$$

and the mixture-weighted stream variation proportionality constant is

$$a_Z = \sum_{n=1}^N \left(\frac{h_{\text{stream},\max,n}^* - h_{\text{stream},\min,n}^*}{h_{o,\max}^* - h_{o,\min}^*} \right) Z'_n. \quad (3.384)$$

Applying the filtering operation in (3.364) to both sides of (3.382) yields

$$\tilde{h}^* = \tilde{h}_{\min,Z}^* + \left(h_o^* - h_{o,\min}^* \right) \tilde{a}_Z, \quad (3.385)$$

where the two mixture-weighted quantities are now expressed in terms of the augmented filtered mixture fraction as

$$\tilde{h}_{\min,Z}^* = \sum_{n=1}^N h_{\text{stream},\min,n}^* \tilde{Z}'_n \quad (3.386)$$

and

$$\tilde{a}_Z = \sum_{n=1}^N \frac{h_{\text{stream,max},n}^* - h_{\text{stream,min},n}^*}{h_{o,\text{max}}^* - h_{o,\text{min}}^*} \tilde{Z}'_n. \quad (3.387)$$

This allows the reference conserved enthalpy to be expressed in terms of the filtered quantities provided by the turbulent flame simulation as

$$h_o^*(\tilde{Z}_m, \tilde{h}^*) = h_{o,\text{min}}^* + \frac{\tilde{h}^* - \tilde{h}_{\text{min},Z}^*}{\tilde{a}_Z}. \quad (3.388)$$

Filtered Scalar Dissipation Rate

RANS Model

For RANS turbulence closure models the instantaneous laminar scalar dissipation rate given in (3.343) can be Favre-filtered and expanded to the form

$$\begin{aligned} \bar{\rho} \tilde{\chi} &= 2\bar{\rho} D \frac{\partial \tilde{Z}}{\partial x_i} \frac{\partial \tilde{Z}}{\partial x_i} \\ &= 2\bar{\rho} D \frac{\partial \tilde{Z}}{\partial x_i} \frac{\partial \tilde{Z}}{\partial x_i} + 4\bar{\rho} D \frac{\partial \tilde{Z}''}{\partial x_i} \frac{\partial \tilde{Z}}{\partial x_i} + 2\bar{\rho} D \frac{\partial \tilde{Z}''}{\partial x_i} \frac{\partial \tilde{Z}''}{\partial x_i}. \end{aligned} \quad (3.389)$$

The middle term on the RHS is neglected for constant density flow [57]. The first term is referred to as the mean scalar dissipation rate

$$\bar{\rho} \tilde{\chi}_m = 2\bar{\rho} D \frac{\partial \tilde{Z}}{\partial x_i} \frac{\partial \tilde{Z}}{\partial x_i} \quad (3.390)$$

and the third term is the perturbation scalar dissipation rate $\bar{\rho} \tilde{\chi}_p$. This term can be modeled as

$$\bar{\rho} \tilde{\chi}_p = 2\bar{\rho} D \frac{\partial \tilde{Z}''}{\partial x_i} \frac{\partial \tilde{Z}''}{\partial x_i} \quad (3.391)$$

$$\approx C_\chi \bar{\rho} \frac{\epsilon}{k} \widetilde{Z''^2} \quad (3.392)$$

for RANS-based turbulence closures where $\frac{\epsilon}{k}$ provides an inverse turbulence time scale, $\widetilde{Z''^2}$ is the scalar variance that will be modeled in *Filtered Scalar Variance*, and C_χ is a model constant that typically has a value of 2.0. [54]

Expressing the molecular mass diffusivity as $\bar{\rho} D = \mu / \text{Sc}$, where μ is the molecular viscosity and Sc is the Schmidt number, the modeled total filtered scalar dissipation rate for RANS closures is

$$\begin{aligned} \tilde{\chi} &= \tilde{\chi}_m + \tilde{\chi}_p \\ &\approx \frac{2}{\bar{\rho}} \frac{\mu}{\text{Sc}} \frac{\partial \tilde{Z}}{\partial x_i} \frac{\partial \tilde{Z}}{\partial x_i} + C_\chi \frac{\epsilon}{k} \widetilde{Z''^2}. \end{aligned} \quad (3.393)$$

LES Model

For LES closures (3.393) also applies, so that the total filtered scalar dissipation rate is the sum of the mean and the perturbation scalar dissipation rates. The mean scalar dissipation rate is expressed identically to RANS closures as

$$\tilde{\chi}_m = \frac{2}{\bar{\rho}} \frac{\mu}{Sc} \frac{\partial \tilde{Z}}{\partial x_i} \frac{\partial \tilde{Z}}{\partial x_i}. \quad (3.394)$$

The perturbation scalar dissipation rate $\tilde{\chi}_p$ represents the sub-filter dissipation of scalar variance, and can be modeled by assuming that sub-filter dissipation is in local equilibrium with sub-filter production, and that the sub-filter production can be modeled with a gradient transport assumption as [58]

$$\begin{aligned} \bar{\rho} \tilde{\chi}_p &= 2\rho D \overline{\frac{\partial Z''}{\partial x_i} \frac{\partial Z''}{\partial x_i}} = -2\overline{\rho u_i'' Z''} \frac{\partial \tilde{Z}}{\partial x_i} \\ \bar{\rho} \tilde{\chi}_p &\approx 2 \frac{\mu_t}{Sc_t} \frac{\partial \tilde{Z}}{\partial x_i} \frac{\partial \tilde{Z}}{\partial x_i}, \end{aligned} \quad (3.395)$$

where μ_t is the modeled turbulent eddy viscosity and Sc_t is the turbulent Schmidt number.

This results in the final modeled form for the filtered total scalar dissipation rate for LES closures,

$$\begin{aligned} \tilde{\chi} &= \tilde{\chi}_m + \tilde{\chi}_p \\ &\approx \frac{2}{\bar{\rho}} \left(\frac{\mu}{Sc} + \frac{\mu_t}{Sc_t} \right) \frac{\partial \tilde{Z}}{\partial x_i} \frac{\partial \tilde{Z}}{\partial x_i}. \end{aligned} \quad (3.396)$$

Filtered Mixture Fraction

The primary quantity used to identify the chemical state in Flamelet closure models is the mixture fraction, Z . While there are many different definitions of the mixture fraction that have subtle variations that attempt to capture effects like differential diffusion, they can all be interpreted as a local mass fraction of the chemical elements that originated in the fuel stream. [59] The mixture fraction is a conserved scalar that varies between 0 in the oxidizer stream and 1 in the fuel stream, and is transported in laminar flow by the equation

$$\frac{\partial \rho Z}{\partial t} + \frac{\partial \rho u_i Z}{\partial x_i} = \frac{\partial}{\partial x_i} \left(\rho D \frac{\partial Z}{\partial x_i} \right), \quad (3.397)$$

where D is an effective molecular mass diffusivity.

Applying either temporal Favre filtering for RANS-based treatments or spatial Favre filtering for LES-based treatments yields

$$\frac{\partial \bar{\rho} \tilde{Z}}{\partial t} + \frac{\partial \bar{\rho} \tilde{u}_i \tilde{Z}}{\partial x_i} = -\tau_{Z u_j} + \frac{\partial}{\partial x_i} \left(\bar{\rho} D \frac{\partial \tilde{Z}}{\partial x_i} \right), \quad (3.398)$$

where sub-filter correlations have been neglected in the molecular diffusive flux vector [60] and the turbulent diffusive flux vector is defined as

$$\tau_{Zu_j} \equiv \bar{\rho} \left(\widetilde{Zu_j} - \tilde{Z}\tilde{u}_j \right). \quad (3.399)$$

Similar to species transport, this sub-filter correlation is modeled in both RANS and LES closures with the gradient transport approximation

$$\tau_{Zu_j} \approx -\bar{\rho} D_t \frac{\partial \tilde{Z}}{\partial x_j}, \quad (3.400)$$

where D_t is the turbulent mass diffusivity, modeled as $\bar{\rho} D_t = \mu_t / Sc_t$ where μ_t is the modeled turbulent viscosity from momentum transport and Sc_t is the turbulent Schmidt number. Please see the Fuego theory manual for further details. The molecular mass diffusivity is then expressed similarly as $\bar{\rho} D = \mu / Sc$ so that the final modeled form of the filtered mixture fraction transport equation is

$$\frac{\partial \bar{\rho} \tilde{Z}}{\partial t} + \frac{\partial \bar{\rho} \tilde{u}_i \tilde{Z}}{\partial x_i} = \frac{\partial}{\partial x_i} \left[\left(\frac{\mu}{Sc} + \frac{\mu_t}{Sc_t} \right) \frac{\partial \tilde{Z}}{\partial x_i} \right]. \quad (3.401)$$

In integral form as used in Fuego, the mixture fraction transport equation is

$$\int \frac{\partial \bar{\rho} \tilde{Z}}{\partial t} dV + \int \bar{\rho} \tilde{u}_i \tilde{Z} n_i dS = \int \left(\frac{\mu}{Sc} + \frac{\mu_t}{Sc_t} \right) \frac{\partial \tilde{Z}}{\partial x_i} n_i dS. \quad (3.402)$$

Filtered Scalar Variance

RANS Model

For RANS-based turbulence closures, a transport equation is solved for the filtered scalar variance, $\widetilde{Z''^2}$. This equation can be derived by subtracting (3.398) multiplied by \tilde{Z} from the filter of the multiple of (3.397) and Z , yielding

$$\frac{\partial \bar{\rho} \widetilde{Z''^2}}{\partial t} + \frac{\partial}{\partial x_i} \left(\bar{\rho} \tilde{u}_i \widetilde{Z''^2} \right) = -\frac{\partial}{\partial x_i} \left(\overline{\rho u_i'' Z''^2} \right) + \frac{\partial}{\partial x_i} \left(\overline{\rho D \frac{\partial Z''^2}{\partial x_i}} \right) + 2 \widetilde{Z''^2} \frac{\partial}{\partial x_i} \left(\overline{\rho D \frac{\partial \tilde{Z}}{\partial x_i}} \right) - 2 \overline{\rho u_i'' Z''} \frac{\partial \tilde{Z}}{\partial x_i} - 2 \overline{\rho D \frac{\partial Z''}{\partial x_i} \frac{\partial \tilde{Z}}{\partial x_i}} \quad (3.403)$$

where the filtered mixture fraction variance is defined as $\widetilde{Z''^2} \equiv \widetilde{Z^2} - \tilde{Z}^2$.

All five terms on the RHS of (3.403) require closure models. The first term represents turbulent transport of mixture fraction variance, and is modeled by a gradient-transport assumption as

$$-\overline{\rho u_i'' Z''^2} \approx \frac{\mu_t}{Sc_t} \frac{\partial \widetilde{Z''^2}}{\partial x_i}. \quad (3.404)$$

The second and third terms on the RHS of (3.403) taken together represent molecular diffusion of mixture fraction variance, and is typically neglected with respect to turbulent transport for

sufficiently high Reynolds numbers. Its effects are included here with another gradient-transport assumption of the form

$$\frac{\partial}{\partial x_i} \left(\overline{\rho D \frac{\partial Z''^2}{\partial x_i}} \right) + 2 \overline{Z''^2} \frac{\partial}{\partial x_i} \left(\overline{\rho D \frac{\partial \tilde{Z}}{\partial x_i}} \right) \approx \frac{\partial}{\partial x_i} \left(\frac{\mu}{Sc} \frac{\partial \widetilde{Z''^2}}{\partial x_i} \right). \quad (3.405)$$

The fourth and fifth terms on the RHS of (3.403) represent production and dissipation of mixture fraction variance, respectively. The production term is similarly modeled with a gradient transport assumption as

$$-2 \overline{\rho u_i'' Z''} \frac{\partial \tilde{Z}}{\partial x_i} \approx 2 \frac{\mu_t}{Sc_t} \frac{\partial \tilde{Z}}{\partial x_i} \frac{\partial \tilde{Z}}{\partial x_i}. \quad (3.406)$$

The mixture fraction variance dissipation rate term is equal to the perturbation scalar dissipation rate,

$$2 \overline{\rho D \frac{\partial Z''}{\partial x_i} \frac{\partial Z''}{\partial x_i}} = \bar{\rho} \tilde{\chi}_p, \quad (3.407)$$

previously defined in (3.391) and modeled in (3.392). An identical treatment of this term is used here.

The final modeled form of the filtered scalar variance transport equation for RANS turbulence closure models is

$$\frac{\partial \bar{\rho} \widetilde{Z''^2}}{\partial t} + \frac{\partial}{\partial x_i} \left(\bar{\rho} \tilde{u}_i \widetilde{Z''^2} \right) = \frac{\partial}{\partial x_i} \left[\left(\frac{\mu}{Sc} + \frac{\mu_t}{Sc_t} \right) \frac{\partial \widetilde{Z''^2}}{\partial x_i} \right] + 2 \frac{\mu_t}{Sc_t} \frac{\partial \tilde{Z}}{\partial x_i} \frac{\partial \tilde{Z}}{\partial x_i} - \bar{\rho} \tilde{\chi}_p. \quad (3.408)$$

LES Model

For LES turbulence closures, the filtered scalar variance $\widetilde{Z''^2}$ can be modeled with the scaling law [58]

$$\bar{\rho} \widetilde{Z''^2} \approx C_V \bar{\rho} \Delta^2 \frac{\partial \tilde{Z}}{\partial x_i} \frac{\partial \tilde{Z}}{\partial x_i}, \quad (3.409)$$

where Δ is a length scale corresponding to the grid filter size and C_V is a model coefficient. For the k^{sgs} closure and the non-dynamic Smagorinsky closure, C_V has a fixed value of 0.5. For the dynamic Smagorinsky LES closure, C_V can be dynamically calculated based on the local instantaneous flow-field.

To dynamically evaluate the filtered scalar variance model coefficient, begin by defining the grid filter-scale correlation

$$\begin{aligned} \tau_{Z''^2} &\equiv \bar{\rho} \widetilde{Z''^2} \\ &= \bar{\rho} \widetilde{Z^2} - \bar{\rho} \tilde{Z}^2 \\ &= \overline{\rho Z^2} - \frac{\left(\overline{\rho Z} \right)^2}{\bar{\rho}}. \end{aligned} \quad (3.410)$$

Similarly, define an equivalent correlation at a larger test-filter scale

$$T_{Z''2} \equiv \overline{\overline{\rho Z^2}} - \frac{\left(\overline{\overline{\rho Z}}\right)^2}{\hat{\rho}}. \quad (3.411)$$

Now, define the quantity $L_{Z''2}$ as a combination of these two correlations which reduces to an expression that can be evaluated in closed form,

$$L_{Z''2} \equiv T_{Z''2} - \widehat{\tau_{Z''2}} \quad (3.412)$$

$$L_{Z''2} = \widehat{\overline{\overline{\rho \tilde{Z}^2}}} - \frac{\left(\widehat{\overline{\overline{\rho \tilde{Z}}}}\right)^2}{\hat{\rho}}. \quad (3.413)$$

By modeling the two correlations in (3.412) and equating them to (3.413), the model coefficient C_V can be dynamically evaluated. The correlations at the two filter scales are modeled analogously as

$$\tau_{Z''2} \approx C_V \bar{\rho} \Delta^2 \left(\frac{\partial \tilde{Z}}{\partial x_i} \right)^2 \quad (3.414)$$

$$T_{Z''2} \approx C_V \hat{\rho} \hat{\Delta}^2 \left[\frac{\partial}{\partial x_i} \left(\frac{\widehat{\overline{\overline{\rho \tilde{Z}}}}}{\hat{\rho}} \right) \right]^2, \quad (3.415)$$

where $\hat{\Delta}$ is the characteristic test filter length scale and C_V is assumed to be the same at both scales.

Notice that when the modeled forms of $\tau_{Z''2}$ and $T_{Z''2}$ are inserted into (3.412), C_V appears inside a test filtering operation. Formally solving this system of equations for C_V requires the expensive solution of an additional set of coupled integro-differential equations [61]. Alternatively, it is common practice to remove C_V from the test filter with the assumption that it is varying slowly over distances on the order of the test filter size. This greatly simplifies calculations, although it can result in non-physical oscillations in the modeled value for C_V . The square of the error involved in this approximation is $Q = (L_{Z''2} - C_V M_{Z''2})^2$, where

$$L_{Z''2} = \widehat{\overline{\overline{\rho \tilde{Z}^2}}} - \frac{\left(\widehat{\overline{\overline{\rho \tilde{Z}}}}\right)^2}{\hat{\rho}} \quad (3.416)$$

$$M_{Z''2} = \hat{\rho} \hat{\Delta}^2 \left[\frac{\partial}{\partial x_i} \left(\frac{\widehat{\overline{\overline{\rho \tilde{Z}}}}}{\hat{\rho}} \right) \right]^2 - \widehat{\overline{\overline{\rho \Delta^2 \left(\frac{\partial \tilde{Z}}{\partial x_i} \right)^2}}}. \quad (3.417)$$

Minimizing this error in a least-squares fashion with respect to C_V yields an expression for the modeled coefficient,

$$C_V = \frac{L_{Z''2} M_{Z''2}}{M_{Z''2} M_{Z''2}}, \quad (3.418)$$

that can be used directly in (3.409) for the filtered scalar variance.

Due to the above simplifications, the model coefficient C_V can sometimes fluctuate wildly, possibly leading to numerical instabilities. A common solution to control these oscillations, and the one that is taken here, is to pass the numerator and denominator of (3.418) through a test filter, yielding

$$C_V = \frac{\overline{L_{Z''2} M_{Z''2}}}{\overline{M_{Z''2} M_{Z''2}}}. \quad (3.419)$$

This can be crudely justified by recognizing that C_V was already assumed to vary slowly over distances equal to the test filter size, so that this filtering operation is simply enforcing that assumption.

Turbulent Reacting Mixing Models

In a reacting flow it is sometimes necessary to limit the finite-rate chemistry reaction when it occurs in regions of high turbulence. In this case, turbulent mixing of the reactants limits the finite-rate (laminar) chemical reaction, the time scale of which can be calculated as

$$t_c = N_R \frac{\sum_{i=1}^{N_S} Y_i}{\sum_{i=1}^{N_S} \frac{dY_i}{dt}} \quad (3.420)$$

where N_R is the number of reactions, N_S is the number of species in the reaction, and Y_i represents the concentration of species i . The denominator in (3.420) is the production rate of species i .

Modified EDC Model (PARENTE)

One such model, named here as the PARENTE model, is adopted from [62] which derives explicit dependencies between the EDC model coefficients and the turbulent Reynolds (Re_t) and Damkohler (Da) dimensionless numbers. This model is based on EDC and is used to modify the chemistry reactions prescribed in the gas phase. The PARENTE turbulent reacting mixing model can be used in addition to or in lieu of the built-in EDC turbulent combustion model found in *EDC Turbulent Combustion Model*. The current model is described as

$$Da = \max \left(\min \left(\frac{1}{t_c} \sqrt{\frac{\nu}{\epsilon}}, 10 \right), 1 \times 10^{-10} \right) \quad (3.421)$$

$$Re_t = \frac{k^2}{\nu \epsilon} \quad (3.422)$$

$$C_\tau = \min \left(\frac{C_1}{\sqrt{Da(Re_t + 1)}}, 2.1377 \right) \quad (3.423)$$

$$C_\gamma = \max \left(\min \left(C_2 \sqrt{Da(Re_t + 1)}, 5.0 \right), 0.4082 \right) \quad (3.424)$$

$$\gamma_L = C_\gamma \left(\frac{\nu \epsilon}{k^2} \right)^{\frac{1}{4}} \quad (3.425)$$

$$\tau^* = C_\tau \sqrt{\frac{\nu}{\epsilon}} \quad (3.426)$$

where k represents the turbulent kinetic energy, ϵ is the turbulent dissipation and ν is the kinematic viscosity of the gas. The constants C_1 and C_2 can be specified by the user, but default to $C_1 = 0.05774$ and $C_2 = 0.5$. The modified reaction time scale, τ^* is used to solve the reaction equation from time t^n to $t^{n+\tau^*}$. Following this calculation, the reaction source terms for species ($S_{Y_i}^*$) and enthalpy (S_H^*) can be computed. These computed sources are further scaled via the multiplication of a coefficient κ which is calculated as

$$\kappa = 1 \quad \text{if } \gamma_L \geq 1$$

$$\kappa = \max \left(\min \left(\frac{\gamma_L^{e_1}}{1 - \gamma_L^{e_2}}, 1 \right), 0 \right), \quad \text{otherwise} \quad (3.427)$$

where $e_1 = e_2 = 3.0$ in our implementation of the model. Following the evaluation of κ , the final source terms for species and enthalpy can be calculated as

$$S_{Y_i} = \kappa S_{Y_i}^* \quad (3.428)$$

$$S_H = \kappa S_H^* \quad (3.429)$$

respectively.

Soot Generation Model for Multicomponent Combustion

Soot is an important contributor to radiative exchange within a fire and between a fire and its surroundings. Soot production, destruction and transport at flame scales are still active areas of research, with important chemical/physical processes not understood from a fundamental physics point of view. Basically, soot particles are carbon-rich solid particles generated in regions of excess pyrolyzate, such as on the rich side of a diffusion flame. Unagglomerated soot particles have characteristic dimensions in the range 0.01–0.05 μ .

{it The main purpose of the soot model is for the calculation of the absorption coefficient} in the radiant energy transfer equation. For the current implementation we employ the soot model implemented in the KAMELEON code because it has been used for large turbulent fire calculations with participating media radiation. The model is discussed in Magnussen et al. [44] and Magnussen and Hjertager [63] It is a two-step formulation, first described by Tesner et al. [64]. The model for generation and combustion of soot can be summarized by three principal steps: 1) particle nucleation, where the first solid soot particles (often called radical nuclei) are created as a result of fuel oxidation and pyrolysis, 2) particle growth, whereby the soot particle size increases due to the addition of material which is primarily carbon (10–20% mole fraction hydrogen) through a series of reactions and coagulation, 3) particle oxidation, where soot particles are burned. Additional information is provided in the overview by Haynes and Wagner [65].

Since the soot model is primarily directed at closing emission/absorption terms in the radiative transfer equation, engineering approximations are made with respect to its inclusion in the Navier Stokes equations. Specifically, heats of reaction associated with formation and destruction are not accounted for in the heat balance, and the mass concentrations of soot and radical nuclei are not included in the species mass balances; they are treated as tracers. The model has a significant amount of empiricism associated with it, necessitated by the extreme length scale range of soot processes, its complexity, and the degree to which many processes have yet to be quantified from a first principles perspective. The model choice can be considered to be a pragmatic one based on its prior use in fire calculations.

The present model has been constructed to fit into the same framework as the conceptual model for turbulent combustion outlined in the theory section for the EDC model. In the following subsections, the basic mechanisms of soot formation and destruction are presented. These processes occur on a scale smaller than can be resolved numerically, therefore the following subsections present the basic approach to the subgrid modeling of the elementary mechanisms, suitable for use in a numerical model.

EDC Soot Model

It is important to note that the processes of turbulent soot formation and combustion occur on a scale smaller than can be resolved in a numerical approximation. Thus, the averaged governing equations to be solved numerically must be supplemented with subgrid models to account for these subgrid processes. The conceptual model for subgrid turbulent soot generation and combustion is consistent with the two-zone, turbulent, gas-phase, combustion model presented in the last section (see also Holen [53]). One zone is the flame zone (flame structure) and the other is the surrounding zone.

Soot reactions tend to be slower than gas phase hydrocarbon chemistry. Therefore, the infinitely fast chemistry limit used for the gas phase chemistry is not employed for soot. The current model assumes that the formation and combustion rates are long compared to turbulent mixing rates at flame scales. A steady-state, steady-flow assumption is used in the formulation between the production/destruction rates and the turbulent mixing rates to obtain the soot mass fraction in the flame zone in an algebraic manner (avoiding solution of stiff ordinary differential rate equations).

Criteria for Soot and Radical Nuclei Formation

To start, the first level criteria for formation of soot are

$$Y_{prod} > Y_{lim} \quad \text{and} \quad \gamma\chi > 0 \quad \text{and} \quad T^\circ > T_{lim}, \quad (3.430)$$

where Y_{prod} denotes the mass fraction of products, Y_{lim} and T_{lim} are minimum values of product mass fraction and surrounding temperatures allowing soot generation, and $\gamma\chi$ is the volume fraction of the reaction zone of the current cell. If these conditions are met, then the first step is to

determine how much carbon is available over and above what may potentially react with oxygen to produce CO₂, via the 2-step reaction postulated in the chemistry model (see *EDC Turbulent Combustion Model*). So, first form the elemental mass fraction of excess (over what may potentially form CO₂) carbon in each species,

$$f_{c,i} = \max \left[0, \left(\hat{Y}_i^C - \frac{1}{2} \frac{W_C}{W_O} \hat{Y}_i^O \right) \right], \quad (3.431)$$

where \hat{Y}_i^C is the mass fraction of carbon in species i , and \hat{Y}_i^O the mass fraction of elemental oxygen in species i . For example, for CO (carbon monoxide), $\hat{Y}_{CO}^C = 12/(12 + 16)$, etc. Also, for CO₂, the excess fraction $f_{c,co2} = 0$, while for any species containing oxygen but no carbon, the formula for the excess fraction is constructed to give zero. Hence, the fraction is non-zero only for species containing carbon but excluding carbon dioxide; i.e., the fuel and carbon monoxide species will have non-zero excess carbon fraction. With the 2-step reaction process being considered, the CO can be considered a fuel in the second reaction, in which CO and H₂ are oxidized if enough oxygen is available after the first reaction. Thus, the computed carbon fraction, $f_{c,i}$, is collectively the available carbon in the “fuel species”, comprised of the actual CHNO fuel and CO, and will be zero for other species (compounds). Note that this fraction excludes the carbon in the species that can potentially form CO₂ via oxidation with the oxygen present in the species itself.

Now, the mass fraction of carbon potentially available to produce soot can be computed for the surrounding and flame zones from the following,

$$Y_{c \rightarrow s}^o = \sum_i f_{c,i} Y_i^o \quad Y_{c \rightarrow s}^* = \sum_i f_{c,i} Y_i^*. \quad (3.432)$$

Again, these mass fractions represent the potentially available carbon in the fuels, separated into flame zone and surroundings, for formation of soot. The average mass fraction of soot-producing-carbon is,

$$Y_{c \rightarrow s} = (\gamma\chi) Y_{c \rightarrow s}^* + (1 - \gamma\chi) Y_{c \rightarrow s}^o. \quad (3.433)$$

Now we must compare the amount of oxidant (not counting oxidant present in the fuel compound) actually available for burning these fuels to produce CO₂; any excess carbon is available to produce additional soot and radical nuclei. The amount of oxygen required to react with {it all} of the available soot-producing-carbon ($Y_{c \rightarrow s}$, which already excludes the oxygen present in the fuel compound) to produce CO₂ is

$$Y_{O_2,max} = 2 \frac{W_O}{W_C} Y_{c \rightarrow s}. \quad (3.434)$$

Now, if we can compare this to how much oxygen is actually available, we can decide how much excess carbon is available to produce soot and radical nuclei. Thus the fraction (molar ratio) of excess carbon for producing soot is determined by subtracting off the amount that will go to stoichiometrically react with the available oxygen to ultimately produce CO₂ in the two-step reaction,

$$\xi_c = \frac{Y_{c \rightarrow s}/W_C - Y_{O_2}/W_{O_2}}{Y_{c \rightarrow s}/W_C} = 1 - \frac{Y_{O_2}}{Y_{O_2,max}} \Rightarrow 1 - \min \left(1, \frac{Y_{O_2}}{Y_{O_2,max}} \right), \quad (3.435)$$

where the last expression is the computational implementation, to take care of “lean” conditions where there is excess oxidizer, and which will result in zero mole fraction of carbon to produce soot.

In other words, it is assumed that for a given fraction of existing soot that gets mixed by turbulence into a flame zone, a fraction ξ_c , will contribute to the growth of soot in the flame zone, while the balance, $(1 - \xi_c)$ will be consumed in the production of CO_2 . Implicit in this assumption is that soot entering a flame will be consumed in proportion to the oxygen present. Therefore in fuel lean regions, soot entering flame zones will be preferentially destroyed.

Now we are in a position to determine whether soot and radical nuclei can be formed under present conditions. They will form if

$$Y_{c \rightarrow s} > Y_{soot} \quad X_{c,soot} > 0. \quad (3.436)$$

The first inequality in (3.436) asserts that the available potential-soot-producing carbon in the fuel must exceed the present amount of soot before enabling generation of additional soot. The construction of $Y_{c \rightarrow s}$ sums the total potential soot-producing-carbon, without distinguishing whether the carbon exists as soot or fuel. The second requires enough carbon to exceed the requirements for the combustion reaction; i.e., soot will only be formed under fuel rich conditions.

Soot Formation and Termination Models

In general, soot may be considered to be generated in both the reaction zone and in the surrounding zone. This was the assumption invoked in KAMELEON-II (Holen, et al. [53]). As we shall see, in the present implementation for multicomponent species problems, formation/destruction is assumed to take place only in the surrounding fluid. The mass fraction of fuel in the reaction zone is assumed to be proportional to the mass fraction γ^* , and the reacting fraction of the fuel in the reaction zone, χ . The total rate of radical nuclei formation and destruction is given by a volume averaged sum of the formation within the reaction zone and the surrounding zone.

Assuming the conditions in (3.436) are met, the rates of formation can be computed. The following models for soot formation and termination were originally described by Tesner et al. [64] and have been subsequently modified by Magnussen and co-workers. The elementary mechanisms (subgrid models for the fire code application) of formation and destruction of radical nuclei was described by Tesner et al. [64] in the form,

$$\dot{R}_n = n_0 + (f - g) n - g_0 N n \quad [\text{particles/s} - \text{m}^3], \quad (3.437)$$

where n_0 is the spontaneous origination rate of radical nuclei in particles/(s-m³) (due to fuel oxidation and fuel pyrolysis), f is the linear branching coefficient (whereby radical nuclei react to create additional radical nuclei), g is the linear termination coefficient (where radical nuclei combine with existing radical nuclei), n is the concentration of radical nuclei in particles/m³, g_0 is the linear coefficient of termination on soot particles (where radical nuclei combine with existing soot particles), and N is the particle concentration of soot particles (assumed to be spherical with

uniform diameter d_p) in particles/m³. The spontaneous origination rate of radical nuclei was given by Tesner as

$$n_0 = 1.08a_0\rho Y_{fuel} \exp\left(-\frac{E}{RT}\right). \quad (3.438)$$

The rate of soot particle formation and destruction was given by Tesner et al. as,

$$\dot{N}_{N,f} = (a - bN) n \quad [\text{particles/s} - \text{m}^3]. \quad (3.439)$$

The parameters appearing in the foregoing, as determined by Tesner et al. [64] and Holen, et al. [53], are given in Table 3.7. Tesner et al. [66] provide additional data for various hydrocarbons. In practice, the variables a and b are scaled (multiplied) by 10^{16} while a_0 is scaled (divided) by 10^{16} thereby effectively reducing the nuclei concentration by this amount.

Table 3.7: Soot model parameters (Tesner et al.(1971); Holen, et al.(1994))

a	$f - g$	g_0	b	E/R	ρ_{soot}	a_0	d_p
[1/s]	[1/s]	[cm ³ /part - s]	[cm ³ /part - s]	[K]	[g/cm ³]	[part/g - s]	[cm]
10^5	10^2	10^{-9}	8×10^{-8}	9×10^4	2.0	12.5×10^{33}	17.85×10^{-7}

The elementary formation/destruction models of Tesner have been modified by Magnussen et al. (Holen, et al. [53]) for application to multicomponent fire simulation problems. First, for implementation into a computer program, transport equations for two field variables, radical nuclei and soot concentrations, are needed. For computational reasons, it is convenient to write all transport equations in a standard form,

$$\int \frac{\partial \rho \phi}{\partial t} dV + \int \rho u_j \phi n_j dS = \int \Lambda \frac{\partial \phi}{\partial x_j} n_j dS + \rho S_\phi, \quad (3.440)$$

written for the arbitrary scalar field, ϕ , which will have units of intensity per unit mass (or be dimensionless, such as a mass fraction). Thus the computational variables for the soot model are, respectively, the radical nuclei concentration and soot mass fraction,

$$\beta = \frac{n}{\rho} \quad \text{and} \quad Y_{soot} = \frac{C_{soot}}{\rho} \quad (3.441)$$

where C_{soot} denotes the mass concentration of soot (kg/m³). In terms of these variables, the spontaneous origination of radical nuclei, as modified by Magnussen et al., is determined from,

$$\beta_0^\circ = 1.08a_0 (Y_{c \rightarrow s}^\circ - Y_{soot}) \exp\left(-\frac{E}{RT^\circ}\right), \quad (3.442)$$

$$\beta_0^* = 1.08a_0 (Y_{c \rightarrow s}^* - Y_{soot}) \exp\left(-\frac{E}{RT^*}\right), \quad (3.443)$$

in units of part/kg-sec, which, when compared to Tesner's form, is seen to have been written in terms of the excess soot-producing carbon, rather than simply being proportional to the fuel

concentration, of which only a fraction is available to produce radical nuclei and soot. Similarly, the linear branching and termination reactions for radical nuclei can be written in the form,

$$\frac{\dot{R}_{n,f-g,mod}^{\circ}}{\rho^{\circ}} = \max(0, f_c^{\circ}) (f - g) \beta^{\circ} \equiv \max(0, f_c^{\circ}) \frac{\dot{R}_{n,f-g}^{\circ}}{\rho^{\circ}}, \quad (3.444)$$

$$\frac{\dot{R}_{n,f-g,mod}^{*}}{\rho^{*}} = \max(0, f_c^{*}) (f - g) \beta^{*} \equiv \max(0, f_c^{*}) \frac{\dot{R}_{n,f-g}^{*}}{\rho^{*}}, \quad (3.445)$$

where the scale factors are defined by,

$$f_c^{\circ} = \frac{(Y_{c \rightarrow s}^{\circ} - Y_{soot})}{Y_{c \rightarrow s}^{\circ}} \quad \text{and} \quad f_c^{*} = \frac{(Y_{c \rightarrow s}^{*} - Y_{soot})}{Y_{c \rightarrow s}^{*}}, \quad (3.446)$$

and represent the fraction of soot-producing carbon available in the surroundings and flame zone, respectively. The present formulation reduces the rates by the fraction of soot-producing carbon over and above that which is already present as soot, represented by the last terms in each equation. In contrast, the bilinear termination term for generation of soot is indirectly modified through the soot mass fraction, which is similarly modified (as will be shown shortly). Therefore, the termination term can simply be expressed in terms of the computational variables as,

$$\frac{\dot{R}_{n,g0}^{*}}{\rho^{*}} = g_0 \frac{\rho^{*} Y_{soot}^{*}}{m_p} \beta^{*} \quad \text{and} \quad \frac{\dot{R}_{n,g0}^{\circ}}{\rho^{\circ}} = g_0 \frac{\rho^{\circ} Y_{soot}^{\circ}}{m_p} \beta^{\circ}, \quad (3.447)$$

in which the soot particle concentration has been expressed in terms of the soot mass fraction and an average mass of a soot particle, m_p (kg),

$$N = \frac{(\rho Y_{soot})}{m_p} \quad (3.448)$$

$$m_p = \rho_{soot} \frac{4}{3} \pi \left(\frac{d_p}{2} \right)^3, \quad (3.449)$$

$$m_p^{\circ} = \frac{b}{a} \rho^{\circ} Y_{c \rightarrow s}^{\circ}, \quad (3.450)$$

$$m_p^{*} = \frac{b}{a} \rho^{*} Y_{c \rightarrow s}^{*}, \quad (3.451)$$

See Table 3.7 for data used in these equations. The generation/destruction term for soot are also modified via the scale factors,

$$\frac{\dot{R}_{soot,form,mod}^{*}}{\rho^{*}} = f_c^{*} m_p \left(a - b \frac{(\rho^{*} Y_{soot}^{*})}{m_p} \right) \beta^{*} \equiv f_c^{*} m_p \frac{\dot{R}_{soot,form}^{*}}{\rho^{*}}, \quad (3.452)$$

$$\frac{\dot{R}_{soot,form,mod}^{\circ}}{\rho^{\circ}} = f_c^{\circ} m_p \left(a - b \frac{(\rho^{\circ} Y_{soot}^{\circ})}{m_p} \right) \beta^{\circ} \equiv f_c^{\circ} m_p \frac{\dot{R}_{soot,form}^{\circ}}{\rho^{\circ}}. \quad (3.453)$$

to be used in the elementary source expression for the flame zone and surroundings.

The production/destruction of soot in the reaction zone should approach zero for $Y_{soot}^* \rightarrow Y_{c \rightarrow s}^*$, since production should cease when the amount of soot equals the maximum available soot-producing-carbon in the reaction zone. This is easier to see by substituting this form into the production term,

$$\frac{\dot{R}_{soot,form,mod}^*}{\rho^*} = f_c^* b (\rho^* Y_{c \rightarrow s}^* - \rho^* Y_{soot}^*) \beta^*. \quad (3.454)$$

This term vanishes when the soot mass fraction equals the maximum carbon mass fraction, by virtue of its construction. However, this form is clearly not the form suggested by Tesner [64], the scaling factor notwithstanding.

Soot Combustion Model

The soot combustion model assumes that soot is destroyed in the flame zone based on two factors 1) the rate at which it is mixed into the flame zone, and 2) that there is sufficient oxygen to consume it. The mixing rate is the same as in (3.310) (in the gas phase combustion model section) where the species Y_k are treated as follows: In the cell, the fraction of soot that will burn up in the flame zone is $(1 - \xi_c) Y_{soot}$. In the flame zone, this mass is converted to CO_2 , so its mass fraction in the flame zone is zero. The radical nuclei concentration is treated similarly. Therefore,

$$\frac{\dot{R}_{n,comb}}{\bar{\rho}} = \left(\frac{-(1 - \xi_c) \frac{n}{\rho}}{\tau_{res}} \right) \left(\frac{\gamma \chi}{1 - \gamma \chi} \right) \chi^3, \quad (3.455)$$

$$\frac{\dot{R}_{soot,comb}}{\bar{\rho}} = \left(\frac{-(1 - \xi_c) Y_{soot}}{\tau_{res}} \right) \left(\frac{\gamma \chi}{1 - \gamma \chi} \right) \chi^3. \quad (3.456)$$

It is convenient to define a new timescale,

$$\tau_h = \frac{(1 - \gamma \chi) \tau_{res}}{\chi^3}. \quad (3.457)$$

Calculating Properties of the Reaction Zone

The foregoing models for soot and radical nuclei contain properties corresponding to the flame zone and surroundings. This section discusses the method employed by Magnussen et al. to compute these properties. The flame zone properties are computed by assuming local equilibrium mass transfer due to turbulent mixing between the reaction zone and surroundings. In other words, the production and combustion rates are sufficiently slow that the mass concentrations in the flame zone come to an equilibrium state with the surroundings via the turbulent mixing rate. This equilibrium rate is assumed to instantaneously adjust to the new cell conditions at every time step.

For this steady-state, steady flow approximation, a balance equation can be written for both nucleate particles and soot mass fraction for the flame zone. In words, the radical nuclei

concentration (or soot mass fraction) mixed into the flame zone minus the radical nuclei concentration (or soot mass fraction) mixed out of the flame zone plus the production of radical nuclei (or soot) minus the combustion of radical nuclei (or soot) equals zero. Note that the combustion rates given above are equal to the mixing rates times the fraction of radical nuclei concentration (or soot mass fraction) able to combustion. So the difference in these terms is equal to the soot production rates or,

$$\frac{(\beta^* - \xi_c \beta)}{\tau_h} = \beta_0^* + \frac{f_c^* \dot{R}_{n,f-g}^*}{\rho^*} - \frac{\dot{R}_{n,g0}^*}{\rho^*}, \quad (3.458)$$

$$\frac{(Y_{soot}^* - \xi_c Y_{soot})}{\tau_h} = f_c^* m_p^* \frac{\dot{R}_{soot,form}^*}{\rho^*}. \quad (3.459)$$

Solution of these two algebraic equations with two unknowns gives, β^* and Y_{soot}^* , the radical nuclei and soot concentrations in the flame zone, respectively. Note that the formation/destruction terms are of a bilinear form in the soot and radical nuclei concentrations. Thus, to compute the flame zone values of radical nuclei and soot mass fractions requires the simultaneous solution of this 2×2 system of equations. In particular, substituting for these terms from the formula given above, (3.459) can be solved for Y_{soot}^* using (3.454). The result is that the mass fraction of soot in the flame zone in terms of the radical nuclei concentration.

$$Y_{soot}^* = \frac{\xi_c Y_{soot} + \tau_h f_c^* b \rho^* Y_{c \rightarrow s} \beta^*}{1 + \tau_h f_c^* b \rho^* \beta^*}. \quad (3.460)$$

(3.460) can be used in (3.458) to form a quadratic equation for β^* ,

$$\tilde{a}_s (\beta^*)^2 + \tilde{b}_s \beta^* + \tilde{c}_s = 0, \quad (3.461)$$

where,

$$\tilde{a}_s = f_c^* \tau_h \rho^* (\tilde{\alpha} b + \tau_h a g_0), \quad (3.462)$$

$$\tilde{b}_s = \tilde{\alpha} + \tau_h \rho^* \left(\frac{\xi_c Y_{soot} g_0}{b} - f_c^* b A \right), \quad (3.463)$$

$$\tilde{c}_s = -A, \quad (3.464)$$

$$A = \xi_c \beta + \tau_h \beta_0^*, \quad (3.465)$$

$$\tilde{\alpha} = 1 - \tau_h (f - g) f_c^*. \quad (3.466)$$

The solution is the negative root of the quadratic, here written in a computationally appropriate form,

$$\beta^* = \frac{-2\tilde{c}_s}{\tilde{b}_s + \sqrt{\tilde{b}_s^2 - 4\tilde{a}_s \tilde{c}_s}} \quad (3.467)$$

In the limit where

$$Y_{soot}^* \rightarrow Y_{c \rightarrow s}^* \quad (3.468)$$

then the soot mass fraction becomes static and the radical nuclei concentration can be solved for directly. The result is

$$\beta^* = \frac{\xi_c \beta + \tau_h \beta_o^*}{1 + \tau_h \left(-f_c^* (f - g) + \frac{ag_o}{b} \right)}. \quad (3.469)$$

Calculating Properties of the Surroundings

Having computed the properties of the reaction zone, the properties for the surroundings are calculated from the definition of the cell (average) values,

$$\beta^\circ = \frac{\beta - \gamma \chi \beta^*}{1 - \gamma \chi}, \quad (3.470)$$

$$Y_{soot}^\circ = \frac{Y_{soot} - \gamma \chi Y_{soot}^*}{1 - \gamma \chi}. \quad (3.471)$$

$$\beta^\circ = \min \left(\beta^\circ, \frac{\frac{ag_o}{b} \times 10^{-6}}{\rho^\circ} \right) \quad (3.472)$$

Note that there is an upper bound to the number of nucleate particles based on a 50 percent dense mixture given they are monodisperse at the size given in [Table 3.7](#) with mass given by

$$m = \rho_{soot} \frac{4}{3} \pi \left(\frac{d_p}{2} \right)^3. \quad (3.473)$$

Now we are in a position to specify the transport equations and source terms for the soot model.

Transport Equations and Source Terms

Two transport equations for radical nuclei and soot mass fractions need be solved,

$$\int \frac{\partial \rho \beta}{\partial t} dV + \int \rho \beta u_j n_j dS = \int \frac{\mu_{eff}}{\sigma_Y} \frac{\partial \beta}{\partial x_j} n_j dS + \int \rho S_n dV, \quad (3.474)$$

$$\int \frac{\partial \rho Y_{soot}}{\partial t} dV + \int \rho Y_{soot} u_j n_j dS = \int \frac{\mu_{eff}}{\sigma_Y} \frac{\partial Y_{soot}}{\partial x_j} n_j dS + \int \rho S_{soot} dV. \quad (3.475)$$

In general, the source term, in particles/kg-sec, for radical nuclei is given by,

$$S_n = \gamma \chi \left(\frac{\dot{R}_{n,form,mod}^*}{\rho^*} - \frac{\dot{R}_{n,comb}^*}{\rho^*} \right) + (1 - \gamma \chi) \frac{\dot{R}_{n,form,mod}^\circ}{\rho^\circ} \quad (3.476)$$

where the form of the net formation/destruction source terms is,

$$\frac{\dot{R}_{n,form,mod}^{\circ}}{\rho} = \beta_0^{\circ} + (f - g) \beta^{\circ} \max(0, f_c^{\circ}) - g_0 \frac{aY_{soot}^{\circ}}{bY_{c \rightarrow s}^{\circ}} \beta^{\circ}. \quad (3.477)$$

For each of the reaction and surrounding zones, the (production destruction) of radical nuclei in the flame zone is given by the mixing balance, or

$$\left(\frac{\dot{R}_{n,form}^*}{\rho} - \frac{\dot{R}_{n,comb}^*}{\rho} \right) = \left(\frac{\beta^* - \beta}{\tau_h} \right). \quad (3.478)$$

Substituting gives,

$$S_n = \gamma\chi \left(\frac{\beta^* - \beta}{\tau_h} \right) + (1 - \gamma\chi) \left(\beta_0^{\circ} + (f - g) \beta^{\circ} \max(0, f_c^{\circ}) - g_0 \frac{aY_{soot}^{\circ}}{bY_{c \rightarrow s}^{\circ}} \beta^{\circ} \right), \quad (3.479)$$

The general source term for soot (1/sec) is given by

$$S_{soot} = m_p \gamma\chi \left(\frac{\dot{R}_{soot,form,mod}^*}{\rho^*} - \frac{\dot{R}_{soot,comb}^*}{\rho^*} \right) + (1 - \gamma\chi) m_p \frac{\dot{R}_{soot,form,mod}^{\circ}}{\rho^{\circ}} \quad (3.480)$$

The (production-destruction) of soot in the flame zone is likewise given by the mixing balance, or

$$m_p \left(\frac{\dot{R}_{soot,form}^*}{\rho} - \frac{\dot{R}_{soot,comb}^*}{\rho} \right) = \left(\frac{Y_{soot}^* - Y_{soot}}{\tau_h} \right). \quad (3.481)$$

Substituting gives,

$$S_{soot} = \gamma\chi \left(\frac{Y_{soot}^* - Y_{soot}}{\tau_h} \right) + (1 - \gamma\chi) f_c^{\circ} b \rho^{\circ} (Y_{c \rightarrow s}^{\circ} - Y_{soot}^{\circ}) \beta^{\circ}, \quad (3.482)$$

which also follows the practice of using the scale factor and effective mass for a soot particle in the surroundings, $m_p = b\rho^{\circ}Y_{c \rightarrow s}^{\circ}/a$.

The fact that the soot and radical nuclei concentrations are treated as tracers should be reemphasized. This means that their concentrations in the gas mixture are assumed insignificant such that they do not enter into calculations of density, or other properties of the mixture.

Absorptivity Model

The absorption coefficient submodel calculates a spectrally averaged total absorptivity value for a homogeneous (in thermodynamic state and composition) mixture of gaseous CO₂, H₂O, and soot particles. It should be recognized that this model does not account for either the presence of volatilized hydrocarbon molecules nor for the spectral line broadening effects of N₂ gas. The following implicit assumptions are made:

1. Thermodynamic equilibrium between soot and gas phase.
2. Homogeneous mixture over length scale of interest (cf. input 1)
3. Individual (non agglomerated) spherical soot particles with diameter much smaller than the radiation wavelength (Rayleigh scattering).
4. Absorptivity of the soot varies inversely with radiation wavelength.

The following quantities are required:

1. Length scale indicating the optical path length of interest, L_{cell} in centimeters.
2. Mixture temperature, T , in Kelvin.
3. Total mixture pressure, p_{mix} , in bar.
4. Partial pressures of the CO_2 and H_2O gaseous components, p_{co2} , p_{h2o} , in bar.
5. Soot volume fraction, X_{soot} .

The absorptivity model generates the following output:

- Spectrally averaged absorptivity, α , in cm^{-1} .

The absorptivity is based on empirical correlations for the total emittance of a homogeneous, isothermal mixture with a given optical path length. The correlations used in this model are based on empirical data covering a range of optical path lengths, temperatures, soot concentrations and pressures:

- $1\text{ cm} \leq L_{cell} \leq 10^3\text{ cm}$
- $600K \leq T \leq 2400K$
- $10^{-8} \leq X_{soot} \leq 10^{-5}$
- $0.1\text{ bar} \leq p_{co2}, p_{h2o} \leq 1\text{ bar}$

The absorptivity values provided by the equations in this model are accurate to within 10% - 30% of their value with greater accuracy at higher temperatures, path lengths, and concentrations.

Theory

The total (e.g. integrated over all wavelengths) absorptivity of a homogeneous (in composition and temperature) thickness L_{cell} layer of CO_2 gas, H_2O gas, and soot particles may be expressed in terms of the total emittance of the layer

$$\alpha = -\frac{1}{L_{cell}} \log (1 - \kappa) , \quad (3.483)$$

where α is the total absorptivity and κ is the total emittance. The total emittance of the mixture may be expressed in terms of the total emittance of the soot and gas phase (Siegel and Howell [9],

Eq. (13-145)),

$$\kappa = \kappa_{soot} + \kappa_{gas} - \kappa_{soot}\kappa_{gas}, \quad (3.484)$$

where κ_{soot} and κ_{gas} are the total emittance of the soot and gas phase respectively as if the other phase were not present.

To evaluate the absorptivity within a given control volume, the layer length, L_{cell} , is taken to be the geometric path length through the cell. This assumption (cf. assumption 2) implies that the mixture composition and temperature are uniform within the given cell. For convenience, the hydraulic diameter may be used for the layer thickness (in three dimensions),

$$L_{cell} = 2 \left[\frac{3V}{4\pi} \right]^{1/3}, \quad (3.485)$$

where V is the cell volume. Alternatively, Tezduyar [67] proposes a more expensive length scale for finite element grids,

$$L_{cell} = 2\hat{s} \cdot \left(\sum_{i=1}^{n_e} \nabla \phi_i \right), \quad (3.486)$$

where, L_{cell} is the path length through the element in direction \hat{s} , and ϕ_i is the finite element basis.

Emittance Model

The KAMELEON fire code (Holen, et al. [53]) employs the work of Felske and Tien [68] to provide the emittance of a mixture of CO_2 , H_2O , and soot particles. Assuming the absorptivity of the soot phase varies inversely with wavelength (Rayleigh scattering theory), a closed form expression may be obtained for the total emittance of the soot phase,

$$\kappa_{soot} = 1 - \frac{15}{\pi^4} \Psi^{(3)} \left[1 + \frac{c X_{soot} T L_{cell}}{C_2} \right], \quad (3.487)$$

where, X_{soot} is the soot volume fraction, T is the temperature, $C_2 = 0.01438769$ m-K is the second Planck constant, and $c = 7.0$ (Felske and Charalampopoulos [69] suggest $c = 5.0$). The pentagamma function $\Psi^{(3)}(x)$ is given by Abramowitz and Stegun [70],

$$\Psi^{(n)}(z) = \frac{d^{n+1}}{dz^{n+1}} \log [\Gamma(z)] = (-1)^{n+1} \int_0^\infty \frac{t^n e^{-zt}}{1 - e^{-t}} dt, \quad n = 1, 2, 3, \dots \quad (3.488)$$

(3.488) may be evaluated by the series expansion (Abramowitz and Stegun [70]),

$$\Psi^{(3)}(z) = 6 \sum_{k=0}^{\infty} \frac{1}{(z+k)^4}, \quad (3.489)$$

and by the seven-term asymptotic expansion,

$$\Psi^{(3)}(z) = \frac{2}{z^3} + \frac{3}{z^4} + \frac{2}{z^5} - \frac{1}{z^7} + \frac{4}{3z^9} - \frac{3}{z^{11}} + \frac{10}{z^{13}} + \dots \quad (3.490)$$

(3.490) is accurate to within 1% of the value given by (3.489) for $z > 1.6$ and accurate to within 0.1% of the value given by (3.489) for $z > 2$. A plot of the pentagamma function and the asymptotic expansion are provided in Figure 3.5 for reference.

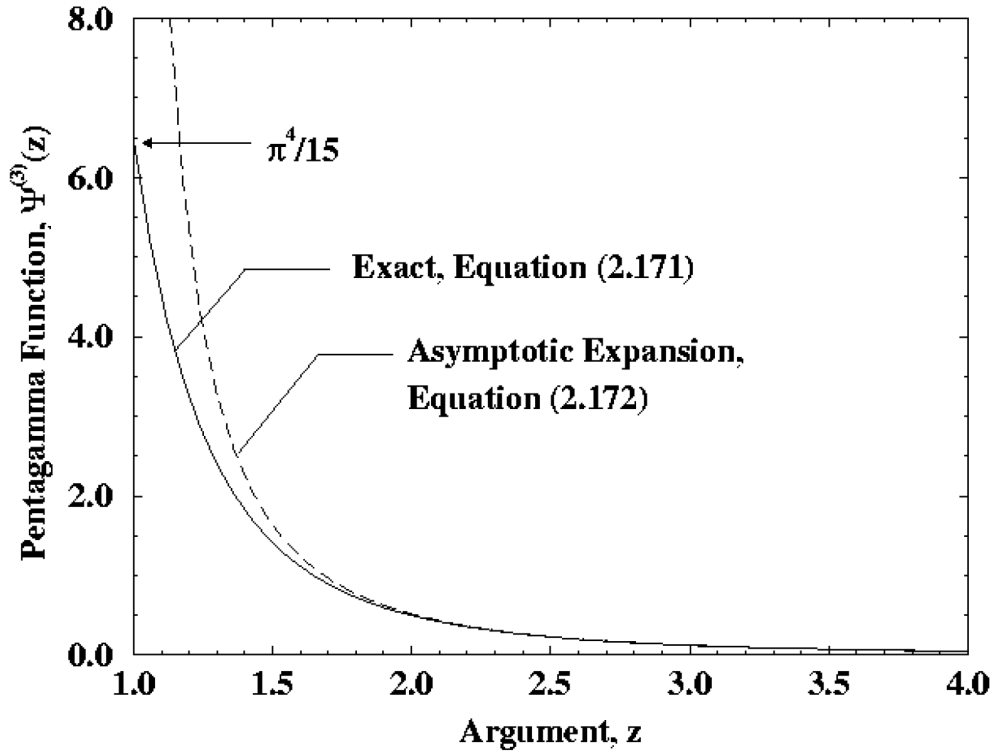


Fig. 3.5: Pentagamma function and asymptotic expansion

The emittance of the gas phase is given by Leckner [71]. Leckner's model is relatively involved and assumes that the path length, L_{cell} , is given in centimeters, the temperature, T , is given in Kelvin, and the pressure, p , is given in bars. Leckner also defines a reference temperature, $T_o = 273$ K, and pressure, $p_o = 1$ bar, for reduction purposes. Two additional quantities used by Leckner are the scaled temperature, $\theta = T/1000$ K and the logarithm of the optical path length, $\lambda_v = \log_{10}(p_v L_{cell})$ where the subscript v represents one of the species CO_2 or H_2O . These quantities are summarized in Table 3.8.

The emittance of the gas phase (cf. (3.484)) is the sum of the CO_2 and H_2O contributions less a

correction factor which accounts for overlap in the CO₂ and H₂O absorption bands,

$$\kappa_{gas} = \kappa_{h2o} + \kappa_{co2} - \Delta\kappa, \quad (3.491)$$

where the species emittance at a given partial pressure and temperature is expressed in terms of a scale emittance, $\kappa_{v,o}$.

$$\frac{\kappa_v}{\kappa_{v,o}} = \exp \left(-\xi (\lambda_{max} - \lambda_v)^2 \right) \left(\frac{AP_E + B}{P_E + A + B - 1} - 1 \right) + 1 \quad (3.492)$$

Table 3.9 summarizes the quantities on the right hand side of (3.492). The scale emittance, $\kappa_{v,o}$, for both species is given by the expressions

$$\log (\kappa_{v,o}) = a_0 + \sum_{i=1}^M a_i \lambda_v^i, \quad (3.493)$$

$$a_i = c_{i0} + \sum_{j=1}^N c_{ij} \theta^j, \quad (3.494)$$

where the coefficients a_i and c_{ij} are given in Table 3.10 and Table 3.11 for CO₂ and H₂O respectively. (Leckner provides several alternative listings for the coefficients for calculating the total emittance of CO₂. The values listed in Table 3.10 are the values employed in the KAMELEON-II-FIRE program (1994).)

The effect of the overlap correction factor in (3.491) is relatively small so Leckner [71] employed an approximate expression obtained from emittance data for a total pressure of 1 bar and temperatures between 1000K and 2200K:

$$\Delta\kappa = \left(\frac{\zeta}{10.7 + 101\zeta} - 0.0089\zeta^{10.4} \right) (\log_{10} [(p_{co2} + p_{h2o}) L_{cell}])^{2.76}, \quad (3.495)$$

where,

$$\zeta = \frac{p_{h2o}}{p_{h2o} + p_{co2}}. \quad (3.496)$$

The following observations are made to clarify the range of applicability of the absorptivity submodel specifically for hydrocarbon combustion applications. The absorptivity model does not account for the presence of volatilized hydrocarbon molecules which may have strong absorption bands in the infrared region. The VULCAN/KAMELEON fire code (Holen, et al. [53]) accounts for the presence hydrocarbon molecules by treating hydrocarbon molecules in the same manner as the CO₂ and H₂O product species (cf. the partial pressure submodel). This is a convenient although questionable assumption which provides for a zeroth order treatment of absorption by hydrocarbon molecules.

Table 3.8: Parameters used in Leckner's gas phase emittance model.

Quantity	Definition
Temperature units, $[T]$	Kelvin
Path length units, $[L_{cell}]$	centimeters
Pressure units, $[p]$	bar
Reference temperature, T_o	273 K
Reference pressure, p_o	1 bar
Scaled path length, λ_v	
Scaled temperature, θ	$T/1000K$

Table 3.9: Species-specific parameters used in (3.492)

Quantity	CO ₂	H ₂ O
Equivalent pressure, P_E	$P_E = p_{mix} \left(1 + 0.28 \frac{p_{co2}}{p_{mix}} \right)$	$P_E = p_{mix} \left(1 + 1.49 \frac{p_{co2}}{p_{mix}} \sqrt{\frac{T_o}{T}} \right)$
	for $T > 700K$	
	$\lambda_{max} = \log_{10} (0.225\theta^2)$	
Maxima location, λ_{max}		$\lambda_{max} = \log_{10} (13.2\theta^2)$
	for $T < 700K$	
	$\lambda_{max} = \log_{10} (0.054\theta^{-2})$	
Coefficient, ξ	$\xi = 1.47$	$\xi = 0.5$
Coefficient, A	$A = 1.0 + 0.1\theta^{-1.45}$	$A = 1.888 - 2.053 \log_{10} \theta$
		$\theta = 2.145$ if $T < 750K$
Coefficient, B	$B = 0.23$	$B = 1.1\theta^{-1.4}$

Table 3.10: Coefficients C_{ij} for calculating the scale total emittance of CO₂ from (3.493) and (3.494), (valid for $T > 400K$).

i	j (N=4)				
(M=3)	0	1	2	3	4
0	-3.9781	2.7353	-1.9882	0.31054	0.015719
1	1.9326	-3.5932	3.7247	-1.4535	0.20132
2	-0.35366	0.61766	-0.84207	0.39859	-0.063356
3	-0.080181	0.31466	-0.19973	0.046532	-0.0033086

Table 3.11: Coefficients C_{ij} for calculating the scale total emittance of H₂O from (3.493) and (3.494), (valid for $T > 400K$).

i	j (N=2)		
(M=2)	0	1	2
0	-2.2118	-1.1987	0.035596
1	0.85667	0.93048	-0.14391
2	-0.10838	-0.17156	0.045915

Fuel Boundary Condition Submodel

In most cases, fires are the result of burning fuel vapor in air. Exceptions include oxygenated and energetic materials that embody both fuel and oxidizer. The source of fuel vapor may be a gas release, the vapor which forms over a liquid surface due to its vapor pressure, liquid fuel which is heated above its vaporization temperature, or solid materials which are heated to the point where combustible gases are released due to pyrolysis reactions. The purpose of this submodel is to provide the mass flux and temperature of fuel vapor which enters the computational domain at the boundaries. This submodel is only required if the source of fuel is a solid or liquid since gas releases can be specified as a flow boundary condition. Since the generation of fuel vapor from these materials involves, as a minimum, representing thermal transport within the material including phase change, a simplified approach is taken here to serve the basic need of present generation fire models. The development of improved, validated models is presently underway. Present generation models are limited to liquid fuels in the form of pools (i.e., a defined amount of fuel constrained in a pool with fixed, known geometry) and spills onto non-absorbing substrates. (See Martinez and Hopkins [72] for a model of fuel spill in a porous medium.) Although the form of the submodel will allow first order estimates of fire growth rates, data acquired to date (Saito et al. [73]) tend to show that relevant flame spread mechanisms include features which occur at lengths scales several orders of magnitude below the resolution of present grids. Additional submodels will be therefore be required to predict flame spread with confidence. The following quantities are required:

1. $T_{fuel,vap}$, the vaporization temperature of the fuel (K).
2. h_{fg} , the heat of vaporization of the fuel (KJ/kg),
3. C_{pl} , the specific heat of the liquid fuel (KJ/kg-K),
4. $T_{fuel,init}$, the initial temperature of the liquid fuel (K),
5. $\alpha_{fuel,liq}$, the absorptivity of the liquid fuel,
6. q''_{rad} , the radiative heat flux incident on the fuel surface,
7. q''_{conv} , the convective heat flux incident on the fuel surface.

The fuel boundary condition submodel generates the following output:

- \dot{m}'' , the mass flux of fuel (kg/m²-s).

The fuel pool will be modeled as a mass of liquid that is gradually converted to vapor which in turn enters the flow field as a distinct species. The fuel vapor generation rate is based on the incident heat flux to the pool surface. Data for heavy hydrocarbon fuels (Gritz, et al. [74, 75]) show the following:

- After the initial transient (which includes flame spread) the fuel burning (and hence vaporization) rate is steady.
- Heating of the fuel is limited to the top 1.5 cm (which greatly exceeds the penetration depth for combined thermal transport in semitransparent media).

- Fuel transport occurs within the pool due to the preservation of a fuel free surface and the presence of a non-uniform heat flux to the fuel surface.
- The temperature at the free surface of the fuel is spatially uniform and approximately equal to the mean of the distillation curve for multi-component fuels.

Given these observations, the present submodel includes two options for calculating the fuel vaporization. These options are used for both pool and spill fires.

Option 1: Constant, Specified Mass Flux

In this option, the output of the submodel will be specified directly by the user. Fuel will be released at the boundaries defined by a fuel free surface. Since the burning rate is constant, the mass flux can be considered constant. Fuel burn rate data (for example, Blinov and Khudiakov [76]) are available as a function of pool size for a variety of fuels. This option neglects the physical process of fuel heating and is therefore only appropriate for steady burning fires. The spatial variation of fuel vaporization is also neglected.

Option 2: Mass Flux as a Function of Incident Heat Flux

Before the surface of the fuel reaches its vaporization temperature, the pool model splits the incoming flux (q'') between the heating of the fuel and vaporization of the fuel. The fraction of heat transferred into the material (f_s) is determined using a linearized approximation for the temperature distribution in the media by (using a user-specified upper bound of p_s)

$$f_s = \min \left(p_s, \frac{T_b - T_f}{\Delta T_{bf}} \right) h \quad (3.497)$$

This leads to an evaporation rate of

$$\dot{m}'' = \frac{q''(1 - f_s)}{h_{fg}}. \quad (3.498)$$

and a rate of change in the pool temperature (T_f) and height (H) of

$$\frac{dT_f}{dt} = \frac{q'' f_s}{H \rho_f C_{p,f}}, \quad (3.499)$$

$$\frac{dH}{dt} = -\frac{\dot{m}''}{\rho_f}. \quad (3.500)$$

Due to low diffusivity and high opacity of hydrocarbon fuels, the temperature gradient in the liquid fuel develops quickly, is considerably larger than the linear approximation, and does not extend to the lower surface of the fuel, so the fuel temperature variation with respect to height is neglected. The transient fuel heating occurs at the same short time and length scales as flame spread. The inclusion of this feature is not suggested until a more rigorous technique for modeling flame spread can be developed.

Fuel Spreading Submodel

The VULCAN/KAMELEON fire code includes a model which represents the spreading of fuel on a non-absorbing substrate. This feature allows the simulation of fires resulting from fuel spills. Various correlations (Mansfield and Linley [77]) and global, quasi-steady-state, algebraic models (Cline and Koenig [78]; Magnoli [79]) have been developed to determine the size of a circular pool fire resulting from a fuel spill. Since these models are global in nature, and do not include the effects of complex geometries resulting from obstacles, they will not be included as submodel options. The following quantities are required:

1. ρ_{fuel} , the density of the liquid fuel,
2. μ_{fuel} , the viscosity of the liquid fuel,
3. $Q_{release}$, the volumetric flow of fuel released by the spill,
4. A_s , the surface area of the element,
5. γ , the surface tension coefficient of the liquid fuel.

The fuel boundary condition submodel generates the following output:

- h , the depth of fuel (m).

The following assumptions are invoked as part of the fuel spreading model presently in VULCAN.

1. The fuel is sufficiently thin for inertial forces to be neglected as compared to shear forces.
2. The velocity components in the fuel are always horizontal.
3. The substrate is smooth, horizontal and non-absorbing.
4. The flow is laminar.
5. The interface between the fuel and air at the front of the spreading fuel is parabolic.
6. The shear stress is zero at the top of the film.

Given the preceding assumptions, the spread of fuel is driven by the difference between hydrostatic pressure due to variations in fuel depth. The transport can then be represented by

$$\frac{\partial h}{\partial t} = \frac{\partial}{\partial x_j} \left(\frac{\rho_{fuel} g h^3}{3\mu_{fuel}} \right) \frac{\partial h}{\partial x_j} + S. \quad (3.501)$$

(3.501) is solved explicitly to track the fuel thickness along the flat surface. Boundary conditions and source terms are defined as follows to represent various physical features.

1. Drains - The depth of fuel is set equal to zero for cells occupied by drains. The volume of fuel transported into the drain cell is removed via a negative source term. occupied by drains. The volume of fuel transported into the drain cell is removed via a negative source term.

2. Obstacles - The fuel depth and the gradient of the fuel depth is set equal to zero at the interface between obstacles and surrounding cells.
3. Release Locations - The source term is defined by the volumetric flow of released fuel divided by the surface area of the element (i.e. $Q_{release}/A_s$).

The fuel will spread up until the hydrostatic pressure gradient is balanced by surface tension forces. Subject to the preceding assumptions, the minimum fuel depth is given by

$$h_{min} = \sqrt{\frac{2s}{\rho_{fuel}g}}, \quad (3.502)$$

where s is the coefficient of surface tension for the fuel. The reduction in fuel depth due to the vaporization of fuel is calculated by the same technique used to define the fuel vapor boundary condition for pool fires.

One-Dimensional Composite Fire Boundary Condition

Conceptual Overview

Fuego includes a boundary condition that is capable of modeling the thermal decomposition and outgassing of a thin sheet of porous material at the boundary surface, initially intended to simulate the combustion of a sheet of carbon fiber composite material. Variation through the material thickness is assumed to be locally one-dimensional. The actual implementation is quite flexible, allowing the simulation of the thermal response of essentially any finite-thickness material that can optionally undergo a user-specified chemical decomposition mechanism.

Figure 3.6 illustrates a two-dimensional representation of the virtual mesh used for this 1D composite fire boundary condition. One layer of elements above the boundary is shown, within which Fuego performs its normal fluid solve using the control volume finite element CVFEM method. The CVFEM sub-control volumes are demarcated with dashed lines. An equal-order interpolation methodology is used, so that all solution variables are stored at the element vertices.

For this boundary condition, a series of independent one-dimensional virtual domains exist behind each CVFEM surface node, and each virtual 1D domain has a cross-sectional area that matches the group of CVFEM boundary sub-control surfaces that contain the single “parent” surface node. A classical cell-centered finite volume methodology is used for the 1D virtual domains, where the discretization, storage, and numerical solutions all occur within the boundary condition implementation and only interact with the main CVFEM flow solution through fluxes and solution variables at the exposed surface.

Each 1D domain is assumed to have a fixed geometry that is filled with a simple porous material that is allowed to react chemically to form gaseous species. Since the overall volume of each element is fixed, the porosity of each volume is assumed to increase as species are converted from solid to gas. It is assumed that the gaseous species within the pores of the solid phase are of

secondary concern, and as such no discrete transport equation is solved for them. The approximation is instead made that all gases generated within the porous material appear instantaneously at the surface of the material as a flux into the main fluid solution. It would be straightforward to solve additional transport equations for fluid flow within the porous material if that level of fidelity were to become necessary, as in the case of oxidative reactions where oxygen must diffuse through the exposed surface into the porous material before reactions may occur.

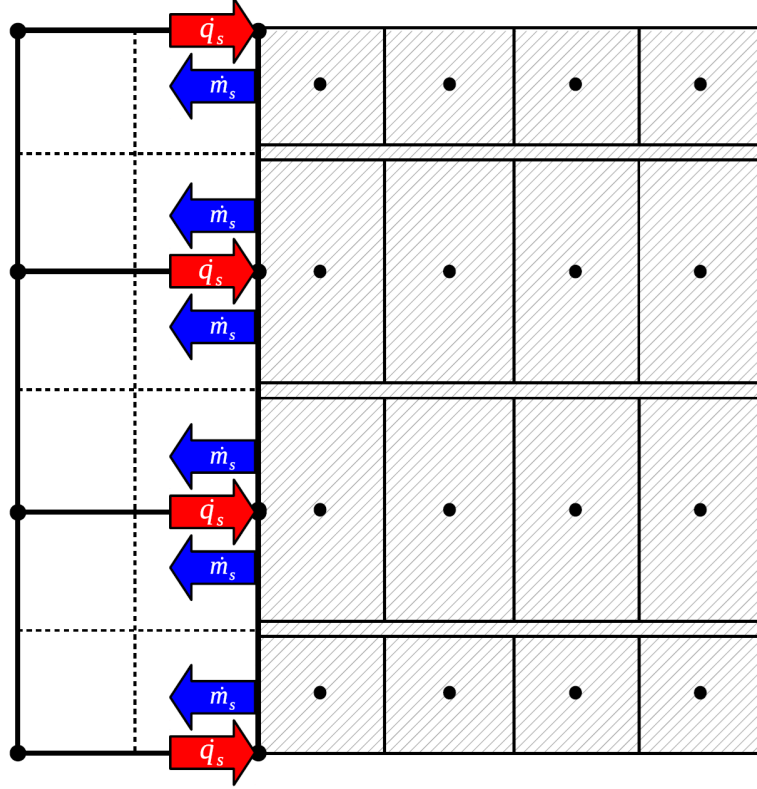


Fig. 3.6: Representative mesh layout for 1-D composite fire boundary condition

Model Formulation

Transport Equations

Within the solid phase of the porous material, one-dimensional transport equations for continuity, chemical species, and energy are solved in the form:

$$\frac{\partial \bar{\rho}}{\partial t} = \dot{\omega}_c''' \quad (3.503)$$

$$\frac{\partial \bar{\rho} Y_k}{\partial t} = \dot{\omega}_k''' \quad (3.504)$$

$$\bar{\rho} \bar{c} \frac{\partial T}{\partial t} = \frac{\partial}{\partial x} \left(\bar{k} \frac{\partial T}{\partial x} \right) + \dot{q}''', \quad (3.505)$$

where $\bar{\rho}$, \bar{c} , and \bar{k} are the mixture-averaged bulk density, specific heat, and thermal conductivity, respectively, Y_k is the mass fraction of chemical species k , T is the temperature of the solid phase, \dot{q}''' is the volumetric heat generation rate due to chemical reactions, $\dot{\omega}_k'''$ is the volumetric mass generation rate of chemical species k , and $\dot{\omega}_c'''$ is the overall mass generation rate computed as $\dot{\omega}_c''' = \sum_k \dot{\omega}_k'''$.

Material Models

The composite material used for this boundary condition is assumed to be of a fixed volume, *i.e.* there is no structural deformation allowed. The bulk density of the multi-species solid mixture is assumed to be a function of the density of each component species in their native porous state, as

$$\bar{\rho} = \left(\sum_k \frac{Y_k}{\rho_k} \right)^{-1}, \quad (3.506)$$

where ρ_k is the porous density of species k , provided as a material model by the user. This model for the mixture bulk density is only used to compute the initial bulk density field, which is subsequently solved directly from (3.503).

The porosity of the mixture is assumed to follow the model

$$\bar{\psi} = \sum_k X_k \psi_k, \quad (3.507)$$

where X_k is the volume fraction of species k ,

$$X_k = \bar{\rho} \frac{Y_k}{\rho_k}, \quad (3.508)$$

and ψ_k is the porosity of pure species k , modeled as

$$\psi_k = 1 - \frac{\rho_i}{\rho_{s0,k}}, \quad (3.509)$$

where $\rho_{s0,k}$ is the density of the solid (non-porous) species k at a reference temperature. Note that the porosity does not appear explicitly in any of the transport equations or subsequent material models, so that it is never computed as part of the boundary condition solution. It would only appear in transport equations for the gaseous species occupying the pores of the solid skeleton, if this level of detail were ever to be added to this model.

In their most detailed form, the bulk thermal conductivity and specific heat are evaluated as a volume average and mass average of the individual species properties, respectively, as

$$\bar{k} = \sum_k X_k k_k \quad (3.510)$$

$$\bar{c} = \sum_k Y_k c_k, \quad (3.511)$$

although a species-independent model for the overall bulk property may be used if the individual species properties are not known.

The last quantities that require a model are the volumetric species mass production rates, $\dot{\omega}'''$, and the volumetric heat production rate, \dot{q}''' . These quantities can be provided by the user in two different ways. The traditional approach is to supply them using standard material property evaluations as a part of the material model definition. These are arbitrary functions that themselves may be dependent on any of the solution variables or other material properties. If a nonreacting material is desired, then these terms may be simply modeled as zero.

The second way of supplying these quantities is by including a chemistry description block in the material model, which allows the user to specify multiple reactions and variable composition gas production.

Boundary Conditions

The exposed surface of the composite material interacts thermally with the environment through several mechanisms, including convective heat transfer and both radiation absorption and emission. These external fluxes must balance the conduction inside the composite material at the surface, as

$$\dot{q}''' = \dot{q}_{\text{conv}}''' + \dot{q}_{\text{rad}}''' \quad (3.512)$$

$$= \dot{q}_{\text{conv}}''' + \epsilon \left(\sigma T_1^4 - \dot{q}_{\text{irr}}''' \right) \quad (3.513)$$

where \dot{q}_{conv}''' is the convective flux imposed on the surface by the external laminar or turbulent boundary condition treatment, T_1 is the temperature solution from the first control volume in the composite material used to model the gray emission, and \dot{q}_{irr}''' is the external radiative flux incident on the surface.

On the back-side of the virtual composite material, optional convective and radiative heat transfer to a quiescent environment is modeled as

$$\dot{q}_{\text{b}}''' = \dot{q}_{\text{b,conv}}''' + \dot{q}_{\text{b,rad}}''' \quad (3.514)$$

$$= h_c (T_N - T_{\text{ref}}) + \sigma \epsilon_b \left(T_N^4 - T_{\text{ref}}^4 \right) \quad (3.515)$$

where h_c is a user-specified convection coefficient, ϵ_b is a user-specified back-side emissivity, T_{ref} is the modeled ambient environment temperature, and T_N is the temperature of the solution node closest to the back-side surface, assumed to be equal to the back-side surface temperature itself.

Numerical Implementation - Original

A segregated, implicit solution technique is used to numerically integrate (3.503)–(3.505). The discretized form of the continuity equation, (3.503), is derived by first integrating it over the finite volume V and the time step Δt to yield

$$\int_{\Delta t} \left[\int_V \frac{\partial \bar{\rho}}{\partial t} dV - \int_V \dot{\omega}_c''' dV \right] dt = 0 \quad (3.516)$$

$$\int_{\Delta t} \left[V_i \frac{\partial \bar{\rho}_i}{\partial t} - V_i \dot{\omega}_{c,i}''' \right] dt = 0. \quad (3.517)$$

Discretizing the temporal derivative using a first-order backward difference approximation and solving for the bulk density at the new time step yields

$$V_i \left(\bar{\rho}_i^{n+1} - \bar{\rho}_i^n \right) - V_i \dot{\omega}_{c,i}''' \Delta t = 0 \quad (3.518)$$

$$\bar{\rho}_i^{n+1} = \bar{\rho}_i^n + \dot{\omega}_{c,i}''' \Delta t. \quad (3.519)$$

where the mesh indices are defined in Figure 3.7. Note that this equation is linearized by evaluating the source term at the most recent estimate of the $n + 1$ solution state.

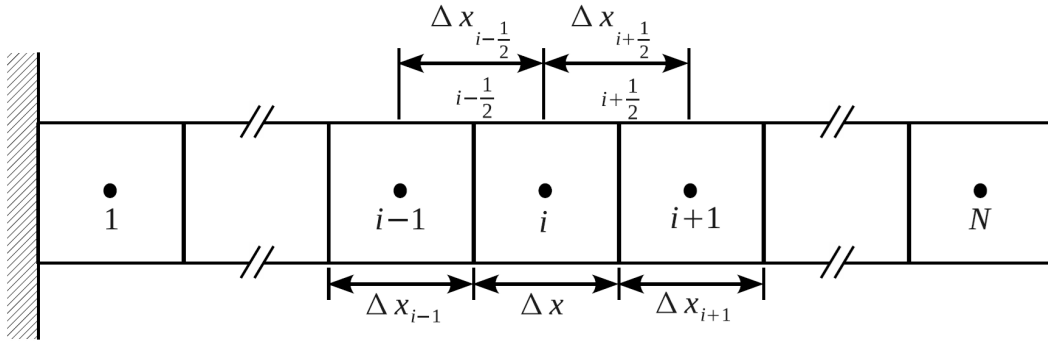


Fig. 3.7: Mesh index definition for 1-D composite fire boundary condition

The species transport equations, (3.504), undergoes an identical transformation,

$$\int_{\Delta t} \left[\int_V \frac{\partial \bar{\rho} Y_k}{\partial t} dV - \int_V \dot{\omega}_k''' dV \right] dt = 0 \quad (3.520)$$

$$\int_{\Delta t} \left[V_i \frac{\partial \bar{\rho}_i Y_k}{\partial t} - V_i \dot{\omega}_{k,i}''' \right] dt = 0. \quad (3.521)$$

$$V_i \left(\bar{\rho}_i^{n+1} Y_{k,i}^{n+1} - \bar{\rho}_i^n Y_{k,i}^n \right) - V_i \dot{\omega}_{k,i}''' \Delta t = 0 \quad (3.522)$$

$$Y_{k,i}^{n+1} = \frac{\bar{\rho}_i^n Y_{k,i}^n + \dot{\omega}_{k,i}''' \Delta t}{\bar{\rho}_i^{n+1}}, \quad (3.523)$$

where the bulk density at the new time level is used from (3.519), and the source term is evaluated from the most recent estimate of the $n + 1$ solution state.

The energy equation also undergoes a similar transformation, but with added complexity due to the inclusion of spatial derivatives. (3.505) is first integrated in both space and time, and the Gauss divergence theorem is used to remove one level of spatial derivatives in the diffusive flux term,

$$\int_{\Delta t} \left[\int_V \bar{\rho} \bar{c} \frac{\partial T}{\partial t} dV - \int_V \frac{\partial}{\partial x} \left(\bar{k} \frac{\partial T}{\partial x} \right) dV - \int_V \dot{q}''' dV \right] dt = 0 \quad (3.524)$$

$$\int_{\Delta t} \left[\int_V \bar{\rho} \bar{c} \frac{\partial T}{\partial t} dV - \int_A n \cdot \left(\bar{k} \frac{\partial T}{\partial x} \right) dA - \int_V \dot{q}''' dV \right] dt = 0. \quad (3.525)$$

Integrating numerically in space yields

$$\int_{\Delta t} \left[\bar{\rho}_i \bar{c}_i V_i \frac{\partial T_i}{\partial t} - \left(\left(-A \bar{k} \frac{\partial T}{\partial x} \right)_{i-\frac{1}{2}} + \left(A \bar{k} \frac{\partial T}{\partial x} \right)_{i+\frac{1}{2}} \right) - \dot{q}_i''' V_i \right] dt = 0 \quad (3.526)$$

$$\int_{\Delta t} \left[\bar{\rho}_i \bar{c}_i V_i \frac{\partial T_i}{\partial t} + A_{i-\frac{1}{2}} \bar{k}_{i-\frac{1}{2}} \left(\frac{T_i - T_{i-1}}{\Delta x_{i-\frac{1}{2}}} \right) - A_{i+\frac{1}{2}} \bar{k}_{i+\frac{1}{2}} \left(\frac{T_{i+1} - T_i}{\Delta x_{i+\frac{1}{2}}} \right) - \dot{q}_i''' V_i \right] dt = 0, \quad (3.527)$$

and then integrating in time and linearizing the equation by evaluating the coefficients at the most recent estimate of the $n + 1$ solution state yields

$$\bar{\rho}_i \bar{c}_i V_i \left(\frac{T_i^{n+1} - T_i^n}{\Delta t} \right) + A_{i-\frac{1}{2}} \bar{k}_{i-\frac{1}{2}} \left(\frac{T_i^{n+1} - T_{i-1}^{n+1}}{\Delta x_{i-\frac{1}{2}}} \right) - A_{i+\frac{1}{2}} \bar{k}_{i+\frac{1}{2}} \left(\frac{T_{i+1}^{n+1} - T_i^{n+1}}{\Delta x_{i+\frac{1}{2}}} \right) - \dot{q}_i''' V_i = 0. \quad (3.528)$$

This leads to a tri-diagonal system of coupled linear equations for the temperature at time level $(n + 1)$, which is solved using a direct method with the DGTSL module of the SLATEC library.

The continuity, species, and energy equations are solved sequentially in the order described, and the solution is repeated until the maximum normalized change in the temperature solution,

$$T_{\text{err}} = \frac{|T^{n+1} - T^*|}{T^{n+1}} \quad (3.529)$$

satisfies the user-specified tolerance, where T^* is the solution from the previous iteration.

Please see the Fuego user's manual for details on the usage of this boundary condition.

Numerical Implementation - New

When using the new form of the composite BC, where the chemical mechanism is specified using a chemistry description, the numerical implementation is slightly different. The finite volume discretization used is the same, but the system of equations is solved monolithically using the user-specified ODE solver. The solver handles time stepping during the sub-integration to reduce the overall error below the specified threshold.

Additionally, when constructing the monolithic system with the new form the DOFs are temperature and N species masses, rather than the prior approach of using temperature, density, and $N - 1$ species mass fractions.

Non-Conformal DG Boundary Condition

Conceptual Overview

The non-conformal boundary condition uses the DG approach described by Domino [80] and is currently implemented for turbulence models, continuity, momentum, and heat conduction. The non-conformal boundary condition is applied where you have two domains, A and B , which share a discontinuous interface with individual sidesets, S_A and S_B . The algorithm is applied in two passes, first iterating over all integration points in S_A and finding the matching face in S_B , then by iterating over the integration points in S_B and finding the matching face in S_A . The generic flux of a scalar, ϕ at an integration point on S_A is

$$\hat{Q}^A = \left[\frac{(q_j^A n_j^A - q_j^B n_j^B)}{2} + \lambda^A (\phi^A - \phi^B) \right] A_f^A + \dot{m}^A \frac{(\phi^A + \phi^B)}{2} + \eta \frac{|\dot{m}^A|}{2} (\phi^A - \phi^B), \quad (3.530)$$

where q_j is the diffusive flux, \dot{m} is the mass flux, and λ is the interior penalty coefficient.

Prior investigations have shown that pressure oscillations can be minimized by using the current integration point normal direction for both diffusive fluxes, so $n_j^B = -n_j^A$.

The penalty term, λ^A is given by

$$\lambda^A = \frac{(\Gamma^A/L^A + \Gamma^B/L^B)}{2}, \quad (3.531)$$

where Γ is the diffusive flux coefficient and L is an element length scale.

The advection coefficient, η , defines the degree of upwinding to use. A value of $\eta = 1$ results in a fully upwind scheme, while $\eta = 0$ results in a central difference scheme. When using a hybrid approach, this value is calculated locally based on the Peclet number.

Continuity

The mass flow rate at the non-conformal boundary includes the pressure stabilization terms, as

$$\dot{m}^A = \left[\frac{(\rho u_j^A + a_p^A G_j^A p - a_p^A \frac{\partial p^A}{\partial x_j}) n_j^A - (\rho u_j^B + a_p^B G_j^B p - a_p^B \frac{\partial p^B}{\partial x_j}) n_j^B}{2} + \lambda^A (p^A - p^B) \right] A_f^A. \quad (3.532)$$

Performance Considerations

There is a computational cost associated with the use of the non-conformal interface. This is largely due to two tasks: a search to match integration points with opposing faces on both non-conformal boundaries, and the resulting changes to the linear system stencil if the interface moves. Preliminary testing has shown that the cost of reinitializing the linear system with a new stencil is at least an order of magnitude greater than the cost of the search. For this reason, the algorithm implemented in Fuego will do an extra search in order to only reinitialize the linear system when the stencil actually changes. The user can expand the search boxes used in the stencil definition in order to reduce the number of linear system reinitializations by setting the “Search Expansion Factor” in the non-conformal boundary condition specification. This number is the approximate diametrical size increase in the stencil in terms of number of elements.

Non-Conformal Moving Walls

The non-conformal BC in Fuego supports having a moving surface as one side of the interface. This is implemented by providing a shell block containing nodal displacements on one side of the interface. Then non-conformal search will search for matching faces in opposing blocks and shells. If a shell is found, an appropriate wall boundary condition is applied at the integration point, while if a non-shell element is found the interior algorithms described previously are used.

Porous-Fluid Coupling Algorithm

This section provides a brief overview of the current porous/fluid coupling algorithm, as it is intended to be used in simulations of composite fires using Fuego to model the fluid region and coupling to Aria to model the porous region.. This is a loosely-coupled algorithm, relying on framework interpolation transfers of nodal fields between the porous region and the low-Mach fluid region and region-region Picard loops to converge the overall problem within a timestep.

Note that the shorthand is adopted where the porous region is described as region A and the low-Mach free fluid region is described as region B , with the interface between them referred to as Γ_{AB} and other boundaries not a part of this interface are referred to as $\Gamma \setminus \Gamma_{AB}$.

Fluid Flow

Bulk Equations

paragraph{Porous Continuity Equation} The porous region contains a condensed phase (the solid skeleton of the porous system) and a gas phase occupying the pores of the condensed phase. The condensed phase is not discussed explicitly in this description, although it interacts with the gas phase through things like its permeability and porosity, and its decomposition which can produce gas-phase mass through chemical source terms.

The porous gas-phase continuity equation within a porous region, to be solved for the gas-phase pressure p_g , is

$$\frac{\partial(\bar{\psi}\rho_g)}{\partial t} + \frac{\partial(\rho_g u_{j,g})}{\partial x_j} = \dot{\omega}_{fg}''', \quad (3.533)$$

where $\bar{\psi}$ is the mixture-averaged condensed-phase porosity, ρ_g is the gas-phase density, and $u_{j,g}$ is the gas-phase velocity vector computed from Darcy's approximation as

$$u_{j,g} = -\frac{\bar{K}}{\mu_g} \left(\frac{\partial p_g}{\partial x_j} + \rho_g g_j \right), \quad (3.534)$$

where \bar{K} is the mixture-averaged condensed-phase permeability, μ_g is the gas-phase viscosity, and g_j is the gravity vector. The term $\dot{\omega}_{fg}'''$ represents the formation rate of gas-phase mass from the condensed phase.

Multiplying (3.533) by an arbitrary test function w and integrating over the domain Ω while integrating the advection term by parts, yields the variational form of the continuity equation that is solved for p_g using the Galerkin finite element method,

$$\int_{\Omega} w \left(\frac{\partial(\bar{\psi}\rho_g)}{\partial t} - \dot{\omega}_{fg}''' \right) d\Omega - \int_{\Omega} \frac{\partial w}{\partial x_j} \rho_g u_{j,g} d\Omega + \int_{\Gamma} w \rho_g u_{j,g} n_j d\Gamma = 0, \quad (3.535)$$

where n_j is the boundary surface normal. The boundary flux term is then split into contributions on the interface between regions A and B and off the interface so that they may be treated separately. The continuity equation then takes the form

$$\int_{\Omega} w \left(\frac{\partial(\bar{\psi}\rho_g)}{\partial t} - \dot{\omega}_{fg}''' \right) d\Omega - \int_{\Omega} \frac{\partial w}{\partial x_j} \rho_g u_{j,g} d\Omega \quad (3.536)$$

$$+ \int_{\Gamma \setminus \Gamma_{AB}} w \rho_g u_{j,g} n_j d\Gamma + \int_{\Gamma_{AB}} w F_A d\Gamma = 0, \quad (3.537)$$

where F_A is the imposed flux on the porous side (A) of the Γ_{AB} interface. A detailed description of the coupling boundary flux is given in Section [Coupling Boundary Conditions](#)

Low-Mach Continuity Equation

The continuity equation within the low-Mach fluid region, to be solved for the pressure p , is

$$\frac{\partial \rho}{\partial t} + \frac{\partial \rho u_j}{\partial x_j} = S, \quad (3.538)$$

where ρ is the fluid density, u_j is the fluid velocity, and S is a generic mass volumetric source term. Integrating (3.538) over a CVFEM control volume and using the Gauss divergence theorem

on the advection and diffusive flux terms, yields the integral form of the continuity equation that is solved,

$$\int_{\Omega} \left(\frac{\partial \rho}{\partial t} - S \right) d\Omega + \int_{\Gamma} \rho u_j n_j d\Gamma = 0. \quad (3.539)$$

Similar to the porous continuity equation, the boundary flux term is split into contributions both on and off the Γ_{AB} interface, yielding

$$\int_{\Omega} \left(\frac{\partial \rho}{\partial t} - S \right) d\Omega + \int_{\Gamma \setminus \Gamma_{AB}} \rho u_j n_j d\Gamma + \int_{\Gamma_{AB}} F_B d\Gamma = 0. \quad (3.540)$$

The interface coupling flux is described in *Coupling Boundary Conditions*.

Low-Mach Momentum Equation

The momentum equation within the low-Mach fluid region, to be solved for the velocity u_i , is

$$\frac{\partial \rho u_i}{\partial t} + \frac{\partial \rho u_j u_i}{\partial x_j} = \frac{\partial \sigma_{ij}}{\partial x_j} + \rho g_i \quad (3.541)$$

where the Cauchy stress tensor is given by

$$\sigma_{ij} = \tau_{ij} - p \delta_{ij} \quad (3.542)$$

in terms of the viscous stress tensor

$$\tau_{ij} = \mu \left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i} \right) - \frac{2}{3} \mu \frac{\partial u_k}{\partial x_k} \delta_{ij}. \quad (3.543)$$

Integrating (3.541) over a CVFEM control volume and using the Gauss divergence theorem on the advection and diffusive flux terms, yields the integral form of the momentum equation that is solved,

$$\int_{\Omega} \frac{\partial \rho u_i}{\partial t} d\Omega + \int_{\Gamma} \rho u_j u_i n_j d\Gamma - \int_{\Gamma} \sigma_{ij} n_j d\Gamma - \int_{\Omega} \rho g_i d\Omega = 0. \quad (3.544)$$

Multiplying this equation by an arbitrary test function w , integrating the advection and stress terms by parts, and splitting the boundary flux terms into on-interface and off-interface contributions yields

$$\int_{\Omega} \frac{\partial \rho u_i}{\partial t} d\Omega - \int_{\Omega} \rho g_i d\Omega + \int_{\Gamma \setminus \Gamma_{AB}} \rho u_j u_i n_j d\Gamma - \int_{\Gamma \setminus \Gamma_{AB}} \sigma_{ij} n_j d\Gamma \quad (3.545)$$

$$+ \int_{\Gamma_{AB}} \rho u_j u_i n_j d\Gamma - \int_{\Gamma_{AB}} \sigma_{ij} n_j d\Gamma = 0. \quad (3.546)$$

Coupling Boundary Conditions

Coupling between the porous and fluid regions is achieved using an interface flux that is imposed as a Robin-style boundary condition. This approach has been used successfully in the past for coupling incompressible Darcy and Stokes flows [81]. Here we generalize the coupling for compressible fluids and Navier-Stokes flow.

The fluxes applied to the porous and fluid continuity equations at the interface Γ_{AB} are

$$F_A = \dot{m}_B \cdot \hat{n} + \beta(p_A - p_B) \quad (3.547)$$

$$F_B = \dot{m}_A \cdot \hat{n} + \beta(p_B - p_A), \quad (3.548)$$

where $\dot{m}_A = \rho_g \vec{u}_g$, $\dot{m}_B = \rho \vec{u}$, and the free constant β is computed as

$$\beta = c \frac{\bar{K} \rho_g}{\mu_g h} \quad (3.549)$$

with h being a measure of the mesh size adjacent to the interface, and c a user-specified scaling coefficient. The same value of β is used on both sides of the interface because that results in excellent mass conservation even on coarse meshes. If a different value of β is used on each side the method is still convergent but worse mass conservation is observed when solving on under-resolved meshes. Some attempts have been made to use an averaged penalty coefficient of the form

$$\beta = \frac{\beta_A + \beta_B}{2} \quad (3.550)$$

$$\beta_A = \frac{\bar{K} \rho_g}{\mu_g h} \quad (3.551)$$

$$\beta_B = \frac{\tau^c}{h}, \text{ or } \beta_B = \frac{\mu}{h P_{ref}}, \quad (3.552)$$

however they resulted in an impractically large number of Picard iterations to converge for some test problems.

A distinguishing condition BC for velocity is applied to the low-mach momentum equation in the form

$$u_j - (u_{j,n}^D + u_{j,t}^D) = 0, \quad (3.553)$$

where $u_{j,n}^D$ is the imposed normal component of velocity and $u_{j,t}^D$ is the imposed tangential component of velocity. The normal component is computed directly from the continuity flux at the interface,

$$u_{j,n}^D = \frac{F_B}{\rho} n_j. \quad (3.554)$$

The tangential component is based on a variation of the classical Beavers-Joseph-Saffman condition [82, 83] for the slip velocity which has been extended to non-planar surfaces in multidimensional flow [84], which defines a provisional model velocity

$$u_j^{\text{BJS}} = -\frac{\sqrt{\bar{K}}}{\alpha\mu} (n_i \tau_{ij}) \quad (3.555)$$

where \bar{K} is the permeability of the porous region at the interface, μ is the viscosity of the local fluid at the interface, τ_{ij} is the viscous stress tensor of the fluid at the interface, and α is a dimensionless model parameter that is a function of the microstructure of the porous material, which has been found to have typical values near 0.1 [83]. The tangential component of this vector quantity is used as the tangential component of the distinguishing condition velocity, and is computed as

$$u_{j,t}^D = u_j^{\text{BJS}} - \left(u_k^{\text{BJS}} n_k \right) n_j. \quad (3.556)$$

Enthalpy Transport

Bulk Equations

Porous Gas-Phase Enthalpy Equation

The gas-phase enthalpy equation within a porous region, to be solved for the gas-phase temperature T_g , is

$$\frac{\partial(\bar{\psi}\rho_g h_g)}{\partial t} + \frac{\partial(\rho_g u_{j,g} h_g)}{\partial x_j} = -\frac{\partial q_j^{h,g}}{\partial x_j} + \left(\frac{\partial(\bar{\psi} p_g)}{\partial t} + u_{j,g} \frac{\partial p_g}{\partial x_j} \right) \quad (3.557)$$

$$+ h_{cv} (\bar{T} - T_g) + \sum_k \left(\dot{\omega}_{s,fk}''' - \dot{\omega}_{s,dk}''' \right) h_{k,g} \quad (3.558)$$

where h_g is the mixture-averaged gas-phase enthalpy, h_{cv} is the volumetric heat transfer coefficient, \bar{T} is the porous condensed-phase temperature, $\left(\dot{\omega}_{s,fk}''' - \dot{\omega}_{s,dk}''' \right)$ is the formation and destruction of gas-phase species due to heterogeneous reactions, and $h_{k,g}$ is the gas-phase enthalpy of chemical species k . The gas-phase energy diffusive flux vector $q_j^{h,g}$ is modeled as

$$q_j^{h,g} = -\bar{\psi}\rho_g D_g \frac{\partial h_g}{\partial x_j}, \quad (3.559)$$

where D_g is the mixture-averaged gas-phase mass diffusivity.

Note that, in (3.558), there is some concern that the pressure spatial derivative term, $u_{j,g} \frac{\partial p_g}{\partial x_j}$, is incorrect. A crude re-derivation of this equation indicates that its form should instead be $\frac{u_{j,g}}{\bar{\psi}} \frac{\partial(\bar{\psi} p_g)}{\partial x_j}$. A more formal re-derivation from first principles is required to decide conclusively on

the correct form of this term, so it is left in its current form for now. Additionally, the diffusive flux vector is also of concern since the current form was derived under the assumption of constant specific heat, equal species mass diffusivities, and unity Lewis number. These assumptions may not be valid in future simulations, meaning that this term should possibly be returned to the standard Fick's law version that includes a contribution due to enthalpy transport by differential diffusion of chemical species. Again, this term is left in its current form for the present work.

Multiplying (3.558) by an arbitrary test function w and integrating over the domain Ω while integrating the advection and diffusion terms by parts, yields the variational form of the enthalpy equation that is solved for h_g using the Galerkin finite element method,

$$\int_{\Omega} w \left(\frac{\partial(\bar{\psi} \rho_g h_g)}{\partial t} - \left(\frac{\partial(\bar{\psi} p_g)}{\partial t} + u_{j,g} \frac{\partial p_g}{\partial x_j} \right) - h_{cv} (\bar{T} - T_g) - \sum_k \left(\dot{\omega}_{s,fk}''' - \dot{\omega}_{s,dk}''' \right) h_{g,k} \right) d\Omega \quad (3.560)$$

$$- \int_{\Omega} \frac{\partial w}{\partial x_j} (\rho_g u_{j,g} h_g) d\Omega + \int_{\Gamma} w (\rho_g u_{j,g} h_g) n_j d\Gamma \quad (3.561)$$

$$- \int_{\Omega} \frac{\partial w}{\partial x_j} q_j^{h,g} d\Omega + \int_{\Gamma} w q_j^{h,g} n_j d\Gamma = 0. \quad (3.562)$$

The boundary flux terms are then split into contributions on the interface between regions A and B and off the interface so that they may be treated separately. The enthalpy equation then takes the form

$$\int_{\Omega} w \left(\frac{\partial(\bar{\psi} \rho_g h_g)}{\partial t} - \left(\frac{\partial(\bar{\psi} p_g)}{\partial t} + u_{j,g} \frac{\partial p_g}{\partial x_j} \right) - h_{cv} (\bar{T} - T_g) - \sum_k \left(\dot{\omega}_{s,fk}''' - \dot{\omega}_{s,dk}''' \right) h_{g,k} \right) d\Omega \quad (3.563)$$

$$- \int_{\Omega} \frac{\partial w}{\partial x_j} (\rho_g u_{j,g} h_g) d\Omega - \int_{\Omega} \frac{\partial w}{\partial x_j} q_j^{h,g} d\Omega \quad (3.564)$$

$$+ \int_{\Gamma \setminus \Gamma_{AB}} w (\rho_g u_{j,g} h_g) n_j d\Gamma + \int_{\Gamma \setminus \Gamma_{AB}} w q_j^{h,g} n_j d\Gamma + \int_{\Gamma_{AB}} w J_A^H d\Gamma = 0. \quad (3.565)$$

where J_A^H is the imposed flux on the porous side (A) of the Γ_{AB} interface. A detailed description of the coupling boundary flux is given in [Coupling Boundary Conditions](#).

Low-Mach Enthalpy Equation

The enthalpy equation within the low-Mach fluid region, to be solved for the fluid temperature T , is

$$\frac{\partial(\rho h)}{\partial t} + \frac{\partial(\rho u_j h)}{\partial x_j} = - \frac{\partial q_j^h}{\partial x_j} - \frac{\partial q_j^r}{\partial x_j} + \left(\frac{\partial p}{\partial t} + u_j \frac{\partial p}{\partial x_j} \right) + \tau_{ij} \frac{\partial u_i}{\partial x_j} \quad (3.566)$$

where h is the mixture-averaged fluid enthalpy, q_j^r is a source term due to radiation absorption and emission, and p is the fluid pressure. The diffusive flux vector is given by

$$q_j^h = -\lambda \frac{\partial T}{\partial x_j} + \sum_k \rho h_k Y_k \hat{u}_{jk}, \quad (3.567)$$

where λ is the mixture thermal conductivity, h_k is the enthalpy of species k , Y_k is the mass fraction of species k , and \hat{u}_{jk} is the diffusion velocity of species k in the j direction.

Integrating (3.566) over a CVFEM control volume and using the Gauss divergence theorem on the advective and diffusive flux terms, yields the integral form of the enthalpy equation to be solved,

$$\int_{\Omega} \frac{\partial(\rho h)}{\partial t} d\Omega + \int_{\Omega} \frac{\partial q_j^r}{\partial x_j} d\Omega - \int_{\Omega} \left(\frac{\partial p}{\partial t} + u_j \frac{\partial p}{\partial x_j} \right) d\Omega - \int_{\Omega} \tau_{ij} \frac{\partial u_i}{\partial x_j} d\Omega \quad (3.568)$$

$$+ \int_{\Gamma} (\rho u_j h) n_j d\Gamma + \int_{\Gamma} q_j^h n_j d\Gamma = 0. \quad (3.569)$$

The boundary flux terms are then split into contributions on the interface between regions A and B and off the interface so that they may be treated separately. The enthalpy equation then takes the form

$$\int_{\Omega} \frac{\partial(\rho h)}{\partial t} d\Omega + \int_{\Omega} \frac{\partial q_j^r}{\partial x_j} d\Omega - \int_{\Omega} \left(\frac{\partial p}{\partial t} + u_j \frac{\partial p}{\partial x_j} \right) d\Omega - \int_{\Omega} \tau_{ij} \frac{\partial u_i}{\partial x_j} d\Omega \quad (3.570)$$

$$+ \int_{\Gamma \setminus \Gamma_{AB}} (\rho u_j h) n_j d\Gamma + \int_{\Gamma \setminus \Gamma_{AB}} q_j^h n_j d\Gamma + \int_{\Gamma_{AB}} w J_B^h d\Gamma = 0, \quad (3.571)$$

where J_B^h is the imposed flux on the fluid side (B) of the Γ_{AB} interface. A detailed description of the coupling boundary condition is given in [Coupling Boundary Conditions](#).

Coupling Boundary Conditions

Coupling enthalpy transport between the porous and fluid regions is complicated by the use of a two temperature model in the porous region.

To resolve this complication the energy flux applied to the fluid region has a diffusive/conductive component from the gas phase in the porous region, an advective component from the gas phase in the porous region, a convective component from the condensed phase in the porous region, and a penalty coefficient to enforce temperature continuity between the porous gas phase and the fluid. This takes the form

$$J_B^h = J_{A,g}^{\text{diff}} + J_{A,g}^{\text{adv}} + J_{A,c}^{\text{conv}} + \left(\frac{\lambda}{h} \right) (T_f - T_g), \quad (3.572)$$

where $J_{A,g}^{\text{diff}}$ is the diffusive energy transport from the porous gas phase, $J_{A,g}^{\text{adv}}$ is the advective energy transport from the porous gas phase, $J_{A,c}^{\text{conv}}$ is the convective energy transport from the porous condensed phase, and $\overline{\left(\frac{\lambda}{h}\right)}$ is the averaged thermal conductivity / mesh size between the porous and fluid regions. As with the flow coupling boundary conditions this same penalty coefficient is used in both regions to get the best energy conservation on coarse meshes.

The advective energy transport component takes the form

$$J_{A,g}^{\text{adv}} = F_B h_{AB}, \quad (3.573)$$

where h_{AB} is the upwinded interface enthalpy (i.e. it is either h_A or h_B depending on the direction of F_B). The convective component from the condensed phase has the form

$$J_{A,c}^{\text{conv}} = (1 - \bar{\psi}) \frac{h_{cv}}{a_s} (T_f - T_c), \quad (3.574)$$

where h_{cv} is the volumetric heat transfer coefficient of the porous region and a_s is the specific surface area (m^2/m^3) of the porous medium. This formulation of the convective component assumes that the convective heat transfer between the condensed phase and the free fluid is consistent with the convective heat transfer in the bulk of the porous medium that results in the volumetric heat transfer term of the bulk equations.

The coupling back to the porous region is derived based on the assumption that

$$J_{A,g}^h + J_{A,c}^h = J_B^{\text{diff}} + J_B^{\text{adv}}, \quad (3.575)$$

that is, the fluid region applies advective and diffusive energy transport components to the porous region as a whole. The flux applied to the condensed phase is assumed to be the same as the convective flux component it applies to the free fluid,

$$J_{A,c}^h = (1 - \bar{\psi}) \frac{h_{cv}}{a_s} (T_c - T_f). \quad (3.576)$$

The flux applied to the porous gas phase is then given by

$$J_{A,g}^h = J_B^{\text{diff}} + J_B^{\text{adv}} - J_{A,c}^h + \overline{\left(\frac{\lambda}{h}\right)} (T_g - T_f), \quad (3.577)$$

where the advective component is computed in the same manner as is done for the advective flux applied to the fluid region,

$$J_B^{\text{adv}} = F_A h_{AB}. \quad (3.578)$$

Species Transport

Bulk Equations

Porous Gas-Phase Species Equation

The gas-phase species equation within a porous region, to be solved for the gas-phase mass fraction $Y_{k,g}$ of species k , is

$$\frac{\partial(\bar{\psi}\rho_g Y_{k,g})}{\partial t} + \frac{\partial(\rho_g u_{j,g} Y_{k,g})}{\partial x_j} = -\frac{\partial q_{kj}^{Y,g}}{\partial x_j} + \left(\dot{\omega}_{s,fk}''' - \dot{\omega}_{s,dk}'''\right) + \left(\dot{\omega}_{g,fk}''' - \dot{\omega}_{g,dk}'''\right) \quad (3.579)$$

where $\left(\dot{\omega}_{s,fk}''' - \dot{\omega}_{s,dk}'''\right)$ is the formation and destruction of gas-phase species due to heterogeneous reactions, and $\left(\dot{\omega}_{g,fk}''' - \dot{\omega}_{g,dk}'''\right)$ is the formation and destruction of gas-phase species due to homogeneous reactions. The gas-phase species diffusion flux vector $q_{kj}^{Y,g}$ is modeled as

$$q_{kj}^{Y,g} = -\bar{\psi}\rho_g D_{k,g} \frac{\partial Y_{k,g}}{\partial x_j}, \quad (3.580)$$

where $D_{k,g}$ is the gas-phase mass diffusivity for species k . Note that if the mass diffusivities are not equal for all species, then an additional correction is required to maintain mass conservation.

Multiplying (3.579) by an arbitrary test function w and integrating over the domain Ω while integrating the advection and diffusion terms by parts, yields the variational form of the species equation that is solved for $Y_{k,g}$ using the Galerkin finite element method,

$$\int_{\Omega} w \left(\frac{\partial(\bar{\psi}\rho_g Y_{k,g})}{\partial t} - \left(\dot{\omega}_{s,fk}''' - \dot{\omega}_{s,dk}'''\right) - \left(\dot{\omega}_{g,fk}''' - \dot{\omega}_{g,dk}'''\right) \right) d\Omega \quad (3.581)$$

$$- \int_{\Omega} \frac{\partial w}{\partial x_j} (\rho_g u_{j,g} Y_{k,g}) d\Omega + \int_{\Gamma} w (\rho_g u_{j,g} Y_{k,g}) n_j d\Gamma \quad (3.582)$$

$$- \int_{\Omega} \frac{\partial w}{\partial x_j} q_{kj}^{Y,g} d\Omega + \int_{\Gamma} w q_{kj}^{Y,g} n_j d\Gamma = 0. \quad (3.583)$$

The boundary flux terms are then split into contributions on the interface between regions A and B and off the interface so that they may be treated separately. The species equation then takes the form

$$\int_{\Omega} w \left(\frac{\partial(\bar{\psi}\rho_g Y_{k,g})}{\partial t} - \left(\dot{\omega}_{s,fk}''' - \dot{\omega}_{s,dk}'''\right) - \left(\dot{\omega}_{g,fk}''' - \dot{\omega}_{g,dk}'''\right) \right) d\Omega \quad (3.584)$$

$$- \int_{\Omega} \frac{\partial w}{\partial x_j} (\rho_g u_{j,g} Y_{k,g}) d\Omega - \int_{\Omega} \frac{\partial w}{\partial x_j} q_{kj}^{Y,g} d\Omega \quad (3.585)$$

$$+ \int_{\Gamma \setminus \Gamma_{AB}} w (\rho_g u_{j,g} Y_{k,g}) n_j d\Gamma + \int_{\Gamma \setminus \Gamma_{AB}} w q_{kj}^{Y,g} n_j d\Gamma + \int_{\Gamma_{AB}} w J_A^{Y_k} d\Gamma = 0. \quad (3.586)$$

where $J_A^{Y_k}$ is the imposed flux on the porous side (A) of the Γ_{AB} interface. A detailed description of the coupling boundary flux is given in *Coupling Boundary Conditions*.

Low-Mach Species Equation

The species equation within the low-Mach fluid region, to be solved for the mass fraction Y_k for species k , is

$$\frac{\partial(\rho Y_k)}{\partial t} + \frac{\partial(\rho u_j Y_k)}{\partial x_j} = -\frac{\partial q_{kj}^Y}{\partial x_j} + \dot{\omega}_k''' \quad (3.587)$$

where $\dot{\omega}_k'''$ is the volumetric mass formation rate of species Y_k , and the diffusive flux vector is given by

$$q_{kj}^Y = -\rho \hat{u}_{j,k} Y_k, \quad (3.588)$$

with $\hat{u}_{j,k}$ being the species diffusion velocity. Several forms for this velocity are possible, with the simplest being

$$\hat{u}_{j,k} = -D \frac{1}{Y_k} \frac{\partial Y_k}{\partial x_j} \quad (3.589)$$

for equal mass diffusivities D for all species. A more complex form is needed for unequal mass diffusivities, which is not presented here.

Integrating (3.587) over a CVFEM control volume and using the Gauss divergence theorem on the advective and diffusive flux terms yields the integral form of the species equation that is solved,

$$\int_{\Omega} \frac{\partial(\rho Y_k)}{\partial t} d\Omega - \int_{\Omega} \dot{\omega}_k''' d\Omega + \int_{\Gamma} (\rho u_j Y_k) n_j d\Gamma + \int_{\Gamma} q_{kj}^Y n_j d\Gamma = 0. \quad (3.590)$$

The boundary flux terms are then split into contributions on the interface between regions A and B and off the interface so that they may be treated separately. The species equation then takes the form

$$\int_{\Omega} \frac{\partial(\rho Y_k)}{\partial t} d\Omega - \int_{\Omega} \dot{\omega}_k''' d\Omega + \int_{\Gamma \setminus \Gamma_{AB}} (\rho u_j Y_k) n_j d\Gamma + \int_{\Gamma \setminus \Gamma_{AB}} q_{kj}^Y n_j d\Gamma + \int_{\Gamma_{AB}} w J_B^{Y_k} d\Gamma = 0. \quad (3.591)$$

where $J_B^{Y_k}$ is the imposed flux on the fluid side (B) of the Γ_{AB} interface. A detailed description of the coupling boundary condition is given in [Coupling Boundary Conditions](#).

Coupling Boundary Conditions

Coupling species transport across the porous-fluid interface is relatively simple compared to enthalpy transport. As with the flow coupling Robin style boundary conditions are applied on both the porous and fluid regions, but with both diffusive and advective flux components.

For the flux of a species k this takes the form:

$$J_A^{Y_k} = J_B^{\text{diff}} + F_A \rho_g Y_{k,AB} + \left(\frac{D_k \rho}{h} \right) (Y_{k,A} - Y_{k,B}) \quad (3.592)$$

$$J_B^{Y_k} = J_A^{\text{diff}} + F_B \rho Y_{k,AB} + \left(\frac{D_k \rho}{h} \right) (Y_{k,B} - Y_{k,A}) \quad (3.593)$$

where $Y_{k,AB}$ is the upwinded interface mass fraction, equivalent to h_{AB} from the enthalpy coupling. Once again the same penalty coefficient is used on each side in order to get good mass conservation even on coarse meshes.

Volume of Fluid Model



Beta Capability

The volume-of-fluid capability is a beta feature. The numerics for laminar isothermal flow are well tested, but advanced features such as non-isothermal reacting and evaporating simulations are more experimental.

The volume-of-fluid (VOF) capability allows simulation of two-phase systems. The phases are tracked with a conserved scalar, α , which is the volume fraction of fluid in a given control volume. This scalar is 1 in the liquid, 0 in the gas, and between 0 and 1 in the transition region.

There are a number of numerical challenges associated with multi-phase modeling using a diffuse interface. Many of these challenges exist regardless of whether one uses a level set or a volume-of-fluid approach, or some hybrid of the two.

Governing Equation

The basic advection equation for the volume fraction (α) in VOF equations is

$$\frac{\partial \alpha}{\partial t} + \vec{u} \cdot \nabla \alpha = S_\alpha \quad (3.594)$$

Expanding the conservative form of the convection term with the chain rule as

$$\nabla \cdot (\vec{u} \alpha) = \vec{u} \cdot \nabla \alpha + \alpha (\nabla \cdot \vec{u}) \quad (3.595)$$

we can express the governing equation in corrected conservative form as

$$\frac{\partial \alpha}{\partial t} + \nabla \cdot (\vec{u} \alpha) = \alpha (\nabla \cdot \vec{u}) + S_\alpha \quad (3.596)$$

where the right hand side term is 0 for an incompressible flow with no phase change. The standard advection operator is typically too diffusive for practical use, so one of two approaches are typically used:

1. A geometric advection operator [85, 86, 87, 88]
2. An additional interface compression term [89, 90]

Several varieties of geometric advection operators have been proposed in the literature [85, 86, 87, 88], many of which are only applicable for 2D quadrilateral meshes. Those that can be extended to higher dimensions often do so at considerable complexity. For a more detailed discussion of the different options, see [91]. The form implemented here uses the interface compression approach to modify the governing equation to be

$$\frac{\partial \alpha}{\partial t} + \nabla \cdot (\vec{u} \alpha) + \nabla \cdot (\vec{u}_c \alpha (1 - \alpha)) = \alpha (\nabla \cdot \vec{u}) + S_\alpha \quad (3.597)$$

where \vec{u}_c is the compressive velocity. A common form for this is

$$\vec{u}_c = C_\alpha |\vec{u}| \vec{n} \quad (3.598)$$

where C_α is a constant and \vec{n} is the normal vector at the phase interface.

When phase change is present, the source term S_α is non-zero. The volume source of liquid due to evaporation is

$$S_\alpha = -\frac{\dot{m}_{evap}}{\rho_L} \quad (3.599)$$

The evaporation term is calculated using the method described by Hardt and Wondra [92] which converts an evaporative mass flux to a volume source using the interphase area, A , and the mass flux predicted by the evaporation model, \dot{m}_{evap}'' , as

$$\dot{m}_{evap} = \dot{m}_{evap}'' A \quad (3.600)$$

$$A = W |\nabla \alpha| \frac{\int |\nabla \alpha| dV}{\int W |\nabla \alpha| dV} \quad (3.601)$$

$$W = \alpha \quad (3.602)$$

One variation of this that we explored is $W = (1 - \alpha)^2$, which is well suited for a diffuse interface approach. We find that the form by Hardt and Wondra is sometimes destabilizing to the diffuse interface and leads to an artificially high surface regression rate. For a very simple 1D problem or a sharp interface method, the $W = (1 - \alpha)^2$ weighting scheme places all the evaporation outside the liquid and does not correctly predict the surface regression rate. However, it can be better for a diffuse interface representation since evaporation acts as a stabilizing liquid sink to counteract interface diffusion. When using a sharp interface method or a 1D problem where there is no interface diffusion, the weighting of Hardt and Wondra ($W = \alpha$) should be used.

Different evaporation models can be used to define \dot{m}_{evap}'' . The model used by Hardt and Wondra uses deviation from the saturation temperature to define the rate as

$$\dot{m}_{evap}'' = K (T - T_{sat}) \quad (3.603)$$

$$K = \frac{2\beta}{2 - \beta} \frac{h_{fg}}{\sqrt{2\pi R}} \frac{\rho_g}{T_{sat}^{1.5}} \quad (3.604)$$

This form is only physically applicable to either fully saturated gasses, or interface temperatures above the boiling point since it does not take into account the gas composition. As such, this model only allows positive evaporation rates (so no condensation) in order to produce realistic behavior during heat-up and away from the hot regions of the simulation. When using a multi-species gas, the more complete Hertz-Knudsen-Langmuir model can be used, where the rate is driven by the difference between the partial pressure of the vapor and the saturation pressure. This model will be added to the available evaporation models once the multi-species gas treatment is complete.

Interface Reconstruction

When the phase interface is represented by a volume-of-fluid variable, numerical diffusion in the normal advection operator causes the interface (cells with volumes of fluid between 0 and 1) to become wider over time. Although the total volume of liquid is conserved, the volume inside the $\alpha = 0.5$ isosurface may change as a result, and the effect of the higher density liquid will be felt well outside the correct liquid interface location. Techniques to counter-act this diffusion in volume-of-fluid include using compressive advection operators in the VOF equation, using a separate anti-diffusive sharpening step, using a geometric reconstruction of the interface to define the advection, and using sharpening and liquid relocation schemes.

With a level set approach, the interface location is defined by the $\phi = 0$ contour. By definition, the interface does not become diffuse, but there are no guarantees of global conservation of liquid either. Techniques to address this typically involve doing a volume-conserving redistancing operation to the level set field. Such operations can preserve the total volume of liquid, but may not preserve the local shape of the liquid-gas interface.

In both VOF and level set approaches, calculation of the interface curvature is required to determine surface tension forces. This is done by taking the divergence of the interface normal vector (\vec{n}).

$$\kappa = -\nabla \cdot \vec{n} \quad (3.605)$$

It should be immediately apparent that κ will contain not only discretization error from the divergence operator, but also discretization error from the calculation of the normal vector, which is itself a gradient of some other quantity (call it α_s here, it could be the level set field, the volume of fluid field, or a smoothed version of either).

$$\vec{n} = \frac{\nabla \alpha_s}{|\nabla \alpha_s| + \epsilon} \quad (3.606)$$

The simple approach to use $\alpha_s = \alpha$ results in considerable noise in the interface normal vector calculation, particularly in regions of high curvature relative to mesh sizes. In the coupled level set VOF approach (CLSVOF), α_s is the companion level set distance function that is either advected along with the volume fraction or redistanced from the volume fraction contour at 0.5 (so $\alpha_s = \phi$).

In Fuego we have implemented both a CLSVOF approach and a diffusive approach, where α_s is a diffusively smoothed version of the volume fraction field, α , using a user-specified Fourier

number (Fo) and number of iterations.

$$\alpha_s^0 = \alpha \quad (3.607)$$

$$\alpha_s^{n+1} = \alpha_s^n + Fo \Delta x^2 \nabla^2 \alpha_s^n \quad (3.608)$$

In addition, recall that the analytical curvature for a spherical drop is a function of radius: $\kappa = 2/r$. This means that if κ is calculated in cells that span an interface of finite, non-zero width, there will also be a spatial variation of κ across that interface. This means that although in reality the surface tension force is applied sharply at the interface, in a diffuse interface scheme it will be applied over a finite interface thickness of several cells.

This noise and spatial variation in the curvature translates directly into noise in the surface tension force and is the sole cause of the so-called “parasitic currents” that plague diffuse interface methods. Many authors have demonstrated that when using a uniform, analytically prescribed curvature the parasitic currents in the classical static drop test drop to machine precision levels if the forces are otherwise properly balanced.

There are various techniques for improving the curvature calculation. Calculation of the normal vector when the VOF field is sharp is very noisy, so VOF schemes often use a smoothed field or higher order least squares technique. Coupled level set VOF (CLSVOF) carries a level set field along to use for calculating this normal vector on a smooth field (the level set field).

Finally, once the interface has been advected and sharpened, and the curvature has been properly calculated, the surface tension force must be integrated into the momentum and continuity equations in such a manner as to not induce spurious velocities. This is known as the “balanced force” technique, and involves including a surface tension force vector consistently with the pressure gradient.

Continuity Equation

In order to handle the high density ratios in multiphase simulations, the form of the continuity equation is modified by dividing both sides of the equation by density. Interpolation of nodal quantities to subcontrol surfaces is an important consideration to note. Quantities interpolated to faces are indicated with an f subscript in the following equations, while nodal quantities are given no subscript.

The regular form of the pressure Poisson equation in discrete form is

$$\sum_f (\tau_f \nabla p_f \cdot A_f) = \sum_f \left((\rho \vec{u})_f \cdot A_f + (\tau \nabla p)_f \cdot A_f \right) \quad (3.609)$$

which includes the projected nodal gradient of pressure ∇p . In the prior form, the projected nodal gradient ∇p was calculated using the divergence theorem as

$$\nabla p = \frac{1}{V} \sum_f p_f A_f \quad (3.610)$$

For the VOF implementation we use the method described by Francois et al. [93] to calculate a projected nodal density-scaled force term (F) based on calculation of forces at faces, which includes both pressure gradient and surface tension forces. The balanced force term is

$$\vec{F} = \nabla p - \kappa \sigma \nabla \alpha \quad (3.611)$$

and the projection from subcontrol faces to nodes is done using the method by Francois as

$$F_j = \frac{\sum_f (\nabla p_{f,j} - (\sigma \kappa)_f \nabla \alpha_{f,j}) / \rho_f}{\sum_f |A_{f,j}|} \quad (3.612)$$

This term is updated after the solution to the pressure equation, when updated values for pressure and volume-of-fluid have both been calculated in order to keep them synchronized. This nodal force term is included as a source term in the momentum equation (negative, scaled by volume, and multiplied by nodal density) and can be added to the fourth order pressure projection scheme in discrete form as

$$\sum_f \left(\left(\frac{\tau}{\rho} \right)_f \nabla p_f \cdot A_f \right) = \sum_f \left(\vec{u}_f \cdot A_f + \left(\tau \vec{F} \right)_f \cdot A_f + \left(\frac{\tau}{\rho} \right)_f (\sigma \kappa)_f \nabla \alpha_f \cdot A_f \right) \quad (3.613)$$

where \vec{F} is the projected nodal density-weighted balanced force, κ is the curvature, and σ is the surface tension.

The advantage of this projection scheme over a simpler approach where one might define the nodal force term (as done by Lin et al. [94]) as

$$F = \frac{1}{V} \left(\sum_f p_f A_f - \sigma \kappa \sum_f \alpha_f A_f \right) \quad (3.614)$$

is that in such a scheme the evaluation of curvature (κ) occurs at a different location than the gradient of the volume-of-fluid term ($\nabla \alpha$). In the presence of gradients in κ —which are guaranteed to occur in real problems—this introduces a force imbalance that can cause significant spurious non-physical currents in the fluid. Spatial inaccuracies in curvature are the primary cause of spurious currents in VOF simulations – and a baseline test for any balanced force implementation is to see how a static drop with a prescribed (not calculated) value for curvature behaves. In a properly balanced force implementation the resulting parasitic currents should be basically zero (on the order of machine precision). However, a weakness of relying solely on a prescribed curvature test is that it fails to show the force imbalance that arises in the projected nodal force calculation due to interpolation of the curvature. The form published by Francois et al. [93] and implemented in Sierra/Fuego not only recovers machine-precision parasitic currents with the prescribed curvature test, but also keeps them acceptably low when using calculated curvatures.

In addition, when there is phase change there is an additional source term on the right hand side of this equation, which is

$$S = \dot{m}_{evap} \left(\frac{1}{\rho_g} - \frac{1}{\rho_L} \right) \quad (3.615)$$

Property Evaluation

When using VOF, you are required to provide three material models. There is one model for the liquid, one for the gas, and one top level model. The top level model simply specifies the names for the materials to use for gas and liquid, as well as specifying properties like surface tension. Properties needed for flow calculations are combined from the liquid and gas properties using the following rules

A simple averaging is used for non-specific quantities,

$$\phi = \alpha \phi_L + (1 - \alpha) \phi_g \quad (3.616)$$

where ϕ is thermal conductivity (k), density (ρ), viscosity (μ), absorption (a), or density-pressure-derivative ($\frac{\partial \rho}{\partial p}$).

For specific quantities, such as specific heat (C_p) and enthalpy (h), a density-weighted average is used

$$\phi = \frac{\alpha \rho_L \phi_L + (1 - \alpha) \rho_g \phi_g}{\rho} \quad (3.617)$$

For properties that are only relevant for the gas phase, the property value is simply copied from the gas phase

$$\phi = \phi_g \quad (3.618)$$

Examples include species enthalpy, mass diffusivity, and molecular weight.

3.3 Particles

Introduction

The transport of particles through a gas-phase flow is of importance to a tremendous range of applications. Applications in the area of combustion and fire science include fuel sprays, suppressant transport and metal particle combustion [95, 96, 97, 98, 99, 100, 101, 102]. These applications typically have a strong coupling between the heat and mass transfer. For example, fuel spray combustion is typically limited by the diffusion of the oxidizer towards the particle. In fire suppressant distribution, the cooling associated with the evaporating suppressant can dramatically slow suppressant evaporation. In metal particle combustion, in order for the metal oxide combustion product to condense out, the enthalpy of condensation must be dissipated; this energy dissipation is a combination of radiative and conductive transport, each of which results in differing heat flux consequences. Also relevant are the transport of contaminants through the atmosphere and the dynamics of clouds [103]. A large number of industrial processes share similar physics including powder manufacturing, painting, coating and ink-jet printing.

This report describes the development of a Lagrangian particle and droplet transport model and its integration with a computational fluid dynamics (CFD) code that solves, on an Eulerian mesh, the

continuum phase. Conservation of mass, momentum and energy are considered for the coupled system allowing combustion along with evaporating and condensing particles. Since examples of this type of flow are typically sprays, this model is sometimes referred to as a spray model, but it can handle general classes of particulate flows. This model is developed to be suitable for modeling evaporating, condensing or combusting flows of particles in continuum gas-phase flows. This model is based partly on the initial implementation of a dilute spray model in the Vulcan fire-physics computational modeling code [104, 105] as described in [106].

Two significant limitations are stipulated that lead to the simplified conservation equations employed. First, the spray must be dilute, that is the volume fraction of the particle phase must be small (i.e. less than 10 percent). Second, the physical density of the particle should be orders of magnitude greater than the continuum (gas) phase and the particle Reynolds numbers should not be too large or additional terms will appear in the particle evolution equations [107].

The Spray Equation

For given physical properties of the particle (composition, density, etc.), the particle field is characterized by the particle locations, velocities, radii and temperatures. This can be expressed in terms of a particle distribution function, f , so that:

$$f(\mathbf{x}_p, \mathbf{u}_p, r_p, T_p; t) d\mathbf{u}_p dr_p dT_p \quad (3.619)$$

is the probable number of droplets per unit volume at location \mathbf{x}_p in the velocity range $(\mathbf{u}_p, \mathbf{u}_p + d\mathbf{u}_p)$, the size range $(r_p, r_p + dr_p)$, and the temperature range $(T_p, T_p + dT_p)$. The evolution of this particle distribution can be described by an equation of the Fokker-Planck form [108]:

$$\frac{df}{dt} + \nabla \cdot (\mathbf{u}_p f) + \nabla_u \cdot \left(\frac{d\mathbf{u}_p}{dt} f \right) + \frac{d}{dr_p} \cdot \left(\frac{dr_p}{dt} f \right) + \frac{d}{dT_p} \cdot \left(\frac{dT_p}{dt} f \right) = \left(\frac{df}{dt} \right)_{coll} + \left(\frac{df}{dt} \right)_{brk}. \quad (3.620)$$

Here, expressions for the particle acceleration, evaporation and heating are required in the third through fifth terms on the left-hand side. Similar models for particle collision and breakup appear on the right-hand side. Such models are available in the literature [100, 109] and are described in the earlier report [106]. Unfortunately, Eq. (3.620) is a differential equation in nine dimensions, a fact which makes direct numerical evolution prohibitive in the general case. The standard alternative is to represent f using a fine-grained distribution and Monte Carlo methods. That is, f is represented by a sufficiently large number of discrete distributions, each representing a number of particles, N_p , with the same particular characteristics $(\mathbf{x}_p, \mathbf{u}_p, r_p, T_p, t)$. All of the N_p particles in a fine-grained distribution share the same evolution equation, and f is found by summing over the discrete distributions.

In this manner, the evolution of f can be described using evolution equations for individual particles. Such evolution equations are provided in *Particle Transport Model*.

Because it is typically prohibitive to track all of the particles in a flow, representative parcels of particles are instead tracked. The particles in a given parcel share a common origin and common

material properties. To further simplify the parcel evolution equations, each parcel consists of mono-disperse (single-diameter) particles so that all particles in the parcel are described by the same set of evolution equations. For flows where the particle size is distributed over a range of values, it is still necessary to track a statistically significant number of parcels to reproduce the mean behavior. Typically, a large number of parcels (tens or hundreds of thousands) are tracked to describe the evolution of the particle field.

A Combined Eulerian-Lagrangian Approach

The typical approach to CFD is to employ Eulerian descriptions of the flow field. Such an Eulerian formulation is employed to evolve the gas-phase continuum flow in the present case using standard methods [110]. To evolve the fine-grained distribution as indicated above, a Lagrangian approach is necessary [111, 112, 113, 114]. Such an approach has been used in other CFD applications including, for example, the popular internal combustion engineering simulation code, KIVA [100]. The coupling between the Eulerian and Lagrangian fields is key to capturing certain relevant physics, and this coupling is described in detail in *Particle Transport Model* and in *Coupling the Lagrangian and Eulerian Fields*.

The present paper presumes that turbulent flow fields will be of interest, and that these turbulent flows cannot be fully resolved. Then, in addition to the continuity, momentum, species, and energy equations, there will be representations for the turbulent fluctuations. It is common to employ two-equation models in Reynolds-averaged Navier-Stokes (RANS) approaches while large-eddy simulation (LES) techniques employ estimates of the subgrid fluctuations based on resolved quantities. For the present purposes, the $k - \epsilon$ turbulence model [115] will be presumed with extensions to other methods being straightforward.

When a particle collides with a solid wall, it is assumed to adhere to the wall if the impact velocity (kinetic energy) is sufficiently high, and bounces otherwise. In general, adherence is the predominant result of collisions for the particles considered here. It is well known that fine powders can be convectively lifted from surfaces and transported elsewhere, but this is beyond the scope of the current study. Models for the particle breakup due to hydrodynamic forces and for particle collisions are also available. For these purposes, models developed elsewhere and available in the literature [100, 109] are employed.

Particle Transport Model

In this section, the equations describing the evolution of parcels of particles are presented. Models are presented for the particle motion and for heat and mass transfer (ie. evaporation and combustion).

Particle Acceleration and Trajectories

Particles with densities much greater than that of the fluid phase (solid or liquid particles in gaseous flows) are primarily affected by drag forces and body forces. In this limit where $\rho_p \gg \rho_g$, the particle acceleration is written [101]:

$$\frac{du_{p,i}}{dt} = \frac{3\rho_g C_D |\mathbf{u}_g - \mathbf{u}_p|}{4\rho_p d_p} (u_{g,i} - u_{p,i}) + \left(\frac{\rho_p - \rho_g}{\rho_p} \right) g_i \quad (3.621)$$

where $u_{p,i}$ and $u_{g,i}$ are the i^{th} component of the particle and gas velocities, respectively, $|\mathbf{u}_g - \mathbf{u}_p|$ is the vector magnitude of the velocity differences, ρ_p and ρ_g are the particle and gas densities, and g_i is the i^{th} component of the acceleration due to body forces. The particle diameter is d_p and this should be considered to be the equivalent particle diameter corresponding to a spherical particle. The effects of non-sphericity on the acceleration can be accounted for through the drag coefficient, C_D . The gas-velocity to be employed in (3.621) is taken from the Eulerian solution of the continuum field. For turbulent flows, the sum of the mean (resolved) velocity and a perturbation to that mean, accounting for the turbulent fluctuations, both contribute to the gas velocity. The effects of turbulent fluctuations are described in the following section, *Particle Dispersion and Turbulence*.

For a spherical particle, the drag coefficient is modeled using standard drag coefficient relations:

$$C_D = \begin{cases} 24(1 + Re_p)^{2/3}/Re_p & \text{for } Re_p < 1000 \\ 0.424 & \text{for } Re_p > 1000 \end{cases} \quad (3.622)$$

The particle Reynolds number, Re_p , is based on the slip velocity and the particle diameter

$$Re_p = \frac{\rho_g d_p |\mathbf{u}_g - \mathbf{u}_p|}{\mu_g} \quad (3.623)$$

Yuen and Chen [116] recommend evaluating the viscosity in (3.623) based on the weighted average of the properties at the gas-phase side of the particle surface (weighted by two-thirds) and the gas-phase properties far from the particle (weighted by one-third), the so called ‘1/3 rule.’ So for properties at the surface and far field identified with a subscripted f and ∞ , respectively, the viscosity would be

$$\mu_g = \mu_\infty/3 + 2\mu_f/3. \quad (3.624)$$

A similar relationship is suggested for other transport coefficients (conductivity, diffusivity, etc.). Note that additional forces are relevant for particles with densities nearer to or less than the continuum phase (bubbles) and for particles with high Reynolds numbers. A comprehensive overview of the force on particles is available from Maxey and Riley [107]. Equation (3.621) can be linearized and written

$$\frac{du_{p,i}}{dt} = \frac{(u_{g,i} - u_{p,i})}{\tau_p} + \left(\frac{\rho_p - \rho_g}{\rho_p} \right) g_i \quad (3.625)$$

$$\tau_p = \frac{4\rho_p d_p}{3\rho_g C_D |\mathbf{u}_g - \mathbf{u}_p|} \quad (3.626)$$

where τ_p is the particle velocity response time. In the small Reynolds number limit where the drag coefficient is inversely proportional to the slip velocity, this linearization is particularly relevant. Given the particle acceleration from Eq. (3.621), the particle trajectories can be determined by integrating the simple ODE

$$\frac{dx_{p,i}}{dt} = u_{p,i} \quad (3.627)$$

Since all the particles within a parcel are the same size, all parcel trajectories are determined by Eqn. (3.627), subject to turbulence effects described below in *Particle Dispersion and Turbulence*.

Particle Dispersion and Turbulence

In CFD modeling of turbulent flows, the full velocity spectrum is generally not resolved. Instead, certain velocity fluctuations are modeled, being represented through the turbulent kinetic energy, k , which is half the sum of the squares of the velocity fluctuations. These velocity fluctuations tend to introduce random fluctuations in the particle velocities that result in real particles being dispersed relative to the mean continuum flow [117, 118]. For Lagrangian particle methods, this phenomenon is modeled in two ways: by perturbing the velocity of parcels of particles and by affecting the spatial extent of the parcel itself.

To account for the effects of the velocity fluctuations on the parcels or particles, the random walk model of Gosman and Ioannides [111], as modified by Shuen et al. [112], is employed. In this approach, the gas velocity employed in equations (3.621), (3.623), (3.625), and (3.626) is the sum of the mean gas velocity, $\langle u_{g,i} \rangle$, and a fluctuating velocity that is sampled from a normal (Gaussian) velocity distribution with a standard deviation given by $\sigma_u = \sqrt{2k/3}$. For LES, k here would be replaced with the subgrid kinetic energy. Sampling from the inverted cumulative distribution function with a random number uniformly distributed between zero and unity, RN , gives the appropriate fluctuating velocity. The total and fluctuating gas velocities are then

$$u_{g,i} = \langle u_{g,i} \rangle + u'_{g,i} \quad (3.628)$$

$$u'_{g,i} = \sqrt{2}\sigma_u \text{erf}^{-1}(2RN - 1) \quad (3.629)$$

where erf^{-1} is the inverse error function. The time during which a given velocity fluctuation affects a given parcel is determined by the expected time that the particle takes to cross the eddy inducing the given velocity fluctuation. Small particles will tend to stay in an eddy for the duration of the eddy lifetime,

$$\tau_e = \sqrt{3/2} C_\mu^{3/4} k / \epsilon \quad (3.630)$$

where ϵ is the turbulent energy dissipation rate and $C_\mu = 0.09$. Larger particles will have sufficient slip velocity to cross the eddy. The eddy-crossing time is estimated as

$$\tau_C = -\tau_p \ln \left[1 - \frac{L_e}{\tau_p |\mathbf{u}_g - \mathbf{u}_p|} \right] \quad (3.631)$$

where the eddy length scale is $L_e = C_\mu^{3/4} k^{3/2} / \epsilon$. The eddy interaction time, that is the time over which a given velocity perturbation affect $u_{g,i}$ in Equations (3.628) and (3.629), is the minimum of τ_e and τ_c . The particles comprising a parcel are presumed to be distributed about the center of the parcel, tracked by Eq. (3.627), in a normal (Gaussian) manner with the standard deviation in each direction given by σ_i , where i is either x , y , or z . This distribution is written

$$f_\sigma(\vec{x}; \vec{x}_o, t) = \frac{N_p}{(2\pi)^{3/2} \sigma_x \sigma_y \sigma_z} \exp \left(- \left[\frac{(x - x_o)^2}{2\sigma_x^2} + \frac{(y - y_o)^2}{2\sigma_y^2} + \frac{(z - z_o)^2}{2\sigma_z^2} \right] \right) \quad (3.632)$$

where N_p is the total number of particles in the parcel and \vec{x}_o is the center of the parcel. The spatial extent of the parcel is thus determined by σ_i . This term is affected by unresolved turbulent fluctuations that act differently on the particles across the parcel. Following Zhou and Yao [113], the spatial extent of the parcel is the mean square displacement over time

$$\sigma_x^2 = \sum \left[u'_{p,i}{}^2 (\Delta t_i)^2 \right] \quad (3.633)$$

where $u'_{p,i}$ satisfies the ODE

$$\frac{du'_{p,i}}{dt} = \frac{(u'_{g,i} - u'_{p,i})}{\tau_p} \quad (3.634)$$

and Δt_i is the time over which $u'_{p,i}$ acts. Equation (3.634) is obtained by subtracting the instantaneous particle equations from the mean particle equations. Note that large particles with large τ_p are not dispersed appreciably.

Mass and Energy Exchange between Particles and the Gas Phase

Particles may exchange mass and energy with the gas phase according to conservation principles across the interface. The physics of mass and energy transfer are described in detail here because the anticipated applications include a wide range of physical phenomena that have not been described together. Included in the phenomena of interest are metal and hydrocarbon particle combustion as well as particle condensation and evaporation. Each of these is anticipated to be strongly energetic in the sense that the product of the evaporation rate with the enthalpy of evaporation and combustion is expected to be substantial. Further, for evaporation and condensation applications, it is expected that vapor pressures will range over a sufficiently wide range that the transition from evaporation to condensation should be correctly described as this often limits evaporation and condensation rates. For metal combustion, the associated temperatures are sufficiently high so that radiative heat transfer must be considered.

The theory for droplet vaporization and combustion has generally been developed based on a large number of simplifications [95, 96] including spherical-symmetry, unity Lewis numbers, droplet surfaces at the boiling temperature, and infinitely fast conduction through the droplet. Recent work provides guidance as to how these assumptions can be relaxed [119].

Theory for spherically symmetric flow

In general, the theory of heat and mass transfer is developed for spherically symmetric systems and then corrected to account for increased transfer associated with advection and asymmetry. In this section, relations are developed based on the assumption of spherical symmetry and corrections is provided in *Extension to multiple oxidizers*.

$$\frac{\partial \rho}{\partial t} + \frac{1}{r^2} \frac{\partial}{\partial r} (\rho r^2 v) = 0 \quad (3.635)$$

can be integrated to give

$$\dot{m}_o = 4\pi \rho r^2 v \quad (3.636)$$

where \dot{m}_o is the rate of mass evaporation from the particle, v is the Stefan velocity, directed normally away from the particle, and ρ is the local vapor density considering both the particle vapor and the continuum gas concentrations. A coordinate transformation to the variable

$$\xi_T = \int_{r_o}^{\infty} \frac{1}{4\pi r^2 (\lambda/c_p)} dr \quad (3.637)$$

or

$$\xi_F = \dot{m}_o \int_{r_o}^{\infty} \frac{1}{4\pi r^2 (\rho D_F)} dr \quad (3.638)$$

which represents the ratio of the Stefan velocity to the thermal diffusion velocity, greatly simplifying the species and energy conservation equations. In Equations (3.637) and (3.638), λ is the thermal conductivity of the vapor, c_p is its specific heat at constant pressure, and D_F is the evaporating species diffusion coefficient. The subscript 0 on the evaporation rate indicates that this evaporation corresponds to that for the spherically-symmetric case; corrections relating the evaporation rate for the spherically-symmetric case to that with finite-slip velocities are provided in *Extension to multiple oxidizers*. In the spherically-symmetric case, conservation equations for conserved scalars, β_k , can be written

$$\frac{\partial \beta_k}{\partial \xi_k} + \frac{\partial^2 \beta_k}{\partial \xi_k^2} = 0 \quad (3.639)$$

for which an analytic solution

$$\beta_k = C_1 + C_2 e^{-\xi_k} \quad (3.640)$$

is readily obtained. In non-reacting, non-radiating flows, any mass fraction or temperature can be a conserved scalar. Other conserved scalars are provided below. The appropriate form of (3.637) and (3.638) to be used depends on the variable represented by (3.639). Since the diffusion coefficients appearing in (3.638) are most important near the particle surface, the diffusion coefficient to be employed is that most relevant at the surface. The choice will be clearly identified below. The application of the Dirichlet boundary conditions at both the surface and far from the

droplet and the application of a Neumann boundary condition at the surface relate the boundary conditions for the conserved scalar to \dot{m}_o through ξ_k

$$\xi_{k,f} = \dot{m}_o \int_{r_o}^{\infty} \frac{1}{4\pi r^2 (\rho D)} dr = \ln \left(1 + \frac{\beta_{\infty} - \beta_f}{-\frac{d\beta}{d\xi_k}|_f} \right) \quad (3.641)$$

where the subscript f indicates quantities evaluated at the droplet surface, the so-called film state. For reacting flows we will consider two such conserved scalars in the present work. The general configuration considered is a fuel droplet reacting in an oxidizing atmosphere; in the event that, for example, an oxidizer droplet is reacting in a fuel atmosphere, then the “oxidizer” and “fuel” described would be switched. Also, if there is no reaction, the results generalize to droplet evaporation where the “fuel” is the evaporating component. Allowing variable properties, the temperature oxidizer coupling function is

$$\beta_{T-O} = \int_{T_o}^T c_p dT + \frac{Y_O W_F q_{comb}}{\nu_O W_O} \quad (3.642)$$

where W_i is the molecular weight and ν_i is the stoichiometric coefficient of species i . The standard enthalpy of combustion for fuel and oxidizer, per unit mass of fuel evaluated at the film temperature, T_f , is

$$q_{comb} = h_{F,f} + \frac{h_{O,f} W_O \nu_O}{W_F \nu_F} - \sum \left[\frac{h_{p,f} W_p \nu_p}{W_F \nu_F} \right] \quad (3.643)$$

where the last summation is taken over the products of reaction, p . Note that when there is no combustion, the second term in Eqn. (3.643) is ignored. For β_{T-O} , the thermal diffusivity is the relevant diffusivity so that Eqn. (3.637) is employed in conjunction with β_{T-O} .

The constants C_1 and C_2 in (3.640) are evaluated using the boundary conditions for the temperature and oxidizer at the droplet surface and in the far field. The boundary conditions employed for temperature are that the heat flux into the particle is balanced by the sum of the enthalpy of vaporization, the heating of the particle and any radiative losses

$$4\pi r^2 \lambda \left(\frac{dT}{dr} \right)_f = -\dot{m}_o c_{p,f} \left(\frac{dT}{\xi_T} \right)_f = \dot{m} h_{vap} + Q_{rad} + m_p c_{v,p} \frac{dT_p}{dt} \quad (3.644)$$

Here, the enthalpy of vaporization is h_{vap} , the particle specific heat is $c_{v,p}$, and the radiative heat loss over the droplet surface is

$$Q_{rad} = 4\pi r_p^2 \alpha (\sigma T_p^4 - G_{in}/4) \quad (3.645)$$

where α is the particle absorptivity, σ is the Stefan-Boltzmann constant, and G_{in} is the incident radiation. The incident radiation is the radiation intensity integrated over all directions, that is, the entire 4π steradian solid angle. If the radiation intensity is I , then $G_{in} = \int_{4\pi} I d\Omega$ where $d\Omega$ is the differential solid angle. If the particle is not opaque, the particle absorptivity will be a function of

the particle size [120]. Note that, in Eqn (3.644), the evaporation rate appears without the subscript 0, indicating that this is the evaporation rate corrected for finite-slip velocities as prescribed in (3.660) below. This is appropriate because the heat and mass flux to the surface are both increased by the relative droplet motion, while the other terms in (3.644) are not affected by that. The oxidizer is presumed to not be absorbed by the surface so that a no-flux boundary condition is employed

$$\frac{dY_O}{d\xi_T} = -Le_{O,f}Y_{O,f}. \quad (3.646)$$

Because the thermal diffusivity is used in definition ξ_T for β_{T-O} , the ratio of the thermal to mass diffusivity in the form of the oxidizer Lewis number appears in (3.646). Taking the derivative of Eqn. (3.642) evaluated at the droplet surface and using (3.644) and (3.646), we obtain

$$\left(\frac{d\beta_{T-O}}{d\xi_T} \right)_f = -h_{vap} - \frac{Q_{rad}}{\dot{m}} - \frac{m_p c_{v,p}}{\dot{m}} \frac{dT_p}{dt} + \int_{T_o}^T \frac{dc_{p,f}}{d\xi_T} dT - \frac{Y_{O,f} W_F q_{comb}}{Le_{O,f} \nu_O W_O} \quad (3.647)$$

that will appear in the denominator of Eqn. (3.641). Also the temperature and oxidizer mass fraction must approach their far field values at large radii. Additional assumptions are required to identify the temperature and the oxidizer mass fraction at the surface; these will be discussed later. Applying these boundary conditions to (3.640) for β_{T-O} provides an expression for the rate of evaporation in terms of ξ_T evaluated at the surface of the particle

$$\xi_{T,f} = m_O \int_{r_o}^{\infty} \frac{1}{4\pi r^2 (\lambda/c_p)} dr = \ln [1 + B_{T-O}] \quad (3.648)$$

where the Spalding transfer number associated with β_{T-O} is

$$B_{T-O} = \frac{\int_{T_o}^T c_{p,\infty} dT - \int_{T_o}^T c_{p,f} dT + q_{comb} \left[\frac{(Y_{O,\infty} - Y_{O,f}) W_F}{\nu_O W_O} \right]}{h_{vap} + \frac{Q_{rad}}{\dot{m}} + \left(\frac{m_p c_{v,p}}{\dot{m}} \frac{dT_p}{dt} \right) - \int_{T_o}^T \frac{dc_{p,f}}{d\xi_T} dT + \frac{Y_{O,f} W_F q_{comb}}{Le_{O,f} \nu_O W_O}} \quad (3.649)$$

Here the subscripts f and ∞ indicate the states at the particle surface (on the gas-phase side of the interface) and the ambient far-field environment, respectively. The denominator in (3.649) represents a variety of potential sinks for the enthalpy at the droplet surface. These sinks include, in the order in which they are written, the enthalpy associated with vaporizing the particle, the radiative losses from the surface, the enthalpy conducted into the particle, the enthalpy flux to the gas-phase due to variable specific heats, and the enthalpy of combustion lost as a consequence of oxidizer leakage. Note that the radiative absorption and emission from the particle surface, Q_{rad} , are included here in the surface boundary condition, but the radiative losses from a flame around the droplet must be accounted for by modifying q_{comb} to provide an effective heat release, the heat release decremented by the near flame radiative losses. As far as the droplet is concerned, these radiative losses are far enough away to not affect the film state except to the extent that radiative flux to the surface is affected.

The above expressions comprise a relatively complete definition of the physics of particle evaporation, combustion, and interaction with a radiative field accounting for variable

thermophysical properties. These expressions are simplified by making the assumption that no oxidizer penetrates a flame to reach the surface, $Y_{O,f} = 0$, and by setting $T_f = T_O$ leading to

$$B_{T-O} = \frac{\int_{T_f}^{T_\infty} c_{p,\infty} dT + \frac{Y_{O,\infty} W_F q_{comb}}{\nu_O W_O}}{h_{vap} \frac{Q_{rad}}{\dot{m}} + \left(\frac{m_p c_{v,p}}{\dot{m}} \frac{dT_p}{dt} \right)} \quad (3.650)$$

where the denominator is referred to as the effective enthalpy

$$h_{eff} = h_{vap} + \frac{Q_{rad}}{\dot{m}} + \left(\frac{m_p c_{v,p}}{\dot{m}} \frac{dT_p}{dt} \right). \quad (3.651)$$

Equation (3.650) is employed in the numerical models. A coupling function for fuel and oxidizer is similar written

$$\beta_{T-O} = \frac{Y_F}{W_F} - \frac{Y_O}{\nu_O W_O} \quad (3.652)$$

For a species that is evaporating, the flux boundary condition is

$$\left(\frac{dY_F}{d\xi_F} \Big|_f \right) = Y_{F,p} - Y_{F,f} \quad (3.653)$$

and the fuel-oxidizer Spalding mass transfer number is

$$B_{F-O} = \frac{Y_{F,f} - Y_{F,\infty} + \frac{Y_{O,\infty} W_F}{\nu_O W_O}}{Y_{F,p} - Y_{F,f}} \quad (3.654)$$

where $Y_{O,f} = 0$ has been assumed as in Eq. (3.650). Note that only one of $Y_{O,\infty}$ or $Y_{F,\infty}$ will be non-zero based on the current assumption of zero leakage through flames; if combustion is occurring then it will be $Y_{F,\infty}$ that is zero. The diffusion coefficient appropriate for the fuel-oxidizer coupling function is that for the fuel so that the second of Eqn (3.637) and (3.638) is used with the diffusion coefficient specifically that of the fuel, and the equivalent of Eqn. (3.648) for the fuel-oxidizer system

$$\xi_{F,f} = \dot{m}_o \int_{r_o}^{\infty} \frac{1}{4\pi r^2 (\rho D_F)} = \ln[1 + B_{F-O}] \quad (3.655)$$

Extension to multiple oxidizers

Multiple oxidizers is discussed in detail later, in *Multiple Oxidizers*.

Correlations for finite slip velocities

The above relationships for the heat and mass transfer are derived for a spherically symmetric field around the droplet and are valid for droplets with zero slip velocity in the absence of buoyancy. Empirical correlations are available in terms of the Nusselt and Sherwood numbers parameterized by Reynolds, Schmidt, and Prandtl numbers to describe the effect of finite slip velocities in modifying the heat and mass transfer by reducing the boundary layer thickness. The Schmidt and Prandtl numbers are defined:

$$Pr = \frac{c_{p,g}\mu_g}{\lambda_g} \quad (3.656)$$

$$Sc = \frac{\mu_g}{\rho_g D_g} \quad (3.657)$$

These quantities with the subscript g represent appropriate averages for transport properties in the gas phase boundary layer around the particle. These gas-phase quantities are evaluated using an appropriate averaging process that will generally be analogous to Eqn. (3.624). The alternative is to tabulate these quantities using Equations (3.664) and (3.665). The Nusselt number describes a dimensionless heat transfer rate to the droplet for a given difference between the ambient and surface temperature,

$$Nu_f = \left(2r_p c_{p,g} \left. \frac{dT}{dr} \right|_f \right) / \left(\int_{T_f}^{T_\infty} c_{p,\infty} dT + \frac{Y_{O,\infty} W_F q_{comb}}{\nu_O W_O} \right) \quad (3.658)$$

The correction to the evaporation rate employed in the present work is based on measurements by Ranz and Marshall (1952)

$$\frac{Nu_f}{Nu_{f,Re=0}} = (1 + 0.3Re^{1/2}Pr^{1/3}) \quad (3.659)$$

This can be introduced into (3.649), and the evaporation rate can be written

$$\dot{m} = \frac{Nu_f}{Nu_{f,Re=0}} \dot{m}_o \quad (3.660)$$

The Nusselt number for zero slip velocity, $Nu_{f,Re=0}$, is included in the relations of the previous section, specifically in Eq. (3.649). A similar correlation,

$$\frac{Sh_f}{Sh_{f,Re=0}} = (1 + 0.3Re^{1/2}Sc^{1/3}) \quad (3.661)$$

can be used for the Sherwood number,

$$Sh_f = \left(-2r_p \left. \frac{dY_F}{dr} \right|_f \right) / \left(Y_{F,f} - Y_{F,\infty} + \frac{Y_{O,\infty} W_F}{\nu_O W_O} \right) \quad (3.662)$$

which is the dimensionless mass transfer coefficient, so that the evaporation rate for finite slip velocities can also be written

$$\dot{m} = \frac{Sh_f}{Sh_{f,Re=0}} \dot{m}_o \quad (3.663)$$

where \dot{m}_o should be taken from Eqn. (3.655).

Evaporation rates and effective diffusivities

The evaporation rate is seen in Eqn (3.648) and (3.655) to depend linearly on an area-weighted gas-phase diffusivity. Since accurate evaluation of this weighting is not feasible for the particle transport model, effect diffusivities,

$$\left(\frac{\lambda}{c_p}\right)_{eff} = \left[4\pi r_p \int_{r_o}^{\infty} \frac{1}{4\pi r^2(\lambda/c_p)} dr\right]^{-1} \quad (3.664)$$

$$(\rho D_F)_{eff} = \left[4\pi r_p \int_{r_o}^{\infty} \frac{1}{4\pi r^2(\rho D_F)} dr\right]^{-1} \quad (3.665)$$

are defined. The one-third rule defined in Eq. (3.624) provides a rough guideline for evaluating these diffusion coefficients in some cases. The effective thermal diffusion coefficient is combined with Equations (3.648) and (3.660) to give an evaporating rate based on thermal diffusion

$$\dot{m} = 4\pi r_p (\rho D_F)_{eff} \frac{Sh_f}{Sh_{f,Re=0}} \ln [1 + B_{T-O}]. \quad (3.666)$$

The strong dependence of the evaporating rate on the diffusion coefficient, coupled with the fact that the diffusion coefficients depend strongly on variations in the compositions and temperature of the gases around the droplet, mean that the reasonable but judicious choice of diffusion coefficients can often match observed experimental measurements. Similarly, the mass transfer driven evaporation rate can be written with the use of Equations (3.655), (3.661), and (3.663)

$$\dot{m} = 4\pi r_p (\rho D_F)_{eff} \frac{Sh_f}{Sh_{f,Re=0}} \ln [1 + B_{T-O}] \quad (3.667)$$

It is necessary that the evaporation rate predicted by Eqn. (3.667) be equal to that predicted by (3.666). These equations show that thermal and mass diffusion vary both through their diffusion coefficients and through difference in boundary layer thickness attributable to finite slip velocities. Both these effects are combined in an effective Lewis number, Le_{eff} , to give

$$Le_{eff} = \frac{\int_{r_o}^{\infty} \frac{dr}{4\pi r^2(\rho D_f)} \frac{Nu_f}{Nu_{f,Re=0}}}{\int_{r_o}^{\infty} \frac{dr}{4\pi r^2(\lambda/c_p)} \frac{Sh_f}{Sh_{f,Re=0}}} \quad (3.668)$$

Equations (3.666), (3.667), and (3.668) are used in the computational model for the evaporation rate. Note that energy and mass conservation must vie the same evaporation rate; this requirement determines the film state as described in the following section, *Closure of film state with effective heat transfer coefficient*.

Closure of film state with effective heat transfer coefficient

The system described by Equations (3.666) and (3.667) is closed with two additional assumptions. First, the film conditions, T_f and Y_f , are related through the Clausius-Clapeyron relationship

$$P_{F,f} = P_{ref} \exp \left[-\frac{h_{vap}}{R} \left(\frac{1}{T_f} - \frac{1}{T_{ref}} \right) \right] \quad (3.669)$$

where the partial pressure gives the mole fraction through $X_F = P_{F,f}/P$ that can subsequently be converted to the mass fraction with the relationship $Y_F = X_F W_F / \sum_k X_k W_k$. As provided by Lefebvre [121], such a relationship is

$$h_{vap} = h_{vap,ref} \left(\frac{T_{crit} - T_f}{T_{crit} - T_{ref}} \right)^{0.38} \quad (3.670)$$

for $T_f < T_{crit}$ and zero otherwise. If the critical point temperature is not provided, the code sets T_{crit} to a very large value, essentially making h_{vap} independent of temperature. Second, the droplet heating is related to the difference between the film temperature and the droplet temperature by assuming an effective internal heat transfer coefficient in the form of an internal Nusselt number, Nu_p , for the particle so that

$$m_p c_{v,p} \frac{dT_p}{dt} = 2\pi r_p Nu_p \lambda_p (T_f - T_p). \quad (3.671)$$

This internal Nusselt number, which is different from that for the external heat transfer indicated in Eqn. (3.660), can be estimated based on the results of numerical studies where the internal droplet was resolved [119, 122]. There the Nusselt numbers for no circulation and for rapid circulation were identified as 6.58 and 17.9, respectively, and a transition region was also identified based on the liquid Peclet number

$$Pe_p = \frac{2\rho_p c_{v,p} U_{surface} r_p}{\lambda_p} \quad (3.672)$$

where

$$U_{surface} = \frac{12.69 |\mathbf{u}_p - \mathbf{u}_g| Re_p^{1/3}}{16} \left(\frac{\mu_g}{\mu_p} \right) \quad (3.673)$$

which is based on the maximum surface velocity, $U_{surface}$. This transition was empirically fitted [119] to

$$Nu_p = 6.58 \left[1.86 + 0.86 \tanh \left[2.245 \log_{10} (Pe_p/30) \right] \right]. \quad (3.674)$$

In [119], both the evaporation rate and the surface temperature were reproduced using Eq. (3.674) with little error compared to simulations incorporating a detailed internal droplet convection model. Naturally, the liquid Peclet number should be zero if the particle is below the freezing temperature of the particle constituent.

The evaporation rates indicated in Equations (3.666) and (3.667) must be equal, subject to the constraints of Equations (3.669) and (3.671) based on the closure approximation employed in this section. Equating Equations (3.666) and (3.667) leads to a nonlinear equation for the surface temperature that is to be solved. This is readily solved using Newton's method as described here. Newton's method is an iterative root-finding method written in the form

$$T_f^{n+1} = T_f^n - g(T_f^n)/g'(T_f^n) \quad (3.675)$$

where $g(T_f) = 0$ is the equation for which the root will be found and the superscript n refers to the iteration number. Equating Equations (3.666) and (3.667) and using (3.668) gives

$$g(T_f) = (1 + B_{T-O})^{Le_{eff}} - 1 - B_{F-O} \quad (3.676)$$

Differentiation gives

$$g'(T_f) = Le_{eff}(1 + B_{T-O})^{(Le_{eff}-1)} \frac{dB_{T-O}}{dT_f} - \frac{dB_{F-O}}{dT_f} \quad (3.677)$$

$$\frac{dB_{T-O}}{dT_f} = \frac{-\left[c_{p,\infty}(T_\infty - T_f) + \frac{Y_{O,\infty}W_F q_{comb}}{\nu_O W_O}\right] \frac{dh_{eff}}{dT_f} - c_{p,\infty} h_{eff}}{h_{eff}} \quad (3.678)$$

$$\frac{dB_{F-O}}{dT_f} = \frac{\left(Y_{F,f} - Y_{F,\infty} + \frac{Y_{O,\infty}W_F}{\nu_O W_O}\right) dY_{F,f}}{(Y_{F,p} - Y_{F,f})^2} \frac{dY_{F,f}}{dT_f} \quad (3.679)$$

$$\frac{dY_{F,f}}{dT_f} = \frac{-W_F W_g \frac{P_{atm}}{P_{ref}} \frac{h_{vap}}{R} \left[\frac{1}{T_f^2} + \frac{0.38}{(T_{crit} - T_f)} \left(\frac{1}{T_f} - \frac{1}{T_{ref}} \right) \right] \exp \left[-\frac{h_{vap}}{R} \left(\frac{1}{T_f} - \frac{1}{T_{ref}} \right) \right]}{W_F - W_g + W_g \frac{P_{atm}}{P_{ref}} \exp \left[-\frac{h_{vap}}{R} \left(\frac{1}{T_f} - \frac{1}{T_{ref}} \right) \right]} \quad (3.680)$$

$$\frac{dh_{eff}}{dT_f} = \frac{2\pi r_p Nu_p \lambda_p}{\dot{m}} - \frac{Le_f [Q_{rad} + 2\pi r_p Nu_p \lambda_p (T_f - T_p)]}{\dot{m}(1 + B_{F-O}) [\ln(1 + B_{F-O})]} \frac{dB_{F-O}}{dT_f} \quad (3.681)$$

In writing these expressions for g' , it is assumed that $Y_{O,f} = 0$ and $T_O = T_f$.

Because of the strong nonlinearities in $g(T_f)$, care must be taken in providing initial conditions to solve these relations. This is conducted through a two-stage procedure. First, the boiling point temperature is identified, then a temperature just under the boiling temperature is used as an initial guess for the iterative solution that determines the film temperature. This procedure arises from a consideration of the shape of the $g(T_f)$. For realistic temperatures, those for which $0 < Y_f < 1$, both $g(T_f)$ and $g'(T_f)$, are monotonically strongly increasing in magnitude. An initial guess with a temperature that is too low (less than T_f) results in a prediction, with the Newton's method, of a very high temperature on the successive iteration due to the small magnitude of the derivative, g' , for a small T . Typically, this second iteration will result in a temperature for which $Y_f > 1$ that leads to negative values of B_m and the iteration diverges into non-physical regimes. However, an

initial guess that is within the physically reasonable regime ($0 < Y_f < 1$) and yet above the final T_f will reliably converge. Therefore, the initial guess of $T = 0.99999T_{boil}$ is used as an initial guess in determining the film temperature. This method has been tested for a wide range of conditions and appears robust except for those scenarios where the denominator of B_T takes on negative values. (Since T_f can be less than T_p and this can result in the denominator of B_T taking on negative values, in which case the iteration may fail.)

Because of the cost and potential stability issues in this method, the fast conduction limit described in *Closure for surface state assuming fast conduction* is used in the code instead.

Closure for surface state assuming fast conduction

The surface state described in the previous section is the most physically realistic state that can be obtained without solving a differential equation for the heat transfer with the droplet. However, determining this state involves the iterative solution of a system of nonlinear equations. In the previous section a robust method of solving these equations is provided, but a simpler approximation may provide suitable results under certain conditions. This simpler approach is to assume that the heat transfer within the droplet is fast relative to the heating of the droplet. This is equivalent to taking the zero Biot number limit, $Bi \approx Nu_f \lambda_f / Nu_p \lambda_p \rightarrow 0$, in Eqn. (3.671), in which case $T_f = T_p$. With the film state determined by the droplet temperature, the film mass fraction is directly obtained from Equations (3.669) and (3.670), and the mass transfer number is obtained from Eq. (3.654) with the evaporation rate following from Eqn. (3.667). The thermal transfer number is obtained from the mass transfer number by equating Equations (3.667) and (3.666), and the droplet heating is obtained by solving for the enthalpy change of the particle in Eq. (3.649) to obtain

$$\frac{m_p c_{v,p}}{\dot{m}} \frac{dT_p}{dt} = -h_{vap} - \frac{Q_{rad}}{\dot{m}} + \frac{\int_{T_f}^{T_\infty} c_{p,\infty} dT + \frac{Y_{O,\infty} W_F q_{comb}}{\nu_O W_O}}{B_{T-O}} \quad (3.682)$$

Sirignano [123] and coworkers have demonstrated that for many conditions, this particular limit is a poor approximation for at least some of the particle lifetime. In the present models, this limit is employed in two situations: (a) if the particle temperature is within 1% of the wet bulb temperature, and (b) if the particle temperature exceeds the wet bulb temperature. As the droplet temperature approaches the wet bulb temperature, employing this limit is inconsequential. In the latter case, the rate of droplet cooling is likely to be over predicted, but this is a scenario for which the convergence of the film state otherwise is not guaranteed. In the interest of creating a more robust model, and because this particular situation is not anticipated to be predominant, we employ this simpler limit.

Conserved scalars and transfer numbers for various applications

In *Correlations for finite slip velocities*, expressions are provided for the particle evaporation rate, (3.667) and (3.666), using a model system of fuel evaporating from the particle and reactive with an oxidizer that diffuses from the ambient gas. Of significance in those expressions are the transfer numbers defined in *Theory for spherically symmetric flow* in Equations (3.650) and (3.649), and these in turn are based upon conserved scalars defined in Equations (3.642) and (3.652). In this appendix, expressions for alternate transfer numbers and conserved scalars are provided for two additional systems: a simpler system in which evaporating or condensation occurs without any reaction in the boundary layer (ie. water droplet evaporation or condensation) and a more complicated system in which multiple oxidizers are involved in the oxidation of the evaporated fuel (relevant to metal oxidation).

Simple evaporation and condensation

When species evaporate or condense but do not otherwise react in the boundary layer surrounding the particle, the species mass fraction of the evaporating/condensing species itself is a conserved scalar. In this case, the mass transfer number corresponding to Eq. (3.649) is simply

$$B_{F-O} = \frac{Y_{F,f} - Y_{F,\infty}}{Y_{F,p} - Y_{F,f}} \quad (3.683)$$

and the heat transfer number corresponding to (3.650) is

$$B_{T-O} = \frac{\int_{T_f}^{T_\infty} c_{p,\infty} dT}{h_{vap} + \frac{Q_{rad}}{\dot{m}} + \left(\frac{m_p c_{v,p}}{\dot{m}} \frac{dT_p}{dt} \right)} \quad (3.684)$$

Multiple Oxidizers

For metal oxidation, it is possible to have multiple oxidizers that simultaneously react with the evaporating metal. These can be expressed with a series of parallel single-oxidizer reactions of the form $F + \nu_{O_i} O_i \rightarrow \sum \nu_{P_{i,j}} P_{j,i}$. Useful conserved scalars that can be formed with this set of reactions include

$$\beta_{F-O} = \frac{Y_F}{W_F} - \sum_i \frac{Y_{O_i}}{\nu_{O_i} W_{O_i}} \quad (3.685)$$

$$\beta_{T-O} = \int_{T_f}^T c_p dT + \sum_i \frac{W_F q_i}{\nu_{O_i} W_{O_i}} Y_{O_i} \quad (3.686)$$

where the enthalpy of reaction of the i^{th} oxidizer, O_i , with F is given by q_i . These are analogous to the conserved scalars defined in Equations (3.642) and (3.652). Using these conserved scalars,

a mass transfer number analogous to Eqn. (3.649) is found to be

$$B_{F-O} = \frac{Y_{F,p} - Y_{F,\infty} + \sum_i \frac{W_F q_i}{\nu_{O_i} W_{O_i}}}{Y_{F,p} - Y_{F,f}} \quad (3.687)$$

and the heat transfer number is

$$B_{T-O} = \frac{\int_{T_f}^{T_\infty} c_{p,\infty} dT + \sum_i \frac{Y_{O_i,\infty} W_F}{\nu_{O_i} W_{O_i}}}{h_{vap} + \frac{Q_{rad}}{\dot{m}} + \left(\frac{m_p c_{v,p}}{\dot{m}} \frac{dT_p}{dt} \right)} \quad (3.688)$$

These can be employed in Eqn. (3.666) and (3.667) to provide expressions for the particle mass burning rate as a function of the various oxidizer mass fractions far from the particle.

Energy Exchange between Particles and the Gas Phase Without Mass Transfer

In this section, a simpler scenario is considered where evaporation from and condensation onto particles is presumed to be negligible. For example, metal particles in dry air at ambient temperatures are unlikely to participate in evaporation or condensation. Models are presented in this section to treat these scenarios.

The models described in the previous sections are ill posed to solve these problems because the formulation is based on a balance between diffusion and the Stefan convective velocity, which is proportional to the negligible \dot{m} . The models in this section are used for the heated particles model.

In the event that there is no mass transfer, the particle heating rate is determined based on the balance between conductive and radiative transfer. The conductive heat transfer can be expressed using an effective heat transfer coefficient, thereby taking advantage of available Nusselt number correlations indicated in *Simple evaporation and condensation*. Then the relationship for the droplet heating is

$$m_p c_{v,p} \frac{dT_p}{dt} = 2\pi Nu_f r_p \lambda_f (T_g - T_f) + 4\pi \alpha r_p^2 (G_{in}/4 - \sigma T_f^4). \quad (3.689)$$

For the particle with no mass transfer, the closure of the film (or surface) temperature is obtained by equating the external heat flux, described by the right hand side of Eqn. (3.689), with the internal heat flux, as indicated on the right hand side of Eqn. (3.671) to obtain the following quartic constraint, which is solved for T_f

$$Nu_f \lambda_f (T_g - T_f) + 2\alpha r_p (G_{in}/4 - \sigma T_f^4) = Nu_p \lambda_p (T_f - T_p) \quad (3.690)$$

Further Notes on Radiative Heat Transfer

The expression employed for radiative droplet heating in Eqn. (3.645) is appropriate for relatively large and opaque particles, referred to as the geometric optics limit. For different particles, the expression for the absorptivity will change, but otherwise the expressions remain appropriate. To identify the appropriate absorptivity, it is useful to compare absorption coefficients for particle clouds. The absorption coefficient, κ , in units of inverse length, is an effective cross-sectional area per volume. For large particles with a condense phase absorptivity give by α , the absorption coefficient is the summation of the cross-sectional areas time the absorptivity

$$\kappa = \frac{1}{V_c} \sum_p \left[\pi r_p^2 \alpha \int_{V_c} f_\sigma(\mathbf{x}; \mathbf{x}_0, t) d\mathbf{x} \right]. \quad (3.691)$$

For comparison, in the small-particle or Rayleigh limit, the absorption coefficient is proportional to the volume fraction with the proportionality coming from the complex index of refraction. For intermediate particles where the particle optical depth is comparable to the particle radius, intermediate limits are appropriate, and the absorption can range from being proportional to the particle area to being proportional to the particle volume. The appropriate absorption coefficient, and from it the particle emissivity, cases can be determined as described in the available texts [120, 124].

Input Parameters for Particle Evolution

A large number of parameters are required to specify the evolution of the particles. The parameters provided in the input file are described in table *Input parameters related to the particle evolution provided through the input file*. In addition, table *Variables passed from the gas-phase Eulerian solver required to evolve the particles* specifies those variables that must be obtained from the gas-phase continuum flow. Because parcels of particles have finite extent and may span several control volumes, it is sometimes necessary to interpolate values of these gas-phase variables from several control volumes.

Table 3.12: Input parameters related to the particle evolution provided through the input file

Variable	Input name	Units	Description
ρ_p	INJP_DENp	kg/m^3	Particle density
T_p	INJP_Tp	K	Particle temperature
W_F	INJP_MWp	$kmol/kg$	Molecular weight of fuel or particle component
$c_{v,p}$	INJP_Clp	$J/kg/K$	Particle specific heat
α	INJP_Emp	none	Particle absorptivity, emissivity
Pr	INJP_Pr	none	Film Prandtl number
Sc	INJP_Sc	none	Film Schmidt number
P_{ref}	INJP_Prefvapp	Pa	Reference pressure for vaporization for particle
T_{ref}	INJP_Trefvapp	K	Reference temperature for vaporization for particle
$h_{vap,ref}$	INJP_Hvaprefp	J/kg	Reference enthalpy of vaporization for particle
T_{crit}	INJP_Tlcp	K	Critical temperature for particle
T_{fr}	INJP_Tlfp	K	Freezing temperature for particle
σ_p	INJP_STp	N/m	Particle surface tension
μ_p	INJP_mup	$kg/m/s$	Particle viscosity
Pr_p	INJP_Prl	none	Particle Prandtl number
q_{comb}		J/kg	Enthalpy of combustion for vapor species at T_f
ν_i		mol/ Fuel	Stoichiometric coefficients (relative to evaporating species)

Table 3.13: Variables passed from the gas-phase Eulerian solver required to evolve the particles

Variable	Units	Description
ρ_∞	kg/m^3	Gas-phase density
P	Pa	Pressure
u_g, v_g, w_g	m/s	Mean gas velocity
k	m^2/s^2	Turbulent kinetic energy
ϵ	m^2/s^3	Turbulent kinetic energy dissipation rate
T_∞	K	Gas temperature
W	kg/mol	Gas-phase mean molecular weight
$Y_{F,\infty}$	N/A	Mass fraction of fuel in gas phase
$Y_{O,\infty}$	N/A	Mass fraction of oxidizer in gas phase
$c_{p,\infty}$	$J/kg/K$	Gas-phase specific heat
μ_g	$kg/m/s$	Gas-phase viscosity
G_{in}	W/m^2	Incident radiation

Coupling the Lagrangian and Eulerian Fields

The previous section provides a description of the particle evolution given a gas-phase environment. In this section the means by which the gas-phase environment for a particle is determined from an Eulerian solution presumed to use a control-volume or similar approach. Following this, the effect of the particle field on the Eulerian field is described. Finally, to ensure that the coupled evolution proceeds in a physically realistic manner, it is necessary to identify limits on the time step size.

Gas-phase environment for parcels

When a parcel of particles spans more than a single control volume, it is appropriate to employ a weighted average of the gas properties in the control volumes spanned by the parcel. The average is weighted by the number of particles in a given control volume, which is obtained from $f_\sigma(\mathbf{x}; \mathbf{x}_0, t)$ defined in Eqn. (3.632). Thus, the average value of a gas-phase variable, ϕ , for a parcel is

$$\bar{\phi} = \frac{\int_V \phi(\mathbf{x}, t) f_\sigma(\mathbf{x}; \mathbf{x}_0, t) d\mathbf{x}}{\int_V f_\sigma(\mathbf{x}; \mathbf{x}_0, t) d\mathbf{x}} \quad (3.692)$$

where the integral volume may span more than one control volume. This procedure is employed for all gas phase variables that appear in the droplet evolution equations in *Particle Transport Model*. For transport properties (i.e. various diffusion coefficients), the gas phase properties employed are evaluated using the ‘1/3 rule’ as indicated for viscosity in Eqn. (3.624).

Effect of the particle phase on the gas phase

The source terms provided in the previous section show how the gas-phase properties affect the conservation of mass, momentum, and energy for the particles. From Newton’s third law, the action of the gas phase on the particles must be balanced by an action of the particles on the gas phase. This action is determined directly from the change in the state of the particle phase as described in the following.

The source of the mass for a given control volume, V_c , located at (\mathbf{x}, t) is determined by summing the changes in the masses of all the particles in that control volume over the gas-phase time step δt

$$S_{mass}(\mathbf{x}, t) = -\frac{1}{V_c} \sum_P \left[\frac{m_p(t + \delta t) - m_p(t)}{\delta t} \int_{V_c} f_\sigma(\mathbf{x}; \mathbf{x}_0, t) d\mathbf{x} \right]. \quad (3.693)$$

Here, the summation is presumed to occur over the P parcels that contribute to the control volume at (\mathbf{x}, t) , and the addend in the square brackets corresponds to the mass change and particle number distribution in each of the P parcels. The similar equation for species concentration incorporates the conversion of the mass of the particle component to the mass of the gas-phase

component. For pure evaporation, the conversion is trivial, but for combustion where the fuel evaporation corresponds to oxidizer consumption and the product formation, the expressions become complicated. The general expression for the species mass source term is

$$S_{mass,i}(\mathbf{x}, t) = -\frac{1}{V_c} \sum_P \left[\frac{\nu_i W_i}{W_F} \frac{m_p(t + \delta t) - m_p(t)}{\delta t} \int_{V_c} f_\sigma(\mathbf{x}; \mathbf{x}_o, t) d\mathbf{x} \right]. \quad (3.694)$$

Here, ν_i is the number of moles of species i produced when a mole of the particle component evaporates, and W_i is the molecular weight of species i . The particle component is presumed to be the fuel, denoted with the subscript F , and for pure evaporation reduces to equation (3.693).

Similarly, the source term for the j -momentum equations is determined by summing the changes in the particle momentum over all of the particles in a control volume.

$$S_{mom,j}(\mathbf{x}, t) = -\frac{1}{V_c} \sum_P \left[\left(\frac{m_p(t + \delta t) u_{p,j}(t + \delta t) - m_p(t) u_{p,j}(t)}{\delta t} - m_p(t) g_j \right) \int_{V_c} f_\sigma(\mathbf{x}; \mathbf{x}_o, t) d\mathbf{x} \right] \quad (3.695)$$

Note that the last term in Eqn. (3.695) describes the effect of gravitational acceleration on the particle field.

The energy source term must account for the enthalpy of vaporization and heat of combustion associated with the particles as well as the particle heating. It must also account for the change in the enthalpy of the gas associated with any gases that evaporated or condensed. In addition, it is necessary to separate out the contributions associated with radiative transport since these do not necessarily affect the local control volume, but are transported over length scales determined by the radiative transport equation. The enthalpy source term is

$$\begin{aligned} S_{enthalpy}(\mathbf{x}, t) = & -S_{rad} - \frac{1}{V_c} \sum_P \left[\frac{m_p(t + \delta t) [h_p(t + \delta t) - h_p(t)]}{\delta t} \int_{V_c} f_\sigma(\mathbf{x}; \mathbf{x}_o, t) d\mathbf{x} \right] \\ & - \frac{1}{V_c} \sum_P \left[\frac{[m_p(t + \delta t) - m_p(t)] \sum_i \nu_i W_i h_i(T_g)/W_F}{\delta t} \int_{V_c} f_\sigma(\mathbf{x}; \mathbf{x}_o, t) d\mathbf{x} \right] \\ & - \frac{1}{V_c} \sum_P \left[\frac{[m_p(t + \delta t) - m_p(t)] (q_{comb} - h_{vap})}{\delta t} \int_{V_c} f_\sigma(\mathbf{x}; \mathbf{x}_o, t) d\mathbf{x} \right] \end{aligned} \quad (3.696)$$

where the radiative transport source term is

$$S_{rad}(\mathbf{x}, t) = \frac{1}{V_c} \sum_P \left[Q_{rad} \int_{V_c} f_\sigma(\mathbf{x}; \mathbf{x}_o, t) d\mathbf{x} \right], \quad (3.697)$$

which provides the interface between the particle field and the radiation transport equation. In general, the solution of the radiant transport equation requires an absorption coefficient and a contribution to the emission. These must be defined in such a way that they agree with the radiation absorbed and emitted by the particle field. To do this, an energy balance for the particle field, but in an Eulerian frame, is employed. Within the energy conservation equation, the radiant

source term appears as a divergence of the radiation heat flux $\nabla \cdot q_R$. This divergence of the radiation heat flux can be related to the radiation intensity and the radiation emitted by [120]

$$\nabla \cdot q_R = \langle 4\kappa_{net}\sigma T_{net}^4 \rangle - G_{in}\langle \kappa_{net} \rangle \quad (3.698)$$

where the subscript *net* indicates that contributions from the particles must be combined with those from gases, smoke, and anything else that participates with the radiative field. The angular brackets indicate that the quantities on the right hand side must be defined based on the appropriate summation or averaging process over all of these participating media, which may have varying temperatures. In order to balance the radiant energy flux in and out of the particle with their contribution to Eqn. (3.698), it is necessary to separate these particle contributions from the net radiant source term. To leading order, this can be done in an additive manner so that the first term in Eqn. (3.698) is split as $\langle 4\kappa_{net}\sigma T_{net}^4 \rangle = \langle 4\kappa_p\sigma T_p^4 \rangle + \langle 4\kappa_{others}\sigma T_{others}^4 \rangle$ and the second term is split with $\langle \kappa_{net} \rangle = \langle \kappa_p \rangle + \langle \kappa_{others} \rangle$. Here the subscript *p* indicates the contribution associated with the particle field and the subscript *others* indicates all other contributions (gases, soot, etc.). Corrections to this leading order approximation are related to the optical thickness of the control volumes over which the radiation solve occurs. These corrections arise because a portion of the intensity is absorbed within the control volume. As long as the control volumes are all optically thin, then this correction is not important, but it must be accounted for in the radiation solve if that term is important.

Separating out only the contribution of the particle field to the radiation solve gives

$$(\nabla \cdot q_R)_p = \langle 4\kappa_p\sigma T_p^4 \rangle - G_{in}\langle \kappa_p \rangle \quad (3.699)$$

where the angular brackets indicate that the quantities on the right hand side must be defined based on the appropriate summation over all of the particles in the control volume. In order to ensure energy conservation between the Lagrangian solution of the particle field and the eulerian solution of the radiation transfer and energy equations, these appropriate sums are obtained by integrating over the particle field in a given control volume as in Eqn. (3.697). Here, the emission and absorption contributions of Q_{rad} are separated

$$S_{rad}(\mathbf{x}, t) = \frac{1}{V_c} \sum_P \left[4\pi r_p^2 \alpha \sigma T_p^4 \int_{V_c} f_\sigma(\mathbf{x}; \mathbf{x}_0, t) d\mathbf{x} \right] - \frac{1}{V_c} \sum_P \left[\pi r_p^2 \alpha \int_{V_c} f_\sigma(\mathbf{x}; \mathbf{x}_0, t) d\mathbf{x} \right] G_{in} \quad (3.700)$$

so that, equating the first and second terms on the right-hand side of Equations (3.699) and (3.700) gives

$$\langle 4\kappa_p\sigma T_p^4 \rangle = \frac{1}{V_c} \sum_P \left[4\pi r_p^2 \alpha \sigma T_p^4 \int_{V_c} f_\sigma(\mathbf{x}; \mathbf{x}_0, t) d\mathbf{x} \right] \quad (3.701)$$

and

$$\langle \kappa_p \rangle = \frac{1}{V_c} \sum_P \left[\pi r_p^2 \alpha \int_{V_c} f_\sigma(\mathbf{x}; \mathbf{x}_0, t) d\mathbf{x} \right] G_{in} \quad (3.702)$$

This information is crucial because it defines an energy conservation interface between the particle field and the radiation field.

The source terms defined in Equations (3.693) through *Effect of the particle phase on the gas phase* should be added directly to the gas-phase conservation equations in the manner appropriate for the chosen numerical method. In the above relations, it was presumed that the dimensions of the conservation equations solved are those of density, density time velocity, and density time enthalpy. The radiant flux source term in Eqn. (3.697) should similarly be added to the radiative transport equation, but in the consistent manner indicated in the above paragraph.

Time step control

The Lagrangian and Eulerian fields are advanced using an explicit operator splitting approach. The particles are presumed to be advanced using an ordinary differential equation solver capable of handling a stiff system. The system can be stiff because the magnitude of the forcing function, the right hand side, of the various particle evolution equations can vary over orders of magnitude. While the particle state is evolved, the gas-phase state is presumed to be constant, except as described in the following paragraph. The source terms indicated in the previous section are accumulated during the particle evolution, and then they are applied to the gas-phase conservation equations as it is advanced.

There are certain requirements that limit the particle evolution time step. Particle should not move more than the length of the control volume side without re-evaluating the gas-phase state using the methods described in *Gas-phase environment for parcels*. Particles should not evolve for longer than the eddy interaction time defined as the lesser of times defined in Eqn. (3.630) and (3.631) without determining a new value for $u_{g,i}$ in Eqn. (3.628) and (3.629). These requirements do not necessarily limit the gas-phase evolution time step.

There are also limitations to the gas-phase evolution time step imposed by the particle evolution. The source of this limitation is the requirement that the treatment of the gas-phase state as constant during the particle evolution not lead to nonphysical or inaccurate results. For example, if the particles transferred almost all of their momentum to the gas phase in a single time step (because they were small, for example), then a problem could arise. Specifically, if the subsequent momentum source term were large, then the gas could be accelerated to velocities that exceed the initial particle velocities. In the subsequent time step, the particles would accelerate and the solution procedure could thereby destabilize. There are two means of avoiding such problems. One approach is to employ an iterative implicit coupled advance of the Eulerian and Lagrangian state. Such an approach would depend on the specific algorithms employed for the Eulerian solver. The approach that will be discussed here is a limitation on the time step size for the coupled Eulerian-Lagrangian system. This limitation is based on the idea that the change in the Eulerian state relative to the Lagrangian state should not be too dramatic. This not only provides stability, but helps limit numerical errors.

For mass conservation, the limitation on the time step is based on the idea that the mass added to the cell should not dramatically affect the pressure (or whatever variable changes to allow additional mass to entry the control volume gas phase). With S_{mass} in units of density per unit time, the appropriate limitation on the time step is a time step that leads to no more than a change

of $\delta_\rho \rho$ in ρ .

$$dt_{max} < \delta_\rho \rho / S_{mass} \quad (3.703)$$

That is, the fractional change in the density is limited to δ_ρ . It is expected that δ_ρ is substantially smaller than 0.1 is appropriate, but this is subject to evaluation and will depend somewhat on the details of a given simulation. for the conservation of species, similar expressions apply, but with the second term that provides a sort of absolute tolerance in addition to the relative tolerance indicated above

$$dt_{max} < \delta_{Y_i} \rho Y_i / S_{mass,i} + \eta_{Y_i} \rho / S_{mass,i}. \quad (3.704)$$

Intuition suggests that the limitation of δ_{Y_i} do not need to be as severe as those on δ_ρ , for stability and that η_{Y_i} should be substantially smaller than typical magnitudes for Y_i for accuracy. Again, the specific values will depend on how sensitive the system is on Y_i . For momentum, a similar expression is provided with relative and absolute tolerances

$$dt_{max} < \delta_u \rho u_i / S_{mom,i} + \eta_u \rho / S_{mom,i} \quad (3.705)$$

where $\delta_u u_i$ and η_u are indicative of the acceptable uncertainties in the velocity field. Time-step control based on the enthalpy exchange is easiest to think of in terms of temperature

$$dt_{max} < \delta_h \rho c_{p,g} T_g / S_{enthalpy} + \eta_h \rho c_{p,g} / S_{enthalpy} \quad (3.706)$$

where $\delta_h T_g$ and η_h are indicative of the acceptable uncertainties in the temperature field footnote{ Instead of using $S_{enthalpy}$ that includes both the chemical and sensible enthalpy changes, it is better to use a measure of the change in the sensible enthalpy source term (see Eqn [Effect of the particle phase on the gas phase](#)) where the next to the last set of brackets includes only the sensible contribution of the species heating and not the chemical enthalpy contribution.

Particle-surface interactions

Particles interacting with a surface can (1) bounce off of the surface and return to the flow with a different trajectory, (2) stick to the surface and remain as a deposit or (3) shatter so that smaller droplets are formed that leave the surface with various trajectories. The appropriate behavior depends primarily on the ratio of the droplet kinetic energy to the surface energy as determined by the Weber number

$$We = \frac{\rho_p d_p |\mathbf{u}_{p,i}|^2}{\sigma} \quad (3.707)$$

where σ is the surface tension of the condensed phase. A complete model description is available in [125] with criteria for droplet sticking and bouncing. That paper did not address droplet shattering in detail and the droplet shattering model is provided in the following paragraph.

Droplets will shatter if the criteria

$$We^{0.5} Re_p^{0.25} > K_{crit} = 57.7 \quad (3.708)$$

is satisfied, which occurs for relatively large droplets traveling at relatively high velocities. Satellite droplets are presumed to form with uniform sizes given by

$$d_{p,shatter} = \max \left(\frac{7.9 \cdot 10^{10} \sigma W e^{1.4} / Re^{2.8}}{\rho_p |\mathbf{u}_{p,i}|^2}, d_p \right) \quad (3.709)$$

where the second term in the max function ensures that the empirical relationship given as the first term does not exceed the original diameter.

If a particle sticks to the surface, it is necessary to track the mass and energy addition to that surface through a field variable added to the solid object surface set. Mass addition should be tracked on a mass per solid-element surface area basis. Similarly, the energy deposited on a surface should be tracked based on $c_{v,p}(T_p - T_{surface})$. Erikson and Gill have developed and implemented a model to provide such an interface for a Calore surface [126].

Verification of Particle Evolution Equations

It is generally necessary to check the implementation of the equations described in *Particle Transport Model* to insure that they produce the expected effect. This process is referred to as verification. There are several stages of verification, many of which are focused on software design details, but many of which are intimately linked with the physics model implementation. In this section, a series of verification tests is presented that provides a test for the correct implementation of the models described in the previous sections. For the particle transport models developed here, this is accomplished by taking certain asymptotic limits of the evolution equations for which an analytical solution exists and insuring that the particle evolution approaches that solution as the particle properties approach the appropriate limiting values. For example, in the limit of zero Reynolds number, certain behavior is expected and the particle should approach that behavior as the diameter approaches zero. The limiting behavior is based on the limiting behavior of the model equations and is not based on matching particular experimental data, although many of the limiting behaviors correspond to well known phenomena. The key objective of the verification process here is to insure that the equations are satisfied, so that when numerical examples are given, nominal values for material and transport properties are employed.

Verification of Particle Momentum and Trajectories

In this section the solutions of Equations (3.621) through (3.627) are tested. For all of these tests, it is presumed that the source terms indicated in (3.693) through *Effect of the particle phase on the gas phase* are set to zero so that the gas velocities and other properties are fixed. Setting the gas-phase source terms to zero is referred to as one-way coupling because the gas phase affects the particular phase, but not vice versa.

Terminal Velocity

Falling particle will reach a terminal velocity if the gas velocity is held fixed that is given by

$$u_{p,i}|_{t \rightarrow \infty} = u_{g,i} + \frac{8(\rho_p - \rho_g)r_p g_i}{3\rho_g C_D |u_{g,i} - u_{p,i}|}. \quad (3.710)$$

Because this terminal velocity includes the Reynolds number dependence of the drag coefficient, the full range of the drag coefficient can be tested by changing, for example, the particle diameter. Note that the slip velocity magnitude does appear in the drag coefficient for a range of values through the Reynolds number. For these Reynolds numbers ($Re_p < 1000$), the drag coefficient can be replaced by the first relationship in Eqn. (3.622) to give

$$u_{p,i}|_{t \rightarrow \infty} = \frac{2(\rho_p - \rho_g)r_p^2 g_i}{9\mu_g(1 + Re_p^{2/3}/6)}. \quad (3.711)$$

For $Re_p > 1000$, using the second relationship from Eqn. (3.622) in Eqn. (3.710) yields simply

$$u_{p,i}|_{t \rightarrow \infty} = u_{g,i} + \sqrt{\frac{8(\rho_p - \rho_g)r_p g_i}{3\rho_g(0.424)}}. \quad (3.712)$$

The full range of terminal velocities obtained by varying r_p is shown in Fig *The Fuego-computed particle velocity and trajectory are compared with predictions from eq-Theory-Particles-smallReynoldsGasVel and eq-Theory-Particles-smallReynoldsPos (left) and error (right). Parameters from Table %s with $d_p = 3 \times 10^{-2}$* . Note that while the velocities are continuous at $Re_p = 1000$, there is a discontinuity in the rate of change of the velocity with diameter of Reynolds number at this point.

Small Reynolds number

In the limit of small Reynolds number the solution of Eqn. (3.621) approaches that given by Eqn. (3.625) and (3.626) where C_D simplifies to $24/Re_p$ so that τ_p approaches a constant value even as the velocity changes. In this case, the analytic solution of Eqn. (3.621), where the gas velocity is fixed, is

$$u_{p,i}|_{t \rightarrow \infty} = \left(u_{g,i}^0 + \frac{(\rho_p - \rho_g)}{\rho_p} \right) \left(1 - e^{-t/\tau_p} \right) + u_{p,i}^0 e^{-t/\tau_p} \quad (3.713)$$

where the superscript 0 indicates the initial conditions. This limit can be tested by approaching zero Reynolds numbers with successively smaller diameters. The solution to Eqn. (3.627) is obtained is similarly obtainable by integration of (3.713) once and is

$$x_{p,i} = x_{p,i}^0 + \left(u_{g,i}^0 + \frac{(\rho_p - \rho_g)}{\rho_p} g_i \tau_p \right) \left(1 - e^{-t/\tau_p} \right) + u_{p,i}^0 \tau_p \left(1 - e^{-t/\tau_p} \right) \quad (3.714)$$

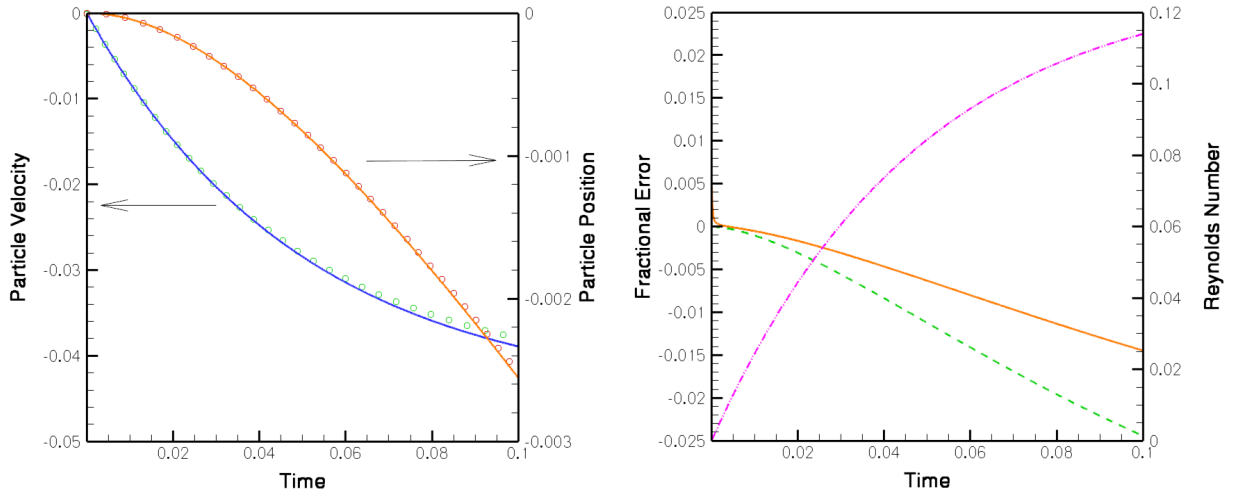


Fig. 3.8: The Fuego-computed particle velocity and trajectory are compared with predictions from (3.713) and (3.714) (left) and error (right). Parameters from Table 3.14 with $d_p = 3 \cdot 10^{-2}$

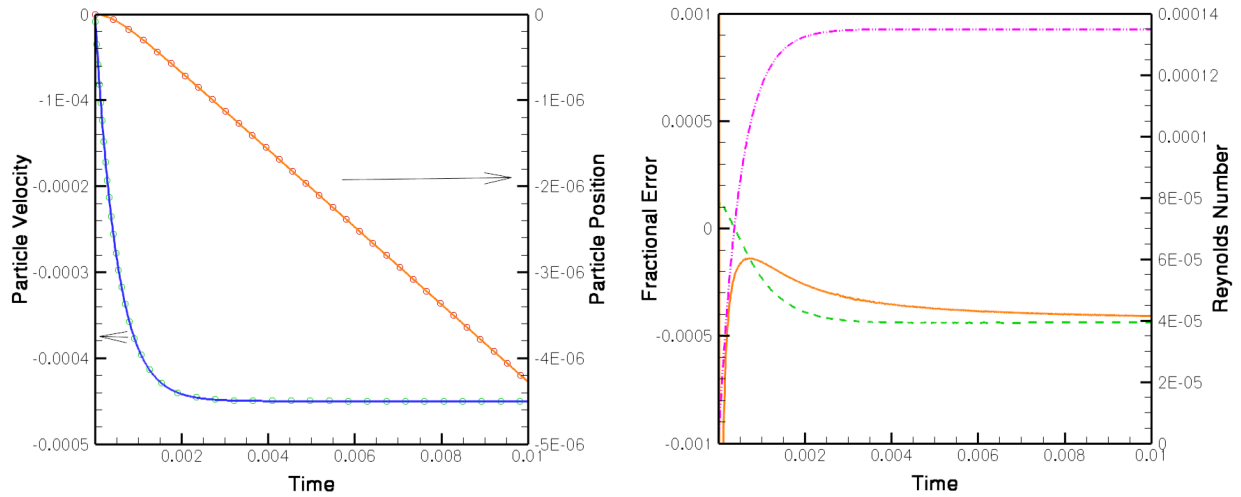


Fig. 3.9: The Fuego-computed particle velocity and trajectory are compared with predictions from (3.713) and (3.714) (left) and error (right). Parameters from Table 3.14 with $d_p = 3 \cdot 10^{-3}$

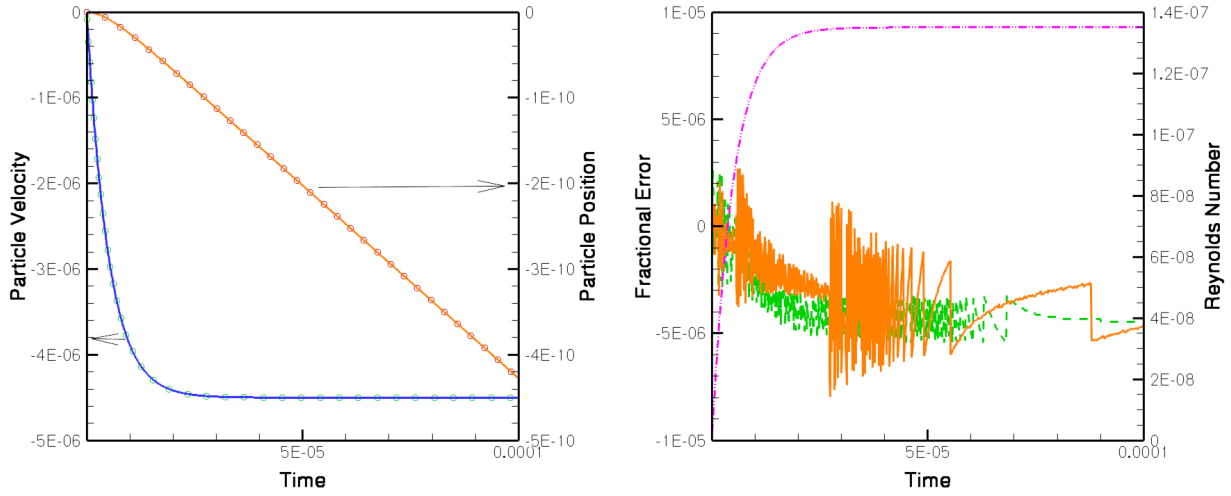


Fig. 3.10: The Fuego-computed particle velocity and trajectory are compared with predictions from (3.713) and (3.714) (left) and error (right). Parameters from Table 3.14 with $d_p = 3 \cdot 10^{-3}$

Using the values indicates in table *Input parameters related to the particle trajectory verification*, and with no heat and mass transfer, particles were evolved from rest using Fuego. The resulting particle trajectories are compared with those predicted in Equations (3.713) and (3.714), and the velocities and positions are plotted in Figure 3.8.

Table 3.14: Input parameters related to the particle trajectory verification

d_p	ρ_p	μ_g	ρ_g	g_i	τ_p	$u_{p,i} _{t \rightarrow \infty}$	$Re_p _{t \rightarrow \infty}$
$3 \cdot 10^{-2}$	10	0.01	1.0	-1.0	$5 \cdot 10^{-2}$	$5 \cdot 10^{-2}$	$1.5 \cdot 10^{-1}$
$3 \cdot 10^{-3}$	10	0.01	1.0	-1.0	$5 \cdot 10^{-4}$	$5 \cdot 10^{-4}$	$1.5 \cdot 10^{-4}$
$3 \cdot 10^{-4}$	10	0.01	1.0	-1.0	$5 \cdot 10^{-6}$	$5 \cdot 10^{-6}$	$1.5 \cdot 10^{-7}$

Turbulent dispersion

Verification of particle dispersion is hampered by the fact that it is a stochastic and not deterministic process. Therefore, verification is conducted by checking that the asymptotic behavior is correct for large numbers of samples. In an isotropic homogeneous turbulent field, the turbulent dispersion of a large number of particles should result in the mean square displacement of particles in proportion to the effective diffusion coefficient and the evolution time. The diffusion coefficient in the limit where τ_e and not τ_C determines the eddy interaction time is proportional to $|\mathbf{u}_g + \mathbf{u}'_g - \mathbf{u}_p|^2 \tau_p$ [127]. For verification purposes, the mean gas velocity is set to zero so that the effective diffusion coefficient is proportional to k . Note that the early time behavior is different, but that the early time behavior decays over the time scale τ_p . The long-time dispersion behavior

averaged over a sufficient statistical sample should be verified to follow [101]

$$\langle x^2 \rangle \propto k \tau_p t \quad (3.715)$$

Such behavior should hold true in a statistical sense for both the dispersion of the mean parcel locations and for the change in the extent of the parcel itself. Because the dispersion is driven by a Gaussian process, it can be expected that statistical differences are reduced in proportion to the inverse of the square root of the number of samples.

To verify the scaling of the particle dispersion, a large number of particles were evolved using specified fluid variables. All of the mean fluid velocities are set to zero and only the turbulent kinetic energy, k , combined with time and length-scale information from the turbulent energy dissipation, ϵ , acted on the particles. In this case, the only force on the particle is associated with $u'_{g,i}$ as defined in Equations (3.628) and (3.629). The parameters related to particle dispersion used in the test are given in *Input parameters related to the particle dispersion verification*. The flow was evolved for a time of 5.0 (nearly $1000\tau_p$) and simulations were carried out for 100, 1000, and 10,000 particles. The results are shown in *The mean square displacement of particles is shown as a function of time. Different curves show the statistical noise associated with different numbers of particles considered*, where the linear dependence of the mean-square displacement with time is evidence for large particle samples. The large number of samples required to get convergence is noteworthy.

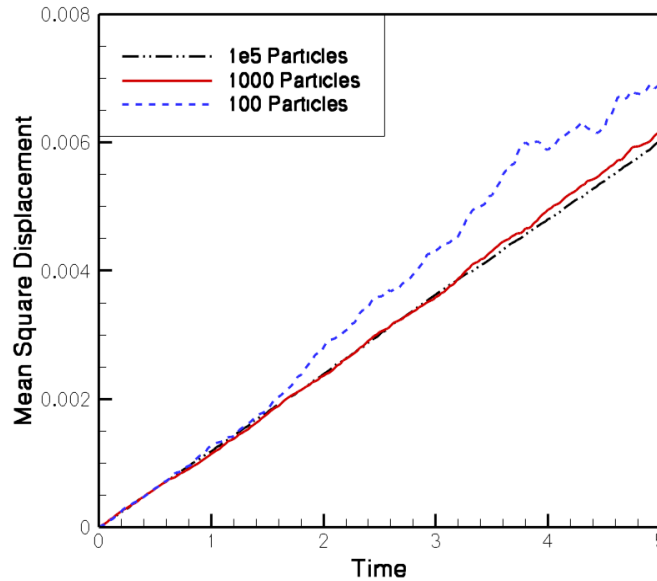


Fig. 3.11: The mean square displacement of particles is shown as a function of time. Different curves show the statistical noise associated with different numbers of particles considered.

Table 3.15: Input parameters related to the particle dispersion verification

Fluid properties		Particle properties	
k	1.0	ρ_p	10.0
ϵ	1.0	d_p	$1 \cdot 10^{-4}$
μ_g	0.01	τ_p	$1 \cdot 10^3/18$

Verification of particle heat and mass transfer

There are a number of terms describing the heat and mass transfer to and from particles that must be tested. In this section, the terms that describe the particle evolution in a constant gas-phase environment will be tested. To accomplish this, the source terms indicated in Eqn. (3.693) through (3.697) are set to zero so that the gas velocities and other properties are fixed (one-way coupling).

Droplet heating and cooling

If the particles have sufficiently high boiling points and enthalpies of vaporization, the evaporation rate in Eqn. (3.666) will be zero which implies $B_{T-O} = 0$. In this case, the droplet heating and cooling are governed by the equations presented in *Conserved scalars and transfer numbers for various applications*. There are several limits that can be tested. The most common situation that is encountered is conduction/convection dominated heating or cooling, and this can be split into high and low Reynolds number regimes. When radiant heating and cooling are the most significant processes, the system evolves according to the radiative heat flux terms resulting in different behavior.

To verify conduction dominated heating or cooling, the particle emissivities are set to zero, eliminating the radiative interaction terms. This leads to a simplification of Eqn. (3.689)

$$\frac{dT_p}{dt} = \left(\frac{6}{\rho_p c_{v,p} d_p^2} \right) \left(\frac{Nu_f \lambda_f Nu_p \lambda_p}{Nu_f \lambda_f + Nu_p \lambda_p} \right) (T_g - T_p) \quad (3.716)$$

for which an analytic solution

$$\frac{T_g - T_p}{T_g - T_{p,0}} = \exp \left[- \left(\frac{6}{\rho_p c_{v,p} d_p^2} \right) \left(\frac{Nu_f \lambda_f Nu_p \lambda_p}{Nu_f \lambda_f + Nu_p \lambda_p} \right) t \right] \quad (3.717)$$

is obtained if T_g , Nu_f , and Nu_p are constant and $T_{p,0}$ is the initial particle temperature. To verify radiant heat flux dominated heating or cooling, the gas-phase diffusion coefficient, λ_f , can be set to zero and non-evaporating particles can be initialized with appropriately high temperatures for cooling or with an appropriate incident flux. Neglecting gas-phase conduction, Eqn. (3.689) is written

$$\frac{dT_p}{dt} = \frac{3\alpha}{\rho_p c_{v,p} r_p} (G_{in}/4 - \sigma T_p^4) \quad (3.718)$$

where the film temperature is equal to the particle temperature for $\lambda_f = 0$. In this case, the analytic solution of Eqn. (3.718) is given in implicit form

$$\frac{1}{2T_\infty^3} \left[\tan^{-1} \left(\frac{T_p}{T_\infty} \right) - \tan^{-1} \left(\frac{T_{p,0}}{T_\infty} \right) \right] - \frac{1}{4T_\infty^3} \left[\ln \left[\left(\frac{T_p T_\infty}{T_{p,0} - T_\infty} \right) \left(\frac{T_{p,0} + T_\infty}{T_p + T_\infty} \right) \right] \right] = \frac{3\alpha\sigma t}{\rho c_{v,p} r_f}. \quad (3.719)$$

where $T_\infty = (G_{in}/4\sigma)^{1/4}$ is the temperature seen by the particle.

Table 3.16: Input parameters related to the particle heating and cooling verification

Fluid properties		Particle properties	
k	1.0	ρ_p	10.0
ϵ	1.0	d_p	$1 \cdot 10^{-4}$
μ_g	0.01	τ_p	$1 \cdot 10^3/18$

The wet bulb temperature

The wet bulb state is the state at which droplet heating is zero that occurs when the heat transfer to the droplet is perfectly balanced by the enthalpy of vaporization and no droplet heating occurs. The droplet temperature will tend to approach the wet-bulb temperature, either by being heated by excess enthalpy conducted through the surface or by being cooled when more enthalpy is used in vaporization than is conducted to the surface. The system of Equations (3.689), (3.666), and (3.669) with $(m_p c_{v,p}/\dot{m})dT_p/dt = 0$ in Eqn. (3.651) determines the wet bulb state. Droplets with initial temperatures set to the wet bulb temperature should evolve with no change in the droplet temperature. It is verified in the following sections that particles initially at the wet bulb temperature do not change temperature, and that droplets approach the wet bulb temperature from other initial droplet temperatures.

d^2 -Law Evaporation and Condensation

For particles at the wet bulb temperature with no radiative losses there will be no particle heating so that B_{T-O} and B_{F-O} are independent of the radius. Then, if the slip velocity is negligible the particle will evaporate at a rate proportional to the diameter squared, following the well-known d^2 -law for particle evaporation

$$\frac{d(d_p^2)}{dt} = -K. \quad (3.720)$$

To test this behavior, droplet heating should be prevented by setting the droplet temperature to the wet bulb temperature, the radiative heat transfer should be inhibited by setting the emissivity to zero, and the particle Reynolds number should approach zero in the sense that errors on the order of $Re^{1/2}$ are introduced by finite slip velocity correlations. In this case, plotting the droplet diameter squared versus time should yield a linear line with a slope given by

$$K = \frac{4\dot{m}}{\pi\rho_p d_p}. \quad (3.721)$$

To verify the d^2 -law for particle evaporation along with the wet-bulb temperature, three water droplets are evolved in an atmosphere of humid air. The wet-bulb temperature is computed separately for those conditions to be 313.9927, and particles are selected at that temperature and 1K above and below that temperature. The basic fluid and particle properties employed in the simulation are provided in *Input parameters related to the verification of droplet evaporation.* and the relationship between the initial particle diameter and temperature are indicated in *Particle initial conditions for verification of droplet evaporation.*. There is no flow and gravitational acceleration is not present so that there is no slip velocity maintaining the zero-Reynolds-number limit. For the conditions given, $K = 9.102 \cdot 10^{-5}$. Using these values the evaporation times for the three particles are 89.0, 109.9, 132.9s for particles labeled 1, 2, and, 3 in *Particle initial conditions for verification of droplet evaporation.*, respectively. The evaporation times predicted in Fuego are 89.1, 109.9, 123.8s in agreement with the predictions. The predictions assume that the deviation from the wet bulb temperature are insignificant. The evolution of d^2 for the particles indicated in *Particle initial conditions for verification of droplet evaporation.* is shown in d^2 (left) and T_p (right) as a function of time for evaporating water droplets; the trajectories are close to linear as expected and a linear curve fit for each d^2 profile gives an R^2 coefficient of unit indicating a high degree of correlation. Also shown in d^2 (left) and T_p (right) as a function of time for evaporating water droplets is the early evolution of the particle temperature. The temperature is shown to converge to the computed wet bulb temperature and to maintain itself at that temperature.

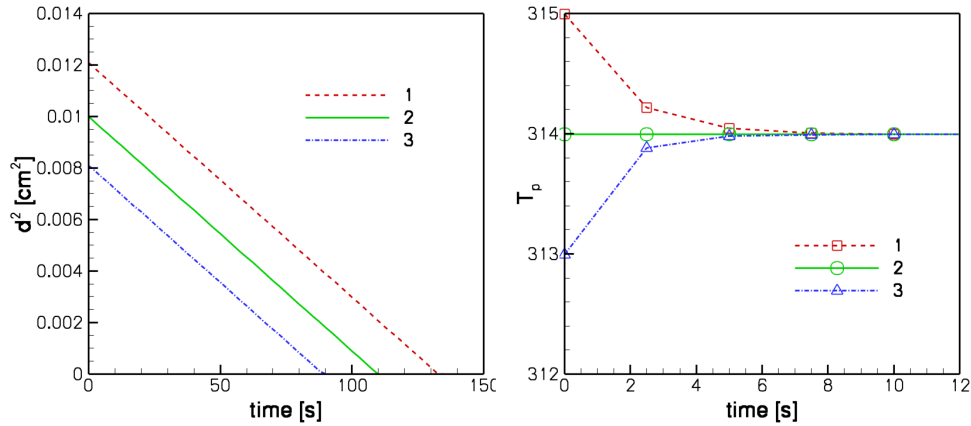


Fig. 3.12: d^2 (left) and T_p (right) as a function of time for evaporating water droplets

Table 3.17: Input parameters related to the verification of droplet evaporation.

Fluid properties		Particle properties		Evaporation properties	
$T_{g,in}$	400.0K	ρ_p	0.791g/cm ³	$h_{vap,ref}$	26.694 · 10 ⁻⁹
$u_{g,in}$	5.0g/cm ³	$u_{p,in}$	5.0cm/s	$T_{vap,ref}$	373.0K
Y_{H_2O}	0.01	$c_{v,p}$	4.184 · 10 ⁷ erg/g/K	$P_{vap,ref}$	1.0Atm
Y_{O_2}	0.23	Y_{H_2O}	1.0	T_{crit}	647.0K
Y_{N_2}	0.76	Pr_f	0.9		
Nu_f	2.0	Sc_f	0.9		

Table 3.18: Particle initial conditions for verification of droplet evaporation.

	1	2	3
$d_p [cm]$	0.11	0.1	0.09
$T_{p,in} [K]$	314.9927	313.9927	312.9927

$d^{1.5}$ -law for Fast Moving Droplets

For particle droplets with large Re_p , the available Nusselt and Sherwood number correlations indicate that the evaporation rate should increase with $Re_p^{1/2}$ for $Re_p \gg 10$. In these cases, setting the particles to the wet-bulb temperature with no radiative losses should lead to

$$\frac{d(d_p^{1.5})}{dt} = -K_{1.5}t \quad (3.722)$$

with

$$\lim_{Re_p \rightarrow \infty} K_{1.5} = \frac{\dot{m}_o Re_p^{1/2} Sc^{1/3}}{\pi \rho_p d_p^{3/2}} \quad (3.723)$$

being the constant rate of evaporation. Note that as the particle diameter approaches zero its Reynolds number must also approach zero so that the evaporation rate should transition to a d^2 -law behavior as it nears the fully evaporated state.

Verification of Lagrangian-Eulerian Coupling

The primary objective of this section is to verify that the source terms seen by the particle are appropriately reflected by equivalent (to the degrees appropriate) source terms in the gas-phase conservation equations. This primarily tests the source terms indicated in Equations (3.693) through (3.697). Because the particles affect the gas phase in the same way that the gas phase affects the particles, this is referred to as two-way coupling. The verification problems are formulated as one-dimensional problems (though the one dimension need not be aligned with the x, y, or z axis) to the maximum extent possible by employing symmetry boundary conditions on the four sides normal to the flow direction and imposing an initial uniform flow in the remaining direction. The general configuration employed is a cylindrical channel as indicated in *The general configuration for verification of two-way coupling. The domain should be of sufficient length that the particles equilibrate with the gas-phase flow.* This avoids any wall effects and provides a means of identifying the transfer from inlet conditions to outlet conditions. Further, in all of these tests gravitational acceleration should be set to zero and the effects of radiation negated by setting the particle emissivity to zero.

Mass conservation

The net mass flux through the system should be constant at steady state. Particles subject to evaporation will be iso-kinetically injected into the flow near the inlet (just downstream to ensure no effect of evaporation on the inlet boundary condition). These particles will be allowed to completely evaporate while they flow with the gas-phase through the domain, and the flow will be allowed to come to steady state as can be indicated by the constant exit mass flux. At steady-state, the inlet and outlet mass fluxes should be related according to the integral conservation relation

$$\int_{inlet} \rho_{in} u_{g,in} dA + \dot{M}_p = \int_{outlet} \rho_{out} u_{g,out} dA \quad (3.724)$$

where \dot{M} is the mass rate of particle injection. The subscripts *in* and *out* indicate the state of the gas-phase fluid at the inlet to the first control volume and the exit from the last control volume. The integral across the inlet and outlet areas will average out any spatial fluctuations. To facilitate reaching steady state flow with a reasonable number of particles, it is recommended that particles with a uniform size be employed. Since particle evaporation rates typically follow the d^2 -law behavior, it is preferable to have a relatively large number of particles injected over the evaporative lifetime of a given particle. The error in Eqn. (3.724) is expected to decrease as the frequency of particle injections is increased.

Species conservation

To test the species conservation, the same verification test is employed, but the mass of the individual species is tracked.

$$\int_{inlet} \rho_{in} Y_{i,in} u_{g,in} dA + \frac{\nu_i W_i Y_{F,P}}{W_F} \dot{M}_p = \int_{outlet} \rho_{out} Y_{i,out} u_{g,out} dA \quad (3.725)$$

The mass source term is written to also account for droplet combustion as relevant. Because the total mass associated with the system changes, the mass fraction of species that do not evaporate from the particle or participate in combustion would tend to decrease; that is, the total mass flux increases while the mass flux of non-evaporating species does not increase. This can act as a second species verification test.

Energy conservation

The configuration for energy conservation is the same as for mass and species conservation: one-dimensional flow with regular iso-kinetic particle injection and allowed to reach steady state. to test the coupled energy conservation, three tests are recommended to cover the range of particle behaviors. These tests would cover non-evaporating particles, evaporating but not combusting particles, and combusting particles.

The equation describing energy conservation for non-evaporating particles, assuming that the temperature of the particles equilibrates with the gas-phase before the outlet plane, is best

written

$$\int_{inlet} \rho_{in} u_{g,in} h_g(T_{g,in}) dA + \dot{M}_p c_{v,p} T_{p,in} = \int_{outlet} \psi_{out} \rho_{out} u_{g,out} h_g(T_{g,in}) dA + \dot{M}_p c_{v,p} T_{eq} \quad (3.726)$$

where h_g represents the gas-phase mixture enthalpy. The equilibrium temperature on the right-hand side of Eqn. (3.726) is obtained through an iterative solution of that nonlinear equation with the initial particle and gas temperatures prescribed as $T_{p,in}$ and $T_{g,in}$, respectively. Here the outflow gas-phase void fraction, ψ_{out} , appears in the first term on the right hand side; it is implicitly included in the other equations in this section, but is unit there based on the state assumptions. Satisfaction of the equality in Eqn. (3.726) provides a test of the first term on the right-hand side of Eqn. *Effect of the particle phase on the gas phase*. Note that the approach to the equilibrium temperature follows an exponential decay of the form \exp^{-t/τ_T} so that extending the domain to double the residence time from t to $2t$ will tend to cause particle and gas temperatures at the outlet to be accordingly closer to a factor of \exp^{-t/τ_T} . The thermal relaxation time constant for the non-evaporating particle is

$$\tau_T = \left(\frac{\rho_p c_{v,p} d_p^2}{6} \right) \left(\frac{Nu_f \lambda_f + Nu_p \lambda_p}{Nu_f \lambda_f Nu_p \lambda_p} \right). \quad (3.727)$$

This provides a additional check on the transient behavior of the system and can be used to evaluate time step control provided by Eqn. (3.706). Regardless of the time step, the enthalpy transfers should be conservative at equilibrium.

for the evaporating or combusting particles, the verification test is set up so that the evaporation of the particles is completed within the domain and that the flow reaches steady state prior to evaluation. In this sense it is the same arrangement as indicated above for mass and species conservation. These tests verify the last two terms in Eqn. *Effect of the particle phase on the gas phase*. With the particles entirely evaporated, an energy balance gives

$$\int_{inlet} \rho_{in} h(T_{g,in}) u_{g,in} dA + \dot{M}_p [h_p(T_{p,in} + q_{comb})] = \int_{outlet} \rho_{out} h(T_{g,out}) u_{g,out} dA \quad (3.728)$$

The initial particle enthalpy is defined

$$h_p(T_{p,in}) = h_F(T_{p,in}) - h_{vap,ref} \quad (3.729)$$

based on the gas-phase enthalpy of the particle species denoted by the subscript F . Again, a nonlinear solution of this equation is required to determine $T_{g,out}$ because of the nonlinear dependence of the enthalpy on temperature. To satisfy this equation, it is necessary that the specific heat for the condense and gaseous phase of the participating species be identical; otherwise particle cooling during evaporation followed by the warming of products to the equilibrium temperature will result in energy differences that would have to be accounted for by tracking each droplet temperature in time through the domain. Further, the critical temperature, T_{crit} in Eqn. (3.644), should be set essentially to an essentially infinite value of force $h_{vap} = h_{vap,ref}$.

Momentum conservation

As particles with excess momentum transfer their momentum to the gas-phase flow, the net flow rate will increase. The momentum verification test described in the present section differs from the other tests in this section in the sense that a single particle or group of particles is injected at an instant and there is no continuous injection. The injected particle(s) is(are) allowed to equilibrate with the gas-phase flow and the net change in momentum is measured. For this purpose, there must be no net pressure change across the domain boundaries. Integral momentum conservation then gives the final equilibrium velocity based on

$$\int_V \rho_g u_{g,in} dV + \sum_p m_p u_{p,in} = \int_V \rho_g u_{eq} dV + \sum_p m_p u_{eq}. \quad (3.730)$$

Presuming that the velocity equilibrates across the domain, u_{eq} can be brought outside of the integral and summation for an explicit expression. As for the temperature equilibration test in Eqn. (3.726), there is an exponential approach to the equilibrium value that can be used to test transient behavior if spatial fluctuations in the gas-phase velocity are not too strong. In that case, the equilibrium velocity is approached with the exponential time constant given by Eqn. (3.625) and (3.626) so that doubling the duration of the test from t to $2t$ should bring the particle and gas velocities closer by a factor of e^{-t/τ_r} . Note that this transient behavior can be used to evaluate the time step criteria provided in Eqn. (3.706) by assessing the error associated with liberal time steps. Regardless of the time step, though, the momentum transfer should be conservative at equilibrium.

Parallel implementation

All of these tests described in the present section are implemented on both one and four processors with domains that cross multiple processors to test the passing of particle information across domain boundaries.

Verification Tests for Lagrangian-Eulerian Coupling

In this section, the verification tests suggested in the previous sections are described. Verification tests for non-reacting iso-kinetic particle flows (energy conservation only), for isothermal and non-iso-kinetic particle flows (momentum conservation only) and for reacting/evaporating particle flows (mass and energy conservation) are all included. Together these test all of Eqn. (3.693) through *Effect of the particle phase on the gas phase*. Not yet covered with documented verification problems is the radiation coupling in (3.697).

Energy conservation verification

Energy conservation for non-evaporating particles was verified in Fuego using nominal parameter values for both the particles and the fluid. For the Eulerian phase, the mass, momentum, and energy equations were evolved, but not the species equations. Under these conditions, fluid properties are manually specified. The fluid properties listed in table *Input parameters related to the verification of energy conservation without evaporation* were specified as constants. Since the fluid viscosity, specific heat, and Prandtl numbers are specified, the thermal conductivity is computed from these quantities. The constant specific heat leads to a linear dependence of enthalpy on temperature, and without loss of generality the enthalpy is set to the temperature $h_g(T_g) = T_g$, corresponding to an enthalpy reference temperature of 0K. The simulation was carried out in the `cylindricalchannel.g` configuration with an inlet/outlet area of 3.10583 (with unit radius, this is nominally π , but low resolution of the circular cross section leads to a smaller area) and a length of 20. Particles are inject iso-kinetically at the downstream location 1 unit from the inlet and flow 19 units to the outlet in 19 time units. With iso-kinetic flow, $Nu_f = 2$ and with zero particle viscosity $Nu_p = 6.58$ from Eqn. (3.674). The particle thermal response time from Eqn. (3.727) is $\tau_T = 3.4$ and is sufficiently small that particle thermally equilibrate while traveling the length of the channel.

There are two ways to carry out the energy conservation verification: using Eqn. (3.726) during a period in which the flow has reached steady state and integrating Eqn. (3.726) in time over the entire simulation. Both of these approaches are employed here. The surface integrals specified in the conservation equation were carried out within Ensign. To compute the flux of the particles and associated enthalpy out of the domain (last terms in (3.726)), the particle deposition tracking in Fuego was employed (keywords: `enthalpy_deposition_density`, `enthalpy_deposition_rate`, etc.) and these quantities were also integrated over the outlet surface. The balance of the steady-state flux is described first. for the two terms on the left hand side of (3.726), the boundary conditions provided the values of 931.749 for the fluid inlet enthalpy flux and 10,000 for the particle inlet enthalpy. The compute equilibrium temperature is 834.1134K from (3.726). The outlet gas temperature was in the range of 883.4 to 834.8 with a mean of 834.1 and the exiting particles are between 832 and 835 K. to supplement the stead-state enthalpy flux, the integrated enthalpy flux is computed and plotted in *The enthalpy over time in nonreacting channel flow with hot particles injected. The net input enthalpy includes the initial domain enthalpy and the enthalpy of all of the injected particles over time. Squares show the enthalpy associated with particles in the domain. Circles show the enthalpy of the particles that have left the domain. Diamonds show the enthalpy associated with fluid in the domain. Triangles show the excess enthalpy of fluid that has left the domain (difference between the outlet enthalpy and inlet that was accounted for in the net input category). The sum of the categories indicated by symbols is shown to agree with the net input enthalpy indicating overall conservation of enthalpy..* The net input enthalpy includes the initial domain enthalpy and the enthalpy of all of the injected particles over time can be compared with the enthalpy in the domain and that which has left the domain. These two quantities agree to within 0.01%, which is taken to be suitable (the integration of quantities within Ensign does not use the same algorithms as employed in Fuego, leading to some error). Also examined in this test is the particle mass deposition rate at the outlet. Because there is no evaporation in this scenario, the complete particle mass injected should be deposited (or pass through) the outlet. Using the

keywords `mass_deposition_density` and `mass_deposition_rate` and integrating these over the outlet surface, *The mass deposition rate integrated over the outlet is shown as a function of time. The corresponding particle inlet mass flux is 0.1.* shows that the mass tracked as crossing the outlet plane matches the input particle mass, 0.1, to within statistical fluctuations.

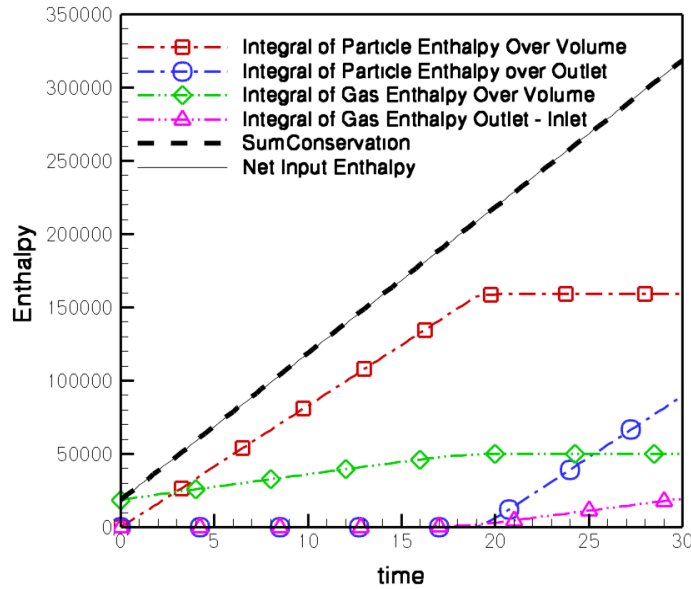


Fig. 3.13: The enthalpy over time in nonreacting channel flow with hot particles injected. The net input enthalpy includes the initial domain enthalpy and the enthalpy of all of the injected particles over time. Squares show the enthalpy associated with particles in the domain. Circles show the enthalpy of the particles that have left the domain. Diamonds show the enthalpy associated with fluid in the domain. Triangles show the excess enthalpy of fluid that has left the domain (difference between the outlet enthalpy and inlet that was accounted for in the net input category). The sum of the categories indicated by symbols is shown to agree with the net input enthalpy indicating overall conservation of enthalpy.

Table 3.19: Input parameters related to the verification of energy conservation without evaporation

Fluid properties		Particle properties	
ρ_{in}	1.0	ρ_p	10.0
$u_{g,in}$	1.0	$u_{p,in}$	1.0
μ_g	0.01	\dot{M}_p	0.1
$c_{p,g}$	1.0	$c_{v,p}$	100.0
$T_{g,in}$	300.0	$T_{p,in}$	1000.0
$h_g(T_{g,in})$	300	λ_p	0.1
Pr	1.0	Pr_f	1.0
Nu_f	2.0	Nu_p	6.58
$\int dA$	3.10583	d_p	0.01

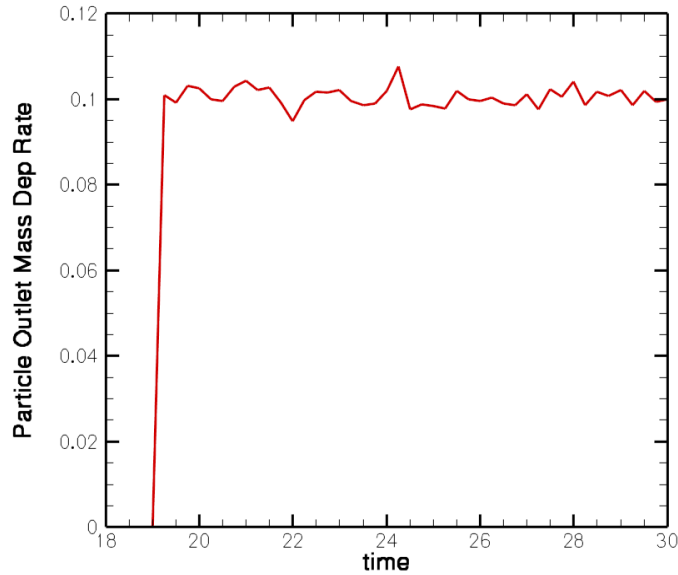


Fig. 3.14: The mass deposition rate integrated over the outlet is shown as a function of time. The corresponding particle inlet mass flux is 0.1.

Momentum conservation verification

To verify the momentum transfer within Fuego, simulations were carried out with ten particles injected into a cylindrical channel. Momentum conservation for non-evaporating particles was verified using nominal parameter values for both the particle and the fluid. For the Eulerian phase, the mass and momentum were evolved, but not the species and energy equations. Under these conditions, fluid properties are manually specified. The fluid and particle properties listed in table *Input parameters related to the verification of energy conservation without evaporation* were specified as constants. The simulation was carried out in the cylindrical_channel.g configuration with an inlet/outlet area of 3.10583 (with unit radius, this is nominally π , but low resolution of the circular cross-section leads to a smaller area), and a length of 20. The channel had symmetry boundary conditions on the side to prevent drag from affecting the momentum field. Particles are injected with an initial velocity of 10 units into a stationary fluid phase at the initial time. The particles were injected at the time origin with a diameter of 0.12, a density of 10, and a velocity of 10. The injection occurred as a line of ten particles across the channel (in the narrow direction). The particle flow direction was oriented lengthwise down the channel and the point of injection was the midpoint of the channel, 10 units from the inlet and outlet. Zero pressure boundary conditions were specified on the inlet and outlet that allowed the flow to continue in the absence of any forces (ie. particle drag).

Based on the values in *Input parameters related to the verification of energy conservation without evaporation*, the initial momentum imparted by the particles is 0.904779 and the computed total momentum at the end of the Fuego simulation is 0.893059. That is, the final combined momentum is 1.3 % less than the initial momentum. The reason for this discrepancy is unclear.

The final particle and fluid momentum are 0.001299 and 0.89176, respectively. The temporal evolution of the fluid, particle, and total momentum is shown in *The momentum associated with the particle phase, the fluid phase, and the combined momentum shown. The right-hand panel shows the initial period in greater detail.* There the fluid momentum has been shifted back one time step (0.01 units) to account for the staggered time-stepping algorithm: the fluid evolves based on the previous time steps' particle momentum transfer. The simulation was evolved for 30 time units, but no changes for the single precision arithmetic was observed after 21 units.

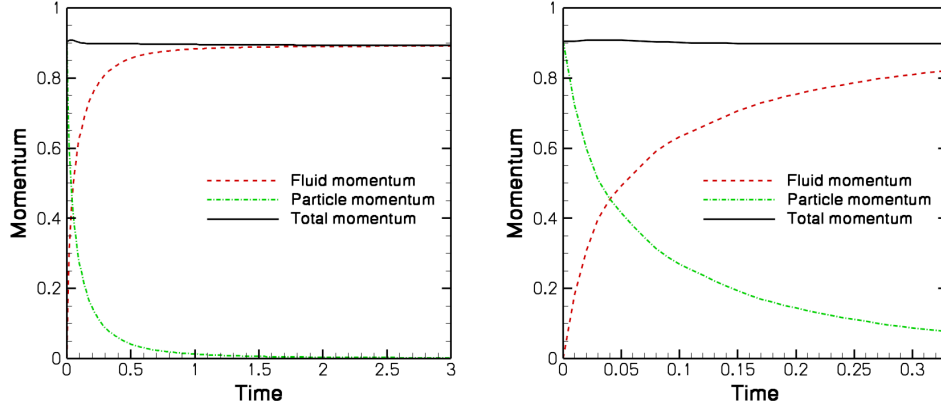


Fig. 3.15: The momentum associated with the particle phase, the fluid phase, and the combined momentum shown. The right-hand panel shows the initial period in greater detail.

The velocity after the equilibration is computed to be $1.4379 \cdot 10^{-2}$ based on Eqn. (3.730). Carrying out the Fuego simulation results in an equilibration average velocity of $1.4356 \cdot 10^{-2}$, which is only 0.1% below the computed value. The predicted resulting momentum and velocity were computed with differing methods within Enight: the momentum was computed using the volume integral while the velocity was computed using the spatial mean frequency. It is not clear what level of numerical error is attributable to the Enight algorithms.

The average gas velocity had equilibrated with within 99% of its final velocity within 1.06 time units. The value of τ_p for this system is $8 \cdot 10^{-2}$ from (3.626) and the momentum and velocity equilibration times are somewhat longer than the suggested response time near the particle equilibration time of a few τ_p since the particles were not injected uniformly through the domain. The result is substantial gas-phase velocity inhomogeneities that take much longer than τ_p to dissipate.

Table 3.20: Input parameters related to the verification of momentum conservation

Fluid properties		Particle properties	
ρ_{in}	1.0	ρ_p	10.0
$u_{g,in}$	0.0	$u_{p,in}$	10.0
μ_g	0.01	N_p	10
$\int dA$	3.10583	d_p	0.12

Mass and energy conservation verification

Mass and energy conservation for evaporating and reacting particles was verified in Fuego using fluid-phase parameter values taken from the thermodynamic databases and evaluated with Cantera. For the eulerian phase, the mass, momentum, energy, and species equations were evolved. The fluid and particle properties listed in table [Input parameters related to the verification of momentum conservation](#) were specified. Note that the critical temperature for water was set to a large value so that the enthalpy of the particles could be computed directly from $h_{vap,ref}$ as in Eqn. (3.729); this initial value of the particle enthalpy is also given in table [Input parameters related to the verification of energy conservation with evaporation](#). using the value of $h_F(T_{p,in})$ computed in Fuego, $-1.342 \cdot 10^{11} \text{ erg/g}$. Since the fluid composition and enthalpy are specified as boundary conditions or evolved within Fuego, the viscosity specific heat, and thermal conductivity are computed from these quantities. The simulation was carried out in the cylindrical_channel.g configuration with an inlet/outlet area of 3.10583 (with unit radius, this is nominally π , but low resolution of the circular cross-section leads to a smaller area) and a length of 20. Particles are injected iso-kinetically at a downstream location 1 unit from the inlet flow 19 units to the outlet in 19 time units.

Net mass conservation is evaluated using Eqn. (3.724). The fluid-phase mass flux at the inlet, using the Cantera computed density of $8.776 \cdot 10^{-4} \text{ g/cm}^3$, is $1.3628 \cdot 10^{-2} \text{ g/s}$. The particle mass flux is $1 \cdot 10^{-4} \text{ g/s}$ and combining the inlet mass fluxes gives an expected outlet mass flux of $1.3728 \cdot 10^{-2} \text{ g/s}$. The value computed from the outlet using Ensight is $1.37284 \cdot 10^{-2} \text{ g/s}$; there is a statistical variation in this quantity with a standard deviation of $3.7 \cdot 10^{-6} \text{ g/s}$. In a similar manner, mass conservation for individual species is evaluated using Eqn. (3.725). For water, the inlet mass flux is $2.7257 \cdot 10^{-5} \text{ gH}_2\text{O/s}$ and the mass injected is $1.0 \cdot 10^{-4} \text{ gH}_2\text{O/s}$. The value computed at the outlet plane is $1.2657 \cdot 10^{-4} \text{ gH}_2\text{O/s}$ which is approximately 1.5% below the expected value of $1.27257 \cdot 10^{-4} \text{ gH}_2\text{O/s}$. The outlet water mass flux has a statistical variation associated with it characterized by a standard deviation of $5.44 \cdot 10^{-8} \text{ gH}_2\text{O/s}$. While there is no source term for species like O_2 , the inlet and outlet mass fluxes can be computed. The inlet flux of O_2 is $3.13466 \cdot 10^{-3} \text{ gO}_2/\text{s}$ and that for the outlet is computed to be $3.13470 \cdot 10^{-3} \text{ gO}_2/\text{s}$, with a standard deviation of $8.5 \cdot 10^{-7} \text{ gO}_2/\text{s}$. Enthalpy conservation is evaluated using (3.728). The enthalpy flux into the domain associated with the fluid phase is $1.0477 \cdot 10^7 \text{ erg/s}$ while that associated with the particle phase is $1.568 \cdot 10^7 \text{ erg/s}$. The difference between these, $5.203 \cdot 10^6 \text{ erg/s}$, is the expected enthalpy flux at the outlet, the right hand side of (3.728). Using Ensight to evaluate the enthalpy flux at the outlet gives $5.10647 \cdot 10^6 \text{ erg/s}$; this is a 2% discrepancy, the source of which is uncertain at this point. There is a statistical variation in the outlet enthalpy flux characterized by a standard deviation of $6.38 \cdot 10^3 \text{ erg/s}$, approximately 0.04% of the total enthalpy change.

Table 3.21: Input parameters related to the verification of energy conservation with evaporation.

Fluid properties		Particle properties		Evaporation properties	
$T_{g,in}$	400.0K	ρ_p	1.0g/cm ³	$h_{vap,ref}$	26.694 · 10 ⁻⁹
$u_{g,in}$	5.0g/cm ³	$u_{p,in}$	5.0cm/s	$T_{vap,ref}$	373.0K
Y_{H_2O}	0.002	\dot{M}	1 · 10 ⁻⁴ g/s	$P_{vap,ref}$	1.0Atm
Y_{O_2}	0.22	$c_{v,p}$	4.184 · 10 ⁷ erg/g/K	T_{crit}	1 · 10 ⁹ K
Y_{N_2}	0.768	$T_{p,in}$	300.0K	$h_p(T_{p,in})$	-1.568 · 10 ¹¹ erg/g
Sc	0.9	Y_{H_2O}	1.0		
Pr	9.0	Pr_f	0.9		
Nu_f	2.0	Sc_f	0.9		
$\int 1dA$	3.10583cm ²	d_p	0.005cm		

Verification summary

To summarize the verification process, the following list shows the equations that are covered by verification tests described in the various sections in this document. The equations that are not fully covered are also indicated. The lack of verification coverage is only significant for the radiative terms.

- *Turbulent dispersion*
 - (3.621), (3.627), (3.628), (3.629), (3.630)
- *Droplet heating and cooling*
 - (3.674), (3.689), (3.690)
- *The wet bulb temperature, d²-Law Evaporation and Condensation*
 - (3.649), (3.651), (3.642), (3.666), (3.667), (3.669), (3.670)
- *d^{1.5}-law for Fast Moving Droplets*
 - (3.659) , (3.660), (3.661), (3.663)
- *Mass conservation, Species conservation, Energy conservation*
 - (3.693), (3.694), *Effect of the particle phase on the gas phase*
- *Momentum conservation*
 - (3.695)
- Equations employed but not yet fully covered in verification tests
 - (3.622), (3.631), (3.645), (3.650), (3.668), (3.682), (3.697), (3.698)

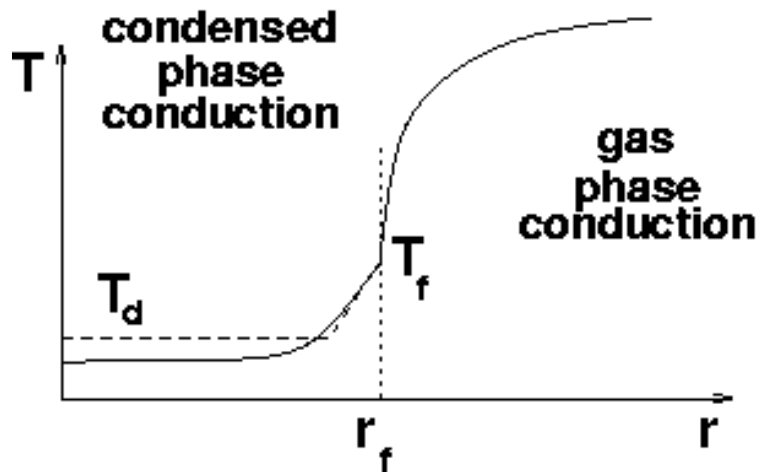


Fig. 3.16: Condensed-phase conduction is approximated based on the difference between the film and mean droplet temperatures and on an estimated heat transfer coefficient that describes a boundary layer thickness. Over this boundary layer thickness, the temperature difference $T_f - T_d$ is presumed to act.

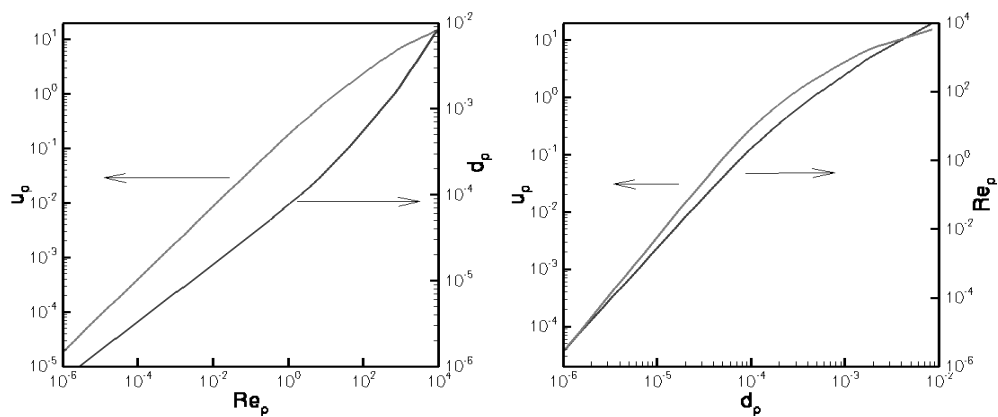


Fig. 3.17: Terminal velocities for particles as a function of diameter and particle Reynolds numbers determined from (3.711)

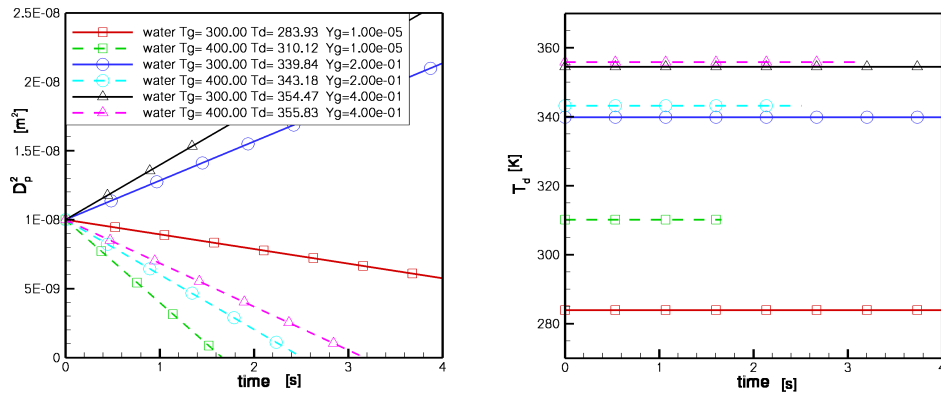


Fig. 3.18: Water droplet evaporation and condensation with initial temperatures set to the wet bulb temperature. Left plot exhibits the linear d^2 -law behavior while the right hand plot shows the droplet temperatures as constant (no heating).

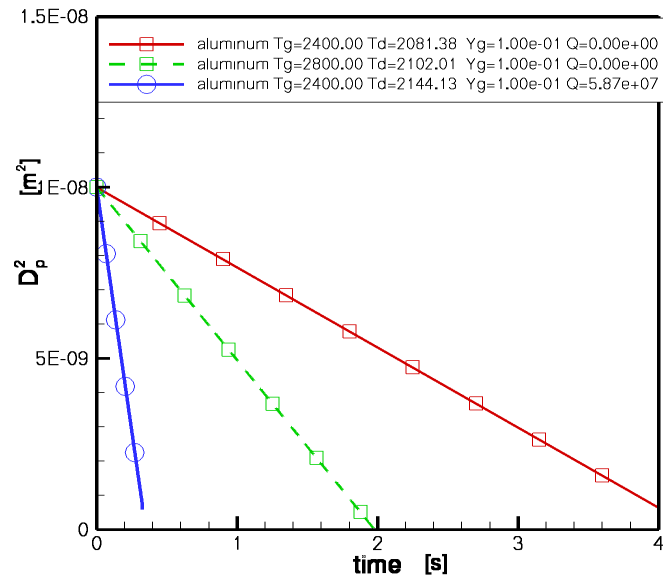


Fig. 3.19: Aluminum particle evaporation with and without combustion with initial temperatures set to the wet bulb temperature showing the linear d^2 -law behavior.

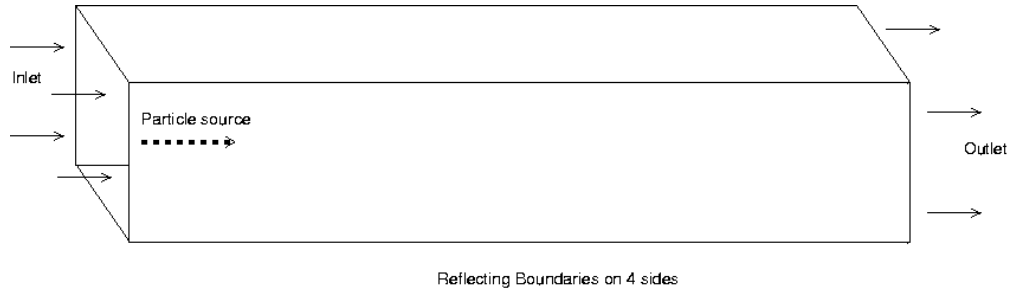


Fig. 3.20: The general configuration for verification of two-way coupling. The domain should be of sufficient length that the particles equilibrate with the gas-phase flow.

Summary

A Lagrangian model for particle transport coupled with an Eulerian solution for the gas phase is presented in detail. Models are presented for particle momentum, heat, and mass transfer, including the effects of turbulence on particle dispersion. Particular attention is paid to heat and mass transfer as these aspects are critical to the anticipated applications and they have not been well documented in other references. The heat and mass transfer models account for film temperatures that differ from particle temperatures in a manner that depends on the relative magnitudes of the internal particle heat transfer, the heat transfer to the particle surface from the gas phase, the heat transfer associated with radiative fluxes, and the enthalpies associated with evaporation and combustion around the particle. Both the evaporation and condensation are permitted. A conservative algorithm for coupling the Lagrangian and Eulerian fields is presented covering mass, species, momentum, and energy transfer between two fields. Models are also specified for the interactions of the Lagrangian field with solid boundaries.

A comprehensive plan to verify the implementation of the physics models is also presented. The verification plan touches on the majority of terms in the implemented physics models. Verification tests are provided for particle momentum, trajectories, heat, and mass transfer in various limiting cases for which analytic solutions can be obtained. Verification tests to evaluate the coupling between the Lagrangian and Eulerian fields are also provided. These verification tests are based on the net conservation of mass, species, energy, and momentum.

Evaluating transport coefficients

The droplet burning rate equations involve the area weighted diffusion coefficients as indicated in Eqn. (3.637) and subsequent equations. While the optimum method of determining the burning rate would involve the evaluation of these integrals as indicated in Equations (3.664) and (3.665), it is useful to estimate the effects of composition and temperature variations when such accurate evaluations are unfeasible. The kinetic theory of gases provides a starting point for such estimates, and a simplified overview of the pertinent results is provided. The single component viscosity is

$$\mu_i = \frac{5}{16} \frac{\sqrt{\pi W_k R T}}{\pi \sigma_k^2 \Omega^{(2,2)*}} \quad (3.731)$$

where σ_k is the Lennard-Jones collision diameter and $\Omega^{(2,2)*}$ is the collision integral. The mixture properties can be obtained using the Wilkes formula that averages based on mole fraction weighting to leading order. A square root dependence on the temperature is evident in (3.731), but the collision integral also includes a temperature dependence and it is found that the viscosities (and the other transport coefficients) are proportional to $T^{0.7}$, an empirical fact that is referred to as Sutherland's law. The kinetic theory of gases is only marginally successful at predicting the thermal conductivity, but the ratio of the thermal conductivity to the specific heat is closely related to the viscosity and the Prandtl number can often be approximated as constant. The binary diffusion coefficient between species i and j is more simply written as the product of the diffusion coefficient and the density since this removes additional pressure and temperature dependencies; this is

$$\rho D_{i,j} = \frac{3}{16} \frac{\bar{W} \sqrt{2\pi RT/W_{i,j}}}{\pi \sigma_{i,j}^2 \Omega^{(1,1)*}}. \quad (3.732)$$

Here the reduced mass and the reduced cross sections are $W_{i,j} = W_i W_j / (W_i + W_j)$ and $\sigma_{i,j}^2 = (\sigma_i + \sigma_j)^2$.

Lagrangian Particle Capabilities

Lagrangian Particle Spray: Diameter Cutoffs

The Fuego Lagrangian particle spray capability has a feature which allows an upper (high) and lower (low) size (diameter) cutoff to be set for particles inserted with a specified distribution (normal, normal mass, etc.). For distribution types with infinite tails like the standard normal distribution, the particle spray can select particle sizes small enough that they do not appear in the application of interest or so large that the assumption of the dilute spray model, inherent to the Fuego Lagrangian particle implementation, is violated. In specific applications where particles experience energetic chemical reactions, such as propellant fires, particles below a certain size range react quickly and disappear without the need to resolve their dynamics. The diameter cutoff feature allows the analyst to use standard distribution types while avoiding undesired particle size ranges. When diameter cutoffs are used, the particle pdf is adjusted accordingly to account for the lack of contribution from particle sizes outside the cutoff limits. The adjusted particle pdf is:

$$pdf_{new}(d) = pdf_{original}(d) H(d - d_{low}) H(d_{high} - d) / \int_{d_{low}}^{d_{high}} pdf_{original}(d) \quad (3.733)$$

where $pdf_{original}(d)$ is the original, uncutoff particle size pdf, $pdf_{new}(d)$ is the new particle pdf including low (d_{low}) and high (d_{high}) particle size cutoffs, the integral is taken on the original particle pdf with these limits, and H is the heaviside step function. This treatment properly normalizes $pdf_{new}(d)$. Figure 3.21 illustrates this for the case of a normal distribution of particle diameters ($\langle d \rangle = 0.5$, $\sigma = 0.1$) with and without diameter cutoffs at $d = 0.3$ and $d = 0.65$. Figure 3.22 shows a section of a Fuego input deck utilizing the diameter cutoff functionality.

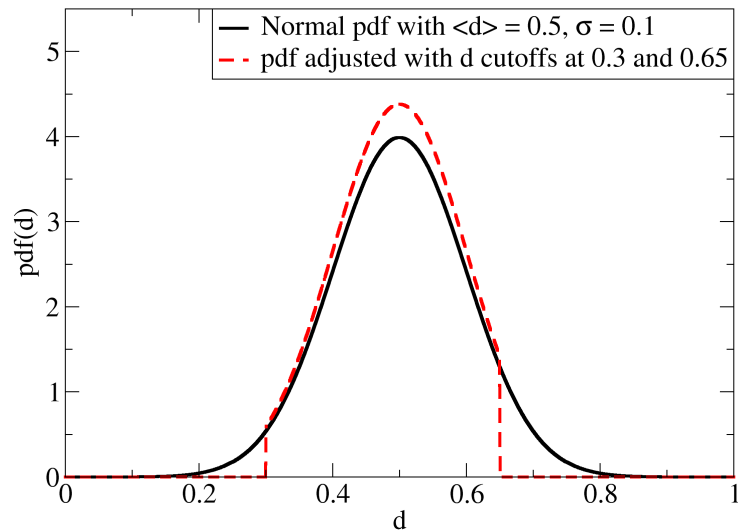


Fig. 3.21: Particle size (diameter) distribution for Lagrangian particle spray with and without diameter cutoffs set at 0.3 and 0.65

```
BEGIN PARTICLE SPRAY spray1
  PARTICLE DEFINITION = solid_particles
  #Spray Geometry
    CENTER = -100.0, 0.0, 0.0
    NOZZLE RADIUS = 10.0
    NORMAL VECTOR = 1 0 0
  #Size Distribution Parameters
    MASS FLOW RATE = 900.0 # g/s
    PARTICLE DIAMETER DISTRIBUTION TYPE = NORMAL
    PARTICLE DIAMETER DISTRIBUTION PARAMETER MEAN = 1.0 # cm
    PARTICLE DIAMETER DISTRIBUTION PARAMETER STDEV = 0.5 # cm
    DIAMETER CUTOFF HIGH = 1.05 # cm
    DIAMETER CUTOFF LOW = 0.95 # cm
    NUMBER REPRESENTED VECTOR = 10 20 30 40 50 60 70
    DIAMETER NUMBER REPRESENTED VECTOR = 0.1 0.4 0.7 1.0 1.3 1.6 1.9
  #Velocity Distribution Parameters
    PARTICLE VELOCITY DISTRIBUTION TYPE = CONSTANT
    PARTICLE VELOCITY DISTRIBUTION PARAMETER VALUE = 1.0 # cm/s
  #Particle Temperature
    TEMPERATURE = 300.0
END PARTICLE SPRAY spray1
```

Fig. 3.22: Lagrangian particle spray section of a Fuego input deck showing use of diameter cutoffs

Lagrangian Particle Spray: Angular Spreading Sprays

The angular spreading spray algorithm was modified in version 4.30 to produce an isotropically spreading particle spray (within the angular limits specified). Previously, the particle trajectories were preferentially aligned with the spray axis. For isotropic spread, the cosine of the polar angle (measured with respect to the spray axis) rather than the angle itself is chosen randomly. The polar angle is then determined from the inverse cosine of this value.

$$\theta = \cos^{-1} [\text{rand} ()] \quad (3.734)$$

Alumina Absorption Model

Fuego allows for a user to specify the radiation absorption model for alumina in reacting aluminum particle simulations like propellant fires. The alumina absorption model, using a FORTRAN subroutine, can now read from a user input file containing data for the alumina absorption coefficient as a function of particle temperature. The file contains two columns defining this function. The first column is temperature; the second is the absorption coefficient. This function is linearly interpolated to find the absorption coefficient at any temperature of interest. [Figure 3.23](#) displays two standard alumina absorption models alongside a user-specified model.

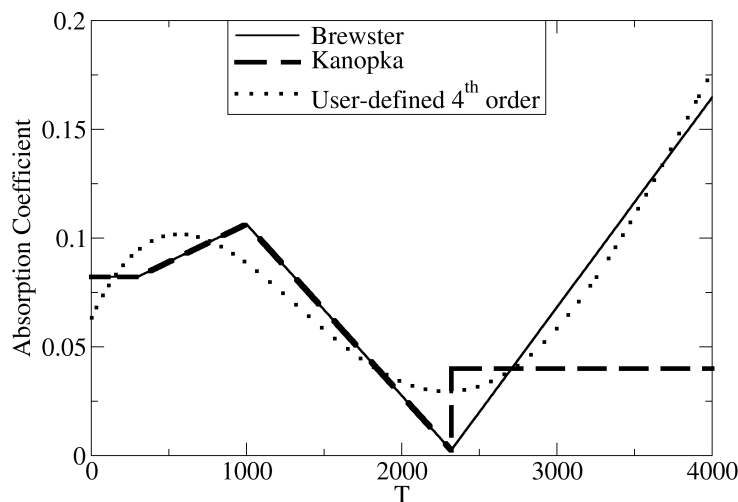


Fig. 3.23: Alumina absorption coefficient for standard models Brewster and Kanopka along with a user-specified model

Emission Multiplier

For propellant fire simulations which use the evaporating Lagrangian particle type, analysts have determined that modifying the particle-radiation coupling can be advantageous to reproducing experimental results. To address this, Fuego has a capability to modify the particle radiation emission with a constant multiplier. When the emission multiplier is not set, a default value of 1 is assumed, and emission equal to absorption when the particle and fluid temperatures are identical. Particle radiant emission E_p and absorption A_p are:

$$E_p = 4\pi\alpha R_p^2 \sigma_{SB} T_p^4 f_E \quad (3.735)$$

$$A_p = 4\pi\alpha R_p^2 \sigma_{SB} T_f^4 \quad (3.736)$$

where α is the particle absorptivity, R_p is the particle radius, σ_{SB} is the Stefan-Boltzmann constant, $T_{p,f}$ are the particle and fluid temperatures respectively, and f_E is the emission multiplier described above.

Lagrangian Particle Spray: Number Represented Function

Lagrangian particle sprays have historically been required to use parcelling (grouping of several particles into a single parcel) with either a constant mass represented per parcel or constant number represented per parcel. In propellant fire applications and other reacting particle environments, a more sophisticated functionality between the number of particles represented per parcel and particle size can increase the efficiency of simulations. For this reason, Fuego includes a capability to allow the analyst to specify this function (parcel size vs. particle diameter). This function is specified by a vector for each (number represented per parcel and diameter). For diameters at or below the lowest specified in the vector, the number represented is constant and equal to the value at the lowest diameter specified. For diameters at or above the highest specified in the vector, the number represented is constant and equal to the value at the largest diameter specified. Intermediate values are linearly interpolated. [Figure 3.24](#) diagrams the way parcelling works for each of the different parcelling schemes.

Lagrangian Particle Insertion: User Definable Mechanism

Previous to version 4.30, Lagrangian particles could be inserted into the domain through two mechanisms: 1) batch introduction of a group of particles at a specified time with the particle configuration defined by a particle data file or a filled shape (i.e. cone, cylinder) with shape parameters or 2) via a particle spray with either a rectangular or circular nozzle and a specified mass flux rate. In cases where users needed a more novel insertion mechanism, Fuego lacked the capability. Fuego now includes a mechanism for particle insertion from file data in which users can specify not only particle positions, velocities, and diameters on insertion, but also particle temperature, number of particles per parcel, and insertion time. Through this method users have a full range of particle insertion options at their disposal. The dynamical form for particle

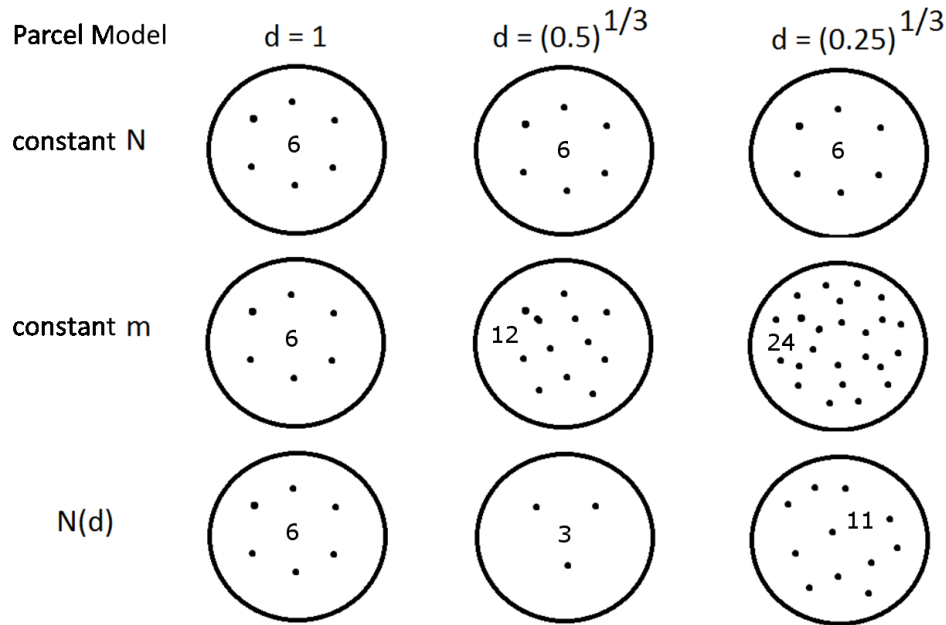


Fig. 3.24: For a Lagrangian particle spray, the number of particles contained within a parcel for three representative particle diameters using constant number, constant mass, and user defined number of particles per parcel. Circles represent parcels with the points inside representing the number of particles contained in the parcel.

introduction is contained within the file data, and does not rely on templated forms for static shapes or sprays, though those capabilities are still available. Users can, for instance, introduce particles from a very specific particle size distribution isotropically through the system with a rate of their choosing or create a particle spray with a conical nozzle with velocity vectors normal to the nozzle. The only limitation lies in the ability of the user to specify this mechanism through the particle data file. Figure 3.25 shows some examples of particle insertion types available with this capability. Figures *Example of particle spread from a conical shaped particle spray nozzle at early times. This nonstandard spray form was generated through the particle creation from file data mechanism. Here particle temperatures are set to be a function of their position with the hottest particles leaving the nozzle near the circular base of the cone.* and *Same simulation as Figure %s but at late time* display a conical particle spray generated with this mechanism from two different perspectives (conical axis lying in the plane of the figure and normal to the figure) at both early and late times in the simulation. In this case, the particle temperature has been designed to be a function of the position at which the particle left the spray nozzle. Many other forms are possible.

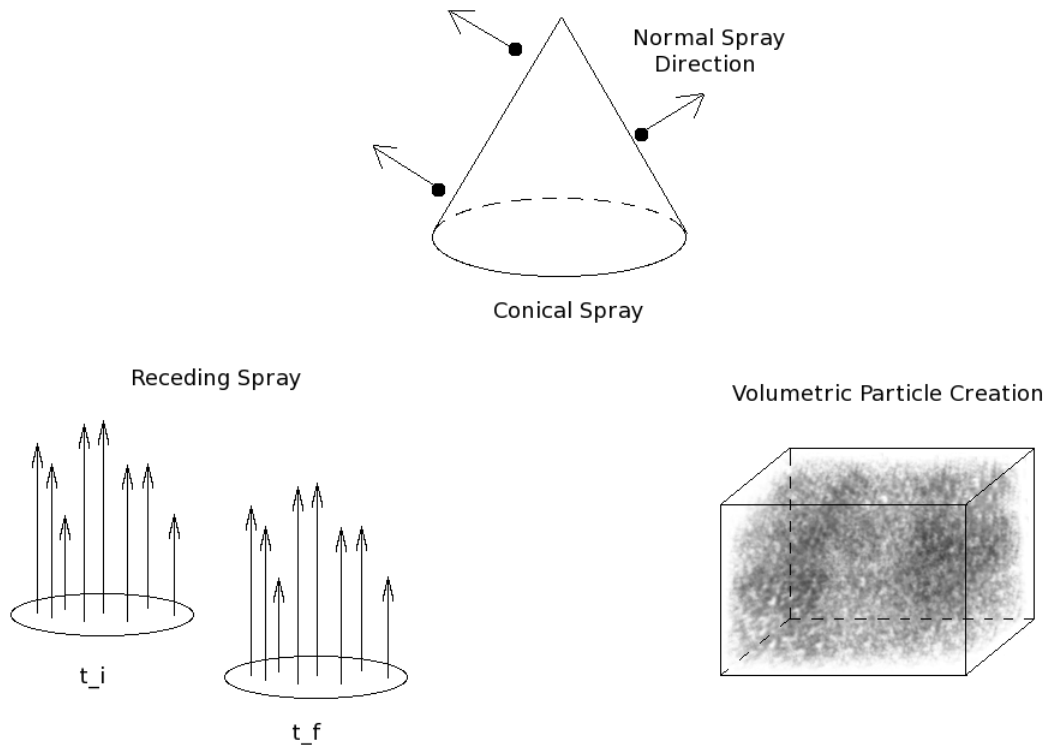


Fig. 3.25: Examples of particle insertion types that can be used with particle insert from file mechanism

3.4 Numerics

We surveyed commercial codes that provide turbulent combustion capabilities and discovered that most of those codes are based on finite volume methods. Between commercial evidence and our own experiences, we came to the conclusion that finite volume methods would provide a robust and stable means of solving the fire math models. Our selection of finite volume methods is constrained by the current implementation of software architecture in the SIERRA Frameworks. The mesh must be unstructured with flow variables located at the element vertices. The domain boundary is coincident with element faces. The discrete equations are assembled on an element-by-element procedure using the SIERRA workset approach for cache-use efficiency. The finite volume approach that we implement is based on the control-volume finite-element method.

Control-volume finite-element methods (CVFEM) are a class of numerical methods for solving the Navier-Stokes equations of fluid mechanics. Although the methods are applicable to the most general case of a compressible flow, they are most commonly applied to incompressible flows. This text is a discussion of the control-volume finite-element methods appropriate for numerical solutions to the low-Mach number Navier-Stokes equations with heat and mass transfer—the equations used to describe physical applications such as a combustion or chemical vapor deposition.

The CVFEM's are a combination of desirable features from both the finite-element method

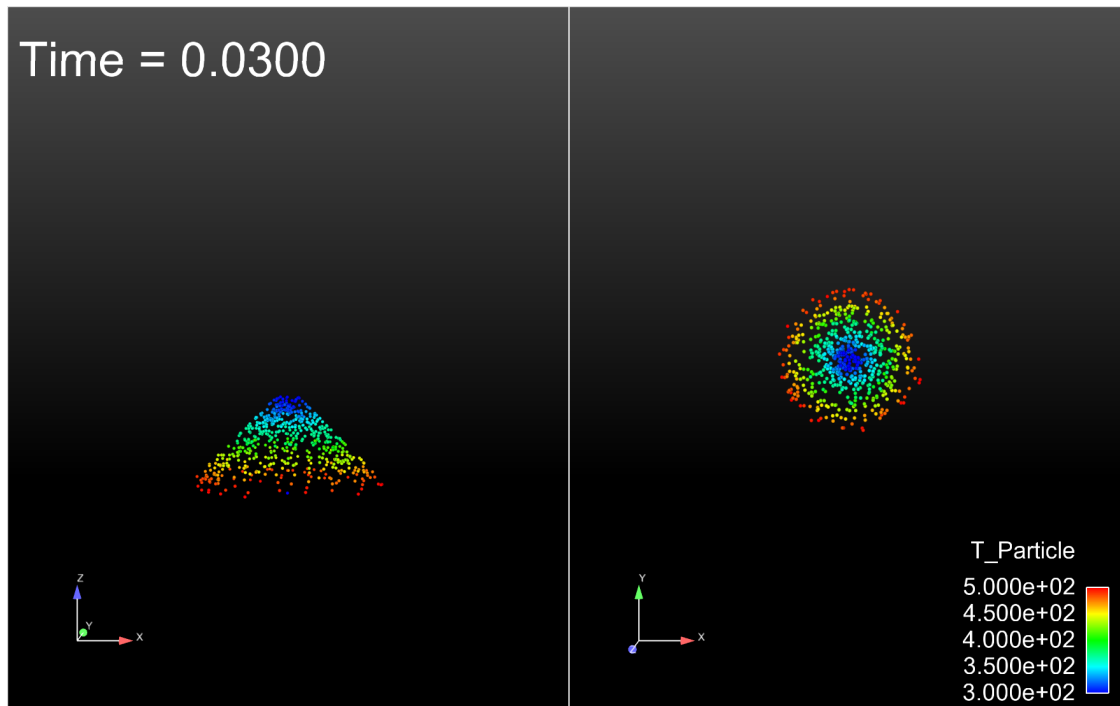


Fig. 3.26: Example of particle spread from a conical shaped particle spray nozzle at early times. This nonstandard spray form was generated through the particle creation from file data mechanism. Here particle temperatures are set to be a function of their position with the hottest particles leaving the nozzle near the circular base of the cone.

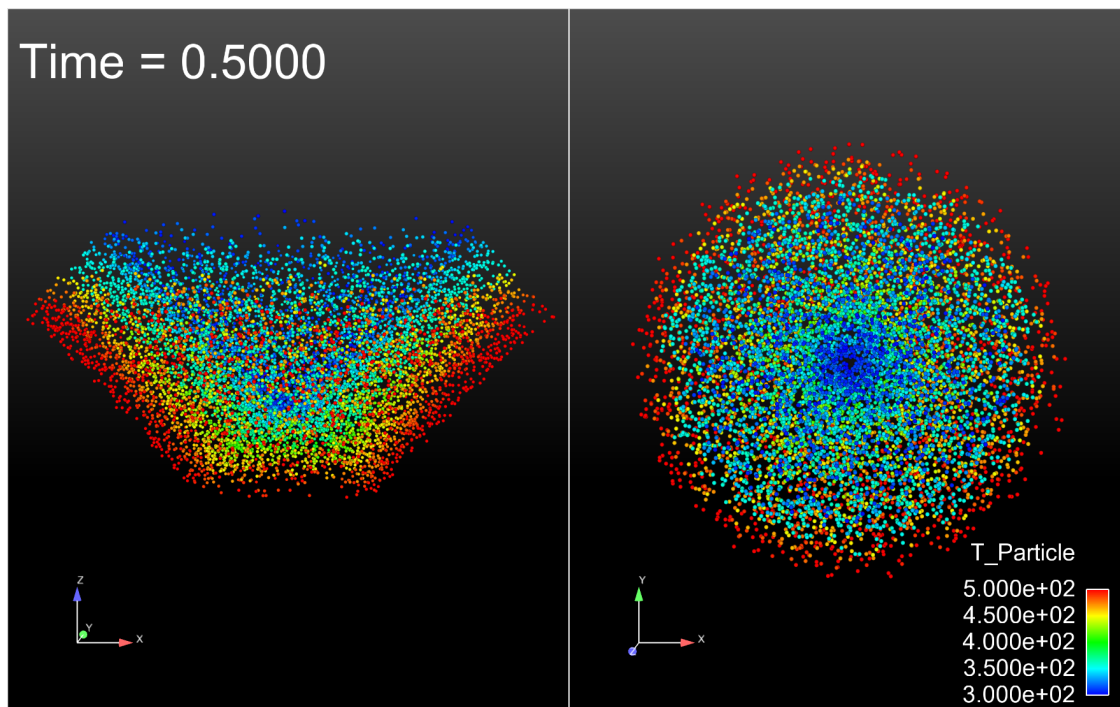


Fig. 3.27: Same simulation as Figure 3.26 but at late time

(FEM) and the finite-volume method (FVM), though the CVFEM is truly a finite-volume method. The CVFEM differed from other FVM's at its inception in that the CVFEM used non-staggered, unstructured meshes like a FEM. Concepts from the finite-element method include: 1) the finite-element data structure and the associated shape functions or interpolation functions, 2) integral equations assembled on an element-by-element basis, an efficient process for cache-based computer architectures, and 3) unstructured meshes with arbitrary connectivity (this is not particular to FEM's, but certainly more common). Reviews for the finite-element method are given by Zienkiewicz and Taylor [128, 129], Tezduyar [67], and Gresho [130]. Concepts from the finite volume method include: 1) physically-based integral formulation constructed from physically-based interpolation functions, 2) conservation properties at the control-volume level, and 3) both a convecting and convected velocity field to avoid pressure-velocity decoupling. Some comprehensive reviews for the finite-volume method are given by Patankar [131], Shyy [132], and Ferziger and Peric [133]. An extensive literature review of control volume finite element methods (CVFEM) is given in *Review of Control Volume Finite Element Methods*.

The standard mesh configuration for vertex-centered CVFEM's has all flow variables collocated at the grid points, also called nodes. The nodes are the vertices of the finite-elements, as shown in Figure 3.28. The finite-volumes, also called control volumes, are centered about the nodes. Each element contains a set of sub-faces that define control-volume surfaces. The sub-faces consist of the segments or surfaces that bisect the element faces.

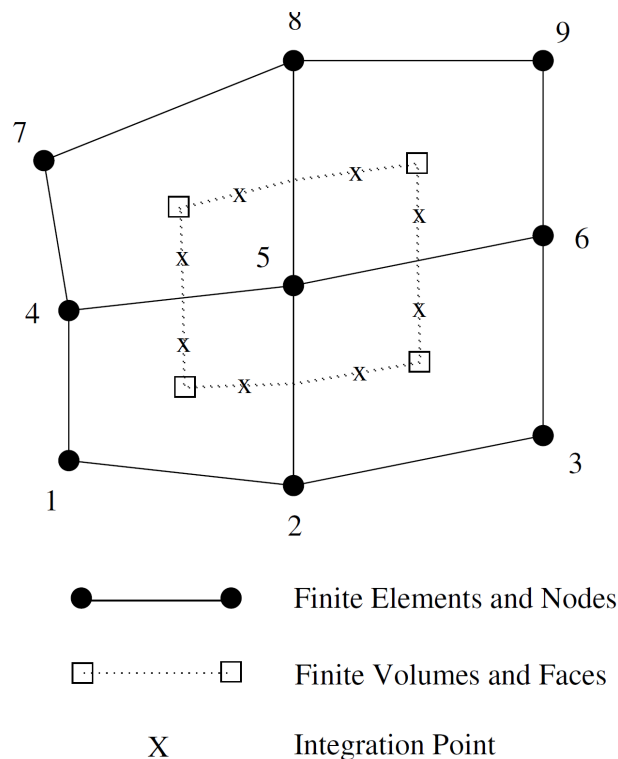


Fig. 3.28: Control Volume is Centered about Finite-Element Node

Review of Control Volume Finite Element Methods

The earliest reference to control-volume finite-element methods is the 1980 work by Baliga and Patankar [134] for the convection-diffusion equation, a refinement of Baliga's 1978 dissertation [135]. Baliga and Patankar [136] first apply their approach to the Navier-Stokes equations of fluid mechanics in 1983. At the same time, Schneider and Zedan [137] develop a control-volume finite-element for heat conduction. Schneider and Raw [138] then develop a control-volume finite-element method for fluid flow in 1986. The work of Baliga/Patankar and Raw/Schneider are the foundations for two of the main control-volume finite-element methods that are used today for fluid mechanics. A third control-volume finite-element method is adapted from Galerkin Least Squares (GLS) finite-element methods by Swaminathan and Voller [139] in 1992, but there is no evidence of widespread use.

There are three difficult issues that must be addressed in all numerical methods for the Navier-Stokes equations: 1) stability at high Reynolds number and Peclet numbers, where pure centered differencing for the convection terms, or the analogs in FEM and FVM, can lead to numerical oscillations, 2) coupling of the pressure and velocity field, where "checker-boarding" can occur when the variables are co-located and use similar interpolations, and 3) updating of the pressure field.

There are three main schools of thought in the CVFEM community for addressing the three issues above. With the Baliga/Patankar approach, upwinding is achieved with exponential shape functions on linear triangular and tetrahedral elements. Originally, pressure-velocity coupling was attained using mixed-order elements. Later, an equal-order scheme was developed that involved pressure terms in the interpolation functions. Convecting and convected velocities were maintained for pressure-velocity coupling. The pressure is solved using a projection method similar to the SIMPLER [140] algorithm. The method is practically limited to triangles and tetrahedra because of the form of the interpolation functions.

With the Raw/Schneider approach, upwinding is achieved using the skewed upwinding or positive influence coefficient approaches. The pressure and velocity are solved fully coupled using an approximation of the transport equations as an interpolation function. Two velocity fields are maintained, a convecting and a convected field. The method is applicable to all element forms and has been successfully implemented in a commercial computational fluid dynamics code, TASCflow [141].

With the Swaminathan/Voller approach, the methods of streamline upwinding and pressure stabilization are adapted from finite-element methods. There is only a small amount of literature on this particular CVFEM.

The following historical synopsis of CVFEM's addresses research for solving the pressure-based incompressible Navier-Stokes equations and the more elementary convection-diffusion equations.

- **1966** Winslow [142] presents a control volume formulation for a Poisson equation based on linear triangular elements. This work is important because it is one of the first applications of the finite volume method on unstructured meshes.

- **1980** Ramadhyani and Patankar [143] compare the accuracy of the Galerkin finite element method with a control volume method for the Laplacian operator. They use bilinear shape functions and rectangular elements, where the control volume method uses the bilinear shape functions as interpolation functions. The numerical errors of the control volume method are half those of the finite element method.
- **1980** Baliga and Patankar [134] introduce a flow-oriented upwind interpolation for convection-diffusion problems on triangular elements, a refinement of 1978 dissertation work [135]. The upwinding is introduced through an interpolation function based on a locally analytic solution to the velocity-aligned transport equation, resulting in exponential shape functions. They solve both radial heat conduction in a rotating hollow cylinder for Peclet numbers up to 100, and the transport of a step scalar field, all with specified velocity fields. The directional upwinding provides better solutions than uniform first-order upwinding.
- **1983** Baliga and Patankar [136] develop a mixed-interpolation scheme for solving the Navier-Stokes equations with heat transfer on triangular elements. The mixed interpolation keeps the pressure from decoupling from velocity. The pressure is solved by applying the continuity equation over macro-triangles. The interpolation function for the convecting velocity contains the pressure gradient. Each macro-triangle is subdivided into four sub-triangles for the momentum and energy equations. The flow-oriented upwind scheme is used to interpolate velocity and temperature for their respective transport equations. The velocity is assumed to vary linearly over the element for computing mass flow rates. The equations are solved in a segregated manner using an approach similar to the SIMPLER method [140]. This work is the first application of the CVFEM for the Navier-Stokes equations.
- **1983** Baliga, Pham, and Patankar [144] apply the mixed-interpolation scheme [136] to fluid flow and heat transfer. They solve flow between rotating cylinders for Reynolds numbers up to 1000, fully developed flow in a square duct with a laterally imposed velocity for Reynolds numbers up to 100, natural convection in rectangular enclosures for Rayleigh numbers up to 10^5 , and natural convection in a trapezoidal enclosure for Rayleigh numbers up to 10^6 .
- **1985** Prakash and Patankar [145] solve the Navier-Stokes equations with an equal-order interpolation for velocity and pressure on triangular elements. The mass flow velocity, used for continuity, is different from that derived from momentum, thus avoiding staggering or mixed-interpolation. The flow-oriented upwind scheme is used to interpolate the convected velocity for the momentum equation while the pressure gradient is treated as an element-constant source term. The coefficient matrices for momentum are used to define the velocities for the continuity equation which include the now-unknown pressure gradient across control volume faces. They use a pressure correction approach similar to the SIMPLER algorithm to update velocity and accelerate convergence. The continuity and momentum equations are segregated in the solution process. They solve flow between rotating cylinders for Reynolds numbers up to 1000, and natural convection in a closed cavity with a Boussinesq-type buoyancy term for Grashof numbers up to 10^5 . The solutions are more accurate than with the mixed interpolation scheme of Baliga [136]. They note problems with negative coefficients during the first iterations of a solution.

- **1985** Ramadhyani and Patankar [146] extend the flow-oriented upwind interpolation scheme from linear triangles to bilinear quadrilateral elements for convection-diffusion problems. Three-point quadratures (Simpsons Rule) are used to evaluate flux integrals, as in all the previously mentioned work. They argue that one-point quadratures are less accurate because of the nonlinear nature of the interpolation functions, but only at intermediate values of cell-Reynolds number. They present solutions for five different test cases, including the convection of scalar profiles and diffusion in rotating systems. After this article, there are no further publications for quadrilateral or hexahedral elements using methods developed by Baliga, Patankar, and Prakash.
- **1986** Schneider and Raw [138] develop a positive influence-coefficient extension to skewed upwind interpolation [147] for convection terms, based on 1985 dissertation work [148]. They apply the scheme to the convection-diffusion equation on quadrilateral elements. Diffusion terms are calculated by integrating the gradients of the isoparametric, bilinear interpolation functions. They solve several convected-scalar cases and claim smooth solutions where the methods of Baliga and Patankar exhibit oscillations. The skewed upwind method has less dependence on the element orientation than flow-oriented streamline upwinding.
- **1986** LeDain-Muir and Baliga [149] extend the flow-oriented upwind interpolation scheme to linear tetrahedral elements in three dimensions for the convection-diffusion problem. Each tetrahedron contains six control volume faces. A single unit normal is calculated for each control volume face. For integration, each face is subdivided into two triangles. A three-point quadrature is used on each triangular subface where the sample points are taken along the midpoints of the triangle edges. They solve radial heat conduction in a rotating hollow sphere, scalar transport of a step profile, and transport with radial convection between concentric spheres.
- **1986** Prakash [150] modifies the flow-oriented upwind interpolation to include source terms from the transport equations on triangular elements, with applications to the incompressible Navier-Stokes equations. The pressure gradient in the momentum equations is treated as a source term in the interpolation function for velocity, directly coupling the pressure to the velocity. The source term has a streamwise-linear influence on the interpolation function. The mass flux is calculated using the new interpolation function instead of assuming a linear variation. The pressure is then calculated through the continuity equation by directly applying the new velocity interpolation function, replacing the SIMPLER scheme but keeping the segregated approach. A pressure correction step is included to make sure the velocity interpolation function satisfies continuity. He solves flow between rotating cylinders up to a Reynolds number of 1000, the lid-driven cavity for Reynolds numbers up to 400, and natural convection in a square cavity for Grashof numbers up to 10^5 . The solutions are more accurate than with the original collocated scheme of Prakash and Patankar [145].
- **1987** Schneider and Raw [151, 152] extend the positive-coefficient, skewed upwind interpolation [138] to the incompressible Navier-Stokes equations on quadrilateral elements. They use an element-local discretization of the transport equations to derive interpolation functions at control volume faces that couple the velocity and pressure. The convection terms are constructed with the positive-coefficient, skewed upwinding. The

skewed upwinding couples all the control volume face values together within an element, so an internal matrix inversion must be applied to calculate individual face values. The momentum and continuity equations are solved all at once as a coupled system. They solve convection of a scalar field with a step profile, the lid-driven cavity for Reynolds numbers up to 1000, the inviscid forward-facing step to test the conservation of total pressure, and flow between rotating cylinders. Grid convergence studies suggest spatial accuracy near second order. They call their method Finite Element Difference Scheme (FIELDS).

- **1987** Schneider [153] extends their algorithm [151] to cylindrical, axisymmetric coordinates and presents solutions for the cylindrical driven cavity.
- **1987** Prakash [154] examines a donor-cell method for replacing flow-oriented upwind interpolation on triangular elements. The donor cell method provides positive coefficients, where the flow-oriented upwinding can yield negative coefficients, leading to oscillations. The donor-cell scheme is applied to several of the previous convected scalar problems and the thermally driven cavity. The scheme exhibits excessive diffusion and is not generally recommended.
- **1988** Hookey, Baliga, and Prakash [155] modify the treatment of the source term in the flow-oriented upwind interpolation for triangular elements relative to the previous source term modifications of Prakash [150]. A crossflow-quadratic multiplier is added for the source term in the interpolation function. They apply the scheme to the convection-diffusion equation for radial heat conduction between rotating cylinders and radial heat conduction in radial flow between cylinders. The new source treatment proves better than the previous scheme of Prakash only when the flow has multidimensional features.
- **1988** Hookey and Baliga [156] apply the flow-oriented upwind interpolation with the modified source treatment [155] to the incompressible Navier-Stokes equations on triangles. Instead of calculating pressure by applying the interpolation functions directly to the continuity equation as was done by Prakash [150], a method similar to SIMPLEC [157] is used. The previous approach converged poorly at higher Reynolds numbers. A pressure correction approach is still employed to force the interpolation function for velocity to satisfy continuity, but with the penalty of an enlarged stencil for the pressure-correction equation. The continuity and momentum equations are solved simultaneously. They solve a polar lid-driven cavity for Reynolds numbers up to 380 and the natural convection for Rayleigh numbers up to 10^6 . Solutions are compared against results from the older methods of Prakash [150] and Baliga [136], and prove to be more accurate.
- **1988** Reviews of control volume finite element methods for fluid flow and heat transfer are given in the Handbook of Numerical Heat Transfer by both Baliga [158] and Schneider [159]. They provide implementation details for many of the methods published to date.
- **1992** Swaminathan and Voller [139, 160] extend of the ideas of the Streamline-Upwind Petrov-Galerkin (SUPG) method [161] to solving the convection-diffusion equation with quadrilateral elements. They solve several convected-scalar problems and compare the results to a FEM implementation of the SUPG scheme. The CVFEM analog of SUPG performs just as well, except for time accurate solutions where the phase error is larger. The

SUPG method provides better solutions than the skew upwinding or flow-oriented upwinding for heat conduction between rotating cylinders, but worse for the scalar transport of a step profile.

- **1992** Baliga and Saabas [162] provide a critical review of control volume finite element methods. They criticize the schemes of Hookey [156] and Raw [151] for being too expensive, computationally. They introduce the mass advection weighted scheme of Saabas where he adapts the concept of positive influence-coefficients from Schneider and Raw to the formulation of Baliga and Prakash. They call the original flow-oriented upwind scheme of Baliga and Patankar FLO, the source-term modified scheme of Prakash FLOS, and the mass advection-weighted scheme of Saabas MAW. The FLO(S) schemes result in mixed-sign off-diagonal coefficients if the triangular elements are obtuse, potentially admitting oscillations. Additionally, many of the schemes developed to date, for CVFEM, over-specify the pressure boundary conditions, leading to poor convergence.
- **1992** Naterer and Schneider [163] extend the approach of Schneider and Raw [151] to compressible flow. An explicit predictor-corrector time integration is used for transient solutions. The influence-coefficient matrices are used to interpolate density, velocity, and internal energy at control volume faces at an intermediate time level. These values are then used to correct the state variables using a forward Euler integration. They solve a transient shock tube problem for an initial pressure ratio of 10, flow through a converging-diverging nozzle with an area ratio of 2, and Mach 3 supersonic flow over a forward-facing step.
- **1993** Swaminathan, Voller, and Patankar [164] extend the streamline-upwind Petrov-Galerkin method and the pressure-stabilized Petrov-Galerkin [161] method to a conservative form for the control volume finite element method. The streamline-upwind control-volume, pressure-stabilized control-volume (SUCV/PSCV) method is applied to the incompressible Navier-Stokes equations with quadrilateral elements. They evaluate the integrals using mid-point quadrature and solve the segregated equations using a SIMPLER approach. They solve the lid-driven cavity at a Reynolds number of 400, natural convection in a square enclosure for a Rayleigh number of 10^5 , and natural convection in a cylindrical annulus at a Rayleigh number of 10^4 .
- **1994** Saabas and Baliga [165, 166] adapt the positive influence-coefficient scheme of Schneider and Raw [151] to triangular and tetrahedral elements and call the method mass advection weighting (MAW). They introduce a new control volume construction for tetrahedral elements. Their tetrahedral element contains one four-point planar face and two three-point planar faces, whereas the control volume construction of LeDain-Muir [149] contained six four-point surfaces. The reduced number of control volume faces makes the MAW scheme less expensive to apply, but the element shape functions become dependent on the shape of each element. They solve for pressure using the original approach of Prakash [145] with a SIMPLER method. The solution technique is segregated. For solving practical problems, they recommend using the FLO scheme for the convection terms and switching to the MAW scheme only if there are problems with negative coefficients. They advise against using the FLOS schemes of Prakash [150] and Hookey [156] because they typically do not provide enough improvement in accuracy to justify their slower convergence properties. Additionally, they claim that carrying pressure gradient terms in

the velocity interpolation function requires the boundary conditions for pressure to be over-specified for inflow/outflow problems. They solve the 2D lid-driven cavity for Reynolds numbers up to 1000, 2D turbulent flow over a backward-facing step using a $k - \epsilon$ turbulence model for a Reynolds number of about 10^6 , 3D natural convection in a cavity for Rayleigh numbers up to 10^6 , and a turbulent jet injection into crossflow for jet Reynolds numbers up to 53600. The MAW scheme is required for the jet problem because of negative coefficient problems with the FLO scheme.

- **1994** Masson, Saabas, and Baliga [167] extend the MAW scheme of Saabas [165] to axisymmetric flows with triangular elements. They solve developing pipe flow for a Reynolds number of 40, pipe flow with a step constriction up to a Reynolds number of 1000, natural convection in a cylindrical enclosure for a Grashof number of 2 and Prandtl number of 2500, and flow through an arterial section for Reynolds numbers up to 350.
- **1994** Masson and Baliga [168] apply the MAW scheme of Saabas [165] to dilute-particle flows with triangular elements. They solve equations for the gas phase and the dispersed phase. They solve for flow through a constricted channel for a Reynolds number of 100 and Stokes numbers between 10^{-3} and 10^{-1} , and for flow in a split inertial separator for a Reynolds number of 200.
- **1994** Karimian and Schneider [169] improve the velocity-pressure coupling of the original Schneider-Raw scheme [151]. The original scheme, referred to as FIELDS, has poor performance for inviscid flow. They improve the coupling by adding a discrete continuity relation to the interpolation functions for the convecting velocity. The additional terms help smooth oscillations that occur for a mass sink test problem. They verify the new interpolation function on the lid-driven cavity for Reynolds numbers up to 3200, and the backward-facing step for Reynolds numbers up to 230.
- **1994** Karimian and Schneider [170] extend the method of Schneider and Raw [151] to both compressible and incompressible flow for the quasi-one-dimensional Euler equations. They solve for flow through a converging-diverging nozzle with an area ratio of 2.035 with and without a shock.
- **1994** Deng *et al.* [171] present a new flux reconstruction scheme to replace the FIELDS scheme of Schneider and Raw [152]. They note that the FIELDS scheme is similar to the original work of Rhie and Chow [172] who were some of the first researchers to solve incompressible flow on collocated grids. Deng takes features of both schemes to create a compact reconstruction that does not require matrix inversions to calculate the integration point values in terms of nodal values. Since they question the consistency of the FIELDS scheme, they call their new scheme consistent physical interpolation (CPI). They apply the scheme to two and three-dimensional Navier-Stokes calculations on structured Cartesian meshes. They solve the lid-driven cavity for Reynolds numbers up to 1000, a 3D lid-driven cavity for Reynolds numbers up to 1000, and turbulent vortex shedding over a square cylinder for a Reynolds number of 22000.
- **1995** Costa *et al.* [173] apply the MAW scheme of Saabas [165] to three-dimensional turbulent flows with tetrahedral elements. They solve a turbulent jet injected into a crossflow for jet Reynolds number up to 53600, and flow through a T-junction in ducts at

Reynolds numbers near 90000.

- **1995** Karimian and Schneider [174] apply control-volume finite-element methods to a shock-tube problem.
- **1995** Karimian and Schneider [175] extend the FIELDS scheme and the convecting velocity corrections to compressible flow. They solve the lid-driven cavity for Reynolds numbers up to 3200, flow over a shallow bump in a channel with Mach numbers from 0.5 to 1.65, and flow through a ramped inlet for a Mach number of 2.5.
- **1995** Padra and Larreteguy [176] develop an error estimator with mesh refinement for the convection-diffusion equation. They use the formulation of Baliga and Patankar [134] with triangles. Larreteguy [177] then extends the scheme to fluid flow with triangles.
- **1996** Harms *et al.* [178] introduce a simplified interpolation function for the control volume finite element method. They develop a method for applying analytic shape functions on nonorthogonal meshes. They apply the scheme to flow between rotating cylinders for Reynolds numbers up to 1000 and the scalar transport of a step profile.
- **1996** Comini *et al.* [179] compare CVFEM and GFEM formulations for the convection-diffusion equations.
- **1996** Neises and Steinbach [180] develop a control volume finite element method based on a mixed interpolation approach to facilitate pressure-velocity coupling and artificial dissipation for convective stability. They begin with a Galerkin finite element method and then manipulate off-diagonal terms to force conservation. They solve a laminar, 3D obstructed channel flow for Reynolds number from 0.1 to 50000.
- **1996** Volker, Burton, and Vanka [181] apply a multigrid solution technique to a control volume finite element method on triangular elements. Linear interpolation is used throughout the triangles with a three-point quadrature to integrate fluxes. The pressure is solved using the SIMPLE method. They solve natural convection problems in square, triangular, and semicircular cavities for Rayleigh numbers up to 10^6 .
- **1996** Botta and Hempel [182] describe a finite-volume projection method for unstructured, triangular meshes with element-centered variables.
- **1997** Darbandi and Schneider [183] develop a scheme for both compressible and incompressible flow using a momentum variable formulation of the Schneider/Raw scheme [151, 152]. The interpolation formula for the convecting velocities is derived from an approximation of the momentum equation with an additional velocity-weighted continuity equation term. Solutions are demonstrated for velocities up to Mach 0.9.
- **1997** Baliga [184] gives an overview of the control volume finite element method as applied to fluid flow.
- **1998** O'Rourke and Sahota [185] develop an edge-based scheme in 3D for the convection operator. The convection operator is constructed from a multidimensional upwind scheme. Within each element, the quadrature points are associated with edge mid-points instead of sub-face mid-points, so the amount of work is reduced over the traditional CVFEM.

- **1998** Gresho and Sani [186] compare CVFEM methods to GFEM methods.
- **1998** Venditti and Baliga [187] describe an error estimation strategy for incompressible flow with CVFEM.
- **2001** Reyes, Rincon, and Damia [188] present a CVFEM approach for turbulent flow with wall functions.
- **2001** Campos Silva and de Moura [189] present a method for 9-noded quad elements with the mass advection weighted scheme.
- **2002** Zhao, Tai and Ahmed [190] implement a 2D CVFEM on triangles for micro flows. They use an upwind scheme where nodal gradient are used to reconstruct the high-order fluxes at the control volume faces.

With respect to fluid flow, the CVFEM methods have been developed primarily for triangular and tetrahedral elements [134, 136, 145, 149, 150, 155, 156, 165, 167]. Development focused on triangular and tetrahedral elements because the shape functions are linear and gradient terms become constant over the element. Constant first derivatives simplify the formulation of many of the schemes. Fewer articles have been published on the use of quadrilateral elements [138, 139, 146, 151, 164] in two dimensions and no articles have been published for CVFEM with hexahedral elements in three dimensions. In addition, there have been CVFEM formulations for the streamline-vorticity equations [39, 191, 192, 193, 194, 195], for the heat equation [137, 196, 197, 198, 199, 200, 201, 202, 203], for flow in porous media [204, 205, 206, 207, 208, 209, 210], for overland flows [211, 212], and for linear elasticity [213, 214].

Flow Solver

The core flow solver is based on a segregated, projection method approach. The projection method is used to compute the pressure field which is consistent with a velocity field that satisfies continuity. A pressure-smoothing approach similar to the Rhie/Chow scheme [172] is used to prevent pressure decoupling on the collocated mesh. An upwind method is used to interpolate convected values to control volume faces. Detailed descriptions of these methods are discussed in the following sections.

Another prevalent CVFEM method in the literature is the FIELDS method [151, 152]. The continuity and momentum equations are fully coupled in this approach. We experimented with this approach and found that the three-dimensional discrete equations were difficult to solve and open boundary conditions difficult to implement.

Projection Method

The role of pressure smoothing, or explicit stabilization, was first developed in the context of collocated finite volume schemes by [172]. Although this original paper did not explore the formal error introduced by this explicit stabilization, [215] later displayed the sensitivity of steady results on relaxation parameters and provided a methodology to circumvent this issue. In general, such early papers (cf. [216]) as well as other more recent papers, (cf. [217]) introduced the role of stabilization almost by happenstance as it entered only through the specific choice of the convecting velocity formula, i.e., the integration point velocity that forms the mass flow rate.

Studies of [218] and [219], each in the context of a finite element algorithm, have commented on the role of stabilization that is provided by the approximation of the derived pressure correction system, namely that $\mathbf{L} \neq \mathbf{D}\mathbf{G}$, where \mathbf{L} is the given discrete Laplacian operator and \mathbf{D} and \mathbf{G} are the chosen discrete divergence and gradient operators, respectively. Numerical algorithms for which the Laplacian operator does not equal the discrete divergence of gradient operator have been termed “approximate projection” algorithms (cf. [220] and [221]) in the context of solenoidal flow; in general for non-solenoidal flow the formalism of the projection derivation results in an affine projection.

Recent work by Sandia National Laboratories has cast the general approximate projection algorithm within a family of smoothing and time scaling choices. The analysis of choice that has been followed is to cast the algorithm in terms of an approximate factorization (cf. [222]), and note the added stabilization (herein also known as *pressure smoothing terms*), and splitting errors. This analysis has been extremely useful in understanding the formal accuracy, and even consistency, of a given numerical scheme.

The analysis of a given computational fluids algorithm in the context of an approximate factorization begins with the discrete momentum and continuity equations written in matrix form. The matrix \mathbf{A} contains discrete, linearized contributions to the momentum equations from the time derivative, convection, and diffusion terms,

$$\begin{bmatrix} \mathbf{A} & \mathbf{G} \\ \mathbf{D} & \mathbf{0} \end{bmatrix} \begin{bmatrix} \mathbf{u}^{n+1} \\ p^{n+1/2} \end{bmatrix} = \begin{bmatrix} \mathbf{f} \\ \mathbf{b} \end{bmatrix}. \quad (3.737)$$

The discrete nodal gradient and nodal divergence are \mathbf{G} and \mathbf{D} respectively (note that the operator \mathbf{D} may include aspects of the algorithm due to a variable density field). The function \mathbf{f} contains the additional terms for the momentum equations, e.g., body force terms, lagged stress tensor terms, etc., while the function \mathbf{b} contains the appropriate terms for a non-solenoidal velocity field, i.e., $-\int \frac{\partial p}{\partial t} dV$. The pressure is appropriately interpreted as the pressure at the $n + \frac{1}{2}$ step, (cf. [223]). The form of the matrix operators can be found in the body of literature for control-volume finite element methods (cf. [158]).

Note that (3.737) is not really complete as the boundary condition values are omitted, however, they are not essential in describing the bulk of the splitting and stabilization analysis as noted by [224]. The boundary conditions would simply enter through an additional vector on the right-hand side and modified entries in the matrix operators.

The approximate factorization of (3.737) takes the general form of

$$\begin{bmatrix} \mathbf{A} & \mathbf{0} \\ \mathbf{D} & \mathbf{B}_1 \end{bmatrix} \begin{bmatrix} \mathbf{I} & \mathbf{B}_2 \mathbf{G} \\ \mathbf{0} & \mathbf{I} \end{bmatrix} = \begin{bmatrix} \mathbf{A} & \mathbf{A} \mathbf{B}_2 \mathbf{G} \\ \mathbf{D} & \mathbf{B}_1 + \mathbf{D} \mathbf{B}_2 \mathbf{G} \end{bmatrix} \quad (3.738)$$

The factor \mathbf{B}_2 determines the projection time scale. The factor \mathbf{B}_1 defines the linear system for pressure. Ideally, \mathbf{B}_1 could be selected to cancel splitting errors in the continuity equation. Practically, the form of \mathbf{B}_1 is governed by implementation and linear solver efficiency.

A completely generalized set of incremental pressure projection methods with potential stabilization can be written by formally defining the operators \mathbf{B}_1 and \mathbf{B}_2 above, here shown as part of the sequence of equations solved,

$$\mathbf{A} \Delta \hat{\mathbf{u}} = \mathbf{f} - \mathbf{G} p^{n-\frac{1}{2}} - \mathbf{A} \mathbf{u}^n, \quad (3.739)$$

$$-\mathbf{L}_1 \Delta p^{n+\frac{1}{2}} = -\mathbf{D} \left(\hat{\mathbf{u}} + \tilde{\tau}_2 \mathbf{G} p^{n-\frac{1}{2}} \right) + \mathbf{L}_2 p^{n-\frac{1}{2}} + b, \quad (3.740)$$

$$\mathbf{u}^{n+1} = \hat{\mathbf{u}} - \tilde{\tau}_3 \mathbf{G} \Delta p^{n+\frac{1}{2}}. \quad (3.741)$$

Laplacian operators acting on a general scalar ϕ , which define the approximate nature of the projection method, are given by,

$$\mathbf{L}_1 \phi = \tau_1 \nabla \phi \cdot d\mathbf{A}, \quad (3.742)$$

$$\mathbf{L}_2 \phi = \tau_2 \nabla \phi \cdot d\mathbf{A}. \quad (3.743)$$

For an approximate projection method,

$$\mathbf{L}_2 \neq \mathbf{D} \tilde{\tau}_2 \mathbf{G}, \quad (3.744)$$

while for an exact projection,

$$\mathbf{L}_2 = \mathbf{D} \tilde{\tau}_2 \mathbf{G}. \quad (3.745)$$

Exact projections can be easily constructed on unstructured collocated meshes (cf. [225]), although classically this results in a wide Laplacian stencil that admits pressure oscillations yet does not add discrete errors in the continuity solve. We assume that τ_i factors defined above are represented by a diagonal matrix that corresponds to a particular time scale of choice. The relationship between $\tilde{\tau}_i$ and τ_i is normalization by a density and volume,

$$\tilde{\tau}_i = \frac{\tau_i}{\rho V}. \quad (3.746)$$

The choice of these scaling factors defines the scheme in terms of both stabilization and projection scaling. For example, the ideal form for $\tilde{\tau}_3$ is the inverse of \mathbf{A} . The exact choice of $\tilde{\tau}_3$ in a practical sense affects the stability of the scheme. The stabilization terms are represented by operators including both $\tilde{\tau}_1$ and $\tilde{\tau}_2$ that are required to prevent velocity and pressure decoupling in schemes for which $\mathbf{L} \neq \mathbf{D} \mathbf{G}$.

Rearrangement of (3.741), in terms of $\hat{\mathbf{u}}$, and substitution of this modified equation into (3.739) and (3.740) provides the full set of splitting and stabilization errors:

$$\begin{aligned} & \begin{bmatrix} \mathbf{AG} \\ \mathbf{D0} \end{bmatrix} \begin{bmatrix} \mathbf{u}^{n+1} \\ p^{n+1/2} \end{bmatrix} = \\ & \begin{bmatrix} \mathbf{f} \\ \mathbf{b} \end{bmatrix} \\ & + \begin{bmatrix} (\mathbf{I} - \mathbf{A}\tilde{\tau}_3) \mathbf{G} \Delta p^{n+\frac{1}{2}} \\ (\mathbf{L}_1 - \mathbf{D}\tilde{\tau}_3 \mathbf{G}) \Delta p^{n+\frac{1}{2}} + (\mathbf{L}_2 - \mathbf{D}\tilde{\tau}_2 \mathbf{G}) p^{n-\frac{1}{2}} \end{bmatrix}. \end{aligned} \quad (3.747)$$

The error appearing in the momentum equation is due to splitting and generally can be repaired by non-linear iteration, although ideally single iteration methods are desired (as shown).

Again it is emphasized that for approximate projection methods, $\mathbf{L}_2 \neq \mathbf{D}\tilde{\tau}_2 \mathbf{G}$, whereas for exact projection methods, which are usually based on staggering velocity and pressure, $\mathbf{L}_2 = \mathbf{D}\tilde{\tau}_2 \mathbf{G}$ and there is no stabilization error (as there is no need to provide stabilization). Frequently, the stabilization terms within (3.740) are included in a modified provisional velocity (cf. [226]), i.e., $\tilde{\mathbf{u}} = \hat{\mathbf{u}} + \tilde{\tau}_2 \mathbf{G} p^{n-\frac{1}{2}}$, that can often hide the true role of stabilization.

A similar analysis for *pressure free* projection methods (cf. [227]) can be carried out, in which case the equations solved are given by,

$$\mathbf{A}\hat{\mathbf{u}} = \mathbf{f} - \mathbf{A}\mathbf{u}^n, \quad (3.748)$$

$$-\mathbf{L}_1 \Delta \phi^{n+1} = -\mathbf{D}\hat{\mathbf{u}} + \mathbf{L}_1 \phi^n + b, \quad (3.749)$$

$$\mathbf{u}^{n+1} = \hat{\mathbf{u}} - \tilde{\tau}_1 \mathbf{G} \phi^{n+1}, \quad (3.750)$$

with errors,

$$\begin{aligned} & \begin{bmatrix} \mathbf{AG} \\ \mathbf{D0} \end{bmatrix} \begin{bmatrix} \mathbf{u}^{n+1} \\ p^{n+1/2} \end{bmatrix} = \\ & \begin{bmatrix} \mathbf{f} \\ \mathbf{b} \end{bmatrix} + \begin{bmatrix} -\mathbf{A}\tilde{\tau}_3 \mathbf{G} \phi^{n+1} + \mathbf{G} p^{n+\frac{1}{2}} \\ (\mathbf{L}_1 - \mathbf{D}\tilde{\tau}_1 \mathbf{G}) \phi^{n+1} \end{bmatrix}. \end{aligned} \quad (3.751)$$

The error term in the continuity equation is retained to emphasize that this algorithm can be considered in the context of an approximate projection method. Assuming that the Laplacian and gradient operators commute, it is necessary to compute $p^{n+1/2} = \mathbf{A}\tau_3 \phi^{n+1}$ to obtain the second-order pressure field, while the relationship $p^{n+\frac{1}{2}} = \phi^{n+1}$ will result in a first-order pressure field with splitting error $(\mathbf{I} - \mathbf{A}\tau_3) \mathbf{G} p^{n+\frac{1}{2}}$ ([228]).

Although the above set of algorithms have been written in terms of a two step scheme, i.e., predict $\hat{\mathbf{u}}$ and correct $\hat{\mathbf{u}}$ by the appropriately scaled scalar gradient, non-linear iterations can also be taken.

In this case, the ϕ^{n+1} and \mathbf{u}^{n+1} state are replaced with the $k + 1$ state, whereas the $n + \frac{1}{2}$ pressure state is replaced by the $k + \frac{1}{2}$ state. For the residual form, the n^{th} state is replaced with the current iterate, k^{th} state. At convergence within the time step, $\phi^{n+1} = \phi^{k+1}$, $\mathbf{u}^{n+1} = \mathbf{u}^{k+1}$, and $p^{n+\frac{1}{2}} = p^{k+\frac{1}{2}}$.

CVFEM operators

SIERRA/Fuego uses the finite volume technique known as the control volume finite element method of [153]. Control volumes (the mesh dual) are constructed about the nodes, as shown in Fig. [Numerics](#). Each element contains a set of subfaces that define control-volume surfaces. The subfaces consist of line segments (2-D) or surfaces (3-D). The 2-D segments are connected between the element centroid and the edge centroids. The 3-D surfaces are connected between the element centroid, the element face centroids, and the edge centroids. Integration points also exist within the subcontrol volume centroids. Such integration points are used for volume integrals such as source terms, the mass matrix, and, if chosen, gradients.

Defining ϕ_K to be the value of ϕ at node K , then the variation of ϕ within an element that contains the point location $\{\mathbf{x}\}$ is given by

$$\phi(\mathbf{x}) = \sum_{K \in \mathcal{N}} N_K(\mathbf{x}) \phi_K, \quad (3.752)$$

where $N_K(\mathbf{x})$ is the shape function associated with node K at position \mathbf{x} , and \mathcal{N} is the set of all nodes that defines the element. For the CVFEM, either trilinear (3-D) or bilinear (2-D) shape functions are used. Currently, Fuego supports heterogeneous element topologies consisting of hex, tet, pyramid, and wedges.

The discrete nodal gradient operator for direction i can be written as a surface integral on control volume L ,

$$\mathbf{G}\phi = (G\phi)_{Li} = \int_{\Gamma_L} \phi(\mathbf{x}) dn_i \approx \sum_{\alpha \in \mathcal{B}_L} \left(\sum_{K \in \mathcal{N}} N_K(\mathbf{x}_\alpha) \phi_K \right) n_i(\mathbf{x}_\alpha) \Delta A_\alpha, \quad (3.753)$$

where \mathcal{B}_L is the set of surface integration points for control volume L . Similarly, the discrete divergence operator at node L acting on vector u_i is

$$\mathbf{D}\mathbf{u} = (Du_i)_L = \int_{\Gamma_L} \rho(\mathbf{x}) u_i(\mathbf{x}) dn_i \approx \sum_{\alpha \in \mathcal{B}_L} \rho(\mathbf{x}_\alpha) \left(\sum_{K \in \mathcal{N}} N_K(\mathbf{x}_\alpha) u_{Ki} \right) n_i(\mathbf{x}_\alpha) \Delta A_\alpha, \quad (3.754)$$

and the Laplacian operator that includes spatially varying timescale, τ , is

$$\mathbf{L}_\tau \phi = (L_\tau \phi)_L = \int_{\Gamma_L} \tau(\mathbf{x}) \frac{\partial \phi}{\partial x_j} dn_j \approx \sum_{\alpha \in \mathcal{B}_L} \tau(\mathbf{x}_\alpha) \left(\sum_{K \in \mathcal{N}} \frac{\partial N_K(\mathbf{x}_\alpha)}{\partial x_j} \phi_K \right) n_j(\mathbf{x}_\alpha) \Delta A_\alpha. \quad (3.755)$$

Note that an alternative to the gradient operator given in (3.753), which is provided via the CVFEM is

$$\mathbf{G}\phi = (G\phi)_{Li} = \int_{\Gamma_L} \frac{\partial \phi}{\partial x_i} dV \approx \sum_{\alpha' \in \mathcal{B}_L} \left(\sum_{K \in \mathcal{N}} \frac{\partial N_K(\mathbf{x}_{\alpha'})}{\partial x_i} \phi_K \right) dV_{\alpha'}, \quad (3.756)$$

where \mathcal{B}_L is now the set of all subcontrol volume integration points for control volume L (for clarity, α' denotes the subcontrol volume integration point location).

The general term $\mathbf{D}\tilde{\tau}\mathbf{G}\phi$ deserves a special note in the case of variable density flows. Specifically, the interpolation is currently provided by the following equation:

$$\mathbf{D}\tilde{\tau}_i\mathbf{G}\phi = \sum_{\alpha \in \mathcal{B}_L} \rho(\mathbf{x}_\alpha) \frac{\tilde{\tau}_i(\mathbf{x}_\alpha)}{\rho(\mathbf{x}_\alpha)} \left(\sum_{K \in \mathcal{N}} N_K(\mathbf{x}_\alpha) \frac{G_{Ki}}{V_K} \right) n_i(\mathbf{x}_\alpha) \Delta A_\alpha, \quad (3.757)$$

$$= \sum_{\alpha \in \mathcal{B}_L} \tilde{\tau}_i(\mathbf{x}_\alpha) \left(\sum_{K \in \mathcal{N}} N_K(\mathbf{x}_\alpha) \frac{G_{Ki}}{V_K} \right) n_i(\mathbf{x}_\alpha) \Delta A_\alpha. \quad (3.758)$$

Smoothing algorithms defined

Now that the smoothing and splitting errors have been formally defined, it is useful to consider three projection algorithms that have been implemented and verified within SIERRA/Fuego in the context of the classic two equation k - ϵ model, with steady method of manufactured solutions (MMS) (cf. [229]).

Fourth-order smoothing with characteristic or time step scaling

In this algorithm, the projection time scales are defined by either

$$\tau = \tau_1 = \tau_2 = \tau_3 = \tau_{char}, \quad (3.759)$$

or

$$\tau = \tau_1 = \tau_2 = \tau_3 = \mathbf{I}\Delta t. \quad (3.760)$$

Here, characteristic scaling, τ_{char} , is a diagonal matrix that represents a time scale based on convection and diffusion contributions, while for time step scaling, the time scale is based on the local time step. The characteristic scaling very closely follows the standard finite element method stabilization parameter.

The smoothing and splitting errors are now given by

$$\begin{bmatrix} \mathbf{A} & \mathbf{G} \\ \mathbf{D} & \mathbf{0} \end{bmatrix} \begin{bmatrix} \mathbf{u}^{n+1} \\ p^{n+1/2} \end{bmatrix} = \begin{bmatrix} \mathbf{f} \\ \mathbf{b} \end{bmatrix} + \begin{bmatrix} (\mathbf{I} - \mathbf{A}\tilde{\tau})\mathbf{G}(p^{n+\frac{1}{2}} - p^{n-\frac{1}{2}}) \\ (\mathbf{L}_\tau - \mathbf{D}\tilde{\tau}\mathbf{G})p^{n+\frac{1}{2}} \end{bmatrix}. \quad (3.761)$$

Of particular interest to this research is the role of the stabilization term, $(\mathbf{L}_\tau - \mathbf{D}\tilde{\tau}\mathbf{G})p^{n+\frac{1}{2}}$, on formal time accuracy when $\tau = \mathbf{I}\Delta t$ (a scheme that has been shown to display more appealing stability characteristics). Clearly, a scheme that uses explicit pressure stabilization with time step scaling is first-order accurate. Expanding this stabilization term shows the fourth-order pressure derivative scaled by a length scale cubed. Therefore, by refining the time step *and* mesh, one might be able to show a second-order accuracy for sufficiently resolved meshes.

In practice, the stabilization terms are carried within the mass flow rate that forms part of the right-hand side of the Pressure Poisson Equation solve and the convection term for the transport of any scalar field. The mass flow rate is defined as

$$\dot{m}^k = \left(\bar{\rho}\hat{\mathbf{u}} + \frac{\overline{\tau\mathbf{G}p^{n-\frac{1}{2}}}}{V} - \bar{\tau}\nabla^h p^{n+\frac{1}{2}} \right) d\mathbf{A}, \quad (3.762)$$

where the introduction of the over bar is noted to represent interpolation of a nodal field to an integration point. Note that in the bulk of the collocated unstructured finite volume literature, the form of the mass flow rate defines the stabilization (the difference between the nodal gradient operator \mathbf{G} and the interior element operator ∇^h). Above we note the independent interpolation of the density and velocity rather than $\bar{\rho}\hat{\mathbf{u}}$, as is done in Stanford's ASC Alliance code CDP. It does seem that the full interpolation of $\bar{\rho}\hat{\mathbf{u}}$ may be more consistent, although the effect of this algorithmic detail has not been explored.

Stabilized smoothing

The stabilized projection algorithm is based on the work of [219], that was derived from the monolithic scheme of [218]. In this algorithm, the projection time scales are defined as

$$\tau_1 = \Delta t \mathbf{I} + \tau_{char}. \quad (3.763)$$

$$\tau_2 = \tau_{char}. \quad (3.764)$$

$$\tau_3 = \tau_{char}. \quad (3.765)$$

With the above definitions, the smoothing and splitting errors are now defined as

$$\begin{bmatrix} \mathbf{A} & \mathbf{G} \\ \mathbf{D} & \mathbf{0} \end{bmatrix} \begin{bmatrix} \mathbf{u}^{n+1} \\ p^{n+\frac{1}{2}} \end{bmatrix} = \begin{bmatrix} \mathbf{f} \\ \mathbf{b} \end{bmatrix} + \begin{bmatrix} (\mathbf{I} - \mathbf{A}\tilde{\tau}_{char})\mathbf{G}(p^{n+\frac{1}{2}} - p^{n-\frac{1}{2}}) \\ (\mathbf{L}_{\tau_{char}} - \mathbf{D}\tilde{\tau}_{char}\mathbf{G})p^{n+\frac{1}{2}} + \Delta t \mathbf{L}(p^{n+\frac{1}{2}} - p^{n-\frac{1}{2}}) \end{bmatrix}. \quad (3.766)$$

The mass flow rate now includes an additional stabilization factor and is now defined as

$$\dot{m}^k = \left(\bar{\rho}\hat{\mathbf{u}} + \frac{\overline{\tau\mathbf{G}p^{n-\frac{1}{2}}}}{V} - \bar{\tau}\nabla^h p^{n+\frac{1}{2}} - \Delta t \mathbf{L}\Delta p^{n+\frac{1}{2}} \right) d\mathbf{A}. \quad (3.767)$$

Note that at full convergence, the stabilized scheme reduces to the fourth-order characteristic scaling algorithm.

Second-order smoothing with characteristic or time step scaling

In fact, the scaled nodal gradient need not be included in the mass flow rate equation, e.g.,

$$\dot{m}^k = \left(\bar{\rho} \bar{\mathbf{u}} - \bar{\tau} \nabla^h p^{n+\frac{1}{2}} \right) d\mathbf{A}. \quad (3.768)$$

This is equivalent to neglecting the $\tilde{\tau}_2 \mathbf{G} p^{n-\frac{1}{2}}$ term in (3.740), or by defining $\tilde{\mathbf{u}} = \hat{\mathbf{u}}$.

The smoothing for this algorithm is provided by the local Laplacian operator. The smoothing and splitting errors for this method are now given by

$$\begin{bmatrix} \mathbf{A} & \mathbf{G} \\ \mathbf{D} & \mathbf{0} \end{bmatrix} \begin{bmatrix} \mathbf{u}^{n+1} \\ p^{n+1/2} \end{bmatrix} = \begin{bmatrix} \mathbf{f} \\ \mathbf{b} \end{bmatrix} + \begin{bmatrix} (\mathbf{I} - \mathbf{A} \tilde{\tau}) \mathbf{G} (p^{n+\frac{1}{2}} - p^{n-\frac{1}{2}}) \\ (\mathbf{L}_\tau - \mathbf{D} \tilde{\tau} \mathbf{G}) \Delta p^{n+\frac{1}{2}} + \mathbf{L}_\tau p^{n-\frac{1}{2}} \end{bmatrix}. \quad (3.769)$$

Zeroth-order smoothing with time step or characteristic scaling

Certainly, the pressure smoothing can be removed, i.e., $\tau_2 = 0$, that leads to the following set of errors,

$$\begin{bmatrix} \mathbf{A} & \mathbf{G} \\ \mathbf{D} & \mathbf{0} \end{bmatrix} \begin{bmatrix} \mathbf{u}^{n+1} \\ p^{n+\frac{1}{2}} \end{bmatrix} = \begin{bmatrix} \mathbf{f} \\ \mathbf{b} \end{bmatrix} + \begin{bmatrix} (\mathbf{I} - \mathbf{A} \tilde{\tau}) \mathbf{G} (p^{n+\frac{1}{2}} - p^{n-\frac{1}{2}}) \\ (\mathbf{L}_\tau - \mathbf{D} \tilde{\tau} \mathbf{G}) (p^{n+\frac{1}{2}} - p^{n-\frac{1}{2}}) \end{bmatrix}. \quad (3.770)$$

where τ is either the characteristic scale, τ_{char} , or the simulation time step, $\mathbf{I} \Delta t$ (with $\tau_1 = \tau_3$).

Although the converged error is zero, this lack of smoothing can lead to a decoupled pressure field in certain flows.

Here, the mass flow rate reduces to a simple interpolation of nodal velocities within the element

$$\dot{m}^k = \left(\bar{\rho} \bar{\mathbf{u}} - \bar{\tau} \nabla^h \Delta p^{n+\frac{1}{2}} \right) d\mathbf{A}. \quad (3.771)$$

The unsmoothed algorithm is very similar to the staggered formulation of SIMPLE, (cf. [131]), with $\tau = A_p^{-1}$ (the inverse of the diagonal matrix from operator $\{\mathbf{b} \mathbf{f} \mathbf{A}\}$). However, by design, the staggered mesh arrangement holds the property that $(\mathbf{L}_\tau - \mathbf{D} \tau \mathbf{G}) = 0$. In this method, no stabilization is added as none is required.

Time integration scheme

At present, three time integration schemes are supported in the code base that include: 1) first-order backward Euler, 2) second-order BDF2 and, 3), the Crank-Nicholson method described in [230].

The general two- and three-state scheme time derivatives are simply written as:

$$\int \frac{\partial \rho \phi}{\partial t} dV = \int \frac{(\gamma_1 \rho^{n+1} \phi^{n+1} + \gamma_2 \rho^n \phi^n + \gamma_3 \rho^{n-1} \phi^{n-1})}{\Delta t} dV \quad (3.772)$$

where γ_i represent the appropriate factors for either Backward Euler or a three-point BDF2 scheme. The above time derivative is either nodally lumped or evaluated at the subcontrol volume quadrature points. For a two-state backward Euler scheme, γ_1 is unity while γ_2 is negative unity. For a given variable time step, the BDF2 factors are,

$$\tau = \Delta t^N / \Delta t^{N-1}, \quad (3.773)$$

$$\gamma_1 = \frac{(1 + 2\tau)}{(1 + \tau)}, \quad (3.774)$$

$$\gamma_2 = (1 + \tau), \quad (3.775)$$

$$\gamma_3 = \frac{\tau^2}{(1 + \tau)}. \quad (3.776)$$

For a fixed time step, the γ -factors reduce to the canonical $(\frac{3}{2}, -2, \frac{1}{2})$ set.

In the Crank-Nicholson implementation, the generalized method is written as

$$\frac{\partial \phi^{n+1}}{\partial t} = \eta \frac{\phi^{n+1} - \phi^n}{\Delta t} + (1 - \eta) \frac{\partial \phi^n}{\partial t}, \quad (3.777)$$

where η is a blending coefficient between 1 and 2. Values of η of unity result in first-order backward Euler, while values of 2 result in second order Crank-Nicholson, i.e.,

$$\frac{\partial \phi^{n+1}}{\partial t} = 2 \frac{(\phi^{n+1} - \phi^n)}{\Delta t} - \frac{\partial \phi^n}{\partial t}. \quad (3.778)$$

A linearization is given by

$$\frac{\partial \phi^{n+1}}{\partial t} = 2 \frac{(\phi^k - \phi^n)}{\Delta t} - \frac{\partial \phi^n}{\partial t}, \quad (3.779)$$

where the old time derivative is computed based on the old solution of the partial differential equation of interest. The above algorithm is especially useful in that it avoids the need to evaluate complex right-hand side source terms at the $n+1$ and n state, e.g., simulations that include the need to compute turbulence production, reaction rate terms, etc.

Variable density

In the case of variable density, the same set of options exists in the code. Specifically, the time derivative is written as (for backward Euler or BDF2),

$$\int \frac{\partial \rho}{\partial t} dV = \int \frac{(\gamma_1 \rho^{n+1} + \gamma_2 \rho^n + \gamma_3 \rho^{n-1})}{\Delta t} dV \quad (3.780)$$

For the Crank/Nicolson scheme, the full time term is

$$\frac{\partial \rho \phi^{n+1}}{\partial t} = \eta \frac{\rho^{n+1} \phi^{n+1} - \rho^n \phi^n}{\Delta t} + (1 - \eta) \frac{\partial \rho \phi^n}{\partial t}, \quad (3.781)$$

where it is noted that the full time derivative at n^{th} state is saved. The linearization is given by

$$\frac{\partial \rho \phi^{n+1}}{\partial t} = \eta \frac{\rho^k \phi^k - \rho^n \phi^n}{\Delta t} + (1 - \eta) \frac{\partial \rho \phi^n}{\partial t}. \quad (3.782)$$

In practice, the usage of the above formula for a second-order density derivative has proven unstable. As such, η is set to unity.

Shifted density iteration

A modified iteration originating from Shunn, Ham and Moin:cite:shunn:2012 is available. The iteration procedure proceeds in words as

- Copy forward density state as $\rho_0 = \rho^n$ as a provisional density for the step
- Solve all scalar transport equations using ρ_0
- Solve momentum, with ρ_0
- Compute ρ_1 from the equation of state using the result of the scalar transport equations
- Compute the density-time derivative using ρ_1
- Solve the pressure Poisson equation using the newly computed density-time derivative and the provisional density and velocity
- Update the mass flux using the newly computed pressure consistent with the pressure Poisson equation
- Project momentum to the new $\rho_1 u_1$ state consistent with the pressure Poisson equation.

For a simplified case of a single scalar Z determining density through an equation of state ρ_R , predict a solution as $Z_0 = Z^n$, $\rho_0 = \rho^n$, $u_0 = u^n$, $p_0 = p^n$, then iterate

$$\int \frac{\gamma_1}{\Delta t} \rho_0 Z_1 dV + \int (\dot{m}_0 Z_1 - \text{Sc}^{-1} \mu \nabla_h Z_1 \cdot A) dS = - \int \frac{1}{\Delta t} (\gamma_2 \rho^n Z^n + \gamma_3 \rho^{n-1} Z^{n-1}) dV \quad (3.783)$$

$$\int \frac{\gamma_1}{\Delta t} \rho_0 u^\star dV + \int (\dot{m}_0 u^\star - 2\mu \nabla_{s,h} u^\star \cdot A) dS = - \int \left[\frac{1}{\Delta t} (\gamma_2 \rho^n u^n + \gamma_3 \rho^{n-1} u^{n-1}) + G p_0 \right] dV \quad (3.784)$$

$$\rho_1 = \rho_R(Z_1), (\partial_t \rho)_1 = \frac{1}{\Delta t} (\gamma_1 \rho_1 + \gamma_2 \rho^n + \gamma_3 \rho^{n-1}) \quad (3.785)$$

$$\int \tau \nabla_h p_1 dS = - \int (\partial_t \rho)_1 dV + \int (\rho_0 u^\star + \tau G p_0) \cdot A dS \quad (3.786)$$

$$\dot{m}_1|_{ip} = [(\rho_0 u^\star + \tau G p_0)|_{ip} - \tau|_{ip} \nabla_h p_1|_{ip}] \cdot A \quad (3.787)$$

$$\int G p_1 dV = \int p_1 A dS \quad (3.788)$$

$$u_1 = \frac{1}{\rho_1} [\rho_0 u^\star - \tau (G p_1 - G p_0)], \quad (3.789)$$

where ∇_s is the symmetric gradient. The procedure helps the iteration procedure to converge more robustly with large density time-derivatives. Numerical evidence:cite:shunn:2012 indicates that recovering second-order accuracy with the iteration process depends strongly on the nature of the state equation, with an MMS at Courant number unity requiring 5 outer-iterations for a density ratio of 5 but 20 with a density ratio of 20.

Volume-averaged properties



Beta Capability

Volume-averaged properties is a beta capability, and will substantially increase the cost of property evaluation.

Different options for computing volume-averaged properties are available. For some property ψ depending on a state parameterization ϕ , we have

$$\overline{\psi}_i = \frac{1}{\text{vol}(\Omega_i)} \int_{\Omega_i} \psi(\phi_h) dV, \quad (3.790)$$

where ϕ_h is the state parameterization represented in the nodal finite element basis. Using a nodal quadrature for the integral is the default and is equivalent to direct nodal evaluation of the properties. Alternatively, the quadrature of the consistent mass matrix can be used. A more accurate quadrature for the properties is useful to reduce aliasing errors arising due to the nonlinear nature of the state relationship. Additionally properties maybe smoothed directly with n -passes of a nodal filter,

$$\hat{\psi}^n = \left(M_L^{-1} M_c \right)^n \psi. \quad (3.791)$$

where M_L and M_c are the lumped and consistent mass matrices respectively, described in Sec. *Discrete system of equations*. This is equivalent to a box-filter on the dual mesh in CVFEM and helps reduce spurious oscillations in the property evaluation, but also introduces an additional ($O(h^2)$) error with each pass.

Discrete system of equations

The full approximate pressure projection scheme for non-uniform density is now written as

$$\eta M_L^k \Delta \hat{u}_i + C_L(\dot{m}^k) \Delta \hat{u}_i - T_{Lj} \Delta \hat{u}_i = -r_i, \quad (3.792)$$

$$-L_{\tau_1 L} \Delta p^{n+\frac{1}{2}} = -D_L(\hat{u}_i) - L_{\tau_1} p^k + (L_2 - D \tilde{\tau}_2 G)_L p^k + b, \quad (3.793)$$

$$u_{Li}^{n+1} = \hat{u}_{Li} - \tilde{\tau} G_{Li} \Delta p^{n+\frac{1}{2}}. \quad (3.794)$$

The variable $-r_i$ is the residual that includes body source terms, pressure gradient, the non-symmetric part of the viscous stress term, $T_{Li}^{ns}u_j^k$, parts of the time term and the left-hand side set of operators acting on the u_i^k state,

$$-ri = -\eta M_L^k u_i^k - C_L(\dot{m}^k)u_i^k + T_{Lj}\Delta u_i^k + T_{Li}^{ns}u_j^k + S_{Li} - (1 - \eta)M_L(\rho \dot{u}_i^n) - G_{Li}p^{n-\frac{1}{2}}. \quad (3.795)$$

The mass matrix, $M_L^k \Delta \hat{u}_i$, is defined by

$$M_L^k \Delta \hat{u}_i = \sum_{\alpha' \in \mathcal{B}_L} \left(\sum_{K \in \mathcal{N}} N_K(\mathbf{x}_{\alpha'}) \frac{\rho_K^k}{\Delta t} \right) \left(\sum_{K \in \mathcal{N}} N_K(\mathbf{x}_{\alpha'}) \Delta \hat{u}_{Ki} \right) dV_{\alpha'}. \quad (3.796)$$

The shape function above, $N_K(\mathbf{x}_{\alpha'})$, is frequently evaluated at \mathbf{x}_N , the coordinates of the vertex associated with the transport equation, i.e., the case where a lumped mass matrix is used.

For simplicity, the central difference operator is provided in $C_{Li} \Delta \hat{u}_i$ as

$$C_L \Delta \hat{u}_i = \sum_{\alpha \in \mathcal{B}_L} m_\alpha^k \left(\sum_{K \in \mathcal{N}} N_K(\mathbf{x}_\alpha) \Delta \hat{u}_{Ki} \right). \quad (3.797)$$

In the preceding equation, the mass flow rate has been linearized within the iteration step and may or may not include the explicit stabilization terms. Moreover, the shape function operator, $N_K(\mathbf{x}_\alpha)$, may be evaluated at the edge midpoints to retain the skew symmetric aspect of the operator C_L . By default, this term is evaluated at the subcontrol surface integration points, which retains the CVFEM canonical 27-point stencil.

The symmetric part of the stress tensor is given by

$$T_{Lj} \hat{u}_i = \sum_{\alpha \in \mathcal{B}_L} \left(\sum_{K \in \mathcal{N}} N_K(\mathbf{x}_\alpha) \mu_K \right) \left(\sum_{K \in \mathcal{N}} \frac{dN_K(\mathbf{x}_\alpha)}{dx_j} \hat{u}_{Ki} \right) n_j(\mathbf{x}_\alpha) \Delta A_\alpha, \quad (3.798)$$

while the non-symmetric stress tensor is given by

$$T_{Li}^{ns} u_j^k = \sum_{\alpha \in \mathcal{B}_L} \left(\sum_{K \in \mathcal{N}} N_K(\mathbf{x}_\alpha) \mu_K \right) \left(\sum_{K \in \mathcal{N}} \frac{dN_K(\mathbf{x}_\alpha)}{dx_i} u_{Kj}^k \right) n_j(\mathbf{x}_\alpha) \Delta A_\alpha - \frac{2}{3} \sum_{\alpha \in \mathcal{B}_L} \left(\sum_{K \in \mathcal{N}} N_K(\mathbf{x}_\alpha) \mu_K \right) \left(\sum_{K \in \mathcal{N}} \frac{dN_K(\mathbf{x}_\alpha)}{dx_p} u_{Kp}^k \right) \quad (3.799)$$

Note that the nodal pressure gradient at node L for control volume L for direction i is defined by (3.753). The operator, S_{Li} , contains the gravitational term as well as the [potentially] subtracted out hydrostatic term,

$$S_{Li} = \sum_{\alpha' \in \mathcal{B}_L} \left(\sum_{K \in \mathcal{N}} N_K(\mathbf{x}_{\alpha'}) (\rho_K^k - \rho^{ref}) \right) g_i dV_{\alpha'}. \quad (3.800)$$

The old time term contribution, $M_L(\rho \dot{u}_i^n)$, is defined by

$$M_L(\rho \dot{u}_i^n) = \sum_{\alpha' \in \mathcal{B}_L} \left(\sum_{K \in \mathcal{N}} N_K(\mathbf{x}_{\alpha'}) \rho_K \dot{u}_{Ki}^n \right) dV_{\alpha'}. \quad (3.801)$$

Again, $\alpha' \in \mathcal{B}_L$ is the set of all subcontrol volume integration points for control volume L , $\alpha' \in \mathcal{B}_L$ is the set of all subcontrol surface integration points for control volume L , and $K \in \mathcal{N}$ is the set of all nodes within the element.

Predictor

In general, there are a number of predictors that are supported. The easiest predictor is a simple predictor in which the old value is mapped into the current iterate. Predictors that incorporate old time derivatives include the forward Euler and Adams-Bashforth methods, e.g.,

$$\phi^{k+1} = \phi^n, \quad (3.802)$$

$$= \phi^n + \Delta t \dot{\phi}^n, \quad (3.803)$$

$$= \phi^n + \frac{\Delta t^n}{2} \left(\left(2 + \frac{\Delta t^n}{\Delta t^{n-1}} \right) \dot{\phi}^n - \frac{\Delta t^n}{\Delta t^{n-1}} \dot{\phi}^{n-1} \right). \quad (3.804)$$

Upwind Interpolation for Convection

We currently support several upwind interpolations for convection. The upwind methods are blended with a centered scheme that becomes dominant below a specified cell-Peclet number.

First Order Upwind

The first scheme is a simple first-order scheme that considers the two nodes adjacent to a control volume face and extrapolates from the node in the upwind direction.

$$\dot{m} \bar{\phi}_{upw} = \frac{1}{2} (\dot{m} + |\dot{m}|) \phi_L + \frac{1}{2} (\dot{m} - |\dot{m}|) \phi_R \quad (3.805)$$

The convention is that flow leaves the control volume to the left (L) and enters the control volume to the right (R). If the mass flow rate at the face is negative in value, then the node to the right will be selected.

Blending Function

The user specified upwind factor controls the blending between the pure upwind operator and a blended user-chosen upwind/central operator.

$$\dot{m} \bar{\phi} = \eta \dot{m} \bar{\phi}_{upw} + (1 - \eta) \left(\chi \dot{m} \bar{\phi}_{upw^{sp}} + (1 - \chi) \dot{m} \bar{\phi}_{cen} \right), \quad (3.806)$$

where η is the user specified first order upwind factor and $\bar{\phi}_{upw^{sp}}$ represents the user specified upwind operator, e.g., MUSCL, modified skew upwind, and even pure upwind.

The centered average of ϕ is computed from the shape functions, so it is based on all nodes in an element. The shape functions are evaluated at the sub-face centroid. The cell-Peclet number, $Pe_{\Delta x}$, is used in the blending function (see [Figure 3.29](#))

$$\chi = \frac{(\zeta Pe_{\Delta x})^2}{5 + (\zeta Pe_{\Delta x})^2}. \quad (3.807)$$

The hybrid upwind factor, ζ , allows one to modify the functional blending function; values of unity result in the normal blending function response in Figure 3.29; values of zero yield a pure central operator, i.e., blending function = 0.0; values $\gg 1$ result in a blending function value of unity, i.e., pure upwind. The constant A is implemented as above with a value of 5. This value can not be changed via the input file.

The cell-Peclet number is computed for each sub-face in the element from the two adjacent left (L) and right (R) nodes.

$$\text{Pe}_{\Delta x} = \frac{\frac{1}{2} (u_{R,i} + u_{L,i}) (x_{R,i} - x_{L,i})}{\nu} \quad (3.808)$$

A dot-product is implied by repeated indices.

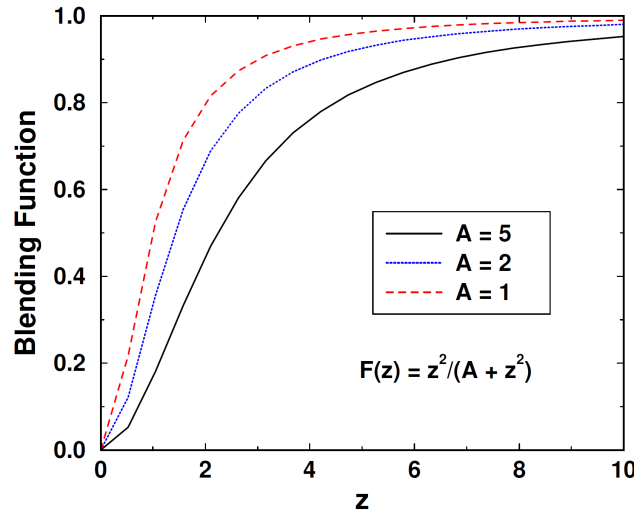


Fig. 3.29: Cell-Peclet number blending function.

Modified Linear Profile Skew Upwind

Modified linear profile skew upwinding is a simplification to the skew upwinding approach in the FIELDS scheme [151, 152]. We omit the physical advection correction terms. Integration point values at control volume subfaces are interpolated from upwind intersection points on the element face. In the original skew upwind scheme, the intersection point could either be interior subface or element faces. The interpolation coefficients were computed by inverting a matrix relation between integration point values and nodal values. The linear profile skew upwinding does not use interior subface intersections – only element face intersections. The modified scheme throws out nodes on an element face that are downwind of an interior subface as shown in Figure 3.30.

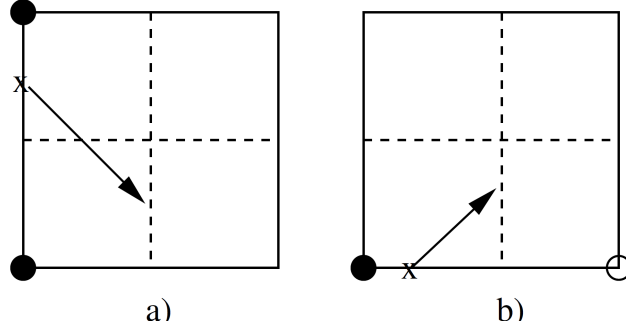


Fig. 3.30: Linear profile skew upwind scheme: a) all nodes on the intersected element face are upwind of the surface, b) omit nodes on intersected element face that are downwind of the surface.

MUSCL

The MUSCL approach (see Chap. 21 of Hirsch [231]) for higher order upwinding is adapted to unstructured meshes. The upwind interpolation is constructed along each edge of an element. The interpolation makes use of the two end nodes of the edge and the centered gradient constructed at the two end nodes. The MUSCL approach constructs an interpolation in one dimension from four (or more) uniformly distributed nodal values. The two edge nodes are ϕ_i and ϕ_{i+1} . The two other nodal values, ϕ_{i-1} and ϕ_{i+2} , are interpolated from the unstructured mesh using the nodal gradient information.

The MUSCL scheme constructs left and right interpolants at the subface of the control volume. Without the limiter functions, the interpolation is

$$\phi_{i+1/2}^L = \phi_i + \frac{1}{4} [(1 - \kappa) (\phi_i - \phi_{i-1}) + (1 + \kappa) (\phi_{i+1} - \phi_i)], \quad (3.809)$$

$$\phi_{i+1/2}^R = \phi_{i+1} - \frac{1}{4} [(1 + \kappa) (\phi_{i+1} - \phi_i) + (1 - \kappa) (\phi_{i+2} - \phi_{i+1})], \quad (3.810)$$

where the $(i + 1/2)$ location is between node i and node $i + 1$. On a uniform mesh, $\kappa = 1/3$ gives a third-order scheme. A second-order upwind scheme is recovered with $\kappa = -1$ and a centered scheme is recovered with $\kappa = 1$.

Limiter functions are introduced to prevent numerical oscillations from occurring.

$$\tilde{\phi}_{i+1/2}^L = \phi_i + \frac{1}{4} \left[(1 - \kappa) \Phi \left(\frac{1}{r_L} \right) (\phi_i - \phi_{i-1}) + (1 + \kappa) \Phi(r_L) (\phi_{i+1} - \phi_i) \right], \quad (3.811)$$

$$\tilde{\phi}_{i+1/2}^R = \phi_{i+1} - \frac{1}{4} \left[(1 + \kappa) \Phi(r_R) (\phi_{i+1} - \phi_i) + (1 - \kappa) \Phi \left(\frac{1}{r_R} \right) (\phi_{i+2} - \phi_{i+1}) \right], \quad (3.812)$$

where

$$r^L = \frac{\phi_i - \phi_{i-1}}{\phi_{i+1} - \phi_i}, \quad (3.813)$$

$$r^R = \frac{\phi_{i+2} - \phi_{i+1}}{\phi_{i+1} - \phi_i}. \quad (3.814)$$

The limiters are selected to be symmetric such that

$$\Phi(r) = r\Phi\left(\frac{1}{r}\right). \quad (3.815)$$

The limited interpolation functions are

$$\tilde{\phi}_{i+1/2}^L = \phi_i + \frac{1}{2}\Phi(r^L)(\phi_{i+1} - \phi_i), \quad (3.816)$$

$$\tilde{\phi}_{i+1/2}^R = \phi_{i+1} - \frac{1}{2}\Phi(r^R)(\phi_{i+1} - \phi_i). \quad (3.817)$$

The interpolation for the points off the element edge is

$$(\phi_i - \phi_{i-1}) = 2\nabla\phi_i\Delta x_{i+1/2} - (\phi_{i+1} - \phi_i), \quad (3.818)$$

$$(\phi_{i+2} - \phi_{i+1}) = 2\nabla\phi_{i+1}\Delta x_{i+1/2} - (\phi_{i+1} - \phi_i), \quad (3.819)$$

where $\Delta x_{i+1/2} = x_{i+1} - x_i$ is the distance vector along the element edge. Symmetric limiter functions are:

$$\text{VanLeer :} \quad \Phi(r) = \frac{r + |r|}{1 + |r|}, \quad (3.820)$$

$$\text{VanAlbada :} \quad \Phi(r) = \frac{r + r^2}{1 + r^2}, \quad (3.821)$$

$$\text{superbee :} \quad \Phi(r) = \max(0, \min(2r, 1), \min(r, 2)). \quad (3.822)$$

Convection at an Inflow and Outflow Boundary

At an open boundary, the first-order and LPS upwind schemes only make use of information on the boundary.

For the MUSCL scheme with the flow leaving the domain at node i , the usual flux limiters are not used. The slopes are compared between $(\phi_i - \phi_{i-1})$ and $(\phi_{i-1} - \phi_{i-2})$. If the slopes are the same sign, the unlimited second order upwinding is used. If the slopes are different, then a local interpolation is used. Estimate the slope $(\phi_{i-1} - \phi_{i-2}) = 2\Delta x\nabla\phi_2 - (\phi_i - \phi_{i-1})$, where $\Delta x_i = x_i - x_{i-1}$ is the distance vector along the element edge. For slopes of the same sign, use a second-order scheme,

$$\tilde{\phi}_i^L = \phi_i + \frac{1}{2}(\nabla\phi_{i-1}\Delta x_i - (\phi_i - \phi_{i-1})), \dots, \quad (3.823)$$

else, use a first-order scheme,

$$\tilde{\phi}_i^L = \phi_i - \frac{1}{2}[(\phi_i - \phi_{i-1})]. \quad (3.824)$$

The boundary is the left (L) side. If the flow enters the domain, then use the local value of ϕ_i .

Nonlinear stabilization operator

The “nonlinear stability operator” (NSO) in Fuego is an artificial viscosity method where the added diffusivity is based on a scaled, pointwise evaluated residual. For a dual volume (Ω_n), associated with a node n , the weak form of the NSO for a scalar variable q is

$$\int_{\partial\Omega_n} \nu(R) (\partial_{x^i} q) g^{ij} dS_j, \text{ where } g^{ij} = \frac{\partial x^i}{\partial \xi^k} \frac{\partial x^j}{\partial \xi^k}. \quad (3.825)$$

where ν depends on the evaluation of a local residual R and the gradient of q as

$$\nu = \sqrt{\frac{R^2}{\frac{\partial q}{\partial x^i} g^{ij} \frac{\partial q}{\partial x^j}}}. \quad (3.826)$$

The local residual can be taken, similar to Shakib:cite:Shakib:1991 but in an incompressible context, as the full residual of the PDE. For a conserved scalar, q , with diffusivity Γ , R would be

$$R = [(\text{Time})_\rho + (\text{Adv.})_{\rho\mathbf{u}} - (\text{Diff.})_{\rho\Gamma}] q, \quad (3.827)$$

with discrete operators representing the individual terms of the advection-diffusion equation. For an equation with a source term, it would also need appear in the local residual calculation. Another possibility for choosing R would be based on the error of performing the chain-rule on the advection operator.

$$R = \tilde{G}_i (\rho u^i q) - [\tilde{I} (\rho u^i) \tilde{G}_i q + (\tilde{I} q) G_i (\rho u^i)] \quad (3.828)$$

where \tilde{G} and \tilde{I} represent interpolation and gradients evaluated at an integration point. Both options are available in Fuego.

The NSO computed from such residuals can add an unnecessarily large amount of dissipation in some cases. For this reason, we limit the NSO coefficient to the upwind value as

$$\nu = \min \left(\nu(R), \frac{1}{10} (\rho u)^i g_{ij} (\rho u)^j \right). \quad (3.829)$$

where $g_{ij} = [g^{ij}]^{-1}$. Additionally, as it's based on the mesh discretization error, the NSO coefficient tends to vary strongly on short length scales. For numerical robustness, we average the NSO viscosity over control volumes, and then interpolate back to the subcontrol surfaces to evaluate the diffusion term; that is,

$$\bar{\nu}_{ip} = \tilde{I} \frac{\overline{\nu \|g^{ij}\|}}{\|g^{ij}\|}. \quad (3.830)$$

This operation effectively smooths the NSO viscosity over a patch of elements. The nonlinear stabilization viscosity is not included at the boundaries.

Variable Density

The discretization of the time derivative requires special attention for variable density flows. The density time-derivative in the continuity equation must be predicted in a continuous manner. The density at the new time level in the convection terms and the transport equation time terms must also be predicted.

The transport equations are solved in conservative form, so density appears in the time derivative. With a segregated solution strategy, the density at the new time level is not available until the transport equations have been solved once. A density predictor is required. A generic time term is written as

$$\frac{\partial \rho \phi}{\partial t} \approx \frac{\rho^{n+1} \phi^{n+1} - \rho^n \phi^n}{\Delta t} \approx \rho^n \frac{\phi^{n+1} - \phi^n}{\Delta t} + \phi^{n+1} \frac{\rho^* - \rho^n}{\Delta t} \quad (3.831)$$

There are two approaches to estimating the new density. The simplest approach is to use the most recent value. The other approach is to use a density predictor. The predicted value of density at the new time level, ρ^* , is computed from the old density and the current density time derivative. Introduce the nodal variable Υ for the discrete density time-derivative such that

$$\Upsilon^* = \frac{\rho^* - \rho^n}{\Delta t} \quad (3.832)$$

$$\rho^* = \rho^n + \Delta t \Upsilon^* \quad (3.833)$$

The density derivative, Υ^* , is always updated at the bottom of the transport equation loop after a new set of temperatures and mass fractions is available. The two approaches are different for the first nonlinear sub-iteration within a time step, but yield equivalent values upon subsequent sub-iterations. The new density is also computed at the bottom of the equation loop. This value is ignored upon subsequent sub-iterations if using the density predictor. But, this new density value will get copied to the old time level when the time step is advanced. It is important to note that this new “old” velocity is not consistent with the density that was used in the old transport equations, but it seems critical to the success of this approach to do so.

For the first nonlinear iteration within a time step, the effect of the density at the new time level is predicted by carrying forward the best approximation of the density time-derivative from the last time step. The continuity equation is implemented as

$$\int \Upsilon^* dV + \int \rho^* \bar{u}_i^{n+1} n_i dS = 0, \quad (3.834)$$

where the density time-derivative is the most recent value and the density in the convection is estimated in the same manner as the transport equations. The density time-derivative, Υ , must be stored as a persistent nodal variable in order to have a good estimate for the continuity equation from step to step.

Open Boundary Conditions

Open boundary conditions are used for boundaries where the flow can go either in or out. The direction of the flow is determined by the local force balance. In this documentation, the open boundary condition is also referred to as the outflow boundary condition. There are two parts to the outflow boundary condition. The first part concerns computing a velocity field that satisfies continuity. The second part concerns selecting the proper convected scalar value depending if the flow is in or out of the domain. Control volume balances are implemented at open boundaries for continuity, momentum, and the other transport equations.

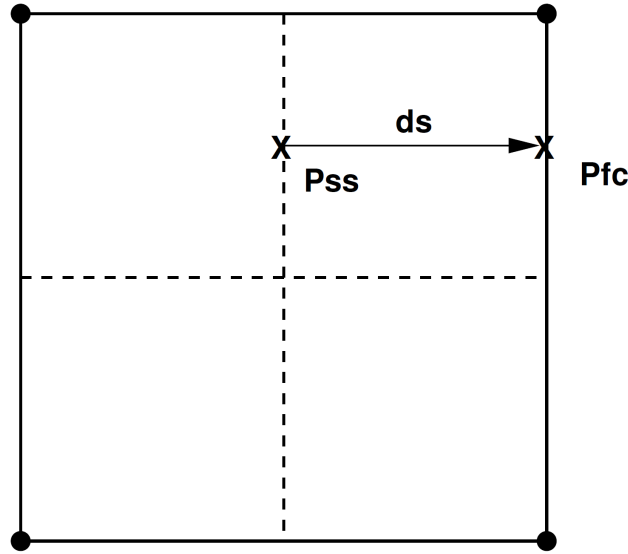


Fig. 3.31: Boundary mass flux integration locations.

A fixed pressure value is specified for the continuity and momentum equations. The nodal values of pressure on the boundary are allowed to float. A mass flux condition is formulated at the boundary in order to drive the boundary pressures towards the specified boundary pressure and to provide a boundary mass flow rate for the other transport equations. The form of the boundary mass flux is similar to the pressure-stabilized interior mass fluxes (see section [Flow Solver](#)). The equation for the mass flux at a boundary face, shown in [Figure 3.31](#), is

$$\dot{m}_{bc} = \rho \bar{u}_i^{n+1} n_i dS \quad (3.835)$$

and the interpolation formula for a single velocity component is

$$\bar{u}^{n+1} = \sum_{bc} N_i U_i^{**} + f \frac{\Delta t}{\rho} \left(\sum_{bc} N_i \sum_j G_{ij} P_j^* - \left(\frac{P_{fc} - P_{ss}^{n+1}}{\Delta s} \right) \frac{\Delta x}{\Delta s} \right) + f \left(\bar{u}^n - \sum_{bc} N_i U_i^n \right) \quad (3.836)$$

The upper case velocities, U_i , are nodal velocities, while the lower case velocity, \bar{u} , is the boundary velocity. The average pressure, P_{ss} , is computed at the opposing subface centroid and evaluated at the new time level, $n + 1$. The boundary pressure, P_{fc} , is evaluated at the boundary subface centroid and is the “specified” pressure. The operator, G_{ij} , is the discrete gradient

operator for node i . In the case of the semi-discrete formulation, the last term is dropped in (3.836) and $f = 1$.

The nodal pressure gradient is required for the momentum balance and the boundary mass flux formulation. The nodal pressure gradient is constructed by a discrete Gauss divergence relation over the control volumes. The pressure at most control volume subfaces is interpolated from the nodes of the parent element, even over inflow, wall, and symmetry boundaries. For outflow boundaries, the specified boundary pressure, P_{fc} , is used.

Nodal velocities on open boundaries are corrected with the projection.

On pressure-specified open boundaries, the flow will sometimes exit and reenter the domain through some sort of entrainment process. The process will look non-physical and is due to the artificially imposed constant pressure. A method of counteracting the reentrance problem is to turn off the convection terms in the momentum equations for control-volume subfaces which have reentrant flow. This condition is optional and can be set on a side-set basis.

If the flow is entrained into the domain, then far-field values must be specified for the scalar variables.

Segregated Solution Procedure

The time integration method is a two-level, backward Euler scheme, requiring data at two time states. The discrete form of the nonlinear equations is

$$\frac{\phi^{n+1} - \phi^n}{\Delta t} = F(\phi^{n+1}, \phi^n). \quad (3.837)$$

Sub-iteration is required within the time step to satisfy the nonlinearities. Over one sub-iteration, the nonlinear equations are solved in a segregated manner. Each segregated equation set is linearized and solved as a linear problem. During the nonlinear iteration process, a temporary variable may be introduced to differentiate the old guess at the state (n) from the new guess at the state (n+1). A temporary variable (*) is introduced to hold the new estimate of the state (n+1). The temporary variable is typically only used in describing the algorithm. Functionally, the (*) variables and (n+1) variables are usually represented by the same array within the code. The only time a temporary variable would be used in the code is if the momentum equations were segregated or if the species diffusion velocities were not pre-computed.

Within the transport equations, the convection terms are linearized by freezing the mass flux (density * velocity * area).

The SIERRA framework provides services to manage the state data between the two time levels. The SIERRA framework services are insufficient because they only swap pointers. The result of the swap is that the estimate of the new solution at time (n+1) uses the solution at (n) instead of (n), which is too far away. After the pointer-swap, the SIERRA/Fuego code additionally copies forward the solution at (n) into the initial guess at (n+1). The array-copy occurs only at the beginning of the sub-iteration process. The SIERRA/Fuego code also manages the updating between (*) and (n+1) for the delta-form of the linear system.

The material properties are evaluated at the top of a nonlinear sub-iteration. Density is a STATE property since it has a time derivative in the continuity equation if properties are variable. Density will always be treated as a state variable, even if it is constant. All other properties are treated as TEMPORARY variables. The general workset algorithm that computes properties evaluates them at the most recent guess of the (n+1) state. There is an additional workset algorithm that evaluates state properties at both state (n) and (n+1). The state property evaluation is only performed during the initialization phase. All material properties are evaluated at the nodes. Sub-face and sub-volume values are averaged using the element shape functions.

A linear solve is performed for each equation set within a nonlinear sub-iteration. There is a solver object associated with each equation set within the SIERRA framework. The solver object contains the matrix connectivity and manages the assembly of the matrix components. There will be ten solver objects for the full turbulent combustion mechanics (the species equations all use the same solver object). There will also be ten repeated sets of connectivity information.

The ordering of the segregated equations during one nonlinear iteration is given in the following list. Reduced equation sets for simplified mechanics maintain the same relative ordering.

1. evaluate material properties using the most recent estimate of temperature and composition
2. evaluate turbulent eddy viscosity if turbulent
3. evaluate combustion model species production rates
4. evaluate soot model production rates
5. evaluate gas and soot absorptivity for radiation model
6. solve x-momentum equation, store new predicted x-velocity until all momentum equations have been evaluated
7. solve y-momentum equation, store new predicted y-velocity until all momentum equations have been evaluated
8. solve z-momentum equation, store new predicted z-velocity until all momentum equations have been evaluated
9. update predicted velocities
10. solve continuity equation using predicted velocities, update new pressure
11. update new mass fluxes at all control volume sub-faces, including boundaries, and use in subsequent transport equations
12. perform the velocity projection and correct all nodal velocities
13. assemble turbulence friction velocity
14. solve the turbulent kinetic energy equation if turbulent, store turbulent kinetic energy until turbulence dissipation equation is solved so that the production and dissipation source terms can be properly linearized

15. solve the turbulence dissipation equation if turbulent, update the turbulent kinetic energy and turbulence dissipation
16. solve the enthalpy equation
17. solve each species equation, do not update species mass fractions until all species equations have been solved
18. solve the soot equation, store soot mass fraction until soot nuclei equation is solved
19. solve the soot nuclei equation, update soot mass fraction and soot nuclei mass fraction
20. compute Nth species mass fraction using summation rule
21. update temperature at new time level
22. compute new density and time derivative of density

This procedure is repeated within a time step until the desired level of nonlinear equation convergence is achieved.

Discrete Transport Equations

The discrete form of the linearized equations are presented in this section. The nonlinear solution procedure consists of repeated approximate Newton linearizations and linear solves of the discrete equation,

$$A\delta\phi = b. \quad (3.838)$$

The matrix A is based on an approximate linearization of F from (3.837) about a predicted value ϕ^* ,

$$A = \frac{1}{\Delta t} - \left. \frac{\partial \tilde{F}}{\partial \phi} \right|_{\phi^*}. \quad (3.839)$$

The right-hand side, b , of the linearized equation represents the residual of the nonlinear equation,

$$b = F(\phi^*, \phi^n) - \frac{\phi^* - \phi^n}{\Delta t} \quad (3.840)$$

If the nonlinear equation is converged, the right-hand side will be zero. The linear equations are solved in delta-form. The solution vector consists of the change in the unknown rather than the new value of the unknown.

There are four solution states in the nonlinear solver algorithm. The time level n is the old time level. The state $*$ represents predicted values at the new time level before the linear solve. The state $**$ represents the values after the linear solve. The time level $(n + 1)$ is the new time level. Within the nonlinear iteration cycle, values at the new time level $(n + 1)$ are copied to the predicted level $*$ before the next iteration.

There are three stages to the assembly of the matrix that result from the linearization. The first stage is the assembly of element contributions. The elements contain control-volume sub-faces that are internal to the mesh. The second state is the assembly of flux boundary conditions. The flux boundary conditions contribute to the control-volume sub-faces on the boundary of the mesh. The flux boundary condition contributions are full element contributions because they may involve both boundary and interior nodes. The third stage is the enforcement of Dirichlet boundary conditions.

The element matrix contributions are processed by first evaluating surface integral fluxes at sub-faces and then evaluating volume integral terms at sub-volumes. The flux is evaluate at a sub-face and then added or subtracted from the two adjacent control-volumes. The sub-face area components are constructed such that the face normal direction points from the left adjacent node to the right adjacent node. Fluxes are subtracted from the left node (L) and added to the right node (R). The left and right adjacent nodes for a give sub-face number within an element are given in Tables *Element variable values and differentials at control-volume faces for hexahedral elements. Face-to-edge number mapping.*, *Element variable values and differentials at control-volume faces for tetrahedral elements. Face-to-edge number mapping.*, and *Element variable values and differentials at control-volume faces for wedge elements. Face-to-edge number mapping.*.

The linearization of each transport equation can be broken into contributions from the time term, convection, diffusion, and sources.

$$A = A^t + A^c + A^d + A^s \quad (3.841)$$

$$b = b^t + b^c + b^d + b^s \quad (3.842)$$

The linear system is assembled on an element-by-element basis. Each element contributes and $N \times N$ element matrix where N is the number of nodes in the element. The nodal contribution from node J for the control volume about node I is $A_{I,J}$. Nodal variables in the following discussion are symbolized by capital letters. Linear averages of variables at face k are

$$\mu_k = \sum_J N_J|_k \mu_J \quad (3.843)$$

$$\kappa_k = \sum_J N_J|_k \kappa_J \quad (3.844)$$

The density predictor (see *Variable Density*) may be used to compute the density at the new time level for the time derivative term.

The convection operator for a face i is $C_{i,J}$ and is described in *Upwind Interpolation for Convection*.

Gradients of variables at face k are:

$$p_x = \sum_J \frac{\partial N_J}{\partial x} \Big|_k P_J \quad p_y = \sum_J \frac{\partial N_J}{\partial y} \Big|_k P_J \quad p_z = \sum_J \frac{\partial N_J}{\partial z} \Big|_k P_J \quad (3.845)$$

$$u_x = \sum_J \frac{\partial N_J}{\partial x} \Big|_k U_J \quad u_y = \sum_J \frac{\partial N_J}{\partial y} \Big|_k U_J \quad u_z = \sum_J \frac{\partial N_J}{\partial z} \Big|_k U_J \quad (3.846)$$

$$v_x = \sum_J \left. \frac{\partial N_J}{\partial x} \right|_k V_J \quad v_y = \sum_J \left. \frac{\partial N_J}{\partial y} \right|_k V_J \quad v_z = \sum_J \left. \frac{\partial N_J}{\partial z} \right|_k V_J \quad (3.847)$$

$$w_x = \sum_J \left. \frac{\partial N_J}{\partial x} \right|_k W_J \quad w_y = \sum_J \left. \frac{\partial N_J}{\partial y} \right|_k W_J \quad w_z = \sum_J \left. \frac{\partial N_J}{\partial z} \right|_k W_J \quad (3.848)$$

$$t_x = \sum_J \left. \frac{\partial N_J}{\partial x} \right|_k T_J \quad t_y = \sum_J \left. \frac{\partial N_J}{\partial y} \right|_k T_J \quad t_z = \sum_J \left. \frac{\partial N_J}{\partial z} \right|_k T_J \quad (3.849)$$

Positive-Flow Convention and Integration Quadrature

The sign on a flux integral is defined such that flow into a control volume is positive and flow out of a control volume is negative. The equations are assembled into the implicit matrix and right-hand side such that the time derivative contribution of an unknown is positive. In reference to the model differential equation, (3.837), any implicit terms that contribute to the control volume balance, $F(\phi)$, in a positive sense must be moved to the implicit left-hand side, switching signs.

The control volume balance is assembled on an element-by-element basis. Each element contributes terms from fluxes over its internal sub-control volume faces and volumetric terms from its internal sub-control volumes. A flux is computed for each sub-control volume face. The flux contribution is then summed into the two adjacent control volumes, adjusting the sign according to whether the flux is in or out of the control volume. The convention is that the sub-face normal direction between two adjacent control volumes is positive from the lower local sub-volume number to the higher sub-volume number in a local node numbering sense. The consistent treatment of fluxes is a requirement for conservation. Each sub-control volume face is numbered the same as the element edge number. The two adjacent control volumes for each edge number are given in Tables *Element variable values and differentials at control-volume faces for hexahedral elements. Face-to-edge number mapping.*, *Element variable values and differentials at control-volume faces for tetrahedral elements. Face-to-edge number mapping.*, and *Element variable values and differentials at control-volume faces for wedge elements. Face-to-edge number mapping.* for different element types.

The elemental flux contributions are assembled into a global control volume matrix. Each control volume balance is written in terms of coefficients multiplying the surrounding nodal values. In terms of matrix terminology for two-dimensional elements, the matrix coefficient for Node 5 of Figure 3.28, associated with the control volume center, is the diagonal term and should be positive. All other nodal coefficients for the control volume balance are the off-diagonal terms and complete one row of a global flux-balance matrix.

The control volume flux integrals are evaluated using numerical quadrature. The integral term for each control volume sub-face and sub-volume is evaluated using a single quadrature point. The number of quadrature points for the surface fluxes in an element is equivalent to the number of sub-faces. For example, a quadrilateral element will have four sub-face quadratures and four sub-volume quadratures. A hexahedral element will have twelve sub-face quadratures and eight sub-volume quadratures.

In three-dimensional elements, the control-volume sub-faces may not be planar. Care must be taken to conserve surface area over a control-volume to prevent non-physical sources and sinks. The sub-faces in a three-dimensional element are defined by bilinear surfaces and the discrete surface area differential is also a bilinear function. Since the quadrature for a bilinear function is exact if evaluated at the mid-point, the current quadrature strategy will ensure surface area conservation.

The quadrature coefficients are customarily derived such that the integration ranges from -1 to 1 , so a mapping is required to quadrature space.

$$\int_a^b F(\xi) d\xi = \frac{b-a}{2} \int_{-1}^1 f(\bar{\xi}) d\bar{\xi} \quad (3.850)$$

$$\xi = \frac{b+a}{2} + \frac{b-a}{2} \bar{\xi} \quad (3.851)$$

The integrand is evaluated at discrete points, called Gauss points, and summed using weighting functions.

$$\int_{-1}^1 F(\bar{\xi}) d\bar{\xi} = w_i F(\bar{\xi}_i) \quad (3.852)$$

For a one-point quadrature, $\bar{\xi}_1 = 0$ and $w_1 = 2$.

X-Momentum, 3D Laminar Transport

The time term is lumped. The time-term contribution is evaluated for each sub-volume.

$$A_{I,I}^t + = \rho_I^* \frac{\Delta V_I}{\Delta t} \quad (3.853)$$

$$b_I^t - = (\rho_I^* U_I^* - \rho_I^n U_I^n) \frac{\Delta V_I}{\Delta t} \quad (3.854)$$

The convection term is computed at each face k and assembled to the left (IL) and right (IR) control volumes.

$$A_{IL,J}^c + = C_{k,J}^* \quad (3.855)$$

$$A_{IR,J}^c - = C_{k,J}^* \quad (3.856)$$

$$b_{IL}^c - = \sum_J C_{k,J}^* U_J^* \quad (3.857)$$

$$b_{IR}^c + = \sum_J C_{k,J}^* U_J^* \quad (3.858)$$

The viscous stress term is computed at each face k and assembled to the left (IL) and right (IR) control volumes. Only the solenoidal part of the stress term is used for the matrix. The stress term may or may not include the molecular viscosity, depending on the user specified model.

$$F_{k,J} = -\mu_k \left(\frac{\partial N_J}{\partial x} \Big|_k A_x + \frac{\partial N_J}{\partial y} \Big|_k A_y + \frac{\partial N_J}{\partial z} \Big|_k A_z \right) \quad (3.859)$$

$$A_{IL,J}^d = F_{k,J} \quad (3.860)$$

$$A_{IR,J}^d = F_{k,J} \quad (3.861)$$

$$\tau_{xx} = \mu_k (u_x^* + u_x^*) \quad (3.862)$$

$$\tau_{xy} = \mu_k (u_y^* + v_x^*) \quad (3.863)$$

$$\tau_{xz} = \mu_k (u_z^* + w_x^*) \quad (3.864)$$

$$f_k = -(\tau_{xx}A_x + \tau_{xy}A_y + \tau_{xz}A_z) \quad (3.865)$$

$$b_{IL}^d = f_k \quad (3.866)$$

$$b_{IR}^d = f_k \quad (3.867)$$

The pressure is assembled in the form of a volume integral. The pressure gradients have been pre-computed at nodes use a surface-integral approximation.

$$b_I^s = \frac{\partial P}{\partial x} \Big|_I^* \Delta V_I \quad (3.868)$$

Y-Momentum, 3D Laminar Transport

The time term is lumped. The time-term contribution is evaluated for each sub-volume.

$$A_{I,I}^t = \rho_I^* \frac{\Delta V_I}{\Delta t} \quad (3.869)$$

$$b_I^t = (\rho_I^* V_I^* - \rho_I^n V_I^n) \frac{\Delta V_I}{\Delta t} \quad (3.870)$$

The convection term is computed at each face k and assembled to the left (IL) and right (IR) control volumes.

$$A_{IL,J}^c = C_{k,J}^* \quad (3.871)$$

$$A_{IR,J}^c = C_{k,J}^* \quad (3.872)$$

$$b_{IL}^c = \sum_J C_{k,J}^* V_J^* \quad (3.873)$$

$$b_{IR}^c = \sum_J C_{k,J}^* V_J^* \quad (3.874)$$

The viscous stress term is computed at each face k and assembled to the left (IL) and right (IR) control volumes. Only the solenoidal part of the stress term is used for the matrix.

$$F_{k,J} = -\mu_k \left(\frac{\partial N_J}{\partial x} \Big|_k A_x + \frac{\partial N_J}{\partial y} \Big|_k A_y + \frac{\partial N_J}{\partial z} \Big|_k A_z \right) \quad (3.875)$$

$$A_{IL,J}^d = F_{k,J} \quad (3.876)$$

$$A_{IR,J}^d = F_{k,J} \quad (3.877)$$

$$\tau_{yx} = \mu_k \left(v_x^* + u_y^* \right) \quad (3.878)$$

$$\tau_{yy} = \mu_k \left(v_y^* + v_y^* \right) \quad (3.879)$$

$$\tau_{yz} = \mu_k \left(v_z^* + w_y^* \right) \quad (3.880)$$

$$f_k = - \left(\tau_{yx} A_x + \tau_{yy} A_y + \tau_{yz} A_z \right) \quad (3.881)$$

$$b_{IL}^d = f_k \quad (3.882)$$

$$b_{IR}^d = f_k \quad (3.883)$$

The pressure is assembled in the form of a volume integral. The pressure gradients have been pre-computed at nodes use a surface-integral approximation.

$$b_I^s = \frac{\partial P}{\partial y} \Big|_I^* \Delta V_I \quad (3.884)$$

Z-Momentum, 3D Laminar Transport

The time term is lumped. The time-term contribution is evaluated for each sub-volume.

$$A_{I,I}^t = \rho_I^* \frac{\Delta V_I}{\Delta t} \quad (3.885)$$

$$b_I^t = \left(\rho_I^* W_I^* - \rho_I^n W_I^n \right) \frac{\Delta V_I}{\Delta t} \quad (3.886)$$

The convection term is computed at each face k and assembled to the left (IL) and right (IR) control volumes.

$$A_{IL,J}^c = C_{k,J}^* \quad (3.887)$$

$$A_{IR,J}^c = C_{k,J}^* \quad (3.888)$$

$$b_{IL}^c = \sum_J C_{k,J}^* W_J^* \quad (3.889)$$

$$b_{IR}^c = \sum_J C_{k,J}^* W_J^* \quad (3.890)$$

The viscous stress term is computed at each face k and assembled to the left (IL) and right (IR) control volumes. Only the solenoidal part of the stress term is used for the matrix.

$$F_{k,J} = -\mu_k \left(\frac{\partial N_J}{\partial x} \Big|_k A_x + \frac{\partial N_J}{\partial y} \Big|_k A_y + \frac{\partial N_J}{\partial z} \Big|_k A_z \right) \quad (3.891)$$

$$A_{IL,J}^d = F_{k,J} \quad (3.892)$$

$$A_{IR,J}^d = F_{k,J} \quad (3.893)$$

$$\tau_{zx} = \mu_k (w_x^* + u_z^*) \quad (3.894)$$

$$\tau_{zy} = \mu_k (w_y^* + v_z^*) \quad (3.895)$$

$$\tau_{zz} = \mu_k (w_z^* + w_z^*) \quad (3.896)$$

$$f_k = -(\tau_{zx}A_x + \tau_{zy}A_y + \tau_{zz}A_z) \quad (3.897)$$

$$b_{IL}^d = f_k \quad (3.898)$$

$$b_{IR}^d = f_k \quad (3.899)$$

The pressure is assembled in the form of a volume integral. The pressure gradients have been pre-computed at nodes use a surface-integral approximation.

$$b_I^s = \frac{\partial P}{\partial z} \Big|_I^* \Delta V_I \quad (3.900)$$

Buoyancy, Momentum Transport

The body force imposed by the buoyancy term can be constructed in one of three ways.

Boussinesq Form

For the Boussinesq approximation, the body force is evaluated at the sub-volume centroid, k , for sub-volume I .

$$b_I^s = \frac{\rho g}{T_o} \left(\sum_J N_J|_k T_J - T_o \right) \Delta V_I \quad (3.901)$$

Differential Form

For the “differential” form, the hydrostatic component of pressure has been removed. The body force is evaluated at the control-volume centroid, for sub-volume I .

$$b_I^s = (\rho_I^* - \rho_o) g \Delta V_I \quad (3.902)$$

Full Form

The body force is evaluated at the control-volume centroid, for sub-volume I .

$$b_I^s = \rho_I^* g \Delta V_I \quad (3.903)$$

Mass Transport – 3D Continuity

The time term is lumped. The time-term contribution is evaluated for each sub-volume.

$$b_I^t = (\rho_I^* - \rho_I^n) \frac{\Delta V_I}{\Delta t} \quad (3.904)$$

The convection term is computed at each face k and assembled to the left (IL) and right (IR) control volumes using the Rhie/Chow scheme from *Flow Solver*.

$$F_{k,J} = -f \Delta t \left(\left. \frac{\partial N_J}{\partial x} \right|_k A_x + \left. \frac{\partial N_J}{\partial y} \right|_k A_y + \left. \frac{\partial N_J}{\partial z} \right|_k A_z \right) \quad (3.905)$$

$$A_{IL,J}^d = F_{k,J} \quad (3.906)$$

$$A_{IR,J}^d = F_{k,J} \quad (3.907)$$

$$u_k^* = \sum_J N_J|_k U_J^* + f \frac{\Delta t}{\rho} \left(\sum_J \left. \frac{\partial P}{\partial x} \right|_J^* - p_x^* \right) + f \left(u_k^n - \sum_J N_J|_J U_J^n \right) \quad (3.908)$$

$$v_k^* = \sum_J N_J|_k V_J^* + f \frac{\Delta t}{\rho} \left(\sum_J \left. \frac{\partial P}{\partial y} \right|_J^* - p_y^* \right) + f \left(v_k^n - \sum_J N_J|_J V_J^n \right) \quad (3.909)$$

$$w_k^* = \sum_J N_J|_k W_J^* + f \frac{\Delta t}{\rho} \left(\sum_J \left. \frac{\partial P}{\partial z} \right|_J^* - p_z^* \right) + f \left(w_k^n - \sum_J N_J|_J W_J^n \right) \quad (3.910)$$

$$\dot{m}_k = \rho (u_k^* A_x + v_k^* A_y + w_k^* A_z) \quad (3.911)$$

$$b_{IL}^c = \dot{m}_k \quad (3.912)$$

$$b_{IR}^c = \dot{m}_k \quad (3.913)$$

Velocity correction and new mass flow rate. . . .

Energy, 3D Laminar Transport

The laminar energy equation is linearized with respect to the temperature. The time term is lumped. The time-term contribution is evaluated for each sub-volume. The density must also be linearized for stability.

$$A_{I,I}^t + = \left(\rho_I^* C_{p,I}^* - \rho_I^* \frac{H_I^*}{T_I^*} \right) \frac{\Delta V_I}{\Delta t} \quad (3.914)$$

$$b_I^t - = (\rho_I^* H_I^* - \rho_I^n H_I^n) \frac{\Delta V_I}{\Delta t} \quad (3.915)$$

The convection term is computed at each face k and assembled to the left (IL) and right (IR) control volumes.

$$A_{IL,J}^c + = C_{k,J}^{n+1} C_{p,J}^* \quad (3.916)$$

$$A_{IR,J}^c - = C_{k,J}^{n+1} C_{p,J}^* \quad (3.917)$$

$$b_{IL}^c - = \sum_J C_{k,J}^{n+1} H_J^* \quad (3.918)$$

$$b_{IR}^c + = \sum_J C_{k,J}^{n+1} H_J^* \quad (3.919)$$

The heat conduction term is computed at each face k and assembled to the left (IL) and right (IR) control volumes.

$$F_{k,J} = -\kappa_k \left(\frac{\partial N_J}{\partial x} \Big|_k A_x + \frac{\partial N_J}{\partial y} \Big|_k A_y + \frac{\partial N_J}{\partial z} \Big|_k A_z \right) \quad (3.920)$$

$$A_{IL,J}^d + = F_{k,J} \quad (3.921)$$

$$A_{IR,J}^d - = F_{k,J} \quad (3.922)$$

$$q_k = -\kappa_k \left(t_x^* A_x + t_y^* A_y + t_z^* A_z \right) \quad (3.923)$$

$$b_{IL}^d - = q_k \quad (3.924)$$

$$b_{IR}^d + = q_k \quad (3.925)$$

Temperature, 3D Laminar Transport

The laminar temperature equation is linearized with respect to the temperature. The time term is lumped. The time-term contribution is evaluated for each sub-volume.

$$A_{I,I}^t + = \rho_I^* \frac{\Delta V_I}{\Delta t} \quad (3.926)$$

$$b_I^t - = (\rho_I^* T_I^* - \rho_I^n T_I^n) \frac{\Delta V_I}{\Delta t} \quad (3.927)$$

The convection term is computed at each face k and assembled to the left (IL) and right (IR) control volumes.

$$A_{IL,J}^c + = C_{k,J}^{n+1} \quad (3.928)$$

$$A_{IR,J}^c - = C_{k,J}^{n+1} \quad (3.929)$$

$$b_{IL}^c - = \sum_J C_{k,J}^{n+1} T_J^* \quad (3.930)$$

$$b_{IR}^c + = \sum_J C_{k,J}^{n+1} T_J^* \quad (3.931)$$

The heat conduction term is computed at each face k and assembled to the left (IL) and right (IR) control volumes.

$$F_{k,J} = -\frac{\kappa_k}{C_{p,k}} \left(\frac{\partial N_J}{\partial x} \Big|_k A_x + \frac{\partial N_J}{\partial y} \Big|_k A_y + \frac{\partial N_J}{\partial z} \Big|_k A_z \right) \quad (3.932)$$

$$A_{IL,J}^d + = F_{k,J} \quad (3.933)$$

$$A_{IR,J}^d - = F_{k,J} \quad (3.934)$$

$$q_k = -\frac{\kappa_k}{C_{p,k}} \left(t_x^* A_x + t_y^* A_y + t_z^* A_z \right) \quad (3.935)$$

$$b_{IL}^d - = q_k \quad (3.936)$$

$$b_{IR}^d + = q_k \quad (3.937)$$

A correction for variable specific heat is applied as a volume term. The correction is computed at the centroid of the sub-volume, k , for control volume I .

$$b_I^d + = \frac{\kappa}{C_p^2} (t_x C_{p,x} + t_y C_{p,y} + t_z C_{p,z}) \Delta V_I \quad (3.938)$$

Species, 3D Laminar Transport

There is a species equations for each species. The mass fraction is Y_s , where s is the species number. The time term is lumped. The time-term contribution is evaluated for each sub-volume.

$$A_{I,I}^t + = \rho_I^* \frac{\Delta V_I}{\Delta t} \quad (3.939)$$

$$b_I^t - = \left(\rho_I^* Y_{s,I}^* - \rho_I^n Y_{s,I}^n \right) \frac{\Delta V_I}{\Delta t} \quad (3.940)$$

The convection term is computed at each face k and assembled to the left (IL) and right (IR) control volumes.

$$A_{IL,J}^c = C_{k,J}^{n+1} \quad (3.941)$$

$$A_{IR,J}^c = C_{k,J}^{n+1} \quad (3.942)$$

$$b_{IL}^c = \sum_J C_{k,J}^{n+1} Y_{s,J}^* \quad (3.943)$$

$$b_{IR}^c = \sum_J C_{k,J}^{n+1} Y_{s,J}^* \quad (3.944)$$

The mass diffusion term is computed at each face k and assembled to the left (IL) and right (IR) control volumes.

$$F_{k,J} = -\rho_k D_{s,k} \left(\frac{\partial N_J}{\partial x} \Big|_k A_x + \frac{\partial N_J}{\partial y} \Big|_k A_y + \frac{\partial N_J}{\partial z} \Big|_k A_z \right) \quad (3.945)$$

$$A_{IL,J}^d = F_{k,J} \quad (3.946)$$

$$A_{IR,J}^d = F_{k,J} \quad (3.947)$$

$$f_k = -\rho_k D_{s,k} \left(y s_x^* A_x + y s_y^* A_y + y s_z^* A_z \right) \quad (3.948)$$

$$b_{IL}^d = f_k \quad (3.949)$$

$$b_{IR}^d = f_k \quad (3.950)$$

X-Momentum, 3D Turbulent Transport

The time term is lumped. The time-term contribution is evaluated for each sub-volume.

$$A_{I,I}^t = \rho_I^* \frac{\Delta V_I}{\Delta t} \quad (3.951)$$

$$b_I^t = (\rho_I^* U_I^* - \rho_I^n U_I^n) \frac{\Delta V_I}{\Delta t} \quad (3.952)$$

The convection term is computed at each face k and assembled to the left (IL) and right (IR) control volumes.

$$A_{IL,J}^c = C_{k,J}^* \quad (3.953)$$

$$A_{IR,J}^c = C_{k,J}^* \quad (3.954)$$

$$b_{IL}^c = \sum_J C_{k,J}^* U_J^* \quad (3.955)$$

$$b_{IR}^c = \sum_J C_{k,J}^* U_J^* \quad (3.956)$$

The viscous stress term is computed at each face k and assembled to the left (IL) and right (IR) control volumes. Only the solenoidal part of the stress term is used for the matrix.

$$F_{k,J} = -(\mu_k + \mu_{T,k}) \left(\left. \frac{\partial N_J}{\partial x} \right|_k A_x + \left. \frac{\partial N_J}{\partial y} \right|_k A_y + \left. \frac{\partial N_J}{\partial z} \right|_k A_z \right) \quad (3.957)$$

$$A_{IL,J}^d = F_{k,J} \quad (3.958)$$

$$A_{IR,J}^d = F_{k,J} \quad (3.959)$$

$$\tau_{xx} = (\mu_k + \mu_{T,k}) (u_x^* + u_x^*) \quad (3.960)$$

$$\tau_{xy} = (\mu_k + \mu_{T,k}) (u_y^* + v_x^*) \quad (3.961)$$

$$\tau_{xz} = (\mu_k + \mu_{T,k}) (u_z^* + w_x^*) \quad (3.962)$$

$$f_k = -(\tau_{xx}A_x + \tau_{xy}A_y + \tau_{xz}A_z) \quad (3.963)$$

$$b_{IL}^d = f_k \quad (3.964)$$

$$b_{IR}^d = f_k \quad (3.965)$$

The pressure is assembled in the form of a volume integral. The pressure gradients have been pre-computed at nodes use a surface-integral approximation.

$$b_I^s = \left. \frac{\partial P}{\partial x} \right|_I^* \Delta V_I \quad (3.966)$$

Y-Momentum, 3D Turbulent Transport

The time term is lumped. The time-term contribution is evaluated for each sub-volume.

$$A_{I,I}^t = \rho_I^* \frac{\Delta V_I}{\Delta t} \quad (3.967)$$

$$b_I^t = (\rho_I^* V_I^* - \rho_I^n V_I^n) \frac{\Delta V_I}{\Delta t} \quad (3.968)$$

The convection term is computed at each face k and assembled to the left (IL) and right (IR) control volumes.

$$A_{IL,J}^c = C_{k,J}^* \quad (3.969)$$

$$A_{IR,J}^c = C_{k,J}^* \quad (3.970)$$

$$b_{IL}^c = \sum_J C_{k,J}^* V_J^* \quad (3.971)$$

$$b_{IR}^c = \sum_J C_{k,J}^* V_J^* \quad (3.972)$$

The viscous stress term is computed at each face k and assembled to the left (IL) and right (IR) control volumes. Only the solenoidal part of the stress term is used for the matrix.

$$F_{k,J} = -(\mu_k + \mu_{T,k}) \left(\left. \frac{\partial N_J}{\partial x} \right|_k A_x + \left. \frac{\partial N_J}{\partial y} \right|_k A_y + \left. \frac{\partial N_J}{\partial z} \right|_k A_z \right) \quad (3.973)$$

$$A_{IL,J}^d = F_{k,J} \quad (3.974)$$

$$A_{IR,J}^d = F_{k,J} \quad (3.975)$$

$$\tau_{yx} = (\mu_k + \mu_{T,k}) (v_x^* + u_y^*) \quad (3.976)$$

$$\tau_{yy} = (\mu_k + \mu_{T,k}) (v_y^* + v_y^*) \quad (3.977)$$

$$\tau_{yz} = (\mu_k + \mu_{T,k}) (v_z^* + w_y^*) \quad (3.978)$$

$$f_k = -(\tau_{yx} A_x + \tau_{yy} A_y + \tau_{yz} A_z) \quad (3.979)$$

$$b_{IL}^d = f_k \quad (3.980)$$

$$b_{IR}^d = f_k \quad (3.981)$$

The pressure is assembled in the form of a volume integral. The pressure gradients have been pre-computed at nodes use a surface-integral approximation.

$$b_I^s = \left. \frac{\partial P}{\partial y} \right|_I^* \Delta V_I \quad (3.982)$$

Z-Momentum, 3D Turbulent Transport

The time term is lumped. The time-term contribution is evaluated for each sub-volume.

$$A_{I,I}^t = \rho_I^* \frac{\Delta V_I}{\Delta t} \quad (3.983)$$

$$b_I^t = (\rho_I^* W_I^* - \rho_I^n W_I^n) \frac{\Delta V_I}{\Delta t} \quad (3.984)$$

The convection term is computed at each face k and assembled to the left (IL) and right (IR) control volumes.

$$A_{IL,J}^c = C_{k,J}^* \quad (3.985)$$

$$A_{IR,J}^c = C_{k,J}^* \quad (3.986)$$

$$b_{IL}^c = \sum_J C_{k,J}^* W_J^* \quad (3.987)$$

$$b_{IR}^c = \sum_J C_{k,J}^* W_J^* \quad (3.988)$$

The viscous stress term is computed at each face k and assembled to the left (IL) and right (IR) control volumes. Only the solenoidal part of the stress term is used for the matrix.

$$F_{k,J} = -(\mu_k + \mu_{T,k}) \left(\frac{\partial N_J}{\partial x} \Big|_k A_x + \frac{\partial N_J}{\partial y} \Big|_k A_y + \frac{\partial N_J}{\partial z} \Big|_k A_z \right) \quad (3.989)$$

$$A_{IL,J}^d = F_{k,J} \quad (3.990)$$

$$A_{IR,J}^d = F_{k,J} \quad (3.991)$$

$$\tau_{zx} = (\mu_k + \mu_{T,k}) (w_x^* + u_z^*) \quad (3.992)$$

$$\tau_{zy} = (\mu_k + \mu_{T,k}) (w_y^* + v_z^*) \quad (3.993)$$

$$\tau_{zz} = (\mu_k + \mu_{T,k}) (w_z^* + w_z^*) \quad (3.994)$$

$$f_k = -(\tau_{zx} A_x + \tau_{zy} A_y + \tau_{zz} A_z) \quad (3.995)$$

$$b_{IL}^d = f_k \quad (3.996)$$

$$b_{IR}^d = f_k \quad (3.997)$$

The pressure is assembled in the form of a volume integral. The pressure gradients have been pre-computed at nodes use a surface-integral approximation.

$$b_I^s = \frac{\partial P}{\partial z} \Big|_I^* \Delta V_I \quad (3.998)$$

Turbulent Kinetic Energy, 3D Turbulent Transport

The time term is lumped. The time-term contribution is evaluated for each sub-volume.

$$A_{I,I}^t = \rho_I^* \frac{\Delta V_I}{\Delta t} \quad (3.999)$$

$$b_I^t = (\rho_I^* K_I^* - \rho_I^n K_I^n) \frac{\Delta V_I}{\Delta t} \quad (3.1000)$$

The convection term is computed at each face k and assembled to the left (IL) and right (IR) control volumes.

$$A_{IL,J}^c = C_{k,J}^{n+1} \quad (3.1001)$$

$$A_{IR,J}^c = C_{k,J}^{n+1} \quad (3.1002)$$

$$b_{IL}^c = \sum_J C_{k,J}^{n+1} K_J^* \quad (3.1003)$$

$$b_{IR}^c = \sum_J C_{k,J}^{n+1} K_J^* \quad (3.1004)$$

The viscous stress term is computed at each face k and assembled to the left (IL) and right (IR) control volumes. Only the solenoidal part of the stress term is used for the matrix.

$$F_{k,J} = -\frac{\mu_{T,k}}{\sigma_k} \left(\left. \frac{\partial N_J}{\partial x} \right|_k A_x + \left. \frac{\partial N_J}{\partial y} \right|_k A_y + \left. \frac{\partial N_J}{\partial z} \right|_k A_z \right) \quad (3.1005)$$

$$A_{IL,J}^d = F_{k,J} \quad (3.1006)$$

$$A_{IR,J}^d = F_{k,J} \quad (3.1007)$$

$$f_k = -\frac{\mu_{T,k}}{\sigma_k} \left(k_x^* A_x + k_y^* A_y + k_z^* A_z \right) \quad (3.1008)$$

$$b_{IL}^d = f_k \quad (3.1009)$$

$$b_{IR}^d = f_k \quad (3.1010)$$

The turbulence production is assembled in the form of a volume integral. The velocity derivatives are computed at the sub-volume centroids.

$$\Phi = 2 \left(u_x^2 + v_y^2 + w_z^2 \right) - \frac{2}{3} (u_x + v_y + w_z)^2 \quad (3.1011)$$

$$+ (u_y + v_x)^2 + (v_z + w_y)^2 + (w_x + u_z)^2 \quad (3.1012)$$

$$b_I^s = \mu_T \Phi \Delta V_I \quad (3.1013)$$

The turbulence dissipation is assembled in the form of a volume integral. The terms are evaluated at the node associated with the control volume.

$$A_{I,I}^s = \rho_I \frac{E_I^*}{K_I^*} \Delta V_I \quad (3.1014)$$

$$b_I^s = \rho_I E_I^* \Delta V_I \quad (3.1015)$$

Turbulence Dissipation, 3D Turbulent Transport

The time term is lumped. The time-term contribution is evaluated for each sub-volume.

$$A_{I,I}^t = \rho_I^* \frac{\Delta V_I}{\Delta t} \quad (3.1016)$$

$$b_I^t = (\rho_I^* E_I^* - \rho_I^n E_I^n) \frac{\Delta V_I}{\Delta t} \quad (3.1017)$$

The convection term is computed at each face k and assembled to the left (IL) and right (IR) control volumes.

$$A_{IL,J}^c = C_{k,J}^{n+1} \quad (3.1018)$$

$$A_{IR,J}^c = C_{k,J}^{n+1} \quad (3.1019)$$

$$b_{IL}^c- = \sum_J C_{k,J}^{n+1} E_J^* \quad (3.1020)$$

$$b_{IR}^c+ = \sum_J C_{k,J}^{n+1} E_J^* \quad (3.1021)$$

The viscous stress term is computed at each face k and assembled to the left (IL) and right (IR) control volumes. Only the solenoidal part of the stress term is used for the matrix. As with the turbulent kinetic energy transport equation, the molecular viscosity may augment the effective diffusivity.

$$F_{k,J} = -\frac{\mu_{T,k}}{\sigma_\epsilon} \left(\frac{\partial N_J}{\partial x} \Big|_k A_x + \frac{\partial N_J}{\partial y} \Big|_k A_y + \frac{\partial N_J}{\partial z} \Big|_k A_z \right) \quad (3.1022)$$

$$A_{IL,J}^d+ = F_{k,J} \quad (3.1023)$$

$$A_{IR,J}^d- = F_{k,J} \quad (3.1024)$$

$$f_k = -\frac{\mu_{T,k}}{\sigma_\epsilon} \left(\epsilon_x^* A_x + \epsilon_y^* A_y + \epsilon_z^* A_z \right) \quad (3.1025)$$

$$b_{IL}^d- = f_k \quad (3.1026)$$

$$b_{IR}^d+ = f_k \quad (3.1027)$$

The velocity derivatives are computed at the sub-volume centroids using velocities at the new time level $(n + 1)$.

$$\Phi = 2 \left(u_x^2 + v_y^2 + w_z^2 \right) - \frac{2}{3} (u_x + v_y + w_z)^2 \quad (3.1028)$$

$$+ (u_y + v_x)^2 + (v_z + w_y)^2 + (w_x + u_z)^2 \quad (3.1029)$$

$$b_I^s+ = \mu_T C_{\epsilon_1} \Phi \frac{E_I^*}{K_I^*} \Delta V_I \quad (3.1030)$$

The turbulence dissipation is assembled in the form of a volume integral. The terms are evaluated at the node associated with the control volume.

$$A_{I,I}^s+ = \rho_I C_{\epsilon_2} \frac{E_I^*}{K_I^*} \Delta V_I \quad (3.1031)$$

$$b_I^s- = \rho_I C_{\epsilon_2} \frac{E_I^*}{K_I^*} E_I^* \Delta V_I \quad (3.1032)$$

Energy, 3D Turbulent Transport

The time term is lumped. The time-term contribution is evaluated for each sub-volume.

$$A_{I,I}^t + = \rho_I^* \frac{\Delta V_I}{\Delta t} \quad (3.1033)$$

$$b_I^t - = (\rho_I^* H_I^* - \rho_I^n H_I^n) \frac{\Delta V_I}{\Delta t} \quad (3.1034)$$

The convection term is computed at each face k and assembled to the left (IL) and right (IR) control volumes.

$$A_{IL,J}^c + = C_{k,J}^{n+1} \quad (3.1035)$$

$$A_{IR,J}^c - = C_{k,J}^{n+1} \quad (3.1036)$$

$$b_{IL}^c - = \sum_J C_{k,J}^{n+1} H_J^* \quad (3.1037)$$

$$b_{IR}^c + = \sum_J C_{k,J}^{n+1} H_J^* \quad (3.1038)$$

The heat conduction term is computed at each face k and assembled to the left (IL) and right (IR) control volumes.

$$F_{k,J} = - \left(\frac{\mu_k}{\text{Pr}} + \frac{\mu_{T,k}}{\text{Pr}_T} \right) \left(\frac{\partial N_J}{\partial x} \Big|_k A_x + \frac{\partial N_J}{\partial y} \Big|_k A_y + \frac{\partial N_J}{\partial z} \Big|_k A_z \right) \quad (3.1039)$$

$$A_{IL,J}^d + = F_{k,J} \quad (3.1040)$$

$$A_{IR,J}^d - = F_{k,J} \quad (3.1041)$$

$$q_k = - \left(\frac{\mu_k}{\text{Pr}} + \frac{\mu_{T,k}}{\text{Pr}_T} \right) \left(h_x^* A_x + h_y^* A_y + h_z^* A_z \right) \quad (3.1042)$$

$$b_{IL}^d - = q_k \quad (3.1043)$$

$$b_{IR}^d + = q_k \quad (3.1044)$$

Species, 3D Turbulent Transport

There is a species equations for each species. The mass fraction is Y_s , where s is the species number. The time term is lumped. The time-term contribution is evaluated for each sub-volume.

$$A_{I,I}^t + = \rho_I^* \frac{\Delta V_I}{\Delta t} \quad (3.1045)$$

$$b_I^t - = \left(\rho_I^* Y_{s,I}^* - \rho_I^n Y_{s,I}^n \right) \frac{\Delta V_I}{\Delta t} \quad (3.1046)$$

The convection term is computed at each face k and assembled to the left (IL) and right (IR) control volumes.

$$A_{IL,J}^c = C_{k,J}^{n+1} \quad (3.1047)$$

$$A_{IR,J}^c = C_{k,J}^{n+1} \quad (3.1048)$$

$$b_{IL}^c = \sum_J C_{k,J}^{n+1} Y_{s,J}^* \quad (3.1049)$$

$$b_{IR}^c = \sum_J C_{k,J}^{n+1} Y_{s,J}^* \quad (3.1050)$$

The mass diffusion term is computed at each face k and assembled to the left (IL) and right (IR) control volumes.

$$F_{k,J} = - \left(\frac{\mu_k}{Sc} + \frac{\mu_{T,k}}{Sc_T} \right) \left(\frac{\partial N_J}{\partial x} \Big|_k A_x + \frac{\partial N_J}{\partial y} \Big|_k A_y + \frac{\partial N_J}{\partial z} \Big|_k A_z \right) \quad (3.1051)$$

$$A_{IL,J}^d = F_{k,J} \quad (3.1052)$$

$$A_{IR,J}^d = F_{k,J} \quad (3.1053)$$

$$f_k = - \left(\frac{\mu_k}{Sc} + \frac{\mu_{T,k}}{Sc_T} \right) \left(y_{s,x}^* A_x + y_{s,y}^* A_y + y_{s,z}^* A_z \right) \quad (3.1054)$$

$$b_{IL}^d = f_k \quad (3.1055)$$

$$b_{IR}^d = f_k \quad (3.1056)$$

The chemical production source terms from the EDC model are applied at the centroid of the control volume. The production term is constructed from the rate, the fine structure mass fractions, and the average mass fractions.

$$A_{I,I}^s = \dot{r}_{s,I} \Delta V_I \quad (3.1057)$$

$$\dot{\omega}_{s,I} = \dot{r}_{s,I} \left(Y_{s,I}^{fs} - Y_{s,I} \right) \quad (3.1058)$$

$$b_I^s = \dot{\omega}_{s,I} \Delta V_I \quad (3.1059)$$

Soot Transport, 3D Turbulent Transport

The time term is lumped. The time-term contribution is evaluated for each sub-volume.

$$A_{I,I}^t = \rho_I^* \frac{\Delta V_I}{\Delta t} \quad (3.1060)$$

$$b_I^t = (\rho_I^* S_I^* - \rho_I^n S_I^n) \frac{\Delta V_I}{\Delta t} \quad (3.1061)$$

The convection term is computed at each face k and assembled to the left (IL) and right (IR) control volumes.

$$A_{IL,J}^c = C_{k,J}^{n+1} \quad (3.1062)$$

$$A_{IR,J}^c = C_{k,J}^{n+1} \quad (3.1063)$$

$$b_{IL}^c = \sum_J C_{k,J}^{n+1} S_J^* \quad (3.1064)$$

$$b_{IR}^c = \sum_J C_{k,J}^{n+1} S_J^* \quad (3.1065)$$

The diffusion term is computed at each face k and assembled to the left (IL) and right (IR) control volumes.

$$F_{k,J} = - \left(\frac{\mu_k}{Sc} + \frac{\mu_{T,k}}{Sc_T} \right) \left(\frac{\partial N_J}{\partial x} \Big|_k A_x + \frac{\partial N_J}{\partial y} \Big|_k A_y + \frac{\partial N_J}{\partial z} \Big|_k A_z \right) \quad (3.1066)$$

$$A_{IL,J}^d = F_{k,J} \quad (3.1067)$$

$$A_{IR,J}^d = F_{k,J} \quad (3.1068)$$

$$f_k = - \left(\frac{\mu_k}{Sc} + \frac{\mu_{T,k}}{Sc_T} \right) (s_x^* A_x + s_y^* A_y + s_z^* A_z) \quad (3.1069)$$

$$b_{IL}^d = f_k \quad (3.1070)$$

$$b_{IR}^d = f_k \quad (3.1071)$$

The soot production source term from the EDC model is applied at the centroid of the control volume.

$$b_I^s = \dot{\omega}_{soot,I} \Delta V_I \quad (3.1072)$$

Soot Nuclei Transport, 3D Turbulent Transport

The time term is lumped. The time-term contribution is evaluated for each sub-volume.

$$A_{I,I}^t = \rho_I^* \frac{\Delta V_I}{\Delta t} \quad (3.1073)$$

$$b_I^t = (\rho_I^* N_I^* - \rho_I^n N_I^n) \frac{\Delta V_I}{\Delta t} \quad (3.1074)$$

The convection term is computed at each face k and assembled to the left (IL) and right (IR) control volumes.

$$A_{IL,J}^c = C_{k,J}^{n+1} \quad (3.1075)$$

$$A_{IR,J}^c = C_{k,J}^{n+1} \quad (3.1076)$$

$$b_{IL}^c = \sum_J C_{k,J}^{n+1} N_J^* \quad (3.1077)$$

$$b_{IR}^c = \sum_J C_{k,J}^{n+1} N_J^* \quad (3.1078)$$

The diffusion term is computed at each face k and assembled to the left (IL) and right (IR) control volumes.

$$F_{k,J} = - \left(\frac{\mu_k}{Sc} + \frac{\mu_{T,k}}{Sc_T} \right) \left(\frac{\partial N_J}{\partial x} \Big|_k A_x + \frac{\partial N_J}{\partial y} \Big|_k A_y + \frac{\partial N_J}{\partial z} \Big|_k A_z \right) \quad (3.1079)$$

$$A_{IL,J}^d = F_{k,J} \quad (3.1080)$$

$$A_{IR,J}^d = F_{k,J} \quad (3.1081)$$

$$f_k = - \left(\frac{\mu_k}{Sc} + \frac{\mu_{T,k}}{Sc_T} \right) \left(n_x^* A_x + n_y^* A_y + n_z^* A_z \right) \quad (3.1082)$$

$$b_{IL}^d = f_k \quad (3.1083)$$

$$b_{IR}^d = f_k \quad (3.1084)$$

The soot nuclei production source term from the EDC model is applied at the centroid of the control volume.

$$b_I^s = \dot{\omega}_{nucl,I} \Delta V_I \quad (3.1085)$$

Discrete Boundary Conditions

The Dirichlet boundary conditions are applied directly in the linear solver. The flux boundary conditions are linearized and then assembled to the linear system. The flux boundary conditions are processed on a face-by-face basis. The data available with each face includes all the data on the parent element.

Symmetry, 3D Momentum

The viscous stresses can only impart a normal force at a symmetry boundary. The only other force contribution is from the pressure. The pressure is integrated over the boundary using the boundary nodal values.

The normal viscous force component is assembled to the right hand side only for the laminar equations.

The viscous stress and sub-face normal are computed at each sub-face on the element face. The integrated sub-face force is assembled to its adjacent node.

$$F_{wi} = \mu \left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i} \right) n_j A_w \quad (3.1086)$$

where n_j is the unit sub-face normal vector and A_w is the area of the sub-face.

Outflow, 3D Mass

The mass flux at a pressure-specified outflow boundary is given by (3.836). The pressure at the face in the equation is P_{fc} and is the specified value (see Figure 3.31). The interior sub-face pressure is P_{ss} and is an average of nodal pressures. The fully assembled Poisson equation for pressure will have positive diagonal coefficients. Note that the form of (3.836) will contribute a positive diagonal value. The nodal pressure gradient, $G_{ij}p_j^{ast}$, contains the influence of the specified pressure. The difference of the nodal pressure gradient and the boundary pressure gradient cancels the influence of the specified pressure in the outflow boundary condition. The specified pressure at the boundary only directly influences the momentum balance.

Outflow, 3D Momentum

The outflow boundary condition is applied to boundaries with either pressure-specified inflow or pressure-specified outflow. The viscous stresses are integrated over the boundary, but the viscous force normal to the boundary is neglected.

If the flow is entering the domain, the convected velocity is a combination of a specified tangential velocity (coflow) and a normal velocity. The normal velocity is constructed from the local nodal values.

If the flow exits the domain, the convected velocity values are interpolated from nodal velocities in the element adjacent to the boundary, similar to the interior scheme discussed in *Upwind Interpolation for Convection*. The convected velocities are blended from an upwind interpolation (nearest boundary node) and centered interpolation. The shape functions for the centered interpolation are taken from the interior sub-face that is directly opposite the boundary sub-face. The upwind scheme will extrapolate from the nearest node and the linear profile skew upwind scheme will interpolate to the boundary sub-face centroid.

Outflow, 3D Energy and Temperature

The outflow boundary condition is applied to boundaries with either pressure-specified inflow or pressure-specified outflow. The heat conduction is integrated over the boundary. The transport of enthalpy by mass diffusion for a multicomponent system is not yet implemented (cdm – 9/26/10).

If the flow is entering the domain, the convected enthalpy is set to a far-field reference value.

The convected enthalpy values are interpolated from nodal enthalpies in the element adjacent to the boundary, similar to the interior scheme discussed in *Upwind Interpolation for Convection*. The convected enthalpies are blended from an upwind interpolation (nearest boundary node) and centered interpolation. The shape functions for the centered interpolation are taken from the interior sub-face that is directly opposite the boundary sub-face. The upwind scheme will extrapolate from the nearest node and the linear profile skew upwind scheme will interpolate to the boundary sub-face centroid.

Outflow, 3D Species and Soot

The outflow boundary condition is applied to boundaries with either pressure-specified inflow or pressure-specified outflow. The mass diffusion is integrated over the boundary.

If the flow is entering the domain, the convected mass fractions are set to far-field reference values.

The convected species mass fraction values are interpolated from nodal mass fractions in the element adjacent to the boundary, similar to the interior scheme discussed in *Upwind Interpolation for Convection*. The convected mass fractions are blended from an upwind interpolation (nearest boundary node) and centered interpolation. The shape functions for the centered interpolation are taken from the interior sub-face that is directly opposite the boundary sub-face. The upwind scheme will extrapolate from the nearest node and the linear profile skew upwind scheme will interpolate to the boundary sub-face centroid.

Outflow, 3D Turbulent Kinetic Energy

The outflow boundary condition is applied to boundaries with either pressure-specified inflow or pressure-specified outflow. If the flow is entering the domain, the convected turbulent kinetic energy is set by one of two ways:

- user specified value for turbulent kinetic energy, e.g. 0.0.,
- calculated entrainment value based on user specified turbulence intensity, T_{in} , and the relationship

$$k_{ip} = \frac{3}{2} (U_{ref} T_{in})^2. \quad (3.1087)$$

The reference velocity at the integration point, U_{ref} , is determined by the current integration point mass flow rate divided by a characteristic area divided by the integration point density.

The convected turbulent kinetic energy is blended from an upwind interpolation (nearest boundary node) and centered interpolation. The shape functions for the centered interpolation are taken from the interior sub-face that is directly opposite the boundary sub-face. The upwind scheme will extrapolate from the nearest node and the linear profile skew upwind scheme will interpolate to the boundary sub-face centroid.

Outflow, 3D Turbulence Dissipation

The outflow boundary condition is applied to boundaries with either pressure-specified inflow or pressure-specified outflow. If the flow is entering the domain, the convected turbulence dissipation rate is set by one of two ways:

- user specified value for turbulent dissipation rate, e.g. 0.0.,
- calculated entrainment value based on user specified turbulence intensity, characteristic length and the relationship

$$\epsilon_{ip} = C_\mu^{3/4} \frac{k_{ip}^{3/2}}{l}, \quad (3.1088)$$

where $l = 0.07L$; L represents the user-specified characteristic length of large turbulent structures. The integration point turbulent kinetic energy is again based on the user specified turbulence intensity in conjunction with Equation (3.1087).

The convected turbulent dissipation rate is blended from an upwind interpolation (nearest boundary node) and centered interpolation. The shape functions for the centered interpolation are taken from the interior sub-face that is directly opposite the boundary sub-face. The upwind scheme will extrapolate from the nearest node and the linear profile skew upwind scheme will interpolate to the boundary sub-face centroid.

Wall, 3D Turbulent Momentum

The effect of the wall force imparted by the wall on the fluid, as outlined in [Wall Functions; Momentum](#), is handled by the standard law of the wall formulation. To explain this procedure, consider a two dimensional element with two faces that consist of a wall boundary side set, [Figure 3.32](#).

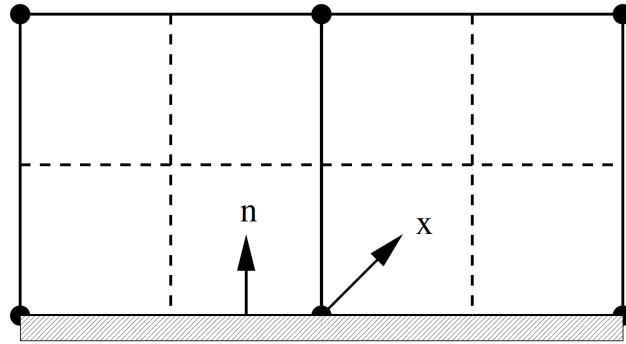


Fig. 3.32: Integration locations for a wall boundary.

The resulting discretization of the i^{th} -component of velocity, for the boundary face that is a wall can be expressed as follows,

$$F_{wi} = - \int \tau_{ij} n_j dS = \lambda_w A_w u_{i||}, \quad (3.1089)$$

where A_w is the area, n_j is the unit normal to the wall, and λ_w is the wall shear stress factor from law of the wall,

$$\lambda_w = \frac{\rho \kappa u_\tau}{\ln(Ey^+)}. \quad (3.1090)$$

The parallel velocity component in (3.1089) is determined by the projection of the nodal velocity onto each of the four (hex) or three (tet) subcontrol boundary faces (see (3.206)). In many respects, this procedure is very much like that of a cell-centered scheme in that the nodal velocity is assumed to act over all boundary faces. The paramount difference is the ability of one nodal velocity to be applied to a multitude of faces of potentially different orientation.

As indicated in *Wall Functions; Momentum*, the friction velocity at the centroid of the boundary face is determined by a nonlinear solution procedure that will now be described. The procedure begins by use of (3.195), rearranged to form the function F ,

$$F(u_\tau) = u_\parallel - \frac{u_\tau}{\kappa} \ln \left(\frac{E \rho Y_p u_\tau}{\mu} \right). \quad (3.1091)$$

The objective is to determine the value of the friction velocity such that the function, F , is minimized. A Newton solve is therefore constructed that has the following standard iteration form,

$$u_\tau^{k+1} = u_\tau^k - \frac{F^k}{F'^k}, \quad (3.1092)$$

where F^k is defined by (3.1093) evaluated at the k^{th} iteration level, and F'^k is defined by

$$F'^k = -\frac{1}{\kappa} \left[1 + \ln \left(\frac{E \rho Y_p u_\tau^k}{\mu} \right) \right]. \quad (3.1093)$$

The procedure by which the normal distance to the wall is determined is based on the method outlined by the vertex-centered CFD code TASCflow [141]. In the procedure, the normal distance to the wall is linked to the grid by the evaluation of the normal distance from the subcontrol volume center to the boundary face. Therefore, the normal distance to the wall can be determined by the following steps:

- Determination of the coordinates of the subcontrol volume center by a shape function loop over all nodes. This step in the procedure mandates a SIERRA heterogeneous (face-element) workset algorithm.
- The determination of a vector, \mathbf{x}_i , from the subcontrol volume center to the respective nodal location.
- The use of the perpendicular projection operator, P_\perp , which is defined by,

$$P_\perp = -n_i n_j, \quad (3.1094)$$

- The determination of the normal distance by

$$Y_p = \sqrt{x_{\perp,1}^2 + x_{\perp,2}^2 + x_{\perp,3}^2}. \quad (3.1095)$$

For convenience, the density and viscosity used in all of the above equations are nodal quantities. In other words, the physical properties are not interpolated to the centroid of the boundary face.

Once the wall shear stress factor is evaluated, it is required that the appropriate component of the velocity parallel to the boundary face is used appropriately within the respective momentum equations. As was discussed in the section on non-orthogonal momentum math models, *Wall Functions; Momentum*, the parallel velocity can be written in component form (see (3.207)).

X-Momentum

The x-momentum wall force, F_{w1} is expressed as

$$F_{w1} = -\lambda_w A_w u_{1\parallel}, \quad (3.1096)$$

where $u_{1\parallel}$ is defined as

$$u_{1\parallel} = \left(1 - n_1^2\right) u_{1,nd} - (1 - n_1 n_2) u_{2,nd} - (1 - n_1 n_3) u_{3,nd}. \quad (3.1097)$$

Note that the form of (3.1097) allows for an implicit treatment of the force imparted by the wall on the fluid by the factor

$$\lambda_w A_w \left(1 - n_1^2\right). \quad (3.1098)$$

Y-Momentum

The y-momentum wall force, F_{w2} is expressed as

$$F_{w2} = -\lambda_w A_w u_{2\parallel}, \quad (3.1099)$$

where $u_{2\parallel}$ is defined as

$$u_{2\parallel} = \left(1 - n_2^2\right) u_{2,nd} - (1 - n_2 n_1) u_{1,nd} - (1 - n_2 n_3) u_{3,nd}. \quad (3.1100)$$

Note that the form of (3.1100) allows for an implicit treatment of the force imparted by the wall on the fluid by the factor

$$\lambda_w A_w \left(1 - n_2^2\right). \quad (3.1101)$$

Z-Momentum

The z-momentum wall force, F_{w3} is expressed as

$$F_{w3} = -\lambda_w A_w u_{3\parallel}, \quad (3.1102)$$

where $u_{3\parallel}$ is defined as

$$u_{3\parallel} = \left(1 - n_3^2\right) u_{3,nd} - (1 - n_3 n_1) u_{1,nd} - (1 - n_3 n_2) u_{2,nd}. \quad (3.1103)$$

Note that the form of (3.1103) allows for an implicit treatment of the force imparted by the wall on the fluid by the factor

$$\lambda_w A_w \left(1 - n_3^2\right). \quad (3.1104)$$

Wall, 3D Turbulent Kinetic Energy

As described in *Wall Functions; Turbulent Kinetic Energy*, the wall boundary condition for turbulent kinetic energy can be applied in a variety of ways. In general, there are two supported methods.

The first method is specify the near-wall turbulent kinetic energy as a Dirichlet condition whose value is determined by the assumption of local equilibrium between production and dissipation of turbulent kinetic energy.

The second method is to solve a transport equation for the near wall turbulent kinetic energy whose form utilizes a modified production and dissipation term based on the assumption of local equilibrium between production and dissipation of turbulent kinetic energy. The use of a full control volume equation for the near wall turbulent kinetic energy in the presence of non-zero convection and diffusion coefficients is a violation of the very tenants of the law of the wall formulation which implicitly assumes pure shear flow behavior. Nevertheless, this method is frequently used.

The Dirichlet method consists of the determination of each integration point turbulent kinetic energy by use of the following equation,

$$k_{ip} = \frac{u_\tau^2}{C_\mu^{1/2}}, \quad (3.1105)$$

The value of u_τ is determined by a nonlinear iteration solve of the law of the wall formulation. The integration point values are area weighted and assembled into the nodal location. The nodal value of the turbulent kinetic energy is given by the accumulated area weighed integration point turbulent kinetic energy divided by the total face area.

Wall, 3D Turbulence Dissipation

Consistent with all of literature, the near-wall value of turbulent dissipation is determined from iteration-lagged values of friction velocity,

$$\epsilon_{ip}^{n+1} = \frac{u_\tau^3}{\kappa Y_p}. \quad (3.1106)$$

As with the implementation of the turbulent kinetic energy, the value computed in (3.1106) is area weighted and assembled to the nodal location. The Dirichlet condition is determined by the assembled quantity divided by the entire area of the boundary faces that are “owned” by the node.

Conjugate Heat Transfer

General Formulation

A conjugate heat transfer problem is one in which conductive heat transfer in a solid region is coupled to the convective heat transfer in a neighboring fluid. In its most general form, the coupling at the boundary is governed by the conservation of energy, such that heat flux out of the solid is equal to heat flux into the fluid:

$$\mathbf{q}_s \cdot \mathbf{n} = \mathbf{q}_f \cdot \mathbf{n} \quad (3.1107)$$

where \mathbf{q}_s and \mathbf{q}_f are the heat flux in the solid and fluid, respectively, and \mathbf{n} is the surface normal directed {em into} the solid and {em out of} the fluid.

The exact form in which equation ((3.1107)) is implemented depends on whether the fluid flow is laminar or turbulent, since different expressions must be used in these cases for \mathbf{q}_f . The heat flux in the solid is always due to conduction alone, but there are several possible choices that could be made for the discretization of this flux in space and time.

Time Integration

In Fuego, conjugate heat transfer is implemented through loose coupling between the fluid and solid regions, meaning that at each time step, each region is solved separately by treating information from the neighboring region as “given”, and no extra iterations are done between regions to ensure convergence at a single time step. The specific algorithm used can be described as a temperature-forward, flux-back scheme. At a given time step n , the fluid equations are solved using the current solid temperature as a Dirichlet boundary condition; the temperature field of the fluid is thus updated to state $n + 1$ everywhere except on the boundary. Then, the heat flux in the fluid at step $n + 1$ is computed and transferred to the solid. Finally, the solid region is solved, updating to state $n + 1$ using the information from the fluid as a flux boundary condition.

Rather than applying the fluid heat flux directly to the solid, we choose to write the solid boundary condition in the form of a convective heat flux boundary condition:

$$\mathbf{q}_s(\mathbf{x}) \cdot \mathbf{n} = h(\mathbf{x}) (T_\infty(\mathbf{x}) - T_s(\mathbf{x})) \quad (3.1108)$$

where h is a convection coefficient, T_∞ is the fluid temperature away from the wall, and T_s is the solid surface temperature. Both h and T_∞ are computed from the fluid temperature field in a way that will be specified, while T_s is left free in the solution of the solid region temperature. This

formulation can be shown to be more stable than the alternative of simply transferring the heat flux in the fluid and applying it as a pure Neumann boundary condition to the solid.

Using superscripts to denote time step, the loosely coupled integration scheme can thus be written as:

$$T_f^{n+1} = T_s^n \quad \text{on } \Gamma_{fs} \quad (3.1109)$$

$$\mathbf{q}_s^{n+1}(\mathbf{x}) \cdot \mathbf{n} = h^{n+1}(\mathbf{x}) \left(T_\infty^{n+1}(\mathbf{x}) - T_s^{n+1}(\mathbf{x}) \right) \quad \text{on } \Gamma_{fs} \quad (3.1110)$$

where Γ_{fs} is the fluid-solid interface.

Discretization of Conduction Region Boundary Condition

The quantity that is needed for a flux boundary condition condition in our CVFEM formulation is the heat flux integrated over the interface surface area associated with each node on the surface. Equation ((3.1110)) is applied to the conduction region at each surface node by assuming that h , T_s and T_∞ can be treated as constants on that node's sub-control surfaces:

$$Q_{s,I}^{n+1} = \int_{SCS_I} \mathbf{q}_s \cdot \mathbf{n} dA = h_I^{n+1} A_I \left(T_{\infty,I}^{n+1} - T_{s,I}^{n+1} \right) \quad (3.1111)$$

where A_I is the surface area associated with node I . The nodal data h_I^{n+1} and $T_{\infty,I}^{n+1}$ are computed from the fluid solution at time step $n + 1$ (see section *Computation of Convection Temperature and Coefficient*), while $T_{s,I}^{n+1}$ is a degree of freedom solved during the conduction region solution.

Computation of Convection Temperature and Coefficient

On the fluid side, the total heat transfer associated with a node on the fluid-solid interface is the integral of the heat flux over that node's sub-control surfaces on the interface:

$$Q_{f,I}^{n+1} = \int_{SCS_I} \mathbf{q}_f^{n+1} \cdot \mathbf{n} dA. \quad (3.1112)$$

Consider the case in which fluid and solid surfaces meshes conform exactly at the interface. Then, every fluid node can be associated with a corresponding solid node, and using Equations ((3.1107)) and ((3.1111)) we have:

$$\begin{aligned} Q_{f,I}^{n+1} &= Q_{s,I}^{n+1} \\ &= h_I^{n+1} A_I \left(T_{\infty,I}^{n+1} - T_{s,I}^{n+1} \right) \\ &\approx h_I^{n+1} A_I \left(T_{\infty,I}^{n+1} - T_{f,I}^{n+1} \right) \end{aligned} \quad (3.1113)$$

where the last line (where $T_{f,I}^{n+1}$ is substituted for $T_{s,I}^{n+1}$) follows approximately from ((3.1109)); this approximation is of the same order accuracy as the time integration scheme, and for steady state it

is exact. In cases in which the surface meshes do not conform exactly, the nodal values of h_I and $T_{\infty,I}$ are passed through an interpolation transfer, introducing a small amount of error.

The total heat transfer $Q_{f,I}$ must be decomposed into two components: $Q_{W,I}$ representing the variables of the fluid at nodes on the surface (“wall”), and $Q_{\infty,I}$ representing variables at nodes away from the surface:

$$Q_{f,I} = Q_{W,I} + Q_{\infty,I} \quad (3.1114)$$

The way in which this decomposition is done depends on whether the flow is laminar or turbulent, as will be discussed. Comparing this decomposition with (*Computation of Convection Temperature and Coefficient*), it is clear that:

$$\begin{aligned} Q_{W,I} &= -h_I A_I T_{f,I} \\ Q_{\infty,I} &= h_I A_I T_{\infty,I} \end{aligned} \quad (3.1115)$$

Rearranging:

$$\begin{aligned} h_I &= -\frac{Q_{W,I}}{T_{f,I} A_I} \\ T_{\infty,I} &= \frac{Q_{\infty,I}}{h_I A_I} \end{aligned} \quad (3.1116)$$

Finally, we must define the decomposition of $Q_{f,I}$ for laminar and turbulent flow. It is possible when using this approach to end up with negative values for h_I , which appear non-physical to the analyst and are detrimental to the numerical stability of the conduction solve since they reduce diagonal dominance of the linear system. Since the choice of these parameters is arbitrary as long as they reproduce the correct energy flux, when this occurs we reverse the sign of h_I and re-compute $T_{\infty,I}$ as

$$T_{\infty,I} = \frac{Q_{f,I}}{h_I} + T_{f,I} \quad (3.1117)$$

Resolution of Boundary Layer

The fluid velocity at the solid surface is zero for laminar flows and turbulent flow models in which the boundary layer is resolved, so all heat transfer in the fluid near walls is due to conduction:

$$\mathbf{q}_f(\mathbf{x}) = -\kappa_f(\mathbf{x}) \nabla T(\mathbf{x}) \quad (3.1118)$$

where κ_f is the thermal conductivity of the fluid. Substituting this into ((3.1112)) and using the finite element interpolation for $T(\mathbf{x})$ gives:

$$Q_{W,I} = - \int_{SCS_I} \kappa_f \sum_J (\mathbf{n} \cdot \nabla N_J) T_J dA \quad (3.1119)$$

where N_J and T_J are respectively the FEM shape function and temperature degree of freedom associated with node J .

The most obvious way of decomposing $Q_{f,I}$ is by breaking the summation into two summations, one over boundary nodes, one over off-boundary nodes:

$$\begin{aligned} Q_{f,I} &= - \int_{SCS_I} \kappa_f \sum_{J \in B} (\mathbf{n} \cdot \nabla N_J) T_J dA \\ Q_{\infty,I} &= - \int_{SCS_I} \kappa_f \sum_{J \notin B} (\mathbf{n} \cdot \nabla N_J) T_J dA \end{aligned} \quad (3.1120)$$

where B is the set of nodes on the wall.

These quantities, when substituted into (*Computation of Convection Temperature and Coefficient*), give the computed values of h_I and $T_{\infty,I}$.

Turbulent flow modeling with wall functions

In turbulent flow where the boundary layer is not resolved, wall boundary conditions are applied by assuming that the first layer of nodes in the fluid lies not exactly on the solid interface, but slightly away from the wall in the turbulent boundary layer. Various laws of the wall can then be used to relate quantities at these nodes to the wall values.

$$Q_{f,I} = c_I A_I (H_I - H_{W,I}) \quad (3.1121)$$

where H_I is the nodal enthalpy, $H_{W,I}$ is the corresponding enthalpy exactly at the wall, and c_I is a coefficient that depends on the flow variables. The most obvious decomposition is to let $Q_{W,I} = -c_I A_I H_{W,I}$ and $Q_{\infty,I} = c_I A_I H_I$. However, this most obvious decomposition is incorrect. The difficulty is that enthalpy is measured on a relative scale, rather than an absolute scale like temperature. For example, consider the case where $H_I = 0$. This does not imply that $T_I = 0$; in Fuego, it usually corresponds to something near standard temperature and pressure. However, the obvious decomposition when substituted into (*Computation of Convection Temperature and Coefficient*) gives $T_{\infty,I} = 0$, which is clearly the wrong value for the conduction region boundary condition.

Thus, we should choose a decomposition that has $Q_{\infty,I} = 0$ only if $T_{\infty,I}$ should be zero. The correct choice is:

$$\begin{aligned} Q_{W,I} &= -c_I T_{W,I} \left(\frac{H_I - H_{W,I}}{T_I - T_{W,I}} \right) \\ Q_{\infty,I} &= c_I T_I \left(\frac{H_I - H_{W,I}}{T_I - T_{W,I}} \right) \end{aligned} \quad (3.1122)$$

where $T_{W,I}$ is the wall temperature (which for conjugate heat transfer has been obtained from the solid at the previous time step), and T_I is the temperature value at node I (slightly away from the wall). These expressions are undefined if $T_I = T_{W,I}$; in that case, the fraction $\Delta H / \Delta T$ is approximated using the limiting value given by the specific heat c_p .

Element Topology and Shape Functions

The standard mesh configuration for cell-centered CVFEM's is to co-locate all flow variables at the nodes, also called grid points. The nodes are the vertices of the finite-elements, as shown in [Figure 3.28](#). The finite-volumes, also called control volumes, are centered about the nodes. Each element contains a set of sub-faces that define control-volumes. The sub-faces consist of the segments or surfaces that bisect the element faces. For example, each control volume on an orthogonal mesh of rectangular elements is defined by four neighboring elements with contributions from the nine nodal values.

Interpolation functions are formed inside each element. In standard finite element methods, the interpolation functions are called shape functions and they are used to evaluate the integral quadratures. The same bilinear or trilinear shape functions are used in CVFEM to construct fluxes at the sub-faces. Finite-element basis functions are used as interpolation functions to integrate fluxes over control volume faces which are internal to an element. The control-volume flux interpolation functions are element based; a restriction by choice, motivated by code development considerations. In an element-based scheme, only information that defines an element may be used to assemble fluxes. Nodal information outside the element cannot be used. As a result, the global spatial accuracy is restricted to second order.

Isoparametric shape functions are used for quadrilateral and hexahedral elements. The geometry of an isoparametric element is approximated with the same shape function as the solution variables so that the bilinear/trilinear variation within remains independent of orientation. Triangular and tetrahedral elements do not require an isoparametric formulation because they are linear. The triangles and tetrahedra can be made to look like isoparametric elements in order to create a general element evaluation algorithm.

Quadrilateral Elements

The quadrilateral element has four nodes and four control volume faces. The element configuration is shown in [Figure 3.33](#). The parametric variables are ξ and η , and they are coincident with the faces of the control volumes. The control volume faces are formed by the straight line segments that connect the bisection points of opposing element edges. The parametric variables have the range $-1 \leq \xi \leq 1$ and $-1 \leq \eta \leq 1$.

Geometric information inside the element is interpolated from the nodal coordinates. Derivatives of the physical coordinates are the most fundamental geometric quantity, contributing to the surface areas and gradients.

$$x = N_k X_k \quad (3.1123)$$

$$y = N_k Y_k \quad (3.1124)$$

$$\frac{\partial x}{\partial \xi} = \frac{\partial N_k}{\partial \xi} X_k \quad (3.1125)$$

$$\frac{\partial y}{\partial \xi} = \frac{\partial N_k}{\partial \xi} Y_k \quad (3.1126)$$

$$\frac{\partial x}{\partial \eta} = \frac{\partial N_k}{\partial \eta} X_k \quad (3.1127)$$

$$\frac{\partial y}{\partial \eta} = \frac{\partial N_k}{\partial \eta} Y_k \quad (3.1128)$$

The subscripts on the shape functions correspond to the element-local node numbering. The isoparametric shape functions and shape function derivatives for quadrilateral elements are given in Table 3.22.

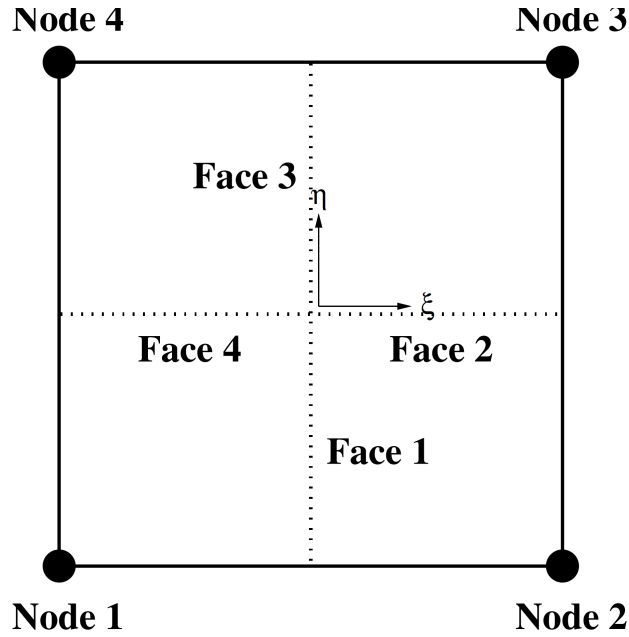


Fig. 3.33: Quadrilateral element topology and numbering

Table 3.22: Nodal shape functions and derivatives for quadrilateral elements

node	N	$\frac{\partial N}{\partial \xi}$	$\frac{\partial N}{\partial \eta}$
1	$\frac{1}{4}(1 - \xi)(1 - \eta)$	$-\frac{1}{4}(1 - \eta)$	$-\frac{1}{4}(1 - \xi)$
2	$\frac{1}{4}(1 + \xi)(1 - \eta)$	$\frac{1}{4}(1 - \eta)$	$-\frac{1}{4}(1 + \xi)$
3	$\frac{1}{4}(1 + \xi)(1 + \eta)$	$\frac{1}{4}(1 + \eta)$	$\frac{1}{4}(1 + \xi)$
4	$\frac{1}{4}(1 - \xi)(1 + \eta)$	$-\frac{1}{4}(1 + \eta)$	$\frac{1}{4}(1 - \xi)$

The physical surface differentials are related to differentials in parametric space. The surface area differentials, $n_i dA$ are derived from their three-dimensional counterpart, (3.1132), where $x_\zeta = 0$, $y_\zeta = 0$, and $z_\zeta = 1$. The derivatives used in the mapping from a differential in parametric space to a differential in physical space are evaluated using (3.1125) and (3.1127). The differential surfaces of a control-volume sub-face are surfaces of constant ξ or η . Along Face 1 and 3, the differential $d\xi = 0$. Along Face 2 and 4, the differential $d\eta = 0$. For the purposes of constructing a

general-purpose computational flux algorithm, integration over both parametric components is retained. On each face, only one surface area component will be non-zero.

$$n_i dS = \begin{bmatrix} -y_\xi & x_\xi \end{bmatrix} d\xi + \begin{bmatrix} y_\eta & -x_\eta \end{bmatrix} d\eta \quad (3.1129)$$

The usefulness of the general approach will become more apparent when triangular elements are considered.

The normals to the control-volume surfaces are positive in the direction of positive coordinate axes. The normal to a ξ -constant face is along the positive ξ -axis. The normal to a η -constant face is along the positive η -axis. The signs on the differentials are selected such that the fluxes have the proper signs relative to the control volume. The values of the element variables and the surface differentials at the control-volume faces are given in [Table 3.23](#). The differential $d\eta$ is negative for Face 3 because the direction for out/in flow from Node 3 to Node 4 is opposite in direction of the surface normal defined by $n_i dS$.

Table 3.23: Element variable values and differentials at control-volume faces for quadrilateral elements. Face-to-edge number mapping.

face	Edge (Node _{out} → Node _{in})	ξ	η	$d\xi$	$d\eta$
1	1 → 2	0	$-\frac{1}{2}$	0	1
2	2 → 3	$\frac{1}{2}$	0	1	0
3	3 → 4	0	$\frac{1}{2}$	0	-1
4	1 → 4	$-\frac{1}{2}$	0	1	0

Volume integrals require the volume differential, $dx dy$. In terms of the element parameters, the volume differential is

$$dx dy = J d\xi d\eta, \quad (3.1130)$$

where

$$J = x_\xi y_\eta - x_\eta y_\xi. \quad (3.1131)$$

The quadrature points and differential values are shown in [Table 3.24](#).

Table 3.24: Element variable values and differentials at sub-control volume centers for quadrilateral elements.

sub-volume	ξ	η	$d\xi d\eta$
1	$-\frac{1}{2}$	$-\frac{1}{2}$	1
2	$\frac{1}{2}$	$-\frac{1}{2}$	1
3	$\frac{1}{2}$	$\frac{1}{2}$	1
4	$-\frac{1}{2}$	$\frac{1}{2}$	1

Triangular Elements

The triangular element has three nodes and three control volume faces. The element configuration is shown in [Figure 3.34](#). The control volume faces run from the centroid of the element to the element edge bisection points.

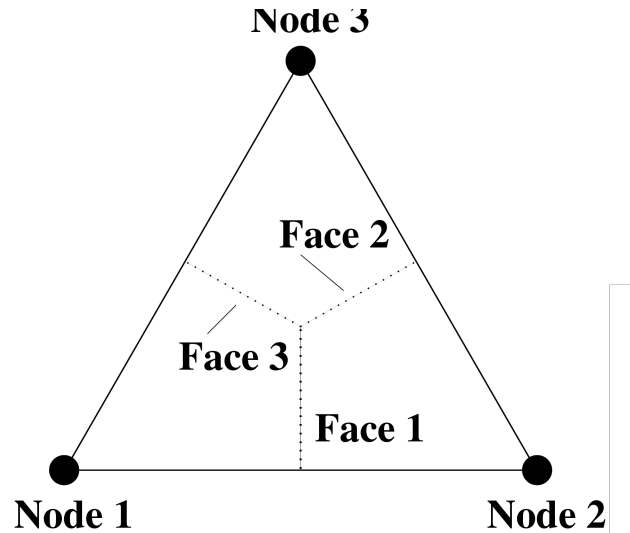


Fig. 3.34: Triangular element topology and numbering

The parametric coordinate system is defined by the triangle natural coordinates, L_1 , L_2 and L_3 , since a Cartesian mapping cannot be defined. The natural coordinates are the shape functions. As an example, the value of L_1 at an interpolation point is the shape function associated with Node 1. The value of L_1 is the fraction of the element triangle area covered by a sub-triangle, formed by the interpolation point and the edge opposite of Node 1, shown in [Figure 3.35](#). For consistency with the quadrilateral element notation, the (ξ, η) parametric variables are defined as $\xi = L_1$ and $\eta = L_2$, where L_3 is defined by the fact that the natural coordinates always sum to one.

The linear shape functions and shape function derivatives for triangular elements are given in [Table 3.25](#).

Table 3.25: Nodal shape functions and derivatives for triangular elements.

node	N	$\frac{\partial N}{\partial \xi}$	$\frac{\partial N}{\partial \eta}$
1	ξ	1	0
2	η	0	1
3	$1 - \xi - \eta$	-1	-1

The surface integrals are tricky because there is no surface that lays on a line of constant ξ or η . Along Face 1, $1/2 > \xi > 1/3$ and $1/2 > \eta > 1/3$. Along Face 2, $1/3 > \xi > 0$ and $1/3 < \eta < 1/2$. Along Face 3, $1/3 < \xi < 1/2$ and $1/3 > \eta > 0$. The integrations are taken from the centroid to the element edges.

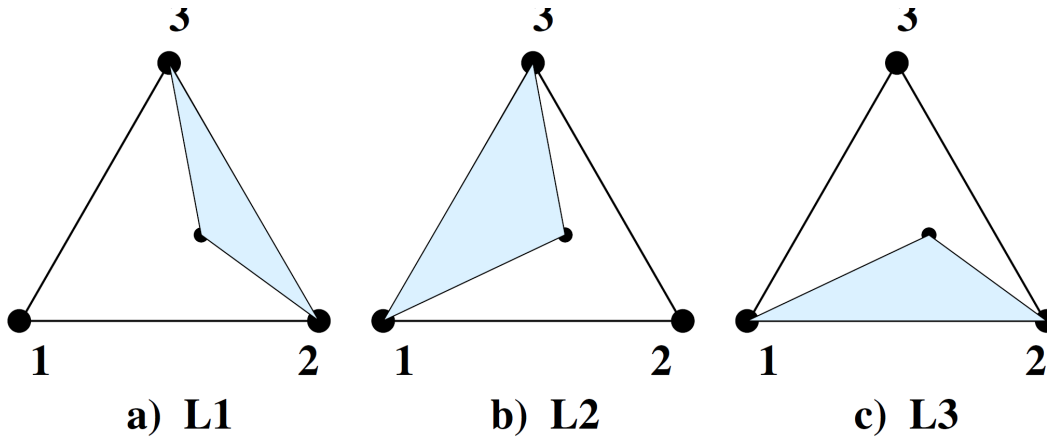


Fig. 3.35: Triangular natural coordinate system, shaded area corresponds to opposite node.

The values of the element variables and the surface differentials at the control-volume faces are given in Table 3.26.

Table 3.26: Element variable values and differentials at control-volume faces for triangular elements

face	Edge (Node _{out} → Node _{in})	ξ	η	$d\xi$	$d\eta$
1	1 → 2	$\frac{5}{12}$	$\frac{5}{12}$	$\frac{1}{6}$	$-\frac{1}{6}$
2	2 → 3	$\frac{1}{6}$	$\frac{5}{12}$	$-\frac{1}{3}$	$-\frac{1}{6}$
3	3 → 1	$\frac{5}{12}$	$\frac{1}{6}$	$\frac{1}{6}$	$\frac{1}{3}$

The form of the volume differentials are the same as with the quadrilateral elements. For volume integrals, quadrature points and differential values are shown in Table 3.27.

Table 3.27: Element variable values and differentials at sub-control volume centers for triangular elements.

sub-volume	ξ	η	$d\xi d\eta$
1	$\frac{7}{12}$	$\frac{5}{24}$	$\frac{1}{6}$
2	$\frac{5}{24}$	$\frac{7}{12}$	$\frac{1}{6}$
3	$\frac{5}{24}$	$\frac{5}{24}$	$\frac{1}{6}$

Hexahedral Elements

For hexahedral elements, there are eight nodes and twelve subfaces defining control volumes, shown in Figure 3.36. The shape functions are trilinear functions of the element variables, ξ , η , and ζ . The shape functions and derivatives at each node are given in Table 3.28.

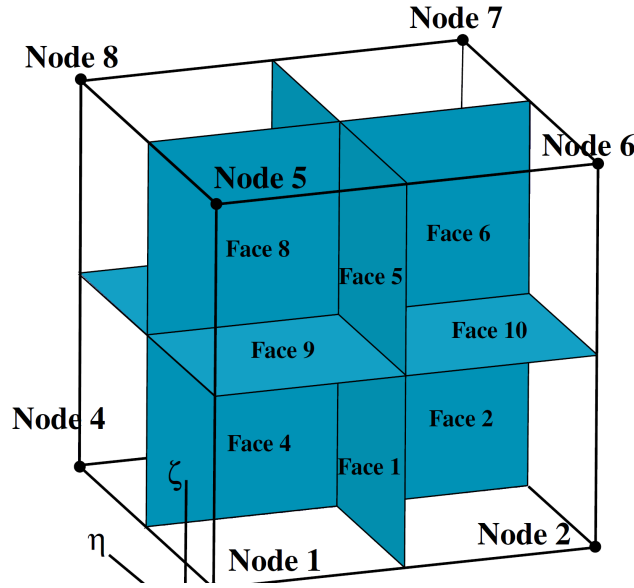


Fig. 3.36: Hexahedral element topology and numbering

Table 3.28: Nodal shape functions and derivatives for hexahedral elements.
Range is (-1,1).

node	N	$\frac{\partial N}{\partial \xi}$	$\frac{\partial N}{\partial \eta}$	$\frac{\partial N}{\partial \zeta}$
1	$\frac{1}{8} (1 - \xi) (1 - \eta) (1 - \zeta)$	$-\frac{1}{8} (1 - \eta) (1 - \zeta)$	$-\frac{1}{8} (1 - \xi) (1 - \zeta)$	$-\frac{1}{8} (1 - \xi) (1 - \eta)$
2	$\frac{1}{8} (1 + \xi) (1 - \eta) (1 - \zeta)$	$\frac{1}{8} (1 - \eta) (1 - \zeta)$	$-\frac{1}{8} (1 + \xi) (1 - \zeta)$	$-\frac{1}{8} (1 + \xi) (1 - \eta)$
3	$\frac{1}{8} (1 + \xi) (1 + \eta) (1 - \zeta)$	$\frac{1}{8} (1 + \eta) (1 - \zeta)$	$\frac{1}{8} (1 + \xi) (1 - \zeta)$	$-\frac{1}{8} (1 + \xi) (1 + \eta)$
4	$\frac{1}{8} (1 - \xi) (1 + \eta) (1 - \zeta)$	$-\frac{1}{8} (1 + \eta) (1 - \zeta)$	$\frac{1}{8} (1 - \xi) (1 - \zeta)$	$-\frac{1}{8} (1 - \xi) (1 + \eta)$
5	$\frac{1}{8} (1 - \xi) (1 - \eta) (1 + \zeta)$	$-\frac{1}{8} (1 - \eta) (1 + \zeta)$	$-\frac{1}{8} (1 - \xi) (1 + \zeta)$	$\frac{1}{8} (1 - \xi) (1 - \eta)$
6	$\frac{1}{8} (1 + \xi) (1 - \eta) (1 + \zeta)$	$\frac{1}{8} (1 - \eta) (1 + \zeta)$	$-\frac{1}{8} (1 + \xi) (1 + \zeta)$	$\frac{1}{8} (1 + \xi) (1 - \eta)$
7	$\frac{1}{8} (1 + \xi) (1 + \eta) (1 + \zeta)$	$\frac{1}{8} (1 + \eta) (1 + \zeta)$	$\frac{1}{8} (1 + \xi) (1 + \zeta)$	$\frac{1}{8} (1 + \xi) (1 + \eta)$
8	$\frac{1}{8} (1 - \xi) (1 + \eta) (1 + \zeta)$	$-\frac{1}{8} (1 + \eta) (1 + \zeta)$	$\frac{1}{8} (1 - \xi) (1 + \zeta)$	$\frac{1}{8} (1 - \xi) (1 + \eta)$

The control volume sub-face numbering, shown in Table 3.29, follows the convention that the face has the same number as the element edge that connects the nodes that define the two adjacent sub-control volumes.

The surface integrals require the vector of differential surface area components, (dA_x, dA_y, dA_z) , which is equivalent to the differential surface area dS multiplied by the unit surface normal vector n_i . Since the control volume surfaces are constructed using four points within an element, it is noted that assuming the surfaces are planar results in an error. Sometimes this error is deemed acceptable, and a faster algorithm is used to compute the surface area and volume. However, when strict conservation is required, an exact algorithm using a polyhedral decomposition is employed to compute the exact volume and surface area. These are detailed below.

Volume and Area Calculation Assuming Planar Surfaces

The differential surface area, $n_i dS$, is calculated in parametric space by taking the cross-product of two differential surface-tangent vectors. Let the surface be described by the collection of points $S(x, y, z)$. For example, a tangent vector in the ξ -direction is $\partial x_i / \partial \xi$. The normal surface area component for all three possible surface parameterizations is

$$n_k dS = \left[\frac{\partial x_i}{\partial \eta} \epsilon_{ijk} \frac{\partial x_j}{\partial \zeta} d\eta d\zeta \right] + \left[\frac{\partial x_i}{\partial \xi} \epsilon_{ijk} \frac{\partial x_j}{\partial \eta} d\xi d\eta \right] + \left[\frac{\partial x_i}{\partial \zeta} \epsilon_{ijk} \frac{\partial x_j}{\partial \xi} d\zeta d\xi \right] \quad (3.1132)$$

where ϵ_{ijk} is the alternating unit tensor and defines the cross product.

$$\epsilon_{ijk} = \begin{cases} 1 & \text{if } ijk \text{ equals an even permutation } 123, 231, \text{ or } 312 \\ 0 & \text{if } ijk \text{ contains a repeated index} \\ -1 & \text{if } ijk \text{ equals an odd permutation } 132, 213, \text{ or } 321 \end{cases} \quad (3.1133)$$

Using a shortened notation,

$$n_i dS = \left[\left| \frac{\partial(y,z)}{\partial(\eta,\zeta)} \right| \left| \frac{\partial(z,x)}{\partial(\eta,\zeta)} \right| \left| \frac{\partial(x,y)}{\partial(\eta,\zeta)} \right| \right] d\eta d\zeta \quad (3.1134)$$

$$+ \left[\left| \frac{\partial(y,z)}{\partial(\zeta,\xi)} \right| \left| \frac{\partial(z,x)}{\partial(\zeta,\xi)} \right| \left| \frac{\partial(x,y)}{\partial(\zeta,\xi)} \right| \right] d\zeta d\xi \quad (3.1135)$$

$$+ \left[\left| \frac{\partial(y,z)}{\partial(\xi,\eta)} \right| \left| \frac{\partial(z,x)}{\partial(\xi,\eta)} \right| \left| \frac{\partial(x,y)}{\partial(\xi,\eta)} \right| \right] d\xi d\eta, \quad (3.1136)$$

where the Jacobian notation is defined by

$$\frac{\partial(x,y)}{\partial(\xi,\eta)} = \begin{bmatrix} x_\xi x_\eta \\ y_\xi y_\eta \end{bmatrix}. \quad (3.1137)$$

The values of the element variables and the surface differentials at the control-volume faces are given in [Table 3.29](#).

Volume integrals require the volume differential, $dx dy dz$. In terms of the element parameters, the volume differential is

$$dx dy dz = J d\xi d\eta d\zeta, \quad (3.1138)$$

where

$$J = x_\xi y_\eta z_\zeta - x_\zeta y_\eta z_\xi \quad (3.1139)$$

$$+ x_\eta y_\zeta z_\xi - x_\xi y_\zeta z_\eta \quad (3.1140)$$

$$+ x_\zeta y_\xi z_\eta - x_\eta y_\xi z_\zeta \quad (3.1141)$$

The quadrature points and differential values are shown in [Table 3.30](#).

Table 3.29: Element variable values and differentials at control-volume faces for hexahedral elements. Face-to-edge number mapping.

face	Edge (Node _{out} → Node _{in})	ξ	η	ζ	$d\eta d\zeta$	$d\zeta d\xi$	$d\xi d\eta$
1	1 → 2	0	$-\frac{1}{2}$	$-\frac{1}{2}$	1	0	0
2	2 → 3	$\frac{1}{2}$	0	$-\frac{1}{2}$	0	1	0
3	3 → 4	0	$\frac{1}{2}$	$-\frac{1}{2}$	-1	0	0
4	1 → 4	$-\frac{1}{2}$	0	$-\frac{1}{2}$	0	1	0
5	5 → 6	0	$-\frac{1}{2}$	$\frac{1}{2}$	1	0	0
6	6 → 7	$\frac{1}{2}$	0	$\frac{1}{2}$	0	1	0
7	7 → 8	0	$\frac{1}{2}$	$\frac{1}{2}$	-1	0	0
8	5 → 8	$-\frac{1}{2}$	0	$\frac{1}{2}$	0	1	0
9	1 → 5	$-\frac{1}{2}$	$-\frac{1}{2}$	0	0	0	1
10	2 → 6	$\frac{1}{2}$	$-\frac{1}{2}$	0	0	0	1
11	3 → 7	$\frac{1}{2}$	$\frac{1}{2}$	0	0	0	1
12	4 → 8	$-\frac{1}{2}$	$\frac{1}{2}$	0	0	0	1

Table 3.30: Element variable values and differentials at sub-control volume centers for hexahedral elements.

sub-volume	ξ	η	ζ	$d\xi d\eta d\zeta$
1	$-\frac{1}{2}$	$-\frac{1}{2}$	$-\frac{1}{2}$	1
2	$\frac{1}{2}$	$-\frac{1}{2}$	$-\frac{1}{2}$	1
3	$\frac{1}{2}$	$\frac{1}{2}$	$-\frac{1}{2}$	1
4	$-\frac{1}{2}$	$\frac{1}{2}$	$-\frac{1}{2}$	1
5	$-\frac{1}{2}$	$-\frac{1}{2}$	$\frac{1}{2}$	1
6	$\frac{1}{2}$	$-\frac{1}{2}$	$\frac{1}{2}$	1
7	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$	1
8	$-\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$	1

Exact Volume and Surface Area Calculation

When the planar surface assumption for the control volumes is insufficient, the volume and surface areas can be calculated exactly. To accomplish this, a set of subcontrol points is constructed that defines the subcontrol surfaces. The locations and numbering of these subcontrol points are shown in [Figure 3.37](#). The coordinates of the edge points are the average of the two adjacent vertices, the coordinates of the facial points are the average of the four vertices defining the face, and the coordinates of the interior point is the average of the eight vertices defining the volume.

The 12 subcontrol surfaces for the Hexahedron are the defined using points in counterclockwise ordering as shown in [Table 3.31](#). These surfaces are further broken down into four triangles defined by the four points on the surface and a simply averaged midpoint. The four triangles are defined by points {5, 1, 2}, {5, 2, 3}, {5, 3, 4}, and {5, 4, 1}, respectively. The area vectors of each

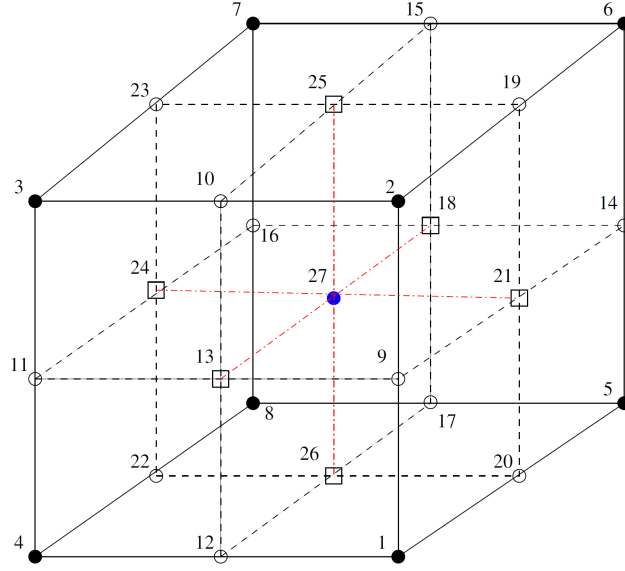


Fig. 3.37: Hexahedron subcontrol points numbering

triangle are summed to calculate the total surface area vector. Noting that the triangles are planar, the area vector of each triangle is calculated exactly using half the cross product of any two right-hand oriented vectors.

Table 3.31: Subcontrol face definitions for exact surface area calculation on hexahedral elements.

Face	Point Set			
1	21	9	13	27
2	25	10	13	27
3	11	13	27	24
4	12	26	27	13
5	14	21	27	18
6	18	15	25	27
7	18	16	24	27
8	17	18	27	26
9	20	21	27	26
10	21	19	25	27
11	23	24	27	25
12	22	26	27	24

The eight subcontrol volumes are defined using the points shown in Table 3.32. The formula to calculate the exact volume is based on the Gauss Divergence formula,

$$V = \int_{\Omega} dV = \int_{\Omega} \frac{\partial x_k}{\partial x_k} dV = \oint_{\partial\Omega} x_k n_k dS. \quad (3.1142)$$

The surfaces for the surface integral are decomposed into triangular facets as in the surface area calculation. To accomplish the decomposition, the coordinates on each face are averaged to the

midpoints, and thus each hexahedral volume is constructed using 14 coordinates—eight vertices and six facial midpoints, resulting in 24 total facets. Since the triangular facets are planar, the normal is constant over the surface. Thus, the surface integral over each triangular facet is equivalent to the scalar product of the outward facing normal area vector and the centroid coordinates, \bar{x} . The total surface integral is the sum of the integrals on each triangular facet,

$$V = \oint_{\partial\Omega} x_k n_k dS = \sum_{i=1}^{24} n_k \int_{\Delta_i} x_k dS = \sum_{i=1}^{24} \bar{x}_k n_k \int_{\Delta_i} dS, \quad \partial\Omega = \bigoplus_{i=1}^{24} \Delta_i. \quad (3.1143)$$

The area vectors are calculated as described above. The centroid coordinates are simply the average of the three vertices constructed the triangular facet.

Table 3.32: Subcontrol volume definitions for exact volume calculation on hexahedral elements.

Volume	Point Set							
1	1	9	13	12	20	21	27	26
2	9	2	10	13	21	19	25	27
3	13	10	3	11	27	25	23	24
4	12	13	11	4	26	27	24	22
5	20	21	27	26	5	14	18	17
6	21	19	25	27	14	6	15	18
7	27	25	23	24	18	15	7	16
8	26	27	24	22	17	18	16	8

Tetrahedral Elements

For tetrahedral elements, there are four nodes and six subfaces defining control volumes, shown in [Figure 3.38](#). The parametric coordinate system is defined by the tetrahedron natural coordinates, L_1, L_2, L_3 , and L_4 , since a Cartesian mapping cannot be defined. The natural coordinates are the shape functions. As an example, the value of L_1 at an interpolation point is the shape function associated with Node 2. The value of L_1 is the fraction of element tetrahedral volume covered by a sub-tetrahedron, formed by the interpolation point and the face opposite of Node 2. For consistency with the hexahedral element notation, the (ξ, η, ζ) parametric variables are defined as $\xi = L_1, \eta = L_2$, and $\zeta = L_3$, where L_4 is defined by the fact that the natural coordinates always sum to one.

Table 3.33: Nodal shape functions and derivatives for tetrahedral elements.
Range is (0,1).

node	N	$\frac{\partial N}{\partial \xi}$	$\frac{\partial N}{\partial \eta}$	$\frac{\partial N}{\partial \zeta}$
1	$1 - \xi - \eta - \zeta$	-1	-1	-1
2	ξ	1	0	0
3	η	0	1	0
4	ζ	0	0	1

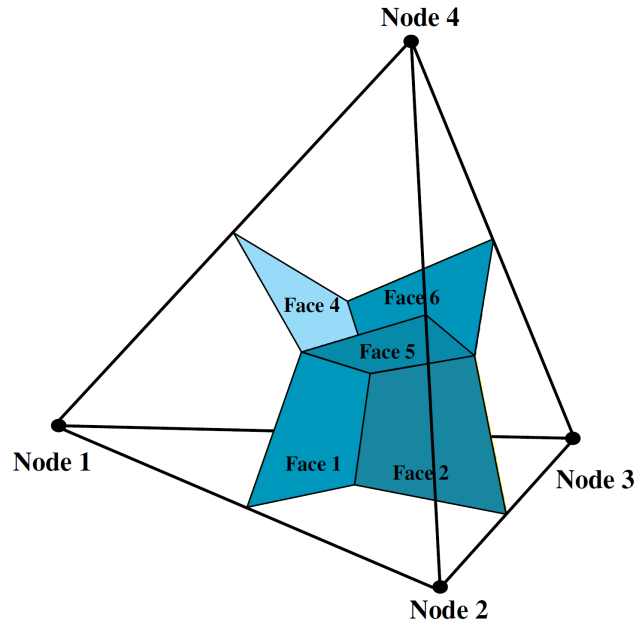


Fig. 3.38: Tetrahedral element topology and numbering

The control volume sub-face numbering, shown in Table 3.34, follows the convention that the face has the same number as the element edge that connects the nodes that define the two adjacent sub-control volumes.

The values of the element variables and the surface differentials at the control-volume faces are given in Table 3.34.

Table 3.34: Element variable values and differentials at control-volume faces for tetrahedral elements. Face-to-edge number mapping.

face	Edge (Node _{out} → Node _{in})	ξ	η	ζ	$d\eta d\zeta$	$d\zeta d\xi$	$d\xi d\eta$
1	1 → 2	$\frac{17}{48}$	$\frac{7}{48}$	$\frac{7}{48}$			
2	2 → 3	$\frac{17}{48}$	$\frac{17}{48}$	$\frac{7}{48}$			
3	1 → 3	$\frac{7}{48}$	$\frac{17}{48}$	$\frac{7}{48}$			
4	1 → 4	$\frac{7}{48}$	$\frac{7}{48}$	$\frac{17}{48}$			
5	2 → 4	$\frac{17}{48}$	$\frac{7}{48}$	$\frac{17}{48}$			
6	3 → 4	$\frac{7}{48}$	$\frac{17}{48}$	$\frac{17}{48}$			

Again the control volumes are constructed using surfaces defined with four points and two methods are available to define the surface area and volume.

Volume and Area Calculation Assuming Planar Surfaces

The form of the volume differentials are the same as with the hexahedral elements. For volume integrals, quadrature points and differential values are shown in [Table 3.35](#).

Table 3.35: Element variable values and differentials at sub-control volume centers for tetrahedral elements.

sub-volume	ξ	η	ζ	$d\xi d\eta d\zeta$
1	$\frac{17}{96}$	$\frac{17}{96}$	$\frac{17}{96}$	
2	$\frac{45}{96}$	$\frac{17}{96}$	$\frac{17}{96}$	
3	$\frac{17}{96}$	$\frac{45}{96}$	$\frac{17}{96}$	
4	$\frac{17}{96}$	$\frac{17}{96}$	$\frac{45}{96}$	

Exact Volume and Surface Area Calculation

Following the approach in *Exact Volume and Surface Area Calculation*, a set of subcontrol coordinates is defined to decompose the tetrahedral element, which are shown in [Figure 3.39](#).

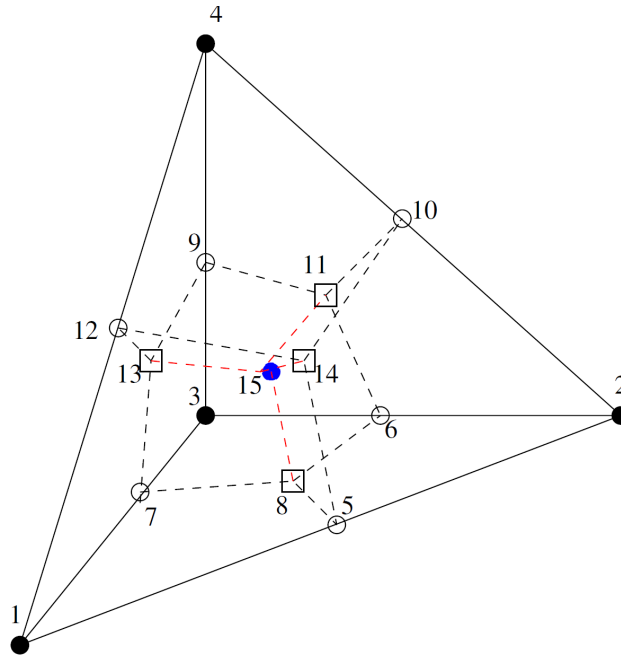


Fig. 3.39: Tetrahedron subcontrol points numbering

Six subcontrol surfaces for the tetrahedron are defined using points in counterclockwise ordering as shown in [Table 3.36](#). Surface area vectors are calculated using the same approach as in *Exact Volume and Surface Area Calculation*.

Table 3.36: Subcontrol face definitions for exact surface area calculation on tetrahedral elements.

Face	Point Set			
1	5	8	15	14
2	8	15	11	6
3	7	13	15	8
4	12	14	15	13
5	14	10	11	15
6	11	9	13	15

Four subcontrol volumes are defined using the points shown in Table 3.37. Since the subcontrol volumes are hexahedrons, the same volume calculation is used as above.

Table 3.37: Subcontrol volume definitions for exact volume calculation on tetrahedral elements.

Volume	Point Set							
1	1	5	8	7	12	14	15	13
2	2	6	8	5	10	11	15	14
3	3	7	8	6	9	13	15	11
4	4	10	14	12	9	11	15	13

Wedge Elements

For wedge elements, there are six nodes and nine subfaces defining control volumes. The parametric coordinate system is a linear hybrid of triangular natural coordinates. The natural coordinates are the shape functions. The local coordinates ξ and η are in the plane of the triangular surfaces while ζ is in the normal direction.

Table 3.38: Nodal shape functions and derivatives for wedge elements. Range is (0,1) and (-1,1).

node	N	$\frac{\partial N}{\partial \xi}$	$\frac{\partial N}{\partial \eta}$	$\frac{\partial N}{\partial \zeta}$
1	$\frac{1}{2}(1 - \xi - \eta)(1 - \zeta)$	$-\frac{1}{2}(1 - \zeta)$	$-\frac{1}{2}(1 - \zeta)$	$-\frac{1}{2}(1 - \xi - \eta)$
2	$\frac{1}{2}\xi(1 - \zeta)$	$\frac{1}{2}(1 - \zeta)$	0	$-\frac{1}{2}\xi$
3	$\frac{1}{2}\eta(1 - \zeta)$	0	$\frac{1}{2}(1 - \zeta)$	$-\frac{1}{2}\eta$
4	$\frac{1}{2}(1 - \xi - \eta)(1 + \zeta)$	$-\frac{1}{2}(1 + \zeta)$	$-\frac{1}{2}(1 + \zeta)$	$\frac{1}{2}(1 - \xi - \eta)$
5	$\frac{1}{2}\xi(1 + \zeta)$	$\frac{1}{2}(1 + \zeta)$	0	$\frac{1}{2}\xi$
6	$\frac{1}{2}\eta(1 + \zeta)$	0	$\frac{1}{2}(1 + \zeta)$	$\frac{1}{2}\eta$

The control volume sub-face numbering, shown in Table 3.39, follows the convention that the face has the same number as the element edge that connects the nodes that define the two adjacent sub-control volumes.

Volume and Area Calculation Assuming Planar Surfaces

The values of the element variables and the surface differentials at the control-volume faces are given in Table 3.39.

Table 3.39: Element variable values and differentials at control-volume faces for wedge elements. Face-to-edge number mapping.

face	Edge (Node _{out} → Node _{in})	ξ	η	ζ	$d\eta d\zeta$	$d\zeta d\xi$	$d\xi d\eta$
1	1 → 2	$\frac{5}{12}$	$\frac{1}{6}$	$-\frac{1}{2}$			
2	2 → 3	$\frac{5}{12}$	$\frac{5}{12}$	$-\frac{1}{2}$			
3	1 → 3	$\frac{1}{6}$	$\frac{5}{12}$	$-\frac{1}{2}$			
4	4 → 5	$\frac{5}{12}$	$\frac{1}{6}$	$\frac{1}{2}$			
5	5 → 6	$\frac{5}{12}$	$\frac{5}{12}$	$\frac{1}{2}$			
6	4 → 6	$\frac{1}{6}$	$\frac{5}{12}$	$\frac{1}{2}$			
7	1 → 4	$\frac{5}{24}$	$\frac{5}{24}$	0			
8	2 → 5	$\frac{7}{12}$	$\frac{5}{24}$	0			
9	3 → 6	$\frac{5}{24}$	$\frac{7}{12}$	0			

For volume integrals, quadrature points and differential values are shown in Table 3.40.

Table 3.40: Element variable values and differentials at sub-control volume centers for wedge elements.

sub-volume	ξ	η	ζ	$d\xi d\eta d\zeta$
1	$\frac{5}{24}$	$\frac{5}{24}$	$-\frac{1}{2}$	
2	$\frac{7}{12}$	$\frac{5}{24}$	$-\frac{1}{2}$	
3	$\frac{5}{24}$	$\frac{7}{12}$	$-\frac{1}{2}$	
4	$\frac{5}{24}$	$\frac{5}{24}$	$\frac{1}{2}$	
5	$\frac{7}{12}$	$\frac{5}{24}$	$\frac{1}{2}$	
6	$\frac{5}{24}$	$\frac{7}{12}$	$\frac{1}{2}$	

Exact Volume and Surface Area Calculation

Following the approach in *Exact Volume and Surface Area Calculation*, a set of subcontrol coordinates is defined to decompose the wedge element, which are shown in Figure 3.40.

Nine subcontrol surfaces for the tetrahedron are the defined using points in counterclockwise ordering as shown in Table 3.41. Surface area vectors are calculated using the same approach as in *Exact Volume and Surface Area Calculation*.

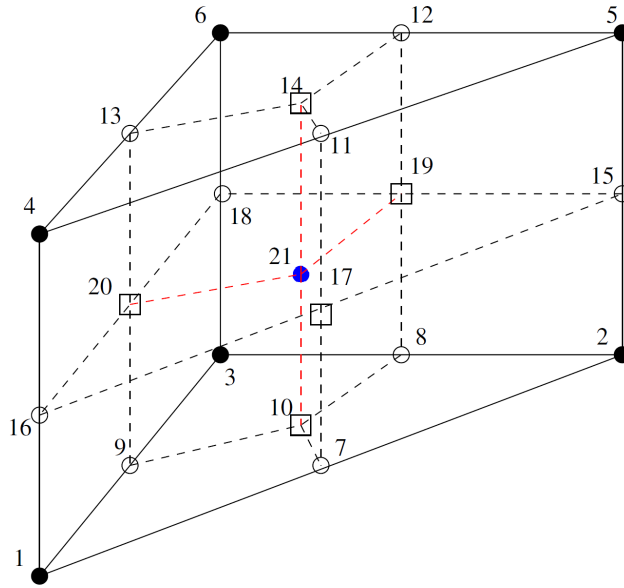


Fig. 3.40: Wedge subcontrol points numbering

Table 3.41: Subcontrol face definitions for exact surface area calculation on wedge elements.

Face	Point Set			
1	7	10	21	17
2	8	10	21	19
3	10	9	20	21
4	11	17	21	14
5	14	12	19	21
6	13	14	21	20
7	16	17	21	20
8	17	15	19	21
9	20	21	19	18

Six subcontrol volumes are defined using the points shown in Table 3.42. Since the subcontrol volumes are hexahedrons, the same volume calculation is used as above.

Table 3.42: Subcontrol volume definitions for exact volume calculation on wedge elements.

Volume	Point Set							
1	1	16	17	6	9	20	21	10
2	10	7	2	7	21	17	15	19
3	9	10	8	2	20	21	19	18
4	20	16	17	21	13	4	10	14
5	21	17	15	19	14	11	4	12
6	20	21	19	18	13	14	11	6

Pyramid Elements

For pyramid elements, there are five nodes and eight subfaces defining control volumes. The local coordinates ξ and η are in the plane of the quadrilateral surfaces while ζ is in the normal direction.

Table 3.43: Nodal shape functions and derivatives for pyramid elements. Range is (-1,1) and (0,1).

node	N	$\frac{\partial N}{\partial \xi}$	$\frac{\partial N}{\partial \eta}$	$\frac{\partial N}{\partial \zeta}$
1	$\frac{1}{4}(1 - \xi)(1 - \eta)(1 - \zeta)$	$-\frac{1}{4}(1 - \eta)(1 - \zeta)$	$-\frac{1}{4}(1 - \xi)(1 - \zeta)$	$-\frac{1}{4}(1 - \xi)(1 - \eta)$
2	$\frac{1}{4}(1 + \xi)(1 - \eta)(1 - \zeta)$	$\frac{1}{4}(1 - \eta)(1 - \zeta)$	$-\frac{1}{4}(1 + \xi)(1 - \zeta)$	$-\frac{1}{4}(1 + \xi)(1 - \eta)$
3	$\frac{1}{4}(1 + \xi)(1 + \eta)(1 - \zeta)$	$\frac{1}{4}(1 + \eta)(1 - \zeta)$	$\frac{1}{4}(1 + \xi)(1 - \zeta)$	$-\frac{1}{4}(1 + \xi)(1 + \eta)$
4	$\frac{1}{4}(1 - \xi)(1 + \eta)(1 - \zeta)$	$-\frac{1}{4}(1 + \eta)(1 - \zeta)$	$\frac{1}{4}(1 - \xi)(1 - \zeta)$	$-\frac{1}{4}(1 - \xi)(1 + \eta)$
5	ζ	0	0	1

The control volume sub-face numbering, shown in Table 3.44, follows the convention that the face has the same number as the element edge that connects the nodes that define the two adjacent sub-control volumes.

Volume and Area Calculation Assuming Planar Surfaces

The values of the element variables and the surface differentials at the control-volume faces are given in Table 3.44.

Table 3.44: Element variable values and differentials at control-volume faces for pyramid elements. Face-to-edge number mapping.

face	Edge (Node _{out} → Node _{in})	ξ	η	ζ	$d\eta d\zeta$	$d\zeta d\xi$	$d\xi d\eta$
1	1 → 2	0	$-\frac{45}{104}$	$\frac{7}{52}$			
2	2 → 3	$\frac{45}{104}$	0	$\frac{7}{52}$			
3	3 → 4	0	$\frac{45}{104}$	$\frac{7}{52}$			
4	1 → 4	$-\frac{45}{104}$	0	$\frac{7}{52}$			
5	1 → 5	$-\frac{7}{24}$	$-\frac{7}{24}$	$\frac{41}{120}$			
6	2 → 5	$\frac{7}{24}$	$-\frac{7}{24}$	$\frac{41}{120}$			
7	3 → 5	$\frac{7}{24}$	$\frac{7}{24}$	$\frac{41}{120}$			
8	4 → 5	$-\frac{7}{24}$	$\frac{7}{24}$	$\frac{41}{120}$			

For volume integrals, quadrature points and differential values are shown in Table 3.45.

Table 3.45: Element variable values and differentials at sub-control volume centers for pyramid elements.

sub-volume	ξ	η	ζ	$d\xi d\eta d\zeta$
1	$-\frac{19}{48}$	$-\frac{19}{48}$	$\frac{41}{240}$	
2	$\frac{19}{48}$	$-\frac{19}{48}$	$\frac{41}{240}$	
3	$\frac{19}{48}$	$\frac{19}{48}$	$\frac{41}{240}$	
4	$-\frac{19}{48}$	$\frac{19}{48}$	$\frac{41}{240}$	
5	0	0	$\frac{3}{5}$	

Exact Volume and Surface Area Calculation

It is noted here that for pyramid elements, the planar assumption is not good even on the reference element. The volume that composes the tip is an octohedron four planar faces and four highly skewed faces. Computations have shown that the planar assumption results in severe conservation errors.

Following the approach in *Exact Volume and Surface Area Calculation*, a set of subcontrol coordinates is defined to decompose the pyramid element, which are shown in Figure 3.41.

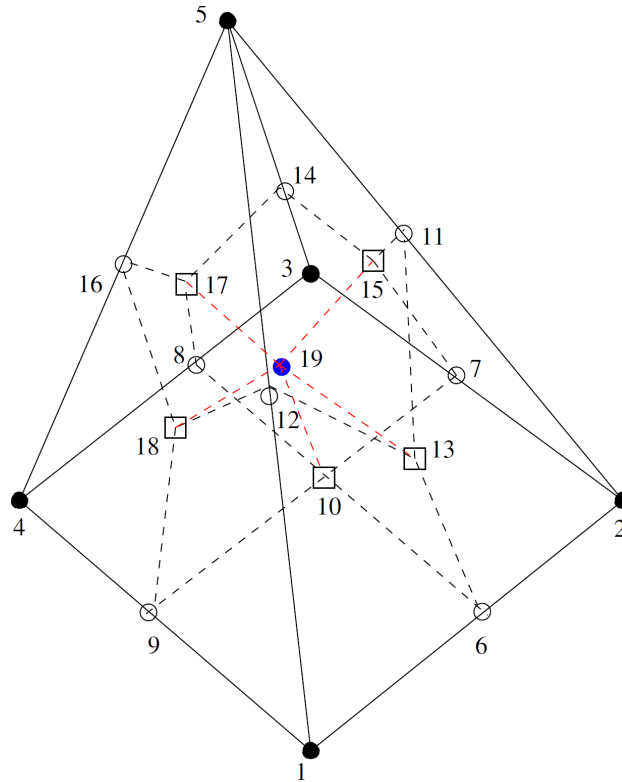


Fig. 3.41: Pyramid subcontrol points numbering

Eight subcontrol surfaces for the tetrahedron are defined using points in counterclockwise ordering as shown in Table 3.46. Surface area vectors are calculated using the same approach as

in *Exact Volume and Surface Area Calculation*.

Table 3.46: Subcontrol face definitions for exact surface area calculation on pyramid elements.

Face	Point Set			
1	6	10	19	13
2	7	10	19	15
3	8	10	19	17
4	9	18	19	10
5	12	13	19	18
6	11	15	19	13
7	14	17	19	15
8	16	18	19	17

Five subcontrol volumes are defined using the points shown in Table 3.47. The first four subcontrol volumes are hexahedrons, so the same volume calculation is used as above. The tip of the pyramid composes an octohedron, but the computation of the volume is only slightly different. The Gauss Divergence Theorem is still used to calculate the volume. However, because the four faces on the pyramid faces must be planar, these faces are decomposed into two triangles—composed of the face midpoint and the pyramid tip vertex—instead of four triangles. The four-point faces interior to the triangle are not planar and are decomposed into four triangles, resulting in 24 triangular facets total. Equation (3.1143) is then applied to compute the volume.

Table 3.47: Subcontrol volume definitions for exact volume calculation on pyramid elements.

Volume	Point Set									
1	1	6	10	9	12	13	19	18		
2	6	2	7	10	13	11	15	19		
3	7	3	8	10	15	14	17	19		
4	9	10	9	4	18	19	17	16		
5	5	19	16	18	12	13	11	15	14	17

Interpolation Functions and Negative Coefficients

A sufficient condition for a monotonic differencing scheme is that all the off-diagonal terms in the stencil be of opposite sign from the diagonal term [232]. Coefficient sets with mixed signs in the off-diagonal entries can potentially admit oscillatory solutions. In this section, the sign convention is that diagonal elements are negative and off-diagonal elements should be greater than or equal to zero. The term “negative coefficients” refers to one or more negative off-diagonal coefficients. Schemes with positive coefficients are usually considered important only when designing upwind convection operators, but they may be just as important for diffusion operators. Monotonic diffusion operators are most useful for artificial viscosity schemes and projection methods in application to the low Mach number Navier-Stokes equations. Positive coefficients are particularly

important for the Poisson equation that arises when calculating a velocity correction to the continuity equation. The computed field for the velocity potential should be smooth so that no oscillations are introduced into the pressure field.

Mixed-sign off-diagonal coefficients commonly arise in finite-element-like methods for describing the diffusion operator. Christie and Hall [233] note that applying the Galerkin finite-element method (GFEM) with bilinear quadrilateral elements to harmonic functions sometimes results in negative coefficients. It was later discovered that there is a threshold element aspect ratio for positivity, and negative coefficients are produced on meshes of rectangular elements above that threshold value. Several authors note that the threshold aspect ratio for the quadrilateral element is $\sqrt{2}$ with GFEM and the value is $\sqrt{3}$ with the control volume finite-element method (CVFEM) [141, 234, 235]. The values for the aspect ratio limits only strictly apply to orthogonal structured meshes. Notably, the five-point difference stencil for the 2D finite-difference method never generates negative coefficients. By deduction, the integral formulas that use extra stencil points, introduced by the element-based methods, generate negative coefficients.

A word on oscillations is required before continuing. Smooth solutions are possible with negative coefficients. Finite-element and finite-volume analysis codes for diffusion processes, such as conduction heat transfer, may never experience oscillations. A forcing function is required to induce the oscillations, like a boundary layer with the convection-diffusion equation [236] or an ill-behaved source term in the continuity equation. The mass balance for a control volume is the source term in the projection method. If the mass balance from a time integration step is particularly bad, the projection scheme must smooth large errors. If there are negative coefficients for the velocity potential, then resulting velocity potential field may be non-smooth, which causes the pressure field to be non-smooth. The result is oscillations which grow into the solution and make for a non-robust solution process.

Causes of negative-coefficients are being studied in order to design robust solution algorithms for the Navier-Stokes equations. The numerical method of primary interest is the CVFEM, though results for the GFEM are included for comparison. The GFEM community describes negative-coefficient effects as “hour-glassing”. The “hour-glass” oscillations are most common to reduced-integration formulations for the diffusion equation, and stabilization methods [130, 237] have been developed to damp the oscillations. In the CVFEM [158, 159], negative coefficients are prevented by shifting the integration points for the diffusion flux formulation out towards the edges of the control volumes and elements. The method is termed “integration point shifting” in this section. There is no general way to control the coefficient signs when skewed quadrilateral elements are used with arbitrary connectivities. Coefficient control is not a panacea for negative coefficients since integration point shifting generally reduces the accuracy. Ultimately, the proper mesh will have no negative coefficients at all.

In this section, the numerics behind negative coefficients are discussed for the diffusion operator given by

$$\int \frac{\partial \phi}{\partial x_i} n_i dS, \quad (3.1144)$$

where the surface differential is defined by (3.1134). Integration point shift functions are derived for the CVFEM diffusion operator. Shift functions are presented for both two and three dimensions

which guarantee coefficient positivity for a particular element aspect ratio. Also, the integration point shifting for CVFEM is shown to be similar to hour-glass stabilization for GFEM.

Positive Coefficients for Orthogonal Meshes

Negative coefficients arise in the off-diagonal coefficients when the aspect ratio of an element becomes large. Consider the elemental flux contributions to the control volume centered about Node 3, shown in Fig. *Control volume faces in a single element. Contributions to the the control volume centered about node 3.* The first off-diagonal node to have a negative coefficient is the side node farthest from the control volume center, Node 4. The negative coefficient is associated with the vertical flux over the long horizontal face. At the integration point on the long horizontal face, the flux is approximated by an average of the difference between Nodes 3 and 2 and the difference between Nodes 4 and 1. The weighting between the two differences is determined by the location of the integration point. The negative coefficient is removed by removing the influence from the Node 4–1 difference. The integration point is shifted farther from Nodes 4 and 1, towards Nodes 2 and 3. The integration points are shifted such that the element-level coefficients are positive, a sufficient condition for global positivity.

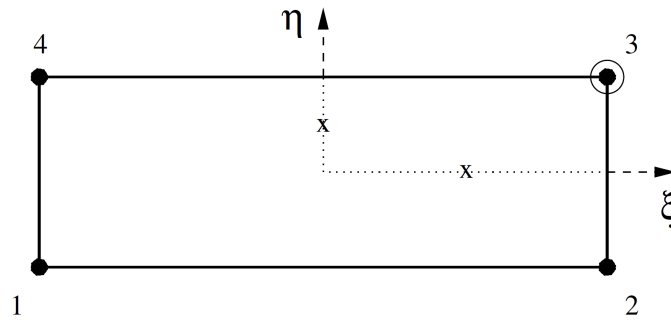


Fig. 3.42: Control volume faces in a single element. Contributions to the the control volume centered about node 3.

Integration point shift functions and the critical aspect ratio are derived for isoparametric quadrilateral elements with bilinear shape functions, and hexahedral elements with trilinear shape functions. Only the orthogonal form is considered. Linear triangles are discussed since they can also produce negative coefficients. For linear elements, only the element geometry (mesh quality) can be modified to control negative coefficients. With isoparametric bilinear and trilinear elements, both the geometry and the location of the integration point control negativity. In addition, integration point shifting is compared to finite-element hour-glass control. Positive coefficients are achieved by either shifting the element integration points or applying the hour-glass stabilization matrix, and in some cases the two are identical.

Aspect Ratio Definition

In this section, the isoparametric coordinates for an element are oriented such that the aspect ratio is greater than or equal to one. In two dimensions, the aspect ratio for an orthogonal element is the ratio between edge lengths. In three dimensions, each element has two aspect ratios since there are potentially three different edge lengths. The aspect ratios are taken relative to the shortest of the edge lengths which has a reference length of one.

Quadrilateral Elements

The coefficients for the diffusion operator result from the combination of two basic second-order accurate diffusion operators: the *edge* operator and the *centroid* operator, shown in Figure 3.43. The two operators represent the extremes in evaluating derivatives using the bilinear shape function within the element. The *edge* scheme always gives positive coefficients while the *centroid* scheme gives rise to negative coefficients above a certain aspect ratio. The *centroid* scheme results from evaluating all the derivatives for the four control volume sub-faces at the element centroid, equivalent to reduced integration [237] in GFEM. The *edge* scheme results from evaluating the derivatives out at the ends of the sub-faces in CVFEM or out at the nodes in GFEM. The *edge* scheme returns the standard five-point finite-difference operator. The traditional CVFEM [159] uses an equal weighting of the *edge* and the *centroid* scheme. The GFEM uses one part *edge* to two parts *centroid*.

The GFEM is more prone to oscillations with high aspect ratio elements than the CVFEM because it contains a larger weighting of the *centroid* scheme. The single-point-integrated GFEM element will be the most unstable since it is a pure *centroid* scheme.

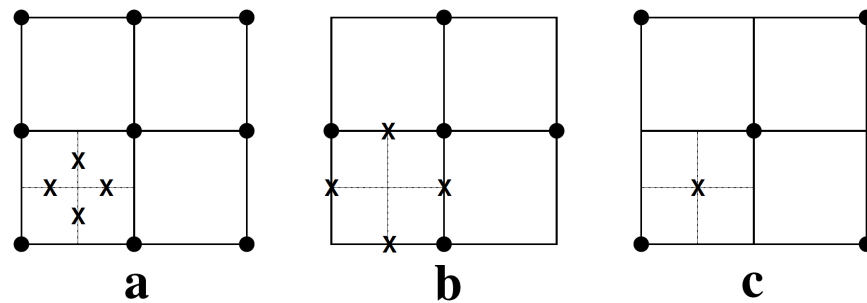


Fig. 3.43: Flux integration points (X) determine nodal (•) contributions to the coefficient stencil: a) mid-face rule of CVFEM, b) *edge*-operator, c) *centroid*-operator (one-point integration).

The coefficient signs for a rectangular element are controlled by moving the integration points away from the centroid of the element. The smallest value of the integration point shift that satisfies coefficient positivity is found by symbolically integrating the diffusion flux over a control volume. Consider the diffusion operator evaluated over a collection of equal-size rectangular elements. Each element is longer by a factor of \mathcal{AR} in the x-direction than the y-direction, where \mathcal{AR} is the aspect ratio of the elements. The integration points can be shifted in the ξ -direction by

s and in the η -direction by t . The element coefficients that contribute to the equation centered at Node 3 in Fig. *Control volume faces in a single element. Contributions to the the control volume centered about node 3.* are:

$$\phi_1 : \frac{1 + \mathcal{AR}^2 - 2s\mathcal{AR}^2 - 2t}{8\mathcal{AR}} \quad (3.1145)$$

$$\phi_2 : \frac{-1 + 3\mathcal{AR}^2 + 2s\mathcal{AR}^2 + 2t}{8\mathcal{AR}} \quad (3.1146)$$

$$\phi_3 : -\frac{3 + 3\mathcal{AR}^2 + 2s\mathcal{AR}^2 + 2t}{8\mathcal{AR}} \quad (3.1147)$$

$$\phi_4 : \frac{3 - \mathcal{AR}^2 + 2s\mathcal{AR}^2 + 2t}{8\mathcal{AR}} \quad (3.1148)$$

The positivity constraint for Node 3 comes from coefficient 4 in the element matrix, where the coefficient becomes negative for large values of aspect ratio. The s -shift removes the effect of aspect ratio, while the t -shift has no effect on negativity. For CVFEM, the integration points on vertical faces, in the longer x -direction, should be shifted from the mid-faces out towards the element edges by s . The y -direction flux is the only flux effected by the shift so the y -direction flux is the flux associated with negative coefficients. The minimal amount of s -shift required to maintain positivity depends upon the aspect ratio,

$$s > \frac{1}{2} \frac{\mathcal{AR}^2 - 3}{\mathcal{AR}^2}, \quad \mathcal{AR} > \sqrt{3}. \quad (3.1149)$$

The maximum aspect ratio for which the unshifted CVFEM remains monotone is $\sqrt{3}$. A similar formula exists for GFEM, where s and t are shifted from the Gauss points at $\pm 1/\sqrt{3}$. The element coefficients that contribute to the equation centered at Node 3 in Fig. *Control volume faces in a single element. Contributions to the the control volume centered about node 3.* are:

$$\phi_1 : \frac{3 + 3\mathcal{AR}^2 - \left(1 + \sqrt{3}s\right)^2 \mathcal{AR}^2 - \left(1 + \sqrt{3}t\right)^2}{12\mathcal{AR}} \quad (3.1150)$$

$$\phi_2 : \frac{-3 + 3\mathcal{AR}^2 + \left(1 + \sqrt{3}s\right)^2 \mathcal{AR}^2 + \left(1 + \sqrt{3}t\right)^2}{12\mathcal{AR}} \quad (3.1151)$$

$$\phi_3 : \frac{-3 - 3\mathcal{AR}^2 - \left(1 + \sqrt{3}s\right)^2 \mathcal{AR}^2 - \left(1 + \sqrt{3}t\right)^2}{12\mathcal{AR}} \quad (3.1152)$$

$$\phi_4 : \frac{3 - 3\mathcal{AR}^2 + \left(1 + \sqrt{3}s\right)^2 \mathcal{AR}^2 + \left(1 + \sqrt{3}t\right)^2}{12\mathcal{AR}} \quad (3.1153)$$

Similar to the CVFEM, the s -shift is the only shift that affects the aspect ratio term. The positivity constraint for Node 3 comes from coefficient 4 in the element matrix,

$$s > \sqrt{\frac{3\mathcal{AR}^2 - 4}{3\mathcal{AR}^2}} - \frac{1}{\sqrt{3}}, \quad \mathcal{AR} > \sqrt{2}. \quad (3.1154)$$

The maximum aspect ratio for which the unshifted GFEM remains monotone is $\sqrt{2}$.

The shift values are very sensitive to the aspect ratio, out to an aspect ratio of about four. The values of the integration point shift function are plotted as a function of the aspect ratio in Figure 3.44. At that aspect ratio, the shifted integration points are near the edge of the element. Since the requisite integration point shift rapidly reaches the element edge for increasing aspect ratio, it can be argued that the maximum shift should always be taken. It will be shown in the section on accuracy that integration point shifting leads to a general loss in accuracy on non-orthogonal meshes. Therefore, it may be desirable to use (3.1149) to compute the minimal shift for each element. The accuracy consideration must be traded against the algorithmic complexity of computing geometry-dependent shape functions for each element.

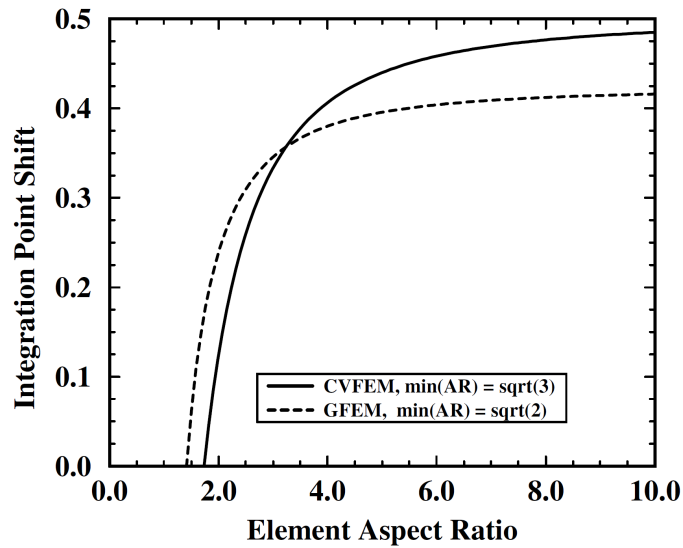


Fig. 3.44: Integration point locations must be shifted out towards the element-edge with increasing element aspect ratio.

Reduced Integration

One-point integration methods for quadrilateral and hexahedral elements are popular because they are computationally efficient. Sometimes, oscillations occur and are called hour-glass modes after the displaced element shapes. Hour-glass stabilization methods prevent the hour-glass oscillations from occurring. The hour-glass terms are derived by examining the eigenmodes of the finite-elements and noting that there are missing mode shapes when the elements are integrated at the center [237]. The stabilization term adds the effects of the missing mode shapes back into the element formulation.

It is shown here that the hour-glass stabilization terms have the same effect on the element coefficients as shifting the integration points in two-dimensional elements. Both the hour-glass stabilization and the integration-point shifting modify the discretization to look more like a five-point scheme; or, more like the *edge* scheme of the previous section. The H-stabilization

method is commonly used [130] in the GFEM with reduced integration. For quadrilateral elements, the element matrix for the hour-glass stabilization term is

$$C_{\text{hg}} \begin{bmatrix} -11 & -11 \\ 1 & -11 \\ -11 & -11 \\ 1 & -11 \end{bmatrix}, \quad (3.1155)$$

where the constant C_{hg} contains scaling information. The first row of the hour-glass stabilization matrix, (3.1155), can also be derived by subtracting the matrix coefficients for the one-point integrated GFEM, (3.1156),

$$\begin{aligned} \phi_1 &: \frac{1+\mathcal{A}\mathcal{R}^2}{4\mathcal{A}\mathcal{R}} \\ \phi_3 &: -\frac{1+\mathcal{A}\mathcal{R}^2}{4\mathcal{A}\mathcal{R}} \\ \phi_2 &: \frac{\mathcal{A}\mathcal{R}^2-1}{4\mathcal{A}\mathcal{R}} \\ \phi_4 &: \frac{1-\mathcal{A}\mathcal{R}^2}{4\mathcal{A}\mathcal{R}} \end{aligned} \quad (3.1156)$$

from the coefficients for the five-point difference scheme, (3.1157),

$$\begin{aligned} \phi_1 &: 0 \\ \phi_3 &: -\frac{1+\mathcal{A}\mathcal{R}^2}{2\mathcal{A}\mathcal{R}} \\ \phi_2 &: \frac{\mathcal{A}\mathcal{R}^2}{2\mathcal{A}\mathcal{R}} \\ \phi_4 &: \frac{1}{2\mathcal{A}\mathcal{R}} \end{aligned} \quad (3.1157)$$

The resulting coefficient set is identical to the hour-glass stabilization matrix, (3.1155), if the multiplier $C_{\text{hg}} = (1 + \mathcal{A}\mathcal{R}^2)/4\mathcal{A}\mathcal{R}$. The hour-glass stabilization matrix is added to a diffusion operator to make it look more like a five-point finite difference scheme. Note that the multiplier used by GFEM practitioners [130] is $C_{\text{hg}} = 1$. The hour-glass stabilization matrix can also be used with other schemes. The multiplier for standard GFEM is $C_{\text{hg}} = (1 + \mathcal{A}\mathcal{R}^2)/6\mathcal{A}\mathcal{R}$. The multiplier for standard CVFEM is $C_{\text{hg}} = (1 + \mathcal{A}\mathcal{R}^2)/8\mathcal{A}\mathcal{R}$, though conservation is only guaranteed on rectangular meshes.

Triangular Elements

Linear triangular elements can also produce negative off-diagonal coefficients.

There are no shift-functions for triangles since the gradients are constant over the element. Negative coefficients result from the geometry of the element.

Nodal coefficients for a triangular element are computed for the diffusion flux contributions, shown in Figure 3.45. The nodal coefficients for the diffusion flux contribution to the control volume centered about Node 1 are

$$\phi_1 : -\frac{1}{2} \frac{\tan \alpha}{\tan \beta} \frac{(1 + \tan^2 \beta)}{(\tan \alpha + \tan \beta)} \quad (3.1158)$$

$$\phi_2 : \frac{1}{2} \frac{(\tan \alpha \tan \beta - 1)}{(\tan \alpha + \tan \beta)} \quad (3.1159)$$

$$\phi_3 : \frac{1}{2} \frac{1}{\tan \beta} \quad (3.1160)$$

where the base edge length is r and the two adjacent vertex angles are α and β . The conditions required to ensure that all the off-diagonal terms remain positive are combined from constraints on all three control volume contributions,

$$\tan \alpha > 0 \quad (3.1161)$$

$$\tan \beta > 0 \quad (3.1162)$$

$$\tan \alpha \tan \beta > 1. \quad (3.1163)$$

The triangular element yields positive off-diagonal coefficients if $0 < \alpha < \pi/2$, $0 < \beta < \pi/2$, and $\pi/2 < \alpha + \beta < \pi$. The triangle must be acute.

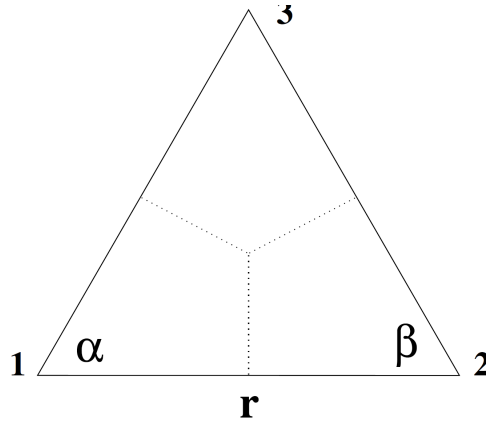


Fig. 3.45: Triangular element geometry in defined by edge length and two vertex angles.

In a previous work [204], quadrilateral elements were subdivided into triangular elements using edge-swapping, along with a Delaunay algorithm, to minimize the effect of negative coefficients. Given a collection of nodes, a Delaunay triangulation of the nodes will generate triangles where the minimum angle between vertices is maximized, leading to near-equilateral triangles that satisfy (3.1161).

Hex Elements

The aspect ratio limit for the CVFEM diffusion operator with orthogonal, hexahedral elements can be as large as $\sqrt{2}$ if the base is square. The three-dimensional element is more difficult to control because there are two aspect ratios. The local element node numbering for a hexahedron is defined in Fig. *Hexahedral element topology and numbering*. Let the edge length between Nodes 1 and 2 be of value A , the edge length between Nodes 1 and 4 be of value B , and the edge length between Nodes 1 and 5 be of value C . These are the ξ , η , and ζ directions. The parametric

representation of the nodal coefficients for the element contribution to the control volume centered about Node 1 is

$$\phi_1 : - \frac{9 (A^2 B^2 + B^2 C^2 + C^2 A^2)}{64 ABC} \quad (3.1164)$$

$$\phi_2 : \frac{3 (-A^2 B^2 + 3 B^2 C^2 - C^2 A^2)}{64 ABC} \quad (3.1165)$$

$$\phi_3 : \frac{3 (-A^2 B^2 + 3 B^2 C^2 + 3 C^2 A^2)}{64 ABC} \quad (3.1166)$$

$$\phi_4 : \frac{3 (-A^2 B^2 - B^2 C^2 + 3 C^2 A^2)}{64 ABC} \quad (3.1167)$$

$$\phi_5 : \frac{(3 A^2 B^2 - B^2 C^2 - C^2 A^2)}{64 ABC} \quad (3.1168)$$

$$\phi_6 : \frac{(3 A^2 B^2 + 3 B^2 C^2 - C^2 A^2)}{64 ABC} \quad (3.1169)$$

$$\phi_7 : \frac{(A^2 B^2 + B^2 C^2 + C^2 A^2)}{64 ABC} \quad (3.1170)$$

$$\phi_8 : \frac{(3 A^2 B^2 - B^2 C^2 + 3 C^2 A^2)}{64 ABC}. \quad (3.1171)$$

The region for positive coefficients is plotted in Fig. *Limits of Edge-Length Ratio for Positive Coefficients in 3D CVFEM*, which comes from examining coefficients for Nodes 2, 4, and 5. The maximum allowable aspect ratios occur for the case of a square base. If the base edges are longer than the vertical edge, then the maximum aspect ratio is $\sqrt{2}$. If the vertical edge is longer than a base edge, the maximum aspect ratio is $\sqrt{3/2}$.

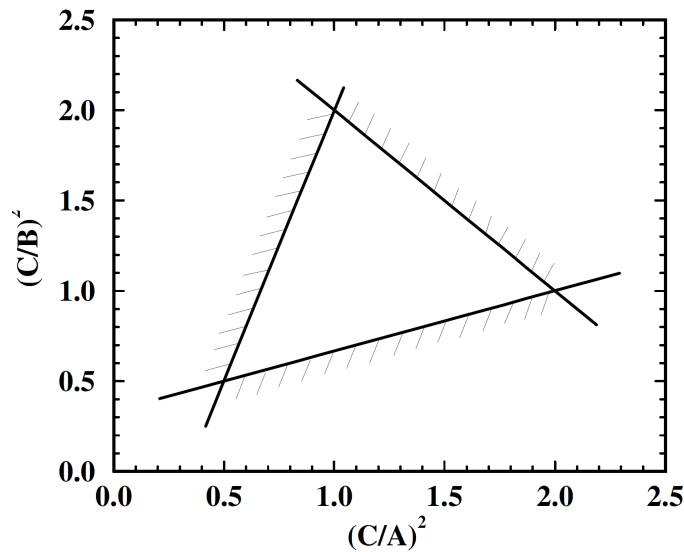


Fig. 3.46: Limits of Edge-Length Ratio for Positive Coefficients in 3D CVFEM.

The integration points in the CVFEM scheme can be shifted by s , t , and u in the ξ , η , and ζ directions. The conditions for positive coefficients are

$$\frac{1}{A^2} \frac{1}{1-2s} - \frac{1}{B^2} \frac{1}{3+2t} > \frac{1}{C^2} \frac{1}{3+2u} \quad (3.1172)$$

$$\frac{1}{B^2} \frac{1}{1-2t} - \frac{1}{C^2} \frac{1}{3+2u} > \frac{1}{A^2} \frac{1}{3+2s} \quad (3.1173)$$

$$\frac{1}{C^2} \frac{1}{1-2u} - \frac{1}{A^2} \frac{1}{3+2s} > \frac{1}{B^2} \frac{1}{3+2t} \quad (3.1174)$$

The relations are nonlinear and require iteration to extract the limiting values of s , t , and u .

The coefficients that are generated by the standard GFEM operator in three dimensions have no allowable maximum aspect ratio. The only element shape that does not have negative coefficients is a cube, and even then the coefficients of the six nearest nodes are zero. The parametric representation of the nodal coefficients for the element contribution to the equation associated with Node 1 are

$$\phi_1 : - \frac{4 (A^2 B^2 + B^2 C^2 + C^2 A^2)}{36ABC} \quad (3.1175)$$

$$\phi_2 : \frac{2 (-A^2 B^2 + 2B^2 C^2 - C^2 A^2)}{36ABC} \quad (3.1176)$$

$$\phi_3 : \frac{(-A^2 B^2 + 2B^2 C^2 + 2C^2 A^2)}{36ABC} \quad (3.1177)$$

$$\phi_4 : \frac{2 (-A^2 B^2 - B^2 C^2 + 2C^2 A^2)}{36ABC} \quad (3.1178)$$

$$\phi_5 : \frac{2 (2A^2 B^2 - B^2 C^2 - C^2 A^2)}{36ABC} \quad (3.1179)$$

$$\phi_6 : \frac{(2A^2 B^2 + 2B^2 C^2 - C^2 A^2)}{36ABC} \quad (3.1180)$$

$$\phi_7 : \frac{(A^2 B^2 + B^2 C^2 + C^2 A^2)}{36ABC} \quad (3.1181)$$

$$\phi_8 : \frac{(2A^2 B^2 - B^2 C^2 + 2C^2 A^2)}{36ABC}. \quad (3.1182)$$

The contributions from Nodes 2, 4, and 5 are such that there will be negative coefficients for any element shape other than a cube.

Single-point integration is also used for the GFEM hexahedral element. Unfortunately, there is not a clear analogy between the integration point shift and hour-glass stabilization in three

dimensions. The element matrix for the hour-glass stabilization term is

$$C_{hg} \begin{bmatrix} -4\alpha\alpha\alpha - 2\alpha \\ 2 - 4\alpha\alpha\alpha - 2 \\ \alpha\alpha - 4\alpha - 2\alpha\alpha \\ 2\alpha - 4\alpha - 2\alpha \\ 2\alpha - 2\alpha - 4\alpha\alpha \\ \alpha\alpha - 2\alpha - 4\alpha \\ -2\alpha\alpha\alpha - 4\alpha \\ 0 - 2\alpha\alpha\alpha - 4 \end{bmatrix}, \quad (3.1183)$$

where the constant C_{hg} contains scaling information. The hour-glass stabilization matrix does act to make the diagonal terms more negative, and the problematic off-diagonal terms more positive.

H-Adaptivity Meshing

Note: we currently only allow uniform refinement with no load balancing (12/01). We have not yet decided on a scheme for integrating fluxes over h-refined meshes. We have not yet decided on a prolongation approach for the mass flow rate at faces.

H-Adaptivity and Flux Construction

The equation assembly in our control volume method is based on integrating fluxes over control volume sub-faces within an element. A typical h-adapted patch of elements is shown in [Figure 3.47](#). The “hanging nodes” do not have control volumes associated with them. Rather, they are constrained to be a linear combination of the two parent edge nodes. There is no element assembly procedure to compute fluxes for the “hanging sub-faces” associated with the hanging nodes that occur along the parent-child element boundary.

One possibility is to create a sub-set of element faces that contain hanging-nodes. The fluxes across the hanging sub-faces can then be processed using local nodal information. This precludes computing localized gradients across the face.

The SIERRA h-adaptive scheme is driven at the element level. Refinement occurs within the element and the topology of refined elements is the same as the parent element. If the topology restriction was relaxed, then the following schemes could be used.

Aftosmis [238] describes a vertex-centered finite-volume scheme on unstructured Cartesian meshes. A transitional set of control volumes are formed about the hanging nodes, shown in [Figure 3.48](#), on unstructured meshes. (This would require a series of specialized master elements to deal with the different transition possibilities in SIERRA and would be a burden on the application teams.)

Kallinderis [239] describes a vertex-centered finite-volume scheme on unstructured quad meshes. Hanging nodes are treated with a constraint condition. The flux construction for a node on a

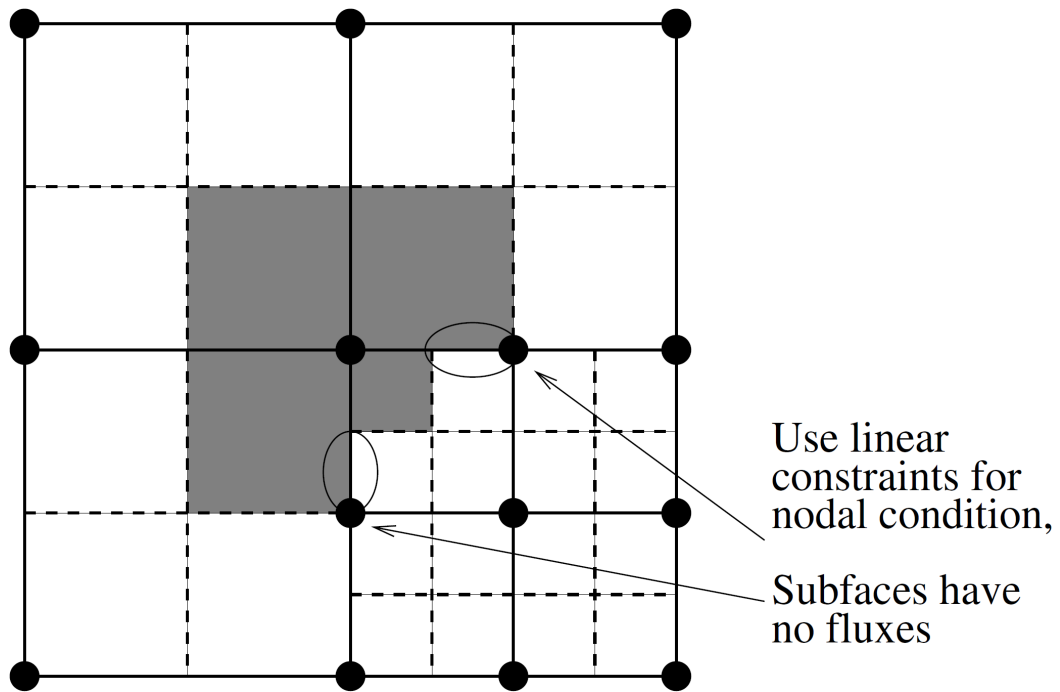


Fig. 3.47: Control volume definition on an h-adapted mesh with hanging nodes. (Four-patch of parent elements with refinement in bottom-right element.)

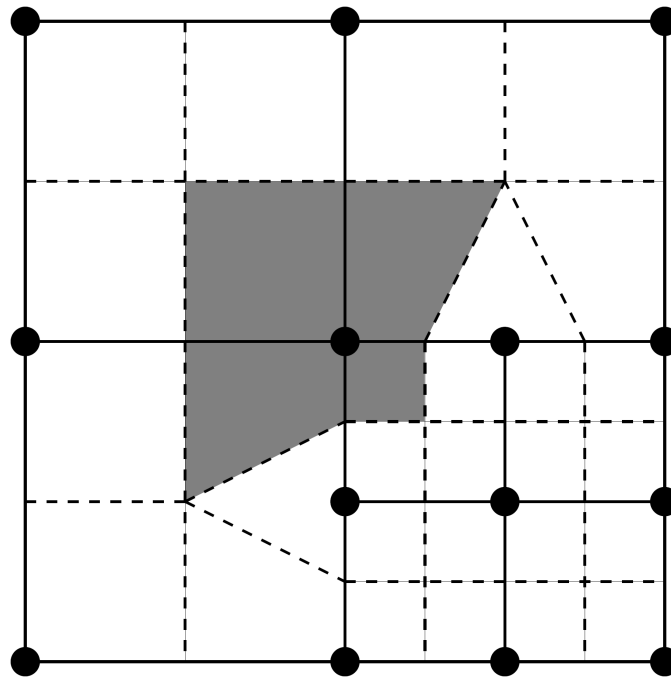


Fig. 3.48: Control volume definition on an h-adapted mesh with transition control volumes about the hanging nodes. (Four-patch of parent elements with refinement in bottom-right element.)

refinement boundary is based on the unrefined parent elements, leading to a non-conservative scheme.

Kallinderis [240] describes a vertex-centered finite-volume scheme on unstructured tetrahedral meshes. Hanging nodes are removed by splitting the elements on the “unrefined” side of the refinement boundary. Mavriplis [241] uses a similar technique, but extends it to a general set of heterogeneous elements, shown in Figure 3.49. (This would require a change to the topology rules in SIERRA as well as splitting elements along the refinement boundary, but there would be little impact on the application codes other than supporting heterogeneous meshes.)

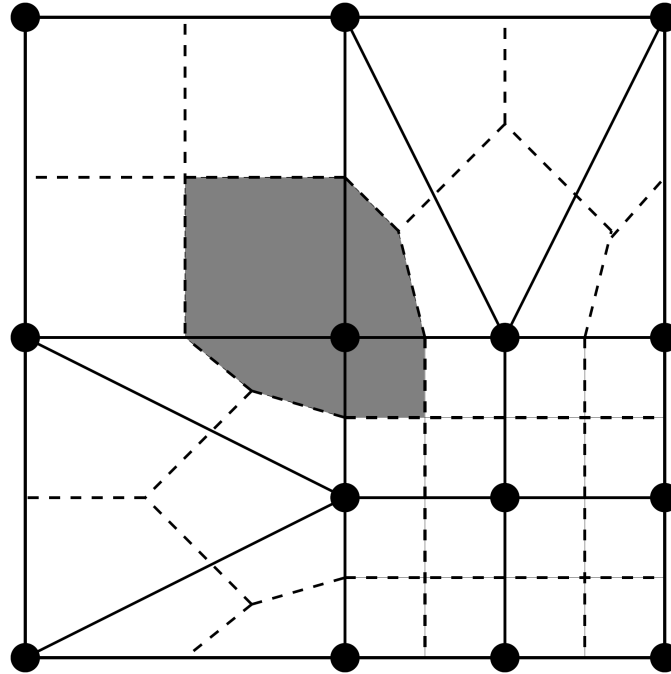


Fig. 3.49: Control volume definition on a heterogeneous h-adapted mesh with no hanging nodes. (Four-patch of parent elements with refinement in bottom-right element and splitting in adjacent parent elements.)

Prolongation and Restriction

Nodal variables are interpolated between levels of the h-adapted mesh hierarchy using the traditional prolongation and restriction operators defined over an element. The prolongation operation is used to compute values for new nodes that arise from element sub-division. The parent element shape functions are used to interpolate values from the parent nodes to the sub-divided nodes.

Prolongation and restriction operators for element variables and face variables are required to maintain mass flow rates that satisfy continuity.

Mass Continuity

Care must be taken to ensure continuity of mass between control volumes that contain hanging sub-faces. Especially since control-volume balances at hanging nodes are replaced by constraint conditions.

We need a list of the hanging faces as well as a means of identifying the hanging nodes on each face.

Nodal Gradients

The nodal gradients are approximated by integrating over the surface of the control volume and applying the discrete form of the Gauss divergence theorem. There are two possible approaches for dealing with the hanging sub-faces. In the first approach, the hanging sub-faces are processed separately. In the second approach, the sub-faces are ignored but the unclosed surface integral is corrected by a reference value, namely the nodal value associated with the control volume centroid,

$$\left. \frac{\partial \phi}{\partial x_i} \right|_p dV = \int (\phi - \phi_p) n_i dS \quad (3.1184)$$

Dynamic Load-Balancing

Dynamic load-balancing is required as the mesh is adaptively refined across parallel processors. Some processors may end up with more refined elements, so the work load increases. We will use the Zoltan dynamics load-balancing package to drive the load-balancing. We need a good measure of the compute load, most likely a combination of the time to assemble equations and the solve them.

4 Simulation Setup

4.1 Input File Basics

Anatomy of an Input Deck

Every Fuego simulation requires a text input file to define the problem. The input file defines everything about your problem except the mesh itself. You can prepare and edit this file in any text editor of your choosing. Using the Sierra Editor in SAW to prepare this file will provide additional context hints and syntax checking in real-time.

Indentation & Capitalization

Indentation is recommended but not required in an Fuego input file. You can indent with tabs or spaces, or choose not to indent things at all - although we highly recommend using indentation (with spaces) to make a consistently readable input file and to avoid errors with commands going in the wrong scope.

Fuego commands are generally not case sensitive, although commands referring to a file (e.g. the mesh file or an Aprepro include file) will be case sensitive.

Comments

Anything that starts with a # or a \$ will be treated as a comment and ignored. Long lines can be split using \\$, like this

```
# This is a comment
$ and so is this

# This is a single line command split on three lines
List of Blocks = block_1 block_2 block_3 \$
                  block_4 block_5 block_6 \$
                  block_7 block_8
```

Structure

The Fuego input file is structured using nested blocks with **Begin** and **End** commands surrounding each block. The outermost block must always be the SIERRA block. Commands directly in this block are referred to as “Domain-level” or “Sierra-level” commands. Other commands must go inside inner blocks, such as “Procedure” or “Fuego Region”, and will be referred to as such. If you put a command in the wrong level, you will get an error when you try to run Fuego.

```
Begin SIERRA Fuego
# Domain-level commands:
# * Fuego materials
# * Finite element model
# * Linear solvers

Begin Procedure FuegoProcedure
# Procedure-level commands

Begin Solution Control Description
# Solution control commands
```

(continues on next page)

```

End

Begin Fuego Region myRegion
  # Region-level commands:
  # * Equation(s) to solve
  # * Initial and Boundary conditions
  # * Post-processors
  # * File output specifications
End
End
End

```

There are two types of commands that can go in the input file - block commands and line commands. Block commands start with the **Begin** keyword and end with the **End** keyword, and can contain line commands or nested blocks.

```

Begin My Block Somename
  # line commands
End My Block Somename

```

The block type and name are optional on the **End** command, but may improve readability for large blocks

```

Begin My Block Somename
  # line commands
End

```

A line command is a command on a single line

```

X_Velocity = 1.2

```

Each line command has a specific block or set of blocks where it is valid. Using it outside those blocks will result in an error about an un-recognized command.

Aprepro Scripting

Fuego input files can use Aprepro commands to do pre-processing, calculations, and variable management within the input file itself. If you have an input file with Aprepro commands, you can choose to manually run Aprepro on that file to generate a processed output file like this

```

aprepro myfile.in > myfile.out

```

However, this is not necessary. When running Fuego it will automatically run Aprepro on the input before parsing it, and the input file contents put in the header of the log file will be the

Aprepro-resolved contents.

This section covers basic Aprepro usage as is relevant to general Fuego use. The complete Aprepro manual can be found with the [SEACAS Documentation](#).

Defining Constants

An Aprepro constant can be declared inside {} anywhere in the input file. Aprepro constants cannot be re-defined later unless they are marked as mutable by using a leading underscore in their name. Aprepro constants can also be defined from the command line when calling Aria. The values supplied at the command line will override values in the input file. Multiple values should be separated with commas or spaces.

```
fuego -i myfile.i --define "N=4"
fuego -i myfile.i --define "N=4 M=2"
fuego -i myfile.i --define "N=4,M=2"
```

When the input file is processed by Aprepro, the contents of the {} sections will be replaced with their values. For example, the following input

```
# Tref = { Tref = 300 } K
IC for Temperature on all_blocks = constant value = {Tref}
BC Flux for Energy on surface_1 = Nat_Conv T_ref={Tref} h=5
```

would evaluate to this (which would be shown in the log file)

```
# Tref = 300 K
IC for Temperature on all_blocks = constant value = 300
BC Flux for Energy on surface_1 = Nat_Conv T_ref=300 h=5
```



Note: While the line `Tref = {Tref = 300} K` may appear redundant in the input, since the term inside the {} is replaced with its actual value, using descriptive labels outside the {} results in a more readable output in the log file.

You can also do math in Aprepro to define functional values, for example to evaluate a correlation to get a convection coefficient you could use Aprepro like this

```
# Tref = { Tref = 300 } K
# D = { D = 0.1 } m
# nu = { nu = 1e-5 } m2/s
# V = { V = 10 } m/s
# Re = { Re = V*D/nu }
# Pr = { Pr = 0.7 }
```

(continues on next page)

(continued from previous page)

```
# k = { k = 0.05 } W/m-K
# Nu = { Nu = 0.332*Re^0.5*Pr^(1/3) }
# h = { h = Nu*k/D } W/m2-K

BC Flux for Energy on surface_1 = Nat_Conv T_ref={Tref} h={h}
```

Including Files

Aprepro can also be used to separate your input into multiple files, which can help with readability and maintainability of your problem setup. For example, you could have a file with your commonly used Fuego user functions that is shared between multiple input setups. You can also put verbose sections of the input file into a separate file to keep the primary file more readable, especially for problems with a large number of blocks. For example:

```
#{include("my_user_functions.inc")}
```

If-Else Logic

Aprepro allows the use of if-else statements, which can be used to conditionally activate parts of your input file. Commands inside a conditional statement will only be included in the actual Fuego input if the condition is true.

```
# Use wall temperature = { use_wall_temp = 1 }
# Do output = { do_output = 0 }

{if(use_wall_temp)}
wall_temperature = 500
{else}
# using an adiabatic wall
{endif}

{if(do_output)}
Begin Results Output Foo
# ...
End
{endif}
```

Loops

Aprepro also allows you to define simple loops, which can be helpful when having to repeat very similar commands many times in an input file. For example, if you want to generate an N by M grid of temperature probes you could do

```
# M = { M = 10 }
# N = { N = 15 }
# x0 = { x0 = 0.1 }
# y0 = { y0 = 0.2 }
# dx = { dx = 0.1 }
# dy = { dy = 0.1 }

{NOECHO} { _i = 0 }
{loop(M)}
{NOECHO} {++_i} { _j = 0 }
{loop(N)}
{NOECHO} {++_j}
{ECHO} # Tprobe_{_i}_{_j}
Begin postprocess point
    Location = {x0 + (_i-1)*dx} {y0 + (_j-1)*dy} 0
    Output Name = Tprobe_{_i}_{_j}
    Function = "temperature"
End
{endloop}
{endloop}
```

Which will generate 150 data probe commands in the input file. You could use similar logic to put those probe variables in the results output or heartbeat output block, which avoids boilerplate and makes it easy to change the probe grid location or resolution. The {NOECHO} and {ECHO} commands turn off and on output to the input file, respectively, and can be used to prevent intermediate output from being printed to the input file.

A similar approach can be used with delimited lists of parameters, such as boundary surfaces. If you had a list of surfaces, each of which needed a wall boundary condition with a different temperature, you could avoid repeating a nearly identical BC block N times using this approach by combining the list of surfaces with a list of wall temperatures.

```
# wall surfaces = { wall_surfs = "surface_1 surface_2 surface_3 surface_4
↪" }
# wall temperatures = { wall_temps = "300 350 400 450" }

{NOECHO} { _j = 0 } { num_wall_bcs = word_count(wall_surfs, " ") }
{loop(num_wall_bcs)}
{ECHO}# Wall BC { ++_j}
```

(continues on next page)

(continued from previous page)

```
Begin Wall Boundary Condition {get_word(_j, wall_surfs, " ")}  
    wall temperature = {get_word(_j, wall_temps, " ")}  
End  
{endloop}
```

4.2 Mesh

Finite element model

The model discretization (mesh database) and the mesh components to be used in the model are defined at the Sierra scope and are later referenced by the application at the Region level. Specification of a mesh database and association of material properties with portions of the mesh are defined within the `Finite Element Model` command block. In coupled analyses the need often arises for different discretizations of the same physical domain. In this case one must supply a different `Finite Element Model` command block for each discretization since only a single mesh can be referenced within this command block.

Defining the Mesh Database and Decomposition Method

The mesh database can be defined as follows:

```
Database name {=|are|is} <StreamName>
```

Where the `<StreamName>` is the base name containing the output or input mesh. If the filename begins with the `/` character, it is an absolute path; otherwise the path to the current directory (where the input file is) will be prepended to the name. If this command line is omitted, then an assumed filename is used; this will be the basename of the input file with a `.g` suffix appended. The most commonly used database formats are Exodus and Genesis (`.e` and `.g` suffix, respectively). The database type defines the format used for the input/output mesh:

The mesh can be partitioned across processes for parallel runs through a variety of decomposition methods; the particular method used can be defined with the following command line:

```
Decomposition Method {=|are|is} <MethodName>
```

The most commonly used parameters for `<MethodName>` are `rcb` (Recursive Coordinate Geometric Bisection) and `rib` (Recursive Inertial Bisection). If this command line is not specified, then the default method is `rcb` if the input mesh file is not already pre-decomposed. A simple example of defining a database name and its decomposition method for a finite element command block is shown below:

```
Begin Finite Element Model The_Model
  Database Name = the_mesh.g
  Decomposition Method = rib
End
```

Mesh Database Part Names and Assemblies

The input mesh database can be created by a variety of mesh generators. These mesh databases can contain parts of the mesh that either have user-defined names or utilize a default internal naming convention used for the meshed entity/part with a numbered index (e.g. surface_1, block_8). Fig. 4.1 below shows an example of a mesh using the default enumerating scheme.

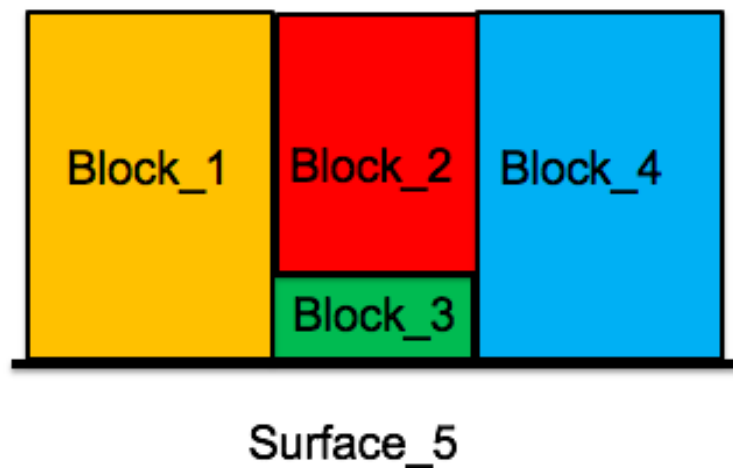


Fig. 4.1: Enumerated blocks

In the following sections, we will refer to Fig. 4.1 to demonstrate example commands of setting up the finite element model command block.

Mesh Part Material Assignments

Assignment of the associated material can be accomplished using a command block:

```
Begin Parameters for Block block_2
  material air
End

Begin Parameters for Block block_3
  material air
```

(continues on next page)

```

End

Begin Parameters for Block block_1
  material air
End

Begin Parameters for Block block_4
  material air
End

```

This individually assigns a material air to the all of the blocks in [Fig. 4.1](#). The above command block can be further shortened by assigning materials to more than one block at a time:

```

Begin Parameters for Block block_1 block_2 block_3 block_4
  material air
End

```

Finite Element Model shows a summary of commands used to assign materials to parts of the mesh.

Assemblies

Assemblies can be defined within Fuego to aggregate mesh parts (block, surface or node list) together. An example of defining a surface and block assembly is shown below:

```

Begin assembly inflow
  surface = surface_1 surface_2
End assembly

Begin assembly fluid_blocks
  block = block_1 block_2
End assembly

```

This block is defined under the finite element command block. It should be noted that each assembly must contain the same type (or rank) of parts (block, surface or node list). Building upon the examples in *Mesh Part Material Assignments*, we can increase the readability of our input deck by doing the following:

```

Begin assembly fluid_blocks
  block = block_1 block_2 block_3 block_4
End assembly

```

```

Begin Parameters for Block fluid_blocks
    material air
End

```

Assemblies can be used in a variety of operations (i.e. boundary conditions and postprocessing operations) and generally help readability of an input file. In addition, these assemblies can be used for transfers to other apps as well. The preferred method for grouping mesh parts together within an input deck is through assemblies.

A summary of commands available for defining assemblies is located in *Finite Element Model*.

Adaptivity

Fuego supports uniform refinement prior to the start of the simulation. This can be requested in the Fuego Region via:

```

Begin Fuego Region myRegion
    Initial Uniform Refinement for 2 iterations
    ...
End

```

4.3 Linear Solvers

Starting with the 5.0 release the Tpetra based linear solvers are the only linear solvers supported.

Within the input file the linear solver command block may optionally appear at the Domain scope as illustrated below. Note that different linear solvers can be used for different equations. A solver command block can appear in the input but not be used, and if no solver command blocks are explicitly specified Fuego will use sensible default linear solvers for each equation.

```

Begin Sierra
    .
    .
    Begin Tpetra Equation Solver first_solver
        .
        Tpetra linear solver commands
        .
    End
    Begin Tpetra Equation Solver second_solver
        .
        Tpetra linear solver commands

```

(continues on next page)

```

.
End
.
Begin Procedure My_Fuego_Procedure
.
  Begin Fuego Region My_Region
    .
    use equation solver first_solver for equation continuity
    use equation solver second_solver for equation X_Momentum
    use equation solver second_solver for equation Y_Momentum
    use equation solver second_solver for equation Z_Momentum
    .
  End
End
.
End Sierra

```

See *[Solver Selection Guidelines](#)* for suggestions on how to select a type of linear solver.

Tpetra Solver Overview

The syntax for specifying a Tpetra-based solver uses a nested command block structure. The outermost level at the Sierra scope is the **BEGIN** TPETRA EQUATION SOLVER block. This block must contain a single nested solver block corresponding to your desired solver type. Several types of direct and iterative linear solvers are provided through the Amesos2 and Belos packages of Trilinos. The supported solver blocks are listed in *[Tpetra Solvers](#)*.

If one of the iterative solvers is selected, it must also have a nested preconditioner block. A variety of preconditioners are supported through the Ifpack2 and MueLu Trilinos packages. See *[Tpetra Preconditioners](#)* for details on the supported preconditioners and their available options.

Solver Error Codes

If the linear solve fails it will report an error code. While these codes are not universally standardized across linear solvers, they can generally be interpreted with the following rules:

Error 1 (or -1)

The linear solver reached the maximum number of linear solver iterations and did not reach a solution. Check the final linear solver residual to see if this is serious or not. Even if the error is not serious (residuals are small) this usually means the solve is much more expensive than necessary and a better solver/preconditioner combination should be used.

Error 2 (or -2, 4, or -4)

The linear solver encountered a fatal internal error. This is often because of a singular matrix or a NaN or Infinite term. This is almost always a serious error.

Error 3 (or -3)

The linear solver had an internal loss of precision. This is often not a serious error, but may indicate the need for a better solver/preconditioner combination. As always, check the final linear solver residual reported in the log file to determine if it is serious or not.

Expert Syntax

The standard solver and preconditioner blocks only support a subset of the commonly used options that are supported by the underlying Trilinos packages. The subset was chosen to cover the vast majority of linear solver use cases within Fuego simulations, but an alternative approach is provided for expert users who would like to be able to pass parameters directly to the underlying Belos, Ifpack2, and MueLu packages. This expert interface is used by providing the path to an XML file containing the definition for a `Teuchos::ParameterList` that is supported by the underlying Trilinos package in one of the **BEGIN** EXPERT BELOS SOLVER, **BEGIN** EXPERT IFPACK2 PRECONDITIONER, or **BEGIN** MUELU PRECONDITIONER blocks as desired.

Sample XML for expert Belos solver

```
<ParameterList>
  <Parameter name="solution method" type="string" value="CG"/>
  <ParameterList name="Belos-Tpetra">
    <Parameter name="Convergence Tolerance" type="double" value="1.e-12"/>
    <Parameter name="Maximum Iterations" type="int" value="1000"/>
  </ParameterList>
</ParameterList>
```

The `solution method` parameter is required. Any additional parameters that are not specified in the Belos-Tpetra sublist will use their default values as defined by Belos.

Sample XML for expert Ifpack2 preconditioner

```
<ParameterList>
  <Parameter name="preconditioner type:" type="string" value="ILUT"/>
  <Parameter name="fact: ilut level-of-fill" type="double" value="1.0"/>
  <Parameter name="fact: drop tolerance" type="double" value="0.0"/>
</ParameterList>
```

The `preconditioner type:` parameter is required. Any additional parameters that are not specified will use their default values as defined by Ifpack2.

Sample XML for expert MueLu preconditioner

```

<ParameterList name="MueLu-Tpetra">
  <Parameter          name="verbosity"                type="string"
  ↪ value="high"/>
  <Parameter          name="coarse: type"              type="string"
  ↪ value="ILUT"/>
  <Parameter          name="coarse: max size"          type="int"
  ↪ value="1000"/>
  <Parameter          name="max levels"                type="int"
  ↪ value="4"/>
  <Parameter          name="smoother: type"            type="string"
  ↪ value="CHEBYSHEV"/>
  <ParameterList      name="smoother: params">
    <Parameter name="chebyshev: degree"                type="int"
    ↪ value="2"/>
    <Parameter name="chebyshev: ratio eigenvalue"      type="double"
    ↪ value="20"/>
    <Parameter name="chebyshev: min eigenvalue"        type="double"
    ↪ value="1.0"/>
    <Parameter name="chebyshev: zero starting solution" type="bool"
    ↪ value="true"/>
    <Parameter name="chebyshev: eigenvalue max iterations" type="int"
    ↪ value="15"/>
  </ParameterList>
  <Parameter          name="aggregation: type"          type="string"
  ↪ value="uncoupled"/>
  <Parameter          name="aggregation: drop tol"      type="double"
  ↪ value="0.02"/>

  <Parameter          name="repartition: enable"        type="bool"
  ↪ value="true"/>
  <Parameter          name="repartition: min rows per proc" type="int"
  ↪ value="1000"/>
  <Parameter          name="repartition: start level"   type="int"
  ↪ value="2"/>
  <Parameter          name="repartition: max imbalance" type="double"
  ↪ value="1.327"/>
  <Parameter          name="repartition: partitioner"   type="string"
  ↪ value="zoltan2"/>
</ParameterList>

```

Unspecified parameters will use their default values as defined by MueLu.

For further documentation of the parameters supported by each Trilinos package please see the corresponding documentation on trilinos.org.

Solver Selection Guidelines

Selecting an appropriate linear solver that is both robust and efficient for a particular Fuego simulation can be confusing. In general there is a tradeoff between solver speed and robustness, and this can significantly affect the total time to solution. The following guidelines for choosing a linear solver may be useful:

1. As a starting point preset solvers are provided for several common types of problems. These are recommended as a starting point for any analysis, and Fuego will use an appropriate preset for each equation type if a specific solver is not specified.
 - For fluids problems the `continuity` preset is recommended for the continuity equation system, and the `scalar_transport` preset is recommended for the momentum equation system, as well as any additional scalar advection-diffusion equations. The `continuity` preset is a GMRES solver with an algebraic multigrid preconditioner provided through the MueLu Trilinos package. The `scalar_transport` preset is a GMRES solver with a SGS preconditioner.
 - The `thermal` preset solver type is recommended as the starting point for general thermal analysis problems. It uses a GMRES solver with Jacobi preconditioning and is a good balance between runtime per linear solver iteration and robustness for most thermal problems.
 - The `thermal_symmetric` and `thermal_bicgstab` presets can be slightly faster for thermal analysis problems that do not involve contact. They are CG and BiCGSTAB solvers respectively, each with Jacobi preconditioning. These trade some robustness for shorter runtime per linear iteration when compared with the GMRES solver used by the `thermal` preset.
 - For more general multiphysics problems involving multiple coupled equations in a single equation system the `multiphysics` preset solver is the recommended starting point. It is a GMRES solver with a DD-ILU preconditioner.
 - When using any of the preset solver types they will print an equivalent solver block to the log file that can be used as a starting point for modifying additional solver or preconditioner settings.
2. If the suggested preset solver shows evidence of struggling to solve a particular problem in the log file (for example frequently approaching or reaching the maximum number of linear iterations) consider using a stronger solver and/or preconditioner. For the solvers move from CG to BiCGSTAB to GMRES. For the preconditioners move from Jacobi to SGS to DD-ILU or DD-ILUT. Initially leave all preconditioner-specific options to their default values.
3. If GMRES/DD-ILU still shows evidence of struggling to solve the problem consider increasing the ILU fill and overlap levels. First trying overlap of 1, then fill of 1, then both is a good balance between the cost and robustness of the preconditioner. See [Dd-Ilu Preconditioner](#). for setting the fill and overlap.

4. If GMRES/DD-ILU with fill and overlap is still insufficient consider:
 - Trying one of the direct solvers (KLU2, SuperLU, or UMFPACK) if your problem is small. These are the most robust option available to Fuego, but significantly more expensive in both memory usage and runtime than the iterative solver options. They also do not scale in parallel effectively, and should not be used for problems requiring more than a single compute node on HPC systems.
 - For multiphysics problems involving many equations in a single monolithic equation system consider moving to a segregated solution scheme involving multiple smaller equation systems. This will make the linear solves easier, but can make nonlinear convergence of the overall problem more challenging. Loosely coupled equations are more likely to be successfully split into separate equation systems without causing nonlinear convergence problems.
5. Multigrid preconditioning can be extremely effective and exhibit excellent parallel scalability, however setup of multigrid preconditioners is an extremely complex topic that is both problem type and problem size dependent. Fuego supports general algebraic multigrid preconditioning through the MueLu package from Trilinos, and can read MueLu XML files for defining the multigrid setup. Please consult the MueLu documentation for suggestions on selecting a multigrid preconditioner.

Tolerance Settings

In addition to selecting the solver and preconditioner type, selecting appropriate convergence tolerance and scaling settings is also important. The default settings (R0 scaling and AUTOMATIC tolerance) are a reasonable starting place, but may need to be adjusted depending on the problem and nonlinear solver settings. Typically setting the RESIDUAL SCALING to either NONE or R0 is most useful.

- NONE applies no scaling to the residual for determining linear solver convergence. If the problem has a low residual to begin with the solver will only take a few iterations, however if the initial residual is large this option may take more linear solver iterations than is ideal. The linear solver tolerance selected with this option should be at least an order of magnitude lower than the nonlinear residual tolerance used in the Newton solve or the nonlinear problem may have trouble converging.
- R0 scales the residual by the initial residual. This is useful if you want a looser acceptance tolerance when the problem starts with high residual, however if the problem starts with a small residual this may cause the solver to take an unreasonably large number of iterations. If the initial residual is large and the linear solver tolerance is too loose this can result in the linear solve producing bad/poorly solved results which may cause convergence problems.

An autonomous linear convergence tolerance can be computed within Fuego, and is declared within the Tpetra solvers by setting the parameter CONVERGENCE TOLERANCE = AUTOMATIC in the Tpetra solver block. This option computes a linear convergence tolerance for each nonlinear iteration based on the initial linear residual and the user-specified nonlinear residual tolerance.

The user can also specify an optional minimum linear residual tolerance (with a default value of 1×10^{-6}) to ensure that the computed linear convergence tolerance within Fuego is not used if it's too small. Below is an example of using this autonomous setting with a specified minimum

```
BEGIN TPETRA EQUATION SOLVER  GMRES_DDILU
  BEGIN GMRES SOLVER
    BEGIN DD-ILU PRECONDITIONER
    END
    MAXIMUM ITERATIONS = 1000
    RESIDUAL SCALING = NONE
    CONVERGENCE TOLERANCE = AUTOMATIC MINIMUM = 1.0E-8
  END
END TPETRA EQUATION SOLVER
```

4.4 Material Properties

Most Fuego simulations require only a single material definition, whose properties are applied to all blocks in the mesh. The different methods for specifying material properties in Fuego are described in the following sections.

Basic Material Properties

For simple problems with constant or non-complex properties, material properties can be specified directly in the material block. For example, for an isothermal laminar flow the only required properties are density and viscosity. To set constant values for these, one could do

```
# At the sierra-block level
Begin Property Specification for Fuego Material air
  Density    = 1.2
  Viscosity  = 1.8e-5
End
```

For a non-isothermal problem, thermal conductivity, specific heat, and a function for enthalpy as a function of temperature are required. These can be specified in a similar manner, using either in-line string functions or tabulated user functions.

```
# At the sierra-block level
Begin Property Specification for Fuego Material air

  Reference Temperature = 300

  # rho = P*W/(R*T) = Ts / T where Ts = P*W/R
```

(continues on next page)

(continued from previous page)

```
# Cp = { Cp = 1000 }
Density = "300 / temperature"
Viscosity = 1.8e-5
Specific_Heat = {Cp}
Thermal_Conductivity = 0.03
Enthalpy = "{Cp}*temperature"
End
```



Note: When properties have dependencies, like temperature here, a reference value for those dependencies must also be specified (reference temperature here).

To use a tabulated function, the function must be defined in a separate user function block

```
# At the sierra-block level
Begin Property Specification for Fuego Material air

Reference Temperature = 300

# rho = P*W/(R*T) = Ts / T where Ts = P*W/R
# Cp = { Cp = 1000 }
Density = "300 / temperature"
Viscosity = 1.8e-5
Specific_Heat = {Cp}
Thermal_Conductivity = 0.03
Function for Enthalpy = hFuncAir
End

Begin Definition For Function hFuncAir
TYPE IS PIECEWISE LINEAR
ABSCISSA = temperature
BEGIN VALUES
    0. 0.0
    5000. {Cp*5000}
END VALUES
END
```



Note: When Fuego calculates temperature from enthalpy, it is assumed that $C_p = \frac{\partial h}{\partial T}$. When providing properties manually, be sure this relationship is true for the specified enthalpy and specific heat.

Units

When defining material properties directly, their units should be in your problem units. For example, if your mesh is in meters you should provide values in MKS units. There is no automatic unit conversion done for directly specified properties.

One exception to this is reference pressure, which should always be specified in atmospheres.

Non-Dimensional parameters

Fuego typically uses the Schmidt number ($Sc = \frac{\mu}{\rho D}$) and Prandtl number ($Pr = \frac{\mu}{\rho \alpha}$) to define diffusion of mass and energy as a function of viscosity. When setting up a turbulent flow problem, these parameters should be specified in the material property block.

```
Begin Property Specification for Fuego Material air
```

```
# Other property definitions
```

```
SCHMIDT_NUMBER = 0.9
```

```
PRANDTL_NUMBER = 0.9
```

```
End
```



Note: These are separate from the turbulent Prandtl and turbulent Schmidt numbers, which are specified in the global constants block.

Cantera Material Properties

For more complicated simulations, particularly those involving multi-species gas mixtures, Fuego supports using Cantera XML files to define the fluid properties. To use Cantera, your material property block should specify a Cantera XML file:

```
# At the sierra-block level
```

```
Begin Property Specification for Fuego Material air
```

```
CANTERA XML FILE = helium.xml
```

```
REFERENCE PRESSURE = 1.0           $ atmosphere
```

```
REFERENCE TEMPERATURE = 298.       $ Kelvin
```

```
REFERENCE MASS_FRACTION N2 = 0.767
```

```
REFERENCE MASS_FRACTION HE = 0.00
```

(continues on next page)

```
REFERENCE MASS_FRACTION O2 = 0.233
```

```
SCHMIDT_NUMBER = 0.9
```

End

The XML file should include a `<phase>` block, which defines the elements and species present and what mixture models to use, and a `<speciesData>` block which defines thermodynamic and transport models for each species.

```
<?xml version="1.0"?>
<ctml>
  <!-- phase gas -->
  <phase dim="3" id="gas">
    <elementArray>He 0 N</elementArray>
    <speciesArray datasrc="#species_data">HE O2 N2 </speciesArray>
    <thermo model="IdealGas"/>
    <transport model="Mix"/>
  </phase>

  <!-- species definitions -->
  <speciesData id="species_data">
    <!-- ... -->
  </speciesData>
</ctml>
```



Note: The order the species are listed in the Cantera XML file is the order used throughout Fuego. The last species listed (N2 here) will be the “fracbal” species that is not solved for, but whose mass fraction is set to achieve a unity mass fraction sum.



Note: Older versions of the Cantera XML file may have referenced an `elements.xml` file in the `<elementArray>` tag, which would have required a separate `elements.xml` file defining valid elements and their molecular weights. This is no longer required unless you are using custom elements not in the periodic table. Simply removing that attribute from the `<elementArray>` will use the default periodic table.

Aside from changing the list of elements and species, one generally should not change any other parts of the `<phase>` block. Within the `<speciesData>` block, there should be an entry for each species, including `<thermo>` and `<transport>` sub-blocks.

```

<!-- species HE -->
<species name="HE">
  <atomArray>He:1 </atomArray>
  <thermo>
    <NASA Tmax="1000.0" Tmin="300.0" P0="100000.0">
      <floatArray name="coeffs" size="7">
        2.500000000E+00, 0.000000000E+00, 0.000000000E+00, 0.
        ↪000000000E+00,
        0.000000000E+00, -7.453750000E+02, 9.153488000E-01</
        ↪floatArray>
      </NASA>
    <NASA Tmax="5000.0" Tmin="1000.0" P0="100000.0">
      <floatArray name="coeffs" size="7">
        2.500000000E+00, 0.000000000E+00, 0.000000000E+00, 0.
        ↪000000000E+00,
        0.000000000E+00, -7.453750000E+02, 9.153489000E-01</
        ↪floatArray>
      </NASA>
    </thermo>
    <transport model="gas_transport">
      <string title="geometry">atom</string>
      <LJ_welldepth units="K">10.200</LJ_welldepth>
      <LJ_diameter units="A">2.580</LJ_diameter>
      <dipoleMoment units="Debye">0.000</dipoleMoment>
      <polarizability units="A3">0.000</polarizability>
      <rotRelax>0.000</rotRelax>
    </transport>
  </species>

```

Thermodynamic Properties

The <thermo> section specifies standard thermodynamic polynomials for the different species over different temperature ranges. The common types of polynomials are described below.

NASA7

The NASA7 polynomial takes 7 coefficients (a_0 - a_6) and expects one or two temperature ranges, as shown below.

```
<thermo>
  <NASA Tmax="1000.0" Tmin="300.0" P0="100000.0">
    <floatArray name="coeffs" size="7">
      2.500000000E+00, 0.000000000E+00, 0.000000000E+00, 0.
      ↪000000000E+00,
      0.000000000E+00, -7.453750000E+02, 9.153488000E-01</floatArray>
    </NASA>
  <NASA Tmax="5000.0" Tmin="1000.0" P0="100000.0">
    <floatArray name="coeffs" size="7">
      2.500000000E+00, 0.000000000E+00, 0.000000000E+00, 0.
      ↪000000000E+00,
      0.000000000E+00, -7.453750000E+02, 9.153489000E-01</floatArray>
    </NASA>
</thermo>
```

These coefficients define the following properties:

$$\frac{C_p(T)}{R} = a_0 + a_1T + a_2T^2 + a_3T^3 + a_4T^4 \quad (4.1)$$

$$\frac{h(T)}{R} = a_0T + \frac{a_1}{2}T^2 + \frac{a_2}{3}T^3 + \frac{a_3}{4}T^4 + \frac{a_4}{5}T^5 + a_5 \quad (4.2)$$

$$\frac{s(T)}{R} = a_0 \ln(T) + a_1T + \frac{a_2}{2}T^2 + \frac{a_3}{3}T^3 + \frac{a_4}{4}T^4 + a_6 \quad (4.3)$$

NASA9

The NASA9 polynomial takes 9 coefficients (a_0 - a_8) and can use an arbitrary number of temperature ranges, as shown below.

```
<thermo>
  <NASA9 Pref="1 bar" Tmax="1000.00" Tmin="200.00">
    <floatArray size="9">
      4.943650540e+04, -6.264116010e+02, 5.301725240e+00, 2.503813816e-
      ↪03,
      -2.127308728e-07, -7.689988780e-10, 2.849677801e-13, -4.
      ↪528198460e+04,
      -7.048279440e+00
    </floatArray>
```

(continues on next page)

```

</NASA9>
<NASA9 Pref="1 bar" Tmax="6000.00" Tmin="1000.00">
  <floatArray size="9">
    1.176962419e+05, -1.788791477e+03, 8.291523190e+00, -9.223156780e-
    ↪05,
    4.863676880e-09, -1.891053312e-12, 6.330036590e-16, -3.
    ↪908350590e+04,
    -2.652669281e+01
  </floatArray>
</NASA9>
<NASA9 Pref="1 bar" Tmax="20000.00" Tmin="6000.00">
  <floatArray size="9">
    -1.544423287e+09, 1.016847056e+06, -2.561405230e+02, 3.369401080e-
    ↪02,
    -2.181184337e-06, 6.991420840e-11, -8.842351500e-16, -8.
    ↪043214510e+06,
    2.254177493e+03
  </floatArray>
</NASA9>
</thermo>

```

These coefficients define the following properties:

$$\frac{C_p(T)}{R} = a_0 T^{-2} + a_1 T^{-1} + a_2 + a_3 T + a_4 T^2 + a_5 T^3 + a_6 T^4 \quad (4.4)$$

$$\frac{h(T)}{R} = -a_0 T^{-1} + a_1 \ln(T) + a_2 T + \frac{a_3}{2} T^2 + \frac{a_4}{3} T^3 + \frac{a_5}{4} T^4 + \frac{a_6}{5} T^5 + a_7 \quad (4.5)$$

$$\frac{s(T)}{R} = -\frac{a_0}{2} T^{-2} - a_1 T^{-1} + a_2 \ln(T) + a_3 T + \frac{a_4}{2} T^2 + \frac{a_5}{3} T^3 + \frac{a_6}{4} T^4 + a_8 \quad (4.6)$$

Shomate

The Shomate polynomial takes 7 coefficients (A-G) and expects one or two temperature ranges, as shown below.

```

<thermo>
  <Shomate Pref="1 bar" Tmax="1400.0" Tmin="298">
    <floatArray size="7">
      72.3870, 8.8501, 0.0007, 0,
      -1.1428, -197.3525, 131.4492
    </floatArray>
  </Shomate>
</thermo>

```

Unlike the NASA polynomials, the Shomate polynomials are not normalized by the ideal gas constant. The expected units of C_p , h , and s are J/mol-K, kJ/mol, and J/mol-K, respectively. These coefficients define the following properties, using a scaled temperature t :

$$t = \frac{T}{1000} \quad (4.7)$$

$$C_p(T) = A + Bt + Ct^2 + Dt^3 + \frac{E}{t^2} \quad (4.8)$$

$$h(T) = At + \frac{B}{2}t^2 + \frac{C}{3}t^3 + \frac{D}{4}t^4 - \frac{E}{t} + F \quad (4.9)$$

$$s(T) = A \ln(t) + Bt + \frac{C}{2}t^2 + \frac{D}{3}t^3 - \frac{E}{2t^2} + G \quad (4.10)$$

Transport Properties

The <transport> block in the Cantera file requires 6 entries:

- **Geometry** - One of `atom` (for monatomic species like He), `linear` (for diatomic species like N₂), or `nonlinear` (for all others)
- **Lennerd Jones Well Depth** - Value in Kelvin for the species interaction potential depth.
- **Lennerd Jones Diameter** - Value in angstroms for the species interaction diameter.
- **Dipole moment** - Value in Debye determined from the molecular structure.
- **Polarizability** - Value in Angstroms cubed which affects interactions between charged and un-charged species.
- **Rotational Relaxation** - Value that affects only thermal conductivity (not typically used in Fuego).

These values are typically tabulated in standard transport data sets from NIST or NASA. For un-common species, methods for estimating L-J parameters from other properties like critical point and boiling point properties can be found in [Estimated Viscosities and Thermal Conductivities of Gases at High Temperatures](#), Svehla, 1962.

```
<transport model="gas_transport">
  <string title="geometry">atom</string>
  <LJ_welldepth units="K">10.200</LJ_welldepth>
  <LJ_diameter units="A">2.580</LJ_diameter>
  <dipoleMoment units="Debye">0.000</dipoleMoment>
  <polarizability units="A3">0.000</polarizability>
  <rotRelax>0.000</rotRelax>
</transport>
```

Resources

The following web resources and SAND reports can provide useful information for constructing Cantera files.

- [NASA Glenn Coefficients for Calculating Thermodynamic Properties of Individual Species \(NASA/TP—2002-211556\)](#)
- [NIST Chemistry Webbook](#)
- [A Fortran computer code package for the evaluation of gas-phase, multicomponent transport properties \(SAND86-8246\)](#)
- [The Chemkin Thermodynamic Data Base \(SAND87-8215B\)](#)

Tabulated Material Properties

When using a flamelet model in Fuego, one must first pre-calculate a table of necessary properties as functions of the transported scalars (*e.g.* mixture fraction). The theory behind this approach is described in detail in *Laminar Flamelet Turbulent Combustion Model*, so this section will focus on how to set up a Fuego material model using a pre-computed flamelet table. You can also refer to the [SpitFire](#) wiki for details on how to generate a flamelet table.

Inside the Property Specification block, you will supply reference values for the property inputs as with the *basic* and *Cantera* materials. The nested Tabular Property Library block then defines the table file to use and links the table variables and properties with Fuego variables and properties (if necessary).

```
BEGIN PROPERTY SPECIFICATION FOR FUEGO MATERIAL gas
  REFERENCE mixture_fraction = 0.0
  REFERENCE scaled_scalar_variance = 0.0
  REFERENCE scalar_diss_rate = 0.0
  REFERENCE temperature = 300.0 # or REFERENCE heat_loss = 0
  REFERENCE pressure = 1.0

  BEGIN TABULAR PROPERTY LIBRARY SLFM
    TABLE UNITS ARE IN MKS
    LIBRARY HDF5 FILE = methane_plume.h5
  END TABULAR PROPERTY LIBRARY SLFM

  SCHMIDT_NUMBER = 0.9
  PRANDTL_NUMBER = 0.9

END PROPERTY SPECIFICATION FOR FUEGO MATERIAL gas
```

Mapping Table Inputs and Outputs

To translate between table outputs and inputs and Fuego variables, most tables can simply use default settings with the most common names used in table generation. However, if your table has non-standard naming for inputs or properties you can specify this mapping manually.



Note: If you have any custom names in the inputs or outputs, no automatic mapping is used for that group so you must supply mapping for all inputs and/or all outputs.

```
BEGIN PROPERTY SPECIFICATION FOR FUEGO MATERIAL gas
# ...

BEGIN TABULAR PROPERTY LIBRARY SLFM
  TABLE UNITS ARE IN MKS
  LIBRARY HDF5 FILE = methane_plume.h5

  USE FIELD mixture_fraction      FOR LIBRARY INPUT mixture_fraction_
↪mean
  USE FIELD scaled_scalar_variance FOR LIBRARY INPUT scaled_scalar_
↪variance_mean
  USE FIELD scalar_diss_rate      FOR LIBRARY INPUT dissipation_rate_
↪stoich_mean
  USE FIELD heat_loss            FOR LIBRARY INPUT enthalpy_defect_
↪stoich_mean

  USE LIBRARY VARIABLE density      FOR PROPERTY density
  USE LIBRARY VARIABLE viscosity    FOR PROPERTY viscosity
  USE LIBRARY VARIABLE temperature FOR PROPERTY temperature
  USE LIBRARY VARIABLE heat_capacity_cp FOR PROPERTY specific_heat
  USE LIBRARY VARIABLE enthalpy     FOR PROPERTY enthalpy
  USE LIBRARY VARIABLE enthalpy_cons FOR PROPERTY conserved_
↪enthalpy

END TABULAR PROPERTY LIBRARY SLFM

END PROPERTY SPECIFICATION FOR FUEGO MATERIAL gas
```

Soot Model Terms

To activate the pre-configured Aksit-Moss SNL soot model you can add the Use Aerosol model `aksit_moss_snl` in the tabular property block. This will add pre-configured soot and radiation source terms, as well as progress variables `soot_moles_per_mass` and `soot_mass_fraction`.

```
BEGIN PROPERTY SPECIFICATION FOR FUEGO MATERIAL gas
# ...
BEGIN TABULAR PROPERTY LIBRARY SLFM
  LIBRARY HDF5 FILE = ./pure-ethylene.h5
  TABLE UNITS ARE IN mks
  USE AEROSOL MODEL aksit_moss_snl
END TABULAR PROPERTY LIBRARY SLFM
END
```

Radiation Terms

If you are not using a pre-made soot model, and have radiative transport active, you will need to define models for absorption and `rad_source` to be passed to the PMR solver. These expressions may be simple copies of table sources, as below, or may be a user-defined function to include effects of soot.

```
BEGIN PROPERTY SPECIFICATION FOR FUEGO MATERIAL gas
# ...

BEGIN TABULAR PROPERTY LIBRARY SLFM
  TABLE UNITS ARE IN MKS
  LIBRARY HDF5 FILE = methane_plume.h5

  ##### RADIATION SOURCE TERMS
  USE LIBRARY SOURCE gas_abs_coeff AS gas_abs_coeff
  USE LIBRARY SOURCE gas_rad_source AS gas_rad_source
  AUXILIARY VARIABLE absorption EXPRESSION = "gas_abs_coeff"
  AUXILIARY VARIABLE rad_source EXPRESSION = "gas_rad_source"

END TABULAR PROPERTY LIBRARY SLFM

END PROPERTY SPECIFICATION FOR FUEGO MATERIAL gas
```

Custom Output

There may be additional tabulated properties you want to output for post-processing or visualization, such as mass fractions. You can output those to nodal fields with the OUTPUT LIBRARY VARIABLE commands, as shown below.

```
BEGIN PROPERTY SPECIFICATION FOR FUEGO MATERIAL gas
# ...

BEGIN TABULAR PROPERTY LIBRARY SLFM
# ...

OUTPUT LIBRARY VARIABLE massfraction-C2H4 AS massfraction-C2H4
OUTPUT LIBRARY VARIABLE massfraction-CO AS massfraction-CO
OUTPUT LIBRARY VARIABLE massfraction-CO2 AS massfraction-CO2
OUTPUT LIBRARY VARIABLE massfraction-O2 AS massfraction-O2

END TABULAR PROPERTY LIBRARY SLFM

END PROPERTY SPECIFICATION FOR FUEGO MATERIAL gas
```

Understanding Tabular Property Clipping Diagnostics

The tabular property evaluators allow for the evaluation of a dependent variable, ϕ , based on the input vector \mathbf{x} . However, a given evaluation point \mathbf{x} may lie outside the bounds of the table. In this scenario, $\phi(\mathbf{x})$ is instead evaluated at the clipped point \mathbf{x}^* , which represents the closest point to \mathbf{x} that lies inside the range of the table.

The impact of the tabular property clipping events on the accuracy of the simulation, moreover, is reported during a Fuego simulation. For each component i of the table input, a one-dimensional linear extrapolation of ϕ is computed as

$$\phi_{i,ext}(\mathbf{x}) = \phi(\mathbf{x}^*) + \frac{\partial \phi}{\partial \mathbf{x}_i}(\mathbf{x}^*) (\mathbf{x}_i - \mathbf{x}_i^*). \quad (4.11)$$

The relative difference between the linear extrapolation in (4.11) and the property evaluated at the clipped point represents the relative interpolant error,

$$e_{\phi,i} = \frac{\phi_{i,ext}(\mathbf{x}) - \phi(\mathbf{x}^*)}{\|\phi\|_{\infty}}, \quad (4.12)$$

where $\|\phi\|_{\infty}$ is the maximum absolute value of ϕ within the bounds of the table. (4.12) provides information regarding the relative error in the interpolant incurred by the clipping event.

Example output of the clipping severity diagnostic for a methane plume fire simulation using the steady laminar flamelet model is shown below. The top row denotes the specific property, ϕ ,

logged in the clipping output. Each clipped component of the input vector is displayed as a separate row. For each row, information regarding logarithmic spacing, constructed table ranges, and the observed range used in the simulation are displayed for each component. For example, *the example output* shows that `dissipation_rate_stoich_mean` is logarithmically spaced. The constructed table ranges, displayed in logarithmic terms, vary from -6.908 to 2.830 . The actual observed ranges required during the simulation, however, vary as widely as -36.84 to 21.69 .

The range of values required during a simulation, however, provides limited information on the impact of the clipping event on the simulation. The total number of clipping events and the minimum, maximum, and average values of the relative interpolant error described in (4.12) are presented in the last four columns. This information provides a better metric on the error induced in the clipping event than the range of values required alone. While the required input range for `enthalpy_defect_stoich_mean` in *the example output* may seem more severe than `dissipation_rate_stoich_mean`, the relative interpolant error is no greater than 0.4% in magnitude. For `dissipation_rate_stoich_mean`, however, the relative interpolant error is as great as 23.64%. In the Fuego logfile, this is marked as a MODERATE severity level warning. This informs the user that errors from property evaluation clipping events may impact the accuracy of a simulation.

Listing 4.1: Example clipping severity output from a Fuego simulation using the steady laminar flamelet model.

↪	-----					
						density ↵
↪						

↪	-----					
						Input Ranges ↵
↪			Rel. Interpolant Err.			

↪	-----					
	Input		Log?	Table Range*		↵
↪	Observed Range*	Count	Min	Max	Avg	

↪	-----					
	dissipation_rate_stoich_mean		Yes	[-6.908, 2.830]		[-
↪	36.84, 21.69]		30802400	-0.8716%		23.65%
	enthalpy_defect_stoich_mean		No	[-2.308e+06, 0.000]		[-4.
↪	408e+16, 1.707e+16]		5012642	-0.3923%		0.005572%

↪	-----					
↪	-----					

Volume-Of-Fluid Material Properties

A volume-of-fluid simulation requires separate material models for the liquid and gas materials, and a top-level material model to tie them together. Any required reference values should be supplied in the top-level material, but the sub-materials can generally be any type (basic, Cantera, etc). In the finite element block, the top level material would be the one assigned to the blocks. Interface properties, such as surface tension, should be provided in the top-level material.

```
BEGIN PROPERTY SPECIFICATION FOR FUEGO MATERIAL fluid_material
  VOF Gas Material = gas_material
  VOF Liquid Material = liquid_material

  INTERFACE_ABSORPTION = 1000
  SURFACE_TENSION = 0.073
  REFERENCE TEMPERATURE = 370.0
END   PROPERTY SPECIFICATION FOR FUEGO MATERIAL fluid_material

BEGIN PROPERTY SPECIFICATION FOR FUEGO MATERIAL gas_material
  density = "300/temperature"
  VISCOSITY = 1.98e-5
  SPECIFIC_HEAT = 1000
  THERMAL_CONDUCTIVITY = 0.025
  ENTHALPY = "1000*temperature"
  ABSORPTION = 0.1
END   PROPERTY SPECIFICATION FOR FUEGO MATERIAL gas_material

BEGIN PROPERTY SPECIFICATION FOR FUEGO MATERIAL liquid_material
  DENSITY = 1000
  VISCOSITY = 1e-3
  SPECIFIC_HEAT = 4184
  THERMAL_CONDUCTIVITY = 0.6
  ENTHALPY = "4184*temperature"
  ABSORPTION = 1.0
END   PROPERTY SPECIFICATION FOR FUEGO MATERIAL liquid_material
```

Composite Material Properties

When using the *composite boundary condition* you must define a separate material to use for the composite. This will define the species in the composite, as well as its physical properties. The composite material will also define the chemistry mechanism to use in the composite and the ODE solver to use to solve it.

The properties in the composite material can be either constants, or functions of composition that will change as the composite decomposes.

```

BEGIN PROPERTY SPECIFICATION FOR FUEGO MATERIAL new_composite
REFERENCE temperature = 300.
REFERENCE mass_fraction Carbon = 0.6724
REFERENCE mass_fraction Epoxy = 0.3276
REFERENCE mass_fraction Char = 0.0

species_density Carbon = 1779. # kg/m^3
species_density Epoxy = 1300. # kg/m^3
species_density Char = 650. # kg/m^3

SUBROUTINE FOR density = density_inverse_mass_average
INPUT VARIABLES FOR density_inverse_mass_average = species_density mass_
↪fraction

THERMAL_CONDUCTIVITY = "0.023*(temperature-273.15)^0.46"
SPECIFIC_HEAT = "750 + 4.1*(temperature-273.15)"

species_emissivity char = 0.95
species_emissivity epoxy = 0.5
species_emissivity carbon = 0.5

SUBROUTINE FOR emissivity = emissivity_mixture_average
INPUT VARIABLES FOR emissivity_mixture_average = species_emissivity_
↪mass_fraction

Begin ODE solver parameters
ODE Solver = LSODE
Absolute Tolerance = 1e-6
Relative Tolerance = 1e-3
End

Begin Chemistry Description MyChem
Begin Reaction R1
Reaction is Epoxy -> 0.5Char + 0.5CH4
Rate Function = Arrhenius A = 3.33e17 Ea = 2.2614e5 R = 8.314
Concentration Function = Standard mu = Automatic
Heat of Reaction = 0.0
End
End
END

```

4.5 Solution Control

All Fuego input files must include either a Solution Control Description or a *Time Control* block in the Procedure section of the input file. When using Solution Control this description contains directives for executing a transient analysis ranging from simple single-region analyses to more complicated multi-region execution including transfer, nested nonlinear iterations, and/or subcycling.

A solution control block has three parts:

System

Controls how the simulation is advanced

Parameters

Parameters to inform how each region is handled by system

Initialization

Additional actions to do at startup before system-defined behavior

See the *command reference* for details on all possible commands.

The system block controls how the analysis progresses by advancing the relevant region(s). For a *Transient Analysis* this corresponds to taking a time step.

When there are multiple regions involved in an analysis, it is often necessary to transfer information between them. Once a *transfer block* is set up, this is accomplished with the *transfer* directive. These more complex analyses may also require explicitly *initialization* of fields. Depending on the simulation types, various *parameters* are required, as is demonstrated in the sections below.

Though its often more favorable to solve fully-coupled physics within a single application, this is not always possible. As an alternative, loose-coupling is often carried out by exchanging solution information across multiple applications. Solution control provides various means of carrying out these analyses depending upon the strength of physics coupling within a solution step.



Note: This section refers to coupling *within* the Sierra ecosystem, e.g. coupling Fuego and Aria via `fuego_aria`.

Transient Analysis

A *Transient* block defines the way transient region(s) advance. *Parameters* are required to describe the time range of the simulation. Additionally, parameters defining the time integration for each Region must also be supplied by the user. The example below demonstrates both a fixed and adaptive timestep transient.

Simple Transient Input

```
Begin Sierra myJob

# Materials, Solvers, Finite Element Model, etc
...

Begin Procedure My_Fuego_Procedure

  Begin Solution Control Description

    Use System Main

    Begin System Main
      Simulation Start Time          = 0.0
      Simulation Termination Time    = 10.0
      Simulation Max Global Iterations = 1000

      Begin Transient Time_Block_1
        Advance My_Fuego_Region
      End
      Begin Transient Time_Block_2
        Advance My_Fuego_Region
      End
    End

    #Parameters for Time_Block_1 behavior
    Begin Parameters For Transient Time_Block_1
      Start Time          = 0.0
      Number of steps = 8
      Initial deltat = 0.1
      Begin Parameters For Fuego Region My_Fuego_Region
        transient step type is fixed
      End
    End

    #Parameters for Time_Block_2 behavior
```

(continues on next page)

(continued from previous page)

```
Begin Parameters For Transient Time_Block_2
  Begin Parameters For Fuego Region My_Fuego_Region
    transient step type is automatic
    CFL LIMIT = 1.0
    Time Step Change Factor = 1.2
  End
End
End

...
End Procedure My_Fuego_Procedure
End Sierra myJob
```



Note: Note that for each named **Begin** Transient [NAME] command block there should be a corresponding **Begin** Parameters for Transient [NAME] command block which defines its parameters.

Multiple regions can be tied together either in a weak or strong coupling using transfers and *Nonlinear* command blocks. The stronger case is demonstrated below. Here, the time step for advancement to the next solution step is negotiated between the coupled Regions based upon their respective Transient parameters. A weaker coupling can be obtained by omitting the **Begin** Nonlinear block and associated **Begin** Parameters for Nonlinear Iteration.

Coupled Transient Input

```
Begin Sierra myJob

# Materials, Solvers, Finite Element Model, etc
...

Begin Procedure My_Fuego_Procedure

  Begin Solution Control Description

    Use System Main

    Begin System Main
      Simulation Start Time           = 0.0
      Simulation Termination Time     = 10.0
      Simulation Max Global Iterations = 1000

    Begin Transient Time_Block
```

(continues on next page)

```

    Begin Nonlinear Iteration
        Advance myAriaRegion
        transfer aria_to_fuego
        Advance myFuegoRegion
        transfer fuego_to_aria
    End Nonlinear Iteration
End
End

Begin Parameters for Nonlinear Iteration
    Converged when "myFuegoRegion.MaxInitialNonlinearResidual(0) < 1E-
→1 && \$
                                myAriaRegion.MaxInitialNonlinearResidual(0) <_
→1E-1 "
End

Begin Parameters For Transient Time_Block_1
    Start Time      = 0.0
    Termination Time = 10.0

    #Aria region-specific parameters for Time_Block_1
    Begin Parameters For Aria Region myAriaRegion
        Time Step Variation      = adaptive
        Initial Time Step Size = 0.001
        Predictor-Corrector Tolerance = 1e-3
        Minimum Time Step Size      = 1e-6
    End

    #Fuego region-specific parameters for Time_Block_1
    Begin Parameters For Fuego Region myFuegoRegion
        transient step type is automatic
        CFL LIMIT = 1.0
        Time Step Change Factor = 1.2
    End
End

End Solution Control Description

Begin transfer fuego_to_aria
    # transfer commands
    ...
End transfer fuego_to_aria

```

(continues on next page)

```

Begin transfer aria_to_fuego
  # transfer commands
  ...
End transfer aria_to_fuego

  ...
End Procedure My_Aria_Procedure
End Sierra myJob

```



Note: Note here that a subblock for parameters of each region participating in the transient is present in the Parameters block of the transient.

Depending on the two physics being coupled, the two regions can have differing time scales. In this case a *Subcycle* command block can be used to subcycle the region with a smaller timescale, rather than forcing the larger timestep region to follow this smaller step. The solution will advance when the two time applications arrive at the same solution time.

Subcycled Transient Input

```

Begin Sierra myJob

  # Materials, Solvers, Finite Element Model, etc
  ...

  Begin Procedure myProcedure

    Begin Solution Control Description

      Use System Main

      Begin System Main
        Simulation Start Time           = 0.0
        Simulation Termination Time     = 10.0
        Simulation Max Global Iterations = 1000

      Begin Transient Time_Block_1
        Transfer Fuego_to_Aria
        Advance My_Aria_Region
        Transfer Aria_to_Fuego
        Begin Subcycle FuegoSubcycle

```

(continues on next page)

```

        Advance FuegoRegion
      End
    End

  End

  Begin Parameters For Transient Time_Block_1
    Start Time      = 0.0
    Number of steps = 8
    Initial deltat = 0.01

    Begin Parameters For Aria Region My_Aria_Region
      Time Step Variation = Fixed
      Initial Time Step Size = 1.
    End

    Begin Parameters for Fuego Region FuegoRegion
      transient step type is fixed
      maximum time step = 0.01
    End
  End

End

Begin transfer fuego_to_aria
...
End transfer fuego_to_aria

Begin transfer aria_to_fuego
...
End transfer aria_to_fuego

Begin Aria Region myAriaRegion
...
End Aria Region myAriaRegion

Begin Fuego Region myFuegoRegion
...
End Fuego Region myFuegoRegion

End Procedure myProcedure
End Sierra myJob

```

Variable Initialization

In the case of transient analysis it is sometimes necessary to initialize a distribution of values before the analysis actually begins. This can be useful, for example, to transfer a field containing initial conditions to a coupled region. This is accomplished by using an *Initialize* command block. An example of this case is provided below

Initialization Transient Input

```
Begin Sierra myJob

# Materials, Solvers, Finite Element Model, etc
...

Begin Procedure My_Aria_Procedure

  Begin Solution Control Description

    Use System Main

    begin initialize init
      Advance FuegoRegion
      Transfer Fuego_to_Aria
    end initialize init

  Begin System Main

    # Note that the system calls out the initialize to use
    Use initialize init

    Simulation Start Time           = 0.0
    Simulation Termination Time      = 10.0
    Simulation Max Global Iterations = 1000

    Begin Transient Time_Block_1
      Transfer Fuego_to_Aria
      Advance My_Aria_Region
      Transfer Aria_to_Fuego
      Begin Subcycle FuegoSubcycle
        Advance FuegoRegion
      End
    End

  End

End
```

(continues on next page)

```

Begin Parameters For Transient Time_Block_1
  Start Time      = 0.0
  Number of steps = 8
  Initial deltat = 0.01

Begin Parameters For Aria Region My_Aria_Region
  Time Step Variation = Fixed
  Initial Time Step Size = 1.
End

Begin Parameters for Fuego Region FuegoRegion
  transient step type is fixed
  maximum time step = 0.01
End
End
End

Begin transfer fuego_to_aria
...
End transfer fuego_to_aria

Begin transfer aria_to_fuego
...
End transfer aria_to_fuego

Begin Aria Region myAriaRegion
...
End Aria Region myAriaRegion

Begin Fuego Region myFuegoRegion
...
End Fuego Region myFuegoRegion

End Procedure myProcedure
End Sierra myJob

```

4.6 Time Control

As an alternate to *solution control*, Fuego has a simplified Time Control block that can be used to set up time stepping parameters for simple, non-coupled problems. If your problem involves transfer, or if you want to be able to change time stepping parameters at different time periods you should use *solution control*.

```
Begin Fuego Procedure Fuego_procedure

    Time Start = 0.0, Stop = 10.0, Status Interval = 10

    begin time control
        BEGIN TIME STEPPING BLOCK time_block
            START TIME IS 0.0
            TIME STEP = 1e-3
            BEGIN PARAMETERS FOR FUEGO REGION fuego_region
                TRANSIENT STEP TYPE IS automatic
                CFL LIMIT = 5.0
                TIME STEP CHANGE FACTOR = 1.5
            END PARAMETERS FOR FUEGO REGION fuego_region
        END TIME STEPPING BLOCK time_block

        Termination time = 1e6
    end time control
```

For the complete list of commands available in the time control block, refer to the *command summary*. The time control block lets you set the time stepping mode (*fixed* or *automatic*), the CFL limit, the initial time step, and the simulation end time.

4.7 Transfer

Transfer allows you to send fields from one region to another. There are two kinds of transfer - copy and interpolate. Copy transfer requires identical meshes (same node/element locations and same global IDs), while interpolation interpolates fields on dissimilar meshes. The benefit of the copy transfer is that a search is not required to transfer the information. With interpolation on the other hand, the sender and receiver need not be of same topology but the source and target destinations should overlap geometrically so that a search can be performed to find entities to interpolate.

The specification of this choice is as follows

```
Begin Transfer Transfer_name
    [TRANSFER_TYPE] [PART_TYPE] [TOPOLOGY] From [SRC_REGION] To [DST_REGION]
```

(continues on next page)

```
...
End
```

Here SRC_REGION/DST_REGION are the participating regions, TRANSFER_TYPE can be COPY or INTERPOLATE, PART_TYPE can be SURFACE or VOLUME, and TOPOLOGY can be CONSTRAINTS, ELEMENTS, or NODES. The remainder of this section discusses the major components that must be present in a Transfer block. See the [command reference](#) for a detailed list of all possible options.

Field

First, one must specify which quantities are to be transferred across regions. Each field transfer will have the general form

```
Send Field [SRC_FIELD] State [SRC_STATE] To [DST_FIELD] State [DST_STATE]
```

The SRC_FIELD/DST_FIELD can be any Fuego field. See the [command reference](#) for more details on specifying a send field.

The SRC_STATE/DST_STATE refer to the state of the field (or NONE if the field is not stated).

Location

The source and destination parts dictate what portion of the domain is transferred. These parts be specified using the [Send Blocks](#) line command.

Note that for copy transfer, each part must be specified in unique *send block* lines

```
# Scope: Sierra > Procedure
Begin Transfer foo
  Send Block block_1 to block_1
  Send Block block_2 to block_2
  ...
End
```

For interpolation transfer on the other hand, they must be specified in one line

```
# Scope: Sierra > Procedure
Begin Transfer foo
  Send Block block_1 block_2 to block_3 block_4
  ...
End
```

Note that in either case, assemblies can be used to simplify the definitions. Alternatively, the transfer send and receive blocks can be specified in block form using [Send Blocks](#) and [Receive](#)

Blocks respectively. This form is useful when one wishes to send or receive all blocks in the mesh, since it allows the use of the `Include All Blocks` alias instead of having to list them manually.

```
# Scope: Sierra > Procedure
Begin Transfer foo
  Begin Send Blocks
    Include All Blocks
  End
  Begin Receive Blocks
    Include Block = block_3 block_4
  End
  ...
End
```

Note that if these lines are omitted, the transfer is performed over the entire domain.

Search Tolerance

In cases of interpolation, a search over the two meshes must be performed to communicate values. By default, the tolerance used in this search is a function of the geometric problem size. For cases in which the source and destination do not overlap, one should specify tolerancing in order to guarantee that values will be mapped to the destination. If the separation between source and destination varies, then one should specify a *Geometric Tolerance* consistent with the largest separation distance.

Extrapolation Handling

When source and destination entities partially overlap, then by default source values will be extrapolated to the destination. This behavior can be explicitly controlled with the *Nodes Outside Region* command. The most common options for the command are described below:

EXTRAPOLATE

This is the default behavior. The sending field is extrapolated beyond the bounds of the sending mesh. This can lead to extrapolation error, such as when a large gradient at the surface causes a negative values when only positive values are acceptable. If this happens, upper and lower bounds can be set per field within the `SEND FIELD` command.

```
Send Field field1 State none To field2 State none Lower Bound 0.0
↪Upper Bound 1.0
```

TRUNCATE

The receiving coordinate is projected back to the surface of the sending mesh to determine a

value. This ensures that the receiving value is not outside of the field values in the sending mesh.

ABORT

If any receiving point is outside the sending mesh by more than the geometric tolerance, abort the simulation. Do not attempt to project, extrapolate, or otherwise handle the point. This can be useful for detecting problem setup errors that otherwise may have been silently extrapolated.

IGNORE

The receiving mesh object can be ignored and will receive no value. This is usually not recommended as it can cause mesh objects just outside to have a zero value when the nodes just inside the mesh might have very large values.

Skeleton Examples

Since several different uses of transfer can arise, some examples are included below.



Note: While this example shows only steady problems, the same basic setup of transfer would apply to transient problems as well.

A skeleton outline of one-way transfer between two regions

```
# Scope: Sierra > Procedure
Begin Solution Control Description
  Use System Main
  Begin System Main
    Begin Sequential MySolveBlock
      Advance first_Region
      transfer my_transfer
      Advance second_Region
    End
  End
  ...
End

Begin Fuego Region first_region
  Use Finite Element Model fem_model_A
  ...
End

Begin Fuego Region second_region
```

(continues on next page)

```

    Use Finite Element Model fem_model_B

    ...
End

Begin Transfer my_transfer
    interpolate volume nodes from first_region to second_region
    send block block_1 to block_1
    send field Afield state new to Bfield state new
End

```

A skeleton outline of two-way transfer between two regions:

```

# Scope: Sierra > Procedure
Begin Solution Control Description
    Use System Main
    Begin System Main
        Begin Sequential MySolveBlock
            Advance RegionA
            Transfer AtoB
            Advance RegionB
            Transfer BtoA
        End
    End
    ...
End

Begin Fuego Region RegionA
    Use Finite Element Model same_model

    ...
End

Begin Aria Region RegionB
    Use Finite Element Model same_model

    ...
End

Begin Transfer AtoB
    copy volume nodes from first_region to second_region
    send block block_1 to block_1
    send field Afield state new to Bfield state new

```

(continues on next page)

End**Begin** Transfer BtoA

```
copy volume nodes from second_region to first_region
send block block_2 to block_2
send field Bsendfield state new to Arecvfield state new
```

End

Transfer is not exclusive to Fuego regions. For example, an *IO Region* can be used to distribute data from external mesh databases.

4.8 Units

By default, Fuego assumes you are operating in centimeter-gram-seconds units (CGS) and Kelvin. If you are using *basic properties* there is no conversion done, but if you are using *Cantera* or *Tabulated* properties, they are converted to your problem units.



Note: It is always recommended you explicitly define your units in the Fuego Region block to avoid accidental unit errors.

Begin Fuego Region Fuego_region

```
Set unit system to MKS
```

When you have a unit system set, Fuego will also use NIST constants for things like the Stefan-Boltzmann constant so you don't have to supply that yourself.

The available pre-defined unit systems are MKS and CGS. To use a custom unit system, you can specify conversion factors for length, mass, and time manually. Some examples of this are shown below.

Begin Fuego Region Fuego_region

```
# Does the same thing as "SET UNIT SYSTEM TO MKS"
SET LENGTH UNIT CONVERSION FACTOR = 100.0
SET MASS    UNIT CONVERSION FACTOR = 1000.0
SET TIME    UNIT CONVERSION FACTOR = 1.0
SET TEMPERATURE UNITS = KELVIN
```

```
Begin Fuego Region Fuego_region
```

```
# Does the same thing as "SET UNIT SYSTEM TO CGS"  
SET LENGTH UNIT CONVERSION FACTOR = 1.0  
SET MASS    UNIT CONVERSION FACTOR = 1.0  
SET TIME    UNIT CONVERSION FACTOR = 1.0  
SET TEMPERATURE UNITS = KELVIN
```

```
Begin Fuego Region Fuego_region
```

```
# use a non-standard mm-g-s unit system  
SET LENGTH UNIT CONVERSION FACTOR = 0.1  
SET MASS    UNIT CONVERSION FACTOR = 1.0  
SET TIME    UNIT CONVERSION FACTOR = 1.0
```

4.9 Solution Options

The Solution Options block in the Fuego region is where you define what equations to solve, sub-models to use, and nonlinear and linear solver configurations. The most commonly used commands in this block are:

- What equations you want to solve
- What linear solver to use for each equation
- The problem coordinate system
- Source terms for your equations
- What smoothing method to use for the pressure solve
- What advection methods to use for each transport equation
- Nonlinear iteration settings (tolerances and iteration counts)
- *EDC combustion model* parameters
- Turbulence model specifications
- Buoyancy model specifications

```
# Sierra > Fuego Procedure > Fuego Region  
# Non-isothermal 2D ksgs example solution options  
Begin Solution Options  
Coordinate System = 2D  
Projection Method = Fourth_Order Smoothing with Timestep Scaling
```

(continues on next page)

```

# DEFINE WHAT EQUATIONS TO SOLVE
Activate Equation Enthalpy

# DEFINE NONLINEAR ITERATION SETTINGS
Minimum Number of Nonlinear Iterations = 1
Maximum Number of Nonlinear Iterations = 2

# DEFINE ADVECTION SCHEME
Upwind Method is MUSCL
UPWIND LIMITER IS SuperBee

Begin Turbulence Model Specification
    Turbulence Model = KSGS
End

End    Solution Options

```

The most common commands are described in the sections below, and the complete list of available commands for the solution options block and the syntax for the different options can be found in *Solution Control Description*.

Equations To Solve

```

# Sierra > Fuego Procedure > Fuego Region
Begin Solution Options

# DEFINE WHAT EQUATIONS TO SOLVE
Activate Equation Continuity
Activate Equation X_Momentum
Activate Equation Y_Momentum
Activate Equation Turbulent_Kinetic_Energy

```

Equations are activated with the `Activate Equation X` command. Every Fuego simulation must have the continuity and momentum equations active, so these are optional to include in `Solution Options`.

The various turbulence models require different equations to be activated. These are noted in the list below, but Fuego will activate them automatically for you (and will warn and ignore any you specify that are not valid for your selected turbulence model).

Possible equations that can be activated are:

- Continuity - Solves mass conservation for pressure - always active by default.

- **X_Momentum** - Solves x-momentum conservation for **x_velocity** - always active by default.
- **Y_Momentum** - Solves y-momentum conservation for **y_velocity** - always active by default.
- **Z_Momentum** - Solves z-momentum conservation for **z_velocity** - always active by default in 3D, inactive in 2D.
- **Enthalpy** - Solves enthalpy conservation for **enthalpy**, activates a post-processor utility to compute **temperature** after each solve.
- **Species** - Solves $N - 1$ species conservation equations for **mass_fraction**, the last species mass fraction is evaluated analytically to ensure the mass fractions sum to 1.
- **Soot** - Solves transport of soot mass fraction for the *EDC soot model*.
- **Nuclei** - Solves transport of soot nuclei number density for the *EDC soot model*. Will be activated automatically if **soot** is activated.
- **EDC_Product** - Solves transport of the EDC product species, used with the *EDC combustion model*. Not commonly used.
- **Mixture_Fraction** - Solves transport of the first fuel mixture fraction when using a *flamelet combustion model*
- **Second_Mixture_Fraction** - Solves transport of the second fuel mixture fraction when using a *flamelet combustion model*
- **Scalar_Variance** - Solves transport of scalar variance when using a *flamelet combustion model*
- **Conserved_Enthalpy** - Solves transport of conserved enthalpy when using a *flamelet combustion model*
- **Progress_Variable** - Solves transport of an arbitrary scalar quantity. This is sometimes used with a *flamelet combustion model*, but can also be used independently to transport an arbitrary scalar quantity.
- **Turbulent_Kinetic_Energy** - Solves transport of turbulent kinetic energy for **turbulent_kinetic_energy**. Required for all turbulence models except SMAG and DSMAG. Will be activated automatically based on the selected turbulence model.
- **Turbulent_Dissipation** - Solves transport of turbulent dissipation for **turbulent_dissipation**. Required for KE, LRKE, and V2F. Will be activated automatically based on the selected turbulence model.
- **Turbulent_Frequency** - Solves transport of turbulent frequency for **turbulent_frequency**. Required for KW, LRKW, SST, LRSST, and SSTDES. Will be activated automatically based on the selected turbulence model.
- **Turbulent_V2** - Solves transport of turbulent frequency for **turbulent_v2**. Required for V2F. Will be activated automatically based on the selected turbulence model.

- **Turbulent_Helmholtz_Function** - Solves transport of turbulent frequency for `turbulent_f`. Required for V2F. Will be activated automatically based on the selected turbulence model.
- **Volume_Of_Fluid** - Solves for a volume fraction of liquid, `volume_of_fluid`, used for multi-phase simulations.
- **X_Solid_Momentum** - Solves for conservation of solid momentum in the x-direction for the granular phase model.
- **Y_Solid_Momentum** - Solves for conservation of solid momentum in the y-direction for the granular phase model.
- **Z_Solid_Momentum** - Solves for conservation of solid momentum in the z-direction for the granular phase model.
- **Solid_Volume_Fraction** - Solves for conservation of solid volume fraction for the granular phase model.

Linear Solvers

Linear solvers are defined *at the sierra level*. In **Solution Options** you can choose which solver to use for each activated equation.

```
# Sierra > Fuego Procedure > Fuego Region
```

```
Begin Solution Options
```

```
USE EQUATION SOLVER muelu_cont FOR EQUATION Continuity
```



Note: The solver lines are optional. If you omit this line for any equation, Fuego will use a default preset solver appropriate for that transport equation.

Coordinate System

The problem coordinate system is specified in the **Solution Options** block in Fuego. The default coordinate system is 3D, so for 3D problems this line can be omitted. The options are 3D, 2D, XAXI, and YAXI.

```
# Sierra > Fuego Procedure > Fuego Region
```

```
Begin Solution Options
```

```
Coordinate System = 2D
```

Source terms

Source terms for the different activated equations can be set in a few different ways - external sources, subroutine or function sources, and point sources. All of these are specified in the Solution Options block.

External Sources

External sources are activated with the `Use External ...` commands below. This will create a nodal source field (*e.g.* `energy_source`) and it is expected that the relevant source values will be transferred into that field from an external region.

Begin Solution Options

```
Use External Continuity Source
Use External Energy Source
Use External Mixture_Fraction Source
Use External Momentum Source
Use External Soot_Mass_Fraction Source
Use External Species Source
```

Subroutines and String Functions

User subroutines can be defined and used as sources using the following command

Begin Solution Options

```
#Source term subroutine = NAME_OF_SUB for equation NAME_OF_EQ [NAME_OF_
→VAR]
Source term subroutine = my_enth_src for equation Enthalpy
Source term subroutine = my_pv_src for equation Progress_Variable my_var
Source term subroutine = my_methane_src for equation Species CH4
```

Similarly, user-defined string functions can be used to define custom source terms

Begin Solution Options

```
#Source term function = "fcn_content" for equation NAME_OF_EQ [NAME_OF_
→VAR]
SOURCE TERM FUNCTION = "x+0*y_velocity" FOR EQUATION X_Momentum
SOURCE TERM FUNCTION = "-10*pv_test" FOR EQUATION progress_variable pv_
→test
```

(continues on next page)

(continued from previous page)

```
SOURCE TERM FUNCTION = "point2d(x-1.2, y+0.3, 0.01, 0.02)*1e-3" FOR
→EQUATION Species CH4
```

The string functions can include time and coordinates (t, x, y, z) and any nodal variables.

Point Sources

Point sources are designed to inject fluid at a given point with specified properties. Each point source requires a location, Mdot, and relevant primitive variables to inject.

Begin Solution Options

Begin Point Source InjectionA

```
Location = sphere at 1.2 -0.3 0 radius = 0.05
Mdot = "3e-5 + 1e-4*t"      # kg/s
temperature = 450
mass_fraction N2A = 0.0
mass_fraction N2B = 1.0
mass_fraction N2C = 0.0
turbulent_kinetic_energy = 1e-2
```

End

The Location can be either `sphere at x y z radius = R` or `node nearest x y z`. If the sphere is used, the source is applied evenly on all nodes inside the specified sphere. If there are no nodes inside the specified sphere, the nearest node is used.

If mesh motion is active, the nodes where the source is applied will be selected only once, using the initial mesh coordinates, so the source will move with the mesh.

The complete list of commands for the point source can be found in [Point Source](#).

Smoothing Method

There are a few different options for smoothing algorithms to select for the continuity equation, as defined in [the theory section on smoothing](#).

The most common choice, which works well for the majority of problems, is fourth order smoothing with timestep scaling (Equation (3.760), as shown in the example below).

Begin Solution Options

```
Projection Method = Fourth_Order Smoothing with Timestep Scaling
```

(continues on next page)

(continued from previous page)

Another common choice is fourth order smoothing with characteristic scaling (Equation :eq:\`numerics-characteristic-scaling\`):

Begin Solution Options

Projection Method = Fourth_Order Smoothing with Characteristic Scaling

A third, less-common choice is *stabilized smoothing*, activated using:

Begin Solution Options

Projection Method = Stabilized SMOOTHING

Advection Methods

Consider transport of an arbitrary scalar ϕ between the control volumes C_0 and C_1 in the figure below.

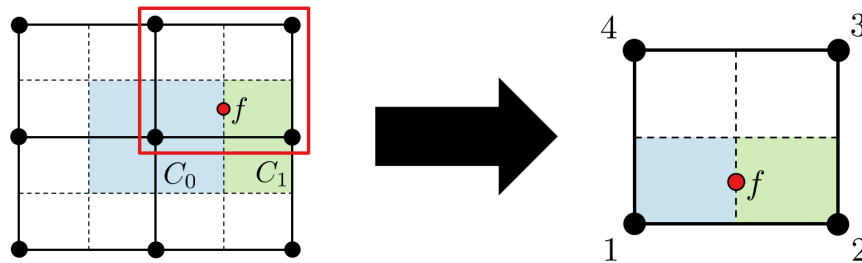


Fig. 4.2: Control volumes for flux across a single face f

In order to calculate the advection of ϕ we need to be able to evaluate $\dot{m}_f \phi_f$. The mass flow rate through the face (\dot{m}_f) is determined by our continuity solve, but for each transport scalar we need to determine ϕ_f . The scheme used to find ϕ_f can come in a number of different flavors. The basic ones are described in the following sections, followed by the blending scheme Fuego uses to combine them.

Central Difference

Using a central difference scheme (CDS) we simply interpolate ϕ from the node locations to the face location using the element weights w_i , so

$$\phi_f^{\text{CDS}} = \sum_i w_i \phi_i \quad (4.13)$$

This provides a second order accurate advection operator, but poor stability properties in regimes with high Peclet numbers.

Upwind Method

Using only the value of the upwind node from the face provides a simple, stable, inexpensive approach, but it is only first order accurate.

$$\phi_f^{\text{UPW}} = \begin{cases} \phi_1 & \dot{m}_f > 0 \\ \phi_2 & \dot{m}_f < 0 \end{cases} \quad (4.14)$$

MUSCL Upwind

Using MUSCL (Monotonic Upstream-centered Scheme for Conservation Laws) we use the gradient of ϕ at the upwind node to project a value of ϕ_f from the node location.

$$\phi_f^{\text{MUSCL}} = \begin{cases} \phi_1 + \Phi(r_1) (\mathbf{r}_{1,f} \cdot \nabla \phi_1) & \dot{m}_f > 0 \\ \phi_2 + \Phi(r_2) (\mathbf{r}_{2,f} \cdot \nabla \phi_2) & \dot{m}_f < 0 \end{cases} \quad (4.15)$$

where we also require a limiter function, $\Phi(r)$ that uses

$$r_1 = \frac{\phi_1 - \hat{\phi}_1}{\phi_2 - \phi_1} \quad (4.16)$$

$$\hat{\phi}_1 = \phi_2 - 2\nabla \phi_1 \cdot (\mathbf{x}_2 - \mathbf{x}_1) \quad (4.17)$$

Fuego Blending Approach

The cell Peclet number is the ratio of advection to diffusion, defined as

$$Pe = \frac{\mathbf{u} \cdot \delta \mathbf{x}}{\nu} \quad (4.18)$$

Fuego uses several blending functions to combine the three advection operators as a function of cell Peclet number.

$$\phi_f = \eta \phi_f^{\text{UPW}} + (1 - \eta) \left[\chi \phi_f^{\text{US}} + (1 - \chi) \phi_f^{\text{CDS}} \right] \quad (4.19)$$

Here, η is the user-specified first order upwind factor (a value between 0 and 1), χ is a central difference blending coefficient that transitions between 0 and 1 depending on the cell Peclet number (described below), and ϕ_f^{US} is the user-specified upwind choice, either ϕ_f^{UPW} or ϕ_f^{MUSCL} .

When you set η , you define what fraction of the advection operator is always first-order upwind, and the remaining fraction is then split between your choice of upwind operator (UPW or MUSCL) and central-difference based on χ .

The blending coefficient is either defined by

$$\chi = \frac{(\zeta Pe)^2}{5 + (\zeta Pe)^2} \quad (4.20)$$

where ζ is specified by HYBRID UPWIND FACTOR and can be any value greater than zero, or by

$$\chi = \frac{1}{2} \left[1 + \tanh \left(\frac{Pe - S}{W} \right) \right] \quad (4.21)$$

where S is defined by HYBRID UPWIND SHIFT and W is defined by HYBRID UPWIND WIDTH.

These parameters can be set globally, or per equation. The per-equation specification takes precedence, so you can set a global configuration, and then override it only for certain equations.

Begin Solution Options

```
Upwind Method = [UPW | MUSCL] # Default "UPW"
First Order Upwind Factor = [0 to 1] # Default 0
Hybrid Upwind Factor = [0+] # Default 1
Upwind Limiter = [None, Superbee, Minmod, Van_Al bada, Van_Leer] #
↳ Default "None"
Hybrid Upwind Method = [Blending | Tanh] # Default "Blending"
Hybrid Upwind Shift = X # Default 0
Hybrid Upwind Width = X # Default 0
```

An example configuration below uses no first-order upwinding, and blends between MUSCL and CDS using the TANH blending rule with different transition points for momentum and the other equations.

Begin Solution Options

```
UPWIND METHOD IS MUSCL
UPWIND LIMITER IS VAN_LEER

FIRST ORDER UPWIND FACTOR = 0.0 # 0% upwinding

HYBRID UPWIND METHOD = TANH
HYBRID UPWIND WIDTH = 2
HYBRID UPWIND SHIFT = 2
```

(continues on next page)

```

HYBRID UPWIND WIDTH = 200 FOR EQUATION X_Momentum
HYBRID UPWIND WIDTH = 200 FOR EQUATION Y_Momentum

HYBRID UPWIND SHIFT = 2000 FOR EQUATION X_Momentum
HYBRID UPWIND SHIFT = 2000 FOR EQUATION Y_Momentum

```

Nonlinear Iteration Settings

The solution options block is where you can define the number of nonlinear iterations and convergence criteria for your active equations. The default nonlinear residual norm tolerance is very small (10^{-15}) so if you do not specify it, Fuego will generally just do the maximum number of iterations defined by `Maximum Number of Nonlinear iterations`. Note that these residuals are dimensional, so scaling a problem from meters to centimeters would require different residual targets to maintain the same behavior.

Begin Solution Options

```

Minimum Number of Nonlinear iterations = 2
Maximum Number of Nonlinear iterations = 4

Nonlinear Residual Norm Tolerance = 1e-6 for Equation Continuity
Nonlinear Residual Norm Tolerance = 1e-6 for Equation X_Momentum
Nonlinear Residual Norm Tolerance = 1e-6 for Equation Y_Momentum
Nonlinear Residual Norm Tolerance = 1e-6 for Equation Z_Momentum

```

You can also control iterations of groups of related equations, which by default only take one iteration per group.

Begin Solution Options

```

Maximum Number of Continuity_Momentum Nonlinear Iterations = 3

```

The sub-groups available to iterate with this command are:

- CONTINUITY_MOMENTUM
- SPECIES
- SPECIES_PRODUCT
- ENERGY_SPECIES
- SOLID PHASE
- GAS_SOLID_MOMENTUM

- MIXTURE FRACTION
- SOOT NUCLEI
- KEPSILON
- KSGS
- V2F
- KOMEGA

Turbulence Model

The Turbulence Model Specification block in solution options is where you define which turbulence model you want to use, along with any global model configuration settings. If you omit this block, it defaults to LAM (no turbulence model). The available options for the model are LAM, KE, LRKE, KW, LRKW, SST, LRSST, RNG, V2F, SSTDES, SMAG, DSMAG, KSGS, DKSGS, and INAGAKI_KSGS.

Begin Solution Options

```
Begin Turbulence Model Specification my_params
    Turbulence Model = KSGS
End
```

The complete list of available commands for the solution options block and the syntax for the different options can be found in *Turbulence Model Specification*.

Buoyancy Model

The Buoyancy Model Specification block in solution options is where you define which buoyancy model you want to use.

Begin Solution Options

```
Begin Buoyancy Model Specification my_buoy
    Buoyancy Model = Differential
End
```

Details about the Buoyancy models can be found in *Conservation of Momentum*. The available options for the buoyancy model are:

- NO_BUOYANCY

$$\int_V \rho g dV = 0$$

- BUOYANT

$$\int_V \rho g dV = \rho g V$$

- DIFFERENTIAL

$$\int_V \rho g dV = g V (\rho - \rho_o)$$

- BOUSSINESQ

$$\int_V \rho g dV = \frac{\rho_o g V}{T_o} (T_o - T)$$

If you use the differential or Boussinesq model you also need to provide reference properties used to calculate ρ_o (and/or T_o).

Begin Solution Options

```
Begin Buoyancy Model Specification my_buoy
  Buoyancy Model = Boussinesq
  Buoyancy Reference Temperature = 300
End
```

Begin Solution Options

```
Begin Buoyancy Model Specification my_buoy
  Buoyancy Model = Differential
  Buoyancy Reference Temperature = 300
  Buoyancy Reference Mass_Fraction O2 = 0.2
  Buoyancy Reference Mass_Fraction N2 = 0.8
  Buoyancy Reference Mass_Fraction He = 0.0
End
```

4.10 Boundary Conditions

Every exposed surface in your mesh must have a sideset on it, and every sideset must have a boundary condition applied. Fuego will throw an error at runtime if these conditions are not met. Boundary conditions are specified in the Fuego Region block, and the available options are described in this section.

Open

An open boundary is a specified-pressure surface that allows flow both into and out of the domain. When the flow is out of the domain (as determined by the continuity solve using the specified pressure) then the conserved quantities are advected out and none of the other values specified in the open boundary condition are used.

When the flow is into the domain, the composition of the entrained fluid is defined by the values you set in the open boundary condition (*e.g.* the entrained enthalpy will be calculated using the specified temperature and mass fraction in the example below).

```
begin open boundary condition on surface surface_4
  pressure = 0.0
  temperature = 300
  mass_fraction He = 0.0
  mass_fraction O2 = 0.233
  mass_fraction N2 = 0.767
end
```

Most specified values can be constants, in-line string functions, tabulated functions, or user subroutines. The complete list of available commands for the open boundary condition and the syntax for the different options can be found in *Open Boundary Condition On Surface*.

Entrained Momentum

By default, the entrained momentum is zero. This is usually the most stable choice, but can make the open boundary condition more dissipative. If you specify non-zero velocities in the open boundary condition, those will be used for the entrained momentum.

There is an additional option one can specify on the open boundary condition to have the entrained momentum be computed from the calculated mass flux and local fluid density. This can improve accuracy and reduce dissipation from the boundary condition, but may sometimes lead to instabilities forming at the open boundary.

```
begin open boundary condition on surface surface_4
  Momentum entrainment = Computed
  # ...
end
```

The options for this command are **Specified** (default), **Computed**, or **Tangential**.

- **Specified** Using this model the entrained velocity (u_e) is the specified value in the boundary condition block (u_s), which is zero by default.

$$u_e = u_s \quad (4.22)$$

- **Computed** Using this model the entrained velocity is calculated using the local density, face area, and calculated mass flow rate in a direction normal to the surface.

$$u_e = \frac{\dot{m}}{\rho A} \hat{n} \quad (4.23)$$

- **Tangential** Using this model takes only the tangential component of the specified velocity. This can be used to prevent normal injection of momentum on non-planar open boundaries.

$$u_e = u_s - (u_s \cdot \hat{n}) \hat{n} \quad (4.24)$$

Entrained Turbulence Quantities

For turbulent flows, you can either set the turbulent quantity directly (*e.g.* turbulent kinetic energy = 1e-3) or you can specify a characteristic length (L) and optional turbulence intensity (I , 10% by default), which will be used to calculate the entrained turbulent quantities using the following relationships.

$$k_e = 1.5 (Iu)^2 \quad (4.25)$$

$$\epsilon_e = \frac{k_e^{1.5} C_\mu^{0.75}}{0.07L} \quad (4.26)$$

$$\omega_e = \frac{\sqrt{k_e}}{\beta^{*0.25} 0.07L} \quad (4.27)$$

Total Pressure

Instead of specifying a pressure, one can set a total pressure on the open boundary instead, where total pressure is defined as

$$p_t = p + \frac{1}{2} \rho u^2 \quad (4.28)$$

Sponge Layer



Beta Capability

The sponge layer is not a well-used or well-tested feature and may introduce artifacts or other undesirable behavior into the solution. Use it with caution.

A so-called “sponge layer” is a technique used to reduce vorticity in the flow as it approaches an open boundary condition, to improve numerical stability. The sponge layer modifies the viscosity

applied in the momentum equations anisotropically near the boundary, increasing it in the tangential directions to damp out oscillations in the flow tangential to the open boundary. This viscosity modification can either be a multiple of the current viscosity, or an absolute value. The sponge layer viscosity modification can be activated abruptly at the specified thickness with a Heaviside function, or blended in linearly as the flow approaches the boundary.

A sponge layer can be activated at an open boundary using the following commands.

```
begin open boundary condition on surface surface_4
  Use Sponge layer
  Sponge Layer Thickness = 0.05
  Sponge Layer Viscosity = 1e-3
  Sponge Layer Viscosity Type = Absolute # [Multiple | Absolute]
  Sponge Layer Blending Function = Linear # [Heaviside | Linear]
  # ...
end
```

Inflow

An inflow boundary is a specified velocity surface that flows into the domain. The properties of the conserved quantities flowing in are calculated using the specified primitive variables on the boundary.

```
begin inflow boundary condition on surface surface_4
  x_velocity = 1.2
  y_velocity = 0
  z_velocity = 0
  temperature = 300
  mass_fraction He = 0.0
  mass_fraction O2 = 0.233
  mass_fraction N2 = 0.767
end
```

The complete list of available commands for the inflow boundary condition and the syntax for the different options can be found in *Inflow Boundary Condition On Surface*.

Use Fluxes



Note: Using the `Use Fluxes` command is a recommended best practice, and may become the default behavior in a future version.

By default, the inflow properties are applied as a dirichlet condition on the nodes on the inflow boundary. If you add the `Use Fluxes` argument, the boundary condition is applied as a mass infusion into the control volumes along the inflow.

```
begin inflow boundary condition on surface surface_4
  Use Fluxes
  # ...
end
```

To further understand the difference, consider the following inflow boundary condition

```
begin inflow boundary condition on surface surface_4
  x_velocity = "(t<5) ? 0 : 1.2"
  y_velocity = 0
  z_velocity = 0
  temperature = 300
  mass_fraction He = 1.0
  mass_fraction O2 = 0.0
  mass_fraction N2 = 0.0
end
```

In this, we have a flow of pure Helium that turns on at 5 seconds. Prior to 5 seconds, the inflow velocity is 0. When using the `Use Fluxes` command, the flow will be un-affected by this boundary condition until 5 seconds. However, using the default form the mass fraction of Helium will be fixed to 1 immediately and Helium will diffusion into the domain before the 5-second activation.

This is also important for other conditions, such as a slow jet in a cross-flow. With the `Use Fluxes` command the concentration of Helium at the boundary nodes will be computed as a balance between the cross-flow into those control volumes and the boundary condition inflow. However, using the dirichlet form that balance will be discarded and those nodes will be fixed at a Helium mass fraction of 1.

Wall

A wall boundary is a boundary through which no advective mass flux is allowed. The wall itself can be stationary, or have a specified velocity tangential to the flow. It can be isothermal, or have a specified `wall_temperature`, for simulations including an enthalpy equation.

```
begin wall boundary condition on surface surface_4
    wall_temperature = 400
end
```

If you specify a value on the wall for a mass fraction, mixture fraction, or progress variable there will be a dirichlet boundary condition applied on the wall nodes for that scalar, which means that there will be diffusion of that scalar through the wall. If this is not desired, do not specify a value on the wall boundary condition.

The complete list of available commands for the wall boundary condition and the syntax for the different options can be found in [Wall Boundary Condition On Surface](#).

Turbulent Wall Modeling

Wall-modeled (as opposed to “wall-resolved”) turbulent simulations use a boundary layer model at the wall. The choice of whether a wall model is used is defined in the `Turbulence Model` block in `Solution Options`. There are two wall-model related options one can specify per boundary. They are `Use equilibrium production model`, which sets the rule that turbulent production and dissipation are equal at the wall, and `law_of_wall_roughness_parameter`, which sets a dimensionless roughness parameter used in the wall models (E , default is 9.8). For more details, see the [wall modeling](#) section in the theory chapter.

```
begin wall boundary condition on surface surface_4
    # ...
    Use equilibrium production model
    law_of_wall_roughness_parameter = 9.8
end
```

Conjugate Heat Transfer

When a wall provides a conjugate heat transfer interface to Sierra/Aria, you can indicate this with the `interface boundary` command. This signals to Fuego that `wall_temperature` will be supplied via a transfer from an external region.

```
begin wall boundary condition on surface surface_4
    # ...
```

(continues on next page)

```

Interface boundary
end

```

Convection Coefficients

By default Fuego calculates linearized flux terms (`flux_linearization_coefficient` and `flux_linearization_temperature`) on the wall in a form that resemble a convection equation, but using a near-wall temperature instead of a far-field reference temperature. This means that the coefficient C here will be mesh-dependent, and will not correspond to an expected convection coefficient from a correlation.

$$q = C(T_{\text{near,wall}} - T_{\text{wall}}) \quad (4.29)$$

For comparison with a conventional correlation for h one would want something of the form

$$q = h(T_{\text{ref}} - T_{\text{wall}}) \quad (4.30)$$

Fuego has a special post-processor for calculating h in this manner on the wall boundaries, with a user-specified T_{ref} . This will produce a `wall_convection_coefficient` nodal field that can be output to Exodus or used in other post-processors.

```

begin wall boundary condition on surface surface_4
  # ...
  Calculate Convection Coefficient using Tref = 350
end

```

Wall Mass Inject

A wall mass-inject boundary is a special kind of wall that allows specified or modeled mass injection. It is like a normal wall in terms of boundary layer treatment, with the addition of a mass injection model that adds mass at specified properties into the control volumes adjacent to the wall nodes.

```

begin wall mass inject boundary condition on surface surface_4
  mass flux = 0.1 # kg/m2/s
  wall_temperature = 300.
  injected temperature = 450.0
  mass_fraction N2 = 0.8
  mass_fraction O2 = 0.2
end

```

The wall mass inject boundary can operate in two modes:

- **Specified Mass Injection** - The user specifies a mass flux, and optional injected temperature, and this is applied uniformly on the boundary. The specified mass flux can be a string function.
- **Pool Mass Injection** - The user sets up a pool model, which models an evaporating pool of fuel. The evaporation model for the pool defines the mass injection rate and temperature as a function of heat flux to the wall.

The complete list of available commands for the wall mass inject boundary condition and the syntax for the different options can be found in *Wall Mass Inject Boundary Condition On Surface*.

Specified Injection

In this mode the user specifies a mass flux (mass per area per time) and optional injected temperature. If the injected temperature is omitted, the injected mass uses the specified wall temperature.

```
begin wall mass inject boundary condition on surface surface_4
  mass flux = 0.1 # kg/m2/s
  injected temperature = 450.0
  # ...
end
```

Pool Model



Note: Units listed below are in MKS for illustration, but should be in whatever unit system your problem is in.

Details about the pool model can be found in *Fuel Boundary Condition Submodel*. Setting up the pool model requires defining a number of pool parameters. These cannot be mixed with the specified injection commands.

```
begin wall mass inject boundary condition on surface surface_4
  pool initial_height = 0.05
  pool initial_temperature = 300.0
  pool ignition_time = 0
  pool evaporation_temperature = 400.0
  pool absorption_coefficient = 1.0
  pool heat_of_evaporation = 1.0e5
  pool specific_heat = 500
```

(continues on next page)

```

pool boil_flash_temperature_difference = 5
pool percent_sensible = 1.0 # all sensible heating until boiling
# ...
end

```

Given a total heat flux to the pool surface as (using the fuel absorption coefficient α from the specified `absorption_coefficient`) we can sum the convective and radiative fluxes to get the total flux, q'' (W/m²) as:

$$q'' = q''_{conv} + q''_{rad} = q''_{conv} + \alpha (q''_{irrad} - \sigma T_f^4), \quad (4.31)$$

Next we can define what percent of that goes to sensible heating (f_s , clipped to lie between 0 and 1) using the user-defined `percent_sensible` (p_s), boiling temperature (T_b , from `evaporation_temperature`), and boil flash temperature difference (ΔT_{bf} , from `boil_flash_temperature_difference`) as:

$$f_s = \min \left(p_s, \frac{T_b - T_f}{\Delta T_{bf}} \right) \quad (4.32)$$

With the heat of vaporization of the fuel (h_{evap} , from `heat_of_evaporation`, J/kg) we can then define the evaporation rate (\dot{m}'' , kg/(m²s)) as

$$\dot{m}'' = \frac{q''(1 - f_s)}{h_{evap}}, \quad (4.33)$$

the time rate of change of the pool temperature (T_f) using the fuel density (ρ_f) fuel specific heat ($C_{p,f}$) and pool height (H) as

$$\frac{dT_f}{dt} = \frac{q'' f_s}{H \rho_f C_{p,f}}, \quad (4.34)$$

and the time rate of change of the pool height as

$$\frac{dH}{dt} = -\frac{\dot{m}''}{\rho_f}. \quad (4.35)$$

The pool height is reduced from the specified initial height until it reaches zero, at which point there is no more evaporation.

The `percent_sensible` parameter lets the user control how much of the incident flux goes into heating the pool vs evaporation. If it is set to 1, the pool will not evaporate until it is near the boiling temperature, at which point f_s will drop and evaporation will begin. If it is set to 0, the pool temperature will not increase and it will begin evaporating immediately.

Prior to the specified `ignition_time`, the pool evaporation model is inactive and the specified `min_injection_mdot` or `mass_flux` is applied instead so that there can be some fuel added to the domain to artificially kick-start ignition.



Warning: The `min_injection_mdot` is a mass flow rate (kg/s) per integration point. This will be mesh dependent (refining the mesh and using the same value will change the mass injected before ignition time). Using `mass_flux` is preferred over `min_injection_mdot`, but if both are supplied then `min_injection_mdot` is still used.

Symmetry

A symmetry boundary automatically applies a zero-gradient symmetry condition to all equations. No arguments are needed for a symmetry boundary.

```
begin symmetry boundary condition on surface surface_4
end
```

Periodic

A periodic boundary condition is tied to an opposing sideset such that flow out of one side of the domain enters the other side of the domain. The periodic boundary should list another sideset to tie itself to, and the usual rule about all sidesets having an applied boundary condition is adjusted slightly in this case, and the tied surface does not need a separate boundary condition block.

```
begin periodic boundary condition on surface surface_4
  paired with surface_1
  Search tolerance = 1e-6
end
```



Note: Periodic boundary conditions tie nodal values together, so the two sidesets must have identical meshes. The sidesets will be translated based on the vector connecting the sideset centroids and each node must match locations with a translated node to within the specified search tolerance.

The complete list of available commands for the periodic boundary condition and the syntax for the different options can be found in *Periodic Boundary Condition On Surface*.

Mass Flux

A mass flux boundary is very similar to an open boundary, except that instead of calculating the mass flux using the specified pressure and continuity equation, the mass flux is specified by the user. When the flow is out of the domain (positive mass flux) then the conserved quantities are advected out and none of the other values specified in the open boundary condition are used. When the flow is into the domain (negative mass flux), the composition of the entrained fluid is defined by the values you set in the mass flux boundary condition (*e.g.* the entrained enthalpy will be calculated using the specified temperature and mass fraction in the example below).

```
begin mass flux boundary condition on surface surface_4
  mass flux = 1.84
  temperature = 300
  mass_fraction He = 0.0
  mass_fraction O2 = 0.233
  mass_fraction N2 = 0.767
end
```

Most specified values can be constants, in-line string functions, tabulated functions, or user subroutines. The complete list of available commands for the mass flux boundary condition and the syntax for the different options can be found in [Mass Flux Boundary Condition On Surface](#).

Like pressure, entrained turbulence quantities can be specified directly, or using the characteristic length (L) and optional turbulence intensity (I , 10% by default) parameters.



Note: You do not need to specify pressure on a mass flux boundary.

Composite

The composite boundary condition is used with the 1D composite sub-model to simulate a burning and off-gassing composite. It requires a separate material definition for the composite material ([Composite Material Properties](#)) and parameters for the initial composite composition, thickness, and backside thermal boundary condition.

```
begin composite boundary condition on surface surface_4
  composite material = new_composite

  composite nodes = 10
  composite thickness = 0.002 # 2 mm
```

(continues on next page)

(continued from previous page)

```
activate backside convection
activate backside radiation
composite backside_convection_coefficient = 10.0
composite backside_reference_temperature = 800.

# Initially hot to accelerate ignition for regression test
composite initial_temperature = 300.
composite initial_mass_fraction Carbon = 0.6724 # 0.6 volume fraction
composite initial_mass_fraction Epoxy = 0.3276 # 0.4 volume fraction
composite initial_mass_fraction Char = 0.0

# Composition of outgassed mixture
turbulent_kinetic_energy = 2.5e-5
soot_mass_fraction = 0.0
soot_nuclei_mass_fraction = 0.0
end
```

The complete list of available commands for the composite boundary condition and the syntax for the different options can be found in *Composite Boundary Condition On Surface*.

1D Composite Model

The details of the 1D composite model are described in the Theory chapter in *One-Dimensional Composite Fire Boundary Condition*. A pseudo-1D domain is created on the boundary nodes of the composite boundary with the number of nodes and thickness of the domain specified by `composite nodes` and `composite thickness`. The composite material defines a series of solid decomposition reactions that produce gas (and other solids like char). An ODE solver evaluates an energy balance and the decomposition ODEs at each composite node, and the net gas produced is applied as a mass injection term at the composite wall (similar to a mass inject boundary condition).

Composite Interface

In some cases, you may want to use a more complicated 3D composite model in Sierra/Aria. Fuego supports this using the `composite interface` boundary, which expects transfers of composite decomposition products from Aria instead of solving them directly.

```
begin composite interface boundary condition on surface surface_4
  external field for composite_species_mass_rate of CH4 = bc_species_diff_
  →adv_flux_normal_CH4
  external field for composite_species_mass_rate of O2 = bc_species_diff_
```

(continues on next page)

```

→adv_flux_normal_O2
  external field for composite_species_mass_rate of N2 = bc_species_diff_
→adv_flux_normal_N2
end

```

Fixed

The fixed boundary condition sets a dirichlet for specified values and is not commonly used in Fuego except for cases where pressure is ill-defined and needs to be fixed somewhere. In that case, a fixed boundary condition can be used on a single-node nodeset to set the pressure (*e.g.* in a case where your domain has no specified pressure boundaries and is not acoustically compressible, such as where you have all wall and mass flux boundaries).

```

begin fixed boundary condition on surface nodeset_1
  pressure = 0.0
end

```



Note: Fuego will detect if the pressure is not fixed anywhere and automatically add a fixed boundary condition on a single node in the domain. If you want to control which node this is set on, you can define a nodeset and set the boundary condition manually.

The complete list of available commands for the fixed boundary condition and the syntax for the different options can be found in *Fixed Boundary Condition On Surface*.

Heat Flux

The heat flux boundary condition is not a complete fluid boundary condition, but can be used on top of another boundary condition to add a specified heat flux.

```

# Define an adiabatic (or fixed temperature) wall
begin wall boundary condition on surface sideset_4
  # ...
end

# And add a heat flux to it
begin heat flux boundary condition on surface sideset_4
  heat_flux = 500.0 # W/m2
end

```

The value specified should be in W/m^2 (or equivalent for your problem units) and use the convention that positive is a flux into the domain.

The complete list of available commands for the heat flux boundary condition and the syntax for the different options can be found in [Heat Flux Boundary Condition On Surface](#).

Non-Conformal

If you have two physically adjacent but non-merged (separately meshed) surfaces with different sidesets on them, Fuego has a non-conformal boundary condition that applies a discontinuous Galerkin contact condition to tie the two sidesets together (allowing transport across them). These two surfaces do not have to be meshed contiguously, and the non-conformal boundary condition should be applied on both sides of the interface.

```
begin nonconformal boundary condition on surface surface_4
  opposing sides = surface_5
end

begin nonconformal boundary condition on surface surface_5
  opposing sides = surface_4
end
```

The complete list of available commands for the non-conformal boundary condition and the syntax for the different options can be found in [Nonconformal Boundary Condition On Surface](#).

4.11 Initial Conditions

An initial condition for the DOF of every active equation is required on every block. The Initial Condition block goes in the Fuego Region.

```
# Fuego Region scope
Begin Initial Condition Block FlowInit
  Volume is block_1
  Pressure    = 0.0
  X_Velocity  = 0.0
  Y_Velocity  = 0.0
End
```

There can be multiple Volume is lines within a given block, and you can have more than one Initial Condition block to apply different conditions to different blocks.

If you omit an initial condition for any active equation on any block, Fuego will throw an error telling you which equation is missing and which blocks it is missing on.

For the complete list of commands, see the [command summary](#)

4.12 Fire Features

MPMD Radiation

This section outlines the use of Multiple-Program-Multiple-Data (MPMD) coupling to perform radiation solves with [Sierra/PMR](#). MPMD coupling allows the radiation solve to occur on an entirely different set of processors using an executable other than Fuego. In the past, PMR was solved with Syrinx (compiled into the Fuego executable). That has been deprecated and removed in favor of MPMD coupling - first with Sierra/Nalu and Scefir, and now with Sierra/PMR.

Input Deck Conversion from Nalu to Sierra/PMR

The new Sierra/PMR capability maintains backwards compatibility with Nalu input decks, so conversion from Sierra/Nalu to Sierra/PMR for the PMR solve requires simply changing the name of the executable in your launch scripts from `nalu` to `pmr`.

Input Deck Conversion from Syrinx to Sierra/PMR

The Conversion of a Fuego/Syrinx input deck to a Fuego MPMD input deck is straightforward. The basic steps to convert a Syrinx case to a Fuego-PMR case are:

1. Delete the Syrinx region, PMR material, PMR solver, and PMR mesh (finite element) blocks from the Fuego input deck.
2. Delete all transfer blocks to or from the PMR region and all calls to them from solution control (see [Transfers](#)).
3. Add the `USE MPMD RADIATION` command to the Fuego input deck in the Fuego region block to activate the MPMD coupling.
4. Add the `SKIP STEPS FOR PMR` command in the Fuego procedure or time blocks if you want to run the PMR solve less frequently than every time step.
5. Specify emissivity, transmissivity, radiation environment temperature, and radiation boundary temperature in the Fuego boundary condition blocks. If any of these are omitted defaults will be used. Default conditions are emissivity = 1, transmissivity = 0, and boundary temperature equal to the local or wall temperature depending on the boundary type.
6. Copy the template PMR input file from [PMR Input Deck](#) and update the name of the mesh file.
7. (Optional) Update the quadrature and numerical scheme in the PMR input deck per your problem requirements.

8. (Optional) If using a particle region, transfer absorption and radiation sources from the particle region to the Fuego region, and transfer scalar flux from the Fuego region to the particle region (see *Particle Radiation Terms*).
9. Run your case using an MPMD launch command (see *Running MPMD Jobs*).

Transfers

In prior Fuego-Syrinx simulations the user would specifically call out which fields to transfer between Fuego and Syrinx in the input file, or call out a pre-defined transfer that would automatically define fields to transfer. With MPMD PMR the transfers are always determined automatically, so nothing is required in the input deck with regards to transfers to or from the PMR code.

The Fuego to PMR transfers are:

- **Radiative source** (rad_source, R_s). This is calculated by Fuego either from the temperature field or combustion model (EDC or mixture fraction) and is sent on all volume nodes. For the non-reacting laminar case, this term is

$$R_s = \frac{\sigma}{\pi} \alpha T^4 \quad (4.36)$$

and for reacting cases it is either tabulated or otherwise modified to include sub-grid effects.

- **Absorption coefficient** (absorption, α). Calculated using the current property model and sent on all volume nodes.
- **Emissivity**. Calculated in Fuego and sent on all surface nodes.
- **Transmissivity**. Calculated in Fuego and sent on all surface nodes.
- **Boundary Source**. Calculated in Fuego using the emissivity (ϵ), transmissivity (τ), radiation boundary temperature (T_b), radiation environment temperature (T_e), and band fractions (f_e , f_b) if doing a spectral calculation. This is sent on all surface nodes.

$$S_{BC} = \sigma \left(\epsilon T_b^4 f_b + \tau T_e^4 f_e \right) \quad (4.37)$$

- **Boundary Beam Source**. Calculated in Fuego on all surface nodes only if the beam model is in use.

The PMR to Fuego transfers are:

- **Scalar flux** (scalar_flux, G). Calculated by the PMR code and sent back on all volume nodes, this is used to apply the radiative source term to the enthalpy equation.

$$S_h = V_{scv} (\alpha G - 4\pi R_s) \quad (4.38)$$

This source term is linearized using the laminar form of R_s to give

$$S_{h,lhs} = V_{scv} \frac{16\alpha\sigma T^3}{C_p} \quad (4.39)$$

- **Radiative Flux** (`radiative_flux`). Calculated by the PMR code this flux vector is sent back on all volume nodes and is primarily used for post-processing.
- **Incident Flux** (`incident_flux`). The incident flux on all surfaces is calculated the PMR code and sent back to Fuego.

PMR Input Deck

The corresponding PMR input deck for Fuego/PMR MPMD calculations is shown in the following sections. The PMR input file is YAML formatted, so spaces and indentation matter. The complete user manual for Sierra/PMR can be found in the PMR user manual at <https://pmr.sandia.gov>, so details about the input options will be omitted here for brevity.

```
simulation_settings:
  mesh: mesh.g
  method: edge_upwind

quadrature:
  type: thurgood
  order: 4
```

Running MPMD Jobs

The command to run in MPMD mode is different from what was used to run Fuego-Syrinx cases. To run Fuego-Syrinx cases you simply ran a parallel job of Fuego using something like

```
mpirun -np 100 fuego -i fire.i
```

In this case one executable would use 100 cores, so both Fuego and Syrinx were taking turns using the same CPU resources. With MPMD runs you are launching two separate MPI jobs with two different codes that can communicate. An example MPMD launch command would look like

```
mpirun -np 100 fuego -i fire.i : -np 100 pmr -i pmr.i
```

However, there is no requirement any more that the two codes use the same number of cores, so depending on the mesh and computational costs you may choose a different allocation per code. For example, if your PMR solve is very expensive you may allocate more cores to PMR than Fuego:

```
mpirun -np 50 fuego -i fire.i : -np 150 pmr -i pmr.i
```

Special care must be taken when submitting MPMD jobs on the HPCs or any queued environment. By default, the two MPMD codes cannot share cores so to launch the case above on an HPC you would need to request an allocation 200 cores. This is unnecessarily wasteful though

since Sierra/PMR would not be using its 150 cores while Fuego runs, and Fuego would not be using its 50 cores while Sierra/PMR runs. To get around this, you must enable oversubscription. Since Fuego and Sierra/PMR run sequentially (never executing at the same time) you can allow them to share resources. To get an allocation of 100 cores and use them all for both codes you must add additional mpi flags:

```
mpiexec --oversubscribe --bind-to core:overload-allowed -np 100 \
fuego -i fire.i : --bind-to core:overload-allowed -np 100 pmr -i pmr.i
```

Keep in mind that the specific command to use can be platform dependent. A more complete example submission script on an HPC may look like

```
#!/bin/bash

#SBATCH --nodes=10
#SBATCH --time=48:00:00
#SBATCH --account=xxxxxxx
#SBATCH --job-name=fire
#SBATCH --partition=batch

nodes=$SLURM_JOB_NUM_NODES
cores=36

module load sierra
export OMPI_MCA_rmaps_base_oversubscribe=1
mpiexec --oversubscribe \
  --bind-to core:overload-allowed \
  --npernode $cores --n $((($cores*$nodes)) fuego -i fire.i : \
  --bind-to core:overload-allowed \
  --npernode $cores --n $((($cores*$nodes)) pmr -i pmr.i
```

On vortex or Sierra, an explicit resource file is required to use MPMD. An example *fuego_pmr.erf* explicit resource file is

```
overlapping-rs : allow
oversubscribe-cpu : allow
app 0 : fuego -i fire.i
app 1 : pmr -i pmr.i
  4 : {host: *; cpu: *; gpu: * } : app 0
  4 : {host: *; cpu: *; gpu: * } : app 1
```

where both apps are set to use 4 ranks on each node and allowed to use both the CPU and GPU. Using this file, a submission script on vortex could look like

```
#!/bin/bash
#BSUB -nnodes 4

module load sierra
export CUDA_LAUNCH_BLOCKING=0
export CUDA_MANAGED_FORCE_DEVICE_ALLOC=1
export TPETRA_ASSUME_GPU_AWARE_MPI=1

rm -f stderr-b stdout-b
jsrun -M "-gpu" -k stderr-b -o stdout-b --erf_input=fuego_pmr.erf
```

Contact sierra-help@sandia.gov if you need more help or encounter issues running MPMD jobs.

Spectral Radiation Transport



Beta Capability

Spectral radiation transport is a beta feature, with limited testing.

Fuego now supports spectral radiation calculations through MPMD coupling with Sierra/PMR. Spectral radiation transport is instantiated in an MPMD Fuego input deck through a spectral model section similar to the following:

```
BEGIN RAD TRANSPORT SPECTRAL MODEL SPECIFICATION spectralRadModel
  Spectral Band Model = LINEAR
  Spectral Subband Model = AVERAGE
  Minimum Frequency = 1.0e13
  Maximum Frequency = 2.0e13
  Number Spectral Bands = 2
END RAD TRANSPORT SPECTRAL MODEL SPECIFICATION
```

The spectral model utilized in Fuego is based on the concept of spectral bands, which means that the user must describe frequency bands over which radiation absorptivities are averaged. In this context, a richer set of data is available to the analyst, including radiation transport terms (banded absorptivity, scalar flux, and radiative flux) than is available through the standard gray (nonbanded) radiation transport description that has been available to Fuego analysts for many years.

Spectral transport makes use of spectral data files that the user must provide. Files are named as: `Species-data.dat`

Where “Species” is the name of the chemical species whose spectral transport properties are

being described. Each species present (all species for EDC simulations) or those output through the use of output variables in mixture fraction simulations must have all necessary spectral data files available in the working directory when a spectral transport calculation is performed. Spectral data files have the following structure:

Table 4.1: Spectral file format

T_1	$F_{1,1}$	$A_{1,1}$
...		
T_1	F_{1,N_1}	A_{1,N_1}
...		
T_M	$F_{M,1}$	$A_{M,1}$
...		
T_M	F_{M,N_M}	A_{M,N_M}

For each temperature T_x of the M temperatures specified in the spectral data files, there are N_X frequencies ($F_{x,1}$ through F_{x,N_X}) specified with corresponding absorptivity values ($A_{x,1}$ through A_{x,N_X}). Temperatures within these files are required to be in non-decreasing order and frequencies must be ordered in increasing value within a specific temperature. Absorptivities are expected to be in units of $cm^2/molecule$, which is a number density weighted absorptivity. Unit conversion within the Fuego input deck is required if the desired units of the simulation are not cgs. The ideal gas law and knowledge of the mass fraction of each species is used to calculate the number density.

For the spectral radiation model specified here, the name of the particular spectral model is given as `spectralRadModel`, but could be any valid string. For this case, a `LINEAR` “Spectral Band Model” is used, where the set of spectral bands, in this case 2 as specified by “Number of Spectral Bands” are equally divided along the linear spectral frequency range with “Minimum Frequency = $1.0e13$ ” Hz and “Maximum Frequency = $2.0e13$ ” Hz. One could alternatively choose a `LOGARITHMIC` Spectral Band Model where bands are divided up equally in logarithmic frequency space.

In this case, we have chosen an `AVERAGE` “Spectral Subband Model”, which indicates that averaging of absorptivities for frequencies within a spectral band will follow standard (unweighted) averaging over the particular spectral band. A `PLANCK_AVERAGE` Spectral Subband Model is also available where integrations are weighted by their Planck black-body spectrum contribution. `PLANCK_AVERAGE_WITH_REFERENCE_TEMPERATURE` is a final choice that a user can select where the band contributions are weighted by the Planck black body-spectrum at a specified user reference temperature. If this final option is used, a user must also specify a reference temperature, which can be done through: Planck Subband Model Reference Temperature = 2000.0 Here the Reference temperature is set to 2000.0 K, but could be any nonzero temperature.

Spectral properties on surfaces



Beta Capability

Spectral surface properties is a beta feature, with limited testing.

Fuego can now perform spectral radiation calculations with banded emissivities defined on surfaces. This is a beta feature, and still requires additional testing. A command similar to the one shown below is used to define banded emissivities on surfaces:

```
BEGIN WALL BOUNDARY CONDITION ON SURFACE SURFACE_1
INTERFACE BOUNDARY
EMISSIVITY SPECTRAL FILE NAME = SURFACE_1_SPECTRAL.DATA
TRANSPARENT BAND EMISSIVITY = 1.0
END WALL BOUNDARY CONDITION ON SURFACE SURFACE_1
```

Similar to the spectral radiation model described in Section *Spectral Radiation Transport*, a spectral data file containing the emissivity bands must be provided for each surface (provided by the `EMISSIVITY SPECTRAL FILE NAME =` command in the above). These files have a structure similar to the species spectral files:

Table 4.2: Spectral file example

T_1	$F_{1,1}$	$\varepsilon_{1,1}$
...		
T_1	F_{1,N_1}	ε_{1,N_1}
...		
T_M	$F_{M,1}$	$\varepsilon_{M,1}$
...		
T_M	F_{M,N_M}	ε_{M,N_M}

Similar to the species spectral files, each temperature T_x of the M temperatures specified in the spectral file have N_x frequencies defined, each with a corresponding emissivity value ($\varepsilon_{x,1}$) through ε_{x,N_x} . Temperatures within these files must be defined in a non-decreasing order, with frequencies within each temperature group ordered in increasing value.

In addition to defining a spectral emissivity file, an additional emissivity for the transparent band can be defined as a general expression; this value is defaulted to 1. In the above example, the transparent band emissivity is defined as a constant (of 1).

Particle Radiation Terms

Particle Radiation contributions are also still available in Fuego simulations with Lagrangian particle types that have radiation contributions (heated, CPD, evaporating, general chemistry, wildfire). Previously, the user would specify transfers of necessary thermal/radiation fields between Fuego and Syrinx, which would resemble the following:

```
BEGIN TRANSFER pmr_to_particle
  COPY VOLUME NODES FROM pmr_region TO particle_region
  SEND BLOCK block_1 TO block_1
  SEND FIELD scalar_flux STATE none TO scalar_flux STATE new
END   TRANSFER pmr_to_particle

BEGIN TRANSFER particle_to_pmr
  COPY VOLUME NODES FROM particle_region to pmr_region
  send field particle_absorption_coeff state new to particle_absorption_
  ↪coeff state none
  SEND FIELD particle_rte_source STATE  new TO particle_rte_source_
  ↪STATE none
END TRANSFER particle_to_pmr
```

For MPMD Radiation transport simulations involving Lagrangian particles, the user now removes the PMR (Syrinx) to particle transfers and instead adds these to the already-present set of fluid to particle transfers as is seen in the following:

```
BEGIN TRANSFER fluid_to_particle
  COPY VOLUME NODES FROM fluid_region TO particle_region
  SEND BLOCK block_1 TO block_1
  #...
  SEND FIELD scalar_flux STATE none TO scalar_flux STATE new
END   TRANSFER fluid_to_particle

BEGIN TRANSFER particle_to_fluid
  COPY VOLUME NODES FROM particle_region TO fluid_region
  SEND BLOCK block_1 TO block_1
  #...
  send field particle_absorption_coeff state new to particle_absorption_
  ↪coeff state none
  send field particle_rte_source state new to particle_rte_source state_
  ↪none
END   TRANSFER particle_to_fluid
```

Beam (Directed Flux)

Fuego users are able to now specify beam or directed flux boundary conditions for use in MPMD simulations with PMR. A beam boundary condition is specified in the Fuego input deck in the following manner:

```
#CONVERGING/DIVERGING BEAM
BEGIN BEAM RADIATION BOUNDARY SPECIFICATION
  BEAM RADIATION BOUNDARY MODEL = FOCUSED
  FOCAL POINT = FX FY FZ
  FOCAL POWER = FP
  FOCUSED BEAM TYPE = DIVERGING (OR CONVERGING)
  CONSTRAIN TO SURFACES = sideA sideB ...
  BEAM TEMPERATURE = BT
END BEAM RADIATION BOUNDARY SPECIFICATION
```

In the above case, a focused beam boundary `BEAM RADIATION BOUNDARY MODEL = FOCUSED` is defined where the `FOCAL POINT` of the beam, either sourcing from or arriving at this point are (`FX`, `FY`, `FZ`). The user specifies whether the beam diverges from this point or converges to this focal point. For the `DIVERGING` beam, the focal point needs to lie outside the simulation domain, whereas for the `CONVERGING` case, the focal point could lie in or outside the domain. The focal power is specified as `FP` which is the integrated power of the beam assuming the intensity is spherically isotropic. For a converging case, the power should be properly scaled to respect the actual solid angle over which the beam converges to the focal point. For instance if the converging beam actually occupies a solid angle of π , the `FOCAL POWER` should be scaled up by a factor of $4 = 4\pi/\pi$ since if the beam were incident from all solid angles, the total beam power would increase by this factor. Beams can be constrained to arrive through the domain through only a fixed set of external surfaces if so desired as indicated through `CONSTRAIN TO SURFACES = sideA sideB ...`. The user also specifies the `BEAM TEMPERATURE` for cases using spectral radiation transport, since the beam power must then be parceled out among the defined spectral bands.

Another “directed” or beam-type boundary condition is available, where the user specifies not a focused beam but rather a plane wave as is seen below:

```
#PLANE WAVE
BEGIN BEAM RADIATION BOUNDARY SPECIFICATION
  BEAM RADIATION BOUNDARY MODEL = PLANE_WAVE
  PLANE WAVE BEAM DIRECTION = WX WY WZ
  PLANE WAVE INTENSITY = PWI
  CONSTRAIN TO SURFACES = sideA sideB ...
  BEAM TEMPERATURE = 3000.0
END BEAM RADIATION BOUNDARY SPECIFICATION
```

In this case, the `BEAM RADIATION BOUNDARY MODEL = PLANE_WAVE`, indicates a plane-wave type source is to be used. The user also specifies the direction for the plane wave as `PLANE WAVE`

BEAM DIRECTION = WX WY WZ. The direction must be non-zero. The intensity of the plane-wave in units of Power per area is defined through PLANE WAVE INTENSITY = PWI. The ability to constrain the beam to only certain external surfaces is identical to that for the focused beam above as is the beam temperature.

One should keep in mind that for both beam types (focused and plane wave), beam vectors at external surfaces are only calculated for nodes on surfaces where the inner (dot) product of outwardly directed area vector and the beam flux vector is less than 0, indicating that a user specified beam must enter rather than exit the simulation domain. Of course, given the physics internal to the domain of simulation, a radiative flux vector can exit the simulation, but we disallow the possibility of a user specifying an outwardly directed beam that does not result from internal physics.

EDC Combustion Model

The eddy-dissipation concept model in Fuego can be activated through a block in solution options describing with a number of associated options. The EDC fuel name determines the species considered to be “fuel” by the underlying chemistry model used by EDC and must match a species name, see *Cantera Material Properties* for details on how to define a set of species in Fuego.

```
# Sierra > Fuego Procedure > Fuego Region > Solution Options
BEGIN EDC MODEL SPECIFICATION
  EDC FUEL NAME = C8H18
  EDC MINIMUM PRODUCT FRACTION = 1.0e-6
  EDC IGNITION TIME = 0.0
  EDC REACTION TIME SCALE = 70.0e-6
  USE SINTEF SOOT MODEL
  MINIMUM SOOT PRODUCTION TEMPERATURE = 900.0
  ACTIVATE ABSORPTION MODEL
  INCLUDE EDC LAMINAR LIMIT MODEL
END
```

Soot and absorption models can also be activated in this command block, with the added requirement that the soot mass fraction and soot nuclei are also activated. Some necessary or useful equations to solve with EDC are

- **Enthalpy** - Solves enthalpy conservation for `enthalpy`, activates a post-processor utility to compute `temperature` after each solve. Required with EDC.
- **Species** - Solves $N - 1$ species conservation equations for `mass_fraction`, the last species mass fraction is evaluated analytically to ensure the mass fractions sum to 1. Required with EDC. One of the species must match the EDC fuel name. See *Cantera Material Properties* for defining transported species.
- **Soot** - Solves transport of soot mass fraction. Needed for soot transport with the EDC model. See *Soot Generation Model for Multicomponent Combustion* for more information.

- `Nuclei` - Solves transport of soot nuclei number density. Necessary for soot transport with the EDC model. Activated automatically if soot is activated.
- `EDC_Product` - Solves transport of the EDC product species. Not commonly used but is needed if product species are being injected into the domain or are present in significant quantities in the ambient.

Additional parameters specific to the EDC model, described in more detail in *EDC Turbulent Combustion Model*, can also be set in this block. Some commonly used options are

- `EDC reaction time scale` - Controls the residence time for the EDC extinction model
- `EDC ignition time` - Offset time for when the EDC model activates (turning on combustion).
- `Minimum soot production temperature` - Minimum temperature at which soot forms.
- `EDC minimum product fraction` - A minimum product mass fraction threshold for EDC.
- `Include EDC laminar limit model` - The EDC model is designed for fully turbulent combustion, and the laminar limit model prevents near singular behavior in near laminar regions of the flow in a RANS context and near fully resolved regions in an LES context

More options are available to be set in this block and can be seen in *command references*.

Flamelet Combustion Model

The use of a flamelet-based combustion model requires generation of a flamelet table, in which details of the fire chemistry have been precomputed. See *Tabulated Material Properties* for more details on how to generate a flamelet-based model for a particular simulation scenario.

This flamelet model encapsulated in the table will contain a number simplifications of the physics in order to make the problem tractable. A common simplification, for example, is that the scenario is decomposable into a single, fixed oxidizer state (commonly called a “stream”) and a single, fixed fuel state. The additional states would require a different flamelet model to be constructed, resulting in a different table, and more transport equations to be described in fuego. For example, a second mixture fraction would need to be solved if an extra inlet with a diluted mixture of heptane was added to a previously pure heptane fire problem.

Equations describing flamelet-based fire scenarios

An example set of an equations to solve for a flamelet case is

```
# Sierra > Fuego Procedure > Fuego Region
Begin Solution Options
  Activate Equation Mixture_Fraction
  Activate Equation Enthalpy
  Activate Equation Conserved_Enthalpy
  Activate Equation Progress_Variable
  Activate Equation Turbulent_Kinetic_Energy

Begin Turbulence Model Specification
  Turbulence Model = KSGS
End

...

End
```



Note: The buoyancy reference state will still need to correspond to the flamelet reference values, usually air with zero heat loss.

No specific options are required in solution options for the flamelet model, other than specifying an appropriate set of equations. Options describing the flamelet are defined in the table creation process and in the *Tabulated Material Properties* section of the input file. Equations you may want to solve:

- **Mixture_Fraction** - Necessary for all flamelet models. Solves transport of the first fuel mixture fraction when using a flamelet combustion model. With a soot model, it's common to add a sink term to the equation representing carbon leaving the gas-phase and becoming soot, see *Aksit-Moss SNL soot model*.
- **Enthalpy** - Solves conservation for enthalpy and activates a post-processor utility to compute temperature after each solve. Radiative heat loss due to gas radiation and soot can be added as well, using *MPMD Radiation* for radiative heat loss. Wall heat loss can be included in two forms. If the temperature of the wall is somewhere between the temperature of a fully burning or fully extinguished flame, then an enthalpy defect heat loss model can capture the heat loss (or gain, potentially) due to the wall. If this is not the case, then a boundary heat loss model would need to be added.
- **Conserved_Enthalpy** - Solves transport for conserved enthalpy—in other words, the enthalpy without loss terms. This equation is part of the heat loss model used by the table. For an enthalpy defect model, the difference between the enthalpy and conserved enthalpy is

used a measure of the integrated heat loss that a fluid parcel has undergone. This can also be used with the boundary heat loss model to describe temperature changes in the reference states of the flamelet model. These two models can also be combined to describe both effects in a “5-dimensional” model, see [Nonadiabatic Property Table Generation](#). It’s important to understand which heat loss effects are encapsulated in the underlying flamelet model as some physical effects may not be able to be captured with the simpler heat loss model. More complicated heat loss models maybe prohibitively expensive and less frequently used.

- **Progress_Variable** - Solves transport of an arbitrary scalar quantity. *Source terms* can be attached to this equation to model for instance the number density and mass fraction of soot in the *Aksit-Moss SNL soot model*. Note that these progress variables are meant to be only weakly coupled with the flow and chemistry, modeling relatively slowly evolving quantities like soot or NOX. This is a different (but related) definition of progress variable compared to what is used in a flamelet-progress variable combustion model.
- **Turbulent_Kinetic_Energy** - Solves the transport equation for the subgrid turbulent kinetic energy, with closure models for higher degree moment terms. This is part of the large-eddy simulation modeling suite and is not necessary for a flamelet calculation. This equation specification is also used to describe to overall turbulent kinetic energy in a RANS context. An algebraic model like Smagorinsky can also be used to avoid solving this equation, with modeling implications.
- **Scalar_Variance** - Solves the transport equation for the subgrid scalar variance, with closure models for higher degree moment terms. An algebraic model is used if this equation is not solved. In a RANS context, this equation will instead model the overall scalar variance.

Boundary condition modifications

Additional boundary conditions are needed for each extra transport equation being solved, except for the enthalpy and conserved enthalpy pair of equations. For those equations, it is only necessary to specify the temperature and the equivalent enthalpy defect and boundary heat loss will be computed given the current or reference composition of the gas at the boundary. An example *Inflow* boundary condition would be

```
Begin inflow boundary condition on surface surface_4
  x_velocity = 0.0
  y_velocity = 0.0
  z_velocity = 0.1
  mixture_fraction = 1.0
  scalar_variance = 1.0e-6
  turbulent_kinetic_energy = 1.0e-6
  temperature = 300.0
  progress_variable soot_moles_per_mass = 0.0
```

(continues on next page)

```

progress_variable soot_mass_fraction = 0.0
End

```

4.13 Post-Processing

Fuego supports a few different post-processing options, described in the sections below.

Boundary Fluxes

All boundary condition blocks support adding integrated flux postprocessors for all equations, for either total flux, advective flux, or diffusive flux. The post-processed quantity will be a global variable you can output in a *heartbeat* or *results output* block, or use in a subsequent *global function* or in another boundary condition string function (*e.g.* to tie the flux out of one BC to the incoming flux of another BC).

```

Begin open boundary condition on Surface surface_4
# ...

# Postprocess [TOTAL | ADVECTIVE | DIFFUSIVE] flux of [EQ_NAME] as
→ [GVAR_NAME]
Postprocess total flux of continuity as mdot4
End

```

Total flux is calculated by constructing the equation residual on the boundary nodes, minus boundary condition contributions. This approach is the most accurate, and will always return the exact flux actually applied through the boundary. If you have a specified flux, this post-processed value may differ from the specified value by an amount on the order of your linear solver tolerance, and will be an accurate indicator of how well converged your equation is. One exception to this is *continuity*, which will automatically use the advective approach for total flux.

Advective flux is calculated by using the *mdot* calculated on the boundary from the continuity equation, multiplied by the advected quantity on the boundary. The advective flux is not relevant on boundary conditions with no advective flux (*e.g.* wall, symmetry), so requesting it there will result in an error.

Diffusive flux is calculated as the difference between total flux (using the residual approach) and the advective flux.

The complete list of flux post-processing commands can be found in each boundary condition block in the *command summary*.

Enthalpy

When post-processing the flux of enthalpy, there is an additional optional command to calculate average temperature on the boundary as well. To activate this option, add a second variable name (Tavg here).

```
Begin open boundary condition on Surface surface_4
# ...

# Postprocess [TOTAL | ADVECTIVE | DIFFUSIVE] flux of enthalpy as [H_
↪NAME] [T_NAME]
Postprocess total flux of enthalpy as hFlux Tavg
End
```

Species

When post-processing the flux of species, the post-processor runs for all species and you must supply aliases for all the resolved species.



Note: The fracbal species (the last one) does not have an equation and should not be included in the post-processed list. If you need its flux, you can calculate total flux and subtract the resolved species.

```
Begin open boundary condition on Surface surface_4
# ...

# Example when your species are CO2, O2, N2
Postprocess total flux of species as mdot_CO2_s4 mdot_O2_s4
Postprocess total flux of continuity as mdot_s4
End

Begin postprocess Global
Output name = mdot_N2_s4
Function = "mdot_s4 - mdot_CO2_s4 - mdot_O2_s4"
End
```

General Post-Processing

Fuego supports a number of general post-processing options. These are specified in the Fuego Region block, and are evaluated in the order listed in the input file. This means that if you have dependencies between post-processors you need to ensure they are ordered in the way you want them to be evaluated.

```
Begin postprocess PP_TYPE
  Output name = var_name
  Function = ""
  Location = block_4
End
```

The available postprocessor types are:

- **Integral** - The `location` should be blocks or sidesets (not a mix) and the `function` can be a function of time, space, or any nodal variables. This will be integrated on the surface or volume and output as a global variable.

$$F = \int_V f dV$$

or

$$F = \int_A f dA$$

- **Average** - The `location` should be blocks or sidesets (not a mix) and the `function` can be a function of time, space, or any nodal variables. This will be integrated on the surface or volume, then normalized by the area or volume and output as a global variable.

$$F = \frac{\int_V f dV}{\int_V dV}$$

or

$$F = \frac{\int_A f dA}{\int_A dA}$$

- **Sum** - Takes a nodal sum of the specified function on the specified blocks.
- **Min** - Finds the nodal minimum of the specified function on the specified blocks.
- **Max** - Finds the nodal maximum of the specified function on the specified blocks.
- **L2_Norm** - Finds the nodal L2 norm of the specified function on the specified blocks.

$$F = \sqrt{\int_V f^2 dV}$$

or

$$F = \sqrt{\int_A f^2 dA}$$

- **Global** - Evaluates a global function (can be a function of time or other global variables). The `location` block should not be provided for global functions.
- **Nodal_Field** - Evaluates the specified function and outputs it as a nodal field. This can be output, or used in subsequent post-processors.
- **Integrated_Flux** - Evaluates the integrated flux of a specified flux function on a surface (cannot be applied to a volume). The `function` should provide 2 or 3 components (separate quoted strings) depending on the problem dimension.



Note: Using the *boundary condition postprocessors* will always provide a more accurate flux calculation. This should only be used for quantities that are not primary DOFs, or on internal sidesets where there is not a boundary condition.

$$F = \int_A \vec{f} \cdot \vec{dA}$$

```
Begin postprocess Integrated_Flux
  Output name = mass_flux
  Location = surface_1
  Function = "density*x_velocity" \$
            "density*y_velocity" \$
            "density*z_velocity"
End
```

- **Point** - Evaluates the specified nodal function at a specific point in space and saves the result to a global variable. For the point post-processor the `location` should be a list of 2 or 3 coordinates (depending on problem dimension). The specified function will be evaluated on the nodes of the element containing the specified point, and then interpolated to the point. If the specified point lies outside the mesh an error will be thrown.

```
Begin postprocess Point
  Output name = TC1
  Location = 1.2 2.2 0.1
  Function = "temperature"
End
```

The complete list of commands for the post-processors can be found in the *command summary*.

Averaging

The averaging block in the Fuego Region lets you calculate Reynolds and Favre averages of any nodal field, as well as provides post-processors for modeled and resolved turbulent quantities.

BEGIN AVERAGING

```
REYNOLDS AVERAGE FIELD temperature AS reynolds_temperature
REYNOLDS AVERAGE FIELD pressure AS reynolds_pressure
REYNOLDS AVERAGE FIELD radiative_flux AS reynolds_rad_flux
FAVRE AVERAGE FIELD temperature AS favre_temperature
FAVRE AVERAGE FIELD pressure AS favre_pressure
FAVRE AVERAGE FIELD radiative_flux AS favre_rad_flux
TIME INTERVAL LENGTH IS 2.0
STARTING TIME IS 2.0
```

```
Compute resolved_reynolds_stress
Compute resolved_favre_stress
Compute resolved_turbulent_kinetic_energy
Compute resolved_production
Compute resolved_dissipation
Compute resolved_scalar_flux of enthalpy
Compute modeled_reynolds_stress
Compute modeled_scalar_flux of enthalpy
```

END AVERAGING



Note: The filter is reset after it reaches the `time interval length` and starts over again. Filtered values just after that point will not have been averaged over a very long time window yet. To get a continuously averaged quantity, use the `Moving Average Filtered Field` with the `Moving Average Time`

The complete list of commands for the averaging block can be found in the [command summary](#).

Pre-Defined Quantities

You can use the `Compute` command to compute several pre-defined quantities, described in the *command summary*.

4.14 User Customization

User Subroutines



Warning: The use of user subroutines should be considered a last resort, especially for production simulations - most custom capabilities can be implemented using GPU-compatible, well-tested capabilities like string functions.

Fuego supports the inclusion of user subroutines for property evaluation, boundary conditions, and initial conditions. It is worth noting that simulations that use user subroutines will not run fully on the GPU on platforms where they are available and there may be a significant performance penalty relative to using string functions in those cases.

Fuego offers support of C and Fortran user subroutines through fixed-interface signatures. It is important to note that C subroutines provide some level of variable-type safety and bounds checking that is not supported with Fortran subroutines.

The C subroutine API is described in the following sections - Fortran is left as an exercise for the reader.

Building Usersubs

The process for building user subroutines is the same for C and Fortran subroutines. When you invoke the `sierra --make` command, the specified input file is scanned for imported shared object (`.so`) files and they are compiled from source files of the same name.

```
sierra --make fuego -i input.i
```



Note: The shared object generally does not have to be re-compiled between Fuego runs unless you change the contents of the C or F file, however it is a good practice to re-compile them when you change what version of Fuego you are running.

Subroutines can also be compiled using the `buildsub` command, which should produce an equivalent output

```
buildsub -e fuego -i input.i
```

Property Subroutines

A C user subroutine must be a text file of suffix `.C` that will be compiled to a shared object file of the same name with the `.so` extension. For example, a file named `mysubs.C` would be compiled to a file named `mysubs.so` and referenced from the Fuego input file using the line command

```
load user plugin file ./mysubs.so USING FUNCTION register_usersubs
```

Note that, since `mysubs.so` is a filename, its input file name is case-sensitive, and considerations must be made concerning its location path.

An example C file to provide a subroutine for enthalpy is shown below.

```
#include "Afgo_UserSubSig.h"

extern "C" void enthalpy_prop(
    const int * num_results,      // result size (usually 1 or num_species)
    double * result,              // result data
    const int * num_inputs,       // number of inputs (e.g. "2" for
    ↪ "temperature mass_fraction" inputs)
    const unsigned int * inp_size, // size of each input (e.g. [1, num_
    ↪ species] for example above)
    const int * num_input_data,   // total inputs size (e.g. 1+num_species_
    ↪ for example above)
    const double * input_data,    // flattened input data (e.g. [T, Y_0, Y_
    ↪ 1, Y_2] for example above)
    const int * num_user_params,  // size of user supplied data (1 here)
    const double * user_params    // user supplied data (data = [Cp])
)
{
    const double T = input_data[0];
    const double Cp = user_params[0];

    // h = Cp*T
    result[0] = Cp*T;
}

extern "C" void register_usersubs() {
    sierra::Afgo::AfgoPropSub::instance().
        registerFunction(std::string("enthalpy_prop"), enthalpy_prop);
}
```

In the code above, the actual function (named `enthalpy_prop`) is declared with the required signature for a property. The function is then registered for use in Fuego with the following command (which must be after the function declaration)

```
sierra::Afgo::AfgoPropSub::instance().registerFunction
```

The registration function takes two arguments:

- "enthalpy_prop" - The name of the evaluator, to be referenced in the input file.
- `enthalpy_prop` - The function being registered.

Once registered, the subroutine can be referred to in a Fuego material using the registered name:

```
Begin Property Specification for Fuego Material air
# ...

subroutine for enthalpy = enthalpy_prop
input variables for enthalpy_prop = temperature
real data for enthalpy_prop = 500
End
```

Source Subroutines

Subroutines for source terms have a different signature than for properties, and cannot take custom inputs.

An example C file to provide a subroutine for a source term for enthalpy is shown below.

```
#include "Afgo_UserSubSig.h"

extern "C" void enth_src(
    const int * num_results,           // result size (usually 1)
    const int * ndims,                 // problem dimension (2 or 3)
    const int * nnodes,                // number of nodes per element
    const double* coords,              // 2D array of coordinates (ndims X_
    nnodes)
    const double* vol_scv,              // 1D array of scv volumes (nnodes)
    const double* x_vel,                // 1D array of X velocity (nnodes)
    const double* y_vel,                // 1D array of Y velocity (nnodes)
    const double* z_vel,                // 1D array of Z velocity (nnodes)
    const double* density,              // 1D array of density (nnodes)
    const double* pressure,             // 1D array of pressure (nnodes)
    const double* time,                 // Current time (scalar)
    double* lhs,                        // 2D LHS array (nnodes x nnodes)
```

(continues on next page)

(continued from previous page)

```
double* rhs // 1D RHS array (nnodes)
)
{
  for(int n = 0; n < *nnodes; ++n) {
    rhs[n] = vol_scv[n]*500;
  }
}

extern "C" void register_usersubs() {
  sierra::Afgo::FMMSFgoUserSub::instance().
    registerFunction(std::string("my_enth_src"), enth_src);
}
```

This is referred to in the input file in the Solution Options block with

```
Source term subroutine = my_enth_src for EQUATION ENTHALPY
```

IC/BC Subroutines

The IC and BC subroutine API is not currently documented. Use at your own peril, and contact sierra-help@sandia.gov if you need help.

Plugin Utilities



Warning: The use of user utilities should be considered a last resort, especially for production simulations - most custom capabilities can be implemented using GPU-compatible, well-tested capabilities like string functions.

User utilities are another way that users can extend Fuego's capabilities using custom compiled code. Unlike a *user subroutine*, which is evaluated with a very controlled and limited API, a user utility is a standalone class with an `execute` function that can be executed at various points during problem execution. A user subroutine cannot generally access field data or perform MPI calls, but a user utility can do both of those things.

The tradeoff is that the API for a user utility is much more general and open, and as such cannot be completely documented here. This section will provide some guidelines on how to define them, and some example utilities. For more advanced capabilities, you will likely need to involve a Fuego developer.

Writing a Utility

In this section we'll write an example utility to calculate a custom field we will use for a boundary flux. The first part of our plugin C file will define the utility class. The API requirements for this class are:

1. It must inherit from `sierra::Afgo::UserUtility`
2. Its constructor must use the following signature - `MyUtil(sierra::Afgo::Region& r)`
3. It must define a static `String name()` method
4. It must define an execute function as `void execute() override`
5. It may define a field registration function as `void register_utility_variables(const stk::mesh::ConstPartVector& parts) override`, but this is not required

Our class definition looks like:

```
#include "Afgo_UserPlugins.h"
#include "Afgo_DiagWriter.h"
#include "tftk_mesh/tftk_Mesh.h"
#include "sierra/Sierra_FieldType.h"
#include "tftk_mesh/tftk_FieldFunctions.h"

using namespace sierra::Afgo;

class MyCustomUtil : public UserUtility
{
public:
    static sierra::String name() { return "MyCustomUtil"; }
    MyCustomUtil(Region& r);
    void execute() override;
    void register_utility_variables(const stk::mesh::ConstPartVector&
    ↪parts) override;
};
```

The next part of the utility C file is the constructor definition. This is where you define when in the execution sequence it should be run. Common choices for when the utility should be run are:

- INITIAL_WORK - Run only once during initialization (before the first time step)
- PRE_TSTEP - Run at the start of each time step before any Newton iteration
- PRE_ITER - Run before beginning nonlinear iteration at each time step
- PRE_SOLVE - Run before each nonlinear iteration
- POST_TSTEP - Run at the end of each time step

- POST_ITER - Run after finishing nonlinear iteration at each time step
- POST_SOLVE - Run after each nonlinear iteration

```
MyCustomUtil::MyCustomUtil(Region& r)
: UserUtility( name(), r, UserUtility::UtilExecution::PRE_SOLVE )
{}
```

The next components of the plugin C file are the definitions for the other functions in the utility

```
void MyCustomUtil::register_utility_variables(const_
↳stk::mesh::ConstPartVector& parts) override {
    afgoout << "Registering my custom fields\n";
    mesh().register_field("some_field", sierra::FieldType::REAL,
        stk::topology::NODE_RANK, sierra::TEMPORARY, 1, parts
    );
}

void MyCustomUtil::execute()
{
    afgoout << "Running my custom utility\n";

    auto my_output = mesh().get_node_field("some_field");
    auto temp = mesh().get_node_field("temperature");

    auto selector = mesh().active_not_aura_selector() &_
↳stk::mesh::selectField(my_flux);

    using F = stk::mesh::NgpField<double>;
    tftk::mesh::for_each_entity_run(
        "DoSomething",
        mesh(),
        selector,
        my_output,
        {temp},
        KOKKOS_LAMBDA(stk::mesh::FastMeshIndex mi, const F& out, const F& T) {
            out(mi,0) = 10 + 4*T(mi,0);
        },
        tftk::HostSpace()
    );
}
```

Finally, we need to register the utility. This is done with the following command:

```
extern "C" void register_usersubs() {
```

(continues on next page)

(continued from previous page)

```
sierra::Afgo::UtilityPluginFactory<MyCustomUtil> register_my_util{};  
}
```

The name of the variable here (register_my_util) does not matter.

Compiling a Utility

User utilities are compiled to shared object (.so) files using the same procedure as *User Subroutines*. The following command will scan the input file for import .so files and compile them from source files of the same name.

```
sierra --make fuego -i input.i
```

Using a Utility

In the Fuego input file, the plugin is loaded using the command (assuming the plugin is defined in My_Utility_Plugin.C)

```
Begin Sierra My_Sierra_Job  
  Load User Plugin file ./My_Utility_Plugin.so USING FUNCTION register_  
  ↪usersubs
```

No changes are needed in the input file to use a plugin utility, it is automatically added to the requested spot and executed.

4.15 Output Reference

Output is divided into two major categories:

- *Results Output* contains binary format simulation results in a form suitable for visualization. While the Results output usually corresponds to the entire model, it can also be applied to portions of the model.
- *Heartbeat Output* is generally written to a text file and provides a convenient means of monitoring intermediate simulation results as well as outputting *post-processed variables*.

This section describes how to use these two output functionalities. A detailed list of all possible commands can be found in the *command reference*.

Results Output

The results output block allows you to print simulation data to an output database. This database can then be visualized/postprocessed (e.g. in [ParaView](#)), used in subsequent simulations, etc. This data includes internal fields such as the DOF solved in each active equation, as well as *post-processed nodal and global fields*.

A list of the quantities available for output is printed to the log immediately after the mesh is read. This list can be previewed by running fuego with the `--check-input` option which will verify the input syntax, read the mesh, print the variables available for output, and then exit.

Fuego Region "fuego_region" has the following fields available for output:

GLOBAL Variables:

- * AREA_X_surface_1
- * AREA_Y_surface_1
- * AREA_Z_surface_1
- * mass_flow_rate_surface_1
- * TOTAL_AREA_surface_1

NODE_RANK Fields:

- * continuity_residual
- * control_volume
- * density
- * density_derivative
- * dpdx
- * dpdy
- * dpdz
- * gas_volume_fraction
- * grad_q
- * mass_diffusivity
- * mass_fraction
- * model_coordinates
- * molecular_weight
- * pressure
- * pressure_gradient
- * turbulent_dissipation
- * turbulent_kinetic_energy
- * turbulent_production_ke
- * turbulent_viscosity
- * u_star
- * u_tmp
- * v_tmp
- * velocity
- * viscosity
- * w_tmp

(continues on next page)

(continued from previous page)

```
* x_velocity
* y_velocity
* ysolve
* z_velocity
FACE_RANK Fields:
* density_bip
* eff_wall_yp_bnd
* mass_flux_bip
* pressure_bip
* viscous_force_bip
* wall_area_bip
* wall_temperature_bip
* wall_yp_bip
ELEMENT_RANK Fields:
* CFL
* mass_flux
* num_wall_faces_on_scv
* subcontrol_volume
```

The available quantities are split by their entity rank. When specifying a field for output, the associated rank should be used. A quantity is added to the output database using a line of the form

```
<entity> variables = <internal name> [as <output name>]
```

where the entity is the rank specified in the output list, and the `internal name` can optionally be overridden in the output database by specifying an `output name` e.g.

```
# Scope: Sierra > Procedure > Fuego Region
begin results output myOutput
  # name of the output database
  Database name = results.e

  # Frequency information
  at step 0 increment = 1

  #Quantities to print
  nodal variables = someMissingField
  nodal variables = density as rho
  element variables = volume_change_ratio
  global variables = avg_temperature
end
```



Note: It is worth noting that since the ExodusII format has no concept of vector/tensor quantities, [Paraview](#) deduces vectors by looking for the same variable with X, Y, and Z (for 3D problems) appended. For example if Ux, Uy, and Uz are all present in an output database, ParaView will group them into a vector called U. Similarly if a field called displacement is found, it will be recognized by ParaView as the nodal displacements and the mesh will automatically be warped by the field. Fuego will handle naming of vector/tensor quantities that are requested for output, but it is worth keeping these facts in mind when specifying your own output names.

Note: It is not fatal to request an output quantity that does not exist. Instead, a warning is printed to the log stating that output is skipped since the field cannot be found. If a certain quantity is missing from your output, this can be a good first thing to check for.



```
*****
**  Warning:
**  In region 'myRegion' Results Output block 'myOutput',
    ↳Nodal variable
**  'someMissingField' was not found and will not be output
*****
```

Heartbeat Output

Unlike [Results Output](#), heartbeats are used to view scalar data such as reduced quantities and running metrics. They can be useful to track progress of simulations, as well as for collecting reduced data for postprocessing.

Available output quantities include:

- Mass flow rate on inflow and open boundaries
- Force, heat flux, and yplus on walls
- Any *postprocessed scalar*

The complete list of commands available in the heartbeat block can be found in the [command summary](#).

As was stated above, the list of the available globals will be printed to the log

```
Fuego Region "myRegion" has the following fields available for output:
GLOBAL Variables:
* AREA_X_surface_1
* AREA_Y_surface_1
```

(continues on next page)

(continued from previous page)

```
* AREA_Z_surface_1
* mass_flow_rate_surface_1
* TOTAL_AREA_surface_1
NODE_RANK Fields:
* ...
EDGE_RANK Fields:
* ...
ELEMENT_RANK Fields:
* ...
```

A quantity is added to the output heartbeat using a line of the form

```
variable = <entity> <internal name> [as <output name>]
```

Note that similar to results output, the `internal name` can be overridden by specifying as `<output name>`

```
# Scope: Sierra > Procedure > Fuego Region
Begin Heartbeat myHb
  # name of the output stream
  Stream name = results.csv

  #Formatting of file
  Timestamp Format = ""
  Precision       = 8   # precision of output
  Format          = csv # delimiter
  Labels          = Off # qnty=123 format
  Legend          = On  # header row of labels

  # Frequency information
  at step 0 increment = 1

  #Quantities to print
  variable = global mass_flow_rate_surface_1 as mdot1
  variable = global TOTAL_AREA_SURFACE_1
  ...
End
```

It is also possible to select a single value from a field by specifying the closest point

```
variable = <entity> <internal name> nearest location <X> <Y> <Z> as
↪<output name>
```

or a specified ID

```
variable = <entity> <internal name> at <entity> <ID> as <output name>
```

For example:

```
# Scope: Sierra > Procedure > Fuego Region
Begin Heartbeat myHb
...
Variable = Node density nearest location 0.0 0.0 0.0 as rhoOrigin
Variable = Node density at node 123 as rho123
variable = element CFL at element 1 as cfl1
...
End
```

Note: Similar to results output, it is not fatal to request heartbeat of a global that does not exist. Instead, a warning is printed



```
*****
** Warning:
** In Heartbeat Output block 'THE_HEARTBEAT':
** Global variable 'someMissingGlobal' does not exist and
  ↳ will not be output
*****
```

The format of the heartbeat output is flexible, and is controlled by the parameters *Precision*, *Format*, *Labels*, and *Legend*.

If no specific format is required, the following parameters

```
# Scope: Sierra > Procedure > Fuego Region
Begin Heartbeat myHb
...
Timestamp Format = ""
Precision        = 8   # precision of output
Format          = csv  # delimiter
Labels          = Off  # qty=123 format
Legend          = On   # header row of labels
End
```

produce a csv file that is human-readable and can be read easily across applications

```
time, avg_temperature, max_temperature
0.00000000e+00, 3.50000000e+02, 4.00000000e+02
1.00000000e-01, 3.60000000e+02, 4.10000000e+02
```

(continues on next page)

(continued from previous page)

```
2.000000000e-01,  3.700000000e+02,  4.200000000e+02
...
```

Output Scheduling

For both results and heartbeat output, the frequency of output is controlled by specifying either the time or step frequency using the following lines respectively

```
# Scope: Sierra > Procedure > Fuego Region
Begin Heartbeat myHb
  at time 0.0 increment = 0.1
  # or
  at step 0 increment = 1
End
```

Note that multiple of these lines can be specified to define a more complex output schedule e.g.

```
# Scope: Sierra > Procedure > Fuego Region
Begin Heartbeat myHb # or results output myOutput
  at time 0.0      increment = 100.0
  at time 300.0    increment = 300.0
  at time 1800.0   increment = 500.0
End
```

Additional times/steps can also be included using by specifying the respective command lines

```
# Scope: Sierra > Procedure > Fuego Region
Begin Heartbeat myHb # or results output myOutput
  Additional Steps = 1 2 3
  # or
  Additional Times = 0.1 0.2 0.3
End
```

By default, the time closest to the requested print interval is written. It is sometimes desired, however, to output exactly at a given print interval. In this case, `Timestep adjustment interval = <N>` can be set, which allows the time step manager to modify the next N steps to attempt to exactly hit the desired output.

```
# Scope: Sierra > Procedure > Fuego Region
Begin Heartbeat myHb # or results output myOutput
  ...
  Timestep adjustment interval = 4
End
```

Finally, note that if a certain output pattern is desired to be used in multiple places, it can be defined as an *output scheduler*

and given as input to the output block(s) in place of the arguments above. Note that the scheduler must be placed **at Sierra scope**:

```
Begin Sierra Job
# Scope: Sierra!
begin Output Scheduler mySched
  at time 0.0    increment = 100.0
  at time 300.0  increment = 300.0
  at time 1800.0 increment = 500.0
  ...
end

...
Begin Procedure theProcedure
  ...

  Begin Fuego Region Fuego_Region
    ...

    # Scope: Sierra > Procedure > Fuego Region
    Begin Heartbeat myHb
      Use Output Scheduler mySched
      ...
    End

    Begin results output myOutput
      Use Output Scheduler mySched
      ...
    End
  End

End Procedure theProcedure
End Sierra Job
```



Note: In the case of steady solutions, Fuego output will contain two “time” steps: an initialization output plane, and the steady solution output. Care must be taken to use the appropriate time if these results are used in subsequent analysis (e.g. to provide the initial condition from file for a transient problem).

4.16 Restart Reference

Restart is used in analyses that may not complete in a single job submission and might complete when continued from a previous termination time. Fuego maintains the ability to restart a previous analysis by recording the internal state variables of a problem in a restart database. In this case the input mesh is supplied from a *Finite Element Model* block and the restart information is obtained from the restart data. See the *command reference* for a detailed description of the lines possible when defining this block.



Note: Often one wishes to initialize an analysis with results from another analysis (e.g. a steady simulation fed into a transient IC). In these cases, a full restart is not necessary. Instead *Input Output Region* can be used to initialize specific fields of interest using the *results* of the other simulation.

If one believes that a simulation might run too long, and wishes to record restart databases, they need only to add a *restart data* block as follows:

```
# Scope: Sierra > Procedure > Fuego Region
Begin restart data data
  at step 0, increment = 1
  database name = restart.e
End
```

The restart database uses the same *output scheduling logic* as the results/heartbeat output, and can similarly be given an *output scheduler* block in place of specifying the commands in the restart block.

To restart a simulation, one must specify either the *restart time* to start at, or enable *automatic restart* (latest available time, start of simulation if not found) **at Sierra scope**. Fuego will read the given database name for the required restart information

```
Begin Sierra Job
  # Scope: Sierra!
  RESTART TIME = 0.00125
  # or: RESTART = auto

  ...
  Begin Procedure theProcedure
    ...

    Begin Fuego Region Fuego_region
      ...

      # Scope: Sierra > Procedure > Fuego Region
```

(continues on next page)

```
Begin restart data data
```

```
...
```

```
# Pointing to a previously created restart database
```

```
database name = restart.e
```

```
End
```

```
End Fuego Region
```

```
End Procedure theProcedure
```

```
End Sierra Job
```

Warning: Specifying an **input database name** that does not exist would skip restart in previous versions, but this behavior is deprecated and moving forward will be treated as an error. To recover this behavior, one should specify only a **database name** line which will be used as both the input and output database. Alternatively, one can use an Aprepro switch to switch between reading and writing a restart database e.g.

```
# Scope: Sierra!
# is_restart = {is_restart = 0} # default to off
{if(is_restart)}
RESTART = auto
{endif}
```



```
# Scope: Sierra > Procedure > Fuego Region
```

```
Begin restart data data
```

```
...
```

```
{if(is_restart)}
```

```
input database name = restart.e
```

```
{else}
```

```
output database name = restart.e
```

```
{endif}
```

```
End
```

The same input can then be used by adding an Aprepro definition to the restart launch line

```
#First run
```

```
launch -n 8 fuego -i fuego.i
```

```
#Restart run
```

```
launch -n 8 fuego -i fuego.i --define "is_restart=1"
```

To save space, options are provided to minimize the retained restart data. These options include

- *Cycle Count* - Cycle timesteps within a single DB file
- *File Cycle Count* - Cycle timesteps within individual DB files
- *Overlay Count* -
Overwrite intermediate time steps within a single DB file

4.17 IO Region

The Input_Output Region (IO Region) lets you use data from a previously-generated Exodus dataset (e.g. output from some other simulation code) in your simulation (as a initial condition, boundary condition, etc. . .).

It uses *transfer* and *solution control* to map the variables from the mesh they are saved on to the current problem's mesh.



Note: Just like other Sierra Regions the Input_Output Region must have its own Finite Element model command block defined. A full description of the syntax can be found in the *command references*

As an example, let us assume that an input mesh for an Input_Output Region contains a nodal variable ConvCoeff that we wish to use in another Region. In this case an outline for one-way transfer of ConvCoeff to a Region, *second_region*, in a steady-state problem would be:

```
Begin Sierra
  Begin Finite Element Model input_transfer
    ...
  End

  Begin Transfer my_first_transfer
    transfer commands for input_output_region to second_region
    SEND field hNd state none TO ConvCoeff state none
    ...
  End

  Begin Procedure My_Procedure
    Begin Solution Control Description
      Use System Main
    Begin System Main
      Begin Sequential MySolveBlock
        Advance io_region
        transfer my_first_transfer
```

(continues on next page)

```

        Advance second_Region
      End
    End
  End

  Begin Input_Output io_region
    USE FINITE ELEMENT MODEL my_input_transfer
  End

  Begin Fuego Region second_region
    use Finite Element Model input_transfer
    USER FIELD REAL NODE SCALAR ConvCoeff on surface_1
    ...
  End
  ...
End
...
End Sierra

```

Controlling Time

The Input_Output Region can also be used with transient data. When the IO region contains transient data, there are several options for controlling which time value to transfer to the solution region. The IO region will interpolate linearly between its time steps when sending data to the solution region.

- **Fixed Time:** If you specify a fixed time in the IO region block, all transfers will use that time from the IO region data set. If the time you specify is larger than the last time in the IO region data set, the last time step will be used.
- **Offset Time:** This is a constant offset added to the simulation time. For example, if an offset of 10 is provided, at simulation time 0 the IO region will send its data at 10 seconds. The offset is added to the simulation time regardless of whether periodicity is active or not.
- **Start Time:** The start time should be provided along with `periodicity time`. If the start time is given as 25 and the periodicity as 10 then the simulation will use the time values between 25 and 35 from the IO region repeatedly once the simulation reaches 25. The default start time is 0.
- **Periodicity Time:** This controls how long of a time period to repeat. The default periodicity is 0 (not enabled).

The syntax for these commands are described in the [command reference section](#).

When `fixed time` is not specified, the IO Region time is selected using the following logic:

```

if simulation_time > start_time and periodicity > 0
    io_region_time = start_time + (simulation_time - start_time) %
    ↪periodicity + offset
else
    io_region_time = simulation_time + offset

```

When fixed time was specified, `io_region_time = fixed_time`

4.18 Particles

Fuego supports inclusion of discrete Lagrangian particles using a particle-in-cell method to transport them. The particles can be zero-mass tracer particles, inertial particles, heated particles, evaporating particles, or combusting particles. For details of the different particle models, refer to the *theory manual*.

Particles are defined in a separate `Particle Region` and coupled to the fluid through *transfers*. The particle region has a background Eulerian mesh, usually the same mesh as the fluid domain, and the particles are modeled using free nodes that move around in that domain. To avoid excessive mesh modification, there is a “particle reservoir” of nodes created on each MPI rank, and as particles are added to the domain they are drawn from the reservoir and “activated”. The particles are advanced in time using sub-steps so their time step is smaller than the overall fluid time step.

```

begin particle region particle_region
    use finite element model femodel
    lagrangian particles
    fluid region name = fluid_region

    minimum particle subcycles = 5
    particle reservoir size = 5000

    particle rebalance step frequency = 6
    particle rebalance imbalance threshold = 1.0

    # ...

end particle region particle_region

```

For the complete list of commands in the particle region, refer to the *command summary*.

Particle Setup

For most particle types, you need to specify three things:

1. **Particle material** - Set material properties for the particles. Required properties depend on what physical particle model is in-use.
2. **Particle definition** - Define what particle material to use, and which physical model to apply.
 - *Tracer* - Mass-less particles, move with the fluid velocity
 - *Inertial* - Has mass, drag, and buoyant forces which control its motion
 - *Heated Inertial* - The same as inertial, but adds an energy transfer between the particle and fluid.
 - *Evaporating* - Same as the heated inertial particle but adds mass transfer via phase change. Requires a **particle interface** to define the evaporation model.
 - *Chemically Reacting, Wildfire* - More advanced models involving heat and mass transfer between the particles and fluid.
3. **Particle insertion** - Some method for how the particles are inserted. Common options are:
 - *Spray* - Set a location, direction, cone angle, and size statistics and uses a pseudo-random algorithm to insert particles.
 - *File* - Use a file of tabulated insertion locations, times, and properties to add particles to the domain.
 - *Location* - Insert particles at the simulation initiation along a line or in a filled shape
 - *Sideset* - Insert particles at a given rate on a sideset. Particles are inserted at face centroids, so behavior may be mesh-dependent.

The insertion methods and definition (physical models) are independent. In the following examples, we show different particle definition setups and different insertion methods, but these can be mixed and matched arbitrarily.

Tracer Particle Example

Tracer (or “Tracker”) particles are the simplest particles to define. They require no properties or model parameters. In the example below, we define the `tracer_particles` definition, and then insert them along `surface_1` using the particle inflow boundary. They can leave through the open boundary on `surface_2`, and will rebound elastically off the wall at `surface_3`.

```
begin particle region particle_region

# ...

begin particle definition tracer_particles
  particle type is tracker
end particle definition tracer_particles

begin particle inflow boundary condition on surface surface_1
  particle definition = solid_particles
end

begin particle open boundary condition on surface surface_2
end

begin particle wall boundary condition on surface surface_3
  particle surface interaction type = rebound
end

end particle region particle_region
```

Inertial Particle Example

Inertial particles have mass, so they require a density and diameter. Since they couple with the fluid momentum equation, they also need the `compute momentum source` command. The following example defines steel inertial particles which will be inserted with the spray model.

```
begin particle region particle_region

# ...
compute momentum source

begin particle material steel
  density = 7800
end particle material steel
```

(continues on next page)

```

begin particle definition metal_particles
  particle type is inertial_particle
  add particle material steel
end particle definition metal_particles

begin particle spray my_spray
  # ...
end particle spray my_spray
end particle region particle_region

```

Spray Model

The spray model is one of the most commonly used particle insertion methods. In the example below, we specify a conical spray with distributions for the particle diameter and velocity. We also specify the mass flow rate (a string function of time) and the number represented.



Note: A spray can generate a large number of particles. To reduce computational costs, we can choose to define a **number represented**, so that each resolved particle represents more than one physical particle. The drag, mass, and other coupling terms between the single resolved particle and the fluid are then multiplied by the number represented.

```

begin particle region particle_region
  # ...

BEGIN PARTICLE SPRAY my_spray
  PARTICLE DEFINITION = metal_particles
  CENTER = 0.0, 0.0, 0.0
  SPRAY_ANGLE = 30 # degrees
  NOZZLE RADIUS = 0.0003
  NORMAL VECTOR = 0 -1 0

  MASS_FLOW_RATE = 0.1*(t>1)*(t<5)
  NUMBER REPRESENTED = 10

  # Step 1: define the distribution type (LOGNORMAL)
  # Step 2: provide required parameters for that distribution
  #         (LOGNORMAL requires LOGMEAN and LOGSTDEV)
  #         resulting size = exp(logMean - sqrt(2) * logStDev * inv_
  ↪erfc(2*R))

```

(continues on next page)

(continued from previous page)

```
#           where R is a uniform random number
PARTICLE DIAMETER DISTRIBUTION TYPE = LOGNORMAL
PARTICLE DIAMETER DISTRIBUTION PARAMETER LOGMEAN = -8.0 #  $\exp(-8) = 3.35e-4$ 
PARTICLE DIAMETER DISTRIBUTION PARAMETER LOGSTDEV = 1.0

# Step 1: define the distribution type (CONSTANT)
# Step 2: provide required parameters for that distribution (CONSTANT_
requires VALUE)
PARTICLE VELOCITY DISTRIBUTION TYPE = CONSTANT
PARTICLE VELOCITY DISTRIBUTION PARAMETER VALUE = 0.00001

END PARTICLE SPRAY my_spray
end particle region particle_region
```

Heated Particle Example

Heated particles require additional coupling source terms (energy, and sometimes radiation) and additional material properties. In the example below, we extend the steel particles to include heat transfer with the fluid, and use an insertion along a line.

```
begin particle region particle_region

  compute momentum source
  compute energy source
  compute RTE particle source

  begin particle material steel
    density = 7800
    specific_heat = 500
    thermal_conductivity = 10
    film_Prandtl_number = 5
    absorptivity = 1.0
  end particle material steel

  begin particle definition metal_particles
    particle type is heated_particle
    add particle material steel
  end particle definition metal_particles

  begin insert particle my_line
```

(continues on next page)

(continued from previous page)

```
# ...  
end insert particle my_line  
end particle region particle_region
```

Heated particles require an initial temperature in their insertion method. In the input below, we specify two points and Number so we insert 5 particles along the line connecting those points.

```
begin particle region particle_region  
  
# ...  
begin insert particle my_line  
  particle definition = metal_particles  
  x1 = 0.0  
  y1 = 0.0  
  z1 = 0.0  
  x2 = 0.0  
  y2 = 0.0  
  z2 = 1.0  
  
  x_velocity = 0.0  
  x_velocity = 0.0  
  x_velocity = 0.0  
  Temperature = 400  
  Diameter = 1e-3  
  Number = 5  
  Number represented = 1  
end insert particle my_line  
end particle region particle_region
```



Note: If we omit point 2 in the particle insertion block, the particles are all inserted at point 1.

Evaporating Particles

Evaporating particles require an additional `particle interface` block to define the evaporation model. In the example below we set up the blocks for a spray of evaporating water drops.

```
begin particle region particle_region  
  
# ...
```

(continues on next page)

(continued from previous page)

```
compute momentum source
compute energy source
compute species source
compute continuity source
compute RTE particle source

begin particle material water
  density = 1000
  specific_heat = 1000
  absorptivity = 0.8
end particle material water

begin particle definition water_particles
  particle type is evaporating_particle
  add particle interface evapWater
  add particle material water
end particle definition water_particles

begin particle interface evapWater
  # ...
end particle interface evapWater

begin particle spray my_spray
  # ...
end particle spray my_spray
end particle region particle_region
```

For the particle interface, we need to define what species is evaporating (so we can apply the source to the right equation on the fluid side) and phase change parameters.

```
begin particle region particle_region

  # ...

  BEGIN PARTICLE INTERFACE evapWater
    PARTICLE SPECIES H2O -1.0
    PRANDTL_NUMBER = 0.9
    SCHMIDT_NUMBER = 0.9
    BEGIN PARTICLE EVAPORATION waterEvaporation
      REFERENCE_HEAT_OF_VAPORIZATION = 2.26e6 # J/kg
      REFERENCE_TEMPERATURE = 373.0 # K
      CRITICAL_TEMPERATURE = 647.0 # K
      REFERENCE_PRESSURE = 1.0 # in atm
```

(continues on next page)

```

    GAS SPECIES H2O 1.0
  END    PARTICLE EVAPORATION waterEvaporation
END    PARTICLE INTERFACE evapWater

# ...

end particle region particle_region

```

Next we can define a spray similar to the earlier example, except now we also must supply a temperature. We also changed the spray to a hollow cone (common sprinkler pattern) by supplying SPRAY_ANGLE_START and SPRAY_ANGLE_END instead of just SPRAY_ANGLE`.

```

begin particle region particle_region

# ...
BEGIN PARTICLE SPRAY my_spray
  PARTICLE DEFINITION = water_particles
  CENTER = 0.0, 0.0, 0.0
  SPRAY_ANGLE_START = 30 # degrees
  SPRAY_ANGLE_END = 45
  NOZZLE RADIUS = 0.01
  NORMAL VECTOR = 0 -1 0

  MASS_FLOW_RATE = 0.1*(t>1)*(t<5)
  NUMBER REPRESENTED = 10
  TEMPERATURE = 300

  # Step 1: define the distribution type (LOGNORMAL)
  # Step 2: provide required parameters for that distribution
  #         (LOGNORMAL requires LOGMEAN and LOGSTDEV)
  #         resulting size = exp(logMean - sqrt(2) * logStDev * inv_
→erfc(2*R))
  #         where R is a uniform random number
  PARTICLE DIAMETER DISTRIBUTION TYPE = LOGNORMAL
  PARTICLE DIAMETER DISTRIBUTION PARAMETER LOGMEAN = -8.0 # exp(-8) = 3.
→35e-4
  PARTICLE DIAMETER DISTRIBUTION PARAMETER LOGSTDEV = 1.0

  # Step 1: define the distribution type (CONSTANT)
  # Step 2: provide required parameters for that distribution (CONSTANT_
→requires VALUE)
  PARTICLE VELOCITY DISTRIBUTION TYPE = CONSTANT
  PARTICLE VELOCITY DISTRIBUTION PARAMETER VALUE = 0.00001

```

(continues on next page)

```

END PARTICLE SPRAY my_spray
end particle region particle_region

```

Modifications to the Fuego Region

The Fuego region needs to be set up to receive *external sources* from the particles. This is done using the Use external X source commands:

```

begin solution options
  use external momentum source
  use external enthalpy source
  use external continuity source
  use external species source

  # ...
End

```

Particle Output

The syntax for outputting particle data is the same as for fluid data. Some generally useful output rules are:

- Anything with a _p suffix is a particle field (e.g. particle velocity or particle temperature)
- Always output coordinates so you can show particles in the correct location
- It is useful to output active so you can filter out inactive reservoir particles
- Particle properties such as radius_p or temperature_p can often be relevant

```

begin particle region particle_region

  # ...

Begin Results Output Label output
  DATABASE Name = particles.e
  At Step 0, Increment = 1
  TITLE Lagrangian Particles
  NODAL Variables = coordinates AS Coords
  NODAL Variables = x_velocity_p AS Ux
  NODAL Variables = y_velocity_p AS Uy

```

(continues on next page)

```

NODAL Variables = z_velocity_p AS Uz
NODAL Variables = radius_p AS R
NODAL Variables = temperature_p AS T
NODAL Variables = number_deposition_density
NODAL Variables = number_deposition_rate
NODAL Variables = mass_deposition_density
NODAL Variables = mass_deposition_rate
NODAL Variables = active
NODAL Variables = particle_is_stuck
End    Results Output Label output

end particle region particle_region

```

Advanced Options

Advanced particle options not covered in this guide include:

- Particle filled shapes
- Wildfire particles

5 Tutorials & Examples

The present section offers references for setting up your own case(s). First the following list is a set of tutorials (many coming from the Sierra100 courses). These tutorials offer step-by-step directions for setting up a simple case

5.1 Sierra 100

Laminar Flow Over A Cylinder

This tutorial shows how to set up a flow over cylinder in Fuego. This module has also been recorded in prior trainings and is available [here](#) or in the player below.

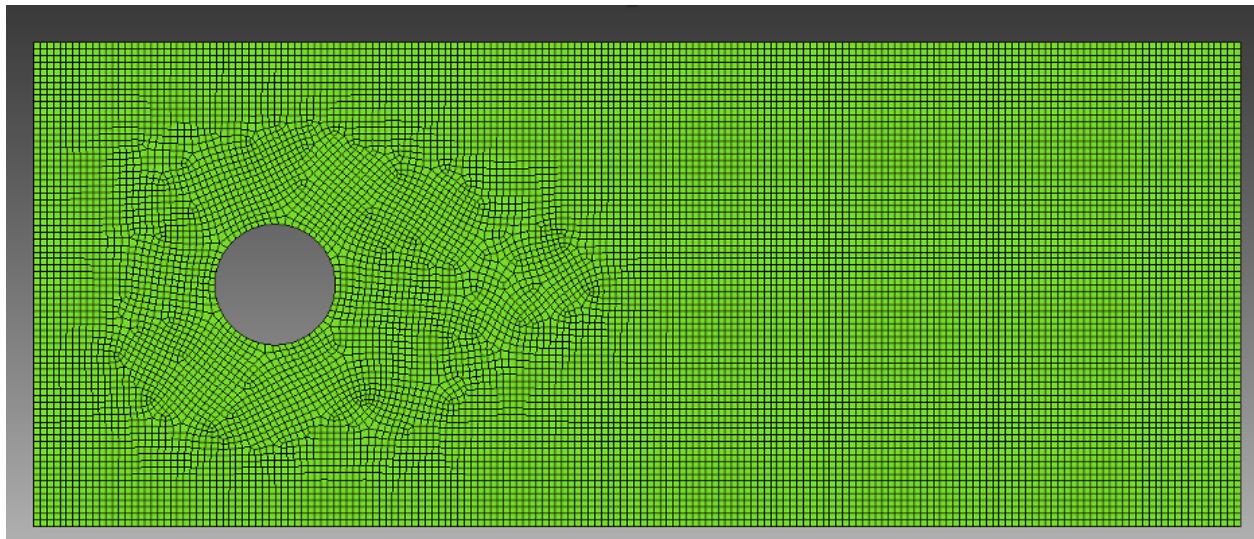
Problem Files

The files required for this tutorial can be downloaded [here](#), or found in a CEE environment at `/projects/sierra100/TF/Fuego`.

- Fuego input file
- Mesh file
- Training slides

Problem Domain/Mesh

In this tutorial we will be simulating a 2D laminar flow over a cylinder cross-section, where the mesh is shown below



The mesh file needed for this tutorial can be downloaded [here](#), or it can be generated manually using the Cubit journal file below.

Cubit Journal File

```
undo on
# { D = 0.1 }

brick x {10*D} y {4*D} z 1
create cylinder height 1 radius {D/2}
move volume 2 x {-3*D}
subtract volume 2 from volume 1

block 1 add surface 11
block all element type QUAD4
```

(continues on next page)

(continued from previous page)

```
surface 11 size {0.054*D}
mesh surface 11

sideset 1 add curve 3
sideset 2 add curve 1
sideset 3 add curve 2 4
sideset 4 add curve 16
set large exodus file on
export genesis "cyl_base.g" dimension 2 block all overwrite
```

Fuego Input File

Every Fuego simulation requires a text input file to define the problem. The following sections show how to set up that file to run a the laminar flow problem using the geometry shown above.

Indentation is recommended but not required in an Fuego input file. You can indent with tabs or spaces, or choose not to indent things at all - although we highly recommend using indentation (with spaces) to make a consistently readable input file and to avoid errors with commands going in the wrong scope.

Anything that starts with a # or a \$ will be treated as a comment and ignored. Long lines can be split using \\$, like this

```
# This is a comment
$ and so is this

# This is a single line command split on three lines
List of Blocks = block_1 block_2 block_3 \$
                  block_4 block_5 block_6 \$
                  block_7 block_8
```

Overall File Structure

The Fuego input file is structured using nested blocks with **Begin** and **End** commands surrounding each block. The outermost block must always be the **SIERRA** block. Commands directly in this block are referred to as “Domain-level” commands. Other commands must go inside inner blocks, such as “Procedure” or “Fuego Region”, and will be referred to as such. If you put a command in the wrong level, you will get an error when you try to run Fuego.

```
Begin SIERRA Fuego
  # Domain-level commands
```

(continues on next page)

```

Begin Procedure FuegoProcedure
  # Procedure-level commands

  Begin Solution Control Description
    # Solution control commands
  End

  Begin Fuego Region myRegion
    # Region-level commands
  End
End
End

```

For more information on the Fuego input file, please see *Anatomy of an Input Deck*.

Defining Materials

Scope: Domain

You must define a material to use for each block in your mesh. A material can be used on multiple blocks, but every block must have a material. The properties required for your material will depend on what equations you are solving.

Generally in Fuego only one material is usually defined. In this particular example for a laminar flow over a cylinder, we need to specify both a density and (dynamic) viscosity for the material

```

Begin Property Specification for Fuego Material air
  Density    = 1.2 #kg/m3
  Viscosity  = 1.8e-5 #Pa s
End

```

In this material definition, we have defined both the density and viscosity as constants.

In general, these properties will be in the units of your problem setup, which we will define in *Fuego Region Parameters*. We will use MKS (meters, kilograms, and seconds) as our length, mass, and time scales, respectively, for this problem.

Define The Finite Element Model

Scope: Domain

The finite element model defines the mesh file that is used in the simulation, the database type (usually ExodusII), and the parallel decomposition method (RIB or RCB). Additionally, the finite element model is used to associate mesh blocks with the materials we defined in the previous *Defining Materials*

```
Begin Finite Element Model cylMesh
  Database Name = cyl_base.g
  Database Type = ExodusII
  Decomposition Method = RIB

  # ASSIGN MATERIALS TO BLOCKS
  Begin Parameters for Block block_1
    Material air
  End   Parameters for Block block_1

End   Finite Element Model cylMesh
```

Typically in Fuego you do not need a large amount of blocks, except in a few cases:

- Elements of different topologies (hex, tet, wedge, pyramid) must go in separate blocks
- You may want different initial conditions in some places based on block specifications

Time Stepping Control

Scope: Procedure

Next we must set up some of the rules and parameters for how we want to set up time stepping for this transient problem.

```
Begin Fuego Procedure Fuego_procedure

  Time Start = 0.0, Stop = 10.0, Status Interval = 10

  begin time control
    BEGIN TIME STEPPING BLOCK  time_block
      START TIME IS  0.0
      TIME STEP = 1e-3
    BEGIN PARAMETERS FOR FUEGO REGION fuego_region
      TRANSIENT STEP TYPE IS automatic
      CFL LIMIT = 5.0
```

(continues on next page)

(continued from previous page)

```
TIME STEP CHANGE FACTOR = 1.5
END   PARAMETERS FOR FUEGO REGION fuego_region
END   TIME STEPPING BLOCK time_block

Termination time = 1e6
end time control
```

Typically in this block we set the start and end times. The status interval is used to print more diagnostics at the specified interval (every 10 steps in this case). In the time stepping control we specify the initial timestep $\Delta t = 1 \times 10^{-3}$, and to also automatically adapt the time step based on the global CFL limit defined as

$$CFL = \frac{u\Delta t}{\Delta x}$$

where u is the element velocity, Δt is the timestep, and Δx is an element length scale. This CFL condition is computed inside each element and the global timestep is chosen based on the global maximum computed CFL on all of the elements in your simulation.

Most Fuego problems are transient, but steady problems can be solved using a pseudo-transient approach.

Fuego Region Parameters

Scope: Region

Here we will set up some parameters in the Fuego region

```
Begin Fuego Region Fuego_region

# SET THE UNITS
Set unit system to MKS

# DECLARE WHICH MESH/FINITE ELEMENT MODEL TO USE
Use Finite Element Model cylMesh
```

Here we tell the region which finite element model to use (the one defined in *Define The Finite Element Model*) and unit system to use MKS (default to CGS if not specified).

One can define their own custom length/mass/time scales using the following commands in the Fuego region scope:

```
# Does the same thing as "SET UNIT SYSTEM TO MKS"
SET LENGTH UNIT CONVERSION FACTOR = 100.0
SET MASS UNIT CONVERSION FACTOR = 1000.0
```

(continues on next page)

(continued from previous page)

```
SET TIME    UNIT CONVERSION FACTOR = 1.0
SET TEMPERATURE UNITS = KELVIN
```

```
# use a non-standard mm-g-s unit system
SET LENGTH UNIT CONVERSION FACTOR = 0.1
SET MASS    UNIT CONVERSION FACTOR = 1.0
SET TIME    UNIT CONVERSION FACTOR = 1.0
```

For more information on setting length/mass/time scales see *Units*.

Solution Options

Scope: Region

The solution options block defines what equations to solve, and sets various numerical algorithm choices

```
Begin Solution Options
  Coordinate System = 2D
  Projection Method = Fourth_Order Smoothing with Timestep Scaling

  # DEFINE WHAT EQUATIONS TO SOLVE
  Activate Equation Continuity
  Activate Equation X_Momentum
  Activate Equation Y_Momentum

  # DEFINE NONLINEAR ITERATION SETTINGS
  Minimum Number of Nonlinear Iterations = 1
  Maximum Number of Nonlinear Iterations = 2

  # DEFINE ADVECTION SCHEME
  Upwind Method is MUSCL
  UPWIND LIMITER IS SuperBee

End    Solution Options
```

We will go into detail on a few sub-blocks of this input file snippet:

```
Projection Method = Fourth_Order Smoothing with Timestep Scaling
```

The common choice here for the pressure projection is the fourth order smoothing with timestep scaling. There are more advanced options shown in *Smoothing Method*, but for most problems this will suffice.

```
# DEFINE WHAT EQUATIONS TO SOLVE
Activate Equation Continuity
Activate Equation X_Momentum
Activate Equation Y_Momentum
```

This defines which equations you want Fuego to solve. In this case, since we are running a two-dimensional problem, we want to activate the continuity equation, as well as the x (streamwise) and y (wall normal) momentum equations (see [Equations To Solve](#) for more on what equations you can activate). For more detail on the continuity and momentum equations Fuego solves, please see [Low Mach Number Equations](#).

```
# DEFINE NONLINEAR ITERATION SETTINGS
Minimum Number of Nonlinear Iterations = 1
Maximum Number of Nonlinear Iterations = 2
```

This sets the minimum and maximum number of nonlinear iterations for Fuego to take when solving the above equations. The default nonlinear residual tolerance is 1×10^{-15} , which is small enough where Fuego will always take the maximum number of nonlinear iterations. For more advanced control, you can set a target nonlinear tolerance per equation (see [Nonlinear Iteration Settings](#)).

```
# DEFINE ADVECTION SCHEME
Upwind Method is MUSCL
UPWIND LIMITER IS SuperBee
```

We must choose which algorithm to use to handle the advection term (“Upwind Method”) in the transport equations. Two common upwinding methods are UPW and MUSCL. UPW is a fully upwinded advective stabilization, which is more dissipative but is low order (1st order). MUSCL is a higher order and less dissipative advection scheme and is 2nd order accurate, but general needs an upwind limiter. For a more advanced discussion on the topic, see [Advection Methods](#).

Boundary Conditions

Scope: Region

Boundary conditions are set in the following input file commands:

```
Begin Inflow Boundary Condition on Surface surface_1
  X_Velocity = (2000*1.8e-5)/(1.2*0.1)
  Y_Velocity = 0.0
End

Begin Open Boundary Condition on Surface surface_2
  Pressure = 0.0
```

(continues on next page)

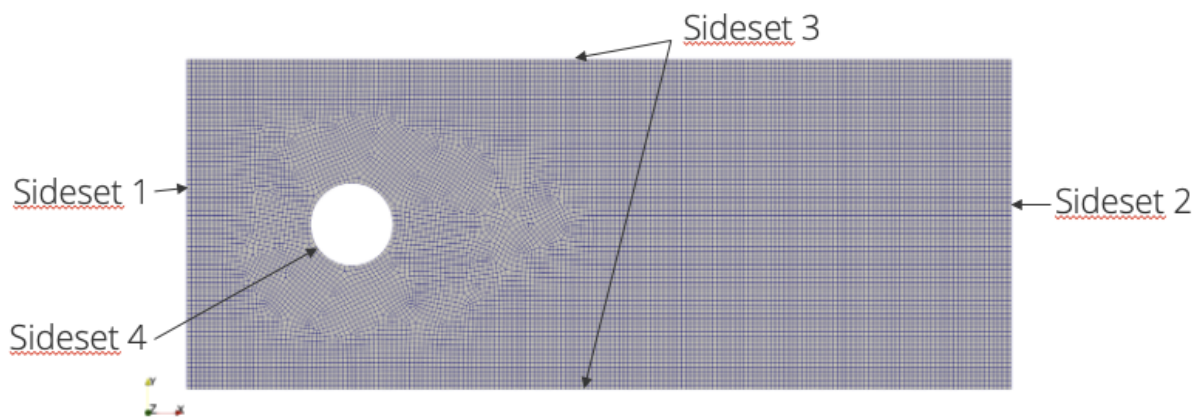
(continued from previous page)

```
End

Begin Symmetry Boundary Condition on Surface surface_3
End

Begin Wall Boundary Condition on Surface surface_4
    post process yplus
End
```

For reference, the mesh with labeled surfaces is shown below



In this mesh we have included the four sidesets that were defined. Sideset 1, 2 and 3 represent the free flow boundaries and sideset 4 represents the solid wall cylinder boundary. A constant inflow velocity is set on surface_1 in the x (streamwise) direction. This corresponds to a uniform inflow. An outflow boundary condition is set on surface_2, where the pressure is specified. On Surface 3, we specify a symmetry boundary condition which effectively allows the flow to “slip” on those surfaces, and do not take parameters in the symmetry boundary condition input block. Last, we specify a wall boundary condition on surface_4, which is effectively a no-slip condition. In this wall boundary condition block, we ask to postprocess “yplus” (y^+), see [Wall Functions; Momentum](#) for its definition.

Results Output

Scope: Region

Results are typically read into an Exodus file, at regular intervals set by the user.

```
Begin Results Output Label Fuego_output
    Database Name = results/cylinder.e
    At Time 0, Interval = 0.1
    Title Turbulent backward facing step flow
```

(continues on next page)

(continued from previous page)

```
Nodal Variables = Pressure as P
Nodal Variables = X_Velocity as Ux
Nodal Variables = Y_Velocity as Uy
Nodal Variables = yplus
End
```

Here, we set the name of the file into the `results/` directory. If this directory does not already exist, then it will be created. We also wish to output various nodal fields pertaining to the solution of this problem, including pressure, velocity, and the `yplus` values we calculated in the previous section. By putting “x” and “y” suffix allows for visualization applications like Paraview and Ensight to store them in a vector field for easier visualization. For reference, the logfile will display all fields (global, element, and nodal) available for output shown below:

```
Fuego Region "Fuego_region" has the following fields available for output:
GLOBAL Variables:
  * AREA_X_surface_1
  * mass_flow_rate_surface_2
  * surface_4_MAXIMUM_YPLUS
NODE_RANK Fields:
  * pressure
  * x_velocity
  * y_velocity
  * yplus
ELEMENT_RANK Fields:
  * CFL
  * ap_ip
  * mass_flux
```

Any of the fields listed here can be output into an exodus file.

Running Fuego and Reading The Logfile

To run Fuego, open a terminal and navigate to where your input files and mesh files are located, and run the following commands:

```
module load sierra
mpirun -n 4 fuego -i flow_over_cylinder.i -o logfile.log
```

The first command loads the `sierra` module which is available on most brokered workstations. If these modules do not exist for your specific workflow, please contact your IT team for information on how to load the `sierra/Fuego` environment on your particular machine. The second command will launch a parallel job with 4 processors running the `fuego` executable. The `-i` option is used to specify the input file you want to run Fuego with, and the `-o` option will specify the logfile output

name. This is an optional command, and omitting it will default to naming the logfile as `flow_over_cylinder.log`.

Some things to look for in a logfile involve

- Error messages (if something went wrong)
- Warnings (deprecation warnings, assumed defaults)
- Solution progression
 - Appropriately small non-linear residual values
 - Linear solver performance
 - Maximum DOF values
 - Time step info

We will make some general observations in this snippet from the logfile.

```
Step 2      , Time = 3.750e-03, dT = 2.250000e-03, Simulation is 0.0%
↳complete

Fuego      Nonlinear Residuals of the Equation Sets      Linear      Linear
Nonlinear      Cell Reynolds number(max) = 250.7      Solver      Solver
Iteration      CFL(max) = 0.1658      Iterations
↳Residual      DOF Min      DOF Max

-----
↳- -----
1 of 2 : SubMechanicsManager
1 of 1 : MomentumSubMech
          X_Momentum          = 7.06e-03      2      2.42e-
↳12 0.00e+00 1.51e+00
          Y_Momentum          = 7.40e-04      2      5.02e-
↳12 -6.93e-01 6.80e-01
          Continuity          = 1.45e-04      16      2.83e-
↳09 -9.17e-01 2.33e+00
          Velocity pre-correction -----
↳-> 0.00      1.51
          Velocity post-correction -----
↳-> 0.01      0.68
Fuego      Nonlinear Residuals of the Equation Sets      Linear      Linear
Nonlinear      Cell Reynolds number(max) = 250.7      Solver      Solver
Iteration      CFL(max) = 0.2487      Iterations
↳Residual      DOF Min      DOF Max

-----
↳- -----
```

(continues on next page)

(continued from previous page)

```
2 of 2 : SubMechanicsManager
1 of 1 : MomentumSubMech
      X_Momentum                = 4.78e-05      2      3.15e-
→12  0.00e+00  6.85e-01
      Y_Momentum                = 1.67e-05      2      4.76e-
→12 -3.08e-01  3.25e-01
      Continuity                 = 4.98e-05     13      3.24e-
→09 -3.86e-01  5.83e-01
      Velocity pre-correction -----
→->    0.00      0.69
      Velocity post-correction -----
→->    0.02      0.62
```

This is a typical output in the logfile you will see when running Fuego. Here we show the maximum CFL number, as well as the performance of the equations systems and their linear solver residuals. You'll want to make sure that your nonlinear residual is going down, and that for each nonlinear residual step that the linear solver residual is suitably small ($O(1e-10)$) to ensure nonlinear convergence of the solution. Also the DOF (degrees of freedom) max and min for each equation system is also shown. If these are unnecessary high (or low), there might be some issue you need to address. Be sure to read the logfile during execution to catch any problems, and make sure that the logfile is outputting what you expect it to.

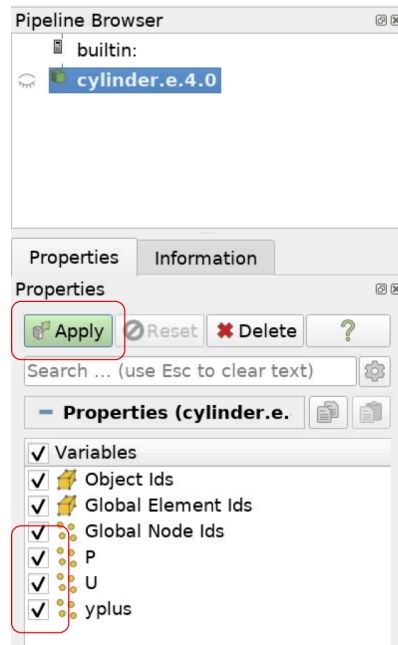
Viewing Results

From a CEE unix environment with graphics, you can launch paraview using

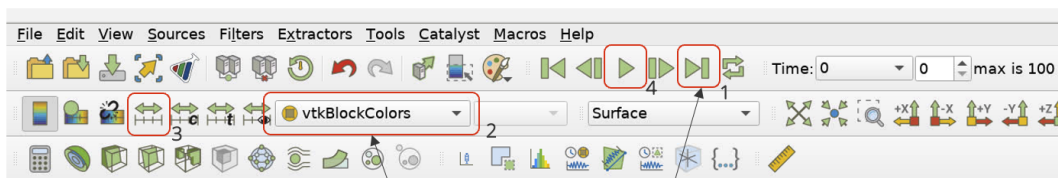
```
module load viz
paraview cylinder.e.4.0
```

Or if you have a local paraview installation, use that and open the exodus file appropriately.

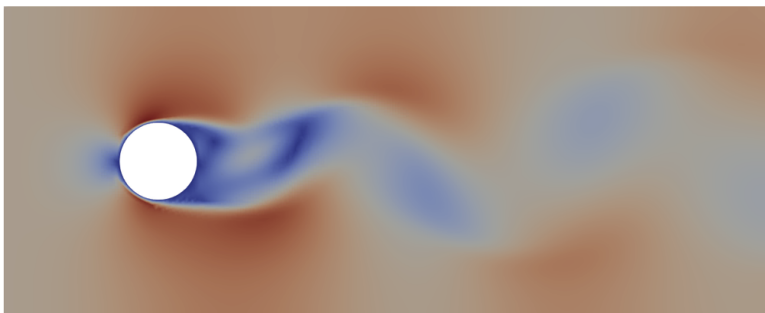
Once you have loaded paraview, you need to select which variables to show and click "Apply"



Once the variables are loaded, you can visualize the flow field using paraview



Go to the last time step (1), then pick "U" to view the velocity vector magnitude (2)

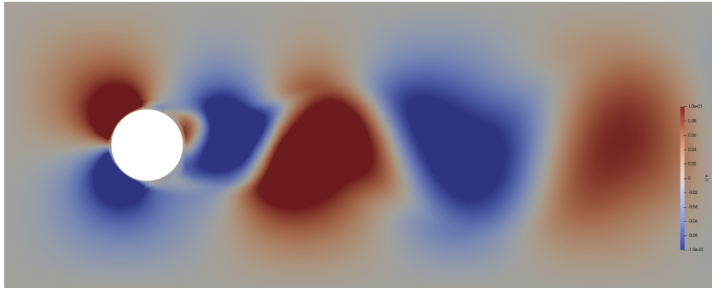
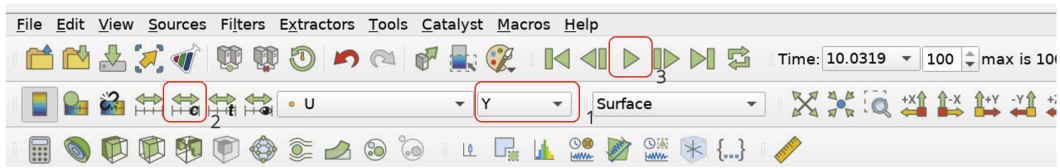


If you show "U" first, the color scale will be off when you go to the final time. Use the "rescale" button (3) to fix it.

Click "Play" (4) to watch the flow evolve.

Here we color the mesh using the velocity field. In the case of a vector field, this is visualized as the vector magnitude. Notice the vortex shedding pattern.

We can visualize this vortex shedding pattern a little differently by coloring the solution with the y-component of the velocity field



Change the visualization to show the Y component of velocity instead of magnitude (1)

Rescale the color range to go from -0.1 to 0.1 (2)

Click play (3) to watch the varying upward/downward flow sections

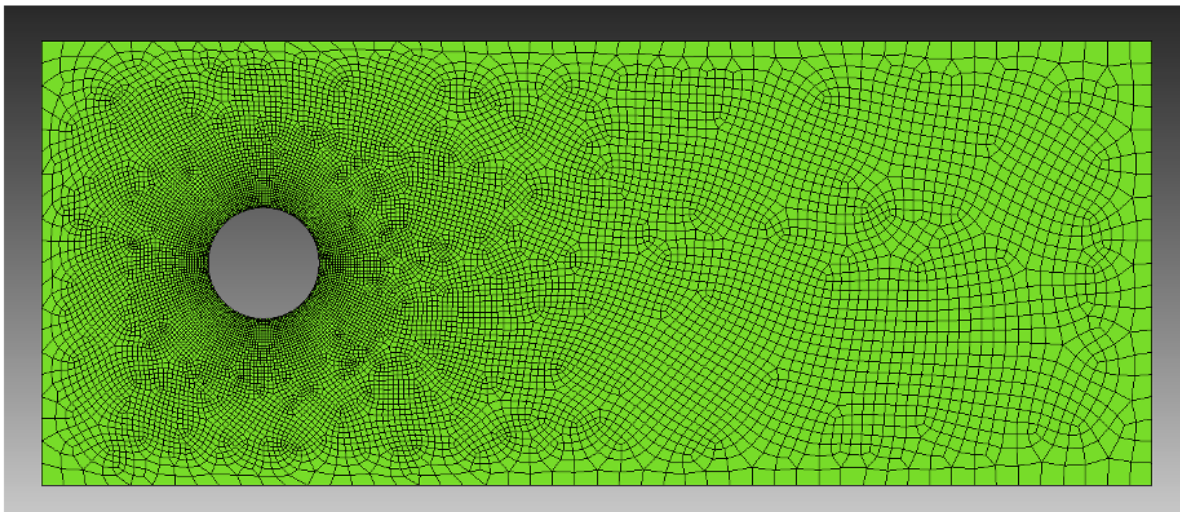
Hit the play button and observe the vortex shedding pattern. At what time does the vortex shedding begin? Is it periodic? Feel free to play around with visualizing different fields.

Changing the Mesh

Let us use a more refined mesh at the cylinder walls. Open the `cyl_better.jou` file and copy/paste into Cubit, or alternatively run it in command line mode

```
module load sierra
cubit -nographics -batch cyl_better.jou
```

- Better cylinder journal file
- Mesh file

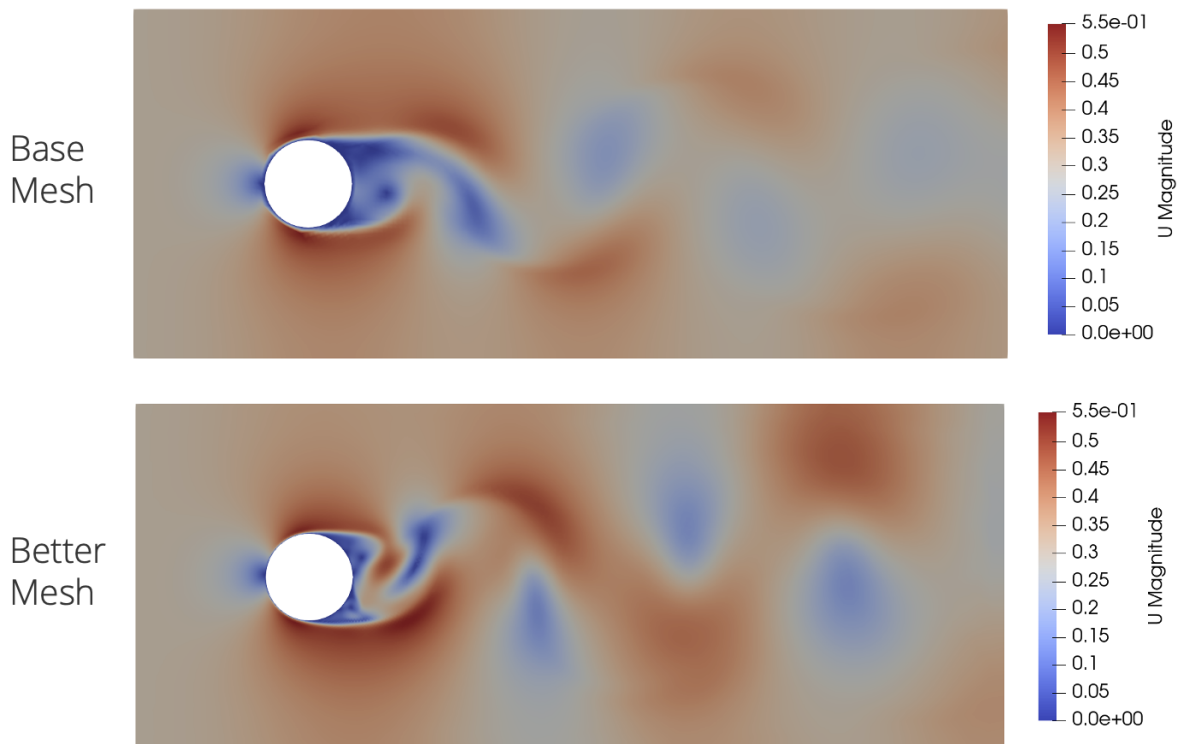


Running the journal file will make a new mesh named `cyl_better.g`. Let's change the mesh file in the finite element block and re-run the simulation.

```
Begin Finite Element Model cylMesh
  Database Name = cyl_better.g
  #other stuff
End Finite Element Model cylMesh
```

```
module load sierra
mpirun -n 4 fuego -i flow_over_cylinder.i
```

Let's take a look at the new results, colored by the velocity vector magnitude, compared to the old mesh



Notice in the execution, while the element count between the two meshes were close, this new mesh took more time steps. This is due to the CFL condition we have placed on the problem

$$CFL = \frac{u\Delta t}{\Delta x}$$

Where the elements near the cylinder wall are now smaller than the previous mesh. Therefore, all things being otherwise equal, the time step must shrink to accommodate this.

Compared to the old and new mesh, we see much better resolution of the vortex shedding event, especially near the cylinder.

Troubleshooting

In general, if your simulation fails to run correctly, or produces results different from what you expected, here are some steps you can take:

- Check if SAW highlighted any syntax in your input file
- Check the log file! Look for:
 - The words “Error” and “Warning”
 - High max CFL
 - Linear and nonlinear residual non-convergence
 - High maximum velocities, or large velocity corrections
- Visualize the solution (ParaView), and look for anomalous regions of flow

Turbulent Flow Over A Cylinder

This tutorial shows how to set up a turbulent over cylinder in Fuego. This module has also been recorded in prior trainings and is available [here](#) or in the player below.

This is an extension of the previous tutorial *Laminar Flow Over A Cylinder*. If you have not completed that tutorial, please do so before starting this tutorial as the information discussed in the laminar flow tutorial will be used in this tutorial.

The governing equations employed by Fuego correctly capture the effects of turbulence, provided a fine enough mesh is used. However, in most cases, the finest mesh we can afford to use is not fine enough to resolve the turbulent fluctuations. In those cases, a better prediction of the flow may be made by adding a turbulence model. For this training, we’re going to use the subgrid scale kinetic energy one-equation (k^{SGS}) model. For an in depth discussion on this model, please see *Subgrid-Scale Kinetic Energy One-Equation LES Model*.

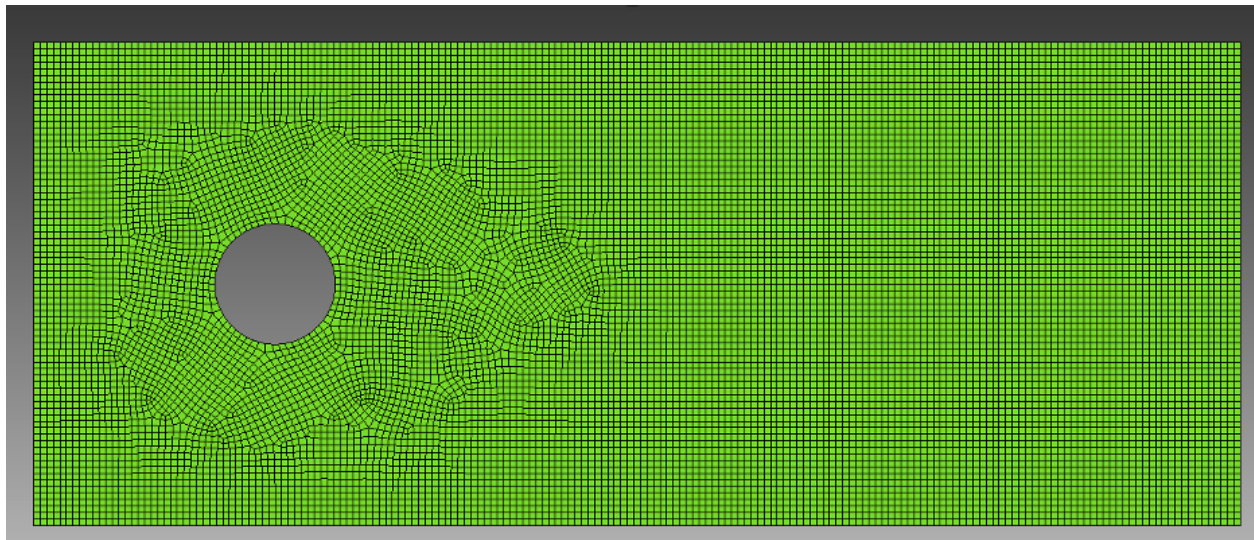
Problem Files

The files required for this tutorial can be downloaded [here](#), or found in a CEE environment at `/projects/sierra100/TF/Fuego`.

- Fuego input file
- Mesh file
- Training slides

Problem Domain/Mesh

The problem domain and mesh are identical to the ones in *Laminar Flow Over A Cylinder*



Adding a Turbulence Model

Activating the Turbulent Kinetic Energy Equation

The first step is to activate the necessary equations and specify the turbulence model within Solution Options

- The k^{SGS} model only requires the activation of the turbulent kinetic energy equation
- We also need to add a block specifying the details of the turbulence model

Begin Solution Options

Coordinate System = 2D

Projection Method = Fourth_Order Smoothing with Timestep Scaling

DEFINE WHAT EQUATIONS TO SOLVE

Activate Equation Continuity

Activate Equation X_Momentum

Activate Equation Y_Momentum

Activate Equation turbulent_kinetic_energy

DEFINE NONLINEAR ITERATION SETTINGS

Minimum Number of Nonlinear Iterations = 1

Maximum Number of Nonlinear Iterations = 2

(continues on next page)

(continued from previous page)

```
# DEFINE ADVECTION SCHEME
Upwind Method is MUSCL
UPWIND LIMITER IS SuperBee

Begin Turbulence Model Specification
    Turbulence Model = KSGS
End

End    Solution Options
```

Here we activate the `turbulent_kinetic_energy` equation. This adds the turbulent kinetic energy (TKE) equation to the list of equations `fuego` will solve each time step. We must specify which turbulence model to use, and in our case we will be using the KSGS model, which goes in the `Turbulence Model Specification`` block.

Applying Boundary Conditions and Initial Conditions

```
Begin Initial Condition Block FlowInit
    Volume is block_1
    Pressure    = 0.0
    X_Velocity  = 0.0
    Y_Velocity  = 0.0
    turbulent_kinetic_energy = 1e-4
End

# SET BOUNDARY CONDITIONS
Begin Inflow Boundary Condition on Surface surface_1
    X_Velocity = (2000*1.8e-5)/(1.2*0.1)
    Y_Velocity = 0.0
    turbulent_kinetic_energy = 1e-4
End

Begin Open Boundary Condition on Surface surface_2
    Pressure = 0.0
    turbulent_kinetic_energy = 1e-4
End

Begin Symmetry Boundary Condition on Surface surface_3
End

Begin Wall Boundary Condition on Surface surface_4
```

(continues on next page)

```

    post process yplus
End

```

The initial and boundary conditions for velocity and pressure are the same for the laminar tutorial, except now we must add conditions for the turbulent kinetic energy which is just a scalar degree of freedom. Note that we did not add a boundary condition for TKE on the wall (surface_4) due to the fact that a zero-flux condition will be defaulted on the wall boundary. TKE must be a positive quantity, and cannot go negative. If during the solution the TKE begins to go negative, Fuego will automatically “clip” the solution values to guarantee this to a lower limit determined by your problem units.

Results Output

Finally, we will add this new scalar DOF to our results output block

```

Begin Results Output Label Fuego_output
  Database Name = results/cylinder.e
  At Time 0, Interval = 0.1
  Title Turbulent backward facing step flow
  Nodal Variables = Pressure as P
  Nodal Variables = X_Velocity as Ux
  Nodal Variables = Y_Velocity as Uy
  Nodal Variables = yplus
  Nodal Variables = turbulent_kinetic_energy as TKE
End

```

Running Fuego and Reading The Logfile

To run Fuego, open a terminal and navigate to where your input files and mesh files are located, and run the following commands:

```

module load sierra
mpirun -n 4 fuego -i flow_over_cylinder_turb.i

```

Let’s take a quick look at the logfile to see what has changed from the laminar case

```

Step 2      , Time = 3.750e-03, dT = 2.250000e-03, Simulation is 0.0%_
↳complete

Fuego      Nonlinear Residuals of the Equation Sets      Linear      Linear
Nonlinear      Cell Reynolds number(max) =      168.6      Solver      Solver

```

(continues on next page)

(continued from previous page)

Iteration	CFL(max) = 0.1636		Iterations	
Residual	DOF Min	DOF Max		

- - - - -				
1 of 2	: SubMechanicsManager			
1 of 1	: MomentumSubMech			
	X_Momentum	= 7.05e-03	2	2.56e-
13	7.01e-02 1.51e+00			
	Y_Momentum	= 6.89e-04	1	1.50e-
07	-7.18e-01 7.05e-01			
	Continuity	= 1.45e-04	17	3.68e-
09	-4.40e-01 8.72e-01			
	Velocity pre-correction	-----		
->	0.07 1.51			
	Velocity post-correction	-----		
->	0.01 0.60			
1 of 1	: KSGSSubMech			
	Turbulent_Kinetic_Energy	= 1.99e-06	1	3.69e-
08	9.87e-05 1.18e-02			
Fuego	Nonlinear Residuals of the Equation Sets		Linear	Linear
Nonlinear	Cell Reynolds number(max) = 168.6		Solver	Solver
Iteration	CFL(max) = 0.2455		Iterations	
Residual	DOF Min	DOF Max		

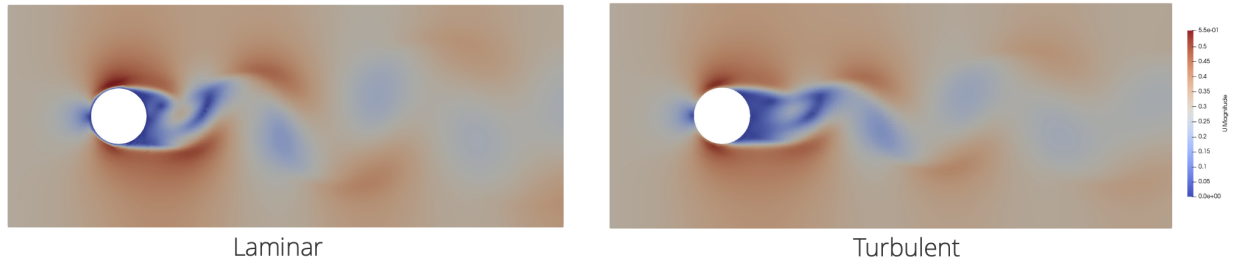
- - - - -				
2 of 2	: SubMechanicsManager			
1 of 1	: MomentumSubMech			
	X_Momentum	= 4.80e-05	1	6.01e-
08	1.16e-02 6.12e-01			
	Y_Momentum	= 1.77e-05	1	2.44e-
07	-3.01e-01 2.97e-01			
	Continuity	= 4.66e-05	13	8.62e-
09	-1.82e-01 4.50e-01			
	Velocity pre-correction	-----		
->	0.01 0.61			
	Velocity post-correction	-----		
->	0.01 0.61			
1 of 1	: KSGSSubMech			
	Turbulent_Kinetic_Energy	= 2.42e-07	1	2.72e-
08	9.87e-05 1.10e-02			
Elapsed wall time = 0.60 seconds (0.21 s this step)				

Notice that we are now tracking the Turbulent_Kinetic_Energy DOF in the logfile. It appears

that the nonlinear residual is decreasing during the nonlinear iteration step, which is the expected behavior. This shows that the TKE is converging. Again, always be checking the logfiles, they contain useful information about convergence rates and the DOF ranges.

Viewing Results

Please follow the steps from the previous tutorial to run ParaView *Viewing Results*.



At what time does the flow break symmetry and start shedding? (Laminar was ~2 s) Does the wake structure look different? Does the boundary layer on the upstream side of the cylinder look different?



Look at the new TKE output field. Where is the turbulent kinetic energy highest?

Changing Boundary Conditions

- Fuego input file with new Bcs

Let's change the constant inflow condition to something more interesting

```
Begin Inflow Boundary Condition on Surface surface_1
X_Velocity = "0.3*(1+0.75*sin(t)) * (y>0 ? 1 : 0.1)"
Y_Velocity = 0.0
```

(continues on next page)

(continued from previous page)

```
Turbulent_Kinetic_Energy = 1.0e-4  
End
```

Here, we replace the inflow boundary condition with a string function, illustrated in the image below



Run the code as usual, and visualize the results



Post-processing

This tutorial demonstrates some capabilities of the Fuego post-processor. This module has also been recorded in prior trainings and is available [here](#) or in the player below.

Problem Files

The files required for this tutorial can be downloaded here, or found in a CEE environment at /projects/sierra100/TF/Fuego.

- Fuego input file
- Mesh file
- Training slides

Flux Post-processors

Flux post-processors are available in boundary condition blocks, e.g.

```
Begin Inflow Boundary Condition on Surface surface_1
  X_Velocity = (2000*1.8e-5)/(1.2*0.1)
  Y_Velocity = 0.0
  turbulent_kinetic_energy = 1e-4
  Postprocess total flux of continuity as mdot_in
End

Begin Open Boundary Condition on Surface surface_2
  Pressure = 0.0
  turbulent_kinetic_energy = 1e-4
  Postprocess total flux of continuity as mdot_out
End
```

Here we wish to postprocess the total mass flux in and out of the domain. For this, the general syntax is:

```
Postprocess [total|advective|diffusive] flux of [EQUATION] as [SOME_NAME]
```

In this particular case, two new global variables are created, mdot_in and mdot_out.

Post-processor Block

Scope: Region

Flux post-processors are available in boundary condition blocks, e.g.

```
Begin Postprocess point
  Output name = p_point
  Location = 0 0
```

(continues on next page)

(continued from previous page)

```
Function = "pressure"  
End
```

In this example, we wish to postprocess the value of pressure at a specific point in space.



This will track the pressure over time at this specific point, with a global variable named `p_point`. Note that the `function = "pressure"` can be changed to another function. For instance, we can make a function that depends on the x-velocity, or some other arbitrary function. This is shown in the integral post-processor in the next block

```
Begin Postprocess integral  
  Output name = total_x_mom  
  Location = all_blocks  
  Function = "density * abs(x_velocity)"  
End
```

In this case, the `integral` post-processor will spatially integrate the specified function over all blocks on the mesh.

For a list of other post-processor blocks see *Postprocess*.

Global Variable Output

Heartbeat Files

Heartbeat files are used in Fuego to track and output global variables, and print them in .csv files or plain text, column-formatted files.

```
Begin Heartbeat myHeartbeat
  Stream Name = heartbeat.csv
  Format = csv
  At Step 0 increment = 1
  Variable is Global time
  Variable is Global mdot_in
  Variable is Global mdot_out
  Variable is Global total_x_mom
  Variable is Global p_point as Pressure
End
```

In this example we have included all of the post-processed global variables from the previous sections. This block command will create a heartbeat file named `heartbeat.csv` in .csv format, and will print out every time iteration in the Fuego run.

Results Output

Additionally we can track global variables in exodus output

```
Begin Results Output Label Fuego_output
  Database Name = results/cylinder.e
  At Time 0, Interval = 0.1
  Title Flow over a cylinder
  Nodal Variables = Pressure as P
  Nodal Variables = X_Velocity as Ux
  Nodal Variables = Y_Velocity as Uy
  Nodal Variables = yplus
  Global Variables = mdot_in
  Global Variables = mdot_out
  Global Variables = total_x_mom
  Global Variables = p_point
End
```

Running Fuego and Reading The Heartbeat File

To run Fuego, open a terminal and navigate to where your input files and mesh files are located, and run the following commands:

```
module load sierra  
mpirun -n 4 fuego -i flow_over_cylinder_postproc.i
```

Let's examine the heartbeat file

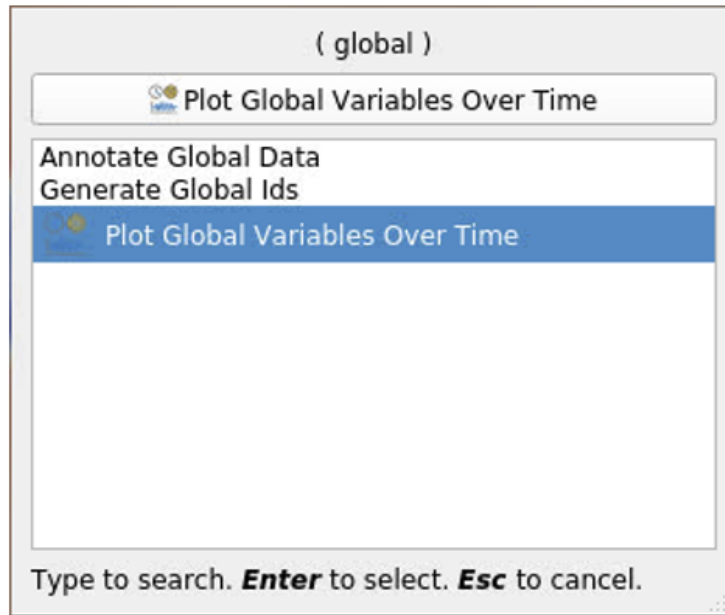
time	,	mdot_in,	mdot_out,	total_x_mom,	Pressure
0.000000e+00	,	0.000000e+00	, 0.000000e+00	, 0.000000e+00	, 0.000000e+00
1.500000e-03	,	-1.440000e-01	, 1.440000e-01	, 1.43907e-01	, 1.19911e+02
3.750000e-03	,	-1.440000e-01	, 1.440000e-01	, 1.44103e-01	, 2.36038e-04
7.125000e-03	,	-1.440000e-01	, 1.440000e-01	, 1.44069e-01	, 5.50133e-04
1.21875e-02	,	-1.440000e-01	, 1.440000e-01	, 1.44024e-01	, 5.55514e-04
1.97813e-02	,	-1.440000e-01	, 1.440000e-01	, 1.44002e-01	, 5.55127e-04
3.11719e-02	,	-1.440000e-01	, 1.440000e-01	, 1.43996e-01	, 5.53956e-04

All of our global variables are printed out in this heartbeat file. The heartbeat file can be visualized using your plotting software of choice (python, Excel, Matlab, even ParaView)

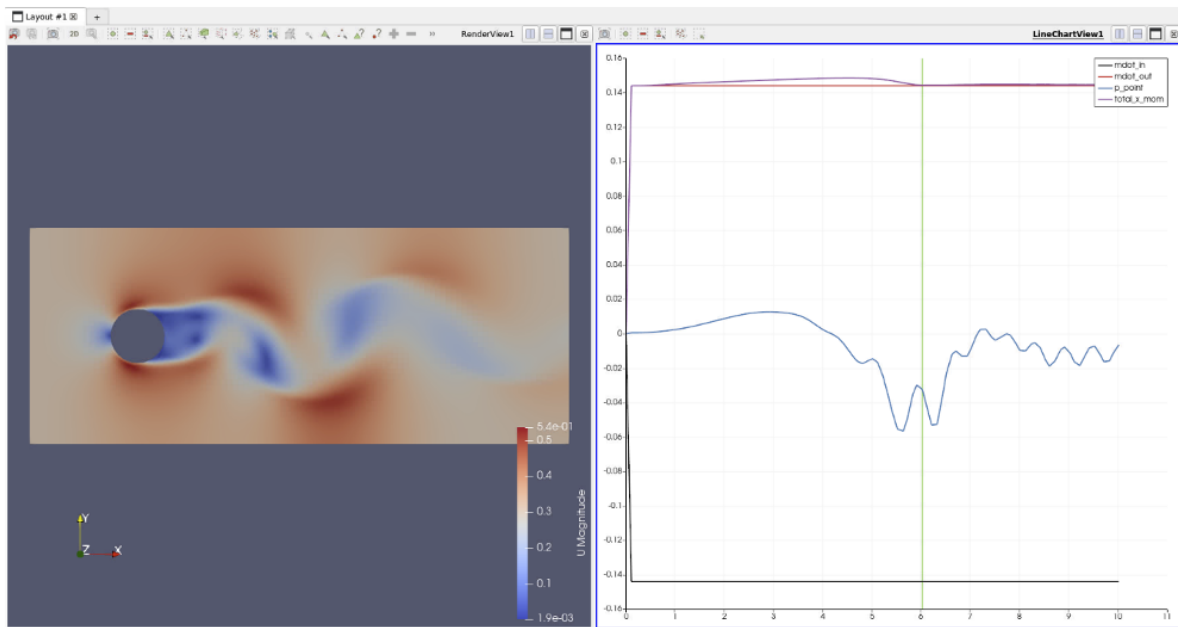
Viewing Results

Please follow the steps from the previous tutorial to run ParaView [Viewing Results](#).

Once the solution is loaded into ParaView, navigate to the top menu **Filters->Data Analysis->Plot Global Variables over Time**.



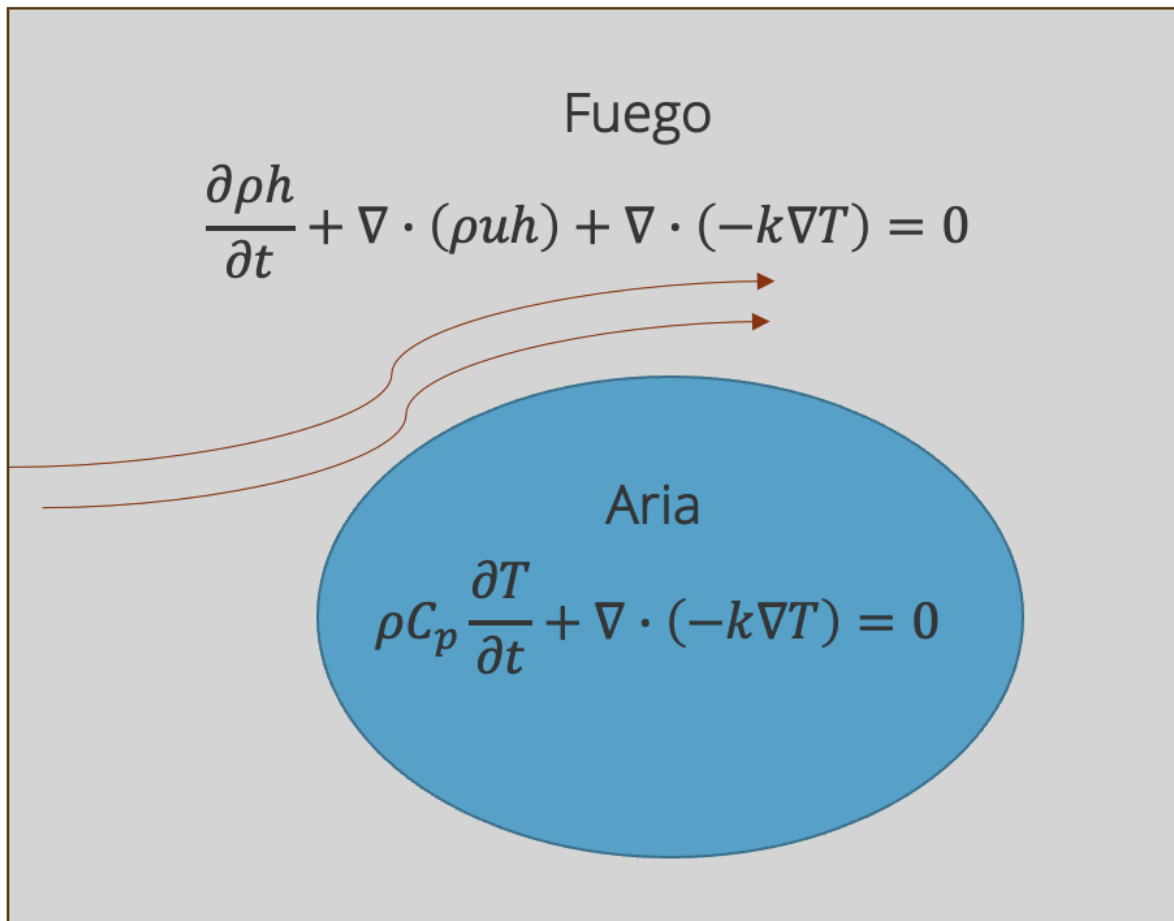
Color the results by velocity magnitude, and view the global variable data as a function of time



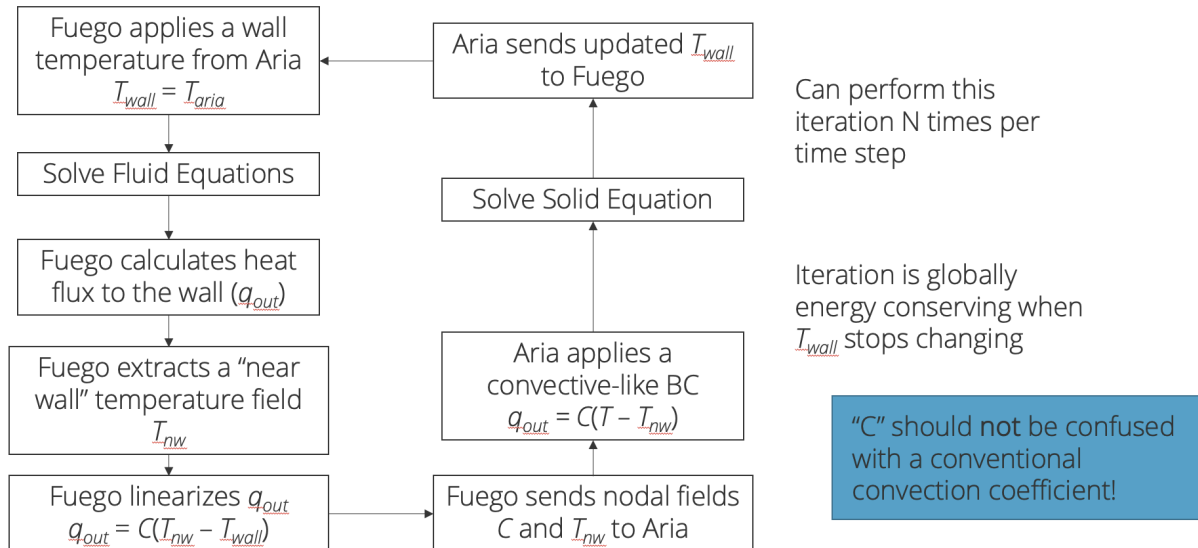
Conjugate Heat Transfer

This tutorial shows how to set up a conjugate heat transfer (CHT) problem. This module has also been recorded in prior trainings and is available [here](#) or in the player below.

For problems with solid heat conduction and non-isothermal flow, you can use the combined `fuego_aria`` executable to run a conjugate heat transfer simulation. A description of the proposed problem is shown below



Here, a heat transfer equation is solved in Aria inside the cylinder domain, and an enthalpy transport equation will be solved in the Fuego (exterior) domain. The coupling between the cylinder temperature computed in Aria and the enthalpy in Fuego at the wall will be done through an iterative procedure illustrated below



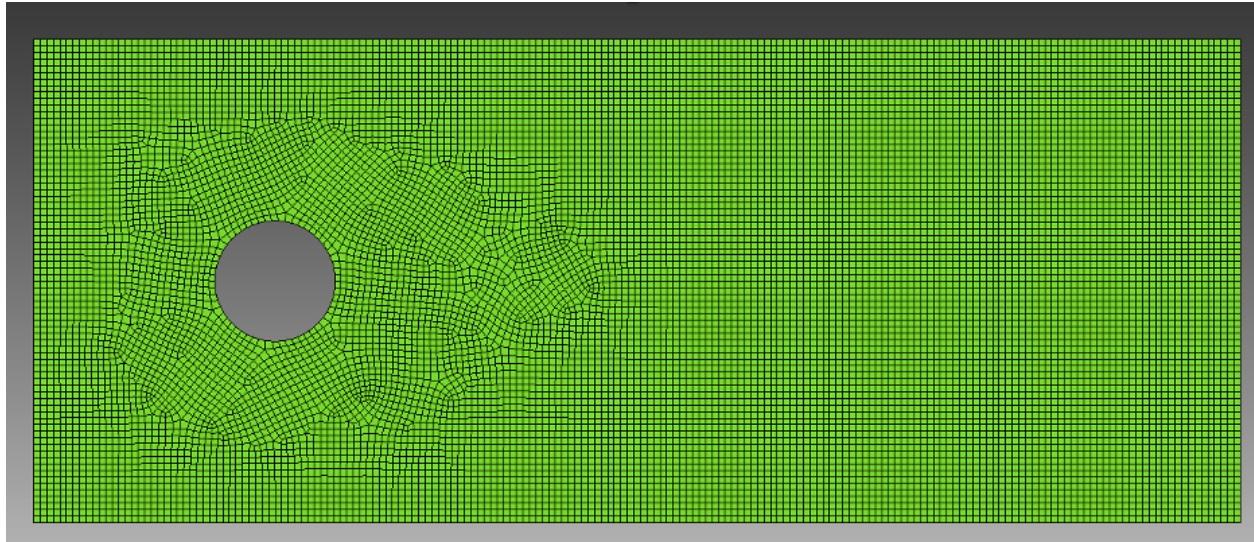
In this particular coupling procedure, Aria will send a wall temperature to Fuego, and Fuego, after doing a fluid solve, will send back a heat flux back to Aria. This is a conditionally stable procedure and in general will be 2nd order accurate.

Problem Files

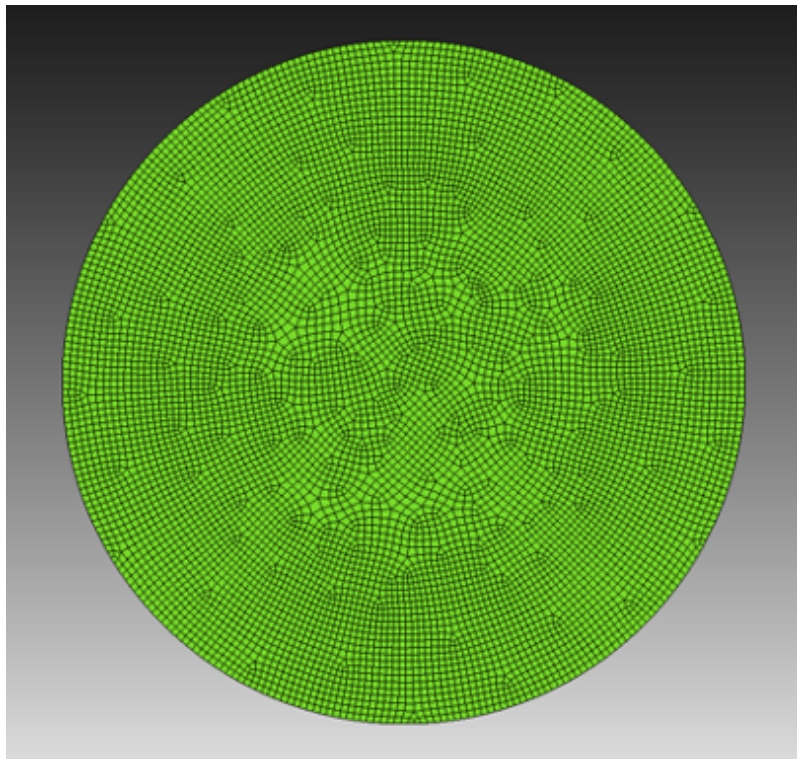
The files required for this tutorial can be downloaded here, or found in a CEE environment at `/projects/sierra100/TF/Fuego`.

- Fuego input file
- Cylinder mesh file
- Pipe mesh file
- Training slides

Problem Domain/Mesh



In this tutorial we will be simulating a 2D CHT problem over a cylinder cross-section, where the Fuego mesh (`cyl_base.g`) is shown above and the Aria mesh (`pipe.g`) is shown below



In the previous tutorials, we did not need to mesh the interior of the cylinder. However, since we are now solving a heat transfer problem inside the cylinder using Aria, we need to mesh the cylinder.

The mesh files needed for this tutorial can be downloaded in the links provided above, or it can be

generated manually using the Cubit journal file below.

Cubit Journal File

```
undo on
# { D = 0.1 }

brick x {10*D} y {4*D} z 1
create cylinder height 1 radius {D/2}
move volume 2 x {-3*D}
subtract volume 2 from volume 1

block 1 add surface 11
block all element type QUAD4
surface 11 size {0.054*D}
mesh surface 11

sideset 1 add curve 3
sideset 2 add curve 1
sideset 3 add curve 2 4
sideset 4 add curve 16
set large exodus file on
export genesis "cyl_base.g" dimension 2 block all overwrite

reset

create cylinder height 1 radius {D/2}
move volume 1 x {-3*D}
block 1 add surface 3
block all element type QUAD4
surface 3 size {0.01*D}
mesh surface 3

sideset 1 add curve 2
set large exodus file on
export genesis "pipe.g" dimension 2 block all overwrite
```

Fuego Aria Input File

There are three things we will cover in this section

- Activating an enthalpy equation in Fuego for non-isothermal flow
- Adding in an Aria region and Aria materials
- Defining the coupling and transfers between Fuego and Aria

The input file we are using can be found [here](#).

Activating Enthalpy

For non-isothermal flow, we need enthalpy, specific heat, thermal conductivity, and a reference temperature

Scope: Domain

```
Begin Property Specification for Fuego Material air
Density    = 1.2*(300/temperature)
Viscosity  = 1.8e-5*(temperature/300)^0.7
Thermal_conductivity = 0.025
Specific_heat      = 1000
Enthalpy          = 1000*(temperature-300)
Reference Temperature = 300.0
End   Property Specification for Fuego Material air
```

In the Fuego material, we provide the function for enthalpy as a string function

$$h = C_p(T - T_{ref})$$

Fuego solves for enthalpy transport (h) and the temperature is post-processed from enthalpy using the above relationship.

Now we are going to activate the enthalpy equation in Fuego

$$\frac{\partial \rho h}{\partial t} + \nabla \cdot (\rho u h) + \nabla \cdot (-k \nabla T) = 0$$

Scope: Region

```
Begin Solution Options
Coordinate System = 2D
Projection Method = Fourth_Order Smoothing with Timestep Scaling

# DEFINE WHAT EQUATIONS TO SOLVE
```

(continues on next page)

```

    Activate Equation Continuity
    Activate Equation X_Momentum
    Activate Equation Y_Momentum
    Activate Equation turbulent_kinetic_energy
    Activate Equation Enthalpy #activate enthalpy equation

    ...
End    Solution Options

```

As well as ICs and BCs

Scope: Region

```

# SET INITIAL CONDITIONS
Begin Initial Condition Block FlowInit
    Volume is block_1
    Pressure = 0.0
    X_Velocity = 0.0
    Y_Velocity = 0.0
    Temperature = 300.0
    turbulent_kinetic_energy = 1e-4
End

# SET BOUNDARY CONDITIONS
Begin Inflow Boundary Condition on Surface surface_1
    X_Velocity = (2000*1.8e-5)/(1.2*0.1)
    Y_Velocity = 0.0
    turbulent_kinetic_energy = 1e-4
    Temperature = 300.0
    Postprocess total flux of continuity as mdot_in
End

Begin Open Boundary Condition on Surface surface_2
    Pressure = 0.0
    turbulent_kinetic_energy = 1e-4
    Temperature = 300.0
    Postprocess total flux of continuity as mdot_out
End

Begin Symmetry Boundary Condition on Surface surface_3
End

Begin Wall Boundary Condition on Surface surface_4

```

(continues on next page)

(continued from previous page)

```
post process yplus
Interface Boundary
Calculate convection coefficient using Tref = 300
End
```

Here we need to put ICs and BC for temperature for the enthalpy equation. We also mark the cylinder wall (surface_4) as an (Interface Boundary), and to postprocess the convection coefficient using a reference temperature of 300 Kelvin.

$$h = \frac{Q}{T_{wall} - T_{ref}}$$

Now we need to add these new variables to the results output block, along with the newly temperature-dependent properties

```
Begin Results Output Label Fuego_output
Database Name = results/fluid.e
At Time 0, Interval = 0.1
Title Flow over a cylinder

...

Nodal Variables = heat_flux
Nodal Variables = wall_temperature as Twall
Nodal Variables = Temperature as T
Nodal Variables = wall_convection_coefficient as hConv
Nodal Variables = density
Nodal Variables = viscosity

...

End
```

Defining Aria Materials

Now we will define the Aria materials that will correspond to the interior of the cylinder

```
Begin Aria Material aluminum_foam
Density = Constant rho = 50
Thermal Conductivity = Constant k = 1
Specific Heat = Constant cp = 50
Heat Conduction = Generalized
End
```

(continues on next page)

```

Begin Aria Material cht_interface
  BC Reference Temperature = user_field name=bc_ref_temp
  Heat Transfer Coefficient = user_field name=bc_ref_h
End

```

The `aluminum_foam` material needs density, thermal conductivity, and specific heat to be defined for the heat equation Aria will be solving.

$$\rho C_p \frac{\partial T}{\partial t} + \nabla \cdot (q) = 0$$

The “generalized” heat conduction model corresponds to the Fourier heat transfer model

$$q = -k \nabla T$$

Next the interface material is defined. Here, the reference temperature is specified to use a `user_field` named `bc_ref_temp`. Similarly the heat transfer coefficient will also use a `user_field` named `bc_ref_h`. These user fields are to be supplied by Fuego and used by Aria.

Defining Aria Solvers and Mesh

Now we need to define the solver that Aria will use (see [Tpetra Solvers](#) for more information)

Scope: Domain

```

BEGIN TPETRA EQUATION SOLVER  HEAT
  BEGIN PRESET SOLVER
    SOLVER TYPE = THERMAL
  END
END TPETRA EQUATION SOLVER

```

As well as which mesh we want the finite element block corresponding to the Aria region to use

```

Begin Finite Element Model Pipe
  Database Name = pipe.g
  Database Type = ExodusII

  Use material aluminum_foam for block_1
  Use material cht_interface for surface_1
End   Finite Element Model Pipe

```

Here we tell the finite element model to use the `aluminum_foam` material for `block_1` and the `cht_interface` material for `surface_1`, which correspond to the interior of the cylinder and interface between the Aria and Fuego regions, respectively.

Adding Aria Region

Next we will define the Aria region in the input file (please see the [Aria User Manual](#) for more information.)

```
Begin Aria Region Solid_region
  Use Linear Solver Heat
  Nonlinear Solution Strategy = Newton
  Use Dof Averaged Nonlinear Residual
  Use Finite Element Model Pipe

  EQ Energy For Temperature On all_blocks Using Q1 With Diff Mass

  IC for temperature on all_blocks = constant value = 500

  User Field Real Node Scalar bc_ref_temp On surface_1
  User Field Real Node Scalar bc_ref_h    On surface_1

  BC Flux for energy on surface_1 = Generalized_Nat_Conv

  Begin Results Output Label Aria_output
    Database Name = results/solid.e
    At Time 0, Interval = 0.1
    Title Temperature In The Solid Pipe
    Nodal Variables = solution->Temperature As T
  End   Results Output Label Aria_output
End    Aria Region Solid_region
```

We will briefly go over some portions of the Aria region.

```
Use Linear Solver Heat
Nonlinear Solution Strategy = Newton
Use Dof Averaged Nonlinear Residual
Use Finite Element Model Pipe
```

In the Aria region we set the linear solver we defined in the previous section, as well as to tell it to use a Newton nonlinear solution strategy. The region must also know what finite element model to use, similar to the Fuego region.

```
EQ Energy For Temperature On all_blocks Using Q1 With Diff Mass

IC for temperature on all_blocks = constant value = 500

User Field Real Node Scalar bc_ref_temp On surface_1
```

(continues on next page)

(continued from previous page)

```
User Field Real Node Scalar bc_ref_h    On surface_1
BC Flux for energy on surface_1 = Generalized_Nat_Conv
```

Here we set what equations we want Aria to solve. In this case, we want Aria to solve the energy equation defined in *Defining Aria Materials*. In addition we also set the initial condition for temperature on the block, as well as a flux boundary condition on the interface `surface_1`.

$$Q = h(T - T_{ref})$$

User fields are also defined on `surface_1` corresponding to the reference temperature fields previously defined on the interface material. These are the fields that will be transferred from Fuego and used in the Aria solves.

Finally a results output block is added on the Aria region

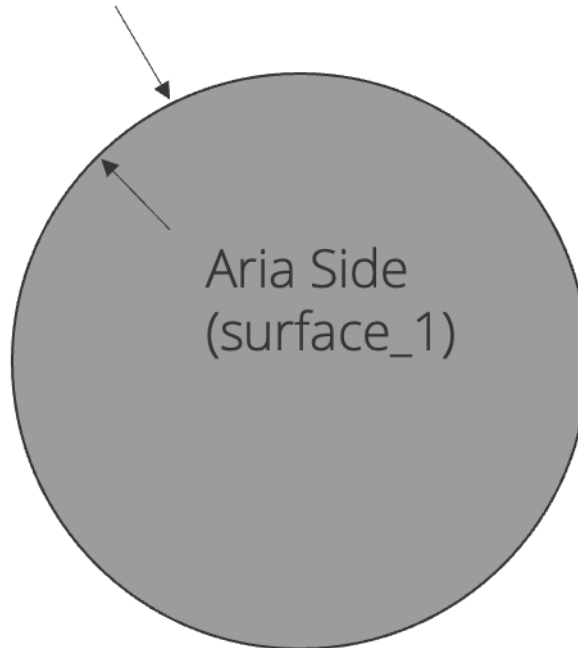
```
Begin Results Output Label Aria_output
  Database Name = results/solid.e
  At Time 0, Interval = 0.1
  Title Temperature In The Solid Pipe
  Nodal Variables = solution->Temperature As T
End Results Output Label Aria_output
```

So that important variables are output to the appropriate Exodus file.

Setup Transfer between Fuego and Aria

Now we will set up the communication pattern between Fuego and Aria using transfers. Recall the surfaces on the Aria and Fuego regions below

Fuego Side
(surface_4)



We will need to set up the field transfers between the Fuego surface (surface_4) and the Aria surface (surface_1)

Scope: Procedure

```
Begin Transfer solid_to_fluid_init
  Interpolate Surface Nodes From Solid_region To Fluid_region
  Send Block surface_1 To surface_4
  Send Field solution->Temperature State Old To Wall_temperature State_
  ↳None
  geometric tolerance = 5e-3
  nodes outside region = abort
End

Begin Transfer fluid_to_solid
  Interpolate Surface Nodes From Fluid_region To Solid_region
  Send Block surface_4 To surface_1
  Send Field flux_linearization_coefficient State None To bc_ref_h_
  ↳State None
  Send Field flux_linearization_temperature State None To bc_ref_temp_
  ↳State None
```

(continues on next page)

```

    geometric tolerance = 5e-3
    nodes outside region = abort
End

Begin Transfer solid_to_fluid
    Interpolate Surface Nodes From Solid_region To Fluid_region
    Send Block surface_1 To surface_4
    Send Field solution->Temperature State New To Wall_temperature State_
    ↪None
    geometric tolerance = 5e-3
    nodes outside region = abort
End

```

First we must define the transfer blocks. This is split up into three separate blocks. Let's look at them a bit more closely.

```

Begin Transfer solid_to_fluid_init
    Interpolate Surface Nodes From Solid_region To Fluid_region
    Send Block surface_1 To surface_4
    Send Field solution->Temperature State Old To Wall_temperature State_
    ↪None
    geometric tolerance = 5e-3
    nodes outside region = abort
End

```

The `solid_to_fluid_init` block is the initial transfer between the Aria (solid) surface to the Fuego (fluid) surface. Because these meshes are not contiguous (e.g. the respective surfaces do not share nodes or sides), we use an interpolation transfer, and set a geometric tolerance for said transfer. This is useful in that the meshes between two separate regions can be quite different, where interpolation can handle the mapping from the Aria surfaces to the Fuego surfaces (and vice versa).

```

Begin Transfer fluid_to_solid
    Interpolate Surface Nodes From Fluid_region To Solid_region
    Send Block surface_4 To surface_1
    Send Field flux_linearization_coefficient State None To bc_ref_h_
    ↪State None
    Send Field flux_linearization_temperature State None To bc_ref_temp_
    ↪State None
    geometric tolerance = 5e-3
    nodes outside region = abort
End

```

This is the transfer block responsible for transferring fluid (Fuego) fields to the solid (Aria) fields,

as we have defined before. The `flux_linearization_coefficient` and `flux_linearization_temperature` are automatically computed in fuego for a wall interface boundary (see section [Wall](#)). These Fuego fields are sent from `surface_4` to the fields `bc_ref_h` and `bc_ref_temp` which are on `surface_1`. Aria will use these fields in the materials and BCs defined in the previous sections.

```
Begin Transfer solid_to_fluid
  Interpolate Surface Nodes From Solid_region To Fluid_region
  Send Block surface_1 To surface_4
  Send Field solution->Temperature State New To Wall_temperature State_
  ↪None
  geometric tolerance = 5e-3
  nodes outside region = abort
End
```

This last transfer block sends fields from Aria to Fuego. In this case, Aria is sending the `solution->Temperature` field to the Fuego `Wall_temperature` field which will be used as a boundary condition in Fuego.

Coupling Fuego and Aria

Solution control lets us control when Fuego and Aria are executed, when transfers happen, and when overall nonlinear convergence is met for a time step.

```
Begin Solution Control Description
  Use System Main

  Begin Initialize Mytransient_init
    Advance Fluid_region
    Advance Solid_region
    Transfer solid_to_fluid_init
  End

  Begin System Main
    Use Initialize Mytransient_init

    Begin Transient Mytransient
      Begin nonlinear CHT
        Advance Fluid_region
        Transfer fluid_to_solid
        Advance Solid_region
        Transfer solid_to_fluid
      End
```

(continues on next page)

(continued from previous page)

```
End
End

Begin Parameters for Nonlinear CHT
  Converged when "(Fluid_region.MaxInitialNonlinearResidual(0) < 1E-1 &&
→ \$
                                Solid_region.MaxInitialNonlinearResidual(0) < 1E-2)┐
→ || \$
                                CURRENT_STEP > 1"

End
End
```

Now that we have two regions, we want to set up a solution control description that will advance both regions and do the appropriate field transfers. First we have

```
Begin Initialize Mytransient_init
  Advance Fluid_region
  Advance Solid_region
  Transfer solid_to_fluid_init
End
```

Which uses the initialization transfer block shown in the previous section. First we initialize the fluid region, the solid region, and then do the transfer from solid to fluid (Aria to Fuego). This ensures that for the first actual time step, Fuego has a non-zero wall temperature.

Next we need to set up a nonlinear loop for the time stepping, and some convergence criteria

```
Begin System Main
  Use Initialize Mytransient_init

  Begin Transient Mytransient
    Begin nonlinear CHT
      Advance Fluid_region
      Transfer fluid_to_solid
      Advance Solid_region
      Transfer solid_to_fluid
    End
  End
End

Begin Parameters for Nonlinear CHT
  Converged when "(Fluid_region.MaxInitialNonlinearResidual(0) < 1E-1 && ┐
→ \$
                                Solid_region.MaxInitialNonlinearResidual(0) < 1E-2)┐
```

(continues on next page)

```
→ || \$
```

```
CURRENT_STEP > 1"
```

```
End
```

The nonlinear loop will continue to execute until the prescribed convergence criteria is satisfied. This corresponds to the coupling algorithm we outlined at the beginning of this tutorial.

Setting up the time stepping parameters is the same as in the previous tutorials, but now we must supply parameters for both the Aria (solid) and Fuego (fluid) regions

```
#-----
Begin Solution Control Description
...

Begin Parameters For Transient Mytransient
  Start Time      = 0.0
  Initial Deltat  = 0.1
  Termination Time = 10.0
$-----
$               Define Fuego time integration settings
$-----
Begin Parameters For Fuego Region Fluid_region
  Transient Step Type Is Automatic
  Cfl Limit = 5.0
  Time Step Change Factor = 1.5
End   Parameters For Fuego Region Fluid_region
$-----
$               Define Aria time integration settings
$-----
Begin Parameters For Aria Region Solid_region
  Initial Time Step Size = 0.1
  Time Step Variation = Adaptive
  Predictor-corrector Tolerance = 1.0e-3
  Time Integration Method = BDF2
End   Parameters For Aria Region Solid_region
End   Parameters For Transient Mytransient
```

Here, we specify a CFL limit for the automatic time stepping in the Fuego region. In the Aria region, we also set an adaptive time step tolerance with a 2nd order time stepping method. For more information on setting up the transient parameters in Aria, please see the [Aria User Manual](#).

Running Aria/Fuego and Reading The Logfile

To run the coupled simulation, open a terminal and navigate to where your input files and mesh files are located, and run the following commands:

```
module load sierra
mpirun -n 4 aria_fuego -i flow_over_cylinder_cht.i
```

A typical time step in a coupled simulation is shown here

```
----- For equation system name: main -----
Global predictor error = 4.245e-07
Time step selection: dt <= 8.734e-01 (based on Predictor-Corrector
↪Tolerance).
Time step selection: dt <= 7.224e-02 (based on Maximum Time Step Size
↪Ratio).
Time step selection: dt <= 1.798e+308 (based on Maximum Time Step Size).
Time step selection: dt <= 8.720e-02 (based on Stability limit for dt
↪Ratio).
Time step selection: dt = 7.224e-02 (Adaptive time stepping result).
Global Variable Value
-----
Region::execute() time for Solid_region: 1.994e-02 sec.
Transfer Solid_To_Fluid, time 2.578, time step 0.03612
  Fluid_region.MaxInitialNonlinearResidual:
    system -> X_Momentum : 1.59529e-07
    system -> Y_Momentum : 1.15093e-07
    system -> Continuity : 2.06795e-07
    system -> Turbulent_Kinetic_Energy : 2.9319e-09
    system -> Enthalpy : 0.0339821
  Fluid_region.MaxInitialNonlinearResidual() = 0.0339821
  Solid_region.MaxInitialNonlinearResidual:
    system -> main : 3.702e-06
  Solid_region.MaxInitialNonlinearResidual() = 3.702e-06
Minimum Time Step Selection:
      min          max
0.000e+00    3.611e-02    Fluid_region
1.000e-16    7.224e-02    Solid_region
0.000e+00    3.611e-02    CHT
Transient Mytransient: dt = 0.03611
Transient Mytransient, step 67, time 2.6144e+00, time step 3.6106e-02, 26.
↪14% complete
```

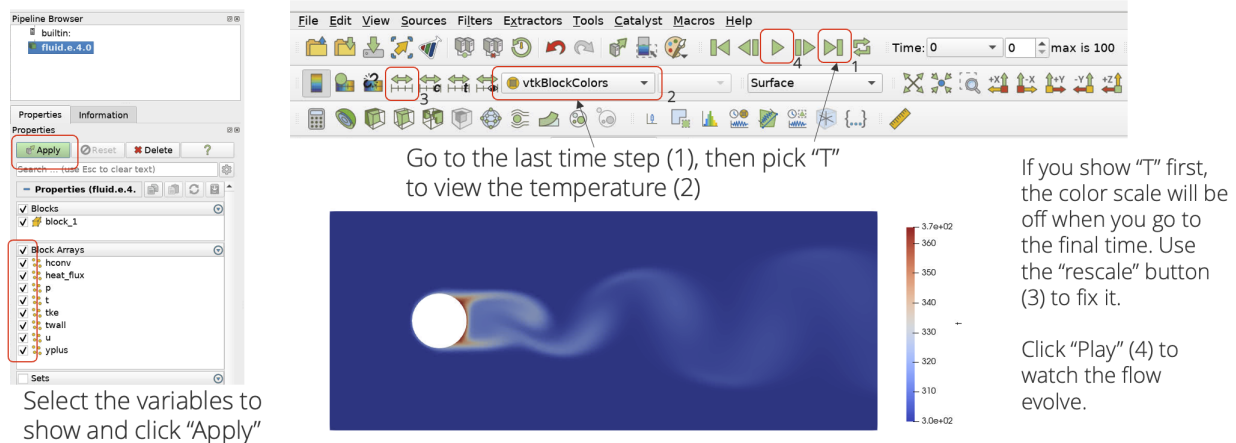
A couple of things to note here is the CHT residual summary. Here we show that Fluid_region.MaxInitialNonlinearResidual and

`Solid_region.MaxInitialNonlinearResidual` are both within the acceptable tolerance, so the coupled solution is converged. The timestep is chosen such that the minimum allowable timestep between both regions is selected. In this case, the `Fluid_region` is selecting the next timestep based on the CFL condition.

Viewing Results

Please follow the steps from the previous tutorial to run ParaView *Viewing Results*.

First lets visualize the fluid domain temperature.



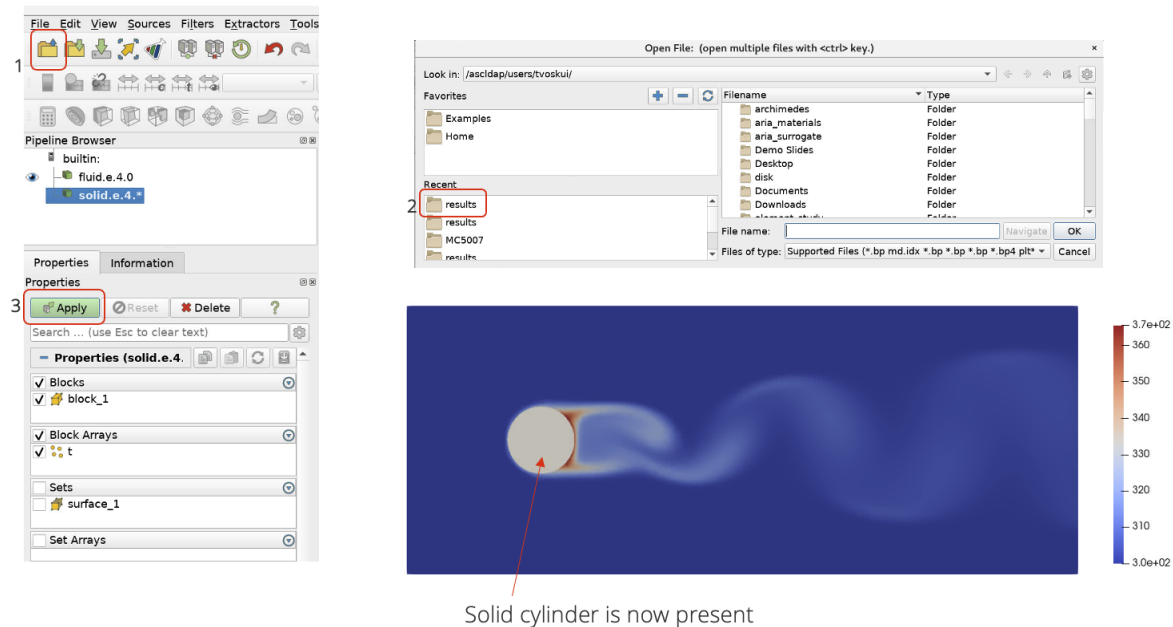
Go to the last time step (1), then pick "T" to view the temperature (2)

If you show "T" first, the color scale will be off when you go to the final time. Use the "rescale" button (3) to fix it.

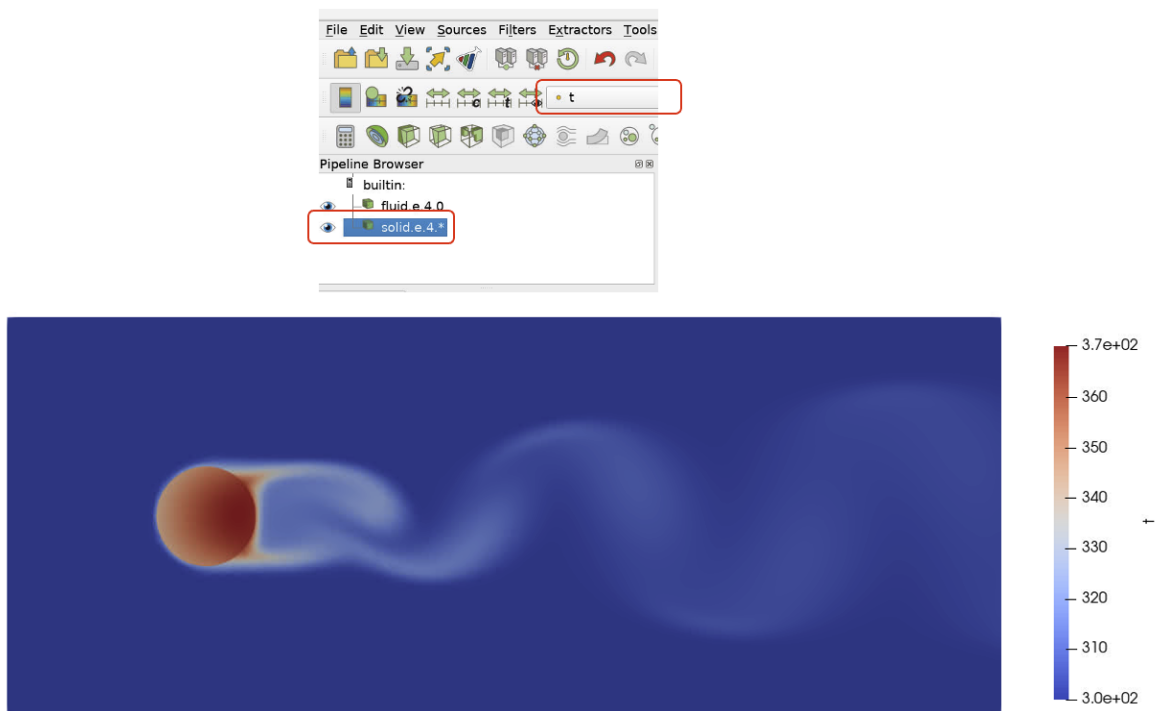
Click "Play" (4) to watch the flow evolve.

Select the variables to show and click "Apply"

Notice that since we have only loaded the Fuego results exodus file, we only see the Fuego region of the flow field. Let's load the solid (Aria) region in the same view.



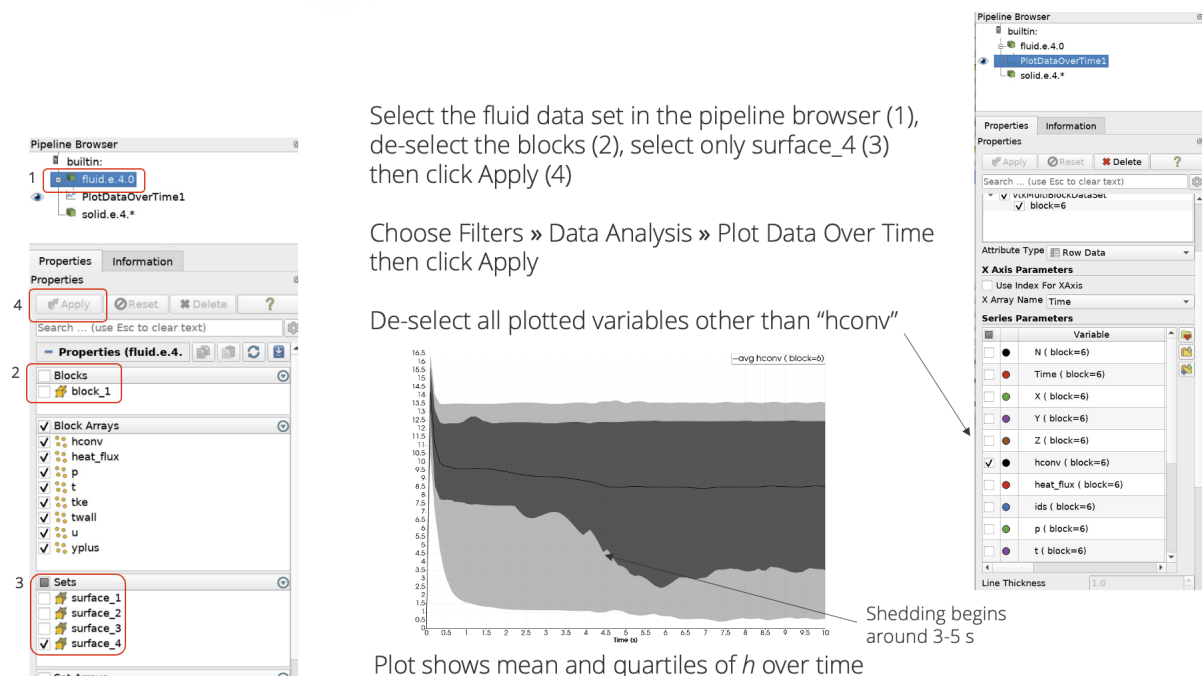
Now we can see the solid region cylinder, we can then color it by temperature as well.



Now we have a full picture of the temperature field on both the solid and fluid domains. Notice that the temperature inside the cylinder is lower at the leading edge compared to the trailing edge. This is an effect of the convective cooling of the relatively cold fluid temperature on the relatively

hot solid temperature. Notice how the temperature of the trailing vortices changes as it travels downstream.

Let's observe the convective coefficient and how it changes with time.



Here we can see how the global convective coefficient changes with time. When the vortex shedding starts, how does the convective coefficient behave? Is this expected?

6 Running & Troubleshooting

6.1 Running Fuego

Loading the Sierra Module

Once you have finished setting up an input file, you are ready to run Fuego. From a CEE or HPC UNIX environment at Sandia, you can load the sierra module to access the latest release of Fuego.

```
$ module load sierra
```

This will load the current release of Sierra. To load other versions of Sierra, you can use the following modules

- `module load sierra/x.x` - Load version `x.x` (e.g. `module load sierra/5.10`)

- `module load sierra/sprint` - Load the latest sprint release (every 3-week release)
- `module load sierra/daily` - Load the daily build of Sierra.



Warning: Using the `sierra/daily` module exposes you to potential bugs and instabilities since it is actively developed. If the nightly Sierra build process fails, the `sierra/daily` executable may not exist, or may be much older than expected.

To see a list of the available Sierra versions (and other useful modules available like `apps/anaconda3` or `apps/matlab`) you can use

```
$ module avail
```

Running Fuego Locally

To run a job on a non-queued system (e.g. a CEE blade or a cee compute machine) you can call `launch` or `mpirun`. For example, to run a job on 4 processors, you would use

```
$ module load sierra
$ launch -n 4 fuego -i demo.i
```

The `launch` command is usually equivalent to using `mpirun` for local execution, but handles setting required MPI flags when running on some HPC systems.

```
$ module load sierra
$ mpirun -np 4 fuego -i demo.i
```

Using the Sierra Script

The `sierra` script included in the `sierra` modules provides some additional functionality for launching `sierra` jobs. Its use is similar to the `launch` and `mpirun` commands.

```
$ sierra --np 4 fuego -i demo.i
```

By default, the `sierra` script will perform extra steps that are not necessary for running Fuego. These include:

- Reading the input file to find the mesh file and running `decomp` on it beforehand.
- Reading the input file to find the output files and running `eput` on them after the simulation is done.

Fuego will automatically decompose your mesh, which is usually faster than manually running `decomp`, and most visualization tools can view decomposed output files so running `eput` to

combine them into a single file is usually not needed either. To use the sierra script without invoking those steps, add the `--run` option.

```
$ sierra --run --np 4 fuego -i demo.i
```

Running Fuego on an HPC

The HPC systems at Sandia use slurm to schedule jobs. To run a job on an HPC you need to submit it to the slurm system along with some additional information (listed below) and the system will put your job in the queue and run at some point later in the future.

- The number of compute nodes to run on.
- The number of cores to use per node.
- The wall-clock duration of the job (slurm will kill the job if it's not done by this time limit).
- A WCID for the job based on the project funding it. The WCID is used for tracking purposes and also determines the job priority. Use the [WC Tool](#) web site to check your WCIDs or get a new one.
- Which queue to submit to (most HPCs have “batch”, “short”, and “long” - “batch” is the standard).

Refer to the [HPC homepage](#) for details about queue limits, core counts per node, and other useful HPC information for the machine you intend to run on.

If you are using SAW to submit your jobs, it can handle collecting the required information and submitting the job to the queue. If you do not use SAW, you can submit your jobs manually using either a batch script or the sierra script.

For simple job submissions you can use the sierra script directly after logging in to the HPC you want to run on. For example, to run Fuego on 360 processors for up to 24 hours you would log in to the HPC and run the following command.

```
$ sierra --run --np 360 fuego -i demo.i --account WCID --queue-name batch_
↪--time-limit 24:00:00
```

For more complicated submissions, you may need to prepare a batch script to perform any custom pre-processing steps you need. There are example submission scripts for different platforms in `/projects/samples` at Sandia. An example script to run Fuego is shown below.

```
#!/bin/bash
#SBATCH --nodes=10                # Number of nodes
#SBATCH --ntasks-per-node=36      # Number of cores per node
#SBATCH --time=24:00:00           # Wall clock time (HH:MM:SS)
#SBATCH --account=PUT_YOUR_WCID_HERE # WC ID
```

(continues on next page)

(continued from previous page)

```
#SBATCH --job-name=test           # Name of job
#SBATCH --partition=batch         # partition/queue name: short or
    ↪ batch

nodes=$SLURM_JOB_NUM_NODES
cores=36

# do any pre-processing steps you need

mpirun --bind-to core --npernode $cores --n $(( $cores*$nodes )) fuego -i
    ↪ demo.i
```

If you saved the above script as `run_fuego` then you would submit it to the queue using

```
$ sbatch run_fuego
```

You can check on the status of your queued jobs using `squeue -u myusername`. To see an estimate of when they will start, use `squeue -u myusername --start`.

Running Fuego on Hops

The Hops (SRN) cluster is comprised of 64 nodes, each with 4x 80 GB H100 NVIDIA GPUs. Compared to traditional HPC clusters like Attaway / Eclipse / Amber / etc., Hops also uses the slurm scheduler, however, each node has significantly more memory bandwidth / compute available. The following are general recommendations for running on Hops, along with example submission batch scripts for both regular MPI jobs and MPMD jobs.

Recommendations:

- Nodes are powerful but limited in number, try to fully utilize them. As a starting point, target around 15M-20M DOFs per linear system per compute node. For Fuego, DOFs are always at the mesh nodes so this is equivalent to sizing 15M-20M mesh nodes per compute node.



Note: Excluding the cost of the first step, node for node you should see order of magnitude speedups 10-20x relative to Eclipse (CTS-1) and 2.5-5x speedups relative to Amber (CTS-2). If you do not see such speedups read below.



Warning: Fuego is designed to run with 1 MPI rank per GPU, or in MPMD mode with PMR, at most 2 MPI ranks per GPU. In regular MPI submissions, do not run more than 1 task per GPU, as performance will degrade due to context switching. Use `nvidia-smi` to verify tasks are split properly across GPUs.

- Configure models on host platforms (e.g. `cee / amber`, etc) as some of the initialization occurs on host when using Hops. Since we are running with only four MPI ranks per node, this initialization cost can be significant compared to host platforms that run on many more ranks per node (e.g., Amber has 112 ranks per node).
- Fuego is partially converted, meaning some algorithms may run on host. To check if there are algorithms running on host, pass `--afgoout debug` to Fuego to write out a list of host algorithms to the log file, then in a terminal `cat fuego.log | grep "Creating host"`. Depending on the algorithms listed, they may severely degrade overall performance.



Note: As example, consider a volume host algorithm that is running on 1 node of Hops vs 1 node of Amber. Excluding the added cost of syncing fields (potentially also the LHS / RHS) to host and back to device, a rough estimate of the respective slowdown is $112 / 4$ since we run with 1 MPI rank per GPU on Hops but 112 ranks per node on Amber. If on the other hand, it is a surface algorithm running on a small portion of your domain (and does not modify LHS), you may potentially get away with running the host algorithm on Hops. A question one might have is, can we thread the legacy host algorithms to fully utilize the remaining CPU cores, the answer is threading of host algorithms requires the same Kokkos conversion that would be required to run on GPU, in which case it would be more beneficial to simply run on the GPU after conversion of the host algorithm.



Warning: All particle physics runs on the host at the moment. We do not have a timeline on when this feature will be converted to run on GPUs.

- Check your preconditioners. If you specify to use the SGS preconditioner, we will respect that irrespective of the potential performance penalties. SGS is not GPU friendly. For scalar transport problems that are diagonally dominant we recommend to start with 1-3 sweeps of the Jacobi preconditioner. If you observe linear convergence issues, switch to SGS2 preconditioner which is also GPU friendly.

- If you suspect host algorithms are hot spots in your GPU runs, email us and we will determine a priority to convert the algorithm. If no host algorithms are reported and you do not see order of magnitude speedups relative to Eclipse, email us at sierra-help@sandia.gov so we can diagnose your issue.

To submit a regular MPI job fully utilizing the four gpus on each node you can call the following script using sbatch

```
#!/bin/bash
#SBATCH --account=fyXXXXX # WC ID
#SBATCH --job-name=test
#SBATCH --partition=batch
#SBATCH --qos=normal

# at the moment only daily is installed and we are waiting on point_
→release to be installed
module load sierra/daily

num_nodes=$SLURM_JOB_NUM_NODES
task_per_node=4
ntasks=$((task_per_node * num_nodes))

launch -n $ntasks fuego -i input.i

# alternatively
# srun -n $ntasks fuego -i input.i
```

The following sbatch command will submit to two nodes, running 4 tasks split across the available gpus, for a total of 8 tasks (MPI ranks) reserving 60 minutes.

```
$ sbatch -N 2 -t 60 submit_hops.sh
```

As mentioned earlier, for regular MPI jobs, each task should bind exactly to one gpu, otherwise if multiple tasks are bound per GPU a performance penalty may be incurred due to context switching. To verify, you have the correct distribution of tasks, ssh to a compute node where your job is running and use `nvidia-smi`

```
$ squeue -u [username]
$ ssh hops[node]
$ nvidia-smi
```

For this example, for a given node, the `nvidia-smi` output should show 4 tasks running, each on their own gpu along with the memory consumed per GPU.

Similarly, to submit a MPMD job you can also use the launch wrapper

```
#!/bin/bash
#SBATCH --account=fyXXXXX # WC ID
#SBATCH --job-name=test
#SBATCH --partition=batch
#SBATCH --qos=normal

module load sierra/daily

num_nodes=$SLURM_JOB_NUM_NODES
fuego_task_per_node=4
pmr_task_per_node=4

ntasks_fuego=$((fuego_task_per_node * num_nodes))
ntasks_pmr=$((pmr_task_per_node * num_nodes))

launch -n $ntasks_fuego fuego -i fuego_input.i : -n $ntasks_pmr pmr -i_
↪pmr_input.i
```

where the following sbatch command will split tasks in a round robin fashion across available GPUs.

```
$ sbatch -N 2 --distribution=cyclic -t 60 submit_mpmc_hops.sh
```

Using `nvidia-smi`, you will see each node has 4 tasks of Fuego and 4 tasks of PMR, with one task of Fuego and one task of PMR per GPU for a total of 16 tasks across the 8 GPUs. Fuego and PMR alternate in work and so it is fine to overload each GPU in this case, though each GPU will consume more memory.

Note: The launch wrapper automatically outputs a temporary MPMD config file and passes that file to `srun` i.e.,

```
srun -n $total_tasks --multi-prog mpmc_config_file
```

where the MPMD config file specifies tasks per app i.e.,



```
0-7 fuego -i fuego_input.i
8-15 pmr -i pmr_input.i
```

With this MPMD config file, without passing `--distribution=cyclic` to `srun`, the first 8 tasks of Fuego will land on the first node and the second 8 tasks of PMR will land on the second node. This is not ideal as each GPU will need to frequently context switch between different Fuego (node 1) or PMR (node 2) tasks.



Note: PMR is fully converted to the GPU with the exception of minor initialization costs.

6.2 Diagnostic Output

Fuego is instrumented with a diagnostic output capability that can be very useful when you're debugging problems. You can tell Fuego to write additional information to the log file by enabling different “print masks”, or running various diagnostic checks. A list of common diagnostic options is summarized in the list below. For the complete list you can refer to the Fuego runtime help which is printed if you add `-h` or `--help` to your launch command (e.g. `fuego --help`).

- `--afgoout debug` - This option enables additional debug output in the Fuego log file
- `--afgoout finite_check` - Activate additional checks for NaN or Inf results from core kernels
- `--particlesout particles` - Activate debugging output related to particle transport
- `--check-syntax` - Check the syntax of the input deck (does not load the mesh)
- `--check-input` - Check the syntax of the input deck and mesh load

6.3 Troubleshooting

Sometimes a Fuego job will fail. This can be for a number of reasons, including invalid syntax in the input file, an ill-posed or non-physical problem setup, insufficient mesh resolution, computing hardware issues, or other reasons. When this happens, you will need to look for clues in the log file, the stdout stream (or slurm file), and any results output files (Exodus or Heartbeat). The following sections address some of the common failure modes you might encounter.

Invalid Input Syntax

If you use something in the input file that is not valid syntax, you will get an error in the log file. If the problem is invalid syntax, the error will be at the point in the input file where the invalid syntax was, and will suggest alternative syntax. For example, if we use an invalid unit system (using SI instead of MKS or CGS for the unit system specification)

```
Begin Fuego Region Fuego_region
```

```
# SET THE UNITS
Set unit system to SI
```

The first signal we get that something is wrong is in the stdout stream (or the slurm file if launched on an HPC), but it does not tell us what the specific error is.

```
$ aria -i demo.i
exception on all processors:
Execution terminated due to errors

*** SIERRA ABORT on P0 ***
*** check file for more information ***
```

Looking in the log file, at the end of the file we would see

```
There was 1 error encountered during parse
There were no warnings encountered during parse
Execution terminated due to errors

SIERRA execution failed during parse with the following exception:
Execution terminated due to errors
```

This does not tell us what the error is by itself either, but since it says it was a parse error we should search for the keyword “error” in the section of the log file where it repeats the input file. Then we find

```
Begin Fuego Region Fuego_region

      # SET THE UNITS
      Set unit system to SI
flow_over_cylinder_complete.i:51: Error: No matching command line found.
  ↳for 'Set unit system to SI'
flow_over_cylinder_complete.i:51: The following command lines with
  ↳matching keywords are defined:
flow_over_cylinder_complete.i:51:   Set Unit System To {Cgs|Mks}
flow_over_cylinder_complete.i:51:
```

The error message tells us which line was invalid, and shows the syntax for the best match. In this case, the model we used (SI) is invalid, and it shows us that the available options are CGS and MKS

Deprecated Syntax

Most Fuego problems use stable syntax that does not change from one release to the next, however sometimes features need to be removed for maintainability and sustainability of the codebase. This is done using a deprecation cycle to allow for time to migrate input file syntax.

Fuego uses a 6 to 12 month deprecation cycle (or longer for larger feature deprecations). When a deprecation is introduced, it will result in a warning immediately, which shows up both in the log file and in the stderr/slurm stream. For example, if you used a deprecated post-processor in Fuego version 5.10 you would see the following warning in the stderr/slurm output

```
WARNING: Deprecated feature removed in Version 5.11 detected.
```

The same warning would also show up in the log file

```
demo.i:82: Warning: Deprecated feature removed in Version 5.11 detected.
```

Deprecation version numbers are always odd-numbered (non-release). When the 5.10 version is released, the version of `sierra/daily` is set to 5.11. In this example, this means that using the deprecated post-processor will be an error if you run `sierra/daily` during the 5.11 release cycle, and in the next release (5.12) it will be an error. Pay attention to these warnings and update your inputs in a timely manner to avoid getting errors at future releases.

7 Command References

This section is a comprehensive list of command summaries recognized by the Sierra parser. In general, these summaries include the **Syntax** of how to specify the command, the **Scope** where it can be used, and a **Summary / Description** of what effect it has. User inputs are stylized as follows for convenience:

Sierra Commands

7.1 Domain

Sierra

Summary

Delimits a domain to contain the procedures.

Description

The “Begin Sierra jobid” and “End Sierra jobid” block contains the input commands for the analysis run. The jobid is an arbitrary string identifying the analysis.

begin Sierra *JobIdentifier*

Load User Plugin File *File Name* [Using Function *Function*]

Maximum Warning Count {=} *Max_messages*

Restart {=} {*auto*}

Restart Time {=} *Time*

Serialized Io Group Size {=} *Group-size*

Test Error Messages To File *File_name* And Die On First *types...*

Title

User Subroutine File {=} *File_name*

Diagnostic Stream *File_name* [*Indent-streambuf-flags1*[
↪*Indent-streambuf-flags2*]]

Enable Timer *Timer-enumeration...*

Log *Log-control-name* Every *Interval*

Print Timer Information Every *Procedure-step-interval* Steps
↪{*accumulated* | *checkpointed*}

begin Definition For Function *FunctionName*
end

begin Finite Element Model *Finite-Element-Model-Name*
end

begin Fuego Procedure *ProcedureName*
end

begin Global Constants *empty*
end

begin Output Scheduler *Label*
end

begin Property Specification For Fuego Material *MaterialName*
end

```
begin Tpetra Equation Solver Solver Name
end
```

```
end Sierra JobIdentifier
```

Line Commands

Load User Plugin File

Syntax

Load User Plugin File *File Name* [Using Function *Function*]

Summary

This line command names the source file and registration function for C++ user plugins, subroutines and functions.

Parameter	Value	Default
<i>File Name</i>	string	–

Maximum Warning Count

Syntax

Maximum Warning Count *{=}* *Max_messages*

Summary

Sets the maximum number of warnings before the execution is terminated.

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>Max_messages</i>	integer	–

Restart

Syntax

Restart *{=}* *{auto}*

Summary

Specify automatic restart file read.

Description

NOTE: This command must be placed at the Sierra scope of the input file.

Specify that the analysis should be restarted from the last common time on all restart databases for each Region in the analysis. In addition to this line command, each Region in the analysis (strictly, only the region(s) that will be

restarted) must have a restart block specifying the database to read the restart state data.

By default, use of this command will not cause output files (e.g., results, history, heartbeat, restart) to be overwritten. Instead output files will be written with the same basename and the suffix `-s0000*`. Common visualization packages are written to handle this file organization gracefully in order for the user to view all results seamlessly.

Parameter	Value	Default
<code>{=}</code>	{ = are is }	–
<code>{auto}</code>	{ auto automatic }	–

Restart Time

Syntax

Restart Time `{=}` *Time*

Summary

Specify restart file read at a specified time.

Description

NOTE: This command must be placed at the Sierra scope of the input file.

Specify the time that the analysis will be restarted. In addition to this line command, each Region in the analysis (strictly, only the region(s) that will be restarted) must have a restart block specifying the database to read the restart state data. The restart ‘time’ must be greater than zero and less than or equal to the termination time.

By default, use of this command will cause previous output files (e.g., results, history, heartbeat, restart) to be overwritten. If this command is chosen, the onus is placed on the user to ensure that previous output files are not overwritten.

Parameter	Value	Default
<code>{=}</code>	{ = are is }	–
<i>Time</i>	real	–

Serialized Io Group Size

Syntax

Serialized Io Group Size `{=}` *Group-size*

Summary

Specifies the number of processors which can concurrently perform I/O. Specifying zero disables serialization.

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>Group-size</i>	integer	–

Test Error Messages To File

Syntax

Test Error Messages To File *File_name* And Die On First *types* . .

Summary

Write an error message to the specified file and then die

Parameter	Value	Default
<i>File_name</i>	string	–
<i>types</i>	{error message warning}	–

Title

Syntax

Title

Summary

User-defined title for identifying the analysis. The title continues to the end of the line (including continuation lines)

User Subroutine File

Syntax

User Subroutine File *{=}* *File_name*

Summary

This line command is only for the script that runs the application code. It needs to know where to find the *.F file that contains all the user subroutines that are referenced in the input file. Although a C++ handler is provided, the apps do not do anything with this command, only the script that runs the app. Note that the scope of this command is domain!

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>File_name</i>	string	–

Diagnostic Stream

Syntax

Diagnostic Stream *File_name* [*Indent-streambuf-flags1* [*Indent-streambuf-flags2*]]

Summary

File path to write diagnostic messages to.

Parameter	Value	Default
<i>File_name</i>	string	–

Enable Timer

Syntax

Enable Timer *Timer-enumeration. . .*

Summary

Enables runtime performance metrics data collecting.

Description

The metrics cpu time, wall time, io count, msg count and flop count which are collected within several classes can be selectively enabled by metric class.

CPU and wall time metrics are enabled on all platforms. The remaining metrics are enabled on some platforms based on data availability.

Metrics:

cpu

Display CPU times

wall

Display wall times

Metric classes:

all

Enable all metrics

none

Disable all timers

adaptivity

Enable metrics on adaptivity

algorithm

Enable metrics on algorithms

app1

Enable app defined metrics

contact

Enable metrics on contact

domain

Enable metrics on the domain

material

Enable metrics on materials

mechanics

Enable metrics on mechanics

procedure

Enable metrics on procedures

profile1

Enable app defined profiling metrics

region

Enable metrics on regions

search

Enable metrics on searches

solver

Enable metrics on solvers

transfer

Enable metrics on user functions

Output formats:**table**

Format output as a table

xml

Format output as an XML file

Parameter	Value	Default
<i>Timer-enumeration</i>	{adaptivity algorithm all app1 app2 app3 app4 contact cpu domain elements material mechanics none procedure profile1 profile2 profile3 profile4 recovery region search seconds solver transfer wall}	–

Log**Syntax**

Log *Log-control-name* Every *Interval*

Summary

Sets the maximum number of warnings before the execution is terminated.

Parameter	Value	Default
<i>Log-control-name</i>	string	–
<i>Interval</i>	integer	–

Print Timer Information Every

Syntax

Print Timer Information Every *Procedure-step-interval* Steps {*accumulated* | *checkpointed*}

Summary

Specifies the procedure step count interval to print timer information

Parameter	Value	Default
<i>Procedure-step-interval</i>	integer	–
<i>Checkpointed</i>	{accumulated checkpointed}	–

Definition For Function

Scope

Sierra

Summary

Defines a function in terms of its type and values.

begin Definition For Function *FunctionName*

Abscissa {=} *Name...*

Abscissa Offset {=} *Abscissa_offset*

Abscissa Scale {=} *Abscissa_scale*

At Discontinuity Evaluate To {left | right}

Column Titles *Titles1 Titles2...*

Data File = *filename* [X From Column *xcol* Y From Column *ycol*]

Debug {=} {off | on}

Differentiate Expression {=} *Expr*

Evaluate Expression {=} *Expr*

Evaluate From *x0* To *x1* By *Dx*

Expression Variable: *Expr* = {*element* | *element_sym_tensor* | *element_*
tensor | *element_vector* | *face* | *global* | *nodal* | *nodal_sym_tensor* | *...*

```
↳nodal_tensor | nodal_vector} source_var_name... [ State {new | none |  
↳old} ]
```

Expression Variable: *Expr*

Field Types *Titles1 Titles2...*

Ordinate {=} *Name...*

Ordinate Offset {=} *Ordinate_offset*

Ordinate Scale {=} *Ordinate_scale*

Scale By *x*

```
Type {=} {analytic | constant | multicolumn piecewise linear |  
↳piecewise analytic | piecewise constant | piecewise linear | piecewise_  
↳multivariate | xtable}
```

X Offset {=} *X_offset*

X Scale {=} *X_scale*

Y Offset {=} *Y_offset*

Y Scale {=} *Y_scale*

```
begin Expressions empty  
end
```

```
begin Values empty  
end
```

end Definition For Function *FunctionName*

Line Commands

Abscissa

Syntax

Abscissa {=} *Name...*

Summary

Specifies a string identifier for the independent variable. Optionally specify a scale and/or offset value which transforms the abscissa values into

scaled_abscissa = scale * (abscissa + abscissa_offset).

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>Name</i>	string. . .	–

Abscissa Offset

Syntax

Abscissa Offset *{=}* *Abscissa_offset*

Summary

Alias for X OFFSET

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>Abscissa_offset</i>	real	–

Abscissa Scale

Syntax

Abscissa Scale *{=}* *Abscissa_scale*

Summary

Alias for X SCALE

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>Abscissa_scale</i>	real	–

At Discontinuity Evaluate To

Syntax

At Discontinuity Evaluate To *{left | right}*

Summary

Control the behavior of a piecewise constant function when evaluated at a discontinuity (plus or minus a small tolerance). The default behavior is to take the value to the right of the discontinuity. If “Left” is specified, the value to the left of the discontinuity is taken instead.

Parameter	Value	Default
<i>Option</i>	{left right}	–

Column Titles

Syntax

Column Titles *Titles1 Titles2. . .*

Summary

Name the columns (and also defined the expected number of columns) for Multicolumn Piecewise Linear tabular data.

Parameter	Value	Default
<i>Titles</i>	string1 string2. . .	–

Data File

Syntax

Data File = *filename* [X From Column *xcol* Y From Column *ycol*]

Summary

Function will read tabular data from an input file. Compatible with the piecewise linear function type. File must be of form like:

```
-----  
\# EXAMPLE FILE  
1.099  1191  
1.101  221  
5.9011 133.1  
-----
```

Lines headed by a # are considered comments and will be ignored. Data itself must be in tabular columns separated by whitespace or commas.

Parameter	Value	Default
<i>filename</i>	string	–

Debug

Syntax

Debug {=} {*off* | *on*}

Summary

Prints functions to the log file.

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>Option</i>	{off on}	–

Differentiate Expression

Syntax

Differentiate Expression *{=}* *Expr*

Summary

Specifies the expression of derivative of evaluation expression.

Parameter	Value	Default
<i>{=}</i>	{= are is }	–
<i>Expr</i>	(expression)	–

Evaluate Expression**Syntax**

Evaluate Expression *{=}* *Expr*

Summary

Specifies the expression to evaluate.

Description

This will greatly help with manufactured solutions, and be useful for other purposes as well. This uses the STK expression evaluator to evaluate the provided string. See the STK user manual for details about valid syntax.

```
begin definition for function pressure
  type is analytic
  evaluate expression is "x <= 0.0 ? 0.0 : (x < 0.5 ? x*200.0
    : (x < 1.0 ? (x - 0.5) *50.0 + 100.00 : 150.0))"
end definition for function pressure
```

Parameter	Value	Default
<i>{=}</i>	{= are is }	–
<i>Expr</i>	(expression)	–

Evaluate From**Syntax**

Evaluate From *x0* To *x1* By *Dx*

Summary

Specifies the range and evaluation interval.

Parameter	Value	Default
<i>x0</i>	real	–
<i>x1</i>	real	–
<i>Dx</i>	real	–

Expression Variable:

Syntax

Expression Variable: *Expr* = {*element* | *element_sym_tensor* | *element_tensor* | *element_vector* | *face* | *global* | *nodal* | *nodal_sym_tensor* | *nodal_tensor* | *nodal_vector*} *source_var_name*. . . [State {*new* | *none* | *old*}]

Summary

Specifies what the arguments of an expression correspond to. For example:

```
BEGIN DEFINITION FOR FUNCTION dx_shear
  TYPE = ANALYTIC
  EXPRESSION variable: mx    = NODAL model_coordinates(x)
  EXPRESSION variable: my    = NODAL model_coordinates(y)
  EXPRESSION variable: time = GLOBAL time
  EVALUATE EXPRESSION = "(time/{termTime})*({stretchx}*(mx -
  ↪0.0) + ((my-0.25)/0.5)*{stretchxy})"
END
```

Assuming the above expression is being evaluated on nodes the current values for x and y model coordinates would be placed into mx and my and current analysis time placed into time

Parame-ter	Value	De-fault
<i>Expr</i>	string	—
<i>VarType</i>	{element element_sym_tensor element_tensor element_vector face global nodal nodal_sym_tensor nodal_tensor nodal_vector}	—
<i>source_var_string</i>	string. . .	—

Expression Variable:

Syntax

Expression Variable: *Expr*

Summary

Specifies what the arguments of an expression exists, but does not define it correspond to. For example:

```
BEGIN DEFINITION FOR FUNCTION dx_shear
  TYPE = ANALYTIC
  EXPRESSION variable: mx
  EXPRESSION variable: my
  EXPRESSION variable: time
  EVALUATE EXPRESSION = "(time/{termTime})*({stretchx}*(mx -
```

(continues on next page)

(continued from previous page)

```
←0.0) + ((my-0.25)/0.5)*{stretchxy}) "  
END
```

Call function must determine what each variable actually is based off of the string name

Parameter	Value	Default
<i>Expr</i>	string	–

Field Types

Syntax

Field Types *Titles1 Titles2...*

Summary

The field types (GLOBAL/NODE/ELEMENT) that correspond to the column titles listed for the multicolumn data.

Parameter	Value	Default
<i>Titles</i>	string1 string2...	–

Ordinate

Syntax

Ordinate *{=} Name...*

Summary

Specifies a string identifier for the dependent variable. Optionally specify a scale and/or offset value which transforms the ordinate values into scaled_ordinate = scale * (ordinate + ordinate_offset).

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>Name</i>	string...	–

Ordinate Offset

Syntax

Ordinate Offset *{=} Ordinate_offset*

Summary

Alias for Y OFFSET

Parameter	Value	Default
{=}	{= are is }	–
<i>Ordinate_offset</i>	real	–

Ordinate Scale

Syntax

Ordinate Scale **{=}** *Ordinate_scale*

Summary

Alias for Y SCALE

Parameter	Value	Default
{=}	{= are is }	–
<i>Ordinate_scale</i>	real	–

Scale By

Syntax

Scale By *x*

Summary

Specifies a scale factor to be applied.

Parameter	Value	Default
<i>x</i>	real	–

Type

Syntax

Type **{=}** *{analytic | constant | multicolumn piecewise linear | piecewise analytic | piecewise constant | piecewise linear | piecewise multivariate | xtable}*

Summary

Specifies the type of function.

Parameter	Value	Default
{=}	{= are is }	–
<i>Type</i>	{analytic constant multicolumn piecewise linear piecewise analytic piecewise constant piecewise linear piecewise multivariate xtable }	–

X Offset

Syntax

X Offset {=} *X_offset*

Summary

Sets an offset for the x-axis

Parameter	Value	Default
{=}	{= are is }	—
<i>X_offset</i>	real	—

X Scale**Syntax**

X Scale {=} *X_scale*

Summary

Sets a scale factor for the x-axis

Parameter	Value	Default
{=}	{= are is }	—
<i>X_scale</i>	real	—

Y Offset**Syntax**

Y Offset {=} *Y_offset*

Summary

Sets an offset for the y-axis

Parameter	Value	Default
{=}	{= are is }	—
<i>Y_offset</i>	real	—

Y Scale**Syntax**

Y Scale {=} *Y_scale*

Summary

Sets a scale factor for the y-axis

Parameter	Value	Default
{=}	{= are is }	—
<i>Y_scale</i>	real	—

Values

Scope

Definition For Function

Summary

Lists the values of the function. The values should be listed one pair per line, independent variable first, with whitespace or comma as a separator.

begin Values *empty*

Xyvalues...

end Values *empty*

Line Commands

Xyvalues

Syntax

Xyvalues *Xyvalues...*

Summary

For a piecewise linear function, lists an x-y pair for the nth interpolation point.

Parameter	Value	Default
<i>Xyvalues</i>	real. . .	–

Finite Element Model

Scope

Sierra

Summary

Describes the location and type of the input stream used for defining a geometry model for the enclosing region.

begin Finite Element Model *Finite-Element-Model-Name*

Alias *DatabaseName* As *InternalName*

Component Separator Character *{=}* *Separator*

Create *{edgeset | elemset | faceset | nodeset | sideset | surface}*_
↳ *NewSurfaceName* Add *SurfaceName...*

```

Coordinate System {=} {axisymmetric | barycentric | cartesian |
↳cyclidic | cylindrical | polar | quadriplanar | skew | spherical |
↳toroidal | trilinear}

Database Name {=} StreamName

Database Type {=} {catalyst | catalyst_exodus | cgns | dof | dof_
↳exodus | exodus | exodusii | exonull | generated | genesis | null |
↳parallel_exodus | textmesh}

Decomposition Method {=} {block | cyclic | external | geom_kway | hsfc_
↳/ kway | kway_geom | linear | map | metis_sfc | random | rcb | rib |
↳variable}

Omit Assembly AssemblyList...

Omit Block BlockList...

Omit Volume VolumeList...

Time Scale Factor {=} Scale

Use Generic Names

Use Material MaterialName For VolumeList...

begin Assembly Assembly_Name
end

begin Parameters For Block Blockname
end

end Finite Element Model Finite-Element-Model-Name

```

Line Commands

Alias

Syntax

Alias *DatabaseName* As *InternalName*

Summary

Name the database entity “DatabaseName” as “InternalName”

Description

This “InternalName” may then be referenced in the data file in addition to the original name.

Parameter	Value	Default
<i>DatabaseName</i>	string	–
<i>InternalName</i>	string	–

Component Separator Character

Syntax

Component Separator Character *{=}* *Separator*

Summary

The separator is the single character used to separate the output variable basename (e.g. “stress”) from the suffices (e.g. “xx”, “yy”) when displaying the names of the individual variable components. For example, the default separator is “_”, which results in names similar to “stress_xx”, “stress_yy”, . . . “stress_zx”. To eliminate the separator, specify an empty string (“”) or NONE.

Parameter	Value	Default
<i>{=}</i>	{= is}	–
<i>Separator</i>	string	–

Create

Syntax

Create *{edgeset | elemset | faceset | nodeset | sideset | surface}* *NewSurfaceName*
Add *SurfaceName. . .*

Summary

Create a new set (node, edge, face, element, side/surface) as the union of two or more existing sets. The sets must exist in the mesh database or have been created by a previous CREATE command.

Parameter	Value	Default
<i>GroupType</i>	{edgeset elemset faceset nodeset sideset surface}	–
<i>NewSurfaceName</i>	string	–
<i>SurfaceName</i>	string. . .	–

Coordinate System

Syntax

Coordinate System *{=}* *{axisymmetric | barycentric | cartesian | cyclidic | cylindrical | polar | quadriplanar | skew | spherical | toroidal | trilinear}*

Summary

The interpretation of the geometry data stored in this database. Optional.
Defaults to Cartesian.

Parameter	Value	De- fault
<i>{=}</i>	{ = are is }	–
<i>CoordinateSystem</i>	{ axisymmetric barycentric cartesian cyclidic cylindrical polar quadri- planar skew spherical toroidal trilinear }	–

Database Name

Syntax

Database Name *{=}* *StreamName*

Summary

The base name of the database containing the output results. If the filename begins with the ‘/’ character, it is an absolute path; otherwise, the path to the current directory will be prepended to the name. If this line is omitted, then a filename will be created from the basename of the input file with a “.g” suffix appended.

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>StreamName</i>	string	–

Database Type

Syntax

Database Type *{=}* *{catalyst | catalyst_exodus | cgns | dof | dof_exodus | exodus | exodusii | exonull | generated | genesis | null | parallel_exodus | textmesh}*

Summary

The database type/format used for the mesh.

Parameter	Value	De- fault
<i>{=}</i>	{ = are is }	–
<i>DatabaseTypes</i>	{ catalyst catalyst_exodus cgns dof dof_exodus exodus exodusii exonull generated genesis null parallel_exodus textmesh }	–

Decomposition Method

Syntax

Decomposition Method *{=}* *{block | cyclic | external | geom_kway | hsfk | kway | kway_geom | linear | map | metis_sfc | random | rcb | rib | variable}*

Summary

The decomposition algorithm to be used to partition elements to each processor in a parallel run.

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>Method</i>	{block cyclic external geom_kway hsf kway kway_geom linear map metis_sfc random rcb rib variable}	–

Omit Assembly

Syntax

Omit Assembly *AssemblyList* . .

Summary

Specifies that the element blocks that are in the assemblies in *AssemblyList* will be omitted from the analysis.

Description

If an assembly is used to omit an element block, then it is illegal to refer to that element block later in the file. Any of the element blocks omitted will be removed from any assembly that contains them.

Parameter	Value	Default
<i>AssemblyList</i>	string . .	–

Omit Block

Syntax

Omit Block *BlockList* . .

Summary

Specifies that the element blocks named in the *blockList* be omitted from the analysis.

Description

If an element block is omitted, then it is illegal to refer to it later in the input file e.g an initial condition may not be specified on an omitted element block. The elements, faces, etc are never created and it is as if the omitted element blocks did not exist in the mesh file. If a surface is completely determined by the omitted element block, then it is illegal to specify boundary conditions on that surface. However, if the surface spans multiple element blocks, boundary conditions may be applied on the portion of the surface supported by the element blocks that are not omitted.

Parameter	Value	Default
<i>BlockList</i>	string. . .	–

Omit Volume

Syntax

Omit Volume *VolumeList*. . .

Summary

Specifies that the volumes named in the volumeList be omitted from the analysis.

Description

If a volume is omitted, then it is illegal to refer to it later in the input file e.g an initial condition may not be specified on an omitted volume. The elements, faces, etc are never created and it is as if the omitted volumes did not exist in the mesh file. If a surface is completely determined by the omitted volume, then it is illegal to specify boundary conditions on that surface. However, if the surface spans multiple volumes, boundary conditions may be applied on the portion of the surface supported by the volumes that are not omitted.

Parameter	Value	Default
<i>VolumeList</i>	string. . .	–

Time Scale Factor

Syntax

Time Scale Factor *{=}* *Scale*

Summary

The scale factor to be applied to the times on the mesh database. If the scale factor is 20 and the times on the mesh database are 0.1, 0.2, 0.3, then the application will see the mesh times as 2, 4, 6.

Parameter	Value	Default
<i>{=}</i>	{= is}	–
<i>Scale</i>	real	–

Use Generic Names

Syntax

Use Generic Names

Summary

If this command is present then the name of all blocks and sets in the mesh will be of the form “type_”+id. For example, an element block with id=42 will be named “block_42”; a sideset with id 314 will be named “surface_314”. If there

are any names in the mesh file, those names will be aliases for the blocks and sets. If this command is not present, then if a name is in the mesh file, it will be used as the name and the generic generated name will be an alias. This is used as a workaround in codes that do not correctly handle named blocks and sets or as a workaround in meshes which contain non-user-specified names.

Use Material

Syntax

Use Material *MaterialName* For *VolumeList*...

Summary

Associate the given volumes with the indicated material name.

Parameter	Value	Default
<i>MaterialName</i>	string	–
<i>VolumeList</i>	string...	–

Parameters For Block

Scope

Finite Element Model

Summary

Specifies analysis parameters associated with each element block.

begin Parameters For Block *Blockname*

Active For Procedure *ProcedureName* During Periods *PeriodNames...*

Bending Hourglass {*stiffness* | *viscosity*} {=} *Hgval*

Density Scale Factor {=} *densityScaleFactor*

Deposit Specific Internal Energy *Edep* [Over Time *Tdep* Starting At
Time *Tinit*]

Effective Moduli Model {=} {*elastic* | *probed* | *pronto*}

Element Numerical Formulation {=} {*new* | *old*}

Energy Iteration Tolerance {=} *Eit*

Hourglass {*stiffness* | *viscosity*} {=} *Hgval*

Hourglass {*exponent* | *transition strain*} {=} *Hgval*

Inactive For Procedure *ProcedureName* During Periods *PeriodNames...*

Include All Blocks

Inversion Aversion Exponent *{=}* *ia_exponent*

Inversion Aversion Stiffness *{=}* *ia_stiffness*

Inversion Aversion Transition Jacobian *{=}* *transition_jacobian*

Linear Bulk Viscosity *{=}* *Lbv*

Material *MatName*

Material = *MatName*

Max Energy Iterations *{=}* *Mei*

Membrane Hourglass *{stiffness | viscosity}* *{=}* *Hgval*

Minimum Effective Dilatational Moduli Ratio *{=}* *minEffectiveModuliRatio*

Minimum Effective Shear Moduli Ratio *{=}* *minEffectiveModuliRatio*

Model *{=}* *ModelName*

Nonlocal Regularization Kmeans Cell Size *{=}* *kmeans_cell_size*

Nonlocal Regularization Kmeans Maximum Iterations *{=}* *kmeans_maximum_*
iterations

Nonlocal Regularization Kmeans Tolerance *{=}* *kmeans_tolerance*

Nonlocal Regularization On *stateVariableName* With Length Scale *{=}* *lengthScale* [And Staggering]

Nonlocal Regularization Partitioning Scheme *{=}* *{kmeans | metis | zoltan_hypergraph | zoltan_rcb | zoltan_rib}*

Phase *PhaseLabel* *{=}* *MaterialName*

Quadratic Bulk Viscosity *{=}* *Qbv*

Remove Block *{=}* *ExcludeBlockList...*

Section **{=}** *SectionName*

Solid Mechanics Use Model *ModelName*

Transverse Shear Hourglass **{stiffness | viscosity}** **{=}** *Hgval*

end Parameters For Block *Blockname*

Line Commands

Active For Procedure

Syntax

Active For Procedure *ProcedureName* During Periods *PeriodNames...*

Summary

Lists the solution periods during which the given BC, solver, preconditioner, etc. is active. Multiple uses of this line command within a single block will have a cumulative affect.

Parameter	Value	Default
<i>ProcedureName</i>	string	—
<i>PeriodNames</i>	string. . .	—

Bending Hourglass

Syntax

Bending Hourglass **{stiffness | viscosity}** **{=}** *Hgval*

Summary

Supplies the hourglass stiffness and viscosity parameters for bending deformation in a shell element block.

Parameter	Value	Default
<i>Option</i>	{ stiffness viscosity }	—
{=}	{ = are is }	—
<i>Hgval</i>	real	—

Density Scale Factor

Syntax

Density Scale Factor **{=}** *densityScaleFactor*

Summary

Specifies a scale factor to apply to the density defined in the material. This value must be greater than zero. The default is 1.0 (no scaling).

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>densityScaleFactor</i>	real	–

Deposit Specific Internal Energy

Syntax

Deposit Specific Internal Energy *Edep* [Over Time *Tdep* Starting At Time *Tinit*]

Summary

Defines the amount of specific (per unit mass) internal energy to be deposited in the material. The energy is deposited over time tdep, beginning at time tinit. The optional parameters tdep and tinit both default to zero, so the energy will be deposited instantaneously at time zero if they are not specified. The deposition is uniform in space, so each element in the block has the same amount edep added to its specific internal energy.

Parameter	Value	Default
<i>Edep</i>	real	–

Effective Moduli Model

Syntax

Effective Moduli Model *{=}* { *elastic* | *probed* | *pronto* }

Summary

Specifies the method used to determine the effective moduli. This choice can have a significant effect on the resulting hourglassing behavior. The models are:
* elastic: use the initial elastic moduli * pronto: use the old PRONTO method for computing elastic moduli this approach is straight out of PRONTO, PRESTO's predecessor. This is a bounded tangent method. * probe: Use a pronto-like method, but pass in a an artificial probe strain rate rather than the actual strain.

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>Option</i>	{ elastic probed pronto }	–

Element Numerical Formulation

Syntax

Element Numerical Formulation *{=}* { *new* | *old* }

Summary

Specifies which element numerical formulation to use for this block.

Parameter	Value	Default
<i>{=}</i>	{ = are is }	—
<i>Option</i>	{ new old }	—

Energy Iteration Tolerance

Syntax

Energy Iteration Tolerance *{=}* *Eit*

Summary

Specifies the tolerance criteria for exiting the iterative solve of the implicit internal energy update equation. Applicable when using EOS material models with extracted energy updates.

Parameter	Value	Default
<i>{=}</i>	{ = are is }	—
<i>Eit</i>	real	—

Hourglass

Syntax

Hourglass *{stiffness | viscosity}* *{=}* *Hgval*

Summary

Supplies the hourglass stiffness and viscosity parameters for this element block.

Parameter	Value	Default
<i>Option</i>	{ stiffness viscosity }	—
<i>{=}</i>	{ = are is }	—
<i>Hgval</i>	real	—

Hourglass

Syntax

Hourglass *{exponent | transition strain}* *{=}* *Hgval*

Summary

Supplies the hourglass stiffness and viscosity parameters for this element block.

Parameter	Value	Default
<i>Option</i>	{ exponent transition strain }	—
<i>{=}</i>	{ = are is }	—
<i>Hgval</i>	real	—

Inactive For Procedure

Syntax

Inactive For Procedure *ProcedureName* During Periods *PeriodNames* . .

Summary

Lists the solution periods during which the given BC, solver, preconditioner, etc. is inactive. Multiple uses of this line command within a single block will have a cumulative affect.

Parameter	Value	Default
<i>ProcedureName</i>	string	–
<i>PeriodNames</i>	string. . .	–

Include All Blocks

Syntax

Include All Blocks

Summary

Use this parameters definition for all blocks.

When using this option within the FINITE ELEMENT MODEL command block the PARAMETERS FOR BLOCK will not use a Blockname.

Inversion Aversion Exponent

Syntax

Inversion Aversion Exponent *{=}* *ia_exponent*

Summary

Sets the exponent used to compute the smooth approximate nodal jacobian ratio. A higher exponent results in a more-accurate approximation to the ratio. This is only active for uniform gradient elements. Default = 5.

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>ia_exponent</i>	integer	5

Inversion Aversion Stiffness

Syntax

Inversion Aversion Stiffness *{=}* *ia_stiffness*

Summary

Sets a stiffness parameter for the inversion aversion penalty. This is only active for uniform gradient elements. Default = 1.0e5.

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>ia_stiffness</i>	real	1.e5

Inversion Aversion Transition Jacobian

Syntax

Inversion Aversion Transition Jacobian *{=}* *transition_jacobian*

Summary

Sets the critical relative nodal Jacobian ratio for inversion aversion. If this value is nonzero, an additional recoverable energy term is added which penalizes further element distortion. This energy is only active for uniform gradient elements.

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>transition_jacobian</i>	real	0

Linear Bulk Viscosity

Syntax

Linear Bulk Viscosity *{=}* *Lbv*

Summary

Supplies the linear coefficient for the bulk viscosity computations.

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>Lbv</i>	real	–

Material

Syntax

Material *MatName*

Summary

Associates this element block with its material properties.

Parameter	Value	Default
<i>MatName</i>	string	–

Material =

Syntax

Material = *MatName*

Summary

Associates this element block with its material properties.

Parameter	Value	Default
<i>MatName</i>	string	–

Max Energy Iterations

Syntax

Max Energy Iterations {=} *Mei*

Summary

Specifies the maximum number of iterations to take in solving the implicit internal energy update equation. Applicable when using EOS material models with extracted energy updates.

Parameter	Value	Default
{=}	{ = are is }	–
<i>Mei</i>	integer	–

Membrane Hourglass

Syntax

Membrane Hourglass {*stiffness* | *viscosity*} {=} *Hgval*

Summary

Supplies the hourglass stiffness and viscosity parameters for membrane deformation in a shell or membrane element block.

Parameter	Value	Default
<i>Option</i>	{ stiffness viscosity }	–
{=}	{ = are is }	–
<i>Hgval</i>	real	–

Minimum Effective Dilatational Moduli Ratio

Syntax

Minimum Effective Dilatational Moduli Ratio {=} *minEffectiveModuliRatio*

Summary

Specifies a minimum effective DILATATIONAL moduli ratio. This value keeps the effective moduli from dropping below $\text{minEffectiveModuliRatio} \times \text{ElasticModulus}$. This can aid in keeping the corresponding time step and bulk viscosity terms dropping to zero

Parameter	Value	Default
{=}	{ = are is }	–
<i>minEffectiveModuliRatio</i>	real	–

Minimum Effective Shear Moduli Ratio

Syntax

Minimum Effective Shear Moduli Ratio **{=}** *minEffectiveModuliRatio*

Summary

Specifies a minimum effective SHEAR moduli ratio. This value keeps the effective moduli from dropping below minEffectiveModuliRatio*ElasticModulus. This can aid in keeping the corresponding hourglass stiffness terms dropping to zero

Parameter	Value	Default
{=}	{ = are is }	–
<i>minEffectiveModuliRatio</i>	real	–

Model

Syntax

Model **{=}** *ModelName*

Summary

Associates a solid mechanics material model with this element block. The material parameters for this block are specified in the material denoted by the MATERIAL command.

Parameter	Value	Default
{=}	{ = are is }	–
<i>ModelName</i>	string	–

Nonlocal Regularization Kmeans Cell Size

Syntax

Nonlocal Regularization Kmeans Cell Size **{=}** *kmeans_cell_size*

Summary

This line command specifies the cell size used to construct the background grid for the computation of the centroidal Voronoi tessellation for the Kmeans partitioning scheme.

Parameter	Value	Default
{=}	{ = are is }	–
<i>kmeans_cell_size</i>	real	–

Nonlocal Regularization Kmeans Maximum Iterations

Syntax

Nonlocal Regularization Kmeans Maximum Iterations *{=}*
kmeans_maximum_iterations

Summary

This line command specifies the maximum number of iterations to perform for Lloyd's algorithm for the computation of the centroidal Voronoi tessellation for the Kmeans partitioning scheme.

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>kmeans_maximum_iterations</i>	integer	–

Nonlocal Regularization Kmeans Tolerance

Syntax

Nonlocal Regularization Kmeans Tolerance *{=}* *kmeans_tolerance*

Summary

This line command specifies the relative tolerance for Lloyd's algorithm. Iterations continue until the maximum number is reached or the L2 norm of a vector of all the center steps is less or equal than the tolerance times the cell size of the background grid.

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>kmeans_tolerance</i>	real	–

Nonlocal Regularization On

Syntax

Nonlocal Regularization On *stateVariableName* With Length Scale *{=}*
lengthScale [And Staggering]

Summary

This line command will cause the mesh to be partitioned into sub domains where each sub domain volume is on the order of $lengthScale^3$ and regularizes the governing PDE by averaging the material state variable *stateVariableName* over the sub domain.

Parameter	Value	Default
<i>stateVariableName</i>	string	–
<i>{=}</i>	{ = are is }	–
<i>lengthScale</i>	real	–

Nonlocal Regularization Partitioning Scheme

Syntax

Nonlocal Regularization Partitioning Scheme *{=}* *{kmeans | metis | zoltan_hypergraph | zoltan_rcb | zoltan_rib}*

Summary

This line command specifies the type of partitioning algorithm used to perform the domain decomposition for the nonlocal regularization method

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>PartitioningScheme</i>	{ kmeans metis zoltan_hypergraph zoltan_rcb zoltan_rib }	–

Phase

Syntax

Phase *PhaseLabel* *{=}* *MaterialName*

Summary

Associate phase *PhaseLabel* with material *Material_Name* on this block.

Parameter	Value	Default
<i>PhaseLabel</i>	string	–
<i>{=}</i>	{ = are is }	–
<i>MaterialName</i>	string	–

Quadratic Bulk Viscosity

Syntax

Quadratic Bulk Viscosity *{=}* *Qbv*

Summary

Supplies the quadratic coefficient for the bulk viscosity computations.

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>Qbv</i>	real	–

Remove Block

Syntax

Remove Block *{=}* *ExcludeBlockList. . .*

Summary

List of blocks to exclude.

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>ExcludeBlockList</i>	string. . .	–

Section

Syntax

Section *{=}* *SectionName*

Summary

Specifies the section to use for this element block.

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>SectionName</i>	string	–

Solid Mechanics Use Model

Syntax

Solid Mechanics Use Model *ModelName*

Summary

Associates a solid mechanics material model with this element block. The material parameters for this block are specified in the material denoted by the MATERIAL command.

Parameter	Value	Default
<i>ModelName</i>	string	–

Transverse Shear Hourglass

Syntax

Transverse Shear Hourglass *{stiffness | viscosity} {=} Hgval*

Summary

Supplies the hourglass stiffness and viscosity parameters for transverse shear deformation in a shell element block.

Parameter	Value	Default
<i>Option</i>	{ stiffness viscosity }	–
<i>{=}</i>	{ = are is }	–
<i>Hgval</i>	real	–

Global Constants

Scope

Sierra

Summary

Set of universal constants for a simulation.

begin Global Constants *empty*

Faradays Constant {=} *Faraday*

Gravity Vector {=} *Gravity1 Gravity2 Gravity3*

Ideal Gas Constant {=} *Sigma*

K-E Turbulence Model Parameter *Param* {=} *Value*

K-W Turbulence Model Parameter *Param* {=} *Value*

Les Turbulence Model Parameter *Param* {=} *Value*

Light Speed {=} *LightSpeed*

Planck Constant {=} *PlanckConstant*

Stefan Boltzmann Constant {=} *Sigma*

Turbulence Model *Param* Number {=} *Value*

end Global Constants *empty*

Line Commands

Faradays Constant

Syntax

Faradays Constant {=} *Faraday*

Summary

Faraday's Constant. **NOTE:** Another Faraday's constant value can be specified while using certain code capabilities. This global constants value will be discarded for any other specified Faraday's constant values.

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>Faraday</i>	real	–

Gravity Vector

Syntax

Gravity Vector *{=}* *Gravity1 Gravity2 Gravity3*

Summary

Gravity constant in vector form, acceleration components.

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>Gravity</i>	real1 real2 real3	–

Ideal Gas Constant

Syntax

Ideal Gas Constant *{=}* *Sigma*

Summary

Ideal gas constant. **NOTE:** Another ideal gas constant value can be specified while using certain code capabilities. This global constants value will be discarded for any other specified ideal gas constant values.

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>Sigma</i>	real	–

K-E Turbulence Model Parameter

Syntax

K-E Turbulence Model Parameter *Param* *{=}* *Value*

Summary

$k - \epsilon$ RANS turbulence model parameters.

Parameter	Value	Default
<i>Param</i>	string	–
<i>{=}</i>	{= are is}	–
<i>Value</i>	real	–

K-W Turbulence Model Parameter

Syntax

K-W Turbulence Model Parameter *Param* {=} *Value*

Summary

$k - \omega$ RANS turbulence model parameters.

Parameter	Value	Default
<i>Param</i>	string	–
{=}	{= are is}	–
<i>Value</i>	real	–

Les Turbulence Model Parameter

Syntax

Les Turbulence Model Parameter *Param* {=} *Value*

Summary

LES turbulence model parameters.

Parameter	Value	Default
<i>Param</i>	string	–
{=}	{= are is}	–
<i>Value</i>	real	–

Light Speed

Syntax

Light Speed {=} *LightSpeed*

Summary

Speed of Light. Depending on the units involved in the specific problem by the user, this value will differ.

Parameter	Value	Default
{=}	{= are is}	–
<i>LightSpeed</i>	real	–

Planck Constant

Syntax

Planck Constant {=} *PlanckConstant*

Summary

Planck Constant. Depending on the units involved in the specific problem by the user, this value will differ.

Parameter	Value	Default
<i>{=}</i>	{= are is }	–
<i>PlanckConstant</i>	real	–

Stefan Boltzmann Constant

Syntax

Stefan Boltzmann Constant *{=}* *Sigma*

Summary

Stefan-Boltzmann constant. Depending on the units involved in the specific problem by the user, this value will differ.

Parameter	Value	Default
<i>{=}</i>	{= are is }	–
<i>Sigma</i>	real	–

Turbulence Model

Syntax

Turbulence Model *Param* Number *{=}* *Value*

Summary

Turbulence model Schmidt and Prandtl numbers

Parameter	Value	Default
<i>Param</i>	string	–
<i>{=}</i>	{= are is }	–
<i>Value</i>	real	–

Output Scheduler

Scope

Sierra

Summary

Defines an output scheduler at the domain level which can be used by one or more output blocks (restart, results, heartbeat, history> at region scope. If used by multiple output blocks, their output will be synchronized.

begin Output Scheduler *Label*

Additional Steps *{=}* *List_of_steps...*

Additional Times *{=}* *List_of_times...*

At Step *n* {increment | interval} {=} *m*

At Time *Dt1* {increment | interval} {=} *Dt2*

Output On Signal {=} {sigabrt | sigalrm | sigfpe | sighup | sigill |
↪sigint | sigkill | sigpipe | sigquit | sigsegv | sigterm | sigusr1 |
↪sigusr2}

Start Time {=} *Start_time*

Synchronize Output

Termination Time {=} *Final_time*

Timestep Adjustment Interval {=} *Nsteps*

Use Output Scheduler *Timer_name*

end Output Scheduler *Label*

Line Commands

Additional Steps

Syntax

Additional Steps {=} *List_of_steps* . .

Summary

Additional simulation steps when output should occur.

Parameter	Value	Default
{=}	{= are is}	—
<i>List_of_steps</i>	integer. . .	—

Additional Times

Syntax

Additional Times {=} *List_of_times* . .

Summary

Additional simulation times when output should occur.

Parameter	Value	Default
<i>{=}</i>	{ = are is }	—
<i>List_of_times</i>	real. . .	—

At Step

Syntax

At Step *n* {*increment* | *interval*} {*=*} *m*

Summary

Specify an output interval in terms of the internal iteration step count. The first step specifies the step count at the beginning of this interval and the second step specifies the output frequency to be used within this interval.

Parameter	Value	Default
<i>n</i>	integer	—
<i>Option</i>	{ increment interval }	—
<i>{=}</i>	{ = are is }	—
<i>m</i>	integer	—

At Time

Syntax

At Time *Dt1* {*increment* | *interval*} {*=*} *Dt2*

Summary

Specify an output interval in terms of the internal simulation time. The first time specifies the time at the beginning of this time interval and the second time specifies the output frequency to be used within this interval.

Parameter	Value	Default
<i>Dt1</i>	real	—
<i>Option</i>	{ increment interval }	—
<i>{=}</i>	{ = are is }	—
<i>Dt2</i>	real	—

Output On Signal

Syntax

Output On Signal {*=*} {*sigabrt* | *sigalrm* | *sigfpe* | *sigghup* | *sigill* | *sigint* | *sigkill* | *sigpipe* | *sigquit* | *sigsegv* | *sigterm* | *sigusr1* | *sigusr2*}

Summary

When the specified signal is raised, the output stream associated with this block will be output.

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>Signals</i>	{sigabrt sigalrm sigfpe sighup sigill sigint sigkill sigpipe sigquit sigsegv sigterm sigusr1 sigusr2}	–

Start Time

Syntax

Start Time *{=}* *Start_time*

Summary

Specify the time to start outputting results from this output request block. This time overrides all ‘at time’ and ‘at step’ specifications.

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>Start_time</i>	real	–

Synchronize Output

Syntax

Synchronize Output

Summary

In an analysis with multiple regions, it is sometimes desirable to synchronize the output of results data between the regions. This can be done by adding the SYNCHRONIZE OUTPUT command line to the results output block. If a results block has this set, then it will write output whenever a previous region writes output. The ordering of regions is based on the order in the input file, algorithmic considerations, or by solution control specifications.

Although the USE OUTPUT SCHEDULER command line can also synchronize output between regions, the SYNCHRONIZE OUTPUT command line will synchronize the output with regions where the output frequency is not under the direct control of the Sierra IO system. Examples of this are typically coupled applications where one or more of the codes are not Sierra-based applications such as Alegra and CTH. A results block with SYNCHRONIZE OUTPUT specified will also synchronize its output with the output of the external code.

The SYNCHRONIZE OUTPUT command can be used with other output scheduling commands such as time-based or step-based output specifications.

Termination Time

Syntax

Termination Time *{=}* *Final_time*

Summary

Specify the time to stop outputting results from this output request block.

Parameter	Value	Default
<i>{=}</i>	{= are is }	–
<i>Final_time</i>	real	–

Timestep Adjustment Interval

Syntax

Timestep Adjustment Interval *{=}* *Nsteps*

Summary

Specify the number of steps to ‘look ahead’ and adjust the timestep to ensure that the specified output times or simulation end time will be hit ‘exactly’.

Parameter	Value	Default
<i>{=}</i>	{= are is }	–
<i>Nsteps</i>	integer	–

Use Output Scheduler

Syntax

Use Output Scheduler *Timer_name*

Summary

Associates a predefined output scheduler with this output block (results, restart, heartbeat, or history).

Parameter	Value	Default
<i>Timer_name</i>	string	–

Property Specification For Fuego Material

Scope

Sierra

Summary

Contains the commands to describe the property evaluation scheme for Fuego materials.

begin Property Specification For Fuego Material *MaterialName*

`{absorption | absorption_coefficient | density | density_pressure_`
`↳derivative | emissivity | enthalpy | heat_production_rate | interface_`
`↳absorption | rad_source | scattering_coefficient | specific_heat |`
`↳surface_tension | thermal_conductivity | viscosity} {=} Value`

Bulk Chemistry Description {=} Value

Cantera Xml File {=} FileName

Chemkin Linking File {=} FileName

Chemkin Transport File {=} FileName

Datum Pressure {=} Value

Do Not Clip Mixture Fraction Variables

Function For `{absorption | absorption_coefficient | density | density_`
`↳pressure_derivative | emissivity | enthalpy | heat_production_rate |`
`↳interface_absorption | rad_source | scattering_coefficient | specific_`
`↳heat | surface_tension | thermal_conductivity | viscosity} {=} FuncName`

Function For Mass_Diffusivity SpeciesName {=} FuncName

Function For Species_Density SpeciesName {=} FuncName

Function For Species_Emissivity SpeciesName {=} FuncName

Function For Species_Enthalpy SpeciesName {=} FuncName

Function For Species_Production_Rate SpeciesName {=} FuncName

Function For Species_Specific_Heat SpeciesName {=} FuncName

Function For Species_Thermal_Conductivity SpeciesName {=} FuncName

Input Variables For SubName {=} Variable_list...

Mass_Diffusivity SpeciesName {=} Value

Molecular_Weight SpeciesName {=} Value

Prandtl_Number {=} Value

Real Data For SubName {=} Parameter_list...

```

Reference Variable {=} Value

Reference Mass_Fraction SpeciesName {=} Value

Reference Mole_Fraction SpeciesName {=} Value

Reference Progress_Variable ProgressVariableName [ With Lower Bound_
↳LowerBound And Upper Bound UpperBound ]

Set Schmidt Number For Progress Variable ProgressVariableName {=} Value

Set Thermophoresis Coefficient For Progress Variable_
↳ProgressVariableName {=} Value

Subroutine For PropertyName {=} SubName

Schmidt_Number {=} Value

Species_Density SpeciesName {=} Value

Species_Emissivity SpeciesName {=} Value

Species_Enthalpy SpeciesName {=} Value

Species_Production_Rate SpeciesName {=} Value

Species_Specific_Heat SpeciesName {=} Value

Species_Thermal_Conductivity SpeciesName {=} Value

Turbulent Mixing Model {=} {none | parente} [ModelArgs]...

Use Consistently Volume Averaged Properties [ With Postsmoothing_
↳SmoothingIterations ]

Vof Gas Material {=} GasMaterial

Vof Liquid Material {=} LiquidMaterial

begin Chemistry Description ChemistryDescName
end

begin Ode Solver Parameters SolverName
end

```

```
begin Tabular Property Library LibNameTag
end
```

```
end Property Specification For Fuego Material MaterialName
```

Line Commands

Properties

Syntax

Properties *{absorption | absorption_coefficient | density | density_pressure_derivative | emissivity | enthalpy | heat_production_rate | interface_absorption | rad_source | scattering_coefficient | specific_heat | surface_tension | thermal_conductivity | viscosity} [=] Value*

Summary

Specify a constant or string function value for the property, in consistent units.
For example, viscosity = 1.0e-3

Description

When using a string function, the function may depend on pressure, temperature, mass fraction, or most other properties. When using mass fractions in the string function, use the species name as a suffix, as in

```
Density = "300/temperature + mass_fraction_n2"
```

The function string must be enclosed in quotes if it has spaces or commas. The variable names are not case-sensitive.

Parameter	Value	Default
<i>Properties</i>	{absorption absorption_coefficient density density_pressure_derivative emissivity enthalpy heat_production_rate interface_absorption rad_source scattering_coefficient specific_heat surface_tension thermal_conductivity viscosity}	—
<i>{=}</i>	{= are is}	—
<i>Value</i>	“string”	—

Bulk Chemistry Description

Syntax

Bulk Chemistry Description *{=}* *Value*

Summary

Chemistry description to apply to the bulk region of the material being defined

Description

When multiple chemistry descriptions are included in a material they must apply to different portions of the material i.e. bulk and surface. A bulk description is mandatory in this case and this variable is used to define which description should be referenced. This is only necessary when there are multiple chemistry descriptions for a material.

Parameter	Value	Default
<i>{=}</i>	{= are is }	–
<i>Value</i>	string	–

Cantera Xml File

Syntax

Cantera Xml File *{=}* *FileName*

Summary

Name of the Cantera XML file

Description

Cantera XML files can be generated from CHEMKIN input files or from *.cti files using Cantera's preprocessing utilities. See Chapter 20 for instructions on how to create a Cantera XML file.

Parameter	Value	Default
<i>{=}</i>	{= are is }	–
<i>FileName</i>	string	–

Chemkin Linking File

Syntax

Chemkin Linking File *{=}* *FileName*

Summary

specify the file name for the CHEMKIN linking file, in ASCII format

Description

NOTE: CHEMKIN is no longer supported for property evaluation. Please use Cantera instead.

Parameter	Value	Default
<i>{=}</i>	{= are is }	–
<i>FileName</i>	string	–

Chemkin Transport File

Syntax

Chemkin Transport File *{=}* *FileName*

Summary

specify the file name for the CHEMKIN transport linking file, in ASCII format

Description

NOTE: CHEMKIN is no longer supported for property evaluation. Please use Cantera instead.

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>FileName</i>	string	–

Datum Pressure

Syntax

Datum Pressure *{=}* *Value*

Summary

Specify the datum value for pressure

Description

When running with the acoustic compressibility model, the output pressure can have an arbitrary datum value. If the datum is set equal to the reference (ambient) value, then pressure will be a “gauge” pressure. If the datum is set to zero, then an absolute pressure will result. This is ignored for incompressible flow, where the thermodynamic pressure is always equal to the reference. The datum defaults to the reference pressure for compressible flow.

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>Value</i>	real	–

Do Not Clip Mixture Fraction Variables

Syntax

Do Not Clip Mixture Fraction Variables

Summary

Turns off clipping for mixture fraction EOS input variables. EOS may still clip the variables

Description

Fuego does not clip transported variables, relying on the equation to handle unrealizable values of its input parameters

Function For

Syntax

Function For *{absorption | absorption_coefficient | density | density_pressure_derivative | emissivity | enthalpy | heat_production_rate | interface_absorption | rad_source | scattering_coefficient | specific_heat | surface_tension | thermal_conductivity | viscosity} [=] FuncName*

Summary

Specify a function to use for evaluation of this property.

Description

When specifying a function, you should also specify the name of the independent variable (abscissa). If no abscissa is specified, the temperature will be used.

Parameter	Value	Default
<i>Properties</i>	{absorption absorption_coefficient density density_pressure_derivative emissivity enthalpy heat_production_rate interface_absorption rad_source scattering_coefficient specific_heat surface_tension thermal_conductivity viscosity}	—
<i>{=}</i>	{= are is}	—
<i>FuncName</i>	string	—

Function For Mass_Diffusivity

Syntax

Function For Mass_Diffusivity *SpeciesName [=] FuncName*

Summary

Specify a function to use for evaluating the molecular mass diffusivity for the given species

Description

For the textit{SpeciesName} parameter, you can either specify the actual species name (O2, N2, CH4, etc.) or the word DEFAULT for a global function applied to all unlisted species. Any species-specific functions, if present, will override the DEFAULT function.

If this is a turbulent simulation with species transport, then the same mass diffusivity function must be used for all species. You may optionally supply just a Schmidt number.

Parameter	Value	Default
<i>SpeciesName</i>	string	–
<i>{=}</i>	{= are is}	–
<i>FuncName</i>	string	–

Function For Species_Density

Syntax

Function For Species_Density *SpeciesName* *{=}* *FuncName*

Summary

Specify a function to use for evaluating the species enthalpy for the given species

Description

For the textit{SpeciesName} parameter, you can either specify the actual species name (O2, N2, CH4, etc.) or the word DEFAULT for a global function applied to all unlisted species. Any species-specific functions, if present, will override the DEFAULT function.

This command is only applicable when not using an external third-party library (e.g. Cantera) for property evaluation.

Parameter	Value	Default
<i>SpeciesName</i>	string	–
<i>{=}</i>	{= are is}	–
<i>FuncName</i>	string	–

Function For Species_Emissivity

Syntax

Function For Species_Emissivity *SpeciesName* *{=}* *FuncName*

Summary

Specify a function to use for evaluating the species emissivity

Description

For the textit{SpeciesName} parameter, you can either specify the actual species name (O2, N2, CH4, etc.) or the word DEFAULT for a global function applied to all unlisted species. Any species-specific functions, if present, will override the DEFAULT function.

Parameter	Value	Default
<i>SpeciesName</i>	string	–
<i>{=}</i>	{= are is}	–
<i>FuncName</i>	string	–

Function For Species_Enthalpy

Syntax

Function For Species_Enthalpy *SpeciesName* {=} *FuncName*

Summary

Specify a function to use for evaluating the species enthalpy for the given species

Description

For the textit{SpeciesName} parameter, you can either specify the actual species name (O2, N2, CH4, etc.) or the word DEFAULT for a global function applied to all unlisted species. Any species-specific functions, if present, will override the DEFAULT function.

This command is only applicable when not using an external third-party library (e.g. Cantera) for property evaluation.

Parameter	Value	Default
<i>SpeciesName</i>	string	—
{=}	{= are is}	—
<i>FuncName</i>	string	—

Function For Species_Production_Rate

Syntax

Function For Species_Production_Rate *SpeciesName* {=} *FuncName*

Summary

warning{This command is deprecated and will be removed in the 5.18 release.
It has not had an effect for several releases.}

Parameter	Value	Default
<i>SpeciesName</i>	string	—
{=}	{= are is}	—
<i>FuncName</i>	string	—

Function For Species_Specific_Heat

Syntax

Function For Species_Specific_Heat *SpeciesName* {=} *FuncName*

Summary

Specify a function to use for evaluating the species specific heat

Description

For the textit{SpeciesName} parameter, you can either specify the actual species name (O2, N2, CH4, etc.) or the word DEFAULT for a global function applied

to all unlisted species. Any species-specific functions, if present, will override the DEFAULT function.

Parameter	Value	Default
<i>SpeciesName</i>	string	—
<i>{=}</i>	{= are is}	—
<i>FuncName</i>	string	—

Function For Species_Thermal_Conductivity

Syntax

Function For Species_Thermal_Conductivity *SpeciesName* *{=}* *FuncName*

Summary

Specify a function to use for evaluating the species thermal conductivity

Description

For the textit{SpeciesName} parameter, you can either specify the actual species name (O2, N2, CH4, etc.) or the word DEFAULT for a global function applied to all unlisted species. Any species-specific functions, if present, will override the DEFAULT function.

Parameter	Value	Default
<i>SpeciesName</i>	string	—
<i>{=}</i>	{= are is}	—
<i>FuncName</i>	string	—

Input Variables For

Syntax

Input Variables For *SubName* *{=}* *Variable_list* . . .

Summary

Requests that the given variables be passed to the specified user subroutine as inputs.

Parameter	Value	Default
<i>SubName</i>	string	—
<i>{=}</i>	{= are is}	—
<i>Variable_list</i>	string. . .	—

Mass_Diffusivity

Syntax

Mass_Diffusivity *SpeciesName* *{=}* *Value*

Summary

Specify the molecular mass diffusivity for the given species

Description

For the *SpeciesName* parameter, you can either specify the actual species name (O2, N2, CH4, etc.) or the word DEFAULT for a global value applied to all unlisted species. Any species-specific values, if present, will override the DEFAULT value.

If this is a turbulent simulation with species transport, then the mass diffusivity for all species must be identical. You may optionally supply just a Schmidt number.

When using a string function, the function may depend on pressure, temperature, mass fraction, or most other properties. When using mass fractions in the string function, use the species name as a suffix, as in

```
mass_diffusivity N2 = "0.01*mass_fraction_n2"
```

The function string must be enclosed in quotes if it has spaces or commas. The variable names are not case-sensitive.

Parameter	Value	Default
<i>SpeciesName</i>	string	—
<i>{=}</i>	{= are is}	—
<i>Value</i>	“string”	—

Molecular_Weight

Syntax

Molecular_Weight *SpeciesName* *{=}* *Value*

Summary

Specify the molecular weight for the given species

Description

For the textit{SpeciesName} parameter, you can either specify the actual species name (O2, N2, CH4, etc.) or the word DEFAULT for a global value applied to all unlisted species. Any species-specific values, if present, will override the DEFAULT value.

This command is only applicable when not using a material model (e.g. Cantera, Tabular, etc.).

Parameter	Value	Default
<i>SpeciesName</i>	string	—
<i>{=}</i>	{= are is}	—
<i>Value</i>	real	—

Prandtl_Number

Syntax

Prandtl_Number *{=}* *Value*

Summary

Specify the Prandtl number ($Pr = \mu \cdot c_p / k$) to calculate thermal conductivity as a function of viscosity and specific heat.

Description

If transporting both an energy equation and a species equation in turbulent flow, it is mandatory that this be specified and that it be identical to the Schmidt number, to enforce unity Lewis number ($Le = Sc / Pr$). In all other cases, this is optional.

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>Value</i>	real	–

Real Data For

Syntax

Real Data For *SubName* *{=}* *Parameter_list* . .

Summary

Specifies a list of parameters that should be passed to the specified user subroutine.

Parameter	Value	Default
<i>SubName</i>	string	–
<i>{=}</i>	{ = are is }	–
<i>Parameter_list</i>	real. . .	–

Reference

Syntax

Reference *Variable* *{=}* *Value*

Summary

Specify the reference value for the given variable

Description

A reference value must be specified for all input variables required by the particular material model for property evaluation. In the typical case where Cantera is used for property evaluation, then reference properties must be specified for “temperature”, “pressure”, and “mass_fraction”. Other material models may have different dependencies.

These values will be used for reporting property values at the reference state during startup, and for property evaluation at run-time when a transport equation is not being solved for the required variable or when a reference state is needed for calculating “ambient” properties for entrainment purposes.

Parameter	Value	Default
<i>Variable</i>	string	–
<i>{=}</i>	{= are is }	–
<i>Value</i>	real	–

Reference Mass_Fraction

Syntax

Reference Mass_Fraction *SpeciesName* *{=}* *Value*

Summary

Specify the reference mass fraction for the given species

Description

A reference value must be specified for all input variables required by the particular material model for property evaluation. In the typical case where Cantera is used for property evaluation, then reference properties must be specified for “temperature”, “pressure”, and “mass_fraction”. Other material models may have different dependencies.

These values will be used for reporting property values at the reference state during startup, and for property evaluation at run-time when a transport equation is not being solved for the required variable or when a reference state is needed for calculating “ambient” properties for entrainment purposes.

Parameter	Value	Default
<i>SpeciesName</i>	string	–
<i>{=}</i>	{= are is }	–
<i>Value</i>	real	–

Reference Mole_Fraction

Syntax

Reference Mole_Fraction *SpeciesName* *{=}* *Value*

Summary

Specify the reference mole fraction for the given species

Description

A reference value must be specified for all input variables required by the particular material model for property evaluation. In the typical case where Cantera is used for property evaluation, then reference properties must be

specified for “temperature”, “pressure”, and “mass_fraction”. Other material models may have different dependencies.

These values will be used for reporting property values at the reference state during startup, and for property evaluation at run-time when a transport equation is not being solved for the required variable or when a reference state is needed for calculating “ambient” properties for entrainment purposes.

Parameter	Value	Default
<i>SpeciesName</i>	string	–
<i>{=}</i>	{= are is}	–
<i>Value</i>	real	–

Reference Progress_Variable

Syntax

Reference Progress_Variable *ProgressVariableName* [With Lower Bound *LowerBound* And Upper Bound *UpperBound*]

Summary

Specify the upper and lower bound of a progress variable, in that order

Description

A lower and upper bound must be specified for all progress variables.

Parameter	Value	Default
<i>ProgressVariableName</i>	string	–

Set Schmidt Number For Progress Variable

Syntax

Set Schmidt Number For Progress Variable *ProgressVariableName* *{=}* *Value*

Summary

Specify Schmidt numbers for Progress Variable

Description

Specify Schmidt numbers for Progress Variable

Parameter	Value	Default
<i>ProgressVariableName</i>	string	–
<i>{=}</i>	{= are is}	–
<i>Value</i>	real	–

Set Thermophoresis Coefficient For Progress Variable

Syntax

Set Thermophoresis Coefficient For Progress Variable *ProgressVariableName*
 {=} *Value*

Summary

Set thermophoresis coefficient for progress variable

Parameter	Value	Default
<i>ProgressVariableName</i>	string	–
{=}	{ = are is }	–
<i>Value</i>	real	–

Subroutine For**Syntax**

Subroutine For *PropertyName* {=} *SubName*

Summary

Set the value of the specified variable using a subroutine. If the subroutine should be passed other properties as variables or be given real data, the INPUT VARIABLES FOR and REAL DATA FOR line commands below should be used.

Parameter	Value	Default
<i>PropertyName</i>	string	–
{=}	{ = are is }	–
<i>SubName</i>	string	–

Schmidt_Number**Syntax**

Schmidt_Number {=} *Value*

Summary

Specify the Schmidt number ($Sc = \mu / \rho * D$) to calculate mass diffusivity as a function of viscosity and density.

Description

If transporting both an energy equation and a species equation in turbulent flow, it is mandatory that this be specified and that it be identical to the Prandtl number, to enforce unity Lewis number ($Le = Sc / Pr$). If transporting turbulent species without an energy equation, then either the Schmidt number or an identical mass diffusivity for all species must be specified. In all other cases, this is optional.

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>Value</i>	real	–

Species_Density

Syntax

Species_Density *SpeciesName* *{=}* *Value*

Summary

Specify the species density for the given species

Description

For the SpeciesName parameter, you can either specify the actual species name (O2, N2, CH4, etc.) or the word DEFAULT for a global value applied to all unlisted species. Any species-specific values, if present, will override the DEFAULT value.

This command is only applicable when not using an external third-party library (e.g. Cantera) for property evaluation.

When using a string function, the function may depend on pressure, temperature, mass fraction, or most other properties. When using mass fractions in the string function, use the species name as a suffix, as in

```
Species_Density N2 = "300/temperature + mass_fraction_n2"
```

The function string must be enclosed in quotes if it has spaces or commas. The variable names are not case-sensitive.

Parameter	Value	Default
<i>SpeciesName</i>	string	–
<i>{=}</i>	{= are is}	–
<i>Value</i>	“string”	–

Species_Emissivity

Syntax

Species_Emissivity *SpeciesName* *{=}* *Value*

Summary

Specify the emissivity for the given species

Description

For the *SpeciesName* parameter, you can either specify the actual species name (O2, N2, CH4, etc.) or the word DEFAULT for a global value applied to all

unlisted species. Any species-specific values, if present, will override the DEFAULT value.

When using a string function, the function may depend on pressure, temperature, mass fraction, or most other properties. When using mass fractions in the string function, use the species name as a suffix, as in

```
Species_Emissivity char = "0.9 - 0.01*mass_fraction_epoxy"
```

The function string must be enclosed in quotes if it has spaces or commas. The variable names are not case-sensitive.

Parameter	Value	Default
<i>SpeciesName</i>	string	—
<i>{=}</i>	{= are is}	—
<i>Value</i>	"string"	—

Species_Enthalpy

Syntax

Species_Enthalpy *SpeciesName* *{=}* *Value*

Summary

Specify the species enthalpy for the given species

Description

For the *SpeciesName* parameter, you can either specify the actual species name (O2, N2, CH4, etc.) or the word DEFAULT for a global value applied to all unlisted species. Any species-specific values, if present, will override the DEFAULT value.

This command is only applicable when not using an external third-party library (e.g. Cantera) for property evaluation.

When using a string function, the function may depend on pressure, temperature, mass fraction, or most other properties. When using mass fractions in the string function, use the species name as a suffix, as in

```
Species_enthalpy N2 = "300/temperature + mass_fraction_n2"
```

The function string must be enclosed in quotes if it has spaces or commas. The variable names are not case-sensitive.

Parameter	Value	Default
<i>SpeciesName</i>	string	—
<i>{=}</i>	{= are is}	—
<i>Value</i>	"string"	—

Species_Production_Rate

Syntax

Species_Production_Rate *SpeciesName* {=} *Value*

Summary

warning{This command is deprecated and will be removed in the 5.18 release.
It has not had an effect for several releases.}

Parameter	Value	Default
<i>SpeciesName</i>	string	—
{=}	{= are is}	—
<i>Value</i>	real	—

Species_Specific_Heat

Syntax

Species_Specific_Heat *SpeciesName* {=} *Value*

Summary

Specify the specific heat for the given species

Description

For the *SpeciesName* parameter, you can either specify the actual species name (O2, N2, CH4, etc.) or the word DEFAULT for a global value applied to all unlisted species. Any species-specific values, if present, will override the DEFAULT value.

When using a string function, the function may depend on pressure, temperature, mass fraction, or most other properties. When using mass fractions in the string function, use the species name as a suffix, as in

```
Species_Specific_Heat Al = "300/temperature + mass_fraction_
↵n2"
```

The function string must be enclosed in quotes if it has spaces or commas. The variable names are not case-sensitive.

Parameter	Value	Default
<i>SpeciesName</i>	string	—
{=}	{= are is}	—
<i>Value</i>	“string”	—

Species_Thermal_Conductivity

Syntax

Species_Thermal_Conductivity *SpeciesName* {=} *Value*

Summary

Specify the thermal conductivity for the given species

Description

For the *SpeciesName* parameter, you can either specify the actual species name (O2, N2, CH4, etc.) or the word DEFAULT for a global value applied to all unlisted species. Any species-specific values, if present, will override the DEFAULT value.

When using a string function, the function may depend on pressure, temperature, mass fraction, or most other properties. When using mass fractions in the string function, use the species name as a suffix, as in

```
Species_Thermal_Conductivity A1 = "300/temperature + mass_↵fraction_n2"
```

The function string must be enclosed in quotes if it has spaces or commas. The variable names are not case-sensitive.

Parameter	Value	Default
<i>SpeciesName</i>	string	—
<i>{=}</i>	{ = are is }	—
<i>Value</i>	"string"	—

Turbulent Mixing Model

Syntax

Turbulent Mixing Model *{=}* *{none | parente}* [*ModelArgs*]. . .

Summary

Specify turbulent mixing model to limit reaction chemistry

Description

Use turbulent mixing time scale to determine rate of reaction in chemistry model using EDC 2016 Model (Parente et al. Fuel 2016). Optional model arguments for the Parente turbulent mixing model can be specified on the same line as follow:

```
Turbulent Mixing Model = PARENTE c1 = val1 c2 = val2
```

The gas constants *C1* and *C2* are optional arguments, and have a default value of *C1* = 0.05774 and *C2* = 0.5.

Parameter	Value	Default
<i>{=}</i>	{ = are is }	—
<i>TurbulentMixingModel</i>	{ none parente }	—
<i>ModelArgs</i>	[string]. . .	—

Use Consistently Volume Averaged Properties

Syntax

Use Consistently Volume Averaged Properties [With Postsmoothing
SmoothingIterations]

Summary

Use a wider stencil to evaluate nodal properties, like density

Description

Interpolates input fields to subcontrol volume centers to evaluate a volume average, instead of just using the nodal value. Optionally smooth the result with nodal filter iterations.

Vof Gas Material

Syntax

Vof Gas Material {=} *GasMaterial*

Summary

Name of the material to use for the gas phase in a VOF simulation

Description

Name of the material to use for the gas phase in a VOF simulation.

Parameter	Value	Default
{=}	{= are is}	–
<i>GasMaterial</i>	string	–

Vof Liquid Material

Syntax

Vof Liquid Material {=} *LiquidMaterial*

Summary

Name of the material to use for the liquid phase in a VOF simulation

Description

Name of the material to use for the liquid phase in a VOF simulation.

Parameter	Value	Default
{=}	{= are is}	–
<i>LiquidMaterial</i>	string	–

Tabular Property Library

Scope

Property Specification For Fuego Material

Summary

Begin specification for tabular properties

begin Tabular Property Library *LibNameTag*

Allow Positive Enthalpy Reconstruction

Auxiliary Variable *AuxVarName* Expression *{=}* *Expr*

Enable Full Table Clipping Logging [Every *LogInterval* Steps]

Evaluation Caching Mode *{=}* *{cached | none}*

Ignition Blending Time *{=}* *Value*

Ignition Time *{=}* *Value*

Library Hdf5 File *{=}* *FileName*

Number Of Logged Input Clipping Events *{=}* *Number*

Output Caching Diagnostics

Output Library Variable *LibraryVariable* As *OutputName* [On Output_
↳Block *BlockName*]

Property For Clipping Events Table *{=}* *PropertyName*

Source Term For *ProgressVariableName* Expression *{=}* *Expr*

Table Interpolation Method *{=}* *Value*

Table Units Are In *{cgs | mks}*

Use Aerosol Model *{aksit_moss_snl | none}*

Use Approximate Flamelet Enthalpy Reconstruction

Use Consistently Volume Averaged Properties

Use Field *FuegoField* For Library Input *LibraryInput*

Use Library Source *LibrarySource* As *SourceName*

Use Library Variable *LibraryVariable* For Property {*absorption_coef* |
→*conserved_enthalpy* | *density* | *density_pressure_derivative* | *emissivity* |
→*enthalpy* | *heat_production_rate* | *mass_diffusivity* | *molecular_*
→*weight* | *scattering_coef* | *species_density* | *species_emissivity* |
→*species_enthalpy* | *species_production_rate* | *species_specific_heat* |
→*species_thermal_conductivity* | *specific_heat* | *temperature* | *thermal_*
→*conductivity* | *viscosity*}

end Tabular Property Library *LibNameTag*

Line Commands

Allow Positive Enthalpy Reconstruction

Syntax

Allow Positive Enthalpy Reconstruction

Summary

Allow consistent enthalpy reconstruction to produce positive defect values

Description

If active, the enthalpy defect reconstruction can produce positive values for the reconstruction. Otherwise, positive values are clipped to zero.

Auxiliary Variable

Syntax

Auxiliary Variable *AuxVarName* Expression {=} *Expr*

Summary

Specifies the expression to evaluate for an auxiliary variable.

Description

Specifies the expression to evaluate for an auxiliary variable.

Parameter	Value	Default
<i>AuxVarName</i>	string	—
{=}	{= are is}	—
<i>Expr</i>	(expression)	—

Enable Full Table Clipping Logging

Syntax

Enable Full Table Clipping Logging [Every *LogInterval* Steps]

Summary

Activate detailed logging of table clipping events.

Description

By default table clipping event details are not output to the logfile. This option activates them, but be aware that there can be a substantial performance impact from this logging. Optionally, you can enable logging every N time steps instead of every step.

Evaluation Caching Mode**Syntax**

Evaluation Caching Mode *{=} {cached | none}*

Summary

Set property caching mode in property evaluators

Description

If NONE, do not employ caching in the property evaluators. If CACHED, employ caching in the property evaluators.

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>PropertyCachingMode</i>	{ cached none }	–

Ignition Blending Time**Syntax**

Ignition Blending Time *{=} Value*

Summary

Length of time to blend flamelet model from non-ignited to igniting states

Description

If an ignition time is specified for the flamelet model, it begins at the specified time and is gradually introduced over a finite blending time (this parameter). By “ignition time” + “blending time” all blending effects have been removed.

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>Value</i>	real	1

Ignition Time**Syntax**

Ignition Time *{=} Value*

Summary

Time to begin flamelet ignition

Description

By default ignition is immediate with the flamelet model. This option lets you delay the ignition until a specified time. Prior to that time, the mixture fraction values used in property evaluation are clamped to 0 or 1.

Parameter	Value	Default
<i>{=}</i>	{= are is }	–
<i>Value</i>	real	0

Library Hdf5 File

Syntax

Library Hdf5 File *{=}* *FileName*

Summary

Specify the name (with optional path) for the HDF5 property library generated by fuego_tabular_props

Parameter	Value	Default
<i>{=}</i>	{= are is }	–
<i>FileName</i>	string	–

Number Of Logged Input Clipping Events

Syntax

Number Of Logged Input Clipping Events *{=}* *Number*

Summary

warning{ This command is deprecated and will no longer have any effect. It will be completely removed in a future version of Fuego. }

Parameter	Value	Default
<i>{=}</i>	{= are is }	–
<i>Number</i>	integer	–

Output Caching Diagnostics

Syntax

Output Caching Diagnostics

Summary

Display caching statistics in the logfile.

Description

warning{ This command is deprecated and will no longer have any effect. It will be completely removed in a future version of Fuego. }

Output Library Variable

Syntax

Output Library Variable *LibraryVariable* As *OutputName* [On Output Block *BlockName*]

Summary

Generates output for the requested library variable.

Description

Provide output of the requested property library variable. If the optional output block name is specified, then the variable will only be written to that output block.

Parameter	Value	Default
<i>LibraryVariable</i>	string	–
<i>OutputName</i>	string	–

Property For Clipping Events Table

Syntax

Property For Clipping Events Table *{=}* *PropertyName*

Summary

Set property for displaying clipping event warning log. Default: Use density for clipping event log.

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>PropertyName</i>	string	–

Source Term For

Syntax

Source Term For *ProgressVariableName* Expression *{=}* *Expr*

Summary

Specifies the expression to evaluate.

Description

Specifies the expression to evaluate.

Parameter	Value	Default
<i>ProgressVariableName</i>	string	–
<i>{=}</i>	{= are is }	–
<i>Expr</i>	(expression)	–

Table Interpolation Method

Syntax

Table Interpolation Method *{=}* *Value*

Summary

Which interpolation backend to use (BSPLINE or LAGRANGE)

Description

One may either choose a general B-spline formulation or a Lagrange polynomial as the interpolant for table lookups. The B-spline formulation has better guarantees on continuity and smoothness but is between two to ten times more expensive to evaluate than the Lagrange basis. Lagrange polynomials are likely sufficient for tables with enough resolution for accurate reconstruction of tabulated properties.

Parameter	Value	Default
<i>{=}</i>	{= are is }	–
<i>Value</i>	string	1

Table Units Are In

Syntax

Table Units Are In *{cgs | mks}*

Summary

Describe the length-mass-time unit system used by the table (CGS or MKS)

Description

Shortcut to select meter-kilogram-second (MKS) or centimeter-gram-second unit (CGS) system. This should correspond to the units your table was generated in.

Parameter	Value	Default
<i>UnitSystem</i>	{cgs mks }	–

Use Aerosol Model

Syntax

Use Aerosol Model *{aksit_moss_snl | none}*

Summary

Use preset aerosol model with progress variables

Description

Activate a preset aerosol model for use with tabulated source terms, e.g. for soot modeling.

Parameter	Value	Default
<i>AerosolModel</i>	{aksit_moss_snl none}	–

Use Approximate Flamelet Enthalpy Reconstruction

Syntax

Use Approximate Flamelet Enthalpy Reconstruction

Summary

Use an inexact method of reconstructing the enthalpy defect of nonadiabatic flamelet models.

Description

Uses an inexact method of obtaining the stoichiometric enthalpy defect for nonadiabatic strained laminar flamelet models. In cases with strong heat loss this leads to a large, persistent error and should not be used.

Use Consistently Volume Averaged Properties

Syntax

Use Consistently Volume Averaged Properties

Summary

Use a wider stencil to evaluate nodal properties, like density

Description

Interpolates input fields to subcontrol volume centers to evaluate a volume average, instead of just using the nodal value

Use Field

Syntax

Use Field *FuegoField* For Library Input *LibraryInput*

Summary

Link a Fuego field to the specified property library input variable

Parameter	Value	Default
<i>FuegoField</i>	string	–
<i>LibraryInput</i>	string	–

Use Library Source

Syntax

Use Library Source *LibrarySource* As *SourceName*

Summary

Link property library variable to a tabulated source term

Description

Provide a connection between a tabular property library variable and a source term. The library will be used to evaluate the source term.

Parameter	Value	Default
<i>LibrarySource</i>	string	–
<i>SourceName</i>	string	–

Use Library Variable**Syntax**

Use Library Variable *LibraryVariable* For Property {*absorption_coef* | *conserved_enthalpy* | *density* | *density_pressure_derivative* | *emissivity* | *enthalpy* | *heat_production_rate* | *mass_diffusivity* | *molecular_weight* | *scattering_coef* | *species_density* | *species_emissivity* | *species_enthalpy* | *species_production_rate* | *species_specific_heat* | *species_thermal_conductivity* | *specific_heat* | *temperature* | *thermal_conductivity* | *viscosity*}

Summary

Link property library variable to a Fuego property

Description

Provide a connection between a tabular property library variable and a Fuego property. The library will be used to evaluate the property.

For adiabatic flamelet simulations, the following properties are required:
Density and Viscosity.

For non-adiabatic flamelet simulations, the following properties are required:
Density, Viscosity, Temperature, Specific_Heat, Enthalpy, and
Conserved_Enthalpy. You must also activate both the Enthalpy and
Conserved_Enthalpy equations.

Parameter	Value	Default
<i>Library-Variable</i>	string	–
<i>Fuego-Properties</i>	{absorption_coef conserved_enthalpy density density_pressure_derivative emissivity enthalpy heat_production_rate mass_diffusivity molecular_weight scattering_coef species_density species_emissivity species_enthalpy species_production_rate species_specific_heat species_thermal_conductivity specific_heat temperature thermal_conductivity viscosity}	–

7.2 Transfer

Transfer

Scope

Fuego Procedure

Summary

Transfer region/mesh information. The mechanics/variables information will get sorted out by the calling procedure.

begin Transfer *Transfer_name*

Abort If Field Not Defined On Copy Transfer Send Or Receive Object

Abort If Search Object Outside Of Tolerance

All Fields

Copy {surface | volume} {constraints | elements | nodes} From *From_region_name* To *To_region_name*

Distance Function Is Closest Receive Node To Send Centroid

Exclude Ghosted

From {constraints | elements | nodes} To {constraints | elements | faces | gauss_points | nodes}

Gauss Point Integration Order `{=}` `Order`

Geometric Tolerance `{=}` `Geometric_tolerance`

Inspect With File `{=}` `File_name` Ids `{=}` `ID_list...`

Interpolate `{surface | volume}` `{constraints | elements | nodes}` From `From_region_name` To `To_region_name`

Interpolation Function `Function_Name`

Nodes Outside Region `{=}` `{abort | extrapolate | ignore | project | truncate}`

Parametric Tolerance `{=}` `Parametric_tolerance`

Patch Recovery Evaluation `{=}` `{linear least squares | linear moving least squares | quadratic least squares | quadratic moving least squares}`

Search Coordinate Field `Source_field_name` State `{new | nm1 | nm2 | nm3 | nm4 | none | old}` To `Destination_field_name` State `{new | nm1 | nm2 | nm3 | nm4 | none | old}`

Search Geometric Tolerance `{=}` `Geometric_tolerance`

Search Surface Gap Tolerance `{=}` `Surface_gap_tolerance` [Or Less]

Search Type `{=}` [`{detailed | parallel | proximity}` `{detailed | parallel | proximity}`]

Select One Receiver For Each Send Object

Select One Unique Receiver For Each Send Object

Send `{}` Fields

Send Block `From_blocks...` To `To_blocks...`

Send Field `Source_field_name` State `{new | nm1 | nm2 | nm3 | nm4 | none | old}` To `Destination_field_name` State `{new | nm1 | nm2 | nm3 | nm4 | none | old}` [Lower Bound `Lower_bound` Upper Bound `Upper_bound`]

Toggle Search Warnings `{=}` `{false | no | off | on | true | yes}`

Use Centroid For Geometric Proximity

```
begin Receive Blocks  
end
```

```
begin Send Blocks  
end
```

end Transfer *Transfer_name*

Line Commands

Abort If Field Not Defined On Copy Transfer Send Or Receive Object

Syntax

Abort If Field Not Defined On Copy Transfer Send Or Receive Object

Summary

For testing purposes only. Normally mesh objects in the send or receive mesh which do not have the specified field defined on them are just ignored. This line command allows the construction of tests in which it is known that every mesh object should have the specified field defined on it and to abort if that field is not found.

Abort If Search Object Outside Of Tolerance

Syntax

Abort If Search Object Outside Of Tolerance

Summary

For debugging purposes only. Abort transfer if search object lie outside of specified geometric tolerance.

This command is deprecated. Use “Nodes outside region = abort” instead.

All Fields

Syntax

All Fields

Summary

Select all fields for transfer that have same name and state for source and destination regions.

Copy

Syntax

Copy *{surface | volume} {constraints | elements | nodes}* From
From_region_name To *To_region_name*

Summary

Copy transfer elements, nodes or constraints from one region to another. The copy transfer is very specific in that the sending and receiving mesh parts must have identical global ids for every element to be copied. The copy transfer works by iterating over all the mesh objects in the receiving mesh and using the global id of the receiving mesh object to find a mesh object in the sending mesh with the same global id. The field to transfer is then copied from the sending to receiving objects. There is no interpolation and the actual coordinates of the sending and receiving objects are not used and could be very different. The copy transfer is used in very special cases where the same mesh was read into both the sending and receiving meshes, there was no element death and there was no adaptivity. In this special case, a copy transfer can be much faster than an interpolation transfer.

Parameter	Value	Default
<i>Option1</i>	{ surface volume }	–
<i>Option2</i>	{ constraints elements nodes }	–
<i>From_region_name</i>	string	–
<i>To_region_name</i>	string	–

Distance Function Is Closest Receive Node To Send Centroid

Syntax

Distance Function Is Closest Receive Node To Send Centroid

Summary

To be used in conjunction with “SELECT ONE UNIQUE RECEIVER FOR EACH SEND OBJECT”. This helped in the case where the sending and receiving element blocks did not overlap and an element transfer was using element centroids for the distance computation. The elements were very distorted so that a centroid of a surface element could be far from the surface. It was wanted that the receiving element be the one close to the surface of the block and close to the sending element in the adjacent block. Using the corner nodes was enough since it was a tet mesh with plane faces. In this particular and unusual case this alternative method of matching sending and receiving elements was useful, but it is not expected to be used often or maybe never again.

Exclude Ghosted

Syntax

Exclude Ghosted

Summary

exclude ghosted nodes from a copy transfer

From

Syntax

From *{constraints | elements | nodes}* To *{constraints | elements | faces | gauss_points | nodes}*

Summary

Allows the send/receive mesh objects to be different. For a volume element field transfer (e.g. shell) to a face rank field on a surface, use ‘from elements to faces’.

Parameter	Value	Default
<i>Option1</i>	{constraints elements nodes}	–
<i>Option2</i>	{constraints elements faces gauss_points nodes}	–

Gauss Point Integration Order

Syntax

Gauss Point Integration Order *{=} Order*

Summary

Integration order to use when transferring to Gauss points.

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>Order</i>	integer	–

Geometric Tolerance

Syntax

Geometric Tolerance *{=} Geometric_tolerance*

Summary

This is the dimensional tolerance applied during the initial (coarse) search. If specified, all the coarse search boxes are padded by this value. The default behavior is for this to be 1e-9 times the problem bounding box size (diagonal) plus a 10 percent relative expansion per element. If a value is specified for this, it is used as a padding in place of the default with no relative box expansion.

During the interpolation transfer there is a geometric search based on the coordinates of the send and receive objects. As part of this search, an axis aligned bounding box is contracted for each sending object and GEOMETRIC TOLERANCE is used to make this box bigger than just a tight bounding box. Lists of receiving points are then quickly found within these axis aligned boxes.

If all points in the receiving mesh are within at least one box, no additional searching needs to be done and the search algorithm is fast. If there are still points in the receiving mesh that were outside of EVERY box, then a warning message will be issued about an “expensive search for extrapolation” for these points. This ‘expensive search’ can be very costly if a large number of receiving

objects fall into this category and this line command is provided for those special cases.

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>Geometric_tolerance</i>	real	–

Inspect With File

Syntax

Inspect With File *{=}* *File_name* Ids *{=}* *ID_list* . .

Summary

STK Transfer inspection tool that allows user to output the search results for a specified list of entity ids on the receive mesh. The rank of the entities is deduced from the type of transfer being set up.

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>File_name</i>	string	–
<i>{=}</i>	{= are is}	–
<i>ID_list</i>	integer. . .	–

Interpolate

Syntax

Interpolate *{surface | volume}* *{constraints | elements | nodes}* From *From_region_name* To *To_region_name*

Summary

Interpolate will transfer elements, nodes or constraints from one mesh to another. The interpolation transfer is very general in that the field values to transfer will be interpolated from the sending to receiving mesh based on the coordinates of the sending and receiving mesh objects.

Many line commands can be used to modify the behavior of the interpolation transfer but the basic algorithm is straightforward. Every mesh object in the receiving mesh is converted into a point. For elements this is the average of the nodal coordinates. An element in the sending mesh containing this point is found. If the field to transfer is nodal, the element shape functions are used to interpolate the nodal field to the receiving point. If the field to transfer is elemental, a bi-linear least squares fit based upon neighboring elements is first performed and then used to define the interpolation of the element field at the receiving point.

Parameter	Value	Default
<i>Option1</i>	{ surface volume }	–
<i>Option2</i>	{ constraints elements nodes }	–
<i>From_region_name</i>	string	–
<i>To_region_name</i>	string	–

Interpolation Function

Syntax

Interpolation Function *Function_Name*

Summary

Allows an application defined subroutine to be used for the interpolation. Normally the interpolation transfer will determine the best type of interpolation to use for node or element entities: Basis functions for nodal fields and a neighborhood least squares fit for element fields. This command line can be used to override the default behavior if needed. Available interpolation functions are Sum, Copy, Master_Element, Element_Centroid_Constant and Element_Centroid_Linear.

Sum - Sum surrounding entity values on source and send to destination.

Copy - Copy entity values directly from source to destination.

Master_Element - Send interpolated nodal values from nearest source element to destination.

Element_Centroid_Constant - For nodal field transfer send computed centroid value from the nearest source element to destination. For element field transfer send the element field value from the nearest source element to the destination

Element_Centroid_Linear - Send values of interpolated centroid values obtained using patch recovery to destination. Patch recovery involves a least squares reconstruction and evaluation using a patch of elements surrounding the nearest source element. For linear and higher-order reconstructions one should consult the Patch Recovery Evaluation option.

Parameter	Value	Default
<i>Function_Name</i>	string	–

Nodes Outside Region

Syntax

Nodes Outside Region {=} {*abort* | *extrapolate* | *ignore* | *project* | *truncate*}

Summary

This line command defines what to do when a receiving point is outside the scope of the sending mesh.

IGNORE - The receiving mesh object can be ignored and will receive no value. This is almost never a good idea as it can cause mesh objects just outside to have a zero value when the nodes just inside the mesh might have very large values. This can result in a discontinuous receiving field.

EXTRAPOLATE - This is the default behavior. The sending field is extrapolated beyond the bounds of the sending mesh. This can lead to extrapolation error, such as when a large gradient at the surface causes a negative values when only positive values are acceptable. If this happens to the upper and lower bounds that can be placed on the fields to be transferred with the SEND FIELD command.

TRUNCATE - The receiving coordinate is projected back to the surface of the sending mesh to determine a value. This ensures that the receiving value is not outside of the field values in the sending mesh.

PROJECT - This option is similar to TRUNCATE in which the receiving coordinate is projected back to the surface of the sending mesh to determine a value. In this case more effort is made to make sure that the projection is normal to the surface in the sending mesh. Sometimes gives a better result than Truncate but is a little more expensive to compute.

If the PROJECT option is used in transferring of surface values, the sending mesh should envelope the receiving mesh. Failure to satisfy this condition will generally result in failure of the transfer.

ABORT - If any receiving point is outside the sending mesh by more than the geometric tolerance, abort the simulation. Do not attempt to project, extrapolate, or otherwise handle the point.

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>Option</i>	{abort extrapolate ignore project truncate}	–

Parametric Tolerance

Syntax

Parametric Tolerance *{=}* *Parametric_tolerance*

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>Parametric_tolerance</i>	real	–

Patch Recovery Evaluation

Syntax

Patch Recovery Evaluation *{=}* *{linear least squares | linear moving least*

squares | quadratic least squares | quadratic moving least squares}

Summary

This line command defines the available choices for the patch recovery and evaluation algorithm when using interpolation for element variables. The default option is Linear Least Squares.

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>Option</i>	{linear least squares linear moving least squares quadratic least squares quadratic moving least squares}	–

Search Coordinate Field

Syntax

Search Coordinate Field *Source_field_name* State *{new | nm1 | nm2 | nm3 | nm4 | none | old}* To *Destination_field_name* State *{new | nm1 | nm2 | nm3 | nm4 | none | old}*

Summary

Normally the interpolation transfers use the default coordinate field to determine geometry information. This line command can be used to specify an alternate field.

Parameter	Value	Default
<i>Source_field_name</i>	string	–
<i>Option1</i>	{new nm1 nm2 nm3 nm4 none old}	–
<i>Destination_field_name</i>	string	–
<i>Option2</i>	{new nm1 nm2 nm3 nm4 none old}	–

Search Geometric Tolerance

Syntax

Search Geometric Tolerance *{=}* *Geometric_tolerance*

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>Geometric_tolerance</i>	real	–

Search Surface Gap Tolerance

Syntax

Search Surface Gap Tolerance *{=}* *Surface_gap_tolerance* [Or Less]

Summary

This is the dimensional tolerance applied during the initial (coarse) search. If specified, all the coarse search boxes are padded by this value. The default behavior is for this to be 1e-9 times the problem bounding box size (diagonal) plus a 10 percent relative expansion per element. If a value is specified for this, it is used as a padding in place of the default with no relative box expansion.

This is a tricky parameter best ignored, let it default to something based on the problem size. During the interpolation transfer there is a geometric search based on the coordinates of the send and receive objects. As part of this search, an axis aligned bounding box is contracted for each sending object and SEARCH GAP TOLERANCE is used to make this box bigger than just a tight bounding box. Lists of receiving points are then quickly found within these axis aligned boxes.

If all points in the receiving mesh are within at least one box, no additional searching needs to be done and the search algorithm is fast. If there are still points in the receiving mesh that were outside of EVERY box, then a warning message will be issued about an “expensive search for extrapolation” for these points. This ‘expensive search’ can be very costly if a large number of receiving objects fall into this category and this line command is provided for those special cases.

The OR LESS optional parameter is used when the tolerance must be set to large value for one part of the mesh but much of the mesh needs a much smaller value. In some cases it is necessary for the tolerance to be set to the actual largest surface gap tolerance which may be far too large a gap for the rest of the mesh. Setting OR LESS allows the search tolerance to be reduced in areas of the mesh thus resulting in a faster search.

Parameter	Value	Default
<i>{=}</i>	{= are is }	—
<i>Surface_gap_tolerance</i>	real	—

Search Type

Syntax

Search Type *{=}* [*{detailed | parallel | proximity}* *{detailed | parallel | proximity}* *{detailed | parallel | proximity}*]

Parameter	Value	Default
<i>{=}</i>	{= are is }	—

Select One Receiver For Each Send Object

Syntax

Select One Receiver For Each Send Object

Summary

This option will cause each sending object to be used once and only once. This will have the side effect of some receiving objects not getting any value at all. If you use this option, you will also want to set

NODES OUTSIDE REGION IGNORE

The example which necessitated this option was a case in which there was a delta function defined on an element in the sending mesh. It was desirable that the delta functions be summed into the receiving mesh such that the total value of the sending was conserved. It was better to have only a single element on the receiving side have a non-zero value that was the sum of sending values and not worry about how close the receiving element was to the sending element. A check that this option is working is to use Encore to computer the sum of the values of the sending and receiving fields to make sure the total sum is the same.

Select One Unique Receiver For Each Send Object

Syntax

Select One Unique Receiver For Each Send Object

Summary

An unusual flag to get around an odd problem. Normally each receive object transfers from the nearest sending object so it is almost always the case that a send object will be used multiple times to define a receiving value. This option will cause each sending object to be used only once. This will have the side effect of some receiving objects not getting any value at all. If you use this option, you will also want to set

NODES OUTSIDE REGION IGNORE

or else the uniqueness will be lost for nodes outside the sending region. The example which necessitated this option was a case in which there was a delta function defined on an element in the sending mesh. It was desirable that the delta function be defined on the receiving mesh for only a single element in the neighborhood of the sending element. The analysis was more sensitive to the number of delta functions on the receiving side than the location. So it was better to have only a single element on the receiving side have a non-zero value and not worry about how close the receiving element was to the sending element.

Send

Syntax

Send *//* Fields

Summary

Use predefined transfer semantics provided by the specified name.

Parameter	Value	Default
<i>Predefined-transfer</i>	{ }	—

Send Block

Syntax

Send Block *From_blocks...* To *To_blocks...*

Summary

Add element blocks to a particular same mesh element copy transfer operator.

The copy transfer can have multiple of these lines to define many blocks, but each line sends a single block to a single block:

```
SEND BLOCK block_1 TO block_1
SEND BLOCK block_101 TO block_101
```

The interpolation transfer can have only a single SEND BLOCK line, but can define many from/to blocks:

```
SEND BLOCK block_3 block_5 block_6 TO block_3 block_5
```

Parameter	Value	Default
<i>From_blocks</i>	string...	—
<i>To_blocks</i>	string...	—

Send Field

Syntax

Send Field *Source_field_name* State {*new* | *nm1* | *nm2* | *nm3* | *nm4* | *none* | *old*}
 To *Destination_field_name* State {*new* | *nm1* | *nm2* | *nm3* | *nm4* | *none* | *old*} [
 Lower Bound *Lower_bound* Upper Bound *Upper_bound*]

Summary

Specifies the mapping between source and destination field names. Vector and tensor fields can be subscripted using parenthesis and 1's based or brackets and 0 based. Notes on subscripting:

- Does not work for COPY transfers, only INTERPOLATION type transfers.
- If the field name itself actually contains either parenthesis or brackets then we are in trouble and an error is going to be thrown due to a syntax error in index specification.
- Only a single subscript is allowed so vectors of vectors or higher order tensors can not use double subscripts. But it should be possible to

determine the correct offset within the field and pick out the correct value with a little effort.

- Once subscripted, only a single value will be transferred. It is not possible to transfer multiple values starting at a certain index, instead multiple line commands must be used, as shown above.
- The indexes can be 0 based with brackets or 1 based when using parenthesis. Although this could be very confusing if mixed within a single line command.
- Both the from and to fields can be subscripted independently on the same line.

example

```
SEND FIELD velocity TO velocity
SEND FIELD temp      TO temperature lower bound 0
SEND FIELD x         TO y lower bound 10 upper bound 100
SEND FIELD A(2)      TO B(3) lower bound 10 upper bound 100
SEND FIELD A[1]      TO B[2] lower bound 10 upper bound 100
```

Parameter	Value	Default
<i>Source_field_name</i>	string	—
<i>Option1</i>	{new nm1 nm2 nm3 nm4 none old}	—
<i>Destination_field_name</i>	string	—
<i>Option2</i>	{new nm1 nm2 nm3 nm4 none old}	—

Toggle Search Warnings

Syntax

Toggle Search Warnings *{=}* *{false | no | off | on | true | yes}*

Summary

Specify whether warnings about entities outside of the search domain should be printed. The default behavior is to always print these warning messages.

Parameter	Value	Default
<i>{=}</i>	{= are is}	—
<i>Option</i>	{false no off on true yes}	—

Use Centroid For Geometric Proximity

Syntax

Use Centroid For Geometric Proximity

Summary

STK Transfer option to trigger the use of centroid based proximity comparison

for selecting the best interpolating entity when receive entities lie outside of the domain. Default geometric proximity comparison is based on geometric projection which is more expensive but accurate. The use of this option is a way to reduce computational cost especially for meshes that are fairly regular. However, there is no guarantee of accuracy.

Receive Blocks

Summary

I/O blocks to include in the transfer. Fields defined on these blocks will receive fields from the sending blocks.

begin Receive Blocks

Include All Blocks

Include Block *{=}* *IncludeBlockList...*

Remove Block *{=}* *ExcludeBlockList...*

end Receive Blocks

Line Commands

Include All Blocks

Syntax

Include All Blocks

Summary

Use this parameters definition for all blocks. Example:

```
BEGIN SEND BLOCKS
  INCLUDE ALL BLOCKS
  REMOVE BLOCK = block_0 block_1
END SEND BLOCKS
```

Include Block

Syntax

Include Block *{=}* *IncludeBlockList...*

Summary

List of blocks to exclude. Example:

```
BEGIN SEND BLOCKS
  INCLUDE BLOCK = block_3 block_4
END SEND BLOCKS
```

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>IncludeBlockList</i>	string. . .	–

Remove Block

Syntax

Remove Block *{=}* *ExcludeBlockList. . .*

Summary

List of blocks to exclude. Example:

```
BEGIN SEND BLOCKS
  INCLUDE ALL BLOCKS
  REMOVE BLOCK = block_0 block_1
END SEND BLOCKS
```

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>ExcludeBlockList</i>	string. . .	–

Send Blocks

Summary

I/O blocks to include in the transfer. Fields defined on these blocks will be transferred to fields in the receiving blocks.

begin Send Blocks

Include All Blocks

Include Block *{=}* *IncludeBlockList. . .*

Remove Block *{=}* *ExcludeBlockList. . .*

end Send Blocks

Line Commands

Include All Blocks

Syntax

Include All Blocks

Summary

Use this parameters definition for all blocks. Example:

```
BEGIN SEND BLOCKS
  INCLUDE ALL BLOCKS
  REMOVE BLOCK = block_0 block_1
END SEND BLOCKS
```

Include Block

Syntax

Include Block *{=}* *IncludeBlockList*. . .

Summary

List of blocks to exclude. Example:

```
BEGIN SEND BLOCKS
  INCLUDE BLOCK = block_3 block_4
END SEND BLOCKS
```

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>IncludeBlockList</i>	string. . .	–

Remove Block

Syntax

Remove Block *{=}* *ExcludeBlockList*. . .

Summary

List of blocks to exclude. Example:

```
BEGIN SEND BLOCKS
  INCLUDE ALL BLOCKS
  REMOVE BLOCK = block_0 block_1
END SEND BLOCKS
```

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>ExcludeBlockList</i>	string. . .	–

7.3 Solvers

Tpetra Solver Block

Tpetra Equation Solver

Scope

Sierra

Summary

Use a Tpetra-based linear solver.

Description

If the linear solve fails it will report an error code. While these codes are not universally standardized across linear solvers, they can generally be interpreted with the following rules:

- **maxits**: The linear solver reached the maximum number of linear solver iterations and did not reach a solution. Check the final linear solver residual to see if this is serious or not. Even if the error is not serious (residuals are small) this usually means the solve is much more expensive than necessary and a better solver/preconditioner combination should be used.
- **Error 1 (or -1)**: The linear solver encountered a recoverable internal error. As always, check the final linear solver residual reported in the log file to determine if it is serious or not.
- **Error 2 (or -2, 4, -4, 5, -5)**: The linear solver encountered a fatal internal error. This is often because of a singular matrix or a NaN or Infinite term. This is almost always a serious error.
- **Error 3 (or -3)**: The linear solver had an internal loss of precision. This is often not a serious error, but may indicate the need for a better solver/preconditioner combination. As always, check the final linear solver residual reported in the log file to determine if it is serious or not.

begin Tpetra Equation Solver *Solver Name*

```
Bc Enforcement {=} {exact | exact_no_column_mod | remove | solver |   
↪solver_no_column_mod}
```

```
Failed Solve Matrix Output {=} value
```

```
Matrix Output {=} value
```

```
Matrix Output Time Range {=} value1[ value2]
```

```

Matrix Scaling {=} {diagonal / one_norm}

begin Bicgstab Solver
end

begin Cg Solver
end

begin Expert Belos Solver
end

begin Gcrodr Solver
end

begin Gmres Solver
end

begin Klu2 Solver
end

begin Poly Solver
end

begin Preset Solver
end

begin Preset Solver
end

begin Superlu Solver
end

begin Umfpack Solver
end

end Tpetra Equation Solver Solver Name

```

Line Commands

Bc Enforcement

Syntax

Bc Enforcement *{=}* *{exact | exact_no_column_mod | remove | solver | solver_no_column_mod}*

Summary

Control how Dirichlet BC's are enforced. Fuego only.

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>value</i>	{exact exact_no_column_mod remove solver solver_no_column_mod}	–

Failed Solve Matrix Output

Syntax

Failed Solve Matrix Output *{=}* *value*

Summary

On a failed linear solve, output the matrix to the specified file name.

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>value</i>	string	–

Matrix Output

Syntax

Matrix Output *{=}* *value*

Summary

Output the matrix to the specified file name.

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>value</i>	string	–

Matrix Output Time Range

Syntax

Matrix Output Time Range *{=}* *value1* [*value2*]

Summary

Range of simulation time to output the matrix to file, default is dump out all matrices per step.

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>value</i>	real1[real2]	–

Matrix Scaling

Syntax

Matrix Scaling *{=}* *{diagonal | one_norm}*

Summary

Apply a pre-solve scaling to the matrix.

Description

In some problems scaling the matrix prior to the linear solve can help reduce the condition number of the matrix. This should be used in addition to an appropriate preconditioner.

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>value</i>	{ diagonal one_norm }	–

Tpetra Solvers

Klu2 Solver

Scope

Teko Subblock Solver, Tpetra Equation Solver

Summary

Use a KLU2 direct solver.

begin Klu2 Solver

end Klu2 Solver

Superlu Solver

Scope

Teko Subblock Solver, Tpetra Equation Solver

Summary

Use a SUPERLU direct solver.

begin Superlu Solver

end Superlu Solver

Umfpack Solver

Scope

Teko Subblock Solver, Tpetra Equation Solver

Summary

Use a UMFPACK direct solver.

begin Umfpack Solver

end Umfpack Solver

Cg Solver

Scope

Teko Subblock Solver, Tpetra Equation Solver

Summary

Use a conjugate gradient solver.

begin Cg Solver

Convergence Tolerance `{=} value` [Minimum = `tolerance_floor`]

Maximum Iterations `{=} value`

Residual Output `{=} [{disabled | enabled}]`

Residual Scaling `{=} {none | preconditioned_r0 | r0 | rhs}`

begin Dd-Ilu Preconditioner
end

begin Dd-Ilut Preconditioner

```

end

begin Dd-Parilut Preconditioner
end

begin Expert Ifpack2 Preconditioner
end

begin Fastilu Preconditioner
end

begin Jacobi Preconditioner
end

begin Muelu Preconditioner
end

begin Preset Preconditioner
end

begin Sgs Preconditioner
end

begin Sgs2 Preconditioner
end

end Cg Solver

```

Line Commands

Convergence Tolerance

Syntax

Convergence Tolerance **[=]** *value* [Minimum = *tolerance_floor*]

Summary

Set the convergence tolerance for the linear solver. Setting CONVERGENCE TOLERANCE = AUTOMATIC enables an autonomous linear convergence tolerance calculation, which is computed from a combination of the initial linear residual and user-specified nonlinear residual tolerance. An additional floor value on the linear convergence tolerance can be declared with CONVERGENCE TOLERANCE = AUTOMATIC MINIMUM = <val>, where the default floor value is 1.0e-6

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>value</i>	string	–

Maximum Iterations

Syntax

Maximum Iterations *{=}* *value*

Summary

Set the maximum number of iterations taken by the linear solver

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>value</i>	integer	300

Residual Output

Syntax

Residual Output *{=}* [*{disabled | enabled}*]

Summary

Output the linear residual history to files in the residual_output/ directory, which is created automatically. For timestep NNN and linear system NAME, this will output a .csv file named residual_output/linear_solves_NAME.stepNNN. This file includes information about the linear residual at every iteration across all nonlinear iterations.

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–

Residual Scaling

Syntax

Residual Scaling *{=}* *{none | preconditioned_r0 | r0 | rhs}*

Summary

Set the residual scaling for the linear solver

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>value</i>	{ none preconditioned_r0 r0 rhs }	r0

Bicgstab Solver

Scope

Teko Subblock Solver, Tpetra Equation Solver

Summary

Use a bicgstab solver.

begin Bicgstab Solver

Convergence Tolerance *{=}* *value* [Minimum = *tolerance_floor*]

Maximum Iterations *{=}* *value*

Residual Output *{=}* [*{disabled | enabled}*]

Residual Scaling *{=}* *{none | preconditioned_r0 | r0 | rhs}*

begin Dd-Ilu Preconditioner
end

begin Dd-Ilut Preconditioner
end

begin Dd-Parilut Preconditioner
end

begin Expert Ifpack2 Preconditioner
end

begin Fastilu Preconditioner
end

begin Jacobi Preconditioner
end

begin Muelu Preconditioner
end

begin Preset Preconditioner
end

begin Sgs Preconditioner
end

```
begin Sgs2 Preconditioner
end
```

end Bicgstab Solver

Line Commands

Convergence Tolerance

Syntax

Convergence Tolerance *{=}* *value* [Minimum = *tolerance_floor*]

Summary

Set the convergence tolerance for the linear solver. Setting CONVERGENCE TOLERANCE = AUTOMATIC enables an autonomous linear convergence tolerance calculation, which is computed from a combination of the initial linear residual and user-specified nonlinear residual tolerance. An additional floor value on the linear convergence tolerance can be declared with CONVERGENCE TOLERANCE = AUTOMATIC MINIMUM = <val>, where the default floor value is 1.0e-6

Parameter	Value	Default
<i>{=}</i>	{= are is }	–
<i>value</i>	string	–

Maximum Iterations

Syntax

Maximum Iterations *{=}* *value*

Summary

Set the maximum number of iterations taken by the linear solver

Parameter	Value	Default
<i>{=}</i>	{= are is }	–
<i>value</i>	integer	300

Residual Output

Syntax

Residual Output *{=}* [*{disabled | enabled}*]

Summary

Output the linear residual history to files in the residual_output/ directory, which is created automatically. For timestep NNN and linear system NAME, this will output a .csv file named residual_output/linear_solves_NAME.stepNNN. This

file includes information about the linear residual at every iteration across all nonlinear iterations.

Parameter	Value	Default
<i>{=}</i>	{= are is}	–

Residual Scaling

Syntax

Residual Scaling *{=}* *{none | preconditioned_r0 | r0 | rhs}*

Summary

Set the residual scaling for the linear solver

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>value</i>	{none preconditioned_r0 r0 rhs}	r0

Gmres Solver

Scope

Teko Block Solver, Teko Subblock Solver, Tpetra Equation Solver

Summary

Use a GMRES solver.

begin Gmres Solver

Convergence Tolerance *{=}* *value* [Minimum = *tolerance_floor*]

Maximum Iterations *{=}* *value*

Residual Output *{=}* [*{disabled | enabled}*]

Residual Scaling *{=}* *{none | preconditioned_r0 | r0 | rhs}*

Restart Iterations *{=}* *value*

begin Dd-Ilu Preconditioner
end

begin Dd-Ilut Preconditioner
end

begin Dd-Parilut Preconditioner

```

end

begin Expert Ifpack2 Preconditioner
end

begin Expert Teko Preconditioner
end

begin Fastilu Preconditioner
end

begin Jacobi Preconditioner
end

begin Muelu Preconditioner
end

begin Preset Preconditioner
end

begin Sgs Preconditioner
end

begin Sgs2 Preconditioner
end

begin Teko Preconditioner
end

end Gmres Solver

```

Line Commands

Convergence Tolerance

Syntax

Convergence Tolerance *[**!=** value [Minimum = *tolerance_floor*]*

Summary

Set the convergence tolerance for the linear solver. Setting CONVERGENCE TOLERANCE = AUTOMATIC enables an autonomous linear convergence tolerance calculation, which is computed from a combination of the initial linear residual and user-specified nonlinear residual tolerance. An additional floor value on the linear convergence tolerance can be declared with CONVERGENCE TOLERANCE = AUTOMATIC MINIMUM = <val>, where

the default floor value is 1.0e-6

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>value</i>	string	–

Maximum Iterations

Syntax

Maximum Iterations *{=}* *value*

Summary

Set the maximum number of iterations taken by the linear solver

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>value</i>	integer	300

Residual Output

Syntax

Residual Output *{=}* [*{disabled | enabled}*]

Summary

Output the linear residual history to files in the residual_output/ directory, which is created automatically. For timestep NNN and linear system NAME, this will output a .csv file named residual_output/linear_solves_NAME.stepNNN. This file includes information about the linear residual at every iteration across all nonlinear iterations.

Parameter	Value	Default
<i>{=}</i>	{= are is}	–

Residual Scaling

Syntax

Residual Scaling *{=}* *{none | preconditioned_r0 | r0 | rhs}*

Summary

Set the residual scaling for the linear solver

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>value</i>	{none preconditioned_r0 r0 rhs}	r0

Restart Iterations

Syntax

Restart Iterations `{=}` *value*

Summary

Set the number of iterations between GMRES restarts.

Parameter	Value	Default
<code>{=}</code>	{= are is}	–
<i>value</i>	integer	30

Poly Solver

Scope

Teko Subblock Solver, Tpetra Equation Solver

Summary

Use a poly solver.

begin Poly Solver

Convergence Tolerance `{=}` *value* [Minimum = *tolerance_floor*]

Maximum Iterations `{=}` *value*

Polynomial Order `{=}` *value*

Residual Output `{=}` [*{disabled | enabled}*]

Residual Scaling `{=}` *{none | preconditioned_r0 | r0 | rhs}*

Restart Iterations `{=}` *value*

begin Dd-Ilu Preconditioner
end

begin Dd-Ilut Preconditioner
end

begin Dd-Parilut Preconditioner
end

begin Expert Ifpack2 Preconditioner
end

begin Fastilu Preconditioner

```

end

begin Jacobi Preconditioner
end

begin Muelu Preconditioner
end

begin Preset Preconditioner
end

begin Sgs Preconditioner
end

begin Sgs2 Preconditioner
end

end Poly Solver

```

Line Commands

Convergence Tolerance

Syntax

Convergence Tolerance *{=}* *value* [Minimum = *tolerance_floor*]

Summary

Set the convergence tolerance for the linear solver. Setting CONVERGENCE TOLERANCE = AUTOMATIC enables an autonomous linear convergence tolerance calculation, which is computed from a combination of the initial linear residual and user-specified nonlinear residual tolerance. An additional floor value on the linear convergence tolerance can be declared with CONVERGENCE TOLERANCE = AUTOMATIC MINIMUM = <val>, where the default floor value is 1.0e-6

Parameter	Value	Default
<i>{=}</i>	{= are is }	–
<i>value</i>	string	–

Maximum Iterations

Syntax

Maximum Iterations *{=}* *value*

Summary

Set the maximum number of iterations taken by the linear solver

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>value</i>	integer	300

Polynomial Order

Syntax

Polynomial Order *{=}* *value*

Summary

Set the polynomial order of polynomial preconditioner.

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>value</i>	integer	5

Residual Output

Syntax

Residual Output *{=}* [*{disabled | enabled}*]

Summary

Output the linear residual history to files in the residual_output/ directory, which is created automatically. For timestep NNN and linear system NAME, this will output a .csv file named residual_output/linear_solves_NAME.stepNNN. This file includes information about the linear residual at every iteration across all nonlinear iterations.

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–

Residual Scaling

Syntax

Residual Scaling *{=}* *{none | preconditioned_r0 | r0 | rhs}*

Summary

Set the residual scaling for the linear solver

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>value</i>	{ none preconditioned_r0 r0 rhs }	r0

Restart Iterations

Syntax

Restart Iterations **{=}** *value*

Summary

Set the number of iterations between GMRES restarts.

Parameter	Value	Default
{=}	{= are is}	–
<i>value</i>	integer	30

Gcrodr Solver

Scope

Teko Subblock Solver, Tpetra Equation Solver

Summary

Use a GCRODR solver.

begin Gcrodr Solver

Convergence Tolerance **{=}** *value* [Minimum = *tolerance_floor*]

Maximum Iterations **{=}** *value*

Recycled Blocks **{=}** *value*

Residual Output **{=}** [{*disabled* | *enabled*}]

Residual Scaling **{=}** {*none* | *preconditioned_r0* | *r0* | *rhs*}

Restart Iterations **{=}** *value*

begin Dd-Ilu Preconditioner
end

begin Dd-Ilut Preconditioner
end

begin Dd-Parilut Preconditioner
end

begin Expert Ifpack2 Preconditioner
end

```
begin Fastilu Preconditioner
end
```

```
begin Jacobi Preconditioner
end
```

```
begin Muelu Preconditioner
end
```

```
begin Preset Preconditioner
end
```

```
begin Sgs Preconditioner
end
```

```
begin Sgs2 Preconditioner
end
```

end Gcrodr Solver

Line Commands

Convergence Tolerance

Syntax

Convergence Tolerance *{=}* *value* [Minimum = *tolerance_floor*]

Summary

Set the convergence tolerance for the linear solver. Setting CONVERGENCE TOLERANCE = AUTOMATIC enables an autonomous linear convergence tolerance calculation, which is computed from a combination of the initial linear residual and user-specified nonlinear residual tolerance. An additional floor value on the linear convergence tolerance can be declared with CONVERGENCE TOLERANCE = AUTOMATIC MINIMUM = <val>, where the default floor value is 1.0e-6

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>value</i>	string	–

Maximum Iterations

Syntax

Maximum Iterations *{=}* *value*

Summary

Set the maximum number of iterations taken by the linear solver

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>value</i>	integer	300

Recycled Blocks

Syntax

Recycled Blocks *{=}* *value*

Summary

Set the number of recycle blocks in GCRODR.

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>value</i>	integer	5

Residual Output

Syntax

Residual Output *{=}* [*{disabled | enabled}*]

Summary

Output the linear residual history to files in the residual_output/ directory, which is created automatically. For timestep NNN and linear system NAME, this will output a .csv file named residual_output/linear_solves_NAME.stepNNN. This file includes information about the linear residual at every iteration across all nonlinear iterations.

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–

Residual Scaling

Syntax

Residual Scaling *{=}* *{none | preconditioned_r0 | r0 | rhs}*

Summary

Set the residual scaling for the linear solver

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>value</i>	{ none preconditioned_r0 r0 rhs }	r0

Restart Iterations

Syntax

Restart Iterations *{=}* *value*

Summary

Set the number of iterations between GCRODR restarts.

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>value</i>	integer	30

Expert Belos Solver

Scope

Teko Block Solver, Teko Subblock Solver, Tpetra Equation Solver

Summary

Use a Belos solver with parameters specified in an XML file.

begin Expert Belos Solver

Xml File *{=}* *value*

```
begin Dd-Ilu Preconditioner
end
```

```
begin Dd-Ilut Preconditioner
end
```

```
begin Dd-Parilut Preconditioner
end
```

```
begin Expert Ifpack2 Preconditioner
end
```

```
begin Expert Teko Preconditioner
end
```

```
begin Fastilu Preconditioner
end
```

```
begin Jacobi Preconditioner
end
```

```
begin Muelu Preconditioner
end
```

```
begin Preset Preconditioner
end
```

```
begin Sgs Preconditioner
end
```

```
begin Sgs2 Preconditioner
end
```

```
begin Teko Preconditioner
end
```

end Expert Belos Solver

Line Commands

Xml File

Syntax

Xml File *{=}* *value*

Summary

Supply the name of an XML file containing a Belos parameter list.

Description

The supplied XML file must contain a ParameterList with a “solution method” string parameter corresponding to the desired type of Belos solver. Any additional parameters that should be passed on to Belos should be part of a nested ParameterList with name=”Belos-Tpetra”.

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>value</i>	string	–

Preset Solver

Scope

Teko Subblock Solver, Tpetra Equation Solver

Summary

Use a preset solver for a specific linear system type.

Description

The preset solver for a given linear system type is intended to provide reasonable behavior for many problems of that type. If this proves to be insufficient, the user must provide more fine-grained control over solver settings. The equivalent command which produces the solver will be output to the log file, and this may be used as a starting point from which changes for robustness may be made (i.e. increasing GMRES subspace size or changing the preconditioner).

begin Preset Solver

Convergence Tolerance `{=} value` [Minimum = `tolerance_floor`]

Maximum Iterations `{=} value`

Residual Output `{=} [{disabled | enabled}]`

Residual Scaling `{=} {none | preconditioned_r0 | r0 | rhs}`

Solver Type `{=} {continuity | high_aspect_continuity | multiphysics_`
`↪| scalar_transport | teko_multiphysics | teko_ns | thermal | thermal_`
`↪bicgstab | thermal_multigrid | thermal_symmetric}`

end Preset Solver

Line Commands

Convergence Tolerance

Syntax

Convergence Tolerance `{=} value` [Minimum = `tolerance_floor`]

Summary

Set the convergence tolerance for the linear solver. Setting CONVERGENCE TOLERANCE = AUTOMATIC enables an autonomous linear convergence tolerance calculation, which is computed from a combination of the initial linear residual and user-specified nonlinear residual tolerance. An additional floor value on the linear convergence tolerance can be declared with

CONVERGENCE TOLERANCE = AUTOMATIC MINIMUM = <val>, where the default floor value is 1.0e-6

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>value</i>	string	–

Maximum Iterations

Syntax

Maximum Iterations *{=}* *value*

Summary

Set the maximum number of iterations taken by the linear solver

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>value</i>	integer	300

Residual Output

Syntax

Residual Output *{=}* [*{disabled | enabled}*]

Summary

Output the linear residual history to files in the residual_output/ directory, which is created automatically. For timestep NNN and linear system NAME, this will output a .csv file named residual_output/linear_solves_NAME.stepNNN. This file includes information about the linear residual at every iteration across all nonlinear iterations.

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–

Residual Scaling

Syntax

Residual Scaling *{=}* *{none | preconditioned_r0 | r0 | rhs}*

Summary

Set the residual scaling for the linear solver

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>value</i>	{ none preconditioned_r0 r0 rhs }	r0

Solver Type

Syntax

Solver Type *{=}* *{continuity | high_aspect_continuity | multiphysics | scalar_transport | teko_multiphysics | teko_ns | thermal | thermal_bicgstab | thermal_multigrid | thermal_symmetric}*

Summary

Set the preset solver type

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>value</i>	{continuity high_aspect_continuity multiphysics scalar_transport teko_multiphysics teko_ns thermal thermal_bicgstab thermal_multigrid thermal_symmetric}	–

Preset Solver

Scope

Teko Block Solver, Teko Subblock Solver, Tpetra Equation Solver

Summary

Use a preset solver for a specific linear system type.

Description

The preset solver for a given linear system type is intended to provide reasonable behavior for many problems of that type. If this proves to be insufficient, the user must provide more fine-grained control over solver settings. The equivalent command which produces the solver will be output to the log file, and this may be used as a starting point from which changes for robustness may be made (i.e. increasing GMRES subspace size or changing the preconditioner).

begin Preset Solver

Convergence Tolerance *{=}* *value* [Minimum = *tolerance_floor*]

Maximum Iterations *{=}* *value*

Residual Output *{=}* [*{disabled | enabled}*]

Residual Scaling *{=}* *{none | preconditioned_r0 | r0 | rhs}*

Solver Type *{=}* *{continuity | high_aspect_continuity | multiphysics | scalar_transport | teko_multiphysics | teko_ns | thermal | thermal_*

→ *bicgstab | thermal_multigrid | thermal_symmetric*}

end Preset Solver

Line Commands

Convergence Tolerance

Syntax

Convergence Tolerance *{=}* *value* [Minimum = *tolerance_floor*]

Summary

Set the convergence tolerance for the linear solver. Setting CONVERGENCE TOLERANCE = AUTOMATIC enables an autonomous linear convergence tolerance calculation, which is computed from a combination of the initial linear residual and user-specified nonlinear residual tolerance. An additional floor value on the linear convergence tolerance can be declared with CONVERGENCE TOLERANCE = AUTOMATIC MINIMUM = <val>, where the default floor value is 1.0e-6

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>value</i>	string	–

Maximum Iterations

Syntax

Maximum Iterations *{=}* *value*

Summary

Set the maximum number of iterations taken by the linear solver

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>value</i>	integer	300

Residual Output

Syntax

Residual Output *{=}* [*{disabled | enabled}*]

Summary

Output the linear residual history to files in the residual_output/ directory, which is created automatically. For timestep NNN and linear system NAME, this will output a .csv file named residual_output/linear_solves_NAME.stepNNN. This

file includes information about the linear residual at every iteration across all nonlinear iterations.

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–

Residual Scaling

Syntax

Residual Scaling *{=}* *{none | preconditioned_r0 | r0 | rhs}*

Summary

Set the residual scaling for the linear solver

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>value</i>	{ none preconditioned_r0 r0 rhs }	r0

Solver Type

Syntax

Solver Type *{=}* *{continuity | high_aspect_continuity | multiphysics | scalar_transport | teko_multiphysics | teko_ns | thermal | thermal_bicgstab | thermal_multigrid | thermal_symmetric}*

Summary

Set the preset solver type

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>value</i>	{ continuity high_aspect_continuity multiphysics scalar_transport teko_multiphysics teko_ns thermal thermal_bicgstab thermal_multigrid thermal_symmetric }	–

Tpetra Preconditioners

Expert Teko Preconditioner

Scope

Expert Belos Solver, Gmres Solver

Summary

Use a teko preconditioner with parameters specified in an XML file.

begin Expert Teko Preconditioner

Define Matrix Subblock Number = *number* Dof = *dof name*

Inverse Method *{=}* *value*

Xml File *{=}* *value*

end Expert Teko Preconditioner

Line Commands

Define Matrix Subblock

Syntax

Define Matrix Subblock Number = *number* Dof = *dof name*

Summary

Define the dof associated with a matrix subblock for block preconditioning.

Description

Define the matrix subblock number and the equation to apply to that subblock.

Please see <https://trilinos.github.io/teko.html> for more details

Parameter	Value	Default
<i>number</i>	integer	–
<i>dof name</i>	string	–

Inverse Method

Syntax

Inverse Method *{=}* *value*

Summary

Supply the name of the inverse method.

Description

The name of the inverse method must correspond to the name of a parameter list specified in the teko XML file. Please see <https://trilinos.github.io/teko.html> for more details

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>value</i>	string	–

Xml File

Syntax

Xml File *{=}* *value*

Summary

Supply the name of an XML file containing a teko parameter list.

Description

The supplied XML file must contain a ParameterList with a valid teko parameter list. Please see <https://trilinos.github.io/teko.html> for more details

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>value</i>	string	–

Teko Preconditioner**Scope**

Expert Belos Solver, Gmres Solver, Teko Block Preconditioner

Summary

Use a teko preconditioner with parameters specified through the input deck.

begin Teko Preconditioner

Define Matrix Subblock Number = *number* Dof = *dof name*

```
begin Block Gauss-Seidel Solver
end
```

```
begin Block Jacobi Solver
end
```

```
begin Hierarchical Block Gauss-Seidel Solver
end
```

```
begin Simple Solver
end
```

```
begin Teko Block Preconditioner Solver Name
end
```

```
begin Teko Block Solver Solver Name
end
```

```
begin Teko Subblock Preconditioner Solver Name
```

end

begin Teko Subblock Solver *Solver Name*
end

end Teko Preconditioner

Line Commands

Define Matrix Subblock

Syntax

Define Matrix Subblock Number = *number* Dof = *dof name*

Summary

Define the dof associated with a matrix subblock for block preconditioning.

Description

Define the matrix subblock number and the equation to apply to that subblock.
Please see <https://trilinos.github.io/teko.html> for more details

Parameter	Value	Default
<i>number</i>	integer	–
<i>dof name</i>	string	–

Teko Subblock Solver

Scope

Teko Preconditioner

Summary

Define a solver to be used for one or more subblock(s)

begin Teko Subblock Solver *Solver Name*

begin Bicgstab Solver
end

begin Cg Solver
end

begin Expert Belos Solver
end

begin Gcrodr Solver

end

begin Gmres Solver
end

begin Klu2 Solver
end

begin Poly Solver
end

begin Preset Solver
end

begin Preset Solver
end

begin Superlu Solver
end

begin Umfpack Solver
end

end Teko Subblock Solver *Solver Name*

Teko Subblock Preconditioner

Scope

Teko Preconditioner

Summary

Define a preconditioner to be used to approximate the inverse of one or more subblock(s)

begin Teko Subblock Preconditioner *Solver Name*

begin Dd-Ilu Preconditioner
end

begin Dd-Ilut Preconditioner
end

begin Dd-Parilut Preconditioner
end

```
begin Expert Ifpack2 Preconditioner
end
```

```
begin Fastilu Preconditioner
end
```

```
begin Jacobi Preconditioner
end
```

```
begin Muelu Preconditioner
end
```

```
begin Preset Preconditioner
end
```

```
begin Sgs Preconditioner
end
```

```
begin Sgs2 Preconditioner
end
```

```
end Teko Subblock Preconditioner Solver Name
```

Teko Block Solver

Scope

Teko Preconditioner

Summary

Define a solver to be used to approximate the inverse of a set of subblocks in a hierarchical block Gauss-Seidel scheme.

```
begin Teko Block Solver Solver Name
```

```
begin Expert Belos Solver
end
```

```
begin Gmres Solver
end
```

```
begin Preset Solver
end
```

```
end Teko Block Solver Solver Name
```

Teko Block Preconditioner

Scope

Teko Preconditioner

Summary

Define a preconditioner to be used to approximate the inverse of a set of subblocks in a hierarchical block Gauss-Seidel scheme.

begin Teko Block Preconditioner *Solver Name*

begin Preset Preconditioner
end

begin Teko Preconditioner
end

end Teko Block Preconditioner *Solver Name*

Hierarchical Block Gauss-Seidel Solver

Scope

Teko Preconditioner

Summary

Use a Hierarchical Block Gauss-Seidel solver inside a Teko preconditioner.

begin Hierarchical Block Gauss-Seidel Solver

Use Block Inverse *block inverse* For Hierarchical Block *number*

Use Subblocks *subblocks...* For Hierarchical Block *number*

Use Upper Triangle *{=}* *{false | true}*

end Hierarchical Block Gauss-Seidel Solver

Line Commands

Use Block Inverse

Syntax

Use Block Inverse *block inverse* For Hierarchical Block *number*

Summary

Define the block solver to be used for a given hierarchical block.

Parameter	Value	Default
<i>block inverse</i>	string	–
<i>number</i>	integer	–

Use Subblocks

Syntax

Use Subblocks *subblocks. . .* For Hierarchical Block *number*

Summary

Map subblock(s) to a hierarchical block, which is a set containing one or more blocks.

Parameter	Value	Default
<i>subblocks</i>	integer. . .	–
<i>number</i>	integer	–

Use Upper Triangle

Syntax

Use Upper Triangle *{=}* *{false | true}*

Summary

Define whether to use upper triangular hierarchical block Gauss-Seidel

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>value</i>	{false true}	FALSE

Muelu Preconditioner

Scope

Bicgstab Solver, Cg Solver, Expert Belos Solver, Gcrodr Solver, Gmres Solver, Poly Solver, Teko Subblock Preconditioner

Summary

Use a MueLu preconditioner.

begin Muelu Preconditioner

Xml File `{=}` *value*

end Muelu Preconditioner

Line Commands

Xml File

Syntax

Xml File `{=}` *value*

Summary

Supply the name of an XML file containing a MueLu parameter list.

Parameter	Value	Default
<code>{=}</code>	{= are is }	–
<i>value</i>	string	–

Jacobi Preconditioner

Scope

Bicgstab Solver, Cg Solver, Expert Belos Solver, Gcrodr Solver, Gmres Solver, Poly Solver, Teko Subblock Preconditioner

Summary

Use a Jacobi preconditioner.

begin Jacobi Preconditioner

Damping Factor `{=}` *value*

Number Of Sweeps `{=}` *value*

end Jacobi Preconditioner

Line Commands

Damping Factor

Syntax

Damping Factor `{=} value`

Summary

Set the relaxation damping factor.

Parameter	Value	Default
<code>{=}</code>	{= are is }	–
<code>value</code>	real	1

Number Of Sweeps

Syntax

Number Of Sweeps `{=} value`

Summary

Set the number of relaxation sweeps to perform.

Parameter	Value	Default
<code>{=}</code>	{= are is }	–
<code>value</code>	integer	3

Sgs2 Preconditioner

Scope

Bicgstab Solver, Cg Solver, Expert Belos Solver, Gcrodr Solver, Gmres Solver, Poly Solver, Teko Subblock Preconditioner

Summary

Use a Two-stage symmetric Gauss-Seidel preconditioner.

begin Sgs2 Preconditioner

Damping Factor `{=} value`

Number Of Sweeps `{=} value`

end Sgs2 Preconditioner

Line Commands

Damping Factor

Syntax

Damping Factor **{=}** *value*

Summary

Set the relaxation damping factor.

Parameter	Value	Default
{=}	{= are is }	–
<i>value</i>	real	1

Number Of Sweeps

Syntax

Number Of Sweeps **{=}** *value*

Summary

Set the number of relaxation sweeps to perform.

Parameter	Value	Default
{=}	{= are is }	–
<i>value</i>	integer	3

Sgs Preconditioner

Scope

Bicgstab Solver, Cg Solver, Expert Belos Solver, Gcrodr Solver, Gmres Solver, Poly Solver, Teko Subblock Preconditioner

Summary

Use a symmetric Gauss-Seidel preconditioner.

begin Sgs Preconditioner

Damping Factor **{=}** *value*

Number Of Sweeps **{=}** *value*

end Sgs Preconditioner

Line Commands

Damping Factor

Syntax

Damping Factor `{=} value`

Summary

Set the relaxation damping factor.

Parameter	Value	Default
<code>{=}</code>	<code>{= are is}</code>	–
<code>value</code>	real	1

Number Of Sweeps

Syntax

Number Of Sweeps `{=} value`

Summary

Set the number of relaxation sweeps to perform.

Parameter	Value	Default
<code>{=}</code>	<code>{= are is}</code>	–
<code>value</code>	integer	3

Dd-Ilu Preconditioner

Scope

Bicgstab Solver, Cg Solver, Expert Belos Solver, Gcrodr Solver, Gmres Solver, Poly Solver, Teko Subblock Preconditioner

Summary

Use a domain-decomposition ILU(k) preconditioner.

begin Dd-Ilu Preconditioner

Fill Level `{=} value`

Number Of Gpu Streams `{=} value`

Reordering `{=} {metis | off | on | rcm}`

Subdomain Overlap Level `{=} value`

Use Thread Parallel Implementation `{=}` `{off | on}`

end Dd-Ilu Preconditioner

Line Commands

Fill Level

Syntax

Fill Level `{=}` `value`

Summary

Set the level of fill, k, for ILU(k).

Parameter	Value	Default
<code>{=}</code>	<code>{= are is}</code>	–
<code>value</code>	integer	0

Number Of Gpu Streams

Syntax

Number Of Gpu Streams `{=}` `value`

Summary

DEVELOPER COMMAND: Set the number of GPU streams to use in GPU builds.

Description

On each MPI rank the local matrix will be further decomposed into this number of diagonal sub-matrices and the ILU preconditioner will be applied to each diagonal sub-matrix in parallel using GPU streams.

Parameter	Value	Default
<code>{=}</code>	<code>{= are is}</code>	–
<code>value</code>	integer	0

Reordering

Syntax

Reordering `{=}` `{metis | off | on | rcm}`

Summary

Choose the reordering method to apply to the local matrices.

Description

By default reordering is ON which corresponds to the reordering method that typically performs best. In CPU builds the default reorder method is RCM, but

in GPU builds it is METIS. METIS usually requires slightly higher iteration counts than RCM, but provides more parallelism for ILU on the GPU and usually runs faster on those machines as a result. If necessary REORDERING = RCM|METIS can be used to ensure the same reordering method is used regardless of machine type.

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>value</i>	{metis off on rcm}	ON

Subdomain Overlap Level

Syntax

Subdomain Overlap Level *{=}* *value*

Summary

Set the level of subdomain overlap used in constructing the preconditioner.

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>value</i>	integer	0

Use Thread Parallel Implementation

Syntax

Use Thread Parallel Implementation *{=}* *{off | on}*

Summary

Control whether or not to use the thread parallel ILU implementation from Kokkos-Kernels.

Description

This will default to the appropriate value based on the platform Aria is built for, but in rare cases it may be helpful to disable the thread parallel implementation in a GPU build for debugging purposes.

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>value</i>	{off on}	ON

Dd-Ilut Preconditioner

Scope

Bicgstab Solver, Cg Solver, Expert Belos Solver, Gcrodr Solver, Gmres Solver, Poly Solver, Teko Subblock Preconditioner

Summary

Use a domain-decomposition ILUT preconditioner.

begin Dd-Ilut Preconditioner

Drop Tolerance `{=}` *value*

Fill Fraction `{=}` *value*

Reordering `{=}` `{metis | off | on | rcm}`

Subdomain Overlap Level `{=}` *value*

end Dd-Ilut Preconditioner

Line Commands

Drop Tolerance

Syntax

Drop Tolerance `{=}` *value*

Summary

Set the drop tolerance for ilut.

Description

Off-diagonal entries of the incomplete factorization smaller than the specified tolerance will be dropped.

Parameter	Value	Default
<code>{=}</code>	<code>{= are is}</code>	–
<i>value</i>	real	0

Fill Fraction

Syntax

Fill Fraction `{=}` *value*

Summary

Set the fill fraction for ilut.

Description

This parameter controls the number of nonzero entries that are kept in each row of the incomplete factorization. Larger values will use more memory and runtime, but may result in a more effective preconditioner.

Parameter	Value	Default
<i>{=}</i>	{= are is }	–
<i>value</i>	real	1

Reordering

Syntax

Reordering *{=}* *{metis | off | on | rcm}*

Summary

Choose the reordering method to apply to the local matrices.

Description

By default reordering is ON which corresponds to the reordering method that typically performs best. In CPU builds the default reorder method is RCM, but in GPU builds it is METIS. METIS usually requires slightly higher iteration counts than RCM, but provides more parallelism for ILU on the GPU and usually runs faster on those machines as a result. If necessary REORDERING = RCM|METIS can be used to ensure the same reordering method is used regardless of machine type.

Parameter	Value	Default
<i>{=}</i>	{= are is }	–
<i>value</i>	{metis off on rcm}	ON

Subdomain Overlap Level

Syntax

Subdomain Overlap Level *{=}* *value*

Summary

Set the level of subdomain overlap used in constructing the preconditioner.

Parameter	Value	Default
<i>{=}</i>	{= are is }	–
<i>value</i>	integer	0

Dd-Parilut Preconditioner

Scope

Bicgstab Solver, Cg Solver, Expert Belos Solver, Gcrodr Solver, Gmres Solver, Poly Solver, Teko Subblock Preconditioner

Summary

Use a domain-decomposition ParILUT preconditioner.

Description

ParILUT is similar to ILUT, but uses an iterative approach to compute the approximate L and U factors in order to enable fine grained parallelism that can improve performance on GPUs.

begin Dd-Parilut Preconditioner

Fill Fraction `{=}` *value*

Maximum Iterations `{=}` *value*

Reordering `{=}` `{metis | off | on | rcm}`

Subdomain Overlap Level `{=}` *value*

end Dd-Parilut Preconditioner

Line Commands

Fill Fraction

Syntax

Fill Fraction `{=}` *value*

Summary

Set the fill level for par_ilut.

Description

This parameter controls the number of nonzero entries that are kept in each row of the incomplete factorization. Larger values will use more memory and runtime, but may result in a more effective preconditioner.

Parameter	Value	Default
<code>{=}</code>	<code>{= are is}</code>	–
<i>value</i>	real	1

Maximum Iterations

Syntax

Maximum Iterations *{=}* *value*

Summary

Set the maximum number of iterations of the par_ilut algorithm.

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>value</i>	integer	5

Reordering

Syntax

Reordering *{=}* *{metis | off | on | rcm}*

Summary

Choose the reordering method to apply to the local matrices.

Description

By default reordering is ON which corresponds to the reordering method that typically performs best. In CPU builds the default reorder method is RCM, but in GPU builds it is METIS. METIS usually requires slightly higher iteration counts than RCM, but provides more parallelism for ILU on the GPU and usually runs faster on those machines as a result. If necessary REORDERING = RCM|METIS can be used to ensure the same reordering method is used regardless of machine type.

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>value</i>	{metis off on rcm}	ON

Subdomain Overlap Level

Syntax

Subdomain Overlap Level *{=}* *value*

Summary

Set the level of subdomain overlap used in constructing the preconditioner.

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>value</i>	integer	0

Fastilu Preconditioner

Scope

Bicgstab Solver, Cg Solver, Expert Belos Solver, Gcrodr Solver, Gmres Solver, Poly Solver, Teko Subblock Preconditioner

Summary

Use a domain-decomposition FastILU(k) preconditioner.

Description

FastILU is a beta preconditioner that uses an iterative approximation to the standard DD-ILU preconditioner in order to expose more parallelism for potentially better performance on GPUs.

begin Fastilu Preconditioner

Damping Factor `{=}` *value*

Fill Level `{=}` *value*

Reordering `{=}` `{metis | off | on | rcm}`

Subdomain Overlap Level `{=}` *value*

Sweeps `{=}` *value*

Triangular Solver Type `{=}` `{fast | standard}`

end Fastilu Preconditioner

Line Commands

Damping Factor

Syntax

Damping Factor `{=}` *value*

Summary

Set the damping factor for the FastILU algorithm.

Parameter	Value	Default
<code>{=}</code>	<code>{= are is}</code>	–
<i>value</i>	real	0.2

Fill Level

Syntax

Fill Level *{=}* *value*

Summary

Set the level of fill, k, for FastILU(k).

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>value</i>	integer	0

Reordering

Syntax

Reordering *{=}* *{metis | off | on | rcm}*

Summary

Choose the reordering method to apply to the local matrices.

Description

By default reordering is ON which corresponds to the reordering method that typically performs best. In CPU builds the default reorder method is RCM, but in GPU builds it is METIS. METIS usually requires slightly higher iteration counts than RCM, but provides more parallelism for ILU on the GPU and usually runs faster on those machines as a result. If necessary REORDERING = RCM|METIS can be used to ensure the same reordering method is used regardless of machine type.

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>value</i>	{metis off on rcm}	ON

Subdomain Overlap Level

Syntax

Subdomain Overlap Level *{=}* *value*

Summary

Set the level of subdomain overlap used in constructing the preconditioner.

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>value</i>	integer	0

Sweeps

Syntax

Sweeps *{=}* *value*

Summary

Set the number of sweeps for iteratively computing the ILU approximation via the FastILU algorithm.

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>value</i>	integer	5

Triangular Solver Type

Syntax

Triangular Solver Type *{=}* *{fast | standard}*

Summary

Set the triangular solver type for FastILU

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>value</i>	{ fast standard }	Fast

Expert Ifpack2 Preconditioner

Scope

Bicgstab Solver, Cg Solver, Expert Belos Solver, Gcrodr Solver, Gmres Solver, Poly Solver, Teko Subblock Preconditioner

Summary

Use an Ifpack2 preconditioner with parameters specified in an XML file.

begin Expert Ifpack2 Preconditioner

Xml File *{=}* *value*

end Expert Ifpack2 Preconditioner

Line Commands

Xml File

Syntax

Xml File *{=}* *value*

Summary

Supply the name of an XML file containing an Ifpack2 parameter list.

Description

The supplied XML file must contain a ParameterList with a “preconditioner type:” string parameter corresponding to the desired type of Ifpack2 preconditioner as well as any additional parameters for that preconditioner.

Parameter	Value	Default
<i>{=}</i>	{= are is }	–
<i>value</i>	string	–

Preset Preconditioner

Scope

Bicgstab Solver, Cg Solver, Expert Belos Solver, Gcrodr Solver, Gmres Solver, Poly Solver, Teko Subblock Preconditioner

Summary

Use a preset preconditioner for a specific linear system type.

Description

The preset preconditioner for a given linear system type is intended to provide reasonable behavior for many problems of that type. If this proves to be insufficient, the user must provide more fine-grained control over preconditioner settings. The equivalent command which produces the preconditioner will be output to the log file, and this may be used as a starting point from which changes for robustness may be made (i.e. increasing Chebyshev degree in a MueLu preconditioner).

begin Preset Preconditioner

```
Preconditioner Type {=} {continuity | high_aspect_continuity |  
↪multiphysics | scalar_transport | teko_multiphysics | teko_ns | thermal_  
↪| thermal_multigrid}
```

end Preset Preconditioner

Line Commands

Preconditioner Type

Syntax

Preconditioner Type {=} {continuity | high_aspect_continuity | multiphysics |
scalar_transport | teko_multiphysics | teko_ns | thermal | thermal_multigrid}

Summary

Set the preset preconditioner type

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>value</i>	{ continuity high_aspect_continuity multiphysics scalar_transport teko_multiphysics teko_ns thermal thermal_multigrid }	–

Preset Preconditioner

Scope

Teko Block Preconditioner

Summary

Use a preset preconditioner for a specific linear system type.

Description

The preset preconditioner for a given linear system type is intended to provide reasonable behavior for many problems of that type. If this proves to be insufficient, the user must provide more fine-grained control over preconditioner settings. The equivalent command which produces the preconditioner will be output to the log file, and this may be used as a starting point from which changes for robustness may be made (i.e. increasing Chebyshev degree in a MueLu preconditioner).

begin Preset Preconditioner

```
Preconditioner Type {=} {continuity | high_aspect_continuity |
↳multiphysics | scalar_transport | teko_multiphysics | teko_ns | thermal
↳| thermal_multigrid}
```

end Preset Preconditioner

Line Commands

Preconditioner Type

Syntax

```
Preconditioner Type {=} {continuity | high_aspect_continuity | multiphysics |
scalar_transport | teko_multiphysics | teko_ns | thermal | thermal_multigrid}
```

Summary

Set the preset preconditioner type

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>value</i>	{continuity high_aspect_continuity multiphysics scalar_transport teko_multiphysics teko_ns thermal thermal_multigrid}	–

Block Gauss-Seidel Solver

Scope

Teko Preconditioner

Summary

Use a Block Gauss-Seidel solver inside a Teko preconditioner.

begin Block Gauss-Seidel Solver

Use Default Inverse *subblock inverse* For Subblocks

Use Inverse *subblock inverse* For Subblock *number*

Use Reordering Heuristic *{=}* *{loss_minimizing | none}*

Use Upper Triangle *{=}* *{false | true}*

end Block Gauss-Seidel Solver

Line Commands

Use Default Inverse

Syntax

Use Default Inverse *subblock inverse* For Subblocks

Summary

Define a default solver/preconditioner to be used for across subblocks.

Parameter	Value	Default
<i>subblock inverse</i>	string	–

Use Inverse

Syntax

Use Inverse *subblock inverse* For Subblock *number*

Summary

Define the solver/preconditioner to be used for a given subblock.

Parameter	Value	Default
<i>subblock inverse</i>	string	–
<i>number</i>	integer	–

Use Reordering Heuristic

Syntax

Use Reordering Heuristic *{=}* *{loss_minimizing | none}*

Summary

DEPRECATED: This command no longer has any effect.

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>value</i>	{ loss_minimizing none }	NONE

Use Upper Triangle

Syntax

Use Upper Triangle *{=}* *{false | true}*

Summary

Define whether to use upper triangular block Gauss-Seidel

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>value</i>	{ false true }	FALSE

Block Jacobi Solver

Scope

Teko Preconditioner

Summary

Use a Block Jacobi solver inside a Teko preconditioner.

begin Block Jacobi Solver

Use Default Inverse *subblock inverse* For Subblocks

Use Inverse *subblock inverse* For Subblock *number*

end Block Jacobi Solver

Line Commands

Use Default Inverse

Syntax

Use Default Inverse *subblock inverse* For Subblocks

Summary

Define a default solver/preconditioner to be used for across subblocks.

Parameter	Value	Default
<i>subblock inverse</i>	string	–

Use Inverse

Syntax

Use Inverse *subblock inverse* For Subblock *number*

Summary

Define the solver/preconditioner to be used for a given subblock.

Parameter	Value	Default
<i>subblock inverse</i>	string	–
<i>number</i>	integer	–

Simple Solver

Scope

Teko Preconditioner

Summary

Use a SIMPLE/SIMPLEC solver inside a Teko preconditioner.

begin Simple Solver

Alpha {=} *value*

Explicit Velocity Inverse Type {=} {*absrowsum* | *diagonal* | *lumped*}

Use Default Inverse *subblock inverse* For Subblocks

Use Inverse *subblock inverse* For Subblock *number*

end Simple Solver

Line Commands

Alpha

Syntax

Alpha *{=}* *value*

Summary

Set alpha value in SIMPLE block decomposition.

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>value</i>	real	1

Explicit Velocity Inverse Type

Syntax

Explicit Velocity Inverse Type *{=}* *{absrowsum | diagonal | lumped}*

Summary

Set inverse type used for velocity term when forming the Pressure Schur complement operator.

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>value</i>	{absrowsum diagonal lumped}	ABSRROWSUM

Use Default Inverse

Syntax

Use Default Inverse *subblock inverse* For Subblocks

Summary

Define a default solver/preconditioner to be used for across subblocks.

Parameter	Value	Default
<i>subblock inverse</i>	string	–

Use Inverse

Syntax

Use Inverse *subblock inverse* For Subblock *number*

Summary

Define the solver/preconditioner to be used for a given subblock.

Parameter	Value	Default
<i>subblock inverse</i>	string	–
<i>number</i>	integer	–

7.4 Procedure

Fuego Procedure

Scope

Sierra

Summary

Contains the commands needed to execute an analysis in this procedure.

begin Fuego Procedure *ProcedureName*

Debug Level *{=} Level*

Disable Conjugate Heat Transfer Symmetry Checks

Maximum Number Of Nonlinear Iterations *{=} Number*

Skip Steps For Pmr *{=} PMRSkip*

Time Start *{=} StartTime* Stop *{=} StopTime* Status Interval *{=} ↵
↵StatusInterval*

begin Average Region *Regionname*
end

begin Fuego Region *Regionname*
end

begin Input_Output Region *Parameter_block_name*
end

begin Particle Region *Regionname*
end

begin Solution Control Description *Name*

end

begin Time Control
end

begin Transfer *Transfer_name*
end

end Fuego Procedure *ProcedureName*

Line Commands

Debug Level

Syntax

Debug Level {=} *Level*

Summary

Set the debug level.

Description

Need description here.

Parameter	Value	Default
{=}	{= are is }	–
<i>Level</i>	integer	–

Disable Conjugate Heat Transfer Symmetry Checks

Syntax

Disable Conjugate Heat Transfer Symmetry Checks

Summary

Disables conjugate heat transfer symmetry checks.

Description

Surface transfers for conjugate heat transfer are symmetric. For example, consider a Dirichlet-Neumann coupling scheme. A transfer from Region A sends the temperature field on surfaces “s1, s2” to be interpolated on surfaces “surf1, surf2” of Region B. If Region B sends the heat flux field only from surface “surf1” to be interpolated on surfaces “s1, s2” of Region A, then this transfer is said to be asymmetric. To be symmetric, Region B needs to send surfaces “surf1, surf2” to surfaces “s1, s2” of Region A. Expert users can bypass this default behavior by activating this flag to ignore checks which would normally throw an error when asymmetric surface transfers are detected. A warning will be printed to the log file.

Maximum Number Of Nonlinear Iterations

Syntax

Maximum Number Of Nonlinear Iterations **{=}** *Number*

Summary

Set the number of procedure nonlinear iterations over the defined regions.

Description

Need description here.

Parameter	Value	Default
{=}	{ = are is }	–
<i>Number</i>	integer	1

Skip Steps For Pmr

Syntax

Skip Steps For Pmr **{=}** *PMRSkip*

Summary

Controls the frequency of PMR solve.

Description

Set the PMR solve interval. Default is 1 (every step). This can also be specified as a time block property. If both are specified, the time block property will be used.

Parameter	Value	Default
{=}	{ = are is }	–
<i>PMRSkip</i>	integer	1

Time Start

Syntax

Time Start **{=}** *StartTime* Stop **{=}** *StopTime* Status Interval **{=}** *StatusInterval*

Summary

Start and stop times for a procedure.

Parameter	Value	Default
{=}	{ = are is }	–
<i>StartTime</i>	real	–
{=}	{ = are is }	–
<i>StopTime</i>	real	–
{=}	{ = are is }	–
<i>StatusInterval</i>	integer	–

Apub Transfer

Summary

transfer region/mesh information. The variables information will get sorted out by the calling procedure.

begin Apub Transfer *Transfer_name*

Nodes Outside Region *Region name*

Surface Gap Tolerance {=} *Gap*

Transfer Between Fuego Fluid Region *From_region_name* {surface | volume}
→ *From_io_entity_name(s)* And Fuego Conduction Region *To_region_name*
→ {surface | volume} *To_io_entity_name(s)* [Using {copy | interp}]

Transfer Between Input Region *From_region_name* And Balance Region *To_*
→ *region_name* [Using {copy | interp}]

Transfer Between Region *From_region_name* {surface | volume} *From_*
→ *io_entity_name(s)* And Region *To_region_name* {surface | volume} *To_io_*
→ *entity_name(s)* [Using {copy | interp}]

Transfer Problem Definition {=} *Problem...*

end Apub Transfer *Transfer_name*

Line Commands

Nodes Outside Region

Syntax

Nodes Outside Region *Region name*

Summary

deal with the nodes that fall outside the intersection of two regions for nodal interpolation

Parameter	Value	Default
<i>Region name</i>	string	–

Surface Gap Tolerance

Syntax

Surface Gap Tolerance {=} *Gap*

Summary

specify the gap tolerance for locating interacting nodes during surface transfers.

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>Gap</i>	real	–

Transfer Between Fuego Fluid Region

Syntax

Transfer Between Fuego Fluid Region *From_region_name* {*surface* | *volume*}
From_io_entity_name(s) And Fuego Conduction Region *To_region_name*
{*surface* | *volume*} *To_io_entity_name(s)* [Using {*copy* | *interp*}]

Summary

transfer from fuego fluids region to fuego heat conduction region

Parameter	Value	Default
<i>From_region_name</i>	string	–
<i>Option1</i>	{ surface volume }	–
<i>From_io_entity_name(s)</i>	string	–
<i>To_region_name</i>	string	–
<i>Option2</i>	{ surface volume }	–
<i>To_io_entity_name(s)</i>	string	–

Transfer Between Input Region

Syntax

Transfer Between Input Region *From_region_name* And Balance Region
To_region_name [Using {*copy* | *interp*}]

Summary

Transfer mesh from source region region to destination region. The mesh transfer uses the specified load-balancing algorithm. The default transfer operator is the Nodal Copy, and can also be specified by including the optional parameters ‘USING COPY’. If ‘USING INTERP’ is specified, then the (general different-mesh) nodal interpolation transfer is used instead. The latter is intended to be used only for testing the nodal interpolation transfer operator.

Parameter	Value	Default
<i>From_region_name</i>	string	–
<i>To_region_name</i>	string	–

Transfer Between Region

Syntax

Transfer Between Region *From_region_name* {*surface* | *volume*}
From_io_entity_name(s) And Region *To_region_name* {*surface* | *volume*}
To_io_entity_name(s) [Using {*copy* | *interp*}]

Summary

transfer from region/block to region/block

Parameter	Value	Default
<i>From_region_name</i>	string	–
<i>Option1</i>	{ surface volume }	–
<i>From_io_entity_name(s)</i>	string	–
<i>To_region_name</i>	string	–
<i>Option2</i>	{ surface volume }	–
<i>To_io_entity_name(s)</i>	string	–

Transfer Problem Definition

Syntax

Transfer Problem Definition {=} *Problem* . .

Summary

problem type for this transfer

Parameter	Value	Default
{=}	{ = are is }	–
<i>Problem</i>	string . .	–

Transfer

Scope

Fuego Procedure

Summary

Transfer region/mesh information. The mechanics/variables information will get sorted out by the calling procedure.

begin Transfer *Transfer_name*

Abort If Field Not Defined On Copy Transfer Send Or Receive Object

Abort If Search Object Outside Of Tolerance

All Fields

Copy {surface | volume} {constraints | elements | nodes} From From_
→region_name To To_region_name

Distance Function Is Closest Receive Node To Send Centroid

Exclude Ghosted

From {constraints | elements | nodes} To {constraints | elements |
→faces | gauss_points | nodes}

Gauss Point Integration Order {=} Order

Geometric Tolerance {=} Geometric_tolerance

Inspect With File {=} File_name Ids {=} ID_list...

Interpolate {surface | volume} {constraints | elements | nodes} From_
→From_region_name To To_region_name

Interpolation Function Function_Name

Nodes Outside Region {=} {abort | extrapolate | ignore | project |
→truncate}

Parametric Tolerance {=} Parametric_tolerance

Patch Recovery Evaluation {=} {linear least squares | linear moving_
→least squares | quadratic least squares | quadratic moving least_
→squares}

Search Coordinate Field Source_field_name State {new | nm1 | nm2 | nm3_
→| nm4 | none | old} To Destination_field_name State {new | nm1 | nm2 |
→nm3 | nm4 | none | old}

Search Geometric Tolerance {=} Geometric_tolerance

Search Surface Gap Tolerance {=} Surface_gap_tolerance [Or Less]

Search Type {=} [{detailed | parallel | proximity} {detailed |
→parallel | proximity} {detailed | parallel | proximity}]

Select One Receiver For Each Send Object

Select One Unique Receiver For Each Send Object

Send {} Fields

Send Block *From_blocks...* To *To_blocks...*

Send Field *Source_field_name* State {*new* | *nm1* | *nm2* | *nm3* | *nm4* | *none* | *old*} To *Destination_field_name* State {*new* | *nm1* | *nm2* | *nm3* | *nm4* | *none* | *old*} [Lower Bound *Lower_bound* Upper Bound *Upper_bound*]

Toggle Search Warnings {=} {*false* | *no* | *off* | *on* | *true* | *yes*}

Use Centroid For Geometric Proximity

begin Receive Blocks
end

begin Send Blocks
end

end Transfer *Transfer_name*

Line Commands

Abort If Field Not Defined On Copy Transfer Send Or Receive Object

Syntax

Abort If Field Not Defined On Copy Transfer Send Or Receive Object

Summary

For testing purposes only. Normally mesh objects in the send or receive mesh which do not have the specified field defined on them are just ignored. This line command allows the construction of tests in which it is known that every mesh object should have the specified field defined on it and to abort if that field is not found.

Abort If Search Object Outside Of Tolerance

Syntax

Abort If Search Object Outside Of Tolerance

Summary

For debugging purposes only. Abort transfer if search object lie outside of specified geometric tolerance.

This command is deprecated. Use “Nodes outside region = abort” instead.

All Fields

Syntax

All Fields

Summary

Select all fields for transfer that have same name and state for source and destination regions.

Copy

Syntax

Copy *{surface | volume}* *{constraints | elements | nodes}* From *From_region_name* To *To_region_name*

Summary

Copy transfer elements, nodes or constraints from one region to another. The copy transfer is very specific in that the sending and receiving mesh parts must have identical global ids for every element to be copied. The copy transfer works by iterating over all the mesh objects in the receiving mesh and using the global id of the receiving mesh object to find a mesh object in the sending mesh with the same global id. The field to transfer is then copied from the sending to receiving objects. There is no interpolation and the actual coordinates of the sending and receiving objects are not used and could be very different. The copy transfer is used in very special cases where the same mesh was read into both the sending and receiving meshes, there was no element death and there was no adaptivity. In this special case, a copy transfer can be much faster than an interpolation transfer.

Parameter	Value	Default
<i>Option1</i>	{ surface volume }	—
<i>Option2</i>	{ constraints elements nodes }	—
<i>From_region_name</i>	string	—
<i>To_region_name</i>	string	—

Distance Function Is Closest Receive Node To Send Centroid

Syntax

Distance Function Is Closest Receive Node To Send Centroid

Summary

To be used in conjunction with “SELECT ONE UNIQUE RECEIVER FOR EACH SEND OBJECT”. This helped in the case where the sending and receiving element blocks did not overlap and an element transfer was using element centroids for the distance computation. The elements were very distorted so that a centroid of a surface element could be far from the surface. It was wanted that the receiving element be the one close to the surface of the block and close to the sending element in the adjacent block. Using the corner nodes was enough since it was a tet mesh with plane faces. In this particular and

unusual case this alternative method of matching sending and receiving elements was useful, but it is not expected to be used often or maybe never again.

Exclude Ghosted

Syntax

Exclude Ghosted

Summary

exclude ghosted nodes from a copy transfer

From

Syntax

From *{constraints | elements | nodes}* To *{constraints | elements | faces | gauss_points | nodes}*

Summary

Allows the send/receive mesh objects to be different. For a volume element field transfer (e.g. shell) to a face rank field on a surface, use ‘from elements to faces’.

Parameter	Value	Default
<i>Option1</i>	{ constraints elements nodes }	–
<i>Option2</i>	{ constraints elements faces gauss_points nodes }	–

Gauss Point Integration Order

Syntax

Gauss Point Integration Order *{=} Order*

Summary

Integration order to use when transferring to Gauss points.

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>Order</i>	integer	–

Geometric Tolerance

Syntax

Geometric Tolerance *{=} Geometric_tolerance*

Summary

This is the dimensional tolerance applied during the initial (coarse) search. If specified, all the coarse search boxes are padded by this value. The default behavior is for this to be 1e-9 times the problem bounding box size (diagonal) plus a 10 percent relative expansion per element. If a value is specified for this, it is used as a padding in place of the default with no relative box expansion.

During the interpolation transfer there is a geometric search based on the coordinates of the send and receive objects. As part of this search, an axis aligned bounding box is contracted for each sending object and GEOMETRIC TOLERANCE is used to make this box bigger than just a tight bounding box. Lists of receiving points are then quickly found within these axis aligned boxes.

If all points in the receiving mesh are within at least one box, no additional searching needs to be done and the search algorithm is fast. If there are still points in the receiving mesh that were outside of EVERY box, then a warning message will be issued about an “expensive search for extrapolation” for these points. This ‘expensive search’ can be very costly if a large number of receiving objects fall into this category and this line command is provided for those special cases.

Parameter	Value	Default
<i>{=}</i>	{= are is }	–
<i>Geometric_tolerance</i>	real	–

Inspect With File

Syntax

Inspect With File *{=}* *File_name* Ids *{=}* *ID_list* . .

Summary

STK Transfer inspection tool that allows user to output the search results for a specified list of entity ids on the receive mesh. The rank of the entities is deduced from the type of transfer being set up.

Parameter	Value	Default
<i>{=}</i>	{= are is }	–
<i>File_name</i>	string	–
<i>{=}</i>	{= are is }	–
<i>ID_list</i>	integer. . .	–

Interpolate

Syntax

Interpolate *{surface | volume}* *{constraints | elements | nodes}* From *From_region_name* To *To_region_name*

Summary

Interpolate will transfer elements, nodes or constraints from one mesh to another. The interpolation transfer is very general in that the field values to transfer will be interpolated from the sending to receiving mesh based on the coordinates of the sending and receiving mesh objects.

Many line commands can be used to modify the behavior of the interpolation transfer but the basic algorithm is straightforward. Every mesh object in the receiving mesh is converted into a point. For elements this is the average of the nodal coordinates. An element in the sending mesh containing this point is found. If the field to transfer is nodal, the element shape functions are used to interpolate the nodal field to the receiving point. If the field to transfer is elemental, a bi-linear least squares fit based upon neighboring elements is first performed and then used to define the interpolation of the element field at the receiving point.

Parameter	Value	Default
<i>Option1</i>	{ surface volume }	–
<i>Option2</i>	{ constraints elements nodes }	–
<i>From_region_name</i>	string	–
<i>To_region_name</i>	string	–

Interpolation Function

Syntax

Interpolation Function *Function_Name*

Summary

Allows an application defined subroutine to be used for the interpolation. Normally the interpolation transfer will determine the best type of interpolation to use for node or element entities: Basis functions for nodal fields and a neighborhood least squares fit for element fields. This command line can be used to override the default behavior if needed. Available interpolation functions are Sum, Copy, Master_Element, Element_Centroid_Constant and Element_Centroid_Linear.

Sum - Sum surrounding entity values on source and send to destination.

Copy - Copy entity values directly from source to destination.

Master_Element - Send interpolated nodal values from nearest source element to destination.

Element_Centroid_Constant - For nodal field transfer send computed centroid value from the nearest source element to destination. For element field transfer send the element field value from the nearest source element to the destination

Element_Centroid_Linear - Send values of interpolated centroid values obtained using patch recovery to destination. Patch recovery involves a least squares reconstruction and evaluation using a patch of elements surrounding the nearest source element. For linear and higher-order reconstructions one should consult the Patch Recovery Evaluation option.

Parameter	Value	Default
<i>Function_Name</i>	string	–

Nodes Outside Region

Syntax

Nodes Outside Region *{=}* *{abort | extrapolate | ignore | project | truncate}*

Summary

This line command defines what to do when a receiving point is outside the scope of the sending mesh.

IGNORE - The receiving mesh object can be ignored and will receive no value. This is almost never a good idea as it can cause mesh objects just outside to have a zero value when the nodes just inside the mesh might have very large values. This can result in a discontinuous receiving field.

EXTRAPOLATE - This is the default behavior. The sending field is extrapolated beyond the bounds of the sending mesh. This can lead to extrapolation error, such as when a large gradient at the surface causes a negative values when only positive values are acceptable. If this happens to the upper and lower bounds that can be placed on the fields to be transferred with the SEND FIELD command.

TRUNCATE - The receiving coordinate is projected back to the surface of the sending mesh to determine a value. This ensures that the receiving value is not outside of the field values in the sending mesh.

PROJECT - This option is similar to TRUNCATE in which the receiving coordinate is projected back to the surface of the sending mesh to determine a value. In this case more effort is made to make sure that the projection is normal to the surface in the sending mesh. Sometimes gives a better result than Truncate but is a little more expensive to compute.

If the PROJECT option is used in transferring of surface values, the sending mesh should envelope the receiving mesh. Failure to satisfy this condition will generally result in failure of the transfer.

ABORT - If any receiving point is outside the sending mesh by more than the geometric tolerance, abort the simulation. Do not attempt to project, extrapolate, or otherwise handle the point.

Parameter	Value	Default
<i>{=}</i>	{= are is }	–
<i>Option</i>	{abort extrapolate ignore project truncate }	–

Parametric Tolerance

Syntax

Parametric Tolerance *{=}* *Parametric_tolerance*

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>Parametric_tolerance</i>	real	–

Patch Recovery Evaluation

Syntax

Patch Recovery Evaluation *{=}* *{linear least squares | linear moving least squares | quadratic least squares | quadratic moving least squares}*

Summary

This line command defines the available choices for the patch recovery and evaluation algorithm when using interpolation for element variables. The default option is Linear Least Squares.

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>Option</i>	{linear least squares linear moving least squares quadratic least squares quadratic moving least squares}	–

Search Coordinate Field

Syntax

Search Coordinate Field *Source_field_name* State *{new | nm1 | nm2 | nm3 | nm4 | none | old}* To *Destination_field_name* State *{new | nm1 | nm2 | nm3 | nm4 | none | old}*

Summary

Normally the interpolation transfers use the default coordinate field to determine geometry information. This line command can be used to specify an alternate field.

Parameter	Value	Default
<i>Source_field_name</i>	string	–
<i>Option1</i>	{ new nm1 nm2 nm3 nm4 none old }	–
<i>Destination_field_name</i>	string	–
<i>Option2</i>	{ new nm1 nm2 nm3 nm4 none old }	–

Search Geometric Tolerance

Syntax

Search Geometric Tolerance *{=}* *Geometric_tolerance*

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>Geometric_tolerance</i>	real	–

Search Surface Gap Tolerance

Syntax

Search Surface Gap Tolerance *{=}* *Surface_gap_tolerance* [Or Less]

Summary

This is the dimensional tolerance applied during the initial (coarse) search. If specified, all the coarse search boxes are padded by this value. The default behavior is for this to be 1e-9 times the problem bounding box size (diagonal) plus a 10 percent relative expansion per element. If a value is specified for this, it is used as a padding in place of the default with no relative box expansion.

This is a tricky parameter best ignored, let it default to something based on the problem size. During the interpolation transfer there is a geometric search based on the coordinates of the send and receive objects. As part of this search, an axis aligned bounding box is contracted for each sending object and SEARCH GAP TOLERANCE is used to make this box bigger than just a tight bounding box. Lists of receiving points are then quickly found within these axis aligned boxes.

If all points in the receiving mesh are within at least one box, no additional searching needs to be done and the search algorithm is fast. If there are still points in the receiving mesh that were outside of EVERY box, then a warning message will be issued about an “expensive search for extrapolation” for these points. This ‘expensive search’ can be very costly if a large number of receiving objects fall into this category and this line command is provided for those special cases.

The OR LESS optional parameter is used when the tolerance must be set to large value for one part of the mesh but much of the mesh needs a much smaller value. In some cases it is necessary for the tolerance to be set to the actual largest surface gap tolerance which may be far too large a gap for the rest of the mesh. Setting OR LESS allows the search tolerance to be reduced in areas of the mesh thus resulting in a faster search.

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>Surface_gap_tolerance</i>	real	–

Search Type

Syntax

Search Type *{=}* [*{detailed | parallel | proximity}* *{detailed | parallel |*

proximity} {detailed | parallel | proximity}]

Parameter	Value	Default
<i>{=}</i>	{= are is}	–

Select One Receiver For Each Send Object

Syntax

Select One Receiver For Each Send Object

Summary

This option will cause each sending object to be used once and only once. This will have the side effect of some receiving objects not getting any value at all. If you use this option, you will also want to set

NODES OUTSIDE REGION IGNORE

The example which necessitated this option was a case in which there was a delta function defined on an element in the sending mesh. It was desirable that the delta functions be summed into the receiving mesh such that the total value of the sending was conserved. It was better to have only a single element on the receiving side have a non-zero value that was the sum of sending values and not worry about how close the receiving element was to the sending element. A check that this option is working is to use Encore to computer the sum of the values of the sending and receiving fields to make sure the total sum is the same.

Select One Unique Receiver For Each Send Object

Syntax

Select One Unique Receiver For Each Send Object

Summary

An unusual flag to get around an odd problem. Normally each receive object transfers from the nearest sending object so it is almost always the case that a send object will be used multiple times to define a receiving value. This option will cause each sending object to be used only once. This will have the side effect of some receiving objects not getting any value at all. If you use this option, you will also want to set

NODES OUTSIDE REGION IGNORE

or else the uniqueness will be lost for nodes outside the sending region. The example which necessitated this option was a case in which there was a delta function defined on an element in the sending mesh. It was desirable that the delta function be defined on the receiving mesh for only a single element in the neighborhood of the sending element. The analysis was more sensitive to the number of delta functions on the receiving side than the location. So it was

better to have only a single element on the receiving side have a non-zero value and not worry about how close the receiving element was to the sending element.

Send

Syntax

Send *{}* Fields

Summary

Use predefined transfer semantics provided by the specified name.

Parameter	Value	Default
<i>Predefined-transfer</i>	{ }	—

Send Block

Syntax

Send Block *From_blocks...* To *To_blocks...*

Summary

Add element blocks to a particular same mesh element copy transfer operator.

The copy transfer can have multiple of these lines to define many blocks, but each line sends a single block to a single block:

```
SEND BLOCK block_1 TO block_1
SEND BLOCK block_101 TO block_101
```

The interpolation transfer can have only a single SEND BLOCK line, but can define many from/to blocks:

```
SEND BLOCK block_3 block_5 block_6 TO block_3 block_5
```

Parameter	Value	Default
<i>From_blocks</i>	string...	—
<i>To_blocks</i>	string...	—

Send Field

Syntax

Send Field *Source_field_name* State *{new | nm1 | nm2 | nm3 | nm4 | none | old}*
 To *Destination_field_name* State *{new | nm1 | nm2 | nm3 | nm4 | none | old}* [
 Lower Bound *Lower_bound* Upper Bound *Upper_bound*]

Summary

Specifies the mapping between source and destination field names. Vector and tensor fields can be subscripted using parenthesis and 1's based or brackets and 0 based. Notes on subscripting:

- Does not work for COPY transfers, only INTERPOLATION type transfers.
- If the field name itself actually contains either parenthesis or brackets then we are in trouble and an error is going to be thrown due to a syntax error in index specification.
- Only a single subscript is allowed so vectors of vectors or higher order tensors can not use double subscripts. But it should be possible to determine the correct offset within the field and pick out the correct value with a little effort.
- Once subscripted, only a single value will be transferred. It is not possible to transfer multiple values starting at a certain index, instead multiple line commands must be used, as shown above.
- The indexes can be 0 based with brackets or 1 based when using parenthesis. Although this could be very confusing if mixed within a single line command.
- Both the from and to fields can be subscripted independently on the same line.

example

```
SEND FIELD velocity TO velocity
SEND FIELD temp      TO temperature lower bound 0
SEND FIELD x          TO y lower bound 10 upper bound 100
SEND FIELD A(2)       TO B(3) lower bound 10 upper bound 100
SEND FIELD A[1]       TO B[2] lower bound 10 upper bound 100
```

Parameter	Value	Default
<i>Source_field_name</i>	string	—
<i>Option1</i>	{ new nm1 nm2 nm3 nm4 none old }	—
<i>Destination_field_name</i>	string	—
<i>Option2</i>	{ new nm1 nm2 nm3 nm4 none old }	—

Toggle Search Warnings

Syntax

Toggle Search Warnings *{=} {false | no | off | on | true | yes}*

Summary

Specify whether warnings about entities outside of the search domain should be printed. The default behavior is to always print these warning messages.

Parameter	Value	Default
<i>{=}</i>	{ = are is }	—
<i>Option</i>	{ false no off on true yes }	—

Use Centroid For Geometric Proximity

Syntax

Use Centroid For Geometric Proximity

Summary

STK Transfer option to trigger the use of centroid based proximity comparison for selecting the best interpolating entity when receive entities lie outside of the domain. Default geometric proximity comparison is based on geometric projection which is more expensive but accurate. The use of this option is a way to reduce computational cost especially for meshes that are fairly regular. However, there is no guarantee of accuracy.

7.5 Time Control

Time Control

Scope

Fuego Procedure

Summary

Defines the time control for the problem in the procedure scope.

Description

This describes the Time Control block which is convenient for single-physics applications. The Solution Control block is described elsewhere, and is appropriate for multiphysics applications.

begin Time Control

Termination Time {=} *Tend*

begin Time Stepping Block *BlockName*
end

end Time Control

Line Commands

Termination Time

Syntax

Termination Time {=} *Tend*

Summary

Specifies the ending time for this solution period.

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>Tend</i>	real	–

Time Stepping Block

Scope

Time Control

Summary

Defines the time and time step parameters for a solution period.

begin Time Stepping Block *BlockName*

Number Of Time Steps *{=}* *Nsteps*

Start Time *{=}* *Tstart*

Time Step *{=}* *Dt0*

begin Parameters For Fuego Region *RegionName*
end

end Time Stepping Block *BlockName*

Line Commands

Number Of Time Steps

Syntax

Number Of Time Steps *{=}* *Nsteps*

Summary

Specifies the number of time steps in this solution period.

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>Nsteps</i>	integer	–

Start Time

Syntax

Start Time *{=}* *Tstart*

Summary

Specifies the starting time for this solution period.

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>Tstart</i>	real	–

Time Step

Syntax

Time Step *{=}* *Dt0*

Summary

Specifies the time step size for this solution period.

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>Dt0</i>	real	–

Parameters For Fuego Region

Scope

Parameters For, Time Stepping Block

Summary

Defines region-specific time stepping data.

begin Parameters For Fuego Region *RegionName*

Cfl Limit *{=}* *Cflmax*

Iteration At Skip Step *{=}* *SkipIter*

Maximum Time Step *{=}* *DtMax*

Minimum Time Step *{=}* *DtMin*

Skip Step *{=}* *SkipStep*

Skip Steps For Pmr *{=}* *PMRSkip*

Time Step Change Factor *{=}* *Change*

Transient Step Type *{=}* {automatic | fixed | invalid time step type}

end Parameters For Fuego Region *RegionName*

Line Commands

Cfl Limit

Syntax

Cfl Limit *{=}* *Cflmax*

Summary

Specifies the maximum value of the CFL number that is used to determine the auto time step.

Description

Need a fuller description here.

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>Cflmax</i>	real	0.5

Iteration At Skip Step

Syntax

Iteration At Skip Step *{=}* *SkipIter*

Summary

Controls the number of iterations in between a skip step.

Description

This is the first implementation of the ability to freeze the Fuego region and allow time advancement without a Fluids/Conduction solve. In the near future, solution control will allow this feature in a more elegant manner.

For now, the iteration count will allow for a given number of iterations when the Afgo regions solve is “active”, that is not during a skip step.

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>SkipIter</i>	integer	–

Maximum Time Step

Syntax

Maximum Time Step *{=}* *DtMax*

Summary

Specifies the maximum allowable time increment for auto timestepping.

Description

Need a fuller description here.

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>DtMax</i>	real	1.0e12

Minimum Time Step

Syntax

Minimum Time Step *{=}* *DtMin*

Summary

Specifies the minimum allowable time increment for auto timestepping.

Description

Need a fuller description here.

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>DtMin</i>	real	0.0

Skip Step

Syntax

Skip Step *{=}* *SkipStep*

Summary

Controls the frequency of Fuego (Afgo) iteration.

Description

This is the first implementation of the ability to freeze the Fuego region and allow time advancement without a Fluids/Conduction solve. In the near future, solution control will allow this feature in a more elegant manner.

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>SkipStep</i>	integer	–

Skip Steps For Pmr

Syntax

Skip Steps For Pmr *{=}* *PMRSkip*

Summary

Controls the frequency of PMR solve.

Description

Set the PMR solve interval. Default is 1 (every step).

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>PMRSkip</i>	integer	1

Time Step Change Factor

Syntax

Time Step Change Factor *{=}* *Change*

Summary

Specify the factor by which the time step is allowed to increase or decrease between time steps.

Description

Need a fuller description here.

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>Change</i>	real	10.0

Transient Step Type

Syntax

Transient Step Type *{=}* *{automatic | fixed | invalid time step type}*

Summary

Specifies whether to use fixed time-stepping or automatic time-stepping.

Description

Need a fuller description here.

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>FuegoStepType</i>	{automatic fixed invalid time step type}	FIXED

7.6 Solution Control

Solution Control Description

Scope

Fuego Procedure

Summary

Contains the commands needed to execute an analysis using the arpeggio procedure that uses Solver Control.

begin Solution Control Description *Name*

Use System *Name*

begin Initialize *Name*
end

begin Parameters For
end

begin System *Name*
end

end Solution Control Description *Name*

Line Commands

Use System

Syntax

Use System *Name*

Summary

This set the name of which system to use.

Parameter	Value	Default
<i>Name</i>	string	—

Initialize

Scope

Solution Control Description

Summary

This block wraps a initializer for a given name. The NAME parameter is the name used to define the initialization block. There can be more than one initialize block in the Solver Control Description block. The “use initialize NAME” line command controls which one is to be used.

begin Initialize *Name*

Advance *Name...* [When *When-expression*]

Event *Name...* [When *When-expression*]

Involve *Name*

Transfer *Name* [When *When-expression*]

end Initialize *Name*

Line Commands

Advance

Syntax

Advance *Name...* [When *When-expression*]

Summary

Used within a Solver Control block to indicate a single step that advances the solution. The name is that matches the physics.

Parameter	Value	Default
<i>Name</i>	string...	—

Event

Syntax

Event *Name...* [When *When-expression*]

Summary

Used within a Solver Control block to indicate a single step that has no time associated with it. It can cause a solution transfer between regions or cause something to print.

Parameter	Value	Default
<i>Name</i>	string. . .	–

Involve

Syntax

Involve *Name*

Summary

Specify a physics participant to a coupled problem solved using matrix-free nonlinear.

Parameter	Value	Default
<i>Name</i>	string	–

Transfer

Syntax

Transfer *Name* [When *When-expression*]

Summary

A Solver Control Transfer line command which executes all transfers defined from the specified region. All transfers with a send region of ‘name’ will be executed.

Parameter	Value	Default
<i>Name</i>	string	–

Parameters For

Scope

Solution Control Description

Summary

A Solver Control PARAMETERS block to set up control data for the SC_type parameter. Inside this block one sets the time step parameters or nonlinear parameters.

begin Parameters For

Converged When *Convergence-expression*

Incremental Number Of Steps {=} *Number*

Initial Deltat {=} *Number*

Number Of Adaptivity Steps `{=}` *Number*

Number Of Steps `{=}` *Number*

Reinitialize Transient

Start Time `{=}` *Number*

Suppress Output From Nonlinear Loop

Termination Time `{=}` *Number*

Time Step Quantum `{=}` *TimeStepQuantum*

Time Step Style *TimeStepStyle...*

Total Change In Time `{=}` *Number*

begin Parameters For Fuego Region *RegionName*
end

end Parameters For

Line Commands

Converged When

Syntax

Converged When *Convergence-expression*

Summary

Set the convergence expression.

Parameter	Value	Default
<i>Convergence-expression</i>	(expression)	–

Incremental Number Of Steps

Syntax

Incremental Number Of Steps `{=}` *Number*

Summary

The incremental number steps to run the time for nonlinear loop. Number of

time steps to run after restarting. NUMBER OF STEPS is total number of steps to run

Parameter	Value	Default
<i>{=}</i>	{= are is }	–
<i>Number</i>	integer	–

Initial Deltat

Syntax

Initial Deltat *{=}* *Number*

Summary

Assign an initial delta T

Parameter	Value	Default
<i>{=}</i>	{= are is }	–
<i>Number</i>	real	–

Number Of Adaptivity Steps

Syntax

Number Of Adaptivity Steps *{=}* *Number*

Summary

The number steps to run the time or nonlinear loop

Parameter	Value	Default
<i>{=}</i>	{= are is }	–
<i>Number</i>	integer	–

Number Of Steps

Syntax

Number Of Steps *{=}* *Number*

Summary

The number steps to run the time for nonlinear loop

Parameter	Value	Default
<i>{=}</i>	{= are is }	–
<i>Number</i>	integer	–

Reinitialize Transient

Syntax

Reinitialize Transient

Summary

Reset time and re-initialize regions each step of the adaptivity loop.

Start Time**Syntax**

Start Time *{=}* *Number*

Summary

Assign a start time.

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>Number</i>	real	–

Suppress Output From Nonlinear Loop**Syntax**

Suppress Output From Nonlinear Loop

Summary

Specify that the nonlinear loop will not output. Output will be handled by calls outside of the nonlinear loop (such as an additional advance region call).

Termination Time**Syntax**

Termination Time *{=}* *Number*

Summary

Assign a final time to stop

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>Number</i>	real	–

Time Step Quantum**Syntax**

Time Step Quantum *{=}* *TimeStepQuantum*

Summary

Set the time stepping quantum time for SNAP style stepping.

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>TimeStepQuantum</i>	real	–

Time Step Style

Syntax

Time Step Style *TimeStepStyle*. . .

Summary

Set the time stepping style.

When CLIP is specified, the time step size will be clipped at the last step of the transient loop so that it ends at the transient loop’s end time. If clip is not specified, the last time is allowed to exceed to the transient loop’s end time and the following transient loop will start at the exceeded end time.

When SNAP is specified, the time step is broken down into “quantum” time units. By default this quantum time is 12 orders of magnitude down from the difference between the start and end time for the transient loop. This value can be overridden using the TIME STEP QUANTUM line command. All time values are “snapped” to multiples of the quantum time by rounding to the nearest quantum multiple.

Parameter	Value	Default
<i>TimeStepStyle</i>	{clip noclip nosnap snap}	CLIP NOSNAP

Total Change In Time

Syntax

Total Change In Time *{=}* *Number*

Summary

Use this number and the initial time to compute termination time.

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>Number</i>	real	–

Parameters For Fuego Region

Scope

Parameters For, Time Stepping Block

Summary

Defines region-specific time stepping data.

begin Parameters For Fuego Region *RegionName*

Cfl Limit *{=}* *Cflmax*

Iteration At Skip Step *{=}* *SkipIter*

Maximum Time Step *{=}* *DtMax*

Minimum Time Step *{=}* *DtMin*

Skip Step *{=}* *SkipStep*

Skip Steps For Pmr *{=}* *PMRSkip*

Time Step Change Factor *{=}* *Change*

Transient Step Type *{=}* *{automatic | fixed | invalid time step type}*

end Parameters For Fuego Region *RegionName*

Line Commands

Cfl Limit

Syntax

Cfl Limit *{=}* *Cflmax*

Summary

Specifies the maximum value of the CFL number that is used to determine the auto time step.

Description

Need a fuller description here.

Parameter	Value	Default
<i>{=}</i>	<i>{= are is}</i>	–
<i>Cflmax</i>	real	0.5

Iteration At Skip Step

Syntax

Iteration At Skip Step *{=}* *SkipIter*

Summary

Controls the number of iterations in between a skip step.

Description

This is the first implementation of the ability to freeze the Fuego region and allow time advancement without a Fluids/Conduction solve. In the near future, solution control will allow this feature in a more elegant manner.

For now, the iteration count will allow for a given number of iterations when the Afgo regions solve is “active”, that is not during a skip step.

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>SkipIter</i>	integer	–

Maximum Time Step

Syntax

Maximum Time Step *{=}* *DtMax*

Summary

Specifies the maximum allowable time increment for auto timestepping.

Description

Need a fuller description here.

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>DtMax</i>	real	1.0e12

Minimum Time Step

Syntax

Minimum Time Step *{=}* *DtMin*

Summary

Specifies the minimum allowable time increment for auto timestepping.

Description

Need a fuller description here.

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>DtMin</i>	real	0.0

Skip Step

Syntax

Skip Step *{=}* *SkipStep*

Summary

Controls the frequency of Fuego (Afgo) iteration.

Description

This is the first implementation of the ability to freeze the Fuego region and allow time advancement without a Fluids/Conduction solve. In the near future, solution control will allow this feature in a more elegant manner.

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>SkipStep</i>	integer	–

Skip Steps For Pmr

Syntax

Skip Steps For Pmr *{=}* *PMRSkip*

Summary

Controls the frequency of PMR solve.

Description

Set the PMR solve interval. Default is 1 (every step).

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>PMRSkip</i>	integer	1

Time Step Change Factor

Syntax

Time Step Change Factor *{=}* *Change*

Summary

Specify the factor by which the time step is allowed to increase or decrease between time steps.

Description

Need a fuller description here.

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>Change</i>	real	10.0

Transient Step Type

Syntax

Transient Step Type *{=}* *{automatic | fixed | invalid time step type}*

Summary

Specifies whether to use fixed time-stepping or automatic time-stepping.

Description

Need a fuller description here.

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>FuegoStepType</i>	{automatic fixed invalid time step type}	FIXED

System

Scope

Solution Control Description

Summary

This block wraps a solver system for a given name. The NAME parameter is the name used to define the system. There can be more than one system block in the Solver Control Description block. The “use system NAME” line command controls which one is to be used.

begin System *Name*

Event *Name...* [When *When-expression*]

Output *Name* [When *When-expression*]

Simulation Max Global Iterations *{=}* *Number*

Simulation Start Time *{=}* *Number*

Simulation Termination Time *{=}* *Number*

Transfer *Name* [When *When-expression*]

Use Initialize *Name*

begin Sequential *Name*
end

begin Transient *Name*

end

end System *Name*

Line Commands

Event

Syntax

Event *Name...* [When *When-expression*]

Summary

Used within a Solver Control block to indicate a single step that has no time associated with it. It can cause a solution transfer between regions or cause something to print.

Parameter	Value	Default
<i>Name</i>	string. . .	–

Output

Syntax

Output *Name* [When *When-expression*]

Summary

A Solver Control Output line command which execute a perform I/O on the region.

Parameter	Value	Default
<i>Name</i>	string	–

Simulation Max Global Iterations

Syntax

Simulation Max Global Iterations {=} *Number*

Summary

The Total number of Solves.

Parameter	Value	Default
{=}	{= are is }	–
<i>Number</i>	integer	–

Simulation Start Time

Syntax

Simulation Start Time {=} *Number*

Summary

Simulation starting time. (by default 0.0)

Parameter	Value	Default
{=}	{= are is}	–
<i>Number</i>	real	–

Simulation Termination Time**Syntax**

Simulation Termination Time {=} *Number*

Summary

The drop dead time.

Parameter	Value	Default
{=}	{= are is}	–
<i>Number</i>	real	–

Transfer**Syntax**

Transfer *Name* [When *When-expression*]

Summary

A Solver Control Transfer line command which executes all transfers defined from the specified region. All transfers with a send region of ‘name’ will be executed.

Parameter	Value	Default
<i>Name</i>	string	–

Use Initialize**Syntax**

Use Initialize *Name*

Summary

This set the name of which initialization to use.

Parameter	Value	Default
<i>Name</i>	string	–

Sequential

Scope

System

Summary

This block is used to wrap a sequential solution. It is used to wrap a sequence of Non-Linear or pseudo time solve step solves.

begin Sequential *Name*

Advance *Name...* [When *When-expression*]

Event *Name...* [When *When-expression*]

Involve *Name*

Output *Name* [When *When-expression*]

Transfer *Name* [When *When-expression*]

begin Nonlinear *Name*
end

end Sequential *Name*

Line Commands

Advance

Syntax

Advance *Name...* [When *When-expression*]

Summary

Used within a Solver Control block to indicate a single step that advances the solution. The name is that matches the physics.

Parameter	Value	Default
<i>Name</i>	string. . .	—

Event

Syntax

Event *Name...* [When *When-expression*]

Summary

Used within a Solver Control block to indicate a single step that has no time associated with it. It can cause a solution transfer between regions or cause something to print.

Parameter	Value	Default
<i>Name</i>	string. . .	–

Involve

Syntax

Involve *Name*

Summary

Specify a physics participant to a coupled problem solved using matrix-free nonlinear.

Parameter	Value	Default
<i>Name</i>	string	–

Output

Syntax

Output *Name* [When *When-expression*]

Summary

A Solver Control Output line command which execute a perform I/O on the region.

Parameter	Value	Default
<i>Name</i>	string	–

Transfer

Syntax

Transfer *Name* [When *When-expression*]

Summary

A Solver Control Transfer line command which executes all transfers defined from the specified region. All transfers with a send region of ‘name’ will be executed.

Parameter	Value	Default
<i>Name</i>	string	–

Transient

Scope

System

Summary

This block is used to wrap a time loop.

begin Transient *Name*

Advance *Name...* [When *When-expression*]

Event *Name...* [When *When-expression*]

Involve *Name*

Output *Name* [When *When-expression*]

Transfer *Name* [When *When-expression*]

begin Nonlinear *Name*
end

begin Subcycle *Name*
end

end Transient *Name*

Line Commands

Advance

Syntax

Advance *Name...* [When *When-expression*]

Summary

Used within a Solver Control block to indicate a single step that advances the solution. The name is that matches the physics.

Parameter	Value	Default
<i>Name</i>	string...	—

Event

Syntax

Event *Name...* [When *When-expression*]

Summary

Used within a Solver Control block to indicate a single step that has no time associated with it. It can cause a solution transfer between regions or cause something to print.

Parameter	Value	Default
<i>Name</i>	string. . .	–

Involve

Syntax

Involve *Name*

Summary

Specify a physics participant to a coupled problem solved using matrix-free nonlinear.

Parameter	Value	Default
<i>Name</i>	string	–

Output

Syntax

Output *Name* [When *When-expression*]

Summary

A Solver Control Output line command which execute a perform I/O on the region.

Parameter	Value	Default
<i>Name</i>	string	–

Transfer

Syntax

Transfer *Name* [When *When-expression*]

Summary

A Solver Control Transfer line command which executes all transfers defined from the specified region. All transfers with a send region of ‘name’ will be executed.

Parameter	Value	Default
<i>Name</i>	string	–

Nonlinear

Scope

Sequential, Transient

Summary

This block is used to wrap a nonlinear solve loop.

begin Nonlinear *Name*

Advance *Name...* [When *When-expression*]

Event *Name...* [When *When-expression*]

Involve *Name*

Output *Name* [When *When-expression*]

Transfer *Name* [When *When-expression*]

begin Subcycle *Name*
end

end Nonlinear *Name*

Line Commands

Advance

Syntax

Advance *Name...* [When *When-expression*]

Summary

Used within a Solver Control block to indicate a single step that advances the solution. The name is that matches the physics.

Parameter	Value	Default
<i>Name</i>	string...	—

Event

Syntax

Event *Name...* [When *When-expression*]

Summary

Used within a Solver Control block to indicate a single step that has no time

associated with it. It can cause a solution transfer between regions or cause something to print.

Parameter	Value	Default
<i>Name</i>	string. . .	–

Involve

Syntax

Involve *Name*

Summary

Specify a physics participant to a coupled problem solved using matrix-free nonlinear.

Parameter	Value	Default
<i>Name</i>	string	–

Output

Syntax

Output *Name* [When *When-expression*]

Summary

A Solver Control Output line command which execute a perform I/O on the region.

Parameter	Value	Default
<i>Name</i>	string	–

Transfer

Syntax

Transfer *Name* [When *When-expression*]

Summary

A Solver Control Transfer line command which executes all transfers defined from the specified region. All transfers with a send region of ‘name’ will be executed.

Parameter	Value	Default
<i>Name</i>	string	–

Subcycle

Scope

Nonlinear, Transient

Summary

This block is used to wrap a subcycle time loop.

begin Subcycle *Name*

Advance *Name...* [When *When-expression*]

Event *Name...* [When *When-expression*]

Involve *Name*

Output *Name* [When *When-expression*]

Transfer *Name* [When *When-expression*]

end Subcycle *Name*

Line Commands

Advance

Syntax

Advance *Name...* [When *When-expression*]

Summary

Used within a Solver Control block to indicate a single step that advances the solution. The name is that matches the physics.

Parameter	Value	Default
<i>Name</i>	string. . .	—

Event

Syntax

Event *Name...* [When *When-expression*]

Summary

Used within a Solver Control block to indicate a single step that has no time associated with it. It can cause a solution transfer between regions or cause something to print.

Parameter	Value	Default
<i>Name</i>	string. . .	–

Involve

Syntax

Involve *Name*

Summary

Specify a physics participant to a coupled problem solved using matrix-free nonlinear.

Parameter	Value	Default
<i>Name</i>	string	–

Output

Syntax

Output *Name* [When *When-expression*]

Summary

A Solver Control Output line command which execute a perform I/O on the region.

Parameter	Value	Default
<i>Name</i>	string	–

Transfer

Syntax

Transfer *Name* [When *When-expression*]

Summary

A Solver Control Transfer line command which executes all transfers defined from the specified region. All transfers with a send region of ‘name’ will be executed.

Parameter	Value	Default
<i>Name</i>	string	–

7.7 Fuego Region

Fuego Region

Scope

Fuego Procedure

Summary

Contains the commands needed to execute an analysis on this region.

begin Fuego Region *Regionname*

Allow Inverted Elements

Calculate Mesh Quality Metrics

Find Maximum Residuals

Initialize With Continuity Solve

Integer Data For Subroutine *SubName* **{=}** *Values...*

Maximum Temperature Allowed From Temperature Extraction **{=}** *Value*

Minimum Temperature Allowed From Temperature Extraction **{=}** *Value*

Nonlinear Residual Plotfile Path **{=}** *Path*

Real Data For Subroutine *SubName* **{=}** *Values...*

Region Participates In Time Step Selection When Inactive **{=}** *{false | ↵
↪true}*

Set Length Unit Conversion Factor **{=}** *Scale*

Set Mass Unit Conversion Factor **{=}** *Scale*

Set Temperature Units **{=}** *{celsius | fahrenheit | kelvin}*

Set Time Unit Conversion Factor **{=}** *Scale*

Set Unit System To *{cgs | mks}*

Setup Warnings Are Errors

```

Use Mpmc Radiation [ With {copy} Transfer ]

Use Reduced Diffusion Stencil

Use Simple Radiation Source With Tref {=} Tref And Emissivity {=}
→Emissivity

Use Solution Steering With Interval {=} Interval

Use Finite Element Model ModelName [ Model Coordinates Are Nodal_
→variable_name ]

Warning Level {=} {minor | moderate | severe}

Initial Uniform Refinement For Num Iterations

begin Averaging OptionsName
end

begin Composite Boundary Condition On Surface Surfacename
end

begin Composite Interface Boundary Condition On Surface Surfacename
end

begin Fixed Boundary Condition On Surface Surfacename
end

begin Heartbeat Label
end

begin Heartbeat Output Label
end

begin Heat Flux Boundary Condition On Surface Surfacename
end

begin History Output Label
end

begin Inflow Boundary Condition On Surface Surfacename
end

begin Initial Condition Block BlockName
end

```

```

begin Mass Flux Boundary Condition On Surface Surfacename
end

begin Nonconformal Boundary Condition On Surface Surfacename
end

begin Open Boundary Condition On Surface Surfacename
end

begin Periodic Boundary Condition On Surface Surfacename
end

begin Post Process On Surface Surfacename
end

begin Postprocess
end

begin Restart Data Label
end

begin Results Output Label
end

begin Solid Object Objectname
end

begin Solution Options OptionsName
end

begin Symmetry Boundary Condition On Surface Surfacename
end

begin Virtual Thermocouple Model On Block BlockName
end

begin Wall Boundary Condition On Surface Surfacename
end

begin Wall Mass Inject Boundary Condition On Surface Surfacename
end

end Fuego Region Regionname

```

Line Commands

Allow Inverted Elements

Syntax

Allow Inverted Elements

Summary

Don't abort on inverted elements

Description

Normal behavior will abort on an inverted element. With this beta option on the element volume will be clipped to epsilon with a warning.

Calculate Mesh Quality Metrics

Syntax

Calculate Mesh Quality Metrics

Summary

Calculates CVFEM mesh quality metrics

Description

Standard finite element quality metrics may not adequately capture mesh quality for CVFEM. Using this will calculate a mesh quality metric that is based on the distance between the dual volume centroid and the node location, normalized to be nominally between 0 and 100. Lower is better here, so a perfect mesh will have a quality metric of 0. Note that boundary elements have a half control volume which makes this calculation less relevant there, so the quality metric is set to 0 on all boundary nodes.

Find Maximum Residuals

Syntax

Find Maximum Residuals

Summary

Locate maximum nonlinear equation residual values on the mesh.

Description

This option will provide the x, y and z location of the maximum nonlinear residual for all equations.

Note that this option may NOT BE USED WITH Prometheus

Initialize With Continuity Solve

Syntax

Initialize With Continuity Solve

Summary

Apply a continuity solve to the initial conditions on the first time step

Description

On the very first time step a continuity solve will be applied to the initial conditions before solving momentum. The solution will proceed in the standard order after that. This option allows initial conditions which may not strictly respect the requirements of incompressibility to be corrected by an initial projection step.

Integer Data For Subroutine

Syntax

Integer Data For Subroutine *SubName* {=} *Values* . .

Summary

List of integer data values to be passed down in to the user subroutine. These values may be changed by the user subroutine.

Parameter	Value	Default
<i>SubName</i>	string	–
{=}	{= are is}	–
<i>Values</i>	integer. . .	–

Maximum Temperature Allowed From Temperature Extraction

Syntax

Maximum Temperature Allowed From Temperature Extraction {=} *Value*

Summary

Enforce a particular maximum temperature that might occur in temperature extraction from enthalpy and composition.

Description

This option specifies the maximum temperature to be allowed to be extracted from enthalpy, given the mixture composition. If a temperature is extracted that is greater than this value, a warning will be printed and the temperature will be reset to the previous value.

Parameter	Value	Default
{=}	{= are is}	–
<i>Value</i>	real	2600.0 K

Minimum Temperature Allowed From Temperature Extraction

Syntax

Minimum Temperature Allowed From Temperature Extraction {=} *Value*

Summary

Enforce a particular minimum temperature that might occur in temperature extraction from enthalpy and composition.

Description

This option specifies the minimum temperature to be allowed to be extracted from enthalpy, given the mixture composition. If a temperature is extracted that is less than this value, a warning will be printed and the temperature will be reset to the previous value.

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>Value</i>	real	250.0 K

Nonlinear Residual Plotfile Path

Syntax

Nonlinear Residual Plotfile Path *{=}* *Path*

Summary

Specify the path to write the nonlinear residual plot files to for this region. Note that all plotfiles will be tagged by the region name as well as the equation name.

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>Path</i>	string	–

Real Data For Subroutine

Syntax

Real Data For Subroutine *SubName* *{=}* *Values* . . .

Summary

List of real data values to be passed down in to the user subroutine. These values may be changed by the user subroutine.

Parameter	Value	Default
<i>SubName</i>	string	–
<i>{=}</i>	{= are is}	–
<i>Values</i>	real. . .	–

Region Participates In Time Step Selection When Inactive

Syntax

Region Participates In Time Step Selection When Inactive *{=}* *{false | true}*

Summary

When region is inactive, it may or may not be able to modify the time step selection

Description

Fuego can be run with inactivating the region solve. However, the question of how to negotiate the time step remains. If this line command is true, then the last Fuego time step will be provided. However, if this line command is false, Fuego will not participate in time step selection criteria.

Parameter	Value	Default
<i>{=}</i>	{= are is }	–
<i>TrueFalse</i>	{false true }	–

Set Length Unit Conversion Factor

Syntax

Set Length Unit Conversion Factor *{=}* *Scale*

Summary

Specify the conversion factor to convert length units to centimeters

Description

Fuego allows arbitrary length units to be specified. The user must provide the conversion factor to convert to centimeters. Users should also ensure complete consistency in their input file and mesh - all units should be in the user-specified system.

Because the default unit system is CGS, use the following convention. If [X] cm = 1 [new length unit], enter “X” for this command. For example, enter 100 if your problem is in meters.

Parameter	Value	Default
<i>{=}</i>	{= are is }	–
<i>Scale</i>	real	–

Set Mass Unit Conversion Factor

Syntax

Set Mass Unit Conversion Factor *{=}* *Scale*

Summary

Specify the conversion factor to convert mass units to grams

Description

Fuego allows arbitrary mass units to be specified. The user must provide the conversion factor to convert to grams. Users should also ensure complete consistency in their input file and mesh - all units should be in the user-specified system.

Because the default unit system is CGS, use the following convention. If [X] g = 1 [new mass unit], enter “X” for this command. For example, enter 1000 if your

problem is in kilograms.

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>Scale</i>	real	–

Set Temperature Units

Syntax

Set Temperature Units *{=}* *{celsius | fahrenheit | kelvin}*

Summary

Specify the scale for temperature units

Description

Fuego allows the user to select from several temperature scales. Users should also ensure complete consistency in their input file and mesh - all units should be in the user-specified system.

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>TempScale</i>	{celsius fahrenheit kelvin}	–

Set Time Unit Conversion Factor

Syntax

Set Time Unit Conversion Factor *{=}* *Scale*

Summary

Specify the conversion factor to convert time units to seconds

Description

Fuego allows arbitrary time units to be specified. The user must provide the conversion factor to convert to seconds. Users should also ensure complete consistency in their input file and mesh - all units should be in the user-specified system.

Because the default unit system is CGS, use the following convention. If [X] s = 1 [new time unit], enter “X” for this command. For example, enter 60 if your problem is in minutes.

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>Scale</i>	real	–

Set Unit System To

Syntax

Set Unit System To *{cgs | mks}*

Summary

Select a common length-mass-time unit system.

Description

Shortcut to select meter-kilogram-second (MKS) or centimeter-gram-second unit (CGS) system. This should correspond to the units your mesh and BCs are in. Using Cantera requires a unit conversion, and CGS units are assumed if you do not specify otherwise.

Explicitly specifying units, either with the “SET UNIT SYSTEM” command or manually with the “SET X UNIT CONVERSION FACTOR” commands also lets Fuego automatically select universal constants like the Stefan-Boltzmann constant, speed of light, and Planck constant (for PMR).

Parameter	Value	Default
<i>UnitSystem</i>	<i>{cgs mks}</i>	–

Setup Warnings Are Errors

Syntax

Setup Warnings Are Errors

Summary

Make problem setup warnings be treated as errors.

Description

By default problem setup warnings are treated as warnings, so things like applying an upwind factor or under-relaxation factor to an equation that isn't active would be a warning.

By adding this option, things like that will be treated as errors instead of warnings.

Use Mpmd Radiation

Syntax

Use Mpmd Radiation [With *{copy}* Transfer]

Summary

Use MPMD radiation coupling - requiring other code to communicate with for PMR through MPI

Description

This solution option indicates that Fuego will be run in MPMD mode, communicating with another executable through MPI for PMR (participating media radiation) coupling. Currently, Nalu is coupled to Fuego for PMR calculations, and several examples exist in the regression test library. Radiation

properties (including boundary radiation properties emissivity and transmissivity) are communicated from Fuego to the PMR code through MPMD. Quadrature rules, radiation mesh, and radiation solver settings are specified in the input deck for that executable. Fuego and the MPMD coupled PMR code are executed simultaneously as:

```
mpirun -n nF fuego -i fuego.i : -n nN nalu -i nalu.i (Nalu)
```

where nF, nN, are the integer numbers of processors to be used for Fuego and Nalu, respectively. fuego.i and nalu.i are the Fuego and Nalu input decks, respectively.

By default it is assumed the Fuego and PMR meshes are different and a general interpolative MPMD transfer will be used. If the meshes are static and identical, a user may instead specify to use a more efficient COPY transfer. COPY transfers are not supported if the fluid side undergoes mesh refinement or there is mesh motion.

Use Reduced Diffusion Stencil

Syntax

Use Reduced Diffusion Stencil

Summary

Use canonical seven point diffusion stencil

Description

The standard diffusion operator supported is represented by the twenty seven node canonical hexahedron stencil. This option allows for all diffusion operators to reduce to the canonical seven point stencil (again hexahedron) by shifting integration point evaluation from the sub surface control volume face centroid to the edge centroid.

This option may be beneficial to use on a highly skewed mesh.

Use Simple Radiation Source With Tref

Syntax

Use Simple Radiation Source With Tref *[=] Tref* And Emissivity *[=] Emissivity*

Summary

Use a difference between the T^4 and T_{ref}^4 to approximate radiation to the far field.

Description

Calculates the scalar_flux term as $4 * \epsilon * \sigma * T_{\text{ref}}^4$ rather than using a PMR solver.

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>Tref</i>	real	–
<i>{=}</i>	{= are is}	–
<i>Emissivity</i>	real	–

Use Solution Steering With Interval

Syntax

Use Solution Steering With Interval *{=}* *Interval*

Summary

Change parameters interactively

Description

Create a solution steering file with parameters that the user can interactively modify during the course of a solution. The file is automatically written during the simulation with the options that are allowed to change. It is read and written again based on the interval set initially in the input file and potentially modified in the steering file. The interval defines how often the file gets read (number of time steps).

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>Interval</i>	integer	1

Use Finite Element Model

Syntax

Use Finite Element Model *ModelName* [Model Coordinates Are *Nodal_variable_name*]

Summary

Associates a predefined finite element model with this region.

Parameter	Value	Default
<i>ModelName</i>	string	–

Warning Level

Syntax

Warning Level *{=}* *{minor | moderate | severe}*

Summary

Set the runtime warning output level.

Description

Control output level of runtime warnings. Choose MINOR (default) to output all warnings, MODERATE to output moderate or severe warnings only, and SEVERE to output only severe errors.

Examples of MINOR errors are mass fractions slightly outside the 0 to 1 range and turbulent production to dissipation ratio clipping.

Examples of MODERATE errors are temperature extraction hitting the clipping bounds or mass fractions farther outside the 0 to 1 bounds (greater than 10 percent out of bounds).

Examples of SEVERE errors are temperature extraction failures or linear solver failures.

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>WarningLevel</i>	{minor moderate severe}	–

Initial Uniform Refinement For

Syntax

Initial Uniform Refinement For *Num* Iterations

Summary

Refine for n number of iterations on initialization

Parameter	Value	Default
<i>Num</i>	integer	–

Averaging

Scope

Fuego Region

Summary

Specify information regarding the Reynolds and Favre averaging. The Reynolds average is the time average of a value:

$$\bar{\phi} = \frac{1}{T} \int \phi(t) dt$$

where $\phi(t)$ is the quantity of interest at time t and the integration is over $(0, T)$, the time interval which is averaged.

The Favre average is the ratio of two Reynolds averages:

$$\tilde{\phi} = \frac{\bar{\rho\phi}}{\bar{\rho}}$$

Where ϕ is the quantity of interest and mass is the mass.

```
begin Averaging OptionsName

  Butterworth A Coefficients {=} Coeffs...

  Butterworth B Coefficients {=} Coeffs...

  Butterworth Filtered Field RegisteredField As AverageField [ On Output_
↳Block BlockName ]

  Butterworth Time Step {=} TimeStep

  Compute {modeled_reynolds_stress | modeled_scalar_flux | resolved_
↳dissipation | resolved_favre_stress | resolved_production | resolved_
↳reynolds_stress | resolved_scalar_flux | resolved_turbulent_kinetic_
↳energy} [ Of FieldName ]

  Favre Average Field RegisteredField As AverageField [ On Output Block_
↳BlockName ]

  Log Output

  Moving Average Filtered Field RegisteredField As AverageField [ On_
↳Output Block BlockName ]

  Moving Average Time {=} emaCharTime

  Reset Filter On Restart

  Reynolds Average Field RegisteredField As AverageField [ On Output_
↳Block BlockName ]

  Starting Time {=} StartingTime

  Time Interval Length {=} IntervallLength

end Averaging OptionsName
```

Line Commands

Butterworth A Coefficients

Syntax

Butterworth A Coefficients *{=}* *Coeffs* . .

Summary

A coefficients for the Butterworth filter

Description

The list of A coefficients for the butterworth filter. See the BUTTERWORTH FILTERED FIELD command documentation for details.

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>Coeffs</i>	real. . .	–

Butterworth B Coefficients

Syntax

Butterworth B Coefficients *{=}* *Coeffs* . .

Summary

B coefficients for the Butterworth filter

Description

The list of B coefficients for the butterworth filter. See the BUTTERWORTH FILTERED FIELD command documentation for details.

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>Coeffs</i>	real. . .	–

Butterworth Filtered Field

Syntax

Butterworth Filtered Field *RegisteredField* As *AverageField* [On Output Block *BlockName*]

Summary

Generates Butterworth filter of field.

Description

The Butterworth filter operations assume that the data is given at a constant time step. The actual time step may vary during the analysis depending on the time stepping mode. The BUTTERWORTH TIME STEP command is used to linearly interpolate the data being produced at a non-constant time step down to some

specified constant time step. The interpolation time step must be larger than zero and ideally should be specified to be smaller than the smallest time step with which the computations will iterate.

One way to obtain the filtering coefficients is with MATLAB. The following is an example of defining a third order Butterworth filter with a pass frequency of 100Hz at data interpolated to a time step of $1.0e - 5$ seconds. The filter is then used to filter acceleration histories of the nodes to 100Hz. The MATLAB code below will give the desired filtering coefficients. The full 16-digit precision of the coefficients returned by MATLAB should be used. If truncated precision numbers are used, the filters can potentially be unstable.

```
clear;
format long e;
passFrequency = 100;
interp_ts = 1.0e-5;
butterCoeff = 2.0*interp_ts*passFrequency;
[bcoeff,acoeff] = butter(3,butterCoeff);
acoeff
bcoeff
```

The computed filtering coefficients can be used in the averaging block as show below. If the analysis time step always remains above $1.0e - 5$ the filter will be valid. If the analysis time step drops below $1.0e - 5$ there could be aliasing issues, and a smaller interpolation time step should be specified.

```
BUTTERWORTH A COEFFICIENTS = 1.0000000000000000e+00 -2.
↪987433650055722e+00 \ $
      2.974946132665442e+00 -9.875122361107358e-01
BUTTERWORTH B COEFFICIENTS = 3.081237301416628e-08 9.
↪243711904249885e-08 \ $
      9.243711904249885e-08 3.081237301416628e-08
BUTTERWORTH TIME STEP = 1e-5
```

Start time and time interval length commands are not used in the Butterworth filter.

Parameter	Value	Default
<i>RegisteredField</i>	string	—
<i>AverageField</i>	string	—

Butterworth Time Step

Syntax

Butterworth Time Step {=} *TimeStep*

Summary

Time step for butterworth filter.

Description

The time step for the butterworth filter. See the BUTTERWORTH FILTERED FIELD command documentation for details.

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>TimeStep</i>	real	0.0

Compute

Syntax

Compute *{modeled_reynolds_stress | modeled_scalar_flux | resolved_dissipation | resolved_favre_stress | resolved_production | resolved_reynolds_stress | resolved_scalar_flux | resolved_turbulent_kinetic_energy}* [Of *FieldName*]

Summary

Compute a modeled or resolved derived quantity.

Description

Select from several pre-defined quantities to compute, described in the list below:

RESOLVED_REYNOLDS_STRESS

$$\overline{S_{ij}} = \overline{\rho u'_i u'_j} = \overline{\rho} (\overline{u_i u_j} - \bar{u}_i \bar{u}_j)$$

RESOLVED_FAVRE_STRESS

$$\widetilde{S_{ij}} = \overline{\rho u''_i u''_j} = \overline{\rho} (\widetilde{u_i u_j} - \widetilde{u}_i \widetilde{u}_j)$$

RESOLVED_TURBULENT_KINETIC_ENERGY

$$\widetilde{k} = \frac{1}{2} \widetilde{u''_k u''_k}$$

RESOLVED_PRODUCTION

$$\widetilde{P_k} = -\widetilde{S_{ij}} \frac{\partial \widetilde{u}_i}{\partial x_j}$$

RESOLVED DISSIPATION

$$\widetilde{\epsilon} = \frac{1}{\rho} \overline{2\mu S''_{ij} \frac{\partial u''_i}{\partial x_j}}$$

$$S''_{ij} = \frac{1}{2} \left(\frac{\partial u''_i}{\partial x_j} + \frac{\partial u''_j}{\partial x_i} \right) - \frac{1}{3} \frac{\partial u''_k}{\partial x_k} \delta_{ij}$$

RESOLVED_SCALAR_FLUX

$$\widetilde{F}_\phi = \bar{\rho} \left(\widetilde{u_i \phi} - \bar{u}_i \widetilde{\phi} \right)$$

MODELED_REYNOLDS_STRESS

$$\widetilde{S}_{ij} = -2\mu^t \widetilde{S}_{ij}^*$$

$$\widetilde{S}_{ij}^* = \frac{1}{2} \left(\frac{\partial \widetilde{u}_i}{\partial x_j} + \frac{\partial \widetilde{u}_j}{\partial x_i} \right)$$

MODELED_SCALAR_FLUX

$$\widetilde{F}_\phi = -\frac{\mu^t}{\text{Pr}^t} \frac{\partial \widetilde{\phi}}{\partial x_i}$$

Parameter	Value	Default
<i>Type</i>	{modeled_reynolds_stress modeled_scalar_flux resolved_dissipation resolved_favre_stress resolved_production resolved_reynolds_stress resolved_scalar_flux resolved_turbulent_kinetic_energy}	–

Favre Average Field

Syntax

Favre Average Field *RegisteredField* As *AverageField* [On Output Block *BlockName*]

Summary

Generates Favre average of field.

Description

The Favre average is the ratio of two Reynolds averages:

$$\tilde{\phi} = \frac{\bar{\rho \phi}}{\bar{\rho}}$$

Where ϕ is the quantity of interest and ρ is the mass.

The field to be averaged must exist in the model being solved. The averaged field will be created and output on the specified block. If the block is not specified, the average will be defined and output on any nodes that the base field exists.

Since the Reynolds average of the mass and the Reynolds average of the field to be Favre averaged must be computed in order to calculate the ratio, these two

extra fields will be created and written to the results file. The Reynolds averaged mass is available as “density_Avg” and the Reynolds average of the mass weighted Favre field, $\bar{\rho}\phi$, will be available as the specified output field name in this line command appended by the string “_Wtd”.

Parameter	Value	Default
<i>RegisteredField</i>	string	–
<i>AverageField</i>	string	–

Log Output

Syntax

Log Output

Summary

Activate verbose logging for filters

Description

When this option is on, additional logging will be output to the log file for each filter, including resets, filter time, and filter active/inactive state.

Moving Average Filtered Field

Syntax

Moving Average Filtered Field *RegisteredField* As *AverageField* [On Output Block *BlockName*]

Summary

Generates an exponential moving average filter.

Description

The moving average filter uses an exponential moving average to update the averaging field (A) using the source field (S) and a blending coefficient (α):

$$\alpha = \min\left(1, \frac{\Delta t}{t_{filt}}\right)$$

$$A = (1 - \alpha)A + \alpha S$$

The filter time (t_{filt}) is defined using the MOVING AVERAGE TIME command.

Parameter	Value	Default
<i>RegisteredField</i>	string	–
<i>AverageField</i>	string	–

Moving Average Time

Syntax

Moving Average Time {=} *emaCharTime*

Summary

The characteristic filter time to use for the moving average filter.

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>emaCharTime</i>	real	0.0

Reset Filter On Restart

Syntax

Reset Filter On Restart

Summary

Reset Favre/Reynolds averages specified in block

Description

This option, on restart, will remove history from averages, restarting the average if the current time > start time of the average. This option is not compatible with Butterworth filtering.

Reynolds Average Field

Syntax

Reynolds Average Field *RegisteredField* As *AverageField* [On Output Block *BlockName*]

Summary

Generates Reynolds average of field.

Description

The Reynolds average is the time average of a value:

$$\bar{\phi} = \frac{1}{T} \int \phi(t) dt$$

where $\phi(t)$ is the quantity of interest at time t and the integral is evaluated over $(0, T)$, the time interval which is averaged.

The field to be averaged must exist in the model being solved. The averaged field will be created and output on the specified block. If the block is not specified, the average will be defined and output on any nodes that the base field exists.

Since the Reynolds average of the mass and the Reynolds average of the field to be Favre averaged must be computed in order to calculate the ratio, these two extra fields will be created and written to the results file. The Reynolds averaged mass is available as “density_Avg” and the Reynolds average of the mass weighted Favre field, $\rho\bar{\phi}$, will be available as the specified output field name in the line command appended by the string “_Wtd”.

Parameter	Value	Default
<i>RegisteredField</i>	string	–
<i>AverageField</i>	string	–

Starting Time

Syntax

Starting Time *{=}* *StartingTime*

Summary

Time for which the averaging starts.

Description

If the starting time is specified then the averaging will not start until the starting time is obtained. All data before the starting time will be ignored and the average will be zero. Once the starting time is reached the averaging will proceed as described under the time interval length parameter with intervals over $(T_0 + nT, T_0(n + 1)T)$ where T_0 is the starting time.

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>StartingTime</i>	real	0.0

Time Interval Length

Syntax

Time Interval Length *{=}* *IntervalLength*

Summary

Time interval length over which average is computed.

Description

If the time interval length is specified as T , The Reynolds or Favre averages specified will be determined over intervals of length T . The intervals will be over $(nT, (n + 1)T)$ for integers n . At the end of one interval, the running average that is being computed will be zeroed out and the averaging starting all over again. This means that soon after an interval change the output field will contain just the average from the beginning of that time interval to the current time. The result is that at every interval boundary there is liable to be a jump or variation in the running average that will be smoothed out over time.

If the starting time parameter is specified then the averaging will not start until the starting time is obtained. All data before the starting time will be ignored and the average will be zero. Once the starting time is reached the averaging will proceed as described with intervals over $(T_0 + nT, T_0(n + 1)T)$ where T_0 is the starting time.

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>IntervalLength</i>	real	REAL_MAX

Heartbeat

Scope

Average Region, Fuego Region, Input_Output Region, Particle Region

Summary

Describes the location and type of the output stream used for outputting the heartbeat information for the enclosing region.

begin Heartbeat *Label*

Additional Steps *{=}* *List_of_steps...*

Additional Times *{=}* *List_of_times...*

Append *{=}* {false | off | on | true}

At Step *n* {increment | interval} *{=}* *m*

At Time *Dt1* {increment | interval} *{=}* *Dt2*

Auto Output {all | element | global | nodal} User Defined Variables [*↪*
In *UserOutputHeartBeatList...*]

Element [*VariableList...*]

Exists *{=}* {abort | append | overwrite}

Face [*VariableList...*]

Format *{=}* {csv | original | spyhis}

Global [*Variables...*]

Labels *{=}* {off | on}

Legend *{=}* {off | on}

Monitor *{=* | the} {history | restart | results}

```

Nodal [ VariableList... ]

Node [ VariableList... ]

Nodeset [ VariableList... ]

Output On Signal {=} {sigabrt | sigalrm | sigfpe | sighup | sigill | ↵
↵sigint | sigkill | sigpipe | sigquit | sigsegv | sigterm | sigusr1 | ↵
↵sigusr2}

Precision {=} Precision

Start Time {=} Start_time

Stream Name {=} OutputFilename

Synchronize Output

Termination Time {=} Final_time

Timestamp Format

Timestep Adjustment Interval {=} Nsteps

Use Output Scheduler Timer_name

Variable {=} {edge | element | face | global | nodal | node} Variable_
↵list...

end Heartbeat Label

```

Line Commands

Additional Steps

Syntax

Additional Steps {=} *List_of_steps...*

Summary

Additional simulation steps when output should occur.

Parameter	Value	Default
{=}	{= are is}	–
<i>List_of_steps</i>	integer...	–

Additional Times

Syntax

Additional Times *{=}* *List_of_times* . .

Summary

Additional simulation times when output should occur.

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>List_of_times</i>	real. . .	–

Append

Syntax

Append *{=}* *{false | off | on | true}*

Summary

Specifies whether the heartbeat file is appended if it exists. By default, the file is appended if restart is requested and not if restart is not requested. This option does not work for automatic restarts because a new heartbeat file is written with each auto restart.

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>Option</i>	{ false off on true }	–

At Step

Syntax

At Step *n* *{increment | interval}* *{=}* *m*

Summary

Specify an output interval in terms of the internal iteration step count. The first step specifies the step count at the beginning of this interval and the second step specifies the output frequency to be used within this interval.

Parameter	Value	Default
<i>n</i>	integer	–
<i>Option</i>	{ increment interval }	–
<i>{=}</i>	{ = are is }	–
<i>m</i>	integer	–

At Time

Syntax

At Time *Dt1* *{increment | interval}* *{=}* *Dt2*

Summary

Specify an output interval in terms of the internal simulation time. The first time specifies the time at the beginning of this time interval and the second time specifies the output frequency to be used within this interval.

Parameter	Value	Default
<i>Dt1</i>	real	—
<i>Option</i>	{increment interval}	—
<i>{=}</i>	{= are is}	—
<i>Dt2</i>	real	—

Auto Output

Syntax

Auto Output *{all | element | global | nodal}* User Defined Variables [In *UserOutputHeartBeatList...*]

Summary

Allows users to automatically output all user output defined variables for the type requested.

Parameter	Value	Default
<i>auto_output_type_4</i>	{all element global nodal}	—

Element

Syntax

Element [*VariableList...*]

Summary

Define the element variables that should be written to the heartbeat database. The syntax is: “element {internal_name} at element {id} as {DBname}” or “element {internal_name} nearest location X, Y, Z as {DBname}”.

Where {internal_name} is the name of the variable in the Sierra application; and {DBname} is the name as it should appear on the heartbeat database.

Exists

Syntax

Exists *{=}* *{abort | append | overwrite}*

Summary

Specify the behavior when creating this database and there is an existing file with the same name. The default behavior is “OVERWRITE” which deletes the existing file and creates a new file of the same name. “APPEND” will (if possible) append the new data to the end of the existing file. “ABORT” will print an error message and end the analysis.

Parameter	Value	Default
<i>{=}</i>	{= is}	–
<i>Option2</i>	{abort append overwrite}	–

Face

Syntax

Face [*VariableList...*]

Summary

Define the face variables that should be written to the heartbeat database. The syntax is: “face {internal_name} at face {id} as {DBname}” or “face {internal_name} nearest location X, Y, Z as {DBname}”.

Where {internal_name} is the name of the variable in the Sierra application; and {DBname} is the name as it should appear on the heartbeat database.

Format

Syntax

Format *{=} {csv | original | spyhis}*

Summary

The stream type/format to be used for the output results. The only three options at this time are ‘Original’ which is the old default Sierra heartbeat format; ‘SpyHis’ which mimics the CTH Spyhis history output format; and ‘CSV’

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>StreamTypes</i>	{csv original spyhis}	–

Global

Syntax

Global [*Variables...*]

Summary

Define the global/reduction variables that should be written to the heartbeat database. The syntax is: “global {internal_name} as {DBname}”.

Where {internal_name} is the name of the variable in the Sierra application; and {DBname} is the name as it should appear on the heartbeat database.

Labels

Syntax

Labels *{=} {off | on}*

Summary

Specifies whether labels will be displayed or just the value of the variable.
Labels will be shown if this line is not present.

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>Option</i>	{ off on }	on

Legend

Syntax

Legend *{=}* *{off | on}*

Summary

Specifies whether a legend will be displayed prior to outputting any variables.
The legend will not be shown unless this line is present. The legend shows the names of the variables that will be written to the heartbeat output stream. If the variable has multiple components, then the component count is shown after the variable e.g., velocity(3).

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>Option</i>	{ off on }	on

Monitor

Syntax

Monitor *{= | the}* *{history | restart | results}*

Summary

Specifies whether a line will be written to the heartbeat stream when either the results, history, and/or restart data are output.

Parameter	Value	Default
<i>Equals</i>	{ = the }	–
<i>Option</i>	{ history restart results }	–

Nodal

Syntax

Nodal [*VariableList...*]

Summary

Define the nodal variables that should be written to the heartbeat database. The syntax is: “nodal {internal_name} at node {id} as {DBname}” or “nodal {internal_name} nearest location X, Y, Z as {DBname}”.

Where {internal_name} is the name of the variable in the Sierra application; and {DBname} is the name as it should appear on the heartbeat database.

Node

Syntax

Node [*VariableList...*]

Summary

Define the nodal variables that should be written to the heartbeat database. The syntax is: “node {internal_name} at node {id} as {DBname}” or “node {internal_name} nearest location X, Y, Z as {DBname}”.

Where {internal_name} is the name of the variable in the Sierra application; and {DBname} is the name as it should appear on the heartbeat database.

Nodeset

Syntax

Nodeset [*VariableList...*]

Summary

Define the nodeset variables that should be written to the heartbeat database. The syntax is: “nodeset {internal_name} at node {id} as {DBname}” or “nodeset {internal_name} nearest location X, Y, Z as {DBname}”.

Where {internal_name} is the name of the variable in the Sierra application; and {DBname} is the name as it should appear on the heartbeat database. This option finds a single value for the {internal_name} specified without having to specify a nodeset id or name.

Output On Signal

Syntax

Output On Signal *{=}* *{sigabrt | sigalrm | sigfpe | sighup | sigill | sigint | sigkill | sigpipe | sigquit | sigsegv | sigterm | sigusr1 | sigusr2}*

Summary

When the specified signal is raised, the output stream associated with this block will be output.

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>Signals</i>	{ sigabrt sigalrm sigfpe sighup sigill sigint sigkill sigpipe sigquit sigsegv sigterm sigusr1 sigusr2 }	–

Precision

Syntax

Precision *{=}* *Precision*

Summary

The precision to be used for the output of real variables (default=5).

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>Precision</i>	integer	5

Start Time

Syntax

Start Time *{=}* *Start_time*

Summary

Specify the time to start outputting results from this output request block. This time overrides all ‘at time’ and ‘at step’ specifications.

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>Start_time</i>	real	–

Stream Name

Syntax

Stream Name *{=}* *OutputFilename*

Summary

The filename of where the heartbeat data should be written. If the filename begins with the ‘/’ character, it is an absolute path; otherwise, the path to the current directory will be prepended to the name. In addition, there are several predefined streams that can be specified. The predefined streams are ‘cout’ or ‘stdout’ specifies standard output; ‘cerr’, ‘stderr’, ‘clog’, or ‘log’ specifies standard error; ‘output’ or ‘outputP0’ specifies Sierra’s standard output which is redirected to the file specified by the ‘-o’ option on the command line. If the file already exists, it is overwritten. If this line is omitted, then a filename will be created from the basename of the input file with a “.hrt” suffix appended.

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>OutputFilename</i>	string	–

Synchronize Output

Syntax

Synchronize Output

Summary

In an analysis with multiple regions, it is sometimes desirable to synchronize the output of results data between the regions. This can be done by adding the SYNCHRONIZE OUTPUT command line to the results output block. If a results block has this set, then it will write output whenever a previous region writes output. The ordering of regions is based on the order in the input file, algorithmic considerations, or by solution control specifications.

Although the USE OUTPUT SCHEDULER command line can also synchronize output between regions, the SYNCHRONIZE OUTPUT command line will synchronize the output with regions where the output frequency is not under the direct control of the Sierra IO system. Examples of this are typically coupled applications where one or more of the codes are not Sierra-based applications such as Alegria and CTH. A results block with SYNCHRONIZE OUTPUT specified will also synchronize its output with the output of the external code.

The SYNCHRONIZE OUTPUT command can be used with other output scheduling commands such as time-based or step-based output specifications.

Termination Time

Syntax

Termination Time *{=}* *Final_time*

Summary

Specify the time to stop outputting results from this output request block.

Parameter	Value	Default
<i>{=}</i>	{= are is }	–
<i>Final_time</i>	real	–

Timestamp Format

Syntax

Timestamp Format

Summary

The format to be used for the timestamp. See ‘man strftime’ for more information.

Timestep Adjustment Interval

Syntax

Timestep Adjustment Interval *{=}* *Nsteps*

Summary

Specify the number of steps to ‘look ahead’ and adjust the timestep to ensure

that the specified output times or simulation end time will be hit ‘exactly’.

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>Nsteps</i>	integer	–

Use Output Scheduler

Syntax

Use Output Scheduler *Timer_name*

Summary

Associates a predefined output scheduler with this output block (results, restart, heartbeat, or history).

Parameter	Value	Default
<i>Timer_name</i>	string	–

Variable

Syntax

Variable *{=}* *{edge | element | face | global | nodal | node}* *Variable_list. . .*

Summary

Define the variables that should be written to the heartbeat output. The user can request that the values of certain variables be output on the heartbeat line. These variables are limited to region and framework control data currently. The syntax is:

```
variable = {entity_type} {internal_name} at
           {entity_type} {entity_id}      as {external_name}
variable = {entity_type} {internal_name} nearest location
           {x,y,z} as {external_name}
```

For global variables, use:

```
variable = global {internal_name} [as {external_name}]
```

Where:

```
entity_type = node, element, face, edge, global
internal_name = Sierra variable name
entity_id = id of the node, element, face, edge that you want
            the specified variable output at.
external_name = name of variable on the database.
```

The names ‘timestep’, and ‘time’ can be specified as variables also. They are the current timestep and simulation time. This line can appear multiple times.

Parameter	Value	Default
<i>{=}</i>	{= are is }	–
<i>Option</i>	{edge element face global nodal node }	–
<i>Variable_list</i>	string...	–

History Output

Scope

Average Region, Fuego Region, Input_Output Region, Particle Region

Summary

Describes the location and type of the output stream used for outputting history for the enclosing region.

begin History Output *Label*

Additional Steps *{=}* *List_of_steps...*

Additional Times *{=}* *List_of_times...*

At Step *n* *{increment | interval}* *{=}* *m*

At Time *Dt1* *{increment | interval}* *{=}* *Dt2*

Auto Output *{all | element | global | nodal}* User Defined Variables [*↪*
*↪*In *UserOutputHistoryList...*]

Database Name *{=}* *StreamName*

Database Type *{=}* *{catalyst | catalyst_exodus | cgns | dof | dof_*
↪exodus | exodus | exodusii | exonull | generated | genesis | null |
↪parallel_exodus | textmesh}

Element [*VariableList...*]

Exists *{=}* *{abort | add_suffix | append | overwrite}*

Face [*VariableList...*]

Flush Interval *{=}* *Option*

```

Global [ Variables... ]

Nodal [ VariableList... ]

Node [ VariableList... ]

Nodeset [ VariableList... ]

Output On Signal {=} {sigabrt | sigalrm | sigfpe | sighup | sigill | ↪
sigint | sigkill | sigpipe | sigquit | sigsegv | sigterm | sigusr1 | ↪
sigusr2}

Overwrite {=} {false | no | off | on | true | yes}

Property PropertyName {=} PropertyValue

Start Time {=} Start_time

Synchronize Output

Termination Time {=} Final_time

Timestep Adjustment Interval {=} Nsteps

Title

Use Dynamic Topology Io

Use Output Scheduler Timer_name

Variable {=} {edge | element | face | global | nodal | node} Variable_
↪list...

end History Output Label

```

Line Commands

Additional Steps

Syntax

Additional Steps {=} *List_of_steps...*

Summary

Additional simulation steps when output should occur.

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>List_of_steps</i>	integer. . .	–

Additional Times

Syntax

Additional Times *{=}* *List_of_times. . .*

Summary

Additional simulation times when output should occur.

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>List_of_times</i>	real. . .	–

At Step

Syntax

At Step *n* *{increment | interval}* *{=}* *m*

Summary

Specify an output interval in terms of the internal iteration step count. The first step specifies the step count at the beginning of this interval and the second step specifies the output frequency to be used within this interval.

Parameter	Value	Default
<i>n</i>	integer	–
<i>Option</i>	{increment interval}	–
<i>{=}</i>	{= are is}	–
<i>m</i>	integer	–

At Time

Syntax

At Time *Dt1* *{increment | interval}* *{=}* *Dt2*

Summary

Specify an output interval in terms of the internal simulation time. The first time specifies the time at the beginning of this time interval and the second time specifies the output frequency to be used within this interval.

Parameter	Value	Default
<i>Dt1</i>	real	–
<i>Option</i>	{increment interval}	–
<i>{=}</i>	{= are is}	–
<i>Dt2</i>	real	–

Auto Output

Syntax

Auto Output *{all | element | global | nodal}* User Defined Variables [In *UserOutputHistoryList. . .*]

Summary

Allows users to automatically output all user output defined variables for the type requested.

Parameter	Value	Default
<i>auto_output_type_2</i>	{all element global nodal}	–

Database Name

Syntax

Database Name *{=} StreamName*

Summary

The base name of the database containing the output history. If the filename begins with the ‘/’ character, it is an absolute path; otherwise, the path to the current directory will be prepended to the name. If this line is omitted, then a filename will be created from the basename of the input file with a “.h” suffix appended.

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>StreamName</i>	string	–

Database Type

Syntax

Database Type *{=} {catalyst | catalyst_exodus | cgns | dof | dof_exodus | exodus | exodusii | exonull | generated | genesis | null | parallel_exodus | textmesh}*

Summary

The database type/format to be used for the output history.

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>Database Types</i>	{catalyst catalyst_exodus cgns dof dof_exodus exodus exodusii exonull generated genesis null parallel_exodus textmesh}	–

Element

Syntax

Element [*VariableList...*]

Summary

Define the element variables that should be written to the history database. The syntax is: “element {internal_name} at element {id} as {DBname}” or “element {internal_name} nearest location X, Y, Z as {DBname}”.

Where {internal_name} is the name of the variable in the Sierra application; and {DBname} is the name as it should appear on the history database.

Exists

Syntax

Exists *{=}* *{abort | add_suffix | append | overwrite}*

Summary

Specify the behavior when creating this database and there is an existing file with the same name. The default behavior is “OVERWRITE” which deletes the existing file and creates a new file of the same name. “APPEND” will (if possible) append the new data to the end of the existing file. “ABORT” will print an error message and end the analysis. “ADD_SUFFIX” will add a -s???? suffix where the ???? is replaced by a sequential number starting at 0002.

Parameter	Value	Default
<i>{=}</i>	{= is}	–
<i>Option2</i>	{abort add_suffix append overwrite}	–

Face

Syntax

Face [*VariableList...*]

Summary

Define the face variables that should be written to the history database. The syntax is: “face {internal_name} at face {id} as {DBname}” or “face {internal_name} nearest location X, Y, Z as {DBname}”.

Where {internal_name} is the name of the variable in the Sierra application; and {DBname} is the name as it should appear on the history database.

Flush Interval

Syntax

Flush Interval *{=}* *Option*

Summary

The minimum time interval (in seconds) at which output will be explicitly flushed to disk. The default is 10 seconds.

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>Option</i>	integer	10

Global

Syntax

Global [*Variables...*]

Summary

Define the global/reduction variables that should be written to the history database. The syntax is: “global {internal_name} as {DBname}”.

Where {internal_name} is the name of the variable in the Sierra application; and {DBname} is the name as it should appear on the history database.

Nodal

Syntax

Nodal [*VariableList...*]

Summary

Define the nodal variables that should be written to the history database. The syntax is: “nodal {internal_name} at node {id} as {DBname}” or “nodal {internal_name} nearest location X, Y, Z as {DBname}”.

Where {internal_name} is the name of the variable in the Sierra application; and {DBname} is the name as it should appear on the history database.

Node

Syntax

Node [*VariableList...*]

Summary

Define the nodal variables that should be written to the history database. The syntax is: “node {internal_name} at node {id} as {DBname}” or “node {internal_name} nearest location X, Y, Z as {DBname}”.

Where {internal_name} is the name of the variable in the Sierra application; and {DBname} is the name as it should appear on the history database.

Nodeset

Syntax

Nodeset [*VariableList*. . .]

Summary

Define the nodeset variables that should be written to the history database. The syntax is: “nodeset {internal_name} at node {id} as {DBname}” or “nodeset {internal_name} nearest location X, Y, Z as {DBname}”.

Where {internal_name} is the name of the variable in the Sierra application; and {DBname} is the name as it should appear on the history database.

Output On Signal

Syntax

Output On Signal *{=}* {*sigabrt* | *sigalrm* | *sigfpe* | *sighup* | *sigill* | *sigint* | *sigkill* | *sigpipe* | *sigquit* | *sigsegv* | *sigterm* | *sigusr1* | *sigusr2*}

Summary

When the specified signal is raised, the output stream associated with this block will be output.

Parameter	Value	Default
<i>{=}</i>	{ = are is }	—
<i>Signals</i>	{ sigabrt sigalrm sigfpe sighup sigill sigint sigkill sigpipe sigquit sigsegv sigterm sigusr1 sigusr2 }	—

Overwrite

Syntax

Overwrite *{=}* {*false* | *no* | *off* | *on* | *true* | *yes*}

Summary

(DEPRECATED, Use EXISTS) Specify whether the database should be overwritten if it exists. The default behavior is to overwrite unless this command is specified in the output block and either off, false, or no is specified.

Parameter	Value	Default
<i>{=}</i>	{ = is }	—
<i>Option2</i>	{ false no off on true yes }	—

Property

Syntax

Property *PropertyName* *{=}* *PropertyValue*

Summary

Define a database property named “PropertyName” with the value “PropertyValue”. If PropertyValue consists of all digits, it will define an integer property. If PropertyValue is “true” or “yes” or “false” or “no”, it will define a logical property; otherwise it will define a string property. Supported properties are typically database dependent; Some history-related properties are:

- VARIABLE_NAME_CASE = upper|lower
- MAX_NAME_LENGTH = value (32)

Parameter	Value	Default
<i>PropertyName</i>	string	–
<i>{=}</i>	{ = are is }	–
<i>PropertyValue</i>	string	–

Start Time

Syntax

Start Time *{=}* *Start_time*

Summary

Specify the time to start outputting results from this output request block. This time overrides all ‘at time’ and ‘at step’ specifications.

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>Start_time</i>	real	–

Synchronize Output

Syntax

Synchronize Output

Summary

In an analysis with multiple regions, it is sometimes desirable to synchronize the output of results data between the regions. This can be done by adding the SYNCHRONIZE OUTPUT command line to the results output block. If a results block has this set, then it will write output whenever a previous region writes output. The ordering of regions is based on the order in the input file, algorithmic considerations, or by solution control specifications.

Although the USE OUTPUT SCHEDULER command line can also synchronize output between regions, the SYNCHRONIZE OUTPUT command line will synchronize the output with regions where the output frequency is not under the direct control of the Sierra IO system. Examples of this are typically coupled applications where one or more of the codes are not Sierra-based applications

such as Alegra and CTH. A results block with SYNCHRONIZE OUTPUT specified will also synchronize its output with the output of the external code.

The SYNCHRONIZE OUTPUT command can be used with other output scheduling commands such as time-based or step-based output specifications.

Termination Time

Syntax

Termination Time *{=}* *Final_time*

Summary

Specify the time to stop outputting results from this output request block.

Parameter	Value	Default
<i>{=}</i>	{= are is }	–
<i>Final_time</i>	real	–

Timestep Adjustment Interval

Syntax

Timestep Adjustment Interval *{=}* *Nsteps*

Summary

Specify the number of steps to ‘look ahead’ and adjust the timestep to ensure that the specified output times or simulation end time will be hit ‘exactly’.

Parameter	Value	Default
<i>{=}</i>	{= are is }	–
<i>Nsteps</i>	integer	–

Title

Syntax

Title

Summary

Specify the title to be used for this specific output block.

Use Dynamic Topology Io

Syntax

Use Dynamic Topology Io

Summary

Specify that the app use IO for dynamic topology modifications where the output files are stored in a single database. Legacy file format for dynamically changing topology results in the creation of multiple files for each output on a mesh modification. This option leverages the ability of netCDF to create mesh

groups within a single database and concatenate all mesh files into one. The names of each mesh group are of the form IOSS_MESH_GROUP-??? where ??? is the 1-based output index 1, 2, . . . , 10, . . . , 100, . . . Please note that netCDF has a current limit of 65,536 groups

Use Output Scheduler

Syntax

Use Output Scheduler *Timer_name*

Summary

Associates a predefined output scheduler with this output block (results, restart, heartbeat, or history).

Parameter	Value	Default
<i>Timer_name</i>	string	–

Variable

Syntax

Variable {=} {edge | element | face | global | nodal | node} *Variable_list* . .

Summary

Define the variables that should be written to the history database. The syntax is: “variable = entity {internal_name} at entity {id} as {DBname}” or “variable = entity {internal_name} nearest location X, Y, Z as {DBname}” or “variable = entity {internal_name} at location X, Y, Z as {DBname}”.

Where {entity} is ‘node’, ‘element’, ‘face’, or ‘edge’; {internal_name} is the name of the variable in the Sierra application; and {DBname} is the name as it should appear on the history database.

Parameter	Value	Default
{=}	{= are is}	–
<i>Option</i>	{edge element face global nodal node}	–
<i>Variable_list</i>	string. . .	–

Restart Data

Scope

Average Region, Fuego Region, Input_Output Region, Particle Region

Summary

Describes the data required to output and input restart data for the enclosing region.

begin Restart Data *Label*

Additional Steps {=} *List_of_steps...*

Additional Times {=} *List_of_times...*

At Step *n* {increment | interval} {=} *m*

At Time *Dt1* {increment | interval} {=} *Dt2*

At Wall Time *Dt1* {increment | interval} {=} *Dt2*

Component Separator Character {=} *Separator*

Cycle Count {=} *Count*

Database Name {=} *StreamName*

Database Type {=} {catalyst | catalyst_exodus | cgns | dof | dof_
↳exodus | exodus | exodusii | exonull | generated | genesis | null |
↳parallel_exodus | textmesh}

Debug Dump

Decomposition Method {=} {block | cyclic | external | geom_kway | hsfc_
↳| kway | kway_geom | linear | map | metis_sfc | random | rcb | rib |
↳variable}

Exists {=} {abort | add_suffix | append | overwrite}

File Cycle Count {=} *Count*

Input Database Name {=} *StreamName*

Optional

Output Database Name {=} *StreamName*

Output On Signal {=} {sigabrt | sigalrm | sigfpe | sighup | sigill |
↳sigint | sigkill | sigpipe | sigquit | sigsegv | sigterm | sigusr1 |
↳sigusr2}

Output Restart State {=} {off | on}

Overlay Count {=} *Count*

Overwrite `{=}` `{false | no | off | on | true | yes}`

Property *PropertyName* `{=}` *PropertyValue*

Restart `{=}` `{auto}`

Restart Time `{=}` *Time*

Start Time `{=}` *Start_time*

Synchronize Output

Shift To Start Time

Termination Time `{=}` *Final_time*

Timestep Adjustment Interval `{=}` *Nsteps*

Use Dynamic Topology Io

Use Output Scheduler *Timer_name*

end Restart Data *Label*

Line Commands

Additional Steps

Syntax

Additional Steps `{=}` *List_of_steps...*

Summary

Additional simulation steps when output should occur.

Parameter	Value	Default
<code>{=}</code>	{= are is}	–
<i>List_of_steps</i>	integer...	–

Additional Times

Syntax

Additional Times `{=}` *List_of_times...*

Summary

Additional simulation times when output should occur.

Parameter	Value	Default
<i>{=}</i>	{= are is}	—
<i>List_of_times</i>	real. . .	—

At Step

Syntax

At Step *n* *{increment | interval}* *{=}* *m*

Summary

Specify an output interval in terms of the internal iteration step count. The first step specifies the step count at the beginning of this interval and the second step specifies the output frequency to be used within this interval.

Parameter	Value	Default
<i>n</i>	integer	—
<i>Option</i>	{increment interval}	—
<i>{=}</i>	{= are is}	—
<i>m</i>	integer	—

At Time

Syntax

At Time *Dt1* *{increment | interval}* *{=}* *Dt2*

Summary

Specify an output interval in terms of the internal simulation time. The first time specifies the time at the beginning of this time interval and the second time specifies the output frequency to be used within this interval.

Parameter	Value	Default
<i>Dt1</i>	real	—
<i>Option</i>	{increment interval}	—
<i>{=}</i>	{= are is}	—
<i>Dt2</i>	real	—

At Wall Time

Syntax

At Wall Time *Dt1* *{increment | interval}* *{=}* *Dt2*

Summary

Write a restart file at a specific wall time since the start of the run. Time string

format allows s, m, h, d for seconds, minutes, hours, days

Parameter	Value	Default
<i>Dt1</i>	string	—
<i>Option</i>	{increment interval}	—
<i>{=}</i>	{= are is}	—
<i>Dt2</i>	string	—

Component Separator Character

Syntax

Component Separator Character *{=} Separator*

Summary

The separator is the single character used to separate the output variable basename (e.g. “stress”) from the suffices (e.g. “xx”, “yy”) when displaying the names of the individual variable components. For example, the default separator is “_”, which results in names similar to “stress_xx”, “stress_yy”, . . . “stress_zx”. To eliminate the separator, specify an empty string (“”) or NONE.

Parameter	Value	Default
<i>{=}</i>	{= is}	—
<i>Separator</i>	string	—

Cycle Count

Syntax

Cycle Count *{=} Count*

Summary

Specify the number of restart steps which will be written to the restart database before previously written steps are overwritten. For example, if the cycle count is 5 and restart is written every 0.1 seconds, the restart system will write 0.1, 0.2, 0.3, 0.4, 0.5 to the database. It will then overwrite the first step with data from time 0.6, the second with time 0.7. At time 0.8, the database would contain data at times 0.6, 0.7, 0.8, 0.4, 0.5. Note that time will not necessarily be monotonically increasing on a database that specifies the cycle count.

Parameter	Value	Default
<i>{=}</i>	{= are is}	—
<i>Count</i>	integer	—

Database Name

Syntax

Database Name *{=} StreamName*

Summary

The database containing the input and/or output restart data. If this analysis is being restarted, restart data will be read from this file. If the analysis is writing restart data, the data will be written to this file. It will be overwritten if it exists (after being read if applicable). If the filename begins with the ‘/’ character, it is an absolute path; otherwise, the path to the current directory will be prepended to the name. See also the ‘Input Database’ and ‘Output Database’ commands.

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>StreamName</i>	string	–

Database Type

Syntax

Database Type *{=}* *{catalyst | catalyst_exodus | cgns | dof | dof_exodus | exodus | exodusii | exonull | generated | genesis | null | parallel_exodus | textmesh}*

Summary

The database type/format used for the restart file.

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>Database-Types</i>	{catalyst catalyst_exodus cgns dof dof_exodus exodus exodusii exonull generated genesis null parallel_exodus textmesh}	–

Debug Dump

Syntax

Debug Dump

Summary

Specify whether the restart system will write the restart data immediately after reading the restart data if the run is restarting. The output data can be compared with the restart input data to determine whether they match.

Decomposition Method

Syntax

Decomposition Method *{=}* *{block | cyclic | external | geom_kway | hsf | kway | kway_geom | linear | map | metis_sfc | random | rcb | rib | variable}*

Summary

The decomposition algorithm to be used to partition elements to each processor in a parallel run.

Pa- rame- ter	Value	De- fault
<i>{=}</i>	{ = are is }	–
<i>Method</i>	{ block cyclic external geom_kway hsfk kway kway_geom linear map metis_sfc random rcb rib variable }	–

Exists

Syntax

Exists *{=}* *{abort | add_suffix | append | overwrite}*

Summary

Specify the behavior when creating this database and there is an existing file with the same name. The default behavior is “OVERWRITE” which deletes the existing file and creates a new file of the same name. “APPEND” will (if possible) append the new data to the end of the existing file. “ABORT” will print an error message and end the analysis. “ADD_SUFFIX” will add a suffix to the file name and output to that file.

Parameter	Value	Default
<i>{=}</i>	{ = is }	–
<i>Option2</i>	{ abort add_suffix append overwrite }	–

File Cycle Count

Syntax

File Cycle Count *{=}* *Count*

Summary

Each restart dump will be written to a separate file suffixed with A,B, . . . The count specifies how many separate files are used before the cycle repeats. For example, if “FILE CYCLE COUNT = 3” is specified, the restart dumps would be written to file-A.rs, file-B.rs, file-C.rs, file-A.rs, . . . The maximum value for the cycle count is 26.

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>Count</i>	integer	–

Input Database Name

Syntax

Input Database Name *{=}* *StreamName*

Summary

The database containing the input restart data. If this analysis is being restarted, restart data will be read from this file. See also the ‘Database’ and ‘Output Database’ commands.

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>StreamName</i>	string	–

Optional

Syntax

Optional

Summary

The database will be read if it exists, but it is not an error if there is no restart database to read for this region during a restarted analysis.

Output Database Name

Syntax

Output Database Name *{=}* *StreamName*

Summary

The database containing the output restart data. If the analysis is writing restart data, the data will be written to this file. It will be overwritten if it exists. See also the ‘Database’ and ‘Input Database’ commands.

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>StreamName</i>	string	–

Output On Signal

Syntax

Output On Signal *{=}* *{sigabrt | sigalrm | sigfpe | sighup | sigill | sigint | sigkill | sigpipe | sigquit | sigsegv | sigterm | sigusr1 | sigusr2}*

Summary

When the specified signal is raised, the output stream associated with this block will be output.

Pa- rame- ter	Value	De- fault
<i>{=}</i>	{= are is}	–
<i>Signals</i>	{sigabrt sigalrm sigfpe sighup sigill sigint sigkill sigpipe sigquit sigsegv sigterm sigusr1 sigusr2}	–

Output Restart State

Syntax

Output Restart State *{=}* *{off | on}*

Summary

Outputs the restarted state to the new restarted results file

Description

NOTE: This command must be placed at the Sierra scope of the input file.
Allows the analyst to visualize the restarted state for debugging

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>Option</i>	{off on}	–

Overlay Count

Syntax

Overlay Count *{=}* *Count*

Summary

Specify the number of restart outputs which will be overlaid on top of the last written step. For example, if restarts are being output every 0.1 seconds and the overlay count is specified as 2, then restart will write times 0.1 to step 1 of the database. It will then write 0.2 and 0.3 also to step 1. It will then increment the database step and write 0.4 to step 2; overlay 0.5 and 0.6 on step 2. . . At the end of the analysis, assuming it runs to completion, the database would have times 0.3, 0.6, 0.9, . . . However, if there were a problem during the analysis, the last step on the database would contain an intermediate step.

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>Count</i>	integer	–

Overwrite

Syntax

Overwrite *{=}* *{false | no | off | on | true | yes}*

Summary

(DEPRECATED, Use EXISTS) Specify whether the restart database should be overwritten if it exists. The default behavior is to overwrite unless this command is specified in the restart block and either off, false, or no is specified.

Parameter	Value	Default
<i>{=}</i>	{ = is }	–
<i>Option2</i>	{ false no off on true yes }	–

Property

Syntax

Property *PropertyName* *{=}* *PropertyValue*

Summary

Define a database property named “PropertyName” with the value “PropertyValue”. If PropertyValue consists of all digits, it will define an integer property. If PropertyValue is “true” or “yes” or “false” or “no”, it will define a logical property; otherwise it will define a string property. If PropertyName consists of multiple strings, they will be concatenated together with “_” separating the individual words. Supported properties are typically database dependent; Current properties are:

- COMPRESSION_LEVEL = [0..9]
- COMPRESSION_SHUFFLE = true|false|on|off
- FILE_TYPE = netcdf4 (forces use of netcdf-4 hdf5-based file)
- INTEGER_SIZE_DB = 4|8
- INTEGER_SIZE_API = 4|8
- LOGGING = true|false|on|off
- MAX_NAME_LENGTH = value

Parameter	Value	Default
<i>PropertyName</i>	string	–
<i>{=}</i>	{ = are is }	–
<i>PropertyValue</i>	string	–

Restart

Syntax

Restart *{=}* *{auto}*

Summary

Specify automatic restart file read.

Description

NOTE: This command must be placed at the Sierra scope of the input file.

Specify that the analysis should be restarted from the last common time on all restart databases for each Region in the analysis. In addition to this line command, each Region in the analysis (strictly, only the region(s) that will be restarted) must have a restart block specifying the database to read the restart state data.

By default, use of this command will not cause output files (e.g., results, history, heartbeat, restart) to be overwritten. Instead output files will be written with the same basename and the suffix `-s0000*`. Common visualization packages are written to handle this file organization gracefully in order for the user to view all results seamlessly.

Parameter	Value	Default
<code>{=}</code>	{ = are is }	–
<code>{auto}</code>	{ auto automatic }	–

Restart Time

Syntax

Restart Time `{=}` *Time*

Summary

Specify restart file read at a specified time.

Description

NOTE: This command must be placed at the Sierra scope of the input file.

Specify the time that the analysis will be restarted. In addition to this line command, each Region in the analysis (strictly, only the region(s) that will be restarted) must have a restart block specifying the database to read the restart state data. The restart ‘time’ must be greater than zero and less than or equal to the termination time.

By default, use of this command will cause previous output files (e.g., results, history, heartbeat, restart) to be overwritten. If this command is chosen, the onus is placed on the user to ensure that previous output files are not overwritten.

Parameter	Value	Default
<code>{=}</code>	{ = are is }	–
<i>Time</i>	real	–

Start Time

Syntax

Start Time `{=}` *Start_time*

Summary

Specify the time to start outputting results from this output request block. This time overrides all 'at time' and 'at step' specifications.

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>Start_time</i>	real	–

Synchronize Output

Syntax

Synchronize Output

Summary

In an analysis with multiple regions, it is sometimes desirable to synchronize the output of results data between the regions. This can be done by adding the SYNCHRONIZE OUTPUT command line to the results output block. If a results block has this set, then it will write output whenever a previous region writes output. The ordering of regions is based on the order in the input file, algorithmic considerations, or by solution control specifications.

Although the USE OUTPUT SCHEDULER command line can also synchronize output between regions, the SYNCHRONIZE OUTPUT command line will synchronize the output with regions where the output frequency is not under the direct control of the Sierra IO system. Examples of this are typically coupled applications where one or more of the codes are not Sierra-based applications such as Alegria and CTH. A results block with SYNCHRONIZE OUTPUT specified will also synchronize its output with the output of the external code.

The SYNCHRONIZE OUTPUT command can be used with other output scheduling commands such as time-based or step-based output specifications.

Shift To Start Time

Syntax

Shift To Start Time

Summary

The shift to start time option allows a user to shift the restart time to the start time of the current region. An example use case would be if a restart time of 0.5 is specified, but the user would like to start the simulation at time 1.0.

Termination Time

Syntax

Termination Time *{=}* *Final_time*

Summary

Specify the time to stop outputting results from this output request block.

Parameter	Value	Default
<i>{=}</i>	{= are is }	–
<i>Final_time</i>	real	–

Timestep Adjustment Interval

Syntax

Timestep Adjustment Interval *{=}* *Nsteps*

Summary

Specify the number of steps to ‘look ahead’ and adjust the timestep to ensure that the specified output times or simulation end time will be hit ‘exactly’.

Parameter	Value	Default
<i>{=}</i>	{= are is }	–
<i>Nsteps</i>	integer	–

Use Dynamic Topology Io

Syntax

Use Dynamic Topology Io

Summary

Specify that the app use IO for dynamic topology modifications where the output files are stored in a single database. Legacy file format for dynamically changing topology results in the creation of multiple files for each output on a mesh modification. This option leverages the ability of netCDF to create mesh groups within a single database and concatenate all mesh files into one. The names of each mesh group are of the form IOSS_MESH_GROUP-??? where ??? is the 1-based output index 1, 2, ..., 10, ..., 100, ... Please note that netCDF has a current limit of 65,536 groups

Use Output Scheduler

Syntax

Use Output Scheduler *Timer_name*

Summary

Associates a predefined output scheduler with this output block (results, restart, heartbeat, or history).

Parameter	Value	Default
<i>Timer_name</i>	string	–

Results Output

Scope

Average Region, Fuego Region, Input_Output Region, Particle Region

Summary

Describes the location and type of the output stream used for outputting results for the enclosing region.

begin Results Output *Label*

Additional Steps *{=} List_of_steps...*

Additional Times *{=} List_of_times...*

At Step *n {increment | interval} {=} m*

At Time *Dt1 {increment | interval} {=} Dt2*

Auto Output *{all | element | global | nodal}* User Defined Variables [
↳In *UserOutputResultsList...*]

Auto Output *{all | element | global | nodal}* Variables

Component Separator Character *{=} Separator*

Database Name *{=} StreamName*

Database Type *{=} {catalyst | catalyst_exodus | cgns | dof | dof_
↳exodus | exodus | exodusii | exonull | generated | genesis | null |
↳parallel_exodus | textmesh}*

Edge *VariableList...*

Element *VariableList...*

Enable Large Ids

Exclude *{=} [ElementBlockList...]*

Exists *{=} {abort | add_suffix | append | overwrite}*

Face *VariableList...*

Flush Interval *{=} Option*

```

Global VariableList...

Include {=} [ ElementBlockList... ]

Nodal VariableList...

Node VariableList...

Nodeset VariableList...

Output Mesh {=} {exposed surface | refined | unrefined}

Output On Signal {=} {sigabrt | sigalrm | sigfpe | sighup | sigill | ↵
↵sigint | sigkill | sigpipe | sigquit | sigsegv | sigterm | sigusr1 | ↵
↵sigusr2}

Overwrite {=} {false | no | off | on | true | yes}

Property PropertyName {=} PropertyValue

Sideset VariableList...

Start Time {=} Start_time

Surface VariableList...

Synchronize Output

Termination Time {=} Final_time

Timeseries Name {=} filename

Timestep Adjustment Interval {=} Nsteps

Title

Use Dynamic Topology Io

Use Output Scheduler Timer_name

begin Catalyst Label
end

end Results Output Label

```

Line Commands

Additional Steps

Syntax

Additional Steps *{=}* *List_of_steps* . .

Summary

Additional simulation steps when output should occur.

Parameter	Value	Default
<i>{=}</i>	{= are is }	—
<i>List_of_steps</i>	integer. . .	—

Additional Times

Syntax

Additional Times *{=}* *List_of_times* . .

Summary

Additional simulation times when output should occur.

Parameter	Value	Default
<i>{=}</i>	{= are is }	—
<i>List_of_times</i>	real. . .	—

At Step

Syntax

At Step *n* *{increment | interval}* *{=}* *m*

Summary

Specify an output interval in terms of the internal iteration step count. The first step specifies the step count at the beginning of this interval and the second step specifies the output frequency to be used within this interval.

Parameter	Value	Default
<i>n</i>	integer	—
<i>Option</i>	{increment interval}	—
<i>{=}</i>	{= are is }	—
<i>m</i>	integer	—

At Time

Syntax

At Time *Dt1* *{increment | interval}* *{=}* *Dt2*

Summary

Specify an output interval in terms of the internal simulation time. The first time specifies the time at the beginning of this time interval and the second time specifies the output frequency to be used within this interval.

Parameter	Value	Default
<i>Dt1</i>	real	–
<i>Option</i>	{increment interval}	–
<i>{=}</i>	{= are is}	–
<i>Dt2</i>	real	–

Auto Output

Syntax

Auto Output *{all | element | global | nodal}* User Defined Variables [In *UserOutputResultsList. . .*]

Summary

Allows users to automatically output all user output defined variables for the type requested.

Parameter	Value	Default
<i>auto_output_type_3</i>	{all element global nodal}	–

Auto Output

Syntax

Auto Output *{all | element | global | nodal}* Variables

Summary

Allows users to automatically output all user output defined variables for the type requested.

Parameter	Value	Default
<i>auto_output_type_3</i>	{all element global nodal}	–

Component Separator Character

Syntax

Component Separator Character *{=}* *Separator*

Summary

The separator is the single character used to separate the output variable basename (e.g. “stress”) from the suffices (e.g. “xx”, “yy”) when displaying the names of the individual variable components. For example, the default separator

is “_”, which results in names similar to “stress_xx”, “stress_yy”, . . . “stress_zx”. To eliminate the separator, specify an empty string (“”) or NONE.

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>Separator</i>	string	–

Database Name

Syntax

Database Name *{=}* *StreamName*

Summary

The base name of the database containing the output results. If the filename begins with the ‘/’ character, it is an absolute path; otherwise, the path to the current directory will be prepended to the name. If this line is omitted, then a filename will be created from the basename of the input file with a “.e” suffix appended.

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>StreamName</i>	string	–

Database Type

Syntax

Database Type *{=}* *{catalyst | catalyst_exodus | cgns | dof | dof_exodus | exodus | exodusii | exonull | generated | genesis | null | parallel_exodus | textmesh}*

Summary

The database type/format to be used for the output results.

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>Database Type</i>	{ catalyst catalyst_exodus cgns dof dof_exodus exodus exodusii exonull generated genesis null parallel_exodus textmesh }	–

Edge

Syntax

Edge *VariableList. . .*

Summary

Define the variables that should be written to the results database. If “variable” is entered, then its name will be used on the output database. If “variable as

db_name” is entered, then “db_name” will be the name used on the database for the internal variable “variable”. Multiple “variable” or “variable as db_name” entries are allowed on the same line. The entities that this variable are written to can also be limited or specified with “exclude list_of_entities” or “include list_of_entities”. Edge variables are not supported for all database types.

Parameter	Value	Default
<i>VariableList</i>	string. . .	–

Element

Syntax

Element *VariableList. . .*

Summary

Define the variables that should be written to the results database. If “variable” is entered, then its name will be used on the output database. If “variable as db_name” is entered, then “db_name” will be the name used on the database for the internal variable “variable”. Multiple “variable” or “variable as db_name” entries are allowed on the same line. The entities that this variable are written to can also be limited or specified with “exclude list_of_entities” or “include list_of_entities”

Parameter	Value	Default
<i>VariableList</i>	string. . .	–

Enable Large Ids

Syntax

Enable Large Ids

Summary

Enable 64 bit entity IDs for output

Exclude

Syntax

Exclude {=} [*ElementBlockList. . .*]

Summary

Specify that the results file will only contain a subset of the element blocks in the analysis model. The element_block_list lists only the blocks which will not be output to the results database.

Parameter	Value	Default
{=}	{= are is }	–

Exists

Syntax

Exists *{=}* *{abort | add_suffix | append | overwrite}*

Summary

Specify the behavior when creating this database and there is an existing file with the same name. The default behavior is “OVERWRITE” which deletes the existing file and creates a new file of the same name. “APPEND” will (if possible) append the new data to the end of the existing file. “ABORT” will print an error message and end the analysis. “ADD_SUFFIX” will add a -s??? suffix where the ??? is replaced by a sequential number starting at 0002.

Parameter	Value	Default
<i>{=}</i>	{ = is }	–
<i>Option2</i>	{ abort add_suffix append overwrite }	–

Face

Syntax

Face *VariableList. . .*

Summary

Define the variables that should be written to the results database. If “variable” is entered, then its name will be used on the output database. If “variable as db_name” is entered, then “db_name” will be the name used on the database for the internal variable “variable”. Multiple “variable” or “variable as db_name” entries are allowed on the same line. The entities that this variable are written to can also be limited or specified with “exclude list_of_entities” or “include list_of_entities”. Face variables are not supported for all database types.

Parameter	Value	Default
<i>VariableList</i>	string. . .	–

Flush Interval

Syntax

Flush Interval *{=}* *Option*

Summary

The minimum time interval (in seconds) at which output will be explicitly flushed to disk. The default is 10 seconds.

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>Option</i>	integer	10

Global

Syntax

Global *VariableList...*

Summary

Define the global variables that should be written to the results database. If “variable” is entered, then its name will be used on the output database. If “variable as db_name” is entered, then “db_name” will be the name used on the database for the internal variable “variable”. Multiple “variable” or “variable as db_name” entries are allowed on the same line.

Parameter	Value	Default
<i>VariableList</i>	string...	–

Include

Syntax

Include *{=}* [*ElementBlockList...*]

Summary

Specify that the results file will only contain a subset of the element blocks in the analysis model. The element_block_list lists only the blocks which will be output to the results database.

Parameter	Value	Default
<i>{=}</i>	{= are is}	–

Nodal

Syntax

Nodal *VariableList...*

Summary

Define the nodal variables that should be written to the results database. If “variable” is entered, then its name will be used on the output database. If “variable as db_name” is entered, then “db_name” will be the name used on the database for the internal variable “variable”. Multiple “variable” or “variable as db_name” entries are allowed on the same line.

Parameter	Value	Default
<i>VariableList</i>	string...	–

Node

Syntax

Node *VariableList...*

Summary

Define the nodal variables that should be written to the results database. If “variable” is entered, then its name will be used on the output database. If “variable as db_name” is entered, then “db_name” will be the name used on the database for the internal variable “variable”. Multiple “variable” or “variable as db_name” entries are allowed on the same line.

Parameter	Value	Default
<i>VariableList</i>	string. . .	–

Nodeset

Syntax

Nodeset *VariableList. . .*

Summary

Define the variables that should be written to the results database. If “variable” is entered, then its name will be used on the output database. If “variable as db_name” is entered, then “db_name” will be the name used on the database for the internal variable “variable”. Multiple “variable” or “variable as db_name” entries are allowed on the same line. The entities that this variable are written to can also be limited or specified with “exclude list_of_entities” or “include list_of_entities”. Nodeset variables are not supported for all database types.

Parameter	Value	Default
<i>VariableList</i>	string. . .	–

Output Mesh

Syntax

Output Mesh {=} {*exposed surface* | *refined* | *unrefined*}

Summary

Use this command to turn on “unrefined” as the output mesh. The default behavior is “refined”, in which field variables are output on the current mesh, which may have been refined (either uniformly or adaptively) or had its topology altered in some way (e.g., dynamic load balancing) with respect to the original mesh read from the input file. By specifying “Output Mesh = unrefined”, all output variables are output only on the original mesh objects read from the input file.

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>OutputMesh</i>	{ <i>exposed surface</i> <i>refined</i> <i>unrefined</i> }	–

Output On Signal

Syntax

Output On Signal *{=}* *{sigabrt | sigalrm | sigfpe | sighup | sigill | sigint | sigkill | sigpipe | sigquit | sigsegv | sigterm | sigusr1 | sigusr2}*

Summary

When the specified signal is raised, the output stream associated with this block will be output.

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>Signals</i>	{ sigabrt sigalrm sigfpe sighup sigill sigint sigkill sigpipe sigquit sigsegv sigterm sigusr1 sigusr2 }	–

Overwrite

Syntax

Overwrite *{=}* *{false | no | off | on | true | yes}*

Summary

(DEPRECATED, Use EXISTS) Specify whether the database should be overwritten if it exists. The default behavior is to overwrite unless this command is specified in the output block and either off, false, or no is specified.

Parameter	Value	Default
<i>{=}</i>	{ = is }	–
<i>Option2</i>	{ false no off on true yes }	–

Property

Syntax

Property *PropertyName* *{=}* *PropertyValue*

Summary

Define a database property named “PropertyName” with the value “PropertyValue”. If PropertyValue consists of all digits, it will define an integer property. If PropertyValue is “true” or “yes” or “false” or “no”, it will define a logical property; otherwise it will define a string property. Supported properties are typically database dependent; Current properties are:

- COMPRESSION_LEVEL = [0..9] (off)
- COMPRESSION_SHUFFLE = true|false|on|off (off)
- FILE_TYPE = netcdf4 (forces use of netcdf-4 hdf5-based file) (netcdf3)

- INTEGER_SIZE_DB = 4|8 (4)
- INTEGER_SIZE_API = 4|8 (4)
- REAL_SIZE_DB = 4|8 (8 is default)
- LOGGING = true|false|on|off (off)
- MAX_NAME_LENGTH = value (32)

Parameter	Value	Default
<i>PropertyName</i>	string	–
<i>{=}</i>	{ = are is }	–
<i>PropertyValue</i>	string	–

Sideset

Syntax

Sideset *VariableList* . . .

Summary

Define the variables that should be written to the results database. If “variable” is entered, then its name will be used on the output database. If “variable as db_name” is entered, then “db_name” will be the name used on the database for the internal variable “variable”. Multiple “variable” or “variable as db_name” entries are allowed on the same line. The entities that this variable are written to can also be limited or specified with “exclude list_of_entities” or “include list_of_entities”. Face variables are not supported for all database types.

Parameter	Value	Default
<i>VariableList</i>	string. . .	–

Start Time

Syntax

Start Time *{=}* *Start_time*

Summary

Specify the time to start outputting results from this output request block. This time overrides all ‘at time’ and ‘at step’ specifications.

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>Start_time</i>	real	–

Surface

Syntax

Surface *VariableList*. . .

Summary

Define the variables that should be written to the results database. If “variable” is entered, then its name will be used on the output database. If “variable as db_name” is entered, then “db_name” will be the name used on the database for the internal variable “variable”. Multiple “variable” or “variable as db_name” entries are allowed on the same line. The entities that this variable are written to can also be limited or specified with “exclude list_of_entities” or “include list_of_entities”. Face variables are not supported for all database types.

Parameter	Value	Default
<i>VariableList</i>	string. . .	–

Synchronize Output

Syntax

Synchronize Output

Summary

In an analysis with multiple regions, it is sometimes desirable to synchronize the output of results data between the regions. This can be done by adding the SYNCHRONIZE OUTPUT command line to the results output block. If a results block has this set, then it will write output whenever a previous region writes output. The ordering of regions is based on the order in the input file, algorithmic considerations, or by solution control specifications.

Although the USE OUTPUT SCHEDULER command line can also synchronize output between regions, the SYNCHRONIZE OUTPUT command line will synchronize the output with regions where the output frequency is not under the direct control of the Sierra IO system. Examples of this are typically coupled applications where one or more of the codes are not Sierra-based applications such as Alegra and CTH. A results block with SYNCHRONIZE OUTPUT specified will also synchronize its output with the output of the external code.

The SYNCHRONIZE OUTPUT command can be used with other output scheduling commands such as time-based or step-based output specifications.

Termination Time

Syntax

Termination Time {=} *Final_time*

Summary

Specify the time to stop outputting results from this output request block.

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>Final_time</i>	real	–

Timeseries Name

Syntax

Timeseries Name *{=}* *filename*

Summary

Optionally specify a filename for a timeseries file that outputs the root database filename in the order that they are written. This is useful when running on large numbers of processors with many mesh-mods that cause simple disk operations to hang.

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>filename</i>	string	–

Timestep Adjustment Interval

Syntax

Timestep Adjustment Interval *{=}* *Nsteps*

Summary

Specify the number of steps to ‘look ahead’ and adjust the timestep to ensure that the specified output times or simulation end time will be hit ‘exactly’.

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>Nsteps</i>	integer	–

Title

Syntax

Title

Summary

Specify the title to be used for this specific output block.

Use Dynamic Topology Io

Syntax

Use Dynamic Topology Io

Summary

Specify that the app use IO for dynamic topology modifications where the

output files are stored in a single database. Legacy file format for dynamically changing topology results in the creation of multiple files for each output on a mesh modification. This option leverages the ability of netCDF to create mesh groups within a single database and concatenate all mesh files into one. The names of each mesh group are of the form IOSS_MESH_GROUP-??? where ??? is the 1-based output index 1, 2, ..., 10, ..., 100, ... Please note that netCDF has a current limit of 65,536 groups

Use Output Scheduler

Syntax

Use Output Scheduler *Timer_name*

Summary

Associates a predefined output scheduler with this output block (results, restart, heartbeat, or history).

Parameter	Value	Default
<i>Timer_name</i>	string	–

Solid Object

Scope

Fuego Region

Summary

Contains the commands needed to create a solid object

begin Solid Object *Objectname*

Cylinder Base {=} *Param21 Param22 Param23* Top {=} *Param51 Param52* ↪ *Param53* Radius {=} *Param8*

Rectangle Center {=} *Param21 Param22 Param23* Xyz {=} *Param51 Param52* ↪ *Param53*

Sphere Center {=} *Param21 Param22 Param23* Radius {=} *Param5*

end Solid Object *Objectname*

Line Commands

Cylinder

Syntax

Cylinder Base {=} *Param21 Param22 Param23* Top {=} *Param51 Param52 Param53* Radius {=} *Param8*

Summary

Specify cylinder as interface surface, with given radius and two endpoints of axis p1(base) and p2(top). The length of the cylinder is norm(p2 - p1).

Parameter	Value	Default
{=}	{ = are is }	—
<i>Param2</i>	real1 real2 real3	—
{=}	{ = are is }	—
<i>Param5</i>	real1 real2 real3	—
{=}	{ = are is }	—
<i>Param8</i>	real	—

Rectangle

Syntax

Rectangle Center {=} *Param21 Param22 Param23* Xyz {=} *Param51 Param52 Param53*

Summary

Specify rectangle as interface surface, with given center and x,y,z sizes.

Parameter	Value	Default
{=}	{ = are is }	—
<i>Param2</i>	real1 real2 real3	—
{=}	{ = are is }	—
<i>Param5</i>	real1 real2 real3	—

Sphere

Syntax

Sphere Center {=} *Param21 Param22 Param23* Radius {=} *Param5*

Summary

Specify sphere as interface surface with given center and radius.

Parameter	Value	Default
<code>{=}</code>	<code>{= are is}</code>	–
<code>Param2</code>	real1 real2 real3	–
<code>{=}</code>	<code>{= are is}</code>	–
<code>Param5</code>	real	–

Virtual Thermocouple Model On Block

Scope

Fuego Region

Summary

Activate the virtual thermocouple model on the requested mesh block

Description

The virtual thermocouple model assumes one-way coupling between the fluid and the thermocouple. The thermocouple is assumed to be small enough that it does not influence the fluid behavior in any way.

For simulations without a PMR solve the radiative terms of the thermocouple equation are neglected.

begin Virtual Thermocouple Model On Block *BlockName*

Compute Steady Solution

Density `{=}` *Value*

Diameter `{=}` *Value*

Emissivity `{=}` *Value*

Heat Capacity `{=}` *Value*

Initial Temperature `{=}` *Value*

Length `{=}` *Value*

Orientation `{=}` *Lx Ly [Lz]*

end Virtual Thermocouple Model On Block *BlockName*

Line Commands

Compute Steady Solution

Syntax

Compute Steady Solution

Summary

Compute the steady-state temperature for the thermocouple given the current FUEGO/SRYINX solutions. By default, the thermocouple temperature will be updated in a time-accurate manner.

Density

Syntax

Density {=} *Value*

Summary

Specify the density of the thermocouple

Parameter	Value	Default
{=}	{= are is}	–
<i>Value</i>	real	–

Diameter

Syntax

Diameter {=} *Value*

Summary

Specify the diameter of the thermocouple (assumed to be a cylinder)

Parameter	Value	Default
{=}	{= are is}	–
<i>Value</i>	real	–

Emissivity

Syntax

Emissivity {=} *Value*

Summary

Specify the emissivity of the thermocouple

Parameter	Value	Default
{=}	{= are is}	–
<i>Value</i>	real	–

Heat Capacity

Syntax

Heat Capacity {=} *Value*

Summary

Specify the heat capacity of the thermocouple

Parameter	Value	Default
{=}	{= are is}	–
<i>Value</i>	real	–

Initial Temperature

Syntax

Initial Temperature {=} *Value*

Summary

Specify the initial temperature of the thermocouple

Parameter	Value	Default
{=}	{= are is}	–
<i>Value</i>	real	–

Length

Syntax

Length {=} *Value*

Summary

Specify the length of the thermocouple (assumed to be a cylinder)

Parameter	Value	Default
{=}	{= are is}	–
<i>Value</i>	real	–

Orientation

Syntax

Orientation {=} *Lx Ly [Lz]*

Summary

Specify the spatial orientation vector of the thermocouple

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>Lx</i>	real	–
<i>Ly</i>	real	–

Solution Options

Scope

Average Region, Fuego Region

Summary

Specify information regarding the governing equations to be solved.

begin Solution Options *OptionsName*

Activate Acoustic Compressibility Algorithm

Activate Equation *{conserved_enthalpy | continuity | edc_product |*
enthalpy | mixture_fraction | nuclei | progress_variable | scalar_
variance | second_mixture_fraction | solid_volume_fraction | soot |
species | temperature | turbulent_dissipation | turbulent_frequency_
| turbulent_helmholtz_function | turbulent_kinetic_energy | turbulent_
v2 | volume_of_fluid | x_momentum | x_solid_momentum | y_momentum | y_
solid_momentum | z_momentum | z_solid_momentum}

Activate Full Surface Cvfem Gradient Operator For Muscl Scheme

Activate Lighthill Tensor Postprocessing

Activate Species Enthalpy Calculations

Activate Viscous Dissipation Source Term

Compute Steady Solution Using Pseudo Transient Method

Coordinate System *{=}* *{2d | 3d | xaxi | yaxi}*

First Order Upwind Factor *{=}* **Value** [For Equation *{conserved_enthalpy_*
| continuity | edc_product | enthalpy | mixture_fraction | nuclei_
| progress_variable | scalar_variance | second_mixture_fraction |
solid_volume_fraction | soot | species | temperature | turbulent_
dissipation | turbulent_frequency | turbulent_helmholtz_function |
turbulent_kinetic_energy | turbulent_v2 | volume_of_fluid | x_momentum_

```

→| x_solid_momentum | y_momentum | y_solid_momentum | z_momentum | z_
→solid_momentum} ]

```

Fix Pressure To **FixedPressure** At A Single Node

```

Hybrid Upwind Factor {=} Value [ For Equation {conserved_enthalpy_
→| continuity | edc_product | enthalpy | mixture_fraction | nuclei_
→| progress_variable | scalar_variance | second_mixture_fraction |
→solid_volume_fraction | soot | species | temperature | turbulent_
→dissipation | turbulent_frequency | turbulent_helmholtz_function |
→turbulent_kinetic_energy | turbulent_v2 | volume_of_fluid | x_momentum_
→| x_solid_momentum | y_momentum | y_solid_momentum | z_momentum | z_
→solid_momentum} ]

```

```

Hybrid Upwind Method {=} {blending | tanh} [ For Equation {conserved_
→enthalpy | continuity | edc_product | enthalpy | mixture_fraction |
→nuclei | progress_variable | scalar_variance | second_mixture_fraction_
→| solid_volume_fraction | soot | species | temperature | turbulent_
→dissipation | turbulent_frequency | turbulent_helmholtz_function |
→turbulent_kinetic_energy | turbulent_v2 | volume_of_fluid | x_momentum_
→| x_solid_momentum | y_momentum | y_solid_momentum | z_momentum | z_
→solid_momentum} ]

```

```

Hybrid Upwind Shift {=} Value [ For Equation {conserved_enthalpy_
→| continuity | edc_product | enthalpy | mixture_fraction | nuclei_
→| progress_variable | scalar_variance | second_mixture_fraction |
→solid_volume_fraction | soot | species | temperature | turbulent_
→dissipation | turbulent_frequency | turbulent_helmholtz_function |
→turbulent_kinetic_energy | turbulent_v2 | volume_of_fluid | x_momentum_
→| x_solid_momentum | y_momentum | y_solid_momentum | z_momentum | z_
→solid_momentum} ]

```

```

Hybrid Upwind Width {=} Value [ For Equation {conserved_enthalpy_
→| continuity | edc_product | enthalpy | mixture_fraction | nuclei_
→| progress_variable | scalar_variance | second_mixture_fraction |
→solid_volume_fraction | soot | species | temperature | turbulent_
→dissipation | turbulent_frequency | turbulent_helmholtz_function |
→turbulent_kinetic_energy | turbulent_v2 | volume_of_fluid | x_momentum_
→| x_solid_momentum | y_momentum | y_solid_momentum | z_momentum | z_
→solid_momentum} ]

```

Include Continuity Residual Term [With Diagnostics]

Lighthill Tensor Smoothing Iterations {=} **Number**

Maximum Number Of Continuity_Momentum Nonlinear Iterations {=} *Number*

Maximum Number Of Energy_Species Nonlinear Iterations {=} *Number*

Maximum Number Of Gas_Solid_Momentum Nonlinear Iterations {=} *Number*

Maximum Number Of Kepsilon Nonlinear Iterations {=} *Number*

Maximum Number Of Komega Nonlinear Iterations {=} *Number*

Maximum Number Of Ksgs Nonlinear Iterations {=} *Number*

Maximum Number Of Mixture Fraction Nonlinear Iterations {=} *Number*

Maximum Number Of Nonlinear Iterations {=} *Number*

Maximum Number Of Solid Phase Nonlinear Iterations {=} *Number*

Maximum Number Of Soot Nuclei Nonlinear Iterations {=} *Number*

Maximum Number Of Species Nonlinear Iterations {=} *Number*

Maximum Number Of Species_Product Nonlinear Iterations {=} *Number*

Maximum Number Of V2F Nonlinear Iterations {=} *Number*

Maximum Wall Time {=} *WallTime* Hours

Minimum Number Of Nonlinear Iterations {=} *Number*

Nonlinear Residual Norm Tolerance {=} *Tolerance* [For Equation
→ {*conserved_enthalpy* | *continuity* | *edc_product* | *enthalpy* | *mixture_*
→ *fraction* | *nuclei* | *progress_variable* | *scalar_variance* | *second_*
→ *mixture_fraction* | *solid_volume_fraction* | *soot* | *species* | *temperature_*
→ | *turbulent_dissipation* | *turbulent_frequency* | *turbulent_helmholtz_*
→ *function* | *turbulent_kinetic_energy* | *turbulent_v2* | *volume_of_fluid_*
→ | *x_momentum* | *x_solid_momentum* | *y_momentum* | *y_solid_momentum* | *z_*
→ *momentum* | *z_solid_momentum*}]

Nonlinear Stabilization Method {=} {*commutation_error* | *none* |
→ *pointwise_residual_error*} [For Equation {*conserved_enthalpy* |
→ *continuity* | *edc_product* | *enthalpy* | *mixture_fraction* | *nuclei* |
→ *progress_variable* | *scalar_variance* | *second_mixture_fraction* | *solid_*
→ *volume_fraction* | *soot* | *species* | *temperature* | *turbulent_dissipation_*
→ | *turbulent_frequency* | *turbulent_helmholtz_function* | *turbulent_*

```

→kinetic_energy | turbulent_v2 | volume_of_fluid | x_momentum | x_
→solid_momentum | y_momentum | y_solid_momentum | z_momentum | z_solid_
→momentum} ]

```

Omit Density Time Derivative In Continuity Equation [For **OmitSteps**_
→Steps And Blend In Over **BlendSteps** Steps]

```

Output Nonlinear Residual Field For Equation {conserved_enthalpy_
→| continuity | edc_product | enthalpy | mixture_fraction | nuclei_
→| progress_variable | scalar_variance | second_mixture_fraction |
→solid_volume_fraction | soot | species | temperature | turbulent_
→dissipation | turbulent_frequency | turbulent_helmholtz_function |
→turbulent_kinetic_energy | turbulent_v2 | volume_of_fluid | x_momentum_
→| x_solid_momentum | y_momentum | y_solid_momentum | z_momentum | z_
→solid_momentum} As ResName [ On Output Block BlockName ]

```

Periodic Constant Momentum Body Source Term {=} **ConstSrc1 ConstSrc2**_
→**ConstSrc3**

Progress Variable Source Evaluation Time {=} {latest | presolve}

```

Projection Method {=} {fourth_order | second_order | stabilized |
→zeroth_order} Smoothing [ With {characteristic | momentum | timestep}_
→Scaling ]

```

Randomize Pressure

Skip Pressure Update If Continuity Solve Fails

```

Source Term Function {=} FuncStr For Equation {conserved_enthalpy_
→| continuity | edc_product | enthalpy | mixture_fraction | nuclei_
→| progress_variable | scalar_variance | second_mixture_fraction |
→solid_volume_fraction | soot | species | temperature | turbulent_
→dissipation | turbulent_frequency | turbulent_helmholtz_function |
→turbulent_kinetic_energy | turbulent_v2 | volume_of_fluid | x_momentum_
→| x_solid_momentum | y_momentum | y_solid_momentum | z_momentum | z_
→solid_momentum} [ VariableName ]

```

```

Source Term Subroutine {=} Subroutine For Equation {conserved_enthalpy_
→| continuity | edc_product | enthalpy | mixture_fraction | nuclei_
→| progress_variable | scalar_variance | second_mixture_fraction |
→solid_volume_fraction | soot | species | temperature | turbulent_
→dissipation | turbulent_frequency | turbulent_helmholtz_function |
→turbulent_kinetic_energy | turbulent_v2 | volume_of_fluid | x_momentum_
→| x_solid_momentum | y_momentum | y_solid_momentum | z_momentum | z_

```

→solid_momentum} [VariableName]

Stop Simulation If Peak Velocity Exceeds MaxVel

Under Relax {conserved_enthalpy | continuity | edc_product | enthalpy_ | mixture_fraction | nuclei | progress_variable | scalar_variance | second_mixture_fraction | solid_volume_fraction | soot | species | temperature | turbulent_dissipation | turbulent_frequency | turbulent_helmholtz_function | turbulent_kinetic_energy | turbulent_v2 | volume_of_fluid | x_momentum | x_solid_momentum | y_momentum | y_solid_momentum | z_momentum | z_solid_momentum} By Urf [With Implicit Term]

Under Relax Momentum By Urf

Under Relax Pressure By Urf

Under Relax Solid_Momentum By Urf

Under Relax Temperature_Extraction By Urf

Upwind Limiter {=} {minmod | none | superbee | van_albada | van_leer} [For Equation {conserved_enthalpy | continuity | edc_product | enthalpy | mixture_fraction | nuclei | progress_variable | scalar_variance | second_mixture_fraction | solid_volume_fraction | soot | species | temperature | turbulent_dissipation | turbulent_frequency | turbulent_helmholtz_function | turbulent_kinetic_energy | turbulent_v2 | volume_of_fluid | x_momentum | x_solid_momentum | y_momentum | y_solid_momentum | z_momentum | z_solid_momentum}]

Upwind Method {=} {lps | muscl | upw} [For Equation {conserved_enthalpy | continuity | edc_product | enthalpy | mixture_fraction | nuclei | progress_variable | scalar_variance | second_mixture_fraction | solid_volume_fraction | soot | species | temperature | turbulent_dissipation | turbulent_frequency | turbulent_helmholtz_function | turbulent_kinetic_energy | turbulent_v2 | volume_of_fluid | x_momentum | x_solid_momentum | y_momentum | y_solid_momentum | z_momentum | z_solid_momentum}]

Use Equation Solver SolverName For Equation {conserved_enthalpy | continuity | edc_product | enthalpy | mixture_fraction | nuclei | progress_variable | scalar_variance | second_mixture_fraction | solid_volume_fraction | soot | species | temperature | turbulent_dissipation | turbulent_frequency | turbulent_helmholtz_function | turbulent_kinetic_energy | turbulent_v2 | volume_of_fluid | x_momentum | x_solid_momentum | y_momentum | y_solid_momentum | z_momentum | z_solid_momentum}

→momentum}

Use External Continuity Source

Use External Energy Source

Use External Mixture_Fraction Source

Use External Momentum Source

Use External Soot_Mass_Fraction Source

Use External Species Source

Use Lumped Velocity Density Interpolation

Use Radiation Source From External Region [Using Classic_
→Linearization]

Use Shifted Density Iteration

Use Skew Symmetric Central Operator [For Equation {conserved_enthalpy_
→| continuity | edc_product | enthalpy | mixture_fraction | nuclei_
→| progress_variable | scalar_variance | second_mixture_fraction |_
→solid_volume_fraction | soot | species | temperature | turbulent_
→dissipation | turbulent_frequency | turbulent_helmholtz_function |_
→turbulent_kinetic_energy | turbulent_v2 | volume_of_fluid | x_momentum_
→| x_solid_momentum | y_momentum | y_solid_momentum | z_momentum | z_
→solid_momentum}]

begin Acoustic Transfer Output *DefinitionName*
end

begin Beam Radiation Boundary Specification *DefinitionName*
end

begin Buoyancy Model Specification *BuoyModelName*
end

begin Edc Model Specification *EdcSpecName*
end

begin Mesh Motion Specification *DefinitionName*
end

```

begin Multiphase Model Specification DefinitionName
end

begin Point Source DefinitionName
end

begin Rad Transport Spectral Model Specification DefinitionName
end

begin Radiation Transport Equation Model Specification RadModelName
end

begin Time Integration Specification TimeIntSpecName
end

begin Turbulence Model Specification TurbSpecName
end

begin Vof Model Specification DefinitionName
end

end Solution Options OptionsName

```

Line Commands

Activate Acoustic Compressibility Algorithm

Syntax

Activate Acoustic Compressibility Algorithm

Summary

Variable thermodynamic pressure

Description

This option will allow for closed system pressurization either through heat-up or inflow of fluid.

The algorithm will add the substantial derivative of pressure, $\frac{\partial p}{\partial t} + u_j \frac{\partial p}{\partial x_j}$, to the laminar or turbulent enthalpy transport equation and to the laminar temperature transport equation. Additionally, the viscous work term $u_i \frac{\partial \tau_{ij}}{\partial x_j}$ will be added to the turbulent enthalpy equation. An implicit term in the continuity solve is added through the time density derivative. As such, Cantera support is required. The convective terms within the continuity solve are neglected.

Caveats for this model:

- 1) The Cantera material model evaluator must be used.
- 2) The initial pressure and any boundary condition pressures must be specified with respect to the datum pressure.

If a zero datum pressure is specified, then all initial and boundary pressures will be in absolute units. If this is a coupled structural simulation, then the surface traction due to this pressure will need to be counteracted with a load on the “back side” that is equivalent to the ambient pressure in absolute units.

If a non-zero datum pressure is specified, then all initial and boundary pressures will be in relative units with respect to this datum. Pressure can then be thought of as a gauge pressure with respect to the datum. The “back side” load in structural simulations must be set accordingly. (For example, if the datum is set equal to the external ambient pressure, 1 atm, and the initial pressure is set to zero, then the initial surface traction force due to pressure will be zero and no “back side” load due to the ambient pressure should be specified.)

Activate Equation

Syntax

Activate Equation {*conserved_enthalpy* | *continuity* | *edc_product* | *enthalpy* | *mixture_fraction* | *nuclei* | *progress_variable* | *scalar_variance* | *second_mixture_fraction* | *solid_volume_fraction* | *soot* | *species* | *temperature* | *turbulent_dissipation* | *turbulent_frequency* | *turbulent_helmholtz_function* | *turbulent_kinetic_energy* | *turbulent_v2* | *volume_of_fluid* | *x_momentum* | *x_solid_momentum* | *y_momentum* | *y_solid_momentum* | *z_momentum* | *z_solid_momentum*}

Summary

Activate the specified equation.

Parameter	Value	Default
<i>Equations</i>	{ <i>conserved_enthalpy</i> <i>continuity</i> <i>edc_product</i> <i>enthalpy</i> <i>mixture_fraction</i> <i>nuclei</i> <i>progress_variable</i> <i>scalar_variance</i> <i>second_mixture_fraction</i> <i>solid_volume_fraction</i> <i>soot</i> <i>species</i> <i>temperature</i> <i>turbulent_dissipation</i> <i>turbulent_frequency</i> <i>turbulent_helmholtz_function</i> <i>turbulent_kinetic_energy</i> <i>turbulent_v2</i> <i>volume_of_fluid</i> <i>x_momentum</i> <i>x_solid_momentum</i> <i>y_momentum</i> <i>y_solid_momentum</i> <i>z_momentum</i> <i>z_solid_momentum</i> }	–

Activate Full Surface Cvfem Gradient Operator For Muscl Scheme

Syntax

Activate Full Surface Cvfem Gradient Operator For Muscl Scheme

Summary

Use full stencil for gradient used in MUSCL convection operator

Description

The default gradient operator for the MUSCL scheme is the edge-based stencil. This option keeps integration points at the subcontrol surface points.

Activate Lighthill Tensor Postprocessing**Syntax**

Activate Lighthill Tensor Postprocessing

Summary

Postprocesses the nodal divergence of the Lighthill tensor

Description

This calculates the nodal divergence of the Lighthill tensor, used for acoustic analysis.

Activate Species Enthalpy Calculations**Syntax**

Activate Species Enthalpy Calculations

Summary

Enables calculation of species enthalpy

Description

This forces the calculation of species enthalpy, needed primarily for coupled Fuego-Aria problems.

Activate Viscous Dissipation Source Term**Syntax**

Activate Viscous Dissipation Source Term

Summary

Add viscous dissipation source term

Description

For low speed viscous dissipation effects, this source term will provide the viscous work source term in the static enthalpy equation. This source term is a subset of the full acoustically compressible source term option, however, the substantial pressure derivative is omitted.

Compute Steady Solution Using Pseudo Transient Method**Syntax**

Compute Steady Solution Using Pseudo Transient Method

Summary

Compute a steady-state solution using the pseudo-transient method (time march to steady solution).

Description

The solution will march forward in time until either the stopping time is reached or the steady convergence criterion is met. Convergence to steady state is detected when all equations meet their nonlinear residual norm tolerances after the first nonlinear iteration, since this will only occur as the solution stops changing between time steps. The nonlinear residual norm tolerances should be set small enough to prevent false positives. Also make sure the simulation time is set to be fairly large, to prevent a premature end to the simulation before convergence is achieved.

If you are using solution control, then you also need to test for a region parameter to stop the simulation. In your PARAMETERS FOR TRANSIENT solution control block, add the line (assuming your Fuego region is called `fuego_region`):

```
CONVERGED WHEN "fuego_region.REGION_STEADY_STATE == 1"
```

Coordinate System

Syntax

Coordinate System *{=}* {2d | 3d | xaxi | yaxi}

Summary

Specify the coordinate system.

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>CoordSys</i>	{2d 3d xaxi yaxi}	3D

First Order Upwind Factor

Syntax

First Order Upwind Factor *{=}* *Value* [For Equation {*conserved_enthalpy* | *continuity* | *edc_product* | *enthalpy* | *mixture_fraction* | *nuclei* | *progress_variable* | *scalar_variance* | *second_mixture_fraction* | *solid_volume_fraction* | *soot* | *species* | *temperature* | *turbulent_dissipation* | *turbulent_frequency* | *turbulent_helmholtz_function* | *turbulent_kinetic_energy* | *turbulent_v2* | *volume_of_fluid* | *x_momentum* | *x_solid_momentum* | *y_momentum* | *y_solid_momentum* | *z_momentum* | *z_solid_momentum*}]

Summary

First-order upwind factor, $0 < x < 1$

Description

This value specifies the explicit upwind blending between pure upwind and the chosen convection operator, e.g., $UPW*(firstOrderUpwind) + (1-firstOrderUpwind)*(blendedUpwindCentral)$.

where UPW is the pure first order upwind value and `blendedUpwindCentral` is a

blend between the selected upwind method and central difference operator based on the local cell Peclet number (see Hybrid Upwind Factor line command). The value can be a time dependent string function.

Values for individual equation sets may be set using optional token. Using both (in either order):

- FIRST ORDER UPWIND FACTOR = String
- FIRST ORDER UPWIND FACTOR = String FOR EQUATION Equations

Will result in the particular equation set to specified value while all others set to general value.

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>Value</i>	“string”	1.0

Fix Pressure To

Syntax

Fix Pressure To *FixedPressure* At A Single Node

Summary

Sets a dirichlet for pressure at a single arbitrary node. This is required for a well posed pressure equation if none of the boundaries specify pressure (e.g. open).

Parameter	Value	Default
<i>FixedPressure</i>	real	0.0

Hybrid Upwind Factor

Syntax

Hybrid Upwind Factor *{=}* *Value* [For Equation *{conserved_enthalpy | continuity | edc_product | enthalpy | mixture_fraction | nuclei | progress_variable | scalar_variance | second_mixture_fraction | solid_volume_fraction | soot | species | temperature | turbulent_dissipation | turbulent_frequency | turbulent_helmholtz_function | turbulent_kinetic_energy | turbulent_v2 | volume_of_fluid | x_momentum | x_solid_momentum | y_momentum | y_solid_momentum | z_momentum | z_solid_momentum}*]

Summary

Hybrid upwinding factor.

Description

The upwind schemes are blended with a centered scheme. The HYBRID UPWIND FACTOR is a multiplier against the cell Peclet number used in the switching scheme (see First Order Upwind Factor line command).

- A HYBRID UPWIND FACTOR = 0.0 results in all centered.
- A HYBRID UPWIND FACTOR = 1.0 results in default hybrid.
- A HYBRID UPWIND FACTOR >> 1.0 results in all upwind.

Values for individual equation sets may be set using optional token. Using both (in either order):

- HYBRID UPWIND FACTOR = String
- HYBRID UPWIND FACTOR = String FOR EQUATION Equations

Will result in the particular equation set to specified value while all others set to general value. The value can be specified as a time dependent string function or a constant.

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>Value</i>	“string”	1.0

Hybrid Upwind Method

Syntax

Hybrid Upwind Method *{=}* *{blending | tanh}* [For Equation *{conserved_enthalpy | continuity | edc_product | enthalpy | mixture_fraction | nuclei | progress_variable | scalar_variance | second_mixture_fraction | solid_volume_fraction | soot | species | temperature | turbulent_dissipation | turbulent_frequency | turbulent_helmholtz_function | turbulent_kinetic_energy | turbulent_v2 | volume_of_fluid | x_momentum | x_solid_momentum | y_momentum | y_solid_momentum | z_momentum | z_solid_momentum}*]

Summary

Hybridizing method between central and upwind using Peclet number

Description

BLENDING and TANH approaches are currently supported.

Function χ determines the ratio between user-chosen upwind (χ) and central ($1-\chi$) operators. The need for an upwind operator is affected by cell-Peclet number.

BLENDING uses HYBRID UPWIND FACTOR (ζ) and the function is,

$$\chi = \frac{(\zeta Pe)^2}{5 + (\zeta Pe)^2}.$$

TAHH follows hyperbolic tangent profile between χ and Pe. It uses shift p (HYBRID UPWIND SHIFT) and width w (HYBRID UPWIND WIDTH) parameters as, $\chi = \frac{1}{2} [1 + \tanh(\frac{Pe-p}{w})]$. Tanh is centered ($\chi = 0.5$) when Peclet number is at the shifting factor p. Width determines how fast χ changes with Peclet number as follows:

- $\chi = 0.5$ at $Pe=p$.
- $\chi = 0.8808$ and 0.1192 at $Pe=p+w$ and $p-w$.
- $\chi = 0.9820$ and 0.0180 at $Pe=p+2w$ and $p-2w$.
- $\chi = 0.9975$ and 0.0025 at $Pe=p+3w$ and $p-3w$.
- $\chi = 0.9997$ and 0.0003 at $Pe=p+4w$ and $p-4w$.

TANH allows users to effectively remove upwind contribution for lower Pe . In the other extreme, one can enforce user-chosen upwind at all Pe if $p < 0.0$ and $w \ll 1.0$ (ex> $p=-1.0$, $w=1e-10$).

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>HybridMethod</i>	{ blending tanh }	BLENDING

Hybrid Upwind Shift

Syntax

Hybrid Upwind Shift *{=}* *Value* [For Equation *{conserved_enthalpy | continuity | edc_product | enthalpy | mixture_fraction | nuclei | progress_variable | scalar_variance | second_mixture_fraction | solid_volume_fraction | soot | species | temperature | turbulent_dissipation | turbulent_frequency | turbulent_helmholtz_function | turbulent_kinetic_energy | turbulent_v2 | volume_of_fluid | x_momentum | x_solid_momentum | y_momentum | y_solid_momentum | z_momentum | z_solid_momentum}*]

Summary

Shifting factor for TANH hybrid approach. Can be specified as a time dependent string function or a constant.

Description

(see HYBRID UPWIND METHOD description)

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>Value</i>	“string”	0.0

Hybrid Upwind Width

Syntax

Hybrid Upwind Width *{=}* *Value* [For Equation *{conserved_enthalpy | continuity | edc_product | enthalpy | mixture_fraction | nuclei | progress_variable | scalar_variance | second_mixture_fraction | solid_volume_fraction | soot | species | temperature | turbulent_dissipation | turbulent_frequency | turbulent_helmholtz_function | turbulent_kinetic_energy |*

*turbulent_v2 | volume_of_fluid | x_momentum | x_solid_momentum |
y_momentum | y_solid_momentum | z_momentum | z_solid_momentum}]*

Summary

Width factor for TANH hybrid approach

Description

Minimum value for this parameter is 1e-10. (see HYBRID UPWIND METHOD description) Can be specified as a time dependent string function or a constant value.

Parameter	Value	Default
<i>{=}</i>	{= are is }	–
<i>Value</i>	“string”	1.0

Include Continuity Residual Term

Syntax

Include Continuity Residual Term [With Diagnostics]

Summary

Include the continuity residual term in transport equations

Description

Continuity is not exactly satisfied during the momentum solve for variable density flows since the mass flux is lagged while the density is updated with new properties. Including the continuity error in momentum can keep the momentum prediction better behaved. The residual should be on the order of the linear solver tolerance for other equations, but including the term can also make the other solves more robust to a bad continuity solve.

This term is always included when using VOF or a deforming mesh.

Lighthill Tensor Smoothing Iterations

Syntax

Lighthill Tensor Smoothing Iterations *{=}* *Number*

Summary

Number of smoothing iterations for the divergence of the Lighthill tensor.

Parameter	Value	Default
<i>{=}</i>	{= are is }	–
<i>Number</i>	integer	–

Maximum Number Of Continuity_Momentum Nonlinear Iterations

Syntax

Maximum Number Of Continuity_Momentum Nonlinear Iterations *{=}* *Number*

Summary

Maximum number of nonlinear iterations to take within the momentum/continuity solve.

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>Number</i>	integer	1

Maximum Number Of Energy_Species Nonlinear Iterations

Syntax

Maximum Number Of Energy_Species Nonlinear Iterations *{=}* *Number*

Summary

Maximum number of nonlinear iterations to take within the energy-species grouping.

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>Number</i>	integer	1

Maximum Number Of Gas_Solid_Momentum Nonlinear Iterations

Syntax

Maximum Number Of Gas_Solid_Momentum Nonlinear Iterations *{=}* *Number*

Summary

Maximum number of nonlinear iterations to take within the gas/solid momentum sets of equations.

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>Number</i>	integer	1

Maximum Number Of Kepsilon Nonlinear Iterations

Syntax

Maximum Number Of Kepsilon Nonlinear Iterations *{=}* *Number*

Summary

Maximum number of nonlinear iterations to take within the k-epsilon turbulence model equations grouping.

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>Number</i>	integer	1

Maximum Number Of Komega Nonlinear Iterations

Syntax

Maximum Number Of Komega Nonlinear Iterations **{=}** *Number*

Summary

Maximum number of nonlinear iterations to take within the k-omega turbulence model equations grouping.

Parameter	Value	Default
{=}	{= are is}	–
<i>Number</i>	integer	1

Maximum Number Of Ksgs Nonlinear Iterations

Syntax

Maximum Number Of Ksgs Nonlinear Iterations **{=}** *Number*

Summary

Maximum number of nonlinear iterations to take within the ksgs turbulence model equations grouping.

Parameter	Value	Default
{=}	{= are is}	–
<i>Number</i>	integer	1

Maximum Number Of Mixture Fraction Nonlinear Iterations

Syntax

Maximum Number Of Mixture Fraction Nonlinear Iterations **{=}** *Number*

Summary

Maximum number of nonlinear iterations to take within the mixture fraction equations grouping.

Parameter	Value	Default
{=}	{= are is}	–
<i>Number</i>	integer	1

Maximum Number Of Nonlinear Iterations

Syntax

Maximum Number Of Nonlinear Iterations **{=}** *Number*

Summary

Maximum number of nonlinear iterations to take within the time step of the Fuego region.

Parameter	Value	Default
<i>{=}</i>	{= are is }	–
<i>Number</i>	integer	1

Maximum Number Of Solid Phase Nonlinear Iterations

Syntax

Maximum Number Of Solid Phase Nonlinear Iterations *{=}* *Number*

Summary

Maximum number of nonlinear iterations to take within the solid momentum/continuity sets of equations.

Parameter	Value	Default
<i>{=}</i>	{= are is }	–
<i>Number</i>	integer	1

Maximum Number Of Soot Nuclei Nonlinear Iterations

Syntax

Maximum Number Of Soot Nuclei Nonlinear Iterations *{=}* *Number*

Summary

Maximum number of nonlinear iterations to take within the soot nuclei equations grouping.

Parameter	Value	Default
<i>{=}</i>	{= are is }	–
<i>Number</i>	integer	1

Maximum Number Of Species Nonlinear Iterations

Syntax

Maximum Number Of Species Nonlinear Iterations *{=}* *Number*

Summary

Maximum number of nonlinear iterations for the species equations. If the EDC product transport feature is active, then the SPECIES_PRODUCT nonlinear iteration count should be set instead.

Parameter	Value	Default
<i>{=}</i>	{= are is }	–
<i>Number</i>	integer	1

Maximum Number Of Species_Product Nonlinear Iterations

Syntax

Maximum Number Of Species_Product Nonlinear Iterations *{=} Number*

Summary

Maximum number of nonlinear iterations to take within the Species/EDC_Product grouping. This is only used if the EDC model is active and the EDC product transport feature is being used.

Parameter	Value	Default
<i>{=}</i>	{= are is }	–
<i>Number</i>	integer	1

Maximum Number Of V2F Nonlinear Iterations

Syntax

Maximum Number Of V2F Nonlinear Iterations *{=} Number*

Summary

Maximum number of nonlinear iterations to take within the v2f turbulence model equations grouping.

Parameter	Value	Default
<i>{=}</i>	{= are is }	–
<i>Number</i>	integer	1

Maximum Wall Time

Syntax

Maximum Wall Time *{=} WallTime* Hours

Summary

Specify a maximum wall time to let the simulation end gracefully and output before slurm kills it.

Parameter	Value	Default
<i>{=}</i>	{= are is }	–
<i>WallTime</i>	real	Infinite

Minimum Number Of Nonlinear Iterations

Syntax

Minimum Number Of Nonlinear Iterations *{=} Number*

Summary

Minimum number of nonlinear iterations to take within the time step of the Fuego region.

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>Number</i>	integer	1

Nonlinear Residual Norm Tolerance

Syntax

Nonlinear Residual Norm Tolerance *{=}* *Tolerance* [For Equation *{conserved_enthalpy | continuity | edc_product | enthalpy | mixture_fraction | nuclei | progress_variable | scalar_variance | second_mixture_fraction | solid_volume_fraction | soot | species | temperature | turbulent_dissipation | turbulent_frequency | turbulent_helmholtz_function | turbulent_kinetic_energy | turbulent_v2 | volume_of_fluid | x_momentum | x_solid_momentum | y_momentum | y_solid_momentum | z_momentum | z_solid_momentum}*]

Summary

Nonlinear convergence tolerance within a time step in the Fuego region.

Values for individual equation sets may be set using the optional token. Using both (in either order):

```
NONLINEAR RESIDUAL NORM TOLERANCE = {Real}
NONLINEAR RESIDUAL NORM TOLERANCE = {Real} FOR EQUATION
↪{Equations}
```

Will result in the particular equation set to specified value while all others set to general value.

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>Tolerance</i>	real	1.0e-15

Nonlinear Stabilization Method

Syntax

Nonlinear Stabilization Method *{=}* *{commutation_error | none | pointwise_residual_error}* [For Equation *{conserved_enthalpy | continuity | edc_product | enthalpy | mixture_fraction | nuclei | progress_variable | scalar_variance | second_mixture_fraction | solid_volume_fraction | soot | species | temperature | turbulent_dissipation | turbulent_frequency | turbulent_helmholtz_function | turbulent_kinetic_energy | turbulent_v2 | volume_of_fluid | x_momentum | x_solid_momentum | y_momentum | y_solid_momentum | z_momentum | z_solid_momentum}*]

Summary

Specify a artificial viscosity stabilization; default is NONE.

Description

Values for individual equation sets may be set using optional token. Using both (in either order):

- NSO METHOD = NSOMethod
- NSO METHOD = NSOMethod FOR EQUATION Equations

Will result in the particular equation set to specified value while all others set to general value.

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>NSOMethod</i>	{ commutation_error none pointwise_residual_error }	NO_NSO

Omit Density Time Derivative In Continuity Equation

Syntax

Omit Density Time Derivative In Continuity Equation [For *OmitSteps* Steps And Blend In Over *BlendSteps* Steps]

Summary

Remove density time derivative in continuity equation

Description

Remove the density time derivative from the continuity equation. This feature is required for closed boundary flows with accumulation.

The optional arguments let you omit it for a certain number of timesteps at the start of the simulation, then gradually include it over a number of steps.

Output Nonlinear Residual Field For Equation

Syntax

Output Nonlinear Residual Field For Equation { *conserved_enthalpy* | *continuity* | *edc_product* | *enthalpy* | *mixture_fraction* | *nuclei* | *progress_variable* | *scalar_variance* | *second_mixture_fraction* | *solid_volume_fraction* | *soot* | *species* | *temperature* | *turbulent_dissipation* | *turbulent_frequency* | *turbulent_helmholtz_function* | *turbulent_kinetic_energy* | *turbulent_v2* | *volume_of_fluid* | *x_momentum* | *x_solid_momentum* | *y_momentum* | *y_solid_momentum* | *z_momentum* | *z_solid_momentum* } As *ResName* [On Output Block *BlockName*]

Summary

Generates output of nonlinear residuals for the requested equation.

Description

Provide nonlinear residual for output for specified equation. If the optional output block name is specified, then the residual will only be written to that output block.

Parameter	Value	Default
<i>Equations</i>	{conserved_enthalpy continuity edc_product enthalpy mixture_fraction nuclei progress_variable scalar_variance second_mixture_fraction solid_volume_fraction soot species temperature turbulent_dissipation turbulent_frequency turbulent_helmholtz_function turbulent_kinetic_energy turbulent_v2 volume_of_fluid x_momentum x_solid_momentum y_momentum y_solid_momentum z_momentum z_solid_momentum}	–
<i>Residuals</i>	string	–

Periodic Constant Momentum Body Source Term

Syntax

Periodic Constant Momentum Body Source Term *{=}* *ConstSrc1 ConstSrc2 ConstSrc3*

Summary

Add constant body force due to periodic config

Description

For periodic BCS, commonly a constant body force is applied to drive the flow. This line command allows one to provide a constant body force in three dimensions. If more complex sources are needed, the user sub source term procedure is required.

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>ConstSrc</i>	real1 real2 real3	–

Progress Variable Source Evaluation Time

Syntax

Progress Variable Source Evaluation Time *{=}* *{latest | presolve}*

Summary

Evaluation point for sequences of interdependent progress variable source terms. Either the most recently nonlinear update is used in the order in which the progress variables are solved, or the progress variable source terms are evaluated together presolve.

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>EvalTime</i>	{latest presolve}	–

Projection Method

Syntax

Projection Method *{=}* *{fourth_order | second_order | stabilized | zeroth_order}*
Smoothing [With *{characteristic | momentum | timestep}* Scaling]

Summary

Specify choice of projection method.

Description

The smoothing choice may include zeroth, second, or fourth order. No smoothing (zeroth) may allow pressure-velocity decoupling.

The scaling term may be specified. This scaling term is related to the factorization approximation to the inverse of the momentum matrix.

Time step scaling may show results that are sensitive to the chosen simulation dt at coarse meshes. This error should vanish as the pressure field approached a linear shape, or refinement is performed. Note that characteristic scaling also has the same error, however, its manifestation is less obvious.

The stabilized option uses a fourth order smoothing term and characteristic scaling along with an additional dt stabilizing term.

In general, the stabilized and “fourth order smoothing” timestep scaling allows for larger time steps. Characteristic scaling seems to limit CFL to below unity, presumably due to stability loss during nodal projection, i.e., splitting error is $(I - \tau A)G(\Delta P)$.

“Momentum Scaling”, uses the diagonal of the momentum equation as the scaling term. While the leading order term with this method will be similar to the timestep scaling scheme, it can sometimes offer better stability since it also includes effects from the other terms in the momentum equation.

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>ProjectionMethod</i>	{ fourth_order second_order stabilized zeroth_order }	–

Randomize Pressure

Syntax

Randomize Pressure

Summary

Set a random pressure field for initial guess

Description

Randomize the initial guess to the linear solve for pressure. The randomization

is imposed after the nonlinear residual is computed.

Skip Pressure Update If Continuity Solve Fails

Syntax

Skip Pressure Update If Continuity Solve Fails

Summary

Do not update the pressure field or mdot if the continuity solve fails

Description

If the continuity solve fails the resulting pressure delta may be large or non-physical. Activating this option skips the pressure update and mdot update when the solver fails. Repeated solver failures should be watched for in the log file.

This is a beta feature.

Source Term Function

Syntax

Source Term Function {=} *FuncStr* For Equation {*conserved_enthalpy* | *continuity* | *edc_product* | *enthalpy* | *mixture_fraction* | *nuclei* | *progress_variable* | *scalar_variance* | *second_mixture_fraction* | *solid_volume_fraction* | *soot* | *species* | *temperature* | *turbulent_dissipation* | *turbulent_frequency* | *turbulent_helmholtz_function* | *turbulent_kinetic_energy* | *turbulent_v2* | *volume_of_fluid* | *x_momentum* | *x_solid_momentum* | *y_momentum* | *y_solid_momentum* | *z_momentum* | *z_solid_momentum*} [*VariableName*]

Summary

Source term string function to use for the given equation. Registered variables with aliases include time (t), spatial coordinates (x,y,z), velocity (u,v,w), density (rho), and pressure (p). Additionally, any valid global variable or nodal variable can be used with its full name. Vector variables, like mass fraction, must be indexed numerically (e.g. “mass_fraction[3]”)

The function string must be enclosed in quotes if it has spaces or commas. For example: Source Term Function for x_momentum = “min(1, 0.1*t)”

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>Func-Str</i>	“string”	–
<i>Equation</i>	{conserved_enthalpy continuity edc_product enthalpy mixture_fraction nuclei progress_variable scalar_variance second_mixture_fraction solid_volume_fraction soot species temperature turbulent_dissipation turbulent_frequency turbulent_helmholtz_function turbulent_kinetic_energy turbulent_v2 volume_of_fluid x_momentum x_solid_momentum y_momentum y_solid_momentum z_momentum z_solid_momentum}	–

Source Term Subroutine

Syntax

Source Term Subroutine *{=}* *Subroutine* For Equation *{conserved_enthalpy | continuity | edc_product | enthalpy | mixture_fraction | nuclei | progress_variable | scalar_variance | second_mixture_fraction | solid_volume_fraction | soot | species | temperature | turbulent_dissipation | turbulent_frequency | turbulent_helmholtz_function | turbulent_kinetic_energy | turbulent_v2 | volume_of_fluid | x_momentum | x_solid_momentum | y_momentum | y_solid_momentum | z_momentum | z_solid_momentum}* [*VariableName*]

Summary

Source term user subroutine for the given equation. This is often useful in verification studies where one wishes to use a manufactured solution and must provide source terms for various governing equations.

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>Sub-routine</i>	string	–
<i>Equations</i>	{conserved_enthalpy continuity edc_product enthalpy mixture_fraction nuclei progress_variable scalar_variance second_mixture_fraction solid_volume_fraction soot species temperature turbulent_dissipation turbulent_frequency turbulent_helmholtz_function turbulent_kinetic_energy turbulent_v2 volume_of_fluid x_momentum x_solid_momentum y_momentum y_solid_momentum z_momentum z_solid_momentum}	–

Stop Simulation If Peak Velocity Exceeds

Syntax

Stop Simulation If Peak Velocity Exceeds *MaxVel*

Summary

Abort the simulation if velocities get too large.

Description

By default Fuego will continue time stepping as the simulation diverges and will go until velocities overflow or solvers start returning NaN or Inf.

If you want it to stop sooner than that, you can set a peak velocity magnitude to abort at.

Parameter	Value	Default
<i>MaxVel</i>	real	infinity

Under Relax

Syntax

Under Relax *{conserved_enthalpy | continuity | edc_product | enthalpy | mixture_fraction | nuclei | progress_variable | scalar_variance | second_mixture_fraction | solid_volume_fraction | soot | species | temperature | turbulent_dissipation | turbulent_frequency | turbulent_helmholtz_function | turbulent_kinetic_energy | turbulent_v2 | volume_of_fluid | x_momentum | x_solid_momentum | y_momentum | y_solid_momentum | z_momentum | z_solid_momentum}* By *Urf* [With Implicit Term]

Summary

Under relaxation factor for the given equation.

Description

Implicit relaxation is applied to the momentum equations. Explicit relaxation is applied to the pressure update. Transport equations are relaxed explicitly unless the “WITH IMPLICI TERM” option is used.

Under-relaxation can be a constant value or a function of time (t). If the function used has spaces or commas, it should be enclosed in quotes. The value will be internally clipped between 1e-6 and 1.

Parameter	Value	Default
<i>Equations</i>	{conserved_enthalpy continuity edc_product enthalpy mixture_fraction nuclei progress_variable scalar_variance second_mixture_fraction solid_volume_fraction soot species temperature turbulent_dissipation turbulent_frequency turbulent_helmholtz_function turbulent_kinetic_energy turbulent_v2 volume_of_fluid x_momentum x_solid_momentum y_momentum y_solid_momentum z_momentum z_solid_momentum}	–
<i>Urf</i>	“string”	1.0

Under Relax Momentum By

Syntax

Under Relax Momentum By *Urf*

Summary

Under relaxation factor for the momentum equations.

Under-relaxation can be a constant value or a function of time (t). If the function used has spaces or commas, it should be enclosed in quotes. The value will be internally clipped between 1e-6 and 1.

Parameter	Value	Default
<i>Urf</i>	“string”	–

Under Relax Pressure By

Syntax

Under Relax Pressure By *Urf*

Summary

Under relaxation factor for the pressure. This is equivalent to specifying an URF on continuity, and is provided for backward compatibility.

Under-relaxation can be a constant value or a function of time (t). If the function used has spaces or commas, it should be enclosed in quotes. The value will be

internally clipped between 1e-6 and 1.

Parameter	Value	Default
<i>Urf</i>	“string”	–

Under Relax Solid_Momentum By

Syntax

Under Relax Solid_Momentum By *Urf*

Summary

Under relaxation factor for the solid-phase momentum equations.

Under-relaxation can be a constant value or a function of time (t). If the function used has spaces or commas, it should be enclosed in quotes. The value will be internally clipped between 1e-6 and 1.

Parameter	Value	Default
<i>Urf</i>	“string”	–

Under Relax Temperature_Extraction By

Syntax

Under Relax Temperature_Extraction By *Urf*

Summary

Under relaxation factor for the temperature extraction from enthalpy

Description

Relax the temperature computed from the enthalpy. This gives a temperature that is not entirely consistent with the current state (composition and enthalpy), and will destroy time-accuracy unless sufficient Picard loops are taken. However, it may be useful for steady-state computations where species and energy equations are not being coupled strongly or solved accurately.

Under-relaxation can be a constant value or a function of time (t). If the function used has spaces or commas, it should be enclosed in quotes. The value will be internally clipped between 1e-6 and 1.

Parameter	Value	Default
<i>Urf</i>	“string”	–

Upwind Limiter

Syntax

Upwind Limiter {=} {minmod | none | superbee | van_albada | van_leer} [For Equation {conserved_enthalpy | continuity | edc_product | enthalpy |

mixture_fraction | nuclei | progress_variable | scalar_variance | second_mixture_fraction | solid_volume_fraction | soot | species | temperature | turbulent_dissipation | turbulent_frequency | turbulent_helmholtz_function | turbulent_kinetic_energy | turbulent_v2 | volume_of_fluid | x_momentum | x_solid_momentum | y_momentum | y_solid_momentum | z_momentum | z_solid_momentum}]

Summary

Specify a limiter for convection operator; default is SUPERBEE.

Description

Limiter functions are valid only for the MUSCL scheme.

Values for individual equation sets may be set using optional token. Using both (in either order):

- UPWIND LIMITER = UpwindLimiter
- UPWIND LIMITER = UpwindLimiter FOR EQUATION Equations

Will result in the particular equation set to specified value while all others set to general value.

Note: Rotational invariance of the code is not expected while using a limiter function.

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>UpwindLimiter</i>	{minmod none superbee van_albada van_leer}	SUPERBEE

Upwind Method

Syntax

Upwind Method *{=}* *{lps | muscl | upw}* [For Equation *{conserved_enthalpy | continuity | edc_product | enthalpy | mixture_fraction | nuclei | progress_variable | scalar_variance | second_mixture_fraction | solid_volume_fraction | soot | species | temperature | turbulent_dissipation | turbulent_frequency | turbulent_helmholtz_function | turbulent_kinetic_energy | turbulent_v2 | volume_of_fluid | x_momentum | x_solid_momentum | y_momentum | y_solid_momentum | z_momentum | z_solid_momentum}]*

Summary

Upwind method for convective terms

Description

All methods are hybrid in the sense that a centered scheme is blended based on the local Peclet number.

Values for individual equation sets may be set using optional token. Using both (in either order):

- UPWIND METHOD = UpwindMethod
- UPWIND METHOD = UpwindMethod FOR EQUATION Equations

Will result in the particular equation set to specified value while all others set to general value.

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>UpwindMethod</i>	{ lps muscl upw }	LPS

Use Equation Solver

Syntax

Use Equation Solver *SolverName* For Equation *{conserved_enthalpy | continuity | edc_product | enthalpy | mixture_fraction | nuclei | progress_variable | scalar_variance | second_mixture_fraction | solid_volume_fraction | soot | species | temperature | turbulent_dissipation | turbulent_frequency | turbulent_helmholtz_function | turbulent_kinetic_energy | turbulent_v2 | volume_of_fluid | x_momentum | x_solid_momentum | y_momentum | y_solid_momentum | z_momentum | z_solid_momentum}*

Summary

Link an equation solver to an equation set.

Description

For example, if a solver block “scalar” was created using the Tpetra package, e.g., BEGIN TPETRA EQUATION SOLVER scalar and the equation set was the u-component of momentum then the line command would be as follows: USE EQUATION SOLVER scalar FOR EQUATION X-Momentum.

This command can be omitted, and a default solver will be assigned (either the HIGH_ASPECT_CONTINUITY or SCALAR_TRANSPORT preset solvers). The default continuity solver is GMRES with the MueLu preconditioner and the default scalar transport solver is GMRES with the SGS preconditioner.

Parameter	Value	Default
<i>Solver Name</i>	string	–
<i>Equations</i>	{conserved_enthalpy continuity edc_product enthalpy mixture_fraction nuclei progress_variable scalar_variance second_mixture_fraction solid_volume_fraction soot species temperature turbulent_dissipation turbulent_frequency turbulent_helmholtz_function turbulent_kinetic_energy turbulent_v2 volume_of_fluid x_momentum x_solid_momentum y_momentum y_solid_momentum z_momentum z_solid_momentum}	–

Use External Continuity Source

Syntax

Use External Continuity Source

Summary

Add external species source term from a transfer

Description

Add source terms to the continuity equation from a nodal field transferred to this region, e.g. from a Fuego particle region. The field continuity_source will be added to the RHS of the continuity equation; this fields should have units of rate-of-change of mass per volume, so that multiplication by the control volume gives the correct source term. The transfer operation should send to the variables at state “none”.

Use External Energy Source

Syntax

Use External Energy Source

Summary

Add external energy source term from a transfer

Description

Add source terms to the temperature or enthalpy equations from a nodal field transferred to this region, e.g. from a Fuego particle region. The field energy_source will be added to the RHS of the energy equation; this fields should have units of rate-of-change of energy per volume, so that multiplication by the control volume gives the correct source term. The transfer operation should send to the variables at state “none”.

Use External Mixture_Fraction Source

Syntax

Use External Mixture_Fraction Source

Summary

Add external species source term from a transfer

Description

Add source terms to the mixture fraction equation from a nodal field transferred to this region, e.g. from a Fuego particle region. The field `mixture_fraction_source` will be added to the RHS of the mixture fraction equation; this fields should have units of rate-of-change of mass per volume, so that multiplication by the control volume gives the correct source term. The transfer operation should send to the variables at state “none”.

Use External Momentum Source**Syntax**

Use External Momentum Source

Summary

Add external momentum source terms from a transfer

Description

Add source terms to the momentum equations from a nodal field transferred to this region, e.g. from a Fuego particle region. The fields `x_momentum_source`, `y_momentum_source`, and `z_momentum_source` will be added to the RHS of the momentum equations; these fields should have units of rate-of-change of momentum per volume, so that multiplication by the control volume gives the correct source term. The transfer operation should send to the variables at state “none”.

Use External Soot_Mass_Fraction Source**Syntax**

Use External Soot_Mass_Fraction Source

Summary

Add external soot source term from a transfer

Description

Add source terms to the soot mass fraction equation from a nodal field transferred to this region, e.g. from a Fuego particle region. The field `soot_mass_fraction_source` will be added to the RHS of the soot mass fraction equation; this fields should have units of rate-of-change of mass per volume, so that multiplication by the control volume gives the correct source term. The transfer operation should send to the variables at state “none”.

Use External Species Source**Syntax**

Use External Species Source

Summary

Add external species source term from a transfer

Description

Add source terms to the species equations from a nodal field transferred to this region, e.g. from a Fuego particle region. The vector field `species_source` will be added to the RHS of the species equation; this fields should have units of rate-of-change of mass of species *i* per volume, so that multiplication by the control volume gives the correct source term. The transfer operation should send to the variables at state “none”.

Use Lumped Velocity Density Interpolation**Syntax**

Use Lumped Velocity Density Interpolation

Summary

Interpolate the density-velocity product

Description

By default the continuity equation interpolates velocity and density separately to sub-control surfaces. This option interpolates the product of density times velocity instead.

Use Radiation Source From External Region**Syntax**

Use Radiation Source From External Region [Using Classic Linearization]

Summary

Add in a source term from participating-media radiation which comes from another region through a transfer.

The USING CLASSIC LINEARIZATION optional argument is no longer used or needed, and will be removed in a future release.

Use Shifted Density Iteration**Syntax**

Use Shifted Density Iteration

Summary

Use a lagged density in the momentum solve but an updated density in the velocity projection

Description

Use a lagged density for momentum solve relative to the velocity projection similar to <https://doi.org/10.1016/j.jcp.2012.01.027>

This is a beta feature.

Use Skew Symmetric Central Operator

Syntax

Use Skew Symmetric Central Operator [For Equation *{conserved_enthalpy | continuity | edc_product | enthalpy | mixture_fraction | nuclei | progress_variable | scalar_variance | second_mixture_fraction | solid_volume_fraction | soot | species | temperature | turbulent_dissipation | turbulent_frequency | turbulent_helmholtz_function | turbulent_kinetic_energy | turbulent_v2 | volume_of_fluid | x_momentum | x_solid_momentum | y_momentum | y_solid_momentum | z_momentum | z_solid_momentum}*]

Summary

The blended central operator will be skew symmetric, default is **false**.

Description

The convection operator is always blended with pure central (see hybrid factor description). For the CVFEM methodology, there is a balance between stability and accuracy. Dotting the momentum equation with velocity and summing yields the kinetic energy equation. If the convection operator is skew symmetric, then this dot product leaves something that is perfectly zero. This means that there can be no generation of kinetic energy and simulations can remain stable.

The full CVFEM stencil (27-pt on a hex mesh) is not skew symmetric. Therefore, in cases where one uses pure central (by specifying a hybrid factor of unity) there can be issues - especially on coarse meshes.

Buoyancy Model Specification

Scope

Solution Options

Summary

Specify buoyancy modeling options.

begin Buoyancy Model Specification *BuoyModelName*

Buoyancy Model *{=}* *{boussinesq | buoyant | concentration |*
↪differential | no_buoyancy}

Buoyancy Reference *VarName* *{=}* *Value*

Buoyancy Reference Mass_Fraction *Species* *{=}* *y0*

Buoyancy Reference Mole_Fraction *Species* *{=}* *x0*

Buoyancy Source Term *{=}* *{consistent | default | lumped}*

end Buoyancy Model Specification *BuoyModelName*

Line Commands

Buoyancy Model

Syntax

Buoyancy Model *{=}* *{boussinesq | buoyant | concentration | differential | no_buoyancy}*

Summary

Specification of Buoyancy model to be used in momentum equations.

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>Buoyancy-Model</i>	{boussinesq buoyant concentration differential no_buoyancy}	NO_BUOYANCY

Buoyancy Reference

Syntax

Buoyancy Reference *VarName* *{=}* *Value*

Summary

Buoyancy reference value for the given field.

Description

If the selected buoyancy model requires material property calculations, then reference values must be specified for all input variables required by the particular material model. In the typical case where Cantera is used for property evaluation, then reference properties must be specified for “temperature”, “pressure”, and “mass_fraction”. Other material models may have different dependencies.

Parameter	Value	Default
<i>VarName</i>	string	–
<i>{=}</i>	{= are is}	–
<i>Value</i>	real	–

Buoyancy Reference Mass_Fraction

Syntax

Buoyancy Reference Mass_Fraction *Species* *{=}* *y0*

Summary

Buoyancy reference mass fraction for calculating density.

Description

If the selected buoyancy model requires material property calculations, then

reference values must be specified for all input variables required by the particular material model. In the typical case where Cantera is used for property evaluation, then reference properties must be specified for “temperature”, “pressure”, and “mass_fraction”. Other material models may have different dependencies.

Parameter	Value	Default
<i>Species</i>	string	–
<i>{=}</i>	{= are is}	–
<i>y0</i>	real	–

Buoyancy Reference Mole_Fraction

Syntax

Buoyancy Reference Mole_Fraction *Species* *{=}* *x0*

Summary

Buoyancy reference mole fraction for calculating density.

Description

If the selected buoyancy model requires material property calculations, then reference values must be specified for all input variables required by the particular material model. In the typical case where Cantera is used for property evaluation, then reference properties must be specified for “temperature”, “pressure”, and “mass_fraction”. Other material models may have different dependencies.

Parameter	Value	Default
<i>Species</i>	string	–
<i>{=}</i>	{= are is}	–
<i>x0</i>	real	–

Buoyancy Source Term

Syntax

Buoyancy Source Term *{=}* *{consistent | default | lumped}*

Summary

Specifies if the buoyancy source term should be lumped, consistent, or the same as the mass matrix.

Description

Using a lumped source term diagonalizes the mass matrix contribution to the linear system, and therefore results in a more diagonally-dominant matrix. This under-integration of the source term may result in more monotonic behavior for certain problems.

Using a different source term and mass matrix lumping may introduce undesirable behavior near high gradient regions. Default behavior is to use the same approach for the mass matrix and buoyancy source term.

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>BuoyancySourceTermType</i>	{ consistent default lumped }	DEFAULT

Edc Model Specification

Scope

Solution Options

Summary

Specify EDC combustion model options.

begin Edc Model Specification *EdcSpecName*

Activate Absorption Model

Activate Co2 Dissociation Model

Activate Hydrogen Dissociation Model

Activate Separate Co Irreversible Oxidation Pathway

Characteristic Length Scale For Absorption Coefficient Determination

→ *{=}* *CharacteristicLength*

Critical Damkohler Number For Suppression *{=}* *dacrit*

Edc Fuel Name *{=}* *Fuel*

Edc Ignition Threshold Temperature *{=}* *IgnTemp*

Edc Ignition Time *{=}* *Igntime*

Edc Minimum Product Fraction *{=}* *Prmin*

Edc Reaction Time Scale *{=}* *Tchem*

Include Edc Laminar Limit Model

Minimum Soot Production Temperature *{=}* *Tsootmin*

Omit Near Wall Combustion

Use Edc Product Transport

Use Explicit Treatment Of Edc Source Terms

Use Pure Oxygen For Oxidizer Mixture

Use Sintef Soot Model

end Edc Model Specification *EdcSpecName*

Line Commands

Activate Absorption Model

Syntax

Activate Absorption Model

Summary

Calculate absorption coefficient for use in conjunction with radiation calculations. This will also ensure that the rte source term includes the convolution over gamma chi. If a rte block is active, you may also elect to compute the rte source term by the use of mean temperatures.

Activate Co2 Dissociation Model

Syntax

Activate Co2 Dissociation Model

Summary

Include effects of CO2 dissociation into CO and O2 at high temperatures

Description

At high temperatures, the equilibrium between CO2, CO, and O2 shifts away from CO2, which can significantly decrease the flame temperature. Activating this model will add this effect to the standard EDC combustion model.

Activate Hydrogen Dissociation Model

Syntax

Activate Hydrogen Dissociation Model

Summary

Include effects of H2 dissociation into H

Description

At temperatures greater than about 2000K, the equilibrium between H2 and H

will yield non-negligible concentrations of H which can significantly decrease flame temperatures. Activating this model will add this effect to the standard EDC combustion model using the correlations of W.W. Erikson, which are derived from the NASA CEA code [242, 243].

Note that the H species must be included in the Cantera input XML file, and neither H nor H₂ should be the “last” species in the list since this species is not independent of the rest (to enforce unity sum) and may be susceptible to more noise than the others. Since temperature and other properties are very sensitive to oscillations in the H and H₂ equilibrium, this noise could be problematic.

Activate Separate Co Irreversible Oxidation Pathway

Syntax

Activate Separate Co Irreversible Oxidation Pathway

Summary

Add CO oxidation pathway as a separate reaction pathway. This should really be used only in the context of a propellant fire in the presence of hydrogen combustion.

Characteristic Length Scale For Absorption Coefficient Determination

Syntax

Characteristic Length Scale For Absorption Coefficient Determination *{=}*
CharacteristicLength

Summary

Override the absorption coefficient normally calculated by fuego with this value.

Parameter	Value	Default
<i>{=}</i>	{= are is }	–
<i>CharacteristicLength</i>	real	–

Critical Damkohler Number For Suppression

Syntax

Critical Damkohler Number For Suppression *{=}* *dacrit*

Summary

Critical Damkohler number at which suppression is activated. A value of 0.0 denotes no suppression.

Parameter	Value	Default
<i>{=}</i>	{= are is }	–
<i>dacrit</i>	real	0.0

Edc Fuel Name

Syntax

Edc Fuel Name {=} *Fuel*

Summary

The name of the EDC fuel species, typically either H2 or a major hydrocarbon

Parameter	Value	Default
{=}	{= are is}	–
<i>Fuel</i>	string	–

Edc Ignition Threshold Temperature**Syntax**

Edc Ignition Threshold Temperature {=} *IgnTemp*

Summary

The temperature below which the ignition model is activated

Description

If the ignition model is requested (through the IGNITE initial condition keyword), then it will be activated if all temperatures in the corresponding initial condition block are below this temperature.

Parameter	Value	Default
{=}	{= are is}	–
<i>IgnTemp</i>	real	1000.0 K

Edc Ignition Time**Syntax**

Edc Ignition Time {=} *IgnTime*

Summary

The time at which the EDC combustion model is activated. No combustion will occur before this time.

Parameter	Value	Default
{=}	{= are is}	–
<i>IgnTime</i>	real	0.0

Edc Minimum Product Fraction**Syntax**

Edc Minimum Product Fraction {=} *Prmin*

Summary

The minimum product fraction, below which the EDC model will be deactivated.

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>Prmin</i>	real	1.0e-6

Edc Reaction Time Scale

Syntax

Edc Reaction Time Scale *{=}* *Tchem*

Summary

Reaction time scale to set extinction

Description

Characteristic time scale of the chemical kinetics. Residence time in the fine structure region will be compared to this to determine if extinction will result.

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>Tchem</i>	real	7.0e-5

Include Edc Laminar Limit Model

Syntax

Include Edc Laminar Limit Model

Summary

Turn on the EDC laminar limit model for low-turbulence situations

Description

Turn on the EDC laminar limit model. This model requires setting three model constants: CtauLam, CgammaLam, and ClamTrans. The model uses a time scale based on a velocity gradient rather than the turb_ke/turb_diss. This appropriate time scale permits the flame to anchor in laminar regions.

Minimum Soot Production Temperature

Syntax

Minimum Soot Production Temperature *{=}* *Tsootmin*

Summary

The minimum temperature for which we allow soot to be produced.

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>Tsootmin</i>	real	300.0 K

Omit Near Wall Combustion

Syntax

Omit Near Wall Combustion

Summary

Disables combustion near the wall nodes by setting the production rate to zero. This prevents the values calculated for the EDC combustion rate to be added to the wall element.

Use Edc Product Transport**Syntax**

Use Edc Product Transport

Summary

Solve a transport equation for the EDC reaction products.

Description

If any of the “product” species (CO₂ or H₂O for a hydrocarbon fuel, or H₂O for hydrogen fuel) are to be injected into the domain either through a boundary condition or initial condition to simulate a diluent stream or ambient concentration, then the EDC model will be unable to differentiate which portion of the product species originated directly from combustion and which did not. This can badly distort the EDC model reaction rate calculation and yield incorrect results.

This option solves a transport equation for EDC products that were generated through combustion rather than evaluating it directly from local mass fractions, eliminating issues with product species injection. (If no product species are being injected, then identical results can be obtained at a lower cost without this option.)

To use this feature, activate this line command and also activate the EDC_Product equation along with the Species equation. Note that no initial or boundary conditions are needed for the EDC_Product equation since it has its own internally-handled special needs. (Any specified initial or boundary conditions will be ignored.)

Also note that a pilot stream will be unable to ignite a flame when using this model. It will be treated as an inert diluent stream, so that the normal ignition model will be required to ignite the flame. This model in its current form should not be used for piloted flames.

Use Explicit Treatment Of Edc Source Terms**Syntax**

Use Explicit Treatment Of Edc Source Terms

Summary

Allow no implicit treatment of EDC combustion source terms

Description

The general form of the EDC combustion model for species k is

$rate * (Y_{fs,k} - Y_k)$ where rate is a function of gammachi and residence times. This option specifies that the full source term be placed on the right hand side of the species transport equation, e.g., `math:rhs+= rate*(Y_{fs,k} - Y_k)` as opposed to: `rhs+= rate * (Y_{fs,k} - Y_k)` and `hs+= rate`. The user will recall that the transport equations are solved in residual form, hence the above rhs form.

Use Pure Oxygen For Oxidizer Mixture

Syntax

Use Pure Oxygen For Oxidizer Mixture

Summary

Use pure oxygen as the oxidizer mixture rather than air

Description

By default, the EDC model assumes a 3.76 N2:O2 ratio for the oxidizer mixture. This option switches to pure O2 as the oxidizer.

Use Sintef Soot Model

Syntax

Use Sintef Soot Model

Summary

Use the SINTEF soot model instead of the standard soot model.

Multiphase Model Specification

Scope

Solution Options

Summary

Specify a number of physical parameters that are to be used for multi-phase flow simulations.

begin Multiphase Model Specification *DefinitionName*

Compute Gas Phase Volume Fraction Using *GasVolFracEnum1*[
→*GasVolFracEnum2*]

Enable Porous Media Model

Multiphase {*compaction_modulus* | *density* | *effective_viscosity_*
→*multiplier* | *maximum_packing_diameter* | *particle_diameter* | *reference_*
→*elastic_modulus* | *solid_velocity* | *soot_density* | *turbulent_schmidt_*
→*number*} {=} *Values*

end Multiphase Model Specification *DefinitionName*

Line Commands

Compute Gas Phase Volume Fraction Using

Syntax

Compute Gas Phase Volume Fraction Using *GasVolFracEnum1*[
GasVolFracEnum2]

Summary

Compute the gas phase volume fraction based on soot and particle volume fraction.

Parameter	Value	Default
<i>GasVolFracEnum</i>	{particle soot}	–

Enable Porous Media Model

Syntax

Enable Porous Media Model

Summary

Invoke porous media (aka clutter model, Darcy flow) model equations; an initial condition for gas volume fraction is expected, however, defaulted to unity.

Multiphase

Syntax

Multiphase {*compaction_modulus* | *density* | *effective_viscosity_multiplier* |
maximum_packing_diameter | *particle_diameter* | *reference_elastic_modulus* |
solid_velocity | *soot_density* | *turbulent_schmidt_number*} {=} *Values*

Summary

Specify parameters for multi-phase simulations.

Description

These values are used in solid and gas phase transport equations

Parameter	Value	Default
<i>Multi-Phase Parameters</i>	{compaction_modulus density effective_viscosity_multiplier maximum_packing_diameter particle_diameter reference_elastic_modulus solid_velocity soot_density turbulent_schmidt_number}	–
<i>{=}</i>	{= are is}	–
<i>Values</i>	real	–

Radiation Transport Equation Model Specification

Scope

Solution Options

Summary

Specify options for radiation model.

begin Radiation Transport Equation Model Specification *RadModelName*

Absorption Coefficient Model *{=}* *{leckner}*

Aluminum_Oxide Emittance Model *{=}* *{brewster | konopka | table}*

Characteristic Length Scale For Absorption Coefficient Determination
→{=} *CharacteristicLength*

Species Name For *{aluminum_oxide | carbon_dioxide | hydrogen_chloride | soot | water}* *{=}* *SpeciesName*

Subgrid Mixing Model *{=}* *{no_subgrid_mixing}*

end Radiation Transport Equation Model Specification *RadModelName*

Line Commands

Absorption Coefficient Model

Syntax

Absorption Coefficient Model *{=}* *{leckner}*

Summary

Specify absorption coefficient model

Description

This option will allow for the specification of an absorption coefficient model. At present, only the Leckner model is supported and is, therefore, the default. Specification of the model requires designation of water and carbon dioxide string names.

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>AbsorpCoeffModel</i>	{leckner}	LECKNER

Aluminum_Oxide Emittance Model

Syntax

Aluminum_Oxide Emittance Model *{=}* *{brewster | konopka | table}*

Summary

Specify alumina emittance model

Description

This option allows for the specification of an absorption coefficient model for alumina. Konopka and Table (user-specified) are also available. For the table model, the user must have T-ABS-MODEL.txt file in the current directory with the following format:

0.000094

4

300.0 0.0821

1000.0 0.1064

2320.0 0.0024

10000.0 0.7435

Where:

0.000094 is DTHTWO (Characteristic diameter of Al₂O₃ Units: [cm]). 4 is the number of points in the linear interpolation absorption model (Absorption as function of temperature). The remaining lines are T, ABS points for the model. Linear interpolation is used between the points. For T less than T₀ (lowest temperature for these points), ABS = ABS(T₀). For T greater than T₀ (highest temperature for these points), ABS = ABS(T₀).

Brewster and Konopka options are equivalent to Table option with:

Konopka:

0.000094

4

300 0.0821

1000 0.10639

2320 0.002384

2320.001 0.04 (at $T = 2320$, this represents a discontinuous change in absorption coefficient)

Brewster:

0.000094

4

300 0.0821

1000 0.10639

2320 0.002384

10000 0.74352 ($T = 10000$ is set high enough that particle temperature is not likely to be this high)

Parameter	Value	Default
<code>{=}</code>	{= are is}	–
<code>AluminaEmittModel</code>	{brewster konopka table}	brewster

Characteristic Length Scale For Absorption Coefficient Determination

Syntax

Characteristic Length Scale For Absorption Coefficient Determination `{=}`

CharacteristicLength

Summary

Override the absorption coefficient normally calculated by fuego with this value.

Parameter	Value	Default
<code>{=}</code>	{= are is}	–
<i>CharacteristicLength</i>	real	–

Species Name For

Syntax

Species Name For `{aluminum_oxide | carbon_dioxide | hydrogen_chloride | soot | water}` `{=}` *SpeciesName*

Summary

Specify string names that map to water, carbon dioxide, hydrogen chloride or aluminum oxide (AKA alumina)

Description

This option will allow for the specification of an absorption coefficient model specification. At present, only the Leckner model is supported for water and carbon dioxide. Hydrogen chloride and aluminum oxide (alumina) provided by tables.

Parameter	Value	De- fault
<i>AbsorpCoeff-Species</i>	{aluminum_oxide carbon_dioxide hydrogen_chloride soot water}	–
<i>{=}</i>	{= are is}	–
<i>SpeciesName</i>	string	–

Subgrid Mixing Model

Syntax

Subgrid Mixing Model *{=}* *{no_subgrid_mixing}*

Summary

Specify subgrid mixing model for RTE source term

Description

This option will allow for the specification of subgrid mixing model. At present, there is no model supported other than to ignore fluctuation effects, which is the default.

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>RTESubgridModel</i>	{no_subgrid_mixing}	no_subgrid_mixing

Time Integration Specification

Scope

Solution Options

Summary

Specify time integration options. Either BDF2 or BDF1.

begin Time Integration Specification *TimeIntSpecName*

Activate Bdf2 [After Step *BDF2StartStep*]

Mass Matrix *{=}* *{consistent | lumped}*

Predictor Algorithm *{=}* *{adams_bashforth | forward_euler | simple}*

Use Second Order Implicit Time Integration [With Blending Coefficient
→{=} *BlendCoeff*]

end Time Integration Specification *TimeIntSpecName*

Line Commands

Activate Bdf2

Syntax

Activate Bdf2 [After Step *BDF2StartStep*]

Summary

Activation of the second-order BDF2 time integrator.

Description

BDF2 is a A-stable, three state time integrator that has been demonstrated to be well performing in the low-Mach application space. When activated, all PDEs with time terms will use BDF2. The user specification for MASS MATRIX is used to determine the usage of the lumped or consistent form.

Use the optional argument to delay the use of BDF2 for some number of time steps. Backward euler will be used for those time steps instead.

Mass Matrix

Syntax

Mass Matrix {=} {*consistent* | *lumped*}

Summary

Specifies if the mass matrix (also known as the capacitance matrix) is to be lumped using row sum rule.

Description

Using a lumped mass matrix diagonalizes the mass matrix contribution to the linear system, and therefore results in a more diagonally-dominant matrix. This under-integration of the mass matrix may result in more monotonic behavior for certain problems.

Parameter	Value	Default
{=}	{ = are is }	–
<i>MassMatrixType</i>	{ consistent lumped }	LUMPED

Predictor Algorithm

Syntax

Predictor Algorithm {=} {*adams_bashforth* | *forward_euler* | *simple*}

Summary

Specification of predictor algorithm

Description

Three predictor algorithms are supported, i.e.,

- Simple Predictor: $\phi^{n+1} = \phi^n$
- Forward Euler: $\phi^{n+1} = \phi^n + \Delta t \dot{\phi}$
- Adams Bashforth: $\phi^{n+1} = \phi^n + \frac{\Delta t}{2}(3\dot{\phi}^n - \dot{\phi}^{n-1})$.

Parameter	Value	Default
<i>{=}</i>	{= are is }	–
<i>PredictorAlgorithm</i>	{adams_bashforth forward_euler simple }	–

Use Second Order Implicit Time Integration

Syntax

Use Second Order Implicit Time Integration [With Blending Coefficient *{=}* *BlendCoeff*]

Summary

Deprecated.

Description

Crank-Nicholson scheme is deprecated, defaults to BDF2.

Turbulence Model Specification

Scope

Solution Options

Summary

Specify turbulence modeling options.

begin Turbulence Model Specification *TurbSpecName*

Activate Deris Flaming Buoyancy Source Term

Activate Deris Nonflaming Buoyancy Source Term

Activate Rodi Buoyancy Source Term

Activate Rodi Density Buoyancy Source Term

Determine Utau Via Nonlinear Law Of The Wall Iteration

Include Molecular Viscosity In K-E Diffusion Coefficient

Limit Turbulent Ke Production To *Value* Times Dissipation

Omit Low Reynolds Turbulence Dissipation Source Term

Omit Near Wall Turbulent Ke Transport Equation

Omit Turbulence Source Terms On Block(S) *BlockList...*

Omit Velocity Divergence In Turbulent Production Term

Time Filter *{=}* *Value*

Turbulence Model *{=}* *{dksgs | dsmag | inagaki_ksgs | ke | ksgs | kw |*
↳lam | lrke | lrkw | lrsst | rng | smag | sst | sstdes | v2f}

Turbulence Model Parameter *{frequency_scaling | kappa | minimum_k |*
↳yplus_crit} *{=}* *Value*

Turbulence Postprocessor *{=}* *{q_criterion | residual_reynolds_stress |*
↳resolved_reynolds_stress}

Under Relax Turbulent_Viscosity By *Urf*

Use Buoyant Vorticity Generation At *Value* Seconds

Use Dynamic Ksgs

Wall Distance Band Size *{=}* *BandSize*

Wall Model *{=}* *{modeled | resolved}*

end Turbulence Model Specification *TurbSpecName*

Line Commands

Activate Deris Flaming Buoyancy Source Term

Syntax

Activate Deris Flaming Buoyancy Source Term

Summary

Add de Ris' flaming buoyancy source term to turbulence transport equations.

Description

This option adds the buoyancy source term of Rodi to the turbulent kinetic energy equation (for all turbulence models) and to some extent, to the dissipation rate equation. Two versions - flaming and non-flaming - are proposed in the paper. The source term of the flaming version is given by $C_{deris}(\rho_F - \rho)g_j k^{0.5}$ where C_{deris} is a user-defined coefficient. The coefficient is set 0.01 for now without sufficient validation efforts.

Reference: JL de Ris, Procedia Engineering 62(2013)13-27

Activate Deris Nonflaming Buoyancy Source Term

Syntax

Activate Deris Nonflaming Buoyancy Source Term

Summary

Add de Ris' non-flaming buoyancy source term to turbulence transport equations.

Description

This option adds the buoyancy source term of Rodi to the turbulent kinetic energy equation (for all turbulence models) and to some extent, to the dissipation rate equation. Non-flaming version is a combination of BVG (Buoyant Vorticity Generation) and Rodi models. The source term is given by $C_{deris}\Delta k^{0.5}(|\nabla\rho \times g| - \nabla\rho \cdot g)$ where C_{deris} is a user-defined coefficient. The coefficient is set 0.01 for now without sufficient validation efforts.

Reference: JL de Ris, Procedia Engineering 62(2013)13-27

Activate Rodi Buoyancy Source Term

Syntax

Activate Rodi Buoyancy Source Term

Summary

Add Rodi's buoyancy source term to turbulence transport equations.

Description

This option adds the buoyancy source term of Rodi to the turbulent kinetic energy equation (for all turbulence models) and in some cases, the dissipation rate equation. The source term is given by, $G_B = \beta \frac{\mu_t}{Pr_t} \frac{\partial T}{\partial x_j} g_j$. For the turbulent kinetic energy, G_B augments the right hand side while for the dissipation rate equation, $C'_{\epsilon 1} C_{\epsilon 4} \frac{1}{T} G_B$ is the augmented right hand side term. The source term is limited to a fraction of the dissipation rate.

This model has not been validated for use in the context of turbulence models other than the standard k - ϵ model.

Activate Rodi Density Buoyancy Source Term

Syntax

Activate Rodi Density Buoyancy Source Term

Summary

Add Rodi's buoyancy source term to turbulence transport equations, modified to be density based

Description

This option adds the buoyancy source term of Rodi to the turbulent kinetic energy equation (for all turbulence models) and in some cases, the dissipation rate equation. The source term is given by, $G_B = \rho_{ref} \frac{\mu_t}{Pr_t} \frac{\partial(1/\rho)}{\partial x_j} g_j$. For the turbulent kinetic energy, G_B augments the right hand side while for the dissipation rate equation, $C'_{\epsilon 1} C_{\epsilon 4} \frac{1}{T} G_B$ is the augmented right hand side term. The source term is limited to a fraction of the dissipation rate.

This model is equivalent to the standard rod model for single component, thermal plumes.

Determine Utau Via Nonlinear Law Of The Wall Iteration**Syntax**

Determine Utau Via Nonlinear Law Of The Wall Iteration

Summary

Use law of the wall nonlinear iteration for utau calculation

Description

Elect to perform a nonlinear iteration of the law-of-the-wall formula to calculate utau, the wall friction velocity at the boundary integration points. This value will be used to calculate the modeled wall shear stress and, when activated, the modeled wall heat flux. Alternatively, the friction velocity will be calculated from the nodal value of turbulent k.e. that may be approximated when a turbulent kinetic energy transport equation is not solved.

Include Molecular Viscosity In K-E Diffusion Coefficient**Syntax**

Include Molecular Viscosity In K-E Diffusion Coefficient

Summary

Augment diffusion coefficient in turbulence equations via molecular viscosity

Description

The standard k-e model does not include the molecular viscosity in the diffusion term. This option adds the molecular viscosity to T_{visc}/σ_i . The line command is also appropriate for the Ksgs and v2-f model.

Limit Turbulent Ke Production**Syntax**

Limit Turbulent Ke Production To *Value* Times Dissipation

Summary

Choose to limit production source terms

Description

This option limits the turbulent ke production to a scale factor of dissipation, $\text{prod} = \min(\text{prod}, \text{limit} * \text{den} * \text{en1})$. In practice, the ratio of production to dissipation is not very high. In some flows, it is useful to specify a value of approximately 1000. The ratio should be checked as part of the analysis to make sure that violation of the physical ratio has not been done. In general, this option is only activated in domain locations where dissipation rate is very small. (Default: 1.0e8)

Parameter	Value	Default
<i>Value</i>	real	—

Omit Low Reynolds Turbulence Dissipation Source Term

Syntax

Omit Low Reynolds Turbulence Dissipation Source Term

Summary

Remove low Reynolds source term from dissipation rate equation.

Description

This option removes the low Reynolds number source term from the dissipation rate equation given by Launder and Sharma. The form is

$$2 * \nu * \mu_t * \left(\frac{\partial^2 u_i}{\partial x_k \partial x_j} \frac{\partial^2 u_i}{\partial x_k \partial x_j} \right).$$

This source term is calculated using a two pass shape function evaluation. The first pass calculates the standard derivatives, $\partial u_i / \partial x_j$. The second pass assumes a piecewise constant interpolation of the standard derivatives from the sub control volume to the nodes. These interpolated values are used in the shape function derivative loop to calculate the second derivatives.

The default is to include the source term while the specification of it is placed as a temporary field, i.e., can be changed as restart.

Omit Near Wall Turbulent Ke Transport Equation

Syntax

Omit Near Wall Turbulent Ke Transport Equation

Summary

Do not construct a transport equation for the near wall turbulent kinetic energy equation.

Description

Rather than solving a transport equation for the near wall turbulent transport equation, assign a Dirichlet condition based on local equilibrium argument

between production and dissipation that yields $K_{wall} = u_\tau^2 / \sqrt{C_\mu}$. This formulation neglects convection and diffusion effects for the near wall equation.

Omit Turbulence Source Terms On Block(S)

Syntax

Omit Turbulence Source Terms On Block(S) *BlockList* . .

Summary

Omit source terms on these sets of I/O block; specified by name.

Description

This line command will result in all turbulence source terms to be zeroed in the respective blocks. We can not yet dial in exact equation sets. Example usage:
OMIT TURBULENCE SOURCE TERMS ON BLOCK(S) block_1 block_2.

Note that the source terms for output will still exist and, in general, be non-zero within the skipped block. However, the contribution to the equation system will be omitted.

Parameter	Value	Default
<i>BlockList</i>	string. . .	–

Omit Velocity Divergence In Turbulent Production Term

Syntax

Omit Velocity Divergence In Turbulent Production Term

Summary

Do not include divergence term in turbulence production

Description

This option removes the divergence term from the turbulent production of kinetic energy. The default is to include this term.

Time Filter

Syntax

Time Filter *[=]* *Value*

Summary

Time filter size for Time Filtered Navier-Stokes model

Description

Turbulent viscosity is normally calculated based on a time scale given by k/ϵ (for k-epsilon models) or T (v2f model). The TFNS model substitutes the minimum of the normal computed value and the user-specified time filter size in the turbulent viscosity calculation. In general, this filter should be no less than twice the physical time step. A non-fatal warning is issued if this condition is violated. (Default: 1.0e32 to essentially deactivate the model)

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>Value</i>	real	1.0e32

Turbulence Model

Syntax

Turbulence Model *{=}* *{dksgs | dsmag | inagaki_ksgs | ke | ksgs | kw | lam | lrke | lrkw | lrsst | rng | smag | sst | sstdes | v2f}*

Summary

Specify type of turbulence model to be used

Description

Fuego provides a broad set of turbulence models which includes both RANS and LES models. For more information regarding specific models, please see the theory documentation. One note of importance is that the SST, SSTDES, and LRSST models require the `min_wall_distance` field, which is a measure of the nearest distance to a wall. Fuego will automatically use Sierra/Krino library to generate this field when using these turbulence models.

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>Turb-Model</i>	{dksgs dsmag inagaki_ksgs ke ksgs kw lam lrke lrkw lrsst rng smag sst sstdes v2f}	LAM

Turbulence Model Parameter

Syntax

Turbulence Model Parameter *{frequency_scaling | kappa | minimum_k | yplus_crit}* *{=}* *Value*

Summary

Specify turbulence model parameters

Parameter	Value	Default
<i>TurbulenceModelParams</i>	{frequency_scaling kappa minimum_k yplus_crit}	–
<i>{=}</i>	{= are is}	–
<i>Value</i>	real	–

Turbulence Postprocessor

Syntax

Turbulence Postprocessor *{=}* *{q_criterion | residual_reynolds_stress | resolved_reynolds_stress}*

Summary

Specify a turbulence postprocessor

Description

A set of post processors for turbulence quantities. Options include RESOLVED_REYNOLDS_STRESS, RESIDUAL_REYNOLDS_STRESS, and Q_CRITERION.

warning{The RESOLVED_REYNOLDS_STRESS and RESIDUAL_REYNOLDS_STRESS options are deprecated and should be replaced with the appropriate post-processor in the averaging block.}

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>TurbPP</i>	{ q_criterion residual_reynolds_stress resolved_reynolds_stress }	RESOLVED_REYNOLDS_STRESS

Under Relax Turbulent_Viscosity By

Syntax

Under Relax Turbulent_Viscosity By *Urf*

Summary

Under relaxation factor for the turbulent viscosity

Parameter	Value	Default
<i>Urf</i>	real	–

Use Buoyant Vorticity Generation

Syntax

Use Buoyant Vorticity Generation At *Value* Seconds

Summary

Use the BVG model of Nicollete and Tieszen

Description

Invoke the buoyant vorticity generation augmentation to the turbulent kinetic energy production term. Allow for a delay in the activation to avoid potential instabilities. The delay will be only in assembling the source term to the appropriate partial differential equation. However, the source term will be computed at all times when the model is activated, and will be limited to a fraction of the dissipation rate.

This model has not been validated for use in the context of turbulence models other than the standard k - ϵ model. The model uses the turbulence model parameters C_{bvg} and $C_{\epsilon 3}$.

Parameter	Value	Default
<i>Value</i>	real	–

Use Dynamic Ksgs

Syntax

Use Dynamic Ksgs

Summary

Compute model coefficients dynamically for KSGS turbulent model

Description

warning{This command is deprecated. Use the DKSGS turbulence model to activate this.}

Wall Distance Band Size

Syntax

Wall Distance Band Size *{=}* *BandSize*

Summary

Band size for the normal distance to the wall calculation. Set to 10x the max face element size if omitted.

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>BandSize</i>	real	–

Wall Model

Syntax

Wall Model *{=}* *{modeled | resolved}*

Summary

Specify wall model to be used

Description

Options are MODELED and RESOLVED; intended for LES usage. Current default depends on which turbulence model is in use. For Smagorinsky LES, the wall value for wall shear stress must be based on a nonlinear iteration for the wall friction velocity, or “DETERMINE UTAU VIA NONLINEAR LAW OF THE WALL ITERATION”.

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>WallModel</i>	{ modeled resolved }	–

Acoustic Transfer Output

Scope

Solution Options

Summary

Specify transfer output of the divergence of the Lighthill tensor to a different mesh for use in acoustic simulations.

begin Acoustic Transfer Output *DefinitionName*

At Step *n {increment | interval} {=} m*

At Time *Dt1 {increment | interval} {=} Dt2*

Force Search In Model Coordinates

Input Mesh Name *{=} MeshName*

Output Mesh Name *{=} MeshName*

Overlap Drop Tolerance *{=} DropTol*

Send Block *From_blocks...* To *To_blocks...*

Source Vector Name *{=} SrcName* [Scaling *{=} ScalingValue*]

Timestep Adjustment Interval *{=} Nsteps*

end Acoustic Transfer Output *DefinitionName*

Line Commands

At Step

Syntax

At Step *n {increment | interval} {=} m*

Summary

Specify an output interval in terms of the internal iteration step count. The first step specifies the step count at the beginning of this interval and the second step specifies the output frequency to be used within this interval.

Parameter	Value	Default
<i>n</i>	integer	—
<i>Option</i>	{increment interval}	—
<i>{=}</i>	{= are is}	—
<i>m</i>	integer	—

At Time

Syntax

At Time *Dt1* *{increment | interval}* *{=}* *Dt2*

Summary

Specify an output interval in terms of the internal simulation time. The first time specifies the time at the beginning of this time interval and the second time specifies the output frequency to be used within this interval.

Parameter	Value	Default
<i>Dt1</i>	real	—
<i>Option</i>	{increment interval}	—
<i>{=}</i>	{= are is}	—
<i>Dt2</i>	real	—

Force Search In Model Coordinates

Syntax

Force Search In Model Coordinates

Summary

By default the geometric search will be repeated in the current coordinates at each time step for problems with mesh motion. Adding this line command will force the search to be done once at the start of the problem in the model coordinates.

Input Mesh Name

Syntax

Input Mesh Name *{=}* *MeshName*

Summary

Specify the input mesh that will be the destination of the transfer.

Parameter	Value	Default
<i>{=}</i>	{= are is}	—
<i>MeshName</i>	string	—

Output Mesh Name

Syntax

Output Mesh Name *{=}* *MeshName*

Summary

Specify where to output the result of the transfer.

Description

The divergence of the Lighthill tensor will be output as divT

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>MeshName</i>	string	–

Overlap Drop Tolerance

Syntax

Overlap Drop Tolerance *{=}* *DropTol*

Summary

Specify an overlap drop tolerance.

Description

Specify a volume fraction to ignore overlaps below. This does not affect conservation, if you set a tolerance of 5% (0.05) and a fluid element overlaps with one acoustic element by 97% and the remaining 3% with other elements, then 100% of the source term would be sent to the single acoustic element. This can reduce the cost and memory use of the acoustic transfer.

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>DropTol</i>	real	–

Send Block

Syntax

Send Block *From_blocks...* To *To_blocks...*

Summary

The acoustic transfer will define one transfer operation per SEND BLOCK line, but can define many from/to blocks per line. If there is a block with mesh motion in the send block list it will update that transfer every time step unless you have forced the search to be done in model coordinates. If you have some moving blocks and some fixed blocks it may be faster to split the moving block into its own transfer as long as this is geometrically reasonable.

```
SEND BLOCK block_3 block_5 block_6 TO block_3 block_5
SEND BLOCK block_7 TO block_1
```

Parameter	Value	Default
<i>From_blocks</i>	string. . .	–
<i>To_blocks</i>	string. . .	–

Source Vector Name

Syntax

Source Vector Name *{=}* *SrcName* [Scaling *{=}* *ScalingValue*]

Summary

Specify the noise source vector name.

Description

The divergence of the Lighthill tensor will be output as divT. This specifies the name of the source vector to transfer. By default this is “div_lighthill_tensor” but if you are using a filter or averaging you should set the name here to the name of the filtered vector.

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>SrcName</i>	string	–

Timestep Adjustment Interval

Syntax

Timestep Adjustment Interval *{=}* *Nsteps*

Summary

Specify the number of steps to ‘look ahead’ and adjust the timestep to ensure that the specified output times or simulation end time will be hit ‘exactly’.

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>Nsteps</i>	integer	–

Point Source

Scope

Solution Options

Summary

Specify a point source.

begin Point Source *DefinitionName*

```

    {contact_angle | edc_product | gas_volume_fraction | mixture_fraction |
    ↪| pressure | progress_variable | scalar_variance | second_mixture_
    ↪fraction | solid_volume_fraction | soot_mass_fraction | soot_nuclei_
    ↪mass_fraction | temperature | turbulent_dissipation | turbulent_
    ↪frequency | turbulent_helmholtz_function | turbulent_kinetic_energy |
    ↪| turbulent_v2 | volume_of_fluid | x_solid_velocity | x_velocity | y_
    ↪solid_velocity | y_velocity | z_solid_velocity | z_velocity} {=} Value

```

```

    Location {=} {node nearest | sphere at} Location1 Location2 Location3 |
    ↪[ Radius {=} RadiusValue ]

```

Mdot {=} Mdot

Mass_Fraction Species {=} Mass fraction

Progress_Variable ProgressVariableName {=} Value

end Point Source DefinitionName

Line Commands

Primitivevariable

Syntax

```

Primitivevariable {contact_angle | edc_product | gas_volume_fraction |
mixture_fraction | pressure | progress_variable | scalar_variance |
second_mixture_fraction | solid_volume_fraction | soot_mass_fraction |
soot_nuclei_mass_fraction | temperature | turbulent_dissipation |
turbulent_frequency | turbulent_helmholtz_function | turbulent_kinetic_energy |
turbulent_v2 | volume_of_fluid | x_solid_velocity | x_velocity | y_solid_velocity |
y_velocity | z_solid_velocity | z_velocity} {=} Value

```

Summary

Assign inflow condition to the specified variable. Specified value can be a constant or a string function of time or global variables. Omitted properties will default to the reference value, and if no reference value is present will default to 0.

Parameter	Value	Default
<i>Primitive-Variable</i>	{contact_angle edc_product gas_volume_fraction mixture_fraction pressure progress_variable scalar_variance second_mixture_fraction solid_volume_fraction soot_mass_fraction soot_nuclei_mass_fraction temperature turbulent_dissipation turbulent_frequency turbulent_helmholtz_function turbulent_kinetic_energy turbulent_v2 volume_of_fluid x_solid_velocity x_velocity y_solid_velocity y_velocity z_solid_velocity z_velocity}	–
<i>{=}</i>	{= are is}	–
<i>Value</i>	“string”	–

Location

Syntax

Location *{=}* {*node nearest* | *sphere at*} *Location1 Location2 Location3* [
Radius *{=}* *RadiusValue*]

Summary

Specify the location of the point source. The source will be applied at the node closest to the specified location. If mesh motion is active, the node will be selected only once at the initial mesh coordinates.

The source location can either be the node nearest to a point, or a sphere where the source is applied evenly on all enclosed nodes. If the sphere does not intersect any nodes, the closest node is used.

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>LocationType</i>	{node nearest sphere at}	–
<i>Location</i>	real1 real2 real3	–

Mdot

Syntax

Mdot *{=}* *Mdot*

Summary

Specify the mass flow rate of the point source. Can be a constant or string function of time and global variables. This should be in mass/time units (e.g. kg/s). Positive (inflow) and negative (outflow) values are both allowed.

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>Mdot</i>	“string”	–

Mass_Fraction

Syntax

Mass_Fraction *Species {=} Mass fraction*

Summary

Set the source mass fraction. Specified value can be a constant or a string function of time or global variables. Omitted mass fractions will default to 0.

Parameter	Value	Default
<i>Species</i>	string	–
<i>{=}</i>	{= are is}	–
<i>Mass fraction</i>	“string”	–

Progress_Variable

Syntax

Progress_Variable *ProgressVariableName {=} Value*

Summary

Set the source progress variable. Specified value can be a constant or a string function of time or global variables.

Parameter	Value	Default
<i>ProgressVariableName</i>	string	–
<i>{=}</i>	{= are is}	–
<i>Value</i>	“string”	–

Mesh Motion Specification

Scope

Solution Options

Summary

Specify mesh motion parameters.

begin Mesh Motion Specification *DefinitionName*

Activate Bdf2 For Mesh Velocity

```
Mesh Motion Model On BlockName {=} {fixed_const_velocity | none |  
→rigid_body | rotation | specified | transferred} [ DisplacementArgs... ]  
→]
```

```
begin Rigid Body Motion Model DefinitionName  
end
```

end Mesh Motion Specification *DefinitionName*

Line Commands

Activate Bdf2 For Mesh Velocity

Syntax

Activate Bdf2 For Mesh Velocity

Summary

Activation of the second-order BDF2 time integrator for mesh velocity.

Description

BDF2 is a A-stable, three state time integrator that has been demonstrated to be well performing in the low-Mach application space. When activated, the mesh velocity calculated from displacements will use BDF2. This is enabled automatically if the overall time integration is set to use BDF2.

Mesh Motion Model On

Syntax

Mesh Motion Model On *BlockName* {=} {*fixed_const_velocity* | *none* | *rigid_body* | *rotation* | *specified* | *transferred*} [*DisplacementArgs...*]

Summary

Specify the mesh motion model to use.

Description

Select the mesh motion model (SPECIFIED, FIXED_CONST_VELOCITY, TRANSFERRED, ROTATION, or NONE) along with any additional model arguments.

The ‘SPECIFIED’ model takes either a set of specified displacements that are string functions of t, x, y, and z, or a specified set of accelerations that are functions of t and any global variables. In the latter mode, an initial velocity, v0, can also be specified (default: 0).

The ‘TRANSFERRED’ and ‘NONE’ options do not take any arguments.

The ‘FIXED_CONST_VELOCITY’ lets you apply motion to a block without actually moving it in the model. This is typically applied to shells, for example to move the top fluid in a lid-driven cavity flow without actually moving the mesh, or to apply motion on a quarter of a circle without actually moving it.

The ‘ROTATION’ model takes a time function for the angle (theta) in rotations (radius/2*pi or degrees/360), and an origin and rotation axis (only for 3D). For example, a rotation of 1 rpm would use ‘theta=t/60’. The rotation uses the right hand rule about the specified axis and origin. Rotation can also be specified by

providing an angular acceleration term that can be a function of time (t) or global variables. An optional initial angular velocity (omega0) can also be specified (default: 0). Both the angular acceleration and angular velocity are specified in revolutions, so they are multiplied internally by 2*pi to convert to radians.

Any blocks that don't have a mesh motion model specified get no displacement. Some example syntax is shown below. You can use the "all_blocks" keyword to specify all blocks, but it cannot be mixed with per-block specifications.

```
MESH MOTION MODEL ON block_1    = SPECIFIED D = 0.1*t y*z
    ↪sin(t)
MESH MOTION MODEL ON block_1    = SPECIFIED v0 = 0.1 0 0 a=0
    ↪0.1 0
MESH MOTION MODEL ON all_blocks = TRANSFERRED
MESH MOTION MODEL ON block_3    = ROTATION  origin=0 0 0 \
    axis=1 0 0 theta=t/60
MESH MOTION MODEL ON block_3    = ROTATION  origin=0 0 0 \
    axis=1 0 0 omega0=1/60 alpha=0
MESH MOTION MODEL ON block_4    = NONE
MESH MOTION MODEL ON block_6    = FIXED_CONST_VELOCITY \
    Velocity=0.1 0*t "0 + 4*x*y"
```

The mesh motion model also generates global variables of its internal state (position, velocity, and acceleration) which can be used in the functional forms. These global variables are output in units of revolutions for consistency with the input units. For example, to smoothly accelerate up to a constant angular velocity of 10 rev/s, you could use:

```
MESH MOTION MODEL ON block_3 = ROTATION  origin=0 0 0 \
    axis=1 0 0 alpha=10-block_3_omega
```

Parameter	Value	De- fault
<i>BlockName</i>	string	–
<i>{=}</i>	{= are is }	–
<i>MeshMotion- Model</i>	{fixed_const_velocity none rigid_body rotation specified transferred }	–

Rigid Body Motion Model

Scope

Mesh Motion Specification

Summary

Specify parameters for 6-DOF rigid body motion.

begin Rigid Body Motion Model *DefinitionName*

Acceleration *{=}* *Acceleration1 Acceleration2 Acceleration3*

Blocks *{=}* *BlockNames...*

Buoyant Object Mass *{=}* *ObjectMass...*

Buoyant Object Pitch Moment *{=}* *Moment*

Buoyant Object Roll Moment *{=}* *Moment*

Buoyant Object Surfaces *{=}* *ObjSurfaces...*

Buoyant Object Yaw Moment *{=}* *Moment*

Initial Velocity *{=}* *InitialVelocity1 InitialVelocity2 InitialVelocity3*

Origin *{=}* *OriginCoords1 OriginCoords2 OriginCoords3*

Pitch Acceleration *{=}* *PitchAcceleration*

Pitch Initial Velocity *{=}* *PitchInitialVelocity*

Roll Acceleration *{=}* *RollAcceleration*

Roll Axis *{=}* *RollAxis1 RollAxis2 RollAxis3*

Roll Initial Velocity *{=}* *RollInitialVelocity*

Under Relaxation *{=}* *UnderRelaxation*

Yaw Acceleration *{=}* *YawAcceleration*

Yaw Axis *{=}* *YawAxis1 YawAxis2 YawAxis3*

Yaw Initial Velocity *{=}* *YawInitialVelocity*

end Rigid Body Motion Model *DefinitionName*

Line Commands

Acceleration

Syntax

Acceleration *{=}* *Acceleration1 Acceleration2 Acceleration3*

Summary

Specify the translational acceleration

Description

Specify the translational acceleration using three quoted string functions. Functions can include time and any global variables as valid variables.

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>Acceleration</i>	“string”1 “string”2 “string”3	–

Blocks

Syntax

Blocks *{=}* *BlockNames. . .*

Summary

Specify which blocks to apply this model to.

Description

Give a list of blocks to apply the model to. The all_blocks command is permitted here too.

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>BlockNames</i>	string. . .	–

Buoyant Object Mass

Syntax

Buoyant Object Mass *{=}* *ObjectMass. . .*

Summary

Specify the buoyant object mass

Description

Specify the buoyant object mass. Optionally, you can specify three values for

mass to act on the three translation directions. This is primarily useful for suppressing motion in a given direction by setting that mass to a very large number. For example, to only allow motion of a 50 kg object in the z-direction you could set the mass to:

```
BUOYANT OBJECT MASS = 1e30 1e30 50
```

Parameter	Value	Default
<i>{=}</i>	{ = are is }	—
<i>ObjectMass</i>	real. . .	—

Buoyant Object Pitch Moment

Syntax

Buoyant Object Pitch Moment *{=}* *Moment*

Summary

Specify the buoyant object pitch moment

Description

Specify the buoyant object moment of inertia about pitch axis. Set to a very large number to disable pitch.

Parameter	Value	Default
<i>{=}</i>	{ = are is }	—
<i>Moment</i>	real	—

Buoyant Object Roll Moment

Syntax

Buoyant Object Roll Moment *{=}* *Moment*

Summary

Specify the buoyant object roll moment

Description

Specify the buoyant object moment of inertia about roll axis. Set to a very large number to disable roll.

Parameter	Value	Default
<i>{=}</i>	{ = are is }	—
<i>Moment</i>	real	—

Buoyant Object Surfaces

Syntax

Buoyant Object Surfaces *{=}* *ObjSurfaces. . .*

Summary

Specify surfaces of a buoyant object.

Description

Specify surfaces of a buoyant object.

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>ObjSurfaces</i>	string. . .	–

Buoyant Object Yaw Moment

Syntax

Buoyant Object Yaw Moment *{=}* *Moment*

Summary

Specify the buoyant object yaw moment

Description

Specify the buoyant object moment of inertia about yaw axis. Set to a very large number to disable yaw.

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>Moment</i>	real	–

Initial Velocity

Syntax

Initial Velocity *{=}* *InitialVelocity1* *InitialVelocity2* *InitialVelocity3*

Summary

Specify the translational initial velocity

Description

Specify the translational initial velocity using three quoted string functions. Functions can include time and any global variables as valid variables.

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>InitialVelocity</i>	“string”1 “string”2 “string”3	–

Origin

Syntax

Origin *{=}* *OriginCoords1* *OriginCoords2* *OriginCoords3*

Summary

Specify the origin for the motion.

Description

The origin coordinates are those around which rotation occurs. This would typically be the center of mass.

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>OriginCoords</i>	real1 real2 real3	–

Pitch Acceleration

Syntax

Pitch Acceleration *{=}* *PitchAcceleration*

Summary

Specify the pitch acceleration

Description

Specify the pitch acceleration using a quoted string function. Functions can include time and any global variables as valid variables. Units are revolutions, not radians.

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>PitchAcceleration</i>	“string”	–

Pitch Initial Velocity

Syntax

Pitch Initial Velocity *{=}* *PitchInitialVelocity*

Summary

Specify the pitch initial velocity

Description

Specify the pitch initial velocity using a quoted string function. Functions can include time and any global variables as valid variables. Units are revolutions, not radians.

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>PitchInitialVelocity</i>	“string”	–

Roll Acceleration

Syntax

Roll Acceleration {=} *RollAcceleration*

Summary

Specify the roll acceleration

Description

Specify the roll acceleration using a quoted string function. Functions can include time and any global variables as valid variables. Units are revolutions, not radians.

Parameter	Value	Default
{=}	{ = are is }	–
<i>RollAcceleration</i>	“string”	–

Roll Axis**Syntax**

Roll Axis {=} *RollAxis1 RollAxis2 RollAxis3*

Summary

Specify the roll axis for rotation.

Description

Rotation is specified using pitch, roll, and yaw.

Roll is rotation about the roll axis. Yaw is rotation about the yaw axis. The pitch axis is defined by the cross product of the roll and yaw axes.

Rotation is applied as roll, then pitch, then yaw. The axes about which these translations happen are fixed.

The roll and yaw axes must be orthogonal. These do not need to be unit vectors, they will be normalized internally.

Parameter	Value	Default
{=}	{ = are is }	–
<i>RollAxis</i>	real1 real2 real3	–

Roll Initial Velocity**Syntax**

Roll Initial Velocity {=} *RollInitialVelocity*

Summary

Specify the roll initial velocity

Description

Specify the roll initial velocity using a quoted string function. Functions can

include time and any global variables as valid variables. Units are revolutions, not radians.

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>RollInitialVelocity</i>	“string”	–

Under Relaxation

Syntax

Under Relaxation *{=}* *UnderRelaxation*

Summary

Specify under-relaxation for the equations of motion

Description

Specify under-relaxation for the equations of motion

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>UnderRelaxation</i>	real	1

Yaw Acceleration

Syntax

Yaw Acceleration *{=}* *YawAcceleration*

Summary

Specify the yaw acceleration

Description

Specify the yaw acceleration using a quoted string function. Functions can include time and any global variables as valid variables. Units are revolutions, not radians.

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>YawAcceleration</i>	“string”	–

Yaw Axis

Syntax

Yaw Axis *{=}* *YawAxis1* *YawAxis2* *YawAxis3*

Summary

Specify the yaw axis for rotation.

Description

Rotation is specified using pitch, roll, and yaw.

Roll is rotation about the roll axis. Yaw is rotation about the yaw axis. The pitch axis is defined by the cross product of the roll and yaw axes.

Rotation is applied as roll, then pitch, then yaw. The axes about which these translations happen are fixed.

The roll and yaw axes must be orthogonal. These do not need to be unit vectors, they will be normalized internally.

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>YawAxis</i>	real1 real2 real3	–

Yaw Initial Velocity

Syntax

Yaw Initial Velocity *{=}* *YawInitialVelocity*

Summary

Specify the yaw initial velocity

Description

Specify the yaw initial velocity using a quoted string function. Functions can include time and any global variables as valid variables. Units are revolutions, not radians.

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>YawInitialVelocity</i>	“string”	–

Vof Model Specification

Scope

Solution Options

Summary

Specify a number of physical parameters that are to be used for multi-phase VOF flow simulations.

begin Vof Model Specification *DefinitionName*

Interface Compression Factor *{=}* *VOFInterfaceCompressionFactor*

Interface Compression Model *{=}* {adaptive | constant | none}

```

Interface Curvature Algorithm {=} {diffusive | level_set | none}

Interface Location Tolerance {=} InterfaceTol

Interface Sharpening Model {=} {non_conserving | none | volume_
↳conerving}

Num Smoother Iterations {=} NumSmootherIters

Phase Change Model {=} {constant | langmuir | none | thermal}↳
↳[ModelArgs]...

Post Process Interface Normal

Smoother Fourier Number {=} SmootherFo

end Vof Model Specification DefinitionName

```

Line Commands

Interface Compression Factor

Syntax

Interface Compression Factor {=} VOFInterfaceCompressionFactor

Summary

Specify parameters for interface compression model for VOF.

Description

Select the VOF interface compression factor (only used in the Constant model)

Parameter	Value	Default
{=}	{= are is}	–
VOFInterfaceCompressionFactor	real	–

Interface Compression Model

Syntax

Interface Compression Model {=} {adaptive | constant | none}

Summary

Specify parameters for interface compression model for VOF.

Description

Select the VOF interface compression model (Constant, Adaptive, or None)

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>VOFInterfaceCompressionModel</i>	{ adaptive constant none }	–

Interface Curvature Algorithm

Syntax

Interface Curvature Algorithm *{=}* *{diffusive | level_set | none}*

Summary

Specify the curvature model to use.

Description

Select the VOF curvature model (DIFFUSIVE, LEVEL_SET, or NONE)

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>VOFSmoothingAlgorithm</i>	{ diffusive level_set none }	–

Interface Location Tolerance

Syntax

Interface Location Tolerance *{=}* *InterfaceTol*

Summary

Specify parameters for interface location for VOF.

Description

Select the tolerance used to identify the interface zone

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>InterfaceTol</i>	real	–

Interface Sharpening Model

Syntax

Interface Sharpening Model *{=}* *{non_conserving | none | volume_conserving}*

Summary

Specify parameters for interface sharpening model for VOF.

Description

Select the VOF interface sharpening model (Volume_Conserving or None)

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>VOFInterfaceSharpeningModel</i>	{ non_conserving none volume_conserving }	–

Num Smoother Iterations

Syntax

Num Smoother Iterations *{=}* *NumSmootherIters*

Summary

Specify parameters for interface smoothing for VOF.

Description

Select the number of smoother iterations for the interface normal calculation

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>NumSmootherIters</i>	integer	–

Phase Change Model

Syntax

Phase Change Model *{=}* { *constant* | *langmuir* | *none* | *thermal* } *[ModelArgs]*. . .

Summary

Specify the phase change model to use.

Description

Select the VOF phase change model (CONSTANT, THERMAL, or NONE)

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>VOFPhaseChangeModel</i>	{ constant langmuir none thermal }	–
<i>ModelArgs</i>	[string]. . .	–

Post Process Interface Normal

Syntax

Post Process Interface Normal

Summary

Post-process the interface normal vector

Smoother Fourier Number

Syntax

Smoother Fourier Number *{=}* *SmootherFo*

Summary

Specify parameters for interface smoothing for VOF.

Description

Select the Fourier number for the interface diffusive smoother

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>SmootherFo</i>	real	–

Postprocess

Scope

Fuego Region

Summary

Defines a custom post-processor block to be run at the end of the time step.

Description

The block type defines the post-processor operation performed, as in:

Begin postprocess Integral

or

Begin postprocess Average

The valid types are described below.

Integral

Perform an integral of the specified scalar function (f) on either the specified volume (blocks) or surface (sidesets) and save the result as a global variable. The function f can depend on time and space (t, x, y, z) and any nodal variable.

On volumes this is:

$$F = \int_V f dV$$

while on surfaces it is:

$$F = \int_A f dA$$

Average

Perform a volume or area weighted average of the specified scalar function (f) on either the specified volume (blocks) or surface (sidesets) and save the result

as a global variable. The function f can depend on time and space (t,x,y,z) and any nodal variable.

On volumes this is:

$$F = \frac{\int_V f dV}{\int_V dV}$$

while on surfaces it is:

$$F = \frac{\int_A f dA}{\int_A dA}$$

Sum

Find the nodal summation of the specified scalar function (f) on either the specified volume (blocks) or surface (sidesets) and save the result as a global variable. The function f can depend on time and space (t,x,y,z) and any nodal variable.

Min

Find the nodal minimum of the specified scalar function (f) on either the specified volume (blocks) or surface (sidesets) and save the result as a global variable. The function f can depend on time and space (t,x,y,z) and any nodal variable.

Max

Find the nodal maximum of the specified scalar function (f) on either the specified volume (blocks) or surface (sidesets) and save the result as a global variable. The function f can depend on time and space (t,x,y,z) and any nodal variable.

L2_Norm

Find the nodal L2-norm of the specified scalar function (f) on either the specified volume (blocks) or surface (sidesets) and save the result as a global variable. The function f can depend on time and space (t,x,y,z) and any nodal variable.

On volumes this is:

$$F = \sqrt{\int_V f^2 dV}$$

while on surfaces it is:

$$F = \sqrt{\int_A f^2 dA}$$

Global

Evaluate a global function (f) and save the result as a global variable. The function f can depend on time and any other global variable. You should not specify any location entries for a global post-processor.

```
Begin postprocess Global
  Output name = maxTempC
  Function = "maxTemp - 273.15"
End
```

Nodal_Field

Evaluate the specified scalar function (f) on either the specified volume (blocks) or surface (sidesets) and save the result in a nodal field. The function f can depend on time and space (t,x,y,z) and any nodal variable.

```
Begin postprocess Nodal_Field
  Output name = nuCalc
  Location = all_blocks
  Function = "viscosity/density"
End
```

Integrated_Flux

Evaluate the integrated flux of a specified vector function (requires 2 or 3 components for f depending on problem dimension). This must be performed on surfaces, not on volumes. The result is saved in a global variable. The functions can depend on time and space (t,x,y,z) and any nodal variable.

$$F = \int_A \vec{f} \cdot \vec{dA}$$

```
Begin postprocess Integrated_Flux
  Output name = mass_flux
  Location = surface_1
  Function = "density*x_velocity" \$
            "density*y_velocity" \$
            "density*z_velocity"
End
```

Point

Evaluate the specified function at a specific point in space and save the result as a global variable. The location to query should be specified as 2 or 3 coordinates in the Location command. If the specified coordinates lie outside all the mesh elements an error will be thrown. The functions can depend on time and space (t,x,y,z) and any nodal variable.

```

Begin postprocess Point
  Output name = TC1
  Location = 1.2 2.2 0.1
  Function = "temperature"
End

```

begin Postprocess

Evaluation Time *{=}* *{after_fluid | after_pmr}*

Function *{=}* *FunctionStr...*

Location *{=}* *MeshEntites...*

Output Name *{=}* *OutputName*

end Postprocess

Line Commands

Evaluation Time

Syntax

Evaluation Time *{=}* *{after_fluid | after_pmr}*

Summary

Define whether to execute the utility after the fluid solve or after the PMR solve. Default behavior is to run after the PMR solve. If the post-processed quantity uses PMR fields, one may want to run them with the same values as were used in the fluid solve (After_Fluid) or with the updated values returned from the PMR solve (After_PMR).

Parameter	Value	Default
<i>{=}</i>	<i>{= are is}</i>	–
<i>PPEvalTime</i>	<i>{after_fluid after_pmr}</i>	After_PMR

Function

Syntax

Function *{=}* *FunctionStr...*

Summary

Provide a string function to evaluate in the post-processor.

Description

A quoted function string for the post-processor to evaluate. For the FLUX post-processor you must provide multiple quoted entries - one per spatial dimension. For all other types you may only provide a single function string.

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>FunctionStr</i>	“string”...	–

Location

Syntax

Location *{=}* *MeshEntites...*

Summary

Mesh locations to evaluate the post-processor at.

Description

A list of block or sideset names to evaluate the post-processor at. For multiple blocks you can either include them all in one line or add separate lines. The aliases “all_blocks” and “all_surfaces” can also be used. You cannot mix blocks and sidesets in a single post-processor block.

```
Begin postprocess nodal_field
  Output name = nuCalc
  Location = all_blocks
  Function = "viscosity/density"
End
```

```
Begin postprocess nodal_field
  Output name = nuCalc
  Location = block_1 block_2 block_3
  Function = "viscosity/density"
End
```

```
Begin postprocess nodal_field
  Output name = nuCalc
  Location = block_1
  Location = block_2
  Location = block_3
  Function = "viscosity/density"
End
```

When using the “POINT” operation the location command should be the coordinates to evaluate the function at.

```

Begin postprocess point
  Output name = TC1
  Location = 1.2 2.2 0.1
  Function = "temperature"
End

```

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>MeshEntites</i>	string...	–

Output Name

Syntax

Output Name *{=}* *OutputName*

Summary

Define the name for the output variable (global or nodal depending on the type).

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>OutputName</i>	string	–

7.8 Particle Region

Particle Region

Scope

Fuego Procedure

Summary

Contains the commands needed to build and solve a set of Lagrangian particles

begin Particle Region *Regionname*

Compute Continuity Source

Compute Energy Source

Compute Mixture_Fraction Source

Compute Momentum Source

Compute Rte Particle Source

```

Compute Soot_Mass_Fraction Source [ With Soot Split {=} SootSplit ]

Compute Source Lhs Contribution

Compute Species Source

Disable Particle Transfer Checks

Fluid Region Name {=} FluidRegionName

Fluid Velocity Fluctuation Model {=} {continuous_random_walk | discrete_random_walk | none}

Include Thermophoretic Force

Interpolate Fluid Variable FluidVarName For Output As ParticleVarName
↳ [ With Vector Size VecSize ]

Lagrangian Particles

Maximum Particle Subcycles {=} SubCycles

Minimum Particle Subcycles {=} SubCycles

Number Of Fluid Species {=} NumSpecies

Particle Maximum Search Grid Dimension {=} Size

Particle Random Number Generator Seed {=} Size

Particle Rebalance Imbalance Threshold {=} Threshold

Particle Rebalance Step Frequency {=} Nsteps

Particle Reservoir Size {=} Size

Perform Initial Particle Rebalance

Use Finite Element Model ModelName [ Model Coordinates Are Nodal_
↳ variable_name ]

begin Create Initial Particle Distribution DistributionName
end

```

```

begin Create Particles From File BlockName
end

begin Fluid Variable Specification SpecificationName
end

begin Heartbeat Label
end

begin Heartbeat Output Label
end

begin History Output Label
end

begin Insert Particle Particlename
end

begin Particle Average AverageName
end

begin Particle Breakup Model ModelName
end

begin Particle Definition Particlename
end

begin Particle Filled Shape FilledShapeName
end

begin Particle Filled Tree FilledTreeName
end

begin Particle Inflow Boundary Condition On Surface Surfacename
end

begin Particle Interaction Model ModelName
end

begin Particle Interface InterfaceName
end

begin Particle Material MaterialName
end

```

```
begin Particle Open Boundary Condition On Surface Surfacename  
end
```

```
begin Particle Postprocess Operation  
end
```

```
begin Particle Postprocessing Name  
end
```

```
begin Particle Spray SprayName  
end
```

```
begin Particle Stats Output Name  
end
```

```
begin Particle Tabular Output Name  
end
```

```
begin Particle Wall Boundary Condition On Surface Surfacename  
end
```

```
begin Restart Data Label  
end
```

```
begin Results Output Label  
end
```

```
end Particle Region Regionname
```

Line Commands

Compute Continuity Source

Syntax

Compute Continuity Source

Summary

Compute continuity source to be transferred to fluid region.

Description

Compute a continuity source term to be used in the fluid continuity equation (pressure). The source term is accumulated in the nodal field called `continuity_source`, which has units of mass change per time per volume. This field should be transferred to the fluid region into the corresponding field.

Compute Energy Source

Syntax

Compute Energy Source

Summary

Compute energy source to be transferred to fluid region.

Description

Compute an energy source term to be used in the fluid energy equation. The source term is accumulated in the nodal field called `energy_source`, which has units of internal energy change per time per volume (the same units as density times dh/dt). This field should be transferred to the fluid region into the corresponding field.

Compute Mixture_Fraction Source**Syntax**

Compute Mixture_Fraction Source

Summary

Compute mixture fraction sources to be transferred to fluid region.

Description

Compute a mixture fraction source terms to be used in the fluid species. The source terms are accumulated in two nodal fields called `mixture_fraction_source` and `second_mixture_fraction_source`, which have units of change in mass per time per volume. These fields should be transferred to the fluid region into the corresponding fields. The specific components of the mixture fraction vector for which a source is added depend on the specific Particle Type being used. For example, for the `WILDFIRE_PARTICLE` type the first variable contains the source for water vapor and the second for the volatile stream.

Compute Momentum Source**Syntax**

Compute Momentum Source

Summary

Compute momentum source to be transferred to fluid region.

Description

Compute a momentum source term to be used in the fluid momentum equation. The source term is accumulated in the nodal field called `momentum_source`, which has units of momentum change per time per volume (the same units as density times du/dt). This field should be transferred to the fluid region into the corresponding field.

Compute Rte Particle Source**Syntax**

Compute Rte Particle Source

Summary

Compute radiative source term in particle energy equation.

Description

Compute a radiative source term in the particle energy equation. This option will require specification of the **ABSORPTIVITY** in the particle material block. This option will also require the transfer of scalar flux in the transfer from a PMR region to the Lagrangian region.

Compute Soot_Mass_Fraction Source

Syntax

Compute Soot_Mass_Fraction Source [With Soot Split {=} *SootSplit*]

Summary

Compute soot mass fraction source to be transferred to fluid region.

Description

Compute a soot mass fraction source terms to be used in the fluid soot mass fraction transport equation. The source terms are accumulated in the nodal fields called `soot_mass_fraction_source`, which have units of change in mass per time per volume. These fields should be transferred to the fluid region into the corresponding fields.

An optional parameter to this line command specifies the split of volatile production directed to soot. Default for this parameter is 0.01.

Compute Source Lhs Contribution

Syntax

Compute Source Lhs Contribution

Summary

Compute LHS matrix contribution from particle source terms to fluid equation.

Description

Compute contributions from particle source terms to the diagonal of the LHS matrix of the corresponding fluid equations. This contribution provides added stability in the fluid solve. The LHS contribution is accumulated in nodal fields called `x_momentum_source_lhs`, `y_momentum_source_lhs`, and `z_momentum_source_lhs` for the momentum equations; `energy_source_lhs` for the energy equation; `species_source_lhs` for the species equations (not yet implemented); and `continuity_source_lhs` for the continuity equation (not yet implemented). These fields should be transferred to the fluid region into the corresponding fields.

Compute Species Source

Syntax

Compute Species Source

Summary

Compute species source to be transferred to fluid region.

Description

Compute a species source term to be used in the fluid species. The source term is accumulated in the nodal field called `species_source`, which has units of mass of species *k* change per time per volume. This field should be transferred to the fluid region into the corresponding field.

Disable Particle Transfer Checks

Syntax

Disable Particle Transfer Checks

Summary

Disables transfer checking on the particle region.

Description

Normally the transfers to the particle region are verified against the particle physics, but this command disables these checks. This is an advanced user option - use with care. Transfer errors are treated as warnings and printed to the log file when this option is active.

Fluid Region Name

Syntax

Fluid Region Name `{=}` *FluidRegionName*

Summary

Inform particle region of the name of the fluid region to which it is coupled. This is needed if associations must be made between particle-related and fluid species, e.g. between oxidizers or products of particle reactions that are coupled to fluid species.

Parameter	Value	Default
<code>{=}</code>	{= are is}	—
<i>FluidRegionName</i>	string	—

Fluid Velocity Fluctuation Model

Syntax

Fluid Velocity Fluctuation Model `{=}` *{continuous_random_walk | discrete_random_walk | none}*

Summary

Specify the fluid velocity fluctuation type for this particle block.

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>FluctVelModelType</i>	{ continuous_random_walk discrete_random_walk none }	–

Include Thermophoretic Force

Syntax

Include Thermophoretic Force

Summary

Include thermophoretic force in momentum source terms.

Description

Compute the thermophoretic force on a Lagrangian particle to be included in the momentum source terms of the particle momentum equations. Thermophoretic force is given by $F_t = -3\pi R_p \frac{\mu^2}{\rho T} G$, where R_p is the particle radius, μ is the fluid viscosity, ρ is the fluid density, T is the fluid temperature, and G is the temperature gradient affecting the particle. The `temperature_gradient` field must be transferred from the fluid region to the corresponding particle region field when using this capability.

Interpolate Fluid Variable

Syntax

Interpolate Fluid Variable *FluidVarName* For Output As *ParticleVarName* [With Vector Size *VecSize*]

Summary

Define an interpolation from the fluid to the particles

Description

The `FluidVarName` should be the name of a fluid variable that is on the receiving end of a transfer operation from a Fuego fluid region. The `ParticleVarName` may be chosen by the user; this same variable should then be listed in the region output block to obtain values in the output file.

Parameter	Value	Default
<i>FluidVarName</i>	string	–
<i>ParticleVarName</i>	string	–

Lagrangian Particles

Syntax

Lagrangian Particles

Summary

Allows the introduction of Lagrangian particles into the flow.

Maximum Particle Subcycles

Syntax

Maximum Particle Subcycles *{=} SubCycles*

Summary

Set maximum number of subcycles to be taken for each particle at each timestep.

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>SubCycles</i>	integer	10

Minimum Particle Subcycles

Syntax

Minimum Particle Subcycles *{=} SubCycles*

Summary

Set minimum number of subcycles to be taken for each particle at each timestep.

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>SubCycles</i>	integer	1

Number Of Fluid Species

Syntax

Number Of Fluid Species *{=} NumSpecies*

Summary

Set number of fluid species in the coupled Fuego region (needed for handling mass fractions transferred from Fuego region)

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>NumSpecies</i>	integer	–

Particle Maximum Search Grid Dimension

Syntax

Particle Maximum Search Grid Dimension *{=} Size*

Summary

This feature is deprecated as the auxiliary structured grid has been replaced by a stk coarse search.

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>Size</i>	integer	0

Particle Random Number Generator Seed

Syntax

Particle Random Number Generator Seed *{=}* *Size*

Summary

Specify a random number generator seed for particle region related rand() calls.

Description

Seeds the pseudo-random number generator used by the boost mersenne twister random number generator within the particle region with the unsigned integer value. The default seed is set to 12345 + Rank ID for a multiprocessor run (12345 + 0 for a single processor run). This command line will also influence the sequence of values produced by the random number generator outside the particle region. Currently random numbers are used in particle probability distribution functions, particle spray, particle breakup model, particle surface interactions and particle dynamics.

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>Size</i>	integer	0

Particle Rebalance Imbalance Threshold

Syntax

Particle Rebalance Imbalance Threshold *{=}* *Threshold*

Summary

Set threshold for the ratio of maximum to minimum particles per processor that triggers dynamic particle rebalance.

Description

Set threshold for the ratio of maximum to minimum particles per processor that triggers dynamic particle rebalance. This is the percent imbalance to trigger a rebalance, so an entry of 0.25 will rebalance when the maximum imbalance is 25 percent off from idea. This command works in tandem with the PARTICLE REBALANCE STEP FREQUENCY command, and Fuego will test the imbalance threshold and if necessary perform a rebalance only on the steps indicated by the frequency command. If no step frequency is specified, no rebalance will be performed. If the minimum number of particles on a processor is zero, a value of one is used for the minimum when computing the max/min ratio.

Parameter	Value	Default
<i>{=}</i>	{= are is }	–
<i>Threshold</i>	real	0.5

Particle Rebalance Step Frequency

Syntax

Particle Rebalance Step Frequency *{=}* *Nsteps*

Summary

Set frequency (in timesteps) with which dynamic rebalancing of the mesh decomposition of the particle region will be recomputed, when necessary.

Description

Set frequency (in timesteps) with which the need for dynamic rebalancing of the mesh decomposition of the particle region will be recomputed. Weighting for the decomposition is based on number of particles per element, so that after redistributing the mesh each processor should include approximately the same number of particles. Note that this command works together with the **PARTICLE REBALANCE IMBALANCE THRESHOLD** command. At the specified frequency, Fuego computes the current level of imbalance (the ratio of maximum to minimum particles per processor). If this imbalance exceeds the specified threshold, a rebalance is performed. If no frequency is specified, the default is to perform no rebalancing.

Parameter	Value	Default
<i>{=}</i>	{= are is }	–
<i>Nsteps</i>	integer	–

Particle Reservoir Size

Syntax

Particle Reservoir Size *{=}* *Size*

Summary

Keep a reservoir of inactive particles to minimize topology changes and reduce the number of output files.

Description

The exodus output file format requires a constant number of mesh objects across all timesteps. In particle simulations, particle mesh objects are created and destroyed whenever they enter or exit the domain; in parallel, particle mesh objects must also be created or destroyed whenever they enter or leave the local mesh partition. This results in a new exodus file being written at each output time step. These files can be concatenated in EnSight or ParaView using filename wildcards, but the number of files can become unwieldy. To alleviate

this, Fuego allows a reservoir of inactive particles to be created and stored. At the beginning of the simulation, the reservoir is filled with the number of inactive particles specified by the `\texttt{PARTICLE RESERVOIR SIZE}` command. When a new particle is introduced to the flow, instead of creating or destroying a mesh object, a particle from the reservoir is activated. Similarly, when a particle passes out of the domain, its mesh object is inactivated and returned to the reservoir. An element variable with field name “active” can be output, equal to 1 for active particles and 0 for inactive; this variable can aid in visualization of only the active particles. If the reservoir shrinks to size zero or grows to twice its specified size, it is reset to the specified size; this results in a change in the number of mesh objects and the need for a new exodus file. However, by choosing a reasonable reservoir size the user can reduce the total number of output files; the ideal size is approximately the maximum number of particles expected on any one processor at any given time throughout the course of the simulation.

Parameter	Value	Default
<i>/=</i>	{ = are is }	–
<i>Size</i>	integer	–

Perform Initial Particle Rebalance

Syntax

Perform Initial Particle Rebalance

Summary

Perform a particle rebalance during initialization

Use Finite Element Model

Syntax

Use Finite Element Model *ModelName* [Model Coordinates Are *Nodal_variable_name*]

Summary

Associates a predefined finite element model with this region.

Parameter	Value	Default
<i>ModelName</i>	string	–

Create Initial Particle Distribution

Scope

Particle Region

Summary

Defines an initial distribution of particles within the region's volume based on the mesh

This feature is DEPRECATED – it is difficult to ensure that particle creation along the elements of the mesh is applied consistently

begin Create Initial Particle Distribution *DistributionName*

Diameter {=} *Diameter*

X_Velocity {=} *X_velocity*

Y_Velocity {=} *Y_velocity*

Z_Velocity {=} *Z_velocity*

Particle Definition {=} *Definition*

end Create Initial Particle Distribution *DistributionName*

Line Commands

Diameter

Syntax

Diameter {=} *Diameter*

Summary

Set initial diameter of particle(s)

Parameter	Value	Default
{=}	{= are is}	–
<i>Diameter</i>	real	–

X_Velocity

Syntax

X_Velocity {=} *X_velocity*

Summary

Set initial x velocity for particle(s).

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>X_velocity</i>	real	–

Y_Velocity

Syntax

Y_Velocity *{=}* *Y_velocity*

Summary

Set initial y velocity for particle(s).

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>Y_velocity</i>	real	–

Z_Velocity

Syntax

Z_Velocity *{=}* *Z_velocity*

Summary

Set initial z velocity for particle(s).

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>Z_velocity</i>	real	–

Particle Definition

Syntax

Particle Definition *{=}* *Definition*

Summary

Set particle definition to be used for this block

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>Definition</i>	string	–

Create Particles From File

Scope

Particle Region

Summary

Block that defines creation of particles using a tab-delimited data file

Description

This block defines the creation of a set of particles based on data in a tab-delimited text file. The first line of this file should contain a single integer, giving the number of particles to be created. The rest of the file defines the particle data, one particle per line. There are two accepted formats for the particle file: 7-columns and 10-columns. The following is required in the correct order for the 7-column format: particle positions x, y, and z; particle velocities u, v, and w; and particle diameter d. 3 additional fields are required for the 10-column format, and the fields should be in this order: particle positions x, y, and z; particle velocities u, v, and w; particle diameter d; particle temperature T; particle number represented (parcelling); and particle insertion time.

Note that the particle temperature and number represented, and insertion time are specified as constant for all particles in the file when using the 7-column format.

begin Create Particles From File *BlockName*

Creation_Time {=} *CreationTime*

Diameter {=} *Diameter*

Filename {=} *FileName*

Interaction List Filename {=} *InteractionListFileName*

Insertion Time {=} *InsertionTime*

Length_Scale_Factor {=} *LengthScaleFactor*

Mass Represented {=} *Mass_represented*

Mass_Fraction *Species* {=} *y0*

Number Represented {=} *Num_represented*

Porosity {=} *Porosity*

Time_Scale_Factor {=} *TimeScaleFactor*

Track Particles For Tabular Output

Temperature {=} *Temperature*

X_Velocity {=} *X_velocity*

Y_Velocity {=} *Y_velocity*

Z_Velocity {=} *Z_velocity*

Particle Definition {=} *Definition*

end Create Particles From File *BlockName*

Line Commands

Creation_Time

Syntax

Creation_Time {=} *CreationTime*

Summary

Set time at which to insert particles

Parameter	Value	Default
{=}	{= are is}	–
<i>CreationTime</i>	real	–

Diameter

Syntax

Diameter {=} *Diameter*

Summary

Set initial diameter of particle(s)

Parameter	Value	Default
{=}	{= are is}	–
<i>Diameter</i>	real	–

Filename

Syntax

Filename *{=}* *FileName*

Summary

Name of file containing tab-delimited particle data.

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>FileName</i>	string	–

Interaction List Filename**Syntax**

Interaction List Filename *{=}* *InteractionListFileName*

Summary

Name of file containing interaction list.

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>InteractionListFileName</i>	string	–

Insertion Time**Syntax**

Insertion Time *{=}* *InsertionTime*

Summary

Set time at which to insert particle; particle will be inserted at the first time step at or after this value.

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>InsertionTime</i>	real	–

Length_Scale_Factor**Syntax**

Length_Scale_Factor *{=}* *LengthScaleFactor*

Summary

Multiplicative factor to be applied to all lengths in the input file (e.g. if file data is in meters and simulation is in cm, this factor should be 0.01)

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>LengthScaleFactor</i>	real	–

Mass Represented

Syntax

Mass Represented {=} *Mass_represented*

Summary

Set mass of physical particles represented by each particle node.

Parameter	Value	Default
{=}	{= are is}	–
<i>Mass_represented</i>	real	–

Mass_Fraction

Syntax

Mass_Fraction *Species* {=} *y0*

Summary

Specification of mass fraction to be used for computing particle mass initial condition.

Parameter	Value	Default
<i>Species</i>	string	–
{=}	{= are is}	–
<i>y0</i>	real	–

Number Represented

Syntax

Number Represented {=} *Num_represented*

Summary

Set number of physical particles represented by each particle node.

Parameter	Value	Default
{=}	{= are is}	–
<i>Num_represented</i>	real	1

Porosity

Syntax

Porosity {=} *Porosity*

Summary

Set initial porosity for particle(s).

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>Porosity</i>	real	–

Time_Scale_Factor

Syntax

Time_Scale_Factor *{=}* *TimeScaleFactor*

Summary

Multiplicative factor to be applied to all time values in the input file

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>TimeScaleFactor</i>	real	–

Track Particles For Tabular Output

Syntax

Track Particles For Tabular Output

Summary

All particles of this description are added to the tabular output

Description

Any particle matching this description will be output to the tabular data

Temperature

Syntax

Temperature *{=}* *Temperature*

Summary

Set initial temperature for particle(s).

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>Temperature</i>	real	–

X_Velocity

Syntax

X_Velocity *{=}* *X_velocity*

Summary

Set initial x velocity for particle(s).

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>X_velocity</i>	real	–

Y_Velocity

Syntax

Y_Velocity *{=}* *Y_velocity*

Summary

Set initial y velocity for particle(s).

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>Y_velocity</i>	real	–

Z_Velocity

Syntax

Z_Velocity *{=}* *Z_velocity*

Summary

Set initial z velocity for particle(s).

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>Z_velocity</i>	real	–

Particle Definition

Syntax

Particle Definition *{=}* *Definition*

Summary

Set particle definition to be used for this block

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>Definition</i>	string	–

Fluid Variable Specification

Scope

Particle Region

Summary

Specify fluid variables that will be used in the absence of transfers from the Fuego fluid region.

begin Fluid Variable Specification *SpecificationName*

Density *{=} Density*

Molecular_Weight *{=} Molec_weight*

Specific_Heat *{=} Specific_heat*

Temperature *{=} Temperature*

Thermal_Conductivity *{=} Thermal_cond*

Turbulent_Dissipation *{=} Turbulent_diss*

Turbulent_Kinetic_Energy *{=} Turbulent_ke*

Viscosity *{=} Viscosity*

X_Velocity *{=} X_velocity*

Y_Velocity *{=} Y_velocity*

Z_Velocity *{=} Z_velocity*

end Fluid Variable Specification *SpecificationName*

Line Commands

Density

Syntax

Density *{=} Density*

Summary

Set density for fluid.

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>Density</i>	real	–

Molecular_Weight

Syntax

Molecular_Weight *{=}* *Molec_weight*

Summary

Set molecular weight for fluid.

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>Molec_weight</i>	real	–

Specific_Heat

Syntax

Specific_Heat *{=}* *Specific_heat*

Summary

Set specific heat for fluid.

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>Specific_heat</i>	real	–

Temperature

Syntax

Temperature *{=}* *Temperature*

Summary

Set temperature for fluid.

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>Temperature</i>	real	–

Thermal_Conductivity

Syntax

Thermal_Conductivity *{=}* *Thermal_cond*

Summary

Set thermal conductivity for fluid.

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>Thermal_cond</i>	real	–

Turbulent_Dissipation

Syntax

Turbulent_Dissipation *{=}* *Turbulent_diss*

Summary

Set turbulent dissipation for fluid.

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>Turbulent_diss</i>	real	–

Turbulent_Kinetic_Energy

Syntax

Turbulent_Kinetic_Energy *{=}* *Turbulent_ke*

Summary

Set turbulent kinetic energy for fluid.

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>Turbulent_ke</i>	real	–

Viscosity

Syntax

Viscosity *{=}* *Viscosity*

Summary

Set viscosity for fluid.

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>Viscosity</i>	real	–

X_Velocity

Syntax

X_Velocity {=} *X_velocity*

Summary

Set x velocity for fluid.

Parameter	Value	Default
{=}	{= are is}	–
<i>X_velocity</i>	real	–

Y_Velocity**Syntax**

Y_Velocity {=} *Y_velocity*

Summary

Set y velocity for fluid.

Parameter	Value	Default
{=}	{= are is}	–
<i>Y_velocity</i>	real	–

Z_Velocity**Syntax**

Z_Velocity {=} *Z_velocity*

Summary

Set z velocity for fluid.

Parameter	Value	Default
{=}	{= are is}	–
<i>Z_velocity</i>	real	–

Particle Postprocessing**Scope**

Particle Region

Summary

Block that defines creation of fields for postprocessing and possible output

Description

Fields will be registered, created and computed throughout the simulation. Users can then request this field to be outputted through the results output.

begin Particle Postprocessing *Name*

Postprocess {*density_p* | *diameter_p* | *film_temperature_p* | *nusselt_*
→*fluid_p* | *nusselt_particle_p* | *specific_heat_p* | *surface_tension_p* |
→*thermal_conductivity_p* | *viscosity_p*}

end Particle Postprocessing *Name*

Line Commands

Postprocess

Syntax

Postprocess {*density_p* | *diameter_p* | *film_temperature_p* | *nusselt_fluid_p* |
nusselt_particle_p | *specific_heat_p* | *surface_tension_p* |
thermal_conductivity_p | *viscosity_p*}

Summary

Designate a postprocess variable to be created and post processed for optional output in results file.

Parameter	Value	Default
<i>Particle-PostProcessID</i>	{ <i>density_p</i> <i>diameter_p</i> <i>film_temperature_p</i> <i>nusselt_fluid_p</i> <i>nusselt_particle_p</i> <i>specific_heat_p</i> <i>surface_tension_p</i> <i>thermal_conductivity_p</i> <i>viscosity_p</i> }	–

Particle Postprocess Operation

Scope

Particle Region

Summary

Defines a custom post-processor block to be run at the end of the time step.

Description

The block type defines the post-processor operation performed, as in:

Begin particle postprocess operation Integral

or

Begin particle postprocess operation Average

The valid types are described below.

Integral

Perform an integral of the specified scalar function (f) on either the specified volume (blocks) or surface (sidesets) and save the result as a global variable. The function f can depend on time and space (t,x,y,z) and any nodal variable.

On volumes this is:

$$F = \int_V f dV$$

while on surfaces it is:

$$F = \int_A f dA$$

Average

Perform a volume or area weighted average of the specified scalar function (f) on either the specified volume (blocks) or surface (sidesets) and save the result as a global variable. The function f can depend on time and space (t,x,y,z) and any nodal variable.

On volumes this is:

$$F = \frac{\int_V f dV}{\int_V dV}$$

while on surfaces it is:

$$F = \frac{\int_A f dA}{\int_A dA}$$

Sum

Find the nodal summation of the specified scalar function (f) on either the specified volume (blocks) or surface (sidesets) and save the result as a global variable. The function f can depend on time and space (t,x,y,z) and any nodal variable.

Min

Find the nodal minimum of the specified scalar function (f) on either the specified volume (blocks) or surface (sidesets) and save the result as a global variable. The function f can depend on time and space (t,x,y,z) and any nodal variable.

Max

Find the nodal maximum of the specified scalar function (f) on either the specified volume (blocks) or surface (sidesets) and save the result as a global

variable. The function f can depend on time and space (t,x,y,z) and any nodal variable.

L2_Norm

Find the nodal L2-norm of the specified scalar function (f) on either the specified volume (blocks) or surface (sidesets) and save the result as a global variable. The function f can depend on time and space (t,x,y,z) and any nodal variable.

On volumes this is:

$$F = \sqrt{\int_V f^2 dV}$$

while on surfaces it is:

$$F = \sqrt{\int_A f^2 dA}$$

Global

Evaluate a global function (f) and save the result as a global variable. The function f can depend on time and any other global variable. You should not specify any location entries for a global post-processor.

```
Begin particle postprocess operation Global
  Output name = maxTempC
  Function = "maxTemp - 273.15"
End
```

Nodal_Field

Evaluate the specified scalar function (f) on either the specified volume (blocks) or surface (sidesets) and save the result in a nodal field. The function f can depend on time and space (t,x,y,z) and any nodal variable.

```
Begin particle postprocess operation Nodal_Field
  Output name = nuCalc
  Location = all_blocks
  Function = "viscosity/density"
End
```

Integrated_Flux

Evaluate the integrated flux of a specified vector function (requires 2 or 3 components for f depending on problem dimension). This must be performed on surfaces, not on volumes. The result is saved in a global variable. The

functions can depend on time and space (t, x, y, z) and any nodal variable.

$$F = \int_A \vec{f} \cdot \vec{dA}$$

```
Begin particle postprocess operation Integrated_Flux
  Output name = mass_flux
  Location = surface_1
  Function = "density*x_velocity" \$
            "density*y_velocity" \$
            "density*z_velocity"
End
```

begin Particle Postprocess Operation

Function *{=}* *FunctionStr...*

Location *{=}* *MeshEntites...*

Output Name *{=}* *OutputName*

end Particle Postprocess Operation

Line Commands

Function

Syntax

Function *{=}* *FunctionStr...*

Summary

Provide a string function to evaluate in the post-processor.

Description

A quoted function string for the post-processor to evaluate. For the FLUX post-processor you must provide multiple quoted entries - one per spatial dimension. For all other types you may only provide a single function string.

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>FunctionStr</i>	“string”...	–

Location

Syntax

Location *{=}* *MeshEntites...*

Summary

Mesh locations to evaluate the post-processor at.

Description

A list of block or sideset names to evaluate the post-processor at. For multiple blocks you can either include them all in one line or add separate lines. The aliases “all_blocks” and “all_surfaces” can also be used. You cannot mix blocks and sidesets in a single post-processor block.

```
Begin particle postprocess operation nodal_field
  Output name = nuCalc
  Location = all_blocks
  Function = "viscosity/density"
End
```

```
Begin particle postprocess operation nodal_field
  Output name = nuCalc
  Location = block_1 block_2 block_3
  Function = "viscosity/density"
End
```

```
Begin particle postprocess operation nodal_field
  Output name = nuCalc
  Location = block_1
  Location = block_2
  Location = block_3
  Function = "viscosity/density"
End
```

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>MeshEntites</i>	string . .	–

Output Name

Syntax

Output Name *{=}* *OutputName*

Summary

Define the name for the output variable (global or nodal depending on the type).

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>OutputName</i>	string	–

Heartbeat

Scope

Average Region, Fuego Region, Input_Output Region, Particle Region

Summary

Describes the location and type of the output stream used for outputting the heartbeat information for the enclosing region.

begin Heartbeat *Label*

Additional Steps *{=} List_of_steps...*

Additional Times *{=} List_of_times...*

Append *{=} {false | off | on | true}*

At Step *n {increment | interval} {=} m*

At Time *Dt1 {increment | interval} {=} Dt2*

Auto Output *{all | element | global | nodal}* User Defined Variables [*↵*
In *UserOutputHeartBeatList...*]

Element [*VariableList...*]

Exists *{=} {abort | append | overwrite}*

Face [*VariableList...*]

Format *{=} {csv | original | spyhis}*

Global [*Variables...*]

Labels *{=} {off | on}*

Legend *{=} {off | on}*

Monitor *{= | the} {history | restart | results}*

Nodal [*VariableList...*]

Node [*VariableList...*]

Nodeset [*VariableList...*]

```

Output On Signal {=} {sigabrt | sigalrm | sigfpe | sighup | sigill |
↪sigint | sigkill | sigpipe | sigquit | sigsegv | sigterm | sigusr1 |
↪sigusr2}

Precision {=} Precision

Start Time {=} Start_time

Stream Name {=} OutputFilename

Synchronize Output

Termination Time {=} Final_time

Timestamp Format

Timestep Adjustment Interval {=} Nsteps

Use Output Scheduler Timer_name

Variable {=} {edge | element | face | global | nodal | node} Variable_
↪list...

end Heartbeat Label

```

Line Commands

Additional Steps

Syntax

Additional Steps {=} *List_of_steps...*

Summary

Additional simulation steps when output should occur.

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>List_of_steps</i>	integer...	–

Additional Times

Syntax

Additional Times {=} *List_of_times...*

Summary

Additional simulation times when output should occur.

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>List_of_times</i>	real. . .	–

Append

Syntax

Append *{=}* *{false | off | on | true}*

Summary

Specifies whether the heartbeat file is appended if it exists. By default, the file is appended if restart is requested and not if restart is not requested. This option does not work for automatic restarts because a new heartbeat file is written with each auto restart.

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>Option</i>	{false off on true}	–

At Step

Syntax

At Step *n* *{increment | interval}* *{=}* *m*

Summary

Specify an output interval in terms of the internal iteration step count. The first step specifies the step count at the beginning of this interval and the second step specifies the output frequency to be used within this interval.

Parameter	Value	Default
<i>n</i>	integer	–
<i>Option</i>	{increment interval}	–
<i>{=}</i>	{= are is}	–
<i>m</i>	integer	–

At Time

Syntax

At Time *Dt1* *{increment | interval}* *{=}* *Dt2*

Summary

Specify an output interval in terms of the internal simulation time. The first time

specifies the time at the beginning of this time interval and the second time specifies the output frequency to be used within this interval.

Parameter	Value	Default
<i>Dt1</i>	real	–
<i>Option</i>	{increment interval}	–
<i>{=}</i>	{= are is}	–
<i>Dt2</i>	real	–

Auto Output

Syntax

Auto Output *{all | element | global | nodal}* User Defined Variables [In *UserOutputHeartBeatList...*]

Summary

Allows users to automatically output all user output defined variables for the type requested.

Parameter	Value	Default
<i>auto_output_type_4</i>	{all element global nodal}	–

Element

Syntax

Element [*VariableList...*]

Summary

Define the element variables that should be written to the heartbeat database. The syntax is: “element {internal_name} at element {id} as {DBname}” or “element {internal_name} nearest location X, Y, Z as {DBname}”.

Where {internal_name} is the name of the variable in the Sierra application; and {DBname} is the name as it should appear on the heartbeat database.

Exists

Syntax

Exists *{=}* *{abort | append | overwrite}*

Summary

Specify the behavior when creating this database and there is an existing file with the same name. The default behavior is “OVERWRITE” which deletes the existing file and creates a new file of the same name. “APPEND” will (if possible) append the new data to the end of the existing file. “ABORT” will print an error message and end the analysis.

Parameter	Value	Default
<i>{=}</i>	{= is}	–
<i>Option2</i>	{abort append overwrite}	–

Face

Syntax

Face [*VariableList...*]

Summary

Define the face variables that should be written to the heartbeat database. The syntax is: “face {internal_name} at face {id} as {DBname}” or “face {internal_name} nearest location X, Y, Z as {DBname}”.

Where {internal_name} is the name of the variable in the Sierra application; and {DBname} is the name as it should appear on the heartbeat database.

Format

Syntax

Format *{=} {csv | original | spyhis}*

Summary

The stream type/format to be used for the output results. The only three options at this time are ‘Original’ which is the old default Sierra heartbeat format; ‘SpyHis’ which mimics the CTH Spyhis history output format; and ‘CSV’

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>StreamTypes</i>	{csv original spyhis}	–

Global

Syntax

Global [*Variables...*]

Summary

Define the global/reduction variables that should be written to the heartbeat database. The syntax is: “global {internal_name} as {DBname}”.

Where {internal_name} is the name of the variable in the Sierra application; and {DBname} is the name as it should appear on the heartbeat database.

Labels

Syntax

Labels *{=} {off | on}*

Summary

Specifies whether labels will be displayed or just the value of the variable.
Labels will be shown if this line is not present.

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>Option</i>	{ off on }	on

Legend

Syntax

Legend *{=}* *{off | on}*

Summary

Specifies whether a legend will be displayed prior to outputting any variables.
The legend will not be shown unless this line is present. The legend shows the names of the variables that will be written to the heartbeat output stream. If the variable has multiple components, then the component count is shown after the variable e.g., velocity(3).

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>Option</i>	{ off on }	on

Monitor

Syntax

Monitor *{= | the}* *{history | restart | results}*

Summary

Specifies whether a line will be written to the heartbeat stream when either the results, history, and/or restart data are output.

Parameter	Value	Default
<i>Equals</i>	{ = the }	–
<i>Option</i>	{ history restart results }	–

Nodal

Syntax

Nodal [*VariableList. . .*]

Summary

Define the nodal variables that should be written to the heartbeat database. The syntax is: “nodal {internal_name} at node {id} as {DBname}” or “nodal {internal_name} nearest location X, Y, Z as {DBname}”.

Where {internal_name} is the name of the variable in the Sierra application; and {DBname} is the name as it should appear on the heartbeat database.

Node

Syntax

Node [*VariableList...*]

Summary

Define the nodal variables that should be written to the heartbeat database. The syntax is: “node {internal_name} at node {id} as {DBname}” or “node {internal_name} nearest location X, Y, Z as {DBname}”.

Where {internal_name} is the name of the variable in the Sierra application; and {DBname} is the name as it should appear on the heartbeat database.

Nodeset

Syntax

Nodeset [*VariableList...*]

Summary

Define the nodeset variables that should be written to the heartbeat database. The syntax is: “nodeset {internal_name} at node {id} as {DBname}” or “nodeset {internal_name} nearest location X, Y, Z as {DBname}”.

Where {internal_name} is the name of the variable in the Sierra application; and {DBname} is the name as it should appear on the heartbeat database. This option finds a single value for the {internal_name} specified without having to specify a nodeset id or name.

Output On Signal

Syntax

Output On Signal *{=}* *{sigabrt | sigalrm | sigfpe | sighup | sigill | sigint | sigkill | sigpipe | sigquit | sigsegv | sigterm | sigusr1 | sigusr2}*

Summary

When the specified signal is raised, the output stream associated with this block will be output.

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>Signals</i>	{ sigabrt sigalrm sigfpe sighup sigill sigint sigkill sigpipe sigquit sigsegv sigterm sigusr1 sigusr2 }	–

Precision

Syntax

Precision *{=}* *Precision*

Summary

The precision to be used for the output of real variables (default=5).

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>Precision</i>	integer	5

Start Time

Syntax

Start Time *{=}* *Start_time*

Summary

Specify the time to start outputting results from this output request block. This time overrides all ‘at time’ and ‘at step’ specifications.

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>Start_time</i>	real	–

Stream Name

Syntax

Stream Name *{=}* *OutputFilename*

Summary

The filename of where the heartbeat data should be written. If the filename begins with the ‘/’ character, it is an absolute path; otherwise, the path to the current directory will be prepended to the name. In addition, there are several predefined streams that can be specified. The predefined streams are ‘cout’ or ‘stdout’ specifies standard output; ‘cerr’, ‘stderr’, ‘clog’, or ‘log’ specifies standard error; ‘output’ or ‘outputP0’ specifies Sierra’s standard output which is redirected to the file specified by the ‘-o’ option on the command line. If the file already exists, it is overwritten. If this line is omitted, then a filename will be created from the basename of the input file with a “.hrt” suffix appended.

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>OutputFilename</i>	string	–

Synchronize Output

Syntax

Synchronize Output

Summary

In an analysis with multiple regions, it is sometimes desirable to synchronize the output of results data between the regions. This can be done by adding the SYNCHRONIZE OUTPUT command line to the results output block. If a results block has this set, then it will write output whenever a previous region writes output. The ordering of regions is based on the order in the input file, algorithmic considerations, or by solution control specifications.

Although the USE OUTPUT SCHEDULER command line can also synchronize output between regions, the SYNCHRONIZE OUTPUT command line will synchronize the output with regions where the output frequency is not under the direct control of the Sierra IO system. Examples of this are typically coupled applications where one or more of the codes are not Sierra-based applications such as Alegria and CTH. A results block with SYNCHRONIZE OUTPUT specified will also synchronize its output with the output of the external code.

The SYNCHRONIZE OUTPUT command can be used with other output scheduling commands such as time-based or step-based output specifications.

Termination Time

Syntax

Termination Time *{=}* *Final_time*

Summary

Specify the time to stop outputting results from this output request block.

Parameter	Value	Default
<i>{=}</i>	{= are is }	–
<i>Final_time</i>	real	–

Timestamp Format

Syntax

Timestamp Format

Summary

The format to be used for the timestamp. See ‘man strftime’ for more information.

Timestep Adjustment Interval

Syntax

Timestep Adjustment Interval *{=}* *Nsteps*

Summary

Specify the number of steps to ‘look ahead’ and adjust the timestep to ensure

that the specified output times or simulation end time will be hit ‘exactly’.

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>Nsteps</i>	integer	–

Use Output Scheduler

Syntax

Use Output Scheduler *Timer_name*

Summary

Associates a predefined output scheduler with this output block (results, restart, heartbeat, or history).

Parameter	Value	Default
<i>Timer_name</i>	string	–

Variable

Syntax

Variable *{=}* *{edge | element | face | global | nodal | node}* *Variable_list. . .*

Summary

Define the variables that should be written to the heartbeat output. The user can request that the values of certain variables be output on the heartbeat line. These variables are limited to region and framework control data currently. The syntax is:

```
variable = {entity_type} {internal_name} at
           {entity_type} {entity_id}      as {external_name}
variable = {entity_type} {internal_name} nearest location
           {x,y,z} as {external_name}
```

For global variables, use:

```
variable = global {internal_name} [as {external_name}]
```

Where:

```
entity_type = node, element, face, edge, global
internal_name = Sierra variable name
entity_id = id of the node, element, face, edge that you want
           the specified variable output at.
external_name = name of variable on the database.
```

The names ‘timestep’, and ‘time’ can be specified as variables also. They are the current timestep and simulation time. This line can appear multiple times.

Parameter	Value	Default
<i>{=}</i>	{= are is }	–
<i>Option</i>	{edge element face global nodal node }	–
<i>Variable_list</i>	string...	–

History Output

Scope

Average Region, Fuego Region, Input_Output Region, Particle Region

Summary

Describes the location and type of the output stream used for outputting history for the enclosing region.

begin History Output *Label*

Additional Steps *{=}* *List_of_steps...*

Additional Times *{=}* *List_of_times...*

At Step *n* *{increment | interval}* *{=}* *m*

At Time *Dt1* *{increment | interval}* *{=}* *Dt2*

Auto Output *{all | element | global | nodal}* User Defined Variables [*↪*
*↪*In *UserOutputHistoryList...*]

Database Name *{=}* *StreamName*

Database Type *{=}* *{catalyst | catalyst_exodus | cgns | dof | dof_*
↪exodus | exodus | exodusii | exonull | generated | genesis | null |
↪parallel_exodus | textmesh}

Element [*VariableList...*]

Exists *{=}* *{abort | add_suffix | append | overwrite}*

Face [*VariableList...*]

Flush Interval *{=}* *Option*

```

Global [ Variables... ]

Nodal [ VariableList... ]

Node [ VariableList... ]

Nodeset [ VariableList... ]

Output On Signal {=} {sigabrt | sigalrm | sigfpe | sighup | sigill | ↪
sigint | sigkill | sigpipe | sigquit | sigsegv | sigterm | sigusr1 | ↪
sigusr2}

Overwrite {=} {false | no | off | on | true | yes}

Property PropertyName {=} PropertyValue

Start Time {=} Start_time

Synchronize Output

Termination Time {=} Final_time

Timestep Adjustment Interval {=} Nsteps

Title

Use Dynamic Topology Io

Use Output Scheduler Timer_name

Variable {=} {edge | element | face | global | nodal | node} Variable_
↪list...

end History Output Label

```

Line Commands

Additional Steps

Syntax

Additional Steps {=} *List_of_steps...*

Summary

Additional simulation steps when output should occur.

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>List_of_steps</i>	integer. . .	–

Additional Times

Syntax

Additional Times *{=}* *List_of_times. . .*

Summary

Additional simulation times when output should occur.

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>List_of_times</i>	real. . .	–

At Step

Syntax

At Step *n* *{increment | interval}* *{=}* *m*

Summary

Specify an output interval in terms of the internal iteration step count. The first step specifies the step count at the beginning of this interval and the second step specifies the output frequency to be used within this interval.

Parameter	Value	Default
<i>n</i>	integer	–
<i>Option</i>	{increment interval}	–
<i>{=}</i>	{= are is}	–
<i>m</i>	integer	–

At Time

Syntax

At Time *Dt1* *{increment | interval}* *{=}* *Dt2*

Summary

Specify an output interval in terms of the internal simulation time. The first time specifies the time at the beginning of this time interval and the second time specifies the output frequency to be used within this interval.

Parameter	Value	Default
<i>Dt1</i>	real	–
<i>Option</i>	{increment interval}	–
<i>{=}</i>	{= are is}	–
<i>Dt2</i>	real	–

Auto Output

Syntax

Auto Output *{all | element | global | nodal}* User Defined Variables [In *UserOutputHistoryList. . .*]

Summary

Allows users to automatically output all user output defined variables for the type requested.

Parameter	Value	Default
<i>auto_output_type_2</i>	{all element global nodal}	–

Database Name

Syntax

Database Name *{=}* *StreamName*

Summary

The base name of the database containing the output history. If the filename begins with the ‘/’ character, it is an absolute path; otherwise, the path to the current directory will be prepended to the name. If this line is omitted, then a filename will be created from the basename of the input file with a “.h” suffix appended.

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>StreamName</i>	string	–

Database Type

Syntax

Database Type *{=}* *{catalyst | catalyst_exodus | cgns | dof | dof_exodus | exodus | exodusii | exonull | generated | genesis | null | parallel_exodus | textmesh}*

Summary

The database type/format to be used for the output history.

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>Database Types</i>	{catalyst catalyst_exodus cgns dof dof_exodus exodus exodusii exonull generated genesis null parallel_exodus textmesh}	–

Element

Syntax

Element [*VariableList...*]

Summary

Define the element variables that should be written to the history database. The syntax is: “element {internal_name} at element {id} as {DBname}” or “element {internal_name} nearest location X, Y, Z as {DBname}”.

Where {internal_name} is the name of the variable in the Sierra application; and {DBname} is the name as it should appear on the history database.

Exists

Syntax

Exists *{=}* *{abort | add_suffix | append | overwrite}*

Summary

Specify the behavior when creating this database and there is an existing file with the same name. The default behavior is “OVERWRITE” which deletes the existing file and creates a new file of the same name. “APPEND” will (if possible) append the new data to the end of the existing file. “ABORT” will print an error message and end the analysis. “ADD_SUFFIX” will add a -s???? suffix where the ???? is replaced by a sequential number starting at 0002.

Parameter	Value	Default
<i>{=}</i>	{= is}	–
<i>Option2</i>	{abort add_suffix append overwrite}	–

Face

Syntax

Face [*VariableList...*]

Summary

Define the face variables that should be written to the history database. The syntax is: “face {internal_name} at face {id} as {DBname}” or “face {internal_name} nearest location X, Y, Z as {DBname}”.

Where {internal_name} is the name of the variable in the Sierra application; and {DBname} is the name as it should appear on the history database.

Flush Interval

Syntax

Flush Interval *{=}* *Option*

Summary

The minimum time interval (in seconds) at which output will be explicitly flushed to disk. The default is 10 seconds.

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>Option</i>	integer	10

Global

Syntax

Global [*Variables...*]

Summary

Define the global/reduction variables that should be written to the history database. The syntax is: “global {internal_name} as {DBname}”.

Where {internal_name} is the name of the variable in the Sierra application; and {DBname} is the name as it should appear on the history database.

Nodal

Syntax

Nodal [*VariableList...*]

Summary

Define the nodal variables that should be written to the history database. The syntax is: “nodal {internal_name} at node {id} as {DBname}” or “nodal {internal_name} nearest location X, Y, Z as {DBname}”.

Where {internal_name} is the name of the variable in the Sierra application; and {DBname} is the name as it should appear on the history database.

Node

Syntax

Node [*VariableList...*]

Summary

Define the nodal variables that should be written to the history database. The syntax is: “node {internal_name} at node {id} as {DBname}” or “node {internal_name} nearest location X, Y, Z as {DBname}”.

Where {internal_name} is the name of the variable in the Sierra application; and {DBname} is the name as it should appear on the history database.

Nodeset

Syntax

Nodeset [*VariableList...*]

Summary

Define the nodeset variables that should be written to the history database. The syntax is: “nodeset {internal_name} at node {id} as {DBname}” or “nodeset {internal_name} nearest location X, Y, Z as {DBname}”.

Where {internal_name} is the name of the variable in the Sierra application; and {DBname} is the name as it should appear on the history database.

Output On Signal

Syntax

Output On Signal *{=}* {*sigabrt | sigalrm | sigfpe | sighup | sigill | sigint | sigkill | sigpipe | sigquit | sigsegv | sigterm | sigusr1 | sigusr2*}

Summary

When the specified signal is raised, the output stream associated with this block will be output.

Parameter	Value	Default
<i>{=}</i>	{ = are is }	—
<i>Signals</i>	{ sigabrt sigalrm sigfpe sighup sigill sigint sigkill sigpipe sigquit sigsegv sigterm sigusr1 sigusr2 }	—

Overwrite

Syntax

Overwrite *{=}* {*false | no | off | on | true | yes*}

Summary

(DEPRECATED, Use EXISTS) Specify whether the database should be overwritten if it exists. The default behavior is to overwrite unless this command is specified in the output block and either off, false, or no is specified.

Parameter	Value	Default
<i>{=}</i>	{ = is }	—
<i>Option2</i>	{ false no off on true yes }	—

Property

Syntax

Property *PropertyName* *{=}* *PropertyValue*

Summary

Define a database property named “PropertyName” with the value “PropertyValue”. If PropertyValue consists of all digits, it will define an integer property. If PropertyValue is “true” or “yes” or “false” or “no”, it will define a logical property; otherwise it will define a string property. Supported properties are typically database dependent; Some history-related properties are:

- VARIABLE_NAME_CASE = upper|lower
- MAX_NAME_LENGTH = value (32)

Parameter	Value	Default
<i>PropertyName</i>	string	–
<i>{=}</i>	{ = are is }	–
<i>PropertyValue</i>	string	–

Start Time

Syntax

Start Time *{=}* *Start_time*

Summary

Specify the time to start outputting results from this output request block. This time overrides all ‘at time’ and ‘at step’ specifications.

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>Start_time</i>	real	–

Synchronize Output

Syntax

Synchronize Output

Summary

In an analysis with multiple regions, it is sometimes desirable to synchronize the output of results data between the regions. This can be done by adding the SYNCHRONIZE OUTPUT command line to the results output block. If a results block has this set, then it will write output whenever a previous region writes output. The ordering of regions is based on the order in the input file, algorithmic considerations, or by solution control specifications.

Although the USE OUTPUT SCHEDULER command line can also synchronize output between regions, the SYNCHRONIZE OUTPUT command line will synchronize the output with regions where the output frequency is not under the direct control of the Sierra IO system. Examples of this are typically coupled applications where one or more of the codes are not Sierra-based applications

such as Alegra and CTH. A results block with SYNCHRONIZE OUTPUT specified will also synchronize its output with the output of the external code.

The SYNCHRONIZE OUTPUT command can be used with other output scheduling commands such as time-based or step-based output specifications.

Termination Time

Syntax

Termination Time *{=}* *Final_time*

Summary

Specify the time to stop outputting results from this output request block.

Parameter	Value	Default
<i>{=}</i>	{= are is }	–
<i>Final_time</i>	real	–

Timestep Adjustment Interval

Syntax

Timestep Adjustment Interval *{=}* *Nsteps*

Summary

Specify the number of steps to ‘look ahead’ and adjust the timestep to ensure that the specified output times or simulation end time will be hit ‘exactly’.

Parameter	Value	Default
<i>{=}</i>	{= are is }	–
<i>Nsteps</i>	integer	–

Title

Syntax

Title

Summary

Specify the title to be used for this specific output block.

Use Dynamic Topology Io

Syntax

Use Dynamic Topology Io

Summary

Specify that the app use IO for dynamic topology modifications where the output files are stored in a single database. Legacy file format for dynamically changing topology results in the creation of multiple files for each output on a mesh modification. This option leverages the ability of netCDF to create mesh

groups within a single database and concatenate all mesh files into one. The names of each mesh group are of the form IOSS_MESH_GROUP-??? where ??? is the 1-based output index 1, 2, . . . , 10, . . . , 100, . . . Please note that netCDF has a current limit of 65,536 groups

Use Output Scheduler

Syntax

Use Output Scheduler *Timer_name*

Summary

Associates a predefined output scheduler with this output block (results, restart, heartbeat, or history).

Parameter	Value	Default
<i>Timer_name</i>	string	–

Variable

Syntax

Variable {=} {edge | element | face | global | nodal | node} *Variable_list* . .

Summary

Define the variables that should be written to the history database. The syntax is: “variable = entity {internal_name} at entity {id} as {DBname}” or “variable = entity {internal_name} nearest location X, Y, Z as {DBname}” or “variable = entity {internal_name} at location X, Y, Z as {DBname}”.

Where {entity} is ‘node’, ‘element’, ‘face’, or ‘edge’; {internal_name} is the name of the variable in the Sierra application; and {DBname} is the name as it should appear on the history database.

Parameter	Value	Default
{=}	{= are is}	–
<i>Option</i>	{edge element face global nodal node}	–
<i>Variable_list</i>	string. . .	–

Insert Particle

Scope

Particle Region

Summary

Block that defines data for an individual particle to be inserted into the flow.

begin Insert Particle *Particlename*

```

Diameter {=} Diameter

Insertion Time {=} InsertionTime

Mass Represented {=} Mass_represented

Mass_Fraction Species {=} y0

Number {=} Number

Number Represented {=} Num_represented

Porosity {=} Porosity

Track Particles For Tabular Output

Temperature {=} Temperature

X {=} x

X1 {=} x

X2 {=} x

X_Velocity {=} X_velocity

Y {=} y

Y1 {=} y

Y2 {=} y

Y_Velocity {=} Y_velocity

Z {=} z

Z1 {=} z

Z2 {=} z

Z_Velocity {=} Z_velocity

Particle Definition {=} Definition

end Insert Particle Particlename

```

Line Commands

Diameter

Syntax

Diameter *{=} Diameter*

Summary

Set initial diameter of particle(s)

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>Diameter</i>	real	–

Insertion Time

Syntax

Insertion Time *{=} InsertionTime*

Summary

Set time at which to insert particle; particle will be inserted at the first time step at or after this value.

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>InsertionTime</i>	real	–

Mass Represented

Syntax

Mass Represented *{=} Mass_represented*

Summary

Set mass of physical particles represented by each particle node.

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>Mass_represented</i>	real	–

Mass_Fraction

Syntax

Mass_Fraction *Species {=} y0*

Summary

Specification of mass fraction to be used for computing particle mass initial condition.

Parameter	Value	Default
<i>Species</i>	string	–
<i>{=}</i>	{= are is}	–
<i>y0</i>	real	–

Number

Syntax

Number *{=}* *Number*

Summary

Set number of particle nodes to insert. For a line insertion, this should be greater than 1, and points (x1,y1,z1) and (x2,y2,z2) should be input. This places a line of equally spaced particle nodes between the two points. If no second point is specified, all particles are placed at the same point. Note that if the NUMBER REPRESENTED command is used, the total number of physical particles equals NUMBER * NUMBER_REPRESENTED

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>Number</i>	integer	–

Number Represented

Syntax

Number Represented *{=}* *Num_represented*

Summary

Set number of physical particles represented by each particle node.

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>Num_represented</i>	real	1

Porosity

Syntax

Porosity *{=}* *Porosity*

Summary

Set initial porosity for particle(s).

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>Porosity</i>	real	–

Track Particles For Tabular Output

Syntax

Track Particles For Tabular Output

Summary

All particles of this description are added to the tabular output

Description

Any particle matching this description will be output to the tabular data

Temperature

Syntax

Temperature *{=}* *Temperature*

Summary

Set initial temperature for particle(s).

Parameter	Value	Default
<i>{=}</i>	{ = are is }	—
<i>Temperature</i>	real	—

X

Syntax

X *{=}* *x*

Summary

Set initial x position for particle(s).

Parameter	Value	Default
<i>{=}</i>	{ = are is }	—
<i>x</i>	real	—

X1

Syntax

X1 *{=}* *x*

Summary

Set x position for point 1 defining a line of particles.

Parameter	Value	Default
<i>{=}</i>	{ = are is }	—
<i>x</i>	real	—

X2

Syntax

X2 {=} *x*

Summary

Set x position for point 2 defining a line of particles.

Parameter	Value	Default
{=}	{= are is}	–
<i>x</i>	real	–

X_Velocity**Syntax**

X_Velocity {=} *X_velocity*

Summary

Set initial x velocity for particle(s).

Parameter	Value	Default
{=}	{= are is}	–
<i>X_velocity</i>	real	–

Y**Syntax**

Y {=} *y*

Summary

Set initial y position for particle(s).

Parameter	Value	Default
{=}	{= are is}	–
<i>y</i>	real	–

Y1**Syntax**

Y1 {=} *y*

Summary

Set y position for point 1 defining a line of particles.

Parameter	Value	Default
{=}	{= are is}	–
<i>y</i>	real	–

Y2

Syntax

Y2 {=} *y*

Summary

Set y position for point 2 defining a line of particles.

Parameter	Value	Default
{=}	{= are is}	–
<i>y</i>	real	–

Y_Velocity

Syntax

Y_Velocity {=} *Y_velocity*

Summary

Set initial y velocity for particle(s).

Parameter	Value	Default
{=}	{= are is}	–
<i>Y_velocity</i>	real	–

Z

Syntax

Z {=} *z*

Summary

Set initial z position for particle(s).

Parameter	Value	Default
{=}	{= are is}	–
<i>z</i>	real	–

Z1

Syntax

Z1 {=} *z*

Summary

Set z position for point 1 defining a line of particles.

Parameter	Value	Default
{=}	{= are is}	–
<i>z</i>	real	–

Z2

Syntax

Z2 {=} *z*

Summary

Set z position for point 2 defining a line of particles.

Parameter	Value	Default
{=}	{= are is}	–
<i>z</i>	real	–

Z_Velocity

Syntax

Z_Velocity {=} *Z_velocity*

Summary

Set initial z velocity for particle(s).

Parameter	Value	Default
{=}	{= are is}	–
<i>Z_velocity</i>	real	–

Particle Definition

Syntax

Particle Definition {=} *Definition*

Summary

Set particle definition to be used for this block

Parameter	Value	Default
{=}	{= are is}	–
<i>Definition</i>	string	–

Particle Average

Scope

Particle Region

Summary

Block that defines averaging over particles in each control volume cell.

begin Particle Average *AverageName*

Average Field *ParticleField* For Output As *CellField*

Averaging Type *{=}* *{mass | number | volume}*

Time Average Using Period *TimePeriod*

end Particle Average *AverageName*

Line Commands

Average Field

Syntax

Average Field *ParticleField* For Output As *CellField*

Summary

Define averaging operation from a field defined onto the particles into another field available for output on the fluid mesh.

Parameter	Value	Default
<i>ParticleField</i>	string	–
<i>CellField</i>	string	–

Averaging Type

Syntax

Averaging Type *{=}* *{mass | number | volume}*

Summary

Type of particle averaging to perform. Options are NUMBER, MASS, and VOLUME for number-weighted, mass-weighted, and volume weighted means.

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>AverageType</i>	{mass number volume}	–

Time Average Using Period

Syntax

Time Average Using Period *TimePeriod*

Summary

Average particle variables over time as well as volume, over specified time period.

Parameter	Value	Default
<i>TimePeriod</i>	real	–

Particle Breakup Model

Scope

Particle Region

Summary

Block that defines a specific particle breakup model.

begin Particle Breakup Model *ModelName*

Breakup Model Type *{=}* *{tab_model}*

Diameter Reduction Factor *{=}* *Factor*

Limit New Particle Splitting

Maximum Number Of Particles Per Parcel *{=}* *Number*

Minimum Mass Per Parcel *{=}* *Mass*

Particle Breakup Parameter *VarName* *{=}* *Value...*

end Particle Breakup Model *ModelName*

Line Commands

Breakup Model Type

Syntax

Breakup Model Type *{=}* *{tab_model}*

Summary

Type of breakup model.

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>ParticleBreakupModelType</i>	{tab_model}	–

Diameter Reduction Factor

Syntax

Diameter Reduction Factor {=} *Factor*

Summary

Limit reduction of particle diameter by the specified value.

Parameter	Value	Default
{=}	{= are is}	–
<i>Factor</i>	real	–

Limit New Particle Splitting**Syntax**

Limit New Particle Splitting

Summary

If a particle has just been created on the previous step, limit breakup to splitting the particle in half.

Maximum Number Of Particles Per Parcel**Syntax**

Maximum Number Of Particles Per Parcel {=} *Number*

Summary

After splitting, create new parcels if, and only if, number of particles in the parcel exceeds specified maximum. This may be superseded by a specified minimum mass.

Parameter	Value	Default
{=}	{= are is}	–
<i>Number</i>	integer	–

Minimum Mass Per Parcel**Syntax**

Minimum Mass Per Parcel {=} *Mass*

Summary

Limit creation of new parcels if parcel mass drops below the specified minimum.

Parameter	Value	Default
{=}	{= are is}	–
<i>Mass</i>	real	–

Particle Breakup Parameter

Syntax

Particle Breakup Parameter *VarName* {=} *Value*...

Summary

Parameter for the particle breakup model.

Description

The breakup model requires a number of user-defined parameters to be entered. An exception will be thrown if any of these required parameters are missing.

The TAB_MODEL breakup model requires parameters PARTICLE_VISCOSITY and PARTICLE_SURFACE_TENSION.

Parameter	Value	Default
<i>VarName</i>	string	–
<i>{=}</i>	{= are is}	–
<i>Value</i>	real. . .	–

Particle Definition

Scope

Particle Region

Summary

Block that defines a particle type, including all of its parameters. At least one of these blocks is necessary in every particle region

begin Particle Definition *Particlename*

Add Particle Interface *InterfaceName*

Add Particle Material *MaterialName*

Minimum_Diameter {=} *MinimumDiameter*

Minimum_Mass {=} *MinimumMass*

Particle Breakup Model {=} *ModelName*

Particle Energy Equation {=} {*chemically_reactive_energy_type* | *constant_particle_energy_type* | *heated_evaporating_energy_type* | *heated_particle_energy_type* | *wild_fire_energy_type*}

Particle Interaction Model {=} *ModelName*

Particle Is Stationary

```
Particle Type {=} {cpd_particle | evaporating_particle | fixed_heated_
↳particle | fixed_particle | generalized_particle | heated_particle |
↳inertial_particle | tracker | wildfire_particle}
```

```
Particle Velocity Equation {=} {inertial_particle_velocity | tracker_
↳particle_velocity}
```

```
Use Particle Species {=} UseParticleSpecies...
```

```
Use Species Material SpeciesMaterialName For SpeciesName
```

```
begin General Chemistry ChemistryName
end
```

```
begin Ode Solver Parameters SolverName
end
```

```
end Particle Definition Particlename
```

Line Commands

Add Particle Interface

Syntax

```
Add Particle Interface InterfaceName
```

Summary

Add a particle interface to the particle definition

Parameter	Value	Default
<i>InterfaceName</i>	string	–

Add Particle Material

Syntax

```
Add Particle Material MaterialName
```

Summary

Add a material component to the particle definition

Parameter	Value	Default
<i>MaterialName</i>	string	–

Minimum_Diameter

Syntax

Minimum_Diameter {=} *MinimumDiameter*

Summary

Set the minimum diameter before deletion of the particle.

Parameter	Value	Default
{=}	{ = are is }	–
<i>MinimumDiameter</i>	real	–

Minimum_Mass**Syntax**

Minimum_Mass {=} *MinimumMass*

Summary

Set the minimum mass before deletion of the particle.

Parameter	Value	Default
{=}	{ = are is }	–
<i>MinimumMass</i>	real	–

Particle Breakup Model**Syntax**

Particle Breakup Model {=} *ModelName*

Summary

Set a breakup model on this particle type

Parameter	Value	Default
{=}	{ = are is }	–
<i>ModelName</i>	string	–

Particle Energy Equation**Syntax**

Particle Energy Equation {=} {*chemically_reactive_energy_type* | *constant_particle_energy_type* | *heated_evaporating_energy_type* | *heated_particle_energy_type* | *wild_fire_energy_type*}

Summary

Specify the particle energy type for this particle block.

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>ParticleEnergyType</i>	{chemically_reactive_energy_type constant_particle_energy_type heated_evaporating_energy_type heated_particle_energy_type wild_fire_energy_type}	–

Particle Interaction Model

Syntax

Particle Interaction Model *{=}* *ModelName*

Summary

Set an interaction model on this particle type

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>ModelName</i>	string	–

Particle Is Stationary

Syntax

Particle Is Stationary

Summary

Marks the particle as being stationary, ie. the particle remains at the same position for the lifetime of the simulation.

Particle Type

Syntax

Particle Type *{=}* {*cpd_particle* | *evaporating_particle* | *fixed_heated_particle* | *fixed_particle* | *generalized_particle* | *heated_particle* | *inertial_particle* | *tracker* | *wildfire_particle*}

Summary

Specify the particle type for this particle block.

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>ParticleType</i>	{cpd_particle evaporating_particle fixed_heated_particle fixed_particle generalized_particle heated_particle inertial_particle tracker wildfire_particle}	–

Particle Velocity Equation

Syntax

Particle Velocity Equation *{=}* *{inertial_particle_velocity | tracker_particle_velocity}*

Summary

Specify the particle velocity type for this particle block.

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>ParticleVelocityType</i>	{inertial_particle_velocity tracker_particle_velocity}	–

Use Particle Species

Syntax

Use Particle Species *{=}* *UseParticleSpecies. . .*

Summary

Specifies the given particle species present within a particle.

Note that the number of species in a particle is determined through the list of species given in the input file.

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>UseParticleSpecies</i>	string. . .	–

Use Species Material

Syntax

Use Species Material *SpeciesMaterialName* For *SpeciesName*

Summary

Assign a material type to a species specified in USE PARTICLE SPECIES

Description

Allows for material types defined in BEGIN PARTICLE MATERIAL to be used as a specific species specified within the PARTICLE DEFINITION block.

Parameter	Value	Default
<i>SpeciesMaterialName</i>	string	–
<i>SpeciesName</i>	string	–

Particle Filled Shape

Scope

Particle Region

Summary

Block that defines data for a particle filled shape.

begin Particle Filled Shape *FilledShapeName*

Diameter {=} *Diameter*

Insertion Time {=} *InsertionTime*

Mass Represented {=} *Mass_represented*

Mass_Fraction *Species* {=} *y0*

Number Represented {=} *Num_represented*

Particle Diameter Distribution Parameter *VarName* {=} *Value*

Particle Diameter Distribution Type {=} {*constant* | *lognormal* | *normal* |
→ *rosin_rammler* | *uniform*}

Particle Velocity Distribution Parameter *VarName* {=} *Value*

Particle Velocity Distribution Type {=} {*constant* | *lognormal* | *normal* |
→ *rosin_rammler* | *uniform*}

Particle Shape Parameter *VarName* {=} *Value...*

Particle Shape Type {=} {*brick* | *cone* | *cylinder* | *sphere*}

Porosity {=} *Porosity*

Track Particles For Tabular Output

Temperature {=} *Temperature*

X_Velocity {=} *X_velocity*

Y_Velocity {=} *Y_velocity*

Z_Velocity {=} *Z_velocity*

Particle Definition {=} *Definition*

end Particle Filled Shape *FilledShapeName*

Line Commands

Diameter

Syntax

Diameter {=} *Diameter*

Summary

Set initial diameter of particle(s)

Parameter	Value	Default
{=}	{= are is}	–
<i>Diameter</i>	real	–

Insertion Time

Syntax

Insertion Time {=} *InsertionTime*

Summary

Set time at which to insert particle; particle will be inserted at the first time step at or after this value.

Parameter	Value	Default
{=}	{= are is}	–
<i>InsertionTime</i>	real	–

Mass Represented

Syntax

Mass Represented {=} *Mass_represented*

Summary

Set mass of physical particles represented by each particle node.

Parameter	Value	Default
{=}	{= are is}	–
<i>Mass_represented</i>	real	–

Mass_Fraction

Syntax

Mass_Fraction *Species* {=} *y0*

Summary

Specification of mass fraction to be used for computing particle mass initial condition.

Parameter	Value	Default
<i>Species</i>	string	–
{=}	{= are is}	–
<i>y0</i>	real	–

Number Represented

Syntax

Number Represented {=} *Num_represented*

Summary

Set number of physical particles represented by each particle node.

Parameter	Value	Default
{=}	{= are is}	–
<i>Num_represented</i>	real	1

Particle Diameter Distribution Parameter

Syntax

Particle Diameter Distribution Parameter *VarName* {=} *Value*

Summary

Parameter for the probability distribution function for particle diameters.

Description

The number and type of parameters needed depends on the type of the probability distribution being used. The available distributions are described below.

CONSTANT

Required Parameters: VALUE

Always returns a constant value

UNIFORM

Required Parameters: MIN, MAX

Returns a random value uniformly distributed between the provided min and max values.

NORMAL

Required Parameters: MEAN, STDEV

Optional Parameters: MIN, MAX

Returns a random value normally distributed using the provided mean and standard deviation. If optional min or max values are provided then the PDF will be clipped to zero outside those bounds. The PDF for this distribution is:

$$P(x) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2\right)$$

LOGNORMAL

Required Parameters: MEAN and STDEV or LOGMEAN and LOGSTDEV or MEDIAN and SCATTER

Optional Parameters: MIN, MAX

Returns a random value with a log-normal distribution. If optional min or max values are provided then the PDF will be clipped to zero outside those bounds. The PDF for this distribution is:

$$P(x) = \frac{1}{x\sigma\sqrt{2\pi}} \exp\left(-\frac{1}{2}\left(\frac{\ln(x)-\mu}{\sigma}\right)^2\right)$$

There are several ways to specify the parameters needed for this distribution (μ and σ). You can provide them directly (using LOGMEAN and LOGSTDEV), or you can provide non-log-scaled parameters and have them calculated.

Keep in mind that unlike a normal distribution, the peak of a lognormal distribution (its mode) is not equal to its mean or median. For a log-normal distribution the mean is

$$\text{Mean} = \mu_X = \exp\left(\mu + \frac{\sigma^2}{2}\right)$$

the median is

$$\text{Median} = \exp(\mu)$$

and the mode (peak of the distribution) is

$$\text{Mode} = \exp\left(\mu - \sigma^2\right)$$

If you provide a mean (μ_X , MEAN) and standard deviation (σ_X , STDEV), then the log-mean (μ) and log-standard deviation (σ) will be calculated using

$$\mu = \ln\left(\frac{\mu_X^2}{\sqrt{\mu_X^2 + \sigma_X^2}}\right)$$

$$\sigma = \sqrt{\ln \left(1 + \frac{\sigma_X^2}{\mu_X^2} \right)}$$

Alternately, if you provide a median (μ^* , MEDIAN) and scatter (σ^* , SCATTER), then log-mean (μ) and log-standard deviation (σ) will be calculated using

$$\mu = \ln (\mu^*)$$

$$\sigma = \ln (\sigma^*)$$

The median and scatter physical interpretation is best understood by the fact that about two thirds of the distribution will lie between μ^*/σ^* and $\mu^*\sigma^*$. Similarly, 95 percent of the distribution will lie between μ^*/σ^{*2} and $\mu^*\sigma^{*2}$. As such, the value for σ^* (SCATTER) must be a value greater than one since it is a multiplier of the median.

ROSIN_RAMMLER

Required Parameters: X, Q

Optional Parameters: MIN, MAX

Returns a random value with a Rosin-Rammler (or Weibull) distribution. If optional min or max values are provided then the PDF will be clipped to zero outside those bounds. The PDF for this distribution is:

$$P(x) = \frac{Q}{X} \left(\frac{x}{X} \right)^{Q-1} \exp \left(- \left(\frac{x}{X} \right)^Q \right)$$

The values provided for Q and X must both be greater than zero.

Parameter	Value	Default
<i>VarName</i>	string	–
<i>{=}</i>	{= are is}	–
<i>Value</i>	“string”	–

Particle Diameter Distribution Type

Syntax

Particle Diameter Distribution Type *{=}* *{constant | lognormal | normal | rosin_rammler | uniform}*

Summary

Probability distribution type for particle diameters

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>DistributionType</i>	{constant lognormal normal rosin_rammler uniform}	CONSTANT

Particle Velocity Distribution Parameter

Syntax

Particle Velocity Distribution Parameter *VarName* {=} *Value*

Summary

Parameters for the probability distribution function for particle velocities.

Description

The number and type of parameters needed depends on the type of the probability distribution being used. The arguments here are the same as those used for the diameter distribution, consult that section for relevant details.

Parameter	Value	Default
<i>VarName</i>	string	–
{=}	{= are is}	–
<i>Value</i>	“string”	–

Particle Velocity Distribution Type

Syntax

Particle Velocity Distribution Type {=} {*constant* | *lognormal* | *normal* | *rosin_rammler* | *uniform*}

Summary

Probability distribution type for particle velocities

Parameter	Value	Default
{=}	{= are is}	–
<i>DistributionType</i>	{ <i>constant</i> <i>lognormal</i> <i>normal</i> <i>rosin_rammler</i> <i>uniform</i> }	CONSTANT

Particle Shape Parameter

Syntax

Particle Shape Parameter *VarName* {=} *Value* . .

Summary

Parameter for the particle filled shape.

Description

For each particle filled shape a particle definition and shape type must be specified. Also, particle diameter distribution type and its appropriate parameters must be specified.

A correctly defined ParticleFilledShape must also meet prescribed parameters. The number and type of parameters needed depends on the type of the particle filled shape being employed.

For all particle filled shapes, CENTER and NUMBER_OF_PARTICLES or PARTICLE_NUMBER_DENSITY parameters must be set. Additional parameters that must be set based on the shape are as follows:

SPHERE	req. params	RADIUS
CYLINDER	req. params	RADIUS, HEIGHT, NORMAL
BRICK	req. params	DIMENSIONS, V1, V2
CONE	req. params	RADIUS, HEIGHT, NORMAL
opt. param		*VELOCITY_NORMAL

Each parameter required should be entered in a separate command line.

Extra notes:

CYLINDER: For the cylinder, the center point is defined at the center of the circular base. The normal is a vector that points in the direction of the cylinders height.

BRICK: For the brick, the center point is defined at the center of the rectangular base, which lies in the plane of V1 and V2 orthogonal vectors. The normal, constructed from the cross product of V1 and V2, points in the direction of the bricks height. Parameter DIMENSIONS specifies width, length and height (in that order). The width and length is defined along V1 and V2 vectors respectively.

CONE: For the cone, the center point is defined at the center of the circular base. The normal is a vector that points in the direction of the cones height (tip).

*VELOCITY_NORMAL is a parameter required when a Particle Velocity Distribution is specified. If particle velocity distribution is not specified, the magnitude of velocity of all particles within a shape is set to 0.

Parameter	Value	Default
<i>VarName</i>	string	—
<i>{=}</i>	{= are is }	—
<i>Value</i>	real. . .	—

Particle Shape Type

Syntax

Particle Shape Type *{=}* *{brick | cone | cylinder | sphere}*

Summary

Particle filled shape type.

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>ShapeType</i>	{ brick cone cylinder sphere }	–

Porosity

Syntax

Porosity *{=}* *Porosity*

Summary

Set initial porosity for particle(s).

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>Porosity</i>	real	–

Track Particles For Tabular Output

Syntax

Track Particles For Tabular Output

Summary

All particles of this description are added to the tabular output

Description

Any particle matching this description will be output to the tabular data

Temperature

Syntax

Temperature *{=}* *Temperature*

Summary

Set initial temperature for particle(s).

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>Temperature</i>	real	–

X_Velocity

Syntax

X_Velocity *{=}* *X_velocity*

Summary

Set initial x velocity for particle(s).

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>X_velocity</i>	real	–

Y_Velocity

Syntax

Y_Velocity *{=}* *Y_velocity*

Summary

Set initial y velocity for particle(s).

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>Y_velocity</i>	real	–

Z_Velocity

Syntax

Z_Velocity *{=}* *Z_velocity*

Summary

Set initial z velocity for particle(s).

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>Z_velocity</i>	real	–

Particle Definition

Syntax

Particle Definition *{=}* *Definition*

Summary

Set particle definition to be used for this block

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>Definition</i>	string	–

Particle Filled Tree

Scope

Particle Region

Summary

Block that defines data for a particle filled tree.

```
begin Particle Filled Tree FilledTreeName
```

```
Particle Tree Parameter VarName {=} Value...
```

```
Particle Tree Type {=} {coniferous | deciduous | shrub}
```

```
begin Crown Info Name  
end
```

```
begin Trunk Info Name  
end
```

```
end Particle Filled Tree FilledTreeName
```

Line Commands

Particle Tree Parameter

Syntax

```
Particle Tree Parameter VarName {=} Value...
```

Summary

Parameter for the particle filled tree.

Description

ParticleFilledTree classes are added to support the following 3D particle filled tree shape categories: DECIDUOUS, CONIFEROUS, SHRUB.

DECIDUOUS tree includes oak, maple, redwood. These types of trees are represented by a sphere, cone and cylinder shapes. Sphere represents the crown, cylinder represents the trunk and the cone represents trunk narrowing inside the crown defined between the bottom surface of the crown (sphere) where its basal radius is that of the trunk and its center.

CONIFEROUS tree includes pine, cedar, spruce, fir. These types of trees are represented by two cones and cylinder shapes. The first cone represents the crown, cylinder represents the trunk and the second cone represents trunk narrowing inside the crown defined between the bottom surface of the crown (cone) where its basal radius is that of the trunk and its tip.

SHRUB, also referred to as a woody plant, includes a broom. They are represented by a sphere and cone shapes. Sphere represents the crown and the cone represents trunk narrowing inside the crown between the bottom surface of the crown (sphere) and its center. Note, a SHRUB is a special case of a DECIDUOUS tree in which the trunk outside the crown is missing.

A correctly defined ParticleFilledTree must meet prescribed parameters. The number and type of parameters needed depends on the type of the particle filled tree being employed.

For all particle filled trees, the following must be set:

- Particle Tree Type
- Particle Crown Definition
- Particle Trunk Definition

For all particle filled trees, the following parameters must also be set:

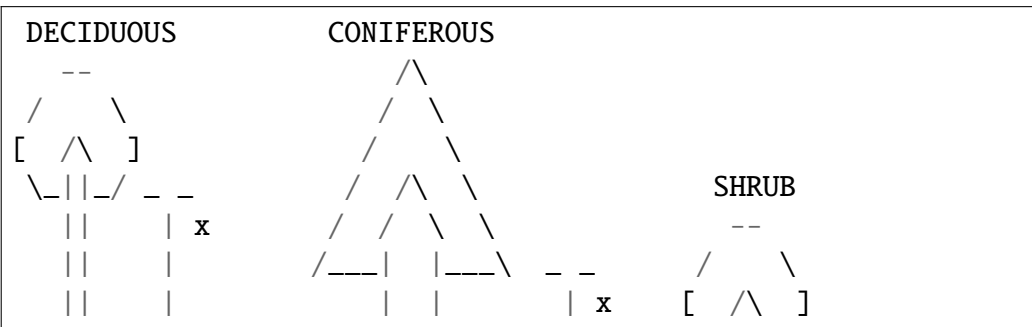
```
CENTER
NORMAL
CROWN_NUMBER_OF_PARTICLES
CROWN_VOLUME_FRACTION
CROWN_PARTICLE_DIAMETER_VARIATION
TRUNK_PARTICLE_DIAMETER_VARIATION
  TRUNK_NUMBER_OF_PARTICLES
  TRUNK_VOLUME_FRACTION
  CROWN_RADIUS
  TRUNK_RADIUS
```

Additional parameters that must be set based on the tree are as follows:

```
DECIDUOUS req. params CROWN_BASE_HEIGHT
CONIFEROUS req. params TREE_HEIGHT, CROWN_BASE_HEIGHT
```

Each parameter required should be entered in a separate command line.

Extra notes:



(continues on next page)

(continued from previous page)

|| _|_ |__| _|_ _|_|_/

CENTER **is** a center point defined at the base of the trunk.

NORMAL **is** a vector perpendicular to the base of the trunk.

TREE_HEIGHT **is** the distance defined between the ground **and**
↳the
 top of the tree crown.

CROWN_BASE_HEIGHT **is** the distance between the ground **and** the
↳base
 of the tree crown (**or** 'x' **in** the above schematic).

CROWN_VOLUME_FRACTION, **or** porosity **for** a porous medium, **is** a
↳percent

 number provided between 0 **and** 1 that describes
 constituent material based on the whole volume.
 For a spherical fuel element, this
↳value should

 never be larger than a sixth of pi.

CROWN_PARTICLE_DIAMETER_VARIATION **is** a percent number
↳provided between

 0 **and** 1 that describes particle
 diameter variation. Within the **range**,
 random particle diameters are selected,
 which are based on the number of
 particles **and** volume fraction params.

TRUNK_PARTICLE_DIAMETER_VARIATION (see above)

TRUNK_VOLUME_FRACTION (see above)

Parameter	Value	Default
<i>VarName</i>	string	—
<i>{=}</i>	{= are is}	—
<i>Value</i>	real. . .	—

Particle Tree Type

Syntax

Particle Tree Type *{=}* *{coniferous | deciduous | shrub}*

Summary

Particle filled tree type. Supported tree types include DECIDUOUS, CONIFEROUS and SHRUB.

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>TreeType</i>	{coniferous deciduous shrub}	–

Crown Info

Scope

Particle Filled Tree

Summary

Block that defines data for a tree crown.

begin Crown Info *Name*

Diameter *{=}* *Diameter*

Insertion Time *{=}* *InsertionTime*

Mass Represented *{=}* *Mass_represented*

Mass_Fraction *Species* *{=}* *y0*

Number Represented *{=}* *Num_represented*

Porosity *{=}* *Porosity*

Track Particles For Tabular Output

Temperature *{=}* *Temperature*

X_Velocity *{=}* *X_velocity*

Y_Velocity *{=}* *Y_velocity*

Z_Velocity *{=}* *Z_velocity*

Particle Definition *{=}* *Definition*

end Crown Info *Name*

Line Commands

Diameter

Syntax

Diameter *{=} Diameter*

Summary

Set initial diameter of particle(s)

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>Diameter</i>	real	–

Insertion Time

Syntax

Insertion Time *{=} InsertionTime*

Summary

Set time at which to insert particle; particle will be inserted at the first time step at or after this value.

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>InsertionTime</i>	real	–

Mass Represented

Syntax

Mass Represented *{=} Mass_represented*

Summary

Set mass of physical particles represented by each particle node.

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>Mass_represented</i>	real	–

Mass_Fraction

Syntax

Mass_Fraction *Species {=} y0*

Summary

Specification of mass fraction to be used for computing particle mass initial condition.

Parameter	Value	Default
<i>Species</i>	string	–
<i>{=}</i>	{= are is}	–
<i>y0</i>	real	–

Number Represented

Syntax

Number Represented *{=}* *Num_represented*

Summary

Set number of physical particles represented by each particle node.

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>Num_represented</i>	real	1

Porosity

Syntax

Porosity *{=}* *Porosity*

Summary

Set initial porosity for particle(s).

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>Porosity</i>	real	–

Track Particles For Tabular Output

Syntax

Track Particles For Tabular Output

Summary

All particles of this description are added to the tabular output

Description

Any particle matching this description will be output to the tabular data

Temperature

Syntax

Temperature *{=}* *Temperature*

Summary

Set initial temperature for particle(s).

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>Temperature</i>	real	–

X_Velocity

Syntax

X_Velocity *{=}* *X_velocity*

Summary

Set initial x velocity for particle(s).

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>X_velocity</i>	real	–

Y_Velocity

Syntax

Y_Velocity *{=}* *Y_velocity*

Summary

Set initial y velocity for particle(s).

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>Y_velocity</i>	real	–

Z_Velocity

Syntax

Z_Velocity *{=}* *Z_velocity*

Summary

Set initial z velocity for particle(s).

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>Z_velocity</i>	real	–

Particle Definition

Syntax

Particle Definition *{=}* *Definition*

Summary

Set particle definition to be used for this block

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>Definition</i>	string	–

Trunk Info

Scope

Particle Filled Tree

Summary

Block that defines data for a tree trunk.

begin Trunk Info *Name*

Diameter *{=}* *Diameter*

Insertion Time *{=}* *InsertionTime*

Mass Represented *{=}* *Mass_represented*

Mass_Fraction *Species* *{=}* *y0*

Number Represented *{=}* *Num_represented*

Porosity *{=}* *Porosity*

Track Particles For Tabular Output

Temperature *{=}* *Temperature*

X_Velocity *{=}* *X_velocity*

Y_Velocity *{=}* *Y_velocity*

Z_Velocity *{=}* *Z_velocity*

Particle Definition *{=}* *Definition*

end Trunk Info *Name*

Line Commands

Diameter

Syntax

Diameter *{=} Diameter*

Summary

Set initial diameter of particle(s)

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>Diameter</i>	real	–

Insertion Time

Syntax

Insertion Time *{=} InsertionTime*

Summary

Set time at which to insert particle; particle will be inserted at the first time step at or after this value.

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>InsertionTime</i>	real	–

Mass Represented

Syntax

Mass Represented *{=} Mass_represented*

Summary

Set mass of physical particles represented by each particle node.

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>Mass_represented</i>	real	–

Mass_Fraction

Syntax

Mass_Fraction *Species {=} y0*

Summary

Specification of mass fraction to be used for computing particle mass initial condition.

Parameter	Value	Default
<i>Species</i>	string	–
<i>{=}</i>	{= are is}	–
<i>y0</i>	real	–

Number Represented

Syntax

Number Represented *{=}* *Num_represented*

Summary

Set number of physical particles represented by each particle node.

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>Num_represented</i>	real	1

Porosity

Syntax

Porosity *{=}* *Porosity*

Summary

Set initial porosity for particle(s).

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>Porosity</i>	real	–

Track Particles For Tabular Output

Syntax

Track Particles For Tabular Output

Summary

All particles of this description are added to the tabular output

Description

Any particle matching this description will be output to the tabular data

Temperature

Syntax

Temperature *{=}* *Temperature*

Summary

Set initial temperature for particle(s).

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>Temperature</i>	real	–

X_Velocity

Syntax

X_Velocity *{=}* *X_velocity*

Summary

Set initial x velocity for particle(s).

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>X_velocity</i>	real	–

Y_Velocity

Syntax

Y_Velocity *{=}* *Y_velocity*

Summary

Set initial y velocity for particle(s).

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>Y_velocity</i>	real	–

Z_Velocity

Syntax

Z_Velocity *{=}* *Z_velocity*

Summary

Set initial z velocity for particle(s).

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>Z_velocity</i>	real	–

Particle Definition

Syntax

Particle Definition *{=}* *Definition*

Summary

Set particle definition to be used for this block

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>Definition</i>	string	–

Particle Inflow Boundary Condition On Surface

Scope

Particle Region

Summary

Defines a boundary condition for particle inflow on a named surface of the mesh.

begin Particle Inflow Boundary Condition On Surface *Surfacename*

Diameter *{=}* *Diameter*

Insertion Time *{=}* *InsertionTime*

Mass Represented *{=}* *Mass_represented*

Mass_Fraction *Species* *{=}* *y0*

Number Represented *{=}* *Num_represented*

Porosity *{=}* *Porosity*

Track Particles For Tabular Output

Temperature *{=}* *Temperature*

X_Velocity *{=}* *X_velocity*

Y_Velocity *{=}* *Y_velocity*

Z_Velocity *{=}* *Z_velocity*

Particle Definition *{=}* *Definition*

end Particle Inflow Boundary Condition On Surface *Surfacename*

Line Commands

Diameter

Syntax

Diameter {=} *Diameter*

Summary

Set initial diameter of particle(s)

Parameter	Value	Default
{=}	{ = are is }	–
<i>Diameter</i>	real	–

Insertion Time

Syntax

Insertion Time {=} *InsertionTime*

Summary

Set time at which to insert particle; particle will be inserted at the first time step at or after this value.

Parameter	Value	Default
{=}	{ = are is }	–
<i>InsertionTime</i>	real	–

Mass Represented

Syntax

Mass Represented {=} *Mass_represented*

Summary

Set mass of physical particles represented by each particle node.

Parameter	Value	Default
{=}	{ = are is }	–
<i>Mass_represented</i>	real	–

Mass_Fraction

Syntax

Mass_Fraction *Species* {=} *y0*

Summary

Specification of mass fraction to be used for computing particle mass initial condition.

Parameter	Value	Default
<i>Species</i>	string	–
<i>{=}</i>	{= are is}	–
<i>y0</i>	real	–

Number Represented

Syntax

Number Represented *{=}* *Num_represented*

Summary

Set number of physical particles represented by each particle node.

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>Num_represented</i>	real	1

Porosity

Syntax

Porosity *{=}* *Porosity*

Summary

Set initial porosity for particle(s).

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>Porosity</i>	real	–

Track Particles For Tabular Output

Syntax

Track Particles For Tabular Output

Summary

All particles of this description are added to the tabular output

Description

Any particle matching this description will be output to the tabular data

Temperature

Syntax

Temperature *{=}* *Temperature*

Summary

Set initial temperature for particle(s).

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>Temperature</i>	real	–

X_Velocity

Syntax

X_Velocity *{=}* *X_velocity*

Summary

Set initial x velocity for particle(s).

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>X_velocity</i>	real	–

Y_Velocity

Syntax

Y_Velocity *{=}* *Y_velocity*

Summary

Set initial y velocity for particle(s).

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>Y_velocity</i>	real	–

Z_Velocity

Syntax

Z_Velocity *{=}* *Z_velocity*

Summary

Set initial z velocity for particle(s).

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>Z_velocity</i>	real	–

Particle Definition

Syntax

Particle Definition *{=}* *Definition*

Summary

Set particle definition to be used for this block

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>Definition</i>	string	–

Particle Interface

Scope

Particle Region

Summary

Block that defines a specific particle-fluid interaction (e.g. reaction or evaporation)

begin Particle Interface *InterfaceName*

```
{critical_porosity | critical_temperature | flame_temperature_  
↪for_transport | fuel_molecular_weight | heat_of_combustion | mass_  
↪diffusivity | prandtl_number | reference_heat_of_vaporization |  
↪reference_pressure | reference_temperature | schmidt_number | thermal_  
↪diffusivity | universal_gas_constant} {=} Value
```

Particle Species *SpeciesName* *StoichCoef*

Reaction Rate Parameter *Name* {=} *RateParam*

begin Particle Evaporation *EvaporationName*
end

begin Particle Reaction *ReactionName*
end

end Particle Interface *InterfaceName*

Line Commands

ParticleInterfaceParameter

Syntax

ParticleInterfaceParameter {critical_porosity | critical_temperature | flame_temperature_for_transport | fuel_molecular_weight | heat_of_combustion | mass_diffusivity | prandtl_number | reference_heat_of_vaporization | reference_pressure | reference_temperature | schmidt_number | thermal_diffusivity | universal_gas_constant} {=} Value

Summary

Constant value for the specified parameter (in consistent units).

Parameter	Value	Default
ParticleInterfaceParameter	{critical_porosity critical_temperature flame_temperature_for_transport fuel_molecular_weight heat_of_combustion mass_diffusivity prandtl_number reference_heat_of_vaporization reference_pressure reference_temperature schmidt_number thermal_diffusivity universal_gas_constant}	—
{=}	{ = are is }	—
Value	real	—

Particle Species

Syntax

Particle Species SpeciesName StoichCoef

Summary

Particle species name and stoichiometric coefficient for this interface.

Parameter	Value	Default
SpeciesName	string	—
StoichCoef	real	—

Reaction Rate Parameter

Syntax

Reaction Rate Parameter Name {=} RateParam

Summary

Specification of reaction rate parameters, E, A, latent heat.

Description

Specification of reaction rate parameters, E, A, latent heat. Note that activation energies should be in per-mole units (e.g. kJ/mol or erg/mol, where energy units are consistent with the rest of the input file)

Parameter	Value	Default
<i>Name</i>	string	–
<i>{=}</i>	{= are is}	–
<i>RateParam</i>	real	–

Particle Evaporation

Scope

Particle Interface

Summary

Sub-block that defines a specific particle evaporation. Maximum one per interface.

begin Particle Evaporation *EvaporationName*

```
{critical_porosity | critical_temperature | flame_temperature_  
→for_transport | fuel_molecular_weight | heat_of_combustion | mass_  
→diffusivity | prandtl_number | reference_heat_of_vaporization |_  
→reference_pressure | reference_temperature | schmidt_number | thermal_  
→diffusivity | universal_gas_constant} {=} Value
```

Gas Species *SpeciesName* *StoichCoef*

end Particle Evaporation *EvaporationName*

Line Commands

Particleinterfaceparameter

Syntax

```
Particleinterfaceparameter {critical_porosity | critical_temperature |  
flame_temperature_for_transport | fuel_molecular_weight |  
heat_of_combustion | mass_diffusivity | prandtl_number |  
reference_heat_of_vaporization | reference_pressure | reference_temperature |  
schmidt_number | thermal_diffusivity | universal_gas_constant} {=} Value
```

Summary

Constant value for the specified parameter (in consistent units).

Parameter	Value	Default
<i>ParticleInterfaceParameter</i>	{critical_porosity critical_temperature flame_temperature_for_transport fuel_molecular_weight heat_of_combustion mass_diffusivity prandtl_number reference_heat_of_vaporization reference_pressure reference_temperature schmidt_number thermal_diffusivity universal_gas_constant}	–
<i>{=}</i>	{= are is}	–
<i>Value</i>	real	–

Gas Species

Syntax

Gas Species *SpeciesName* *StoichCoef*

Summary

GAS species name and stoichiometric coefficient for this interface.

Parameter	Value	Default
<i>SpeciesName</i>	string	–
<i>StoichCoef</i>	real	–

Particle Reaction

Scope

Particle Interface

Summary

Sub-block that defines a specific particle reaction.

begin Particle Reaction *ReactionName*

```

    {critical_porosity | critical_temperature | flame_temperature_
    ↳for_transport | fuel_molecular_weight | heat_of_combustion | mass_
    ↳diffusivity | prandtl_number | reference_heat_of_vaporization |
    ↳reference_pressure | reference_temperature | schmidt_number | thermal_
    ↳diffusivity | universal_gas_constant} {=} Value

```

Gas Species *SpeciesName* *StoichCoef*

Oxidizer {=} *SpeciesName*

Water {=} *SpeciesName*

end Particle Reaction *ReactionName*

Line Commands

Particleinterfaceparameter

Syntax

Particleinterfaceparameter {critical_porosity | critical_temperature |
flame_temperature_for_transport | fuel_molecular_weight |
heat_of_combustion | mass_diffusivity | prandtl_number |
reference_heat_of_vaporization | reference_pressure | reference_temperature |
schmidt_number | thermal_diffusivity | universal_gas_constant} {=} *Value*

Summary

Constant value for the specified parameter (in consistent units).

Pa- rame- ter	Value	De- fault
<i>ParticleInterfaceParameter</i>	{critical_porosity critical_temperature flame_temperature_for_transport fuel_molecular_weight heat_of_combustion mass_diffusivity prandtl_number reference_heat_of_vaporization reference_pressure reference_temperature schmidt_number thermal_diffusivity universal_gas_constant}	—
{=}	{ = are is }	—
<i>Value</i>	real	—

Gas Species

Syntax

Gas Species *SpeciesName* *StoichCoef*

Summary

GAS species name and stoichiometric coefficient for this interface.

Parameter	Value	Default
<i>SpeciesName</i>	string	—
<i>StoichCoef</i>	real	—

Oxidizer

Syntax

Oxidizer *{=}* *SpeciesName*

Summary

Oxidizer species name for this interface (may be more than one).

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>SpeciesName</i>	string	–

Water

Syntax

Water *{=}* *SpeciesName*

Summary

Water species name for this interface.

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>SpeciesName</i>	string	–

Particle Material

Scope

Particle Region

Summary

Block that defines material property data for a particle component

begin Particle Material *MaterialName*

```
{absorptivity | density | emission_multiplier | emissivity | enthalpy_  
→of_fusion | film_prandtl_number | freezing_temperature | molecular_  
→weight | specific_heat | surface_tension | thermal_conductivity |  
→viscosity} {=} EvalFunction...
```

end Particle Material *MaterialName*

Line Commands

Particlematerialproperty

Syntax

Particlematerialproperty {*absorptivity* | *density* | *emission_multiplier* | *emissivity* | *enthalpy_of_fusion* | *film_prandtl_number* | *freezing_temperature* | *molecular_weight* | *specific_heat* | *surface_tension* | *thermal_conductivity* | *viscosity*} {=} *EvalFunction*. . .

Summary

Constant/temperature dependent value for the specified property (in consistent units). The string entered can be a constant, or a function of time which will be evaluated using the stk expression evaluator. You must use ‘T’ as the temperature variable in this function.

Parameter	Value	Default
<i>Particle-MaterialProperty</i>	{ <i>absorptivity</i> <i>density</i> <i>emission_multiplier</i> <i>emissivity</i> <i>enthalpy_of_fusion</i> <i>film_prandtl_number</i> <i>freezing_temperature</i> <i>molecular_weight</i> <i>specific_heat</i> <i>surface_tension</i> <i>thermal_conductivity</i> <i>viscosity</i> }	–
{=}	{= are is}	–
<i>Eval-Function</i>	“string”. . .	0.0

Particle Open Boundary Condition On Surface

Scope

Particle Region

Summary

Defines an open boundary condition and its interaction with particles.

begin Particle Open Boundary Condition On Surface *Surfacename*

end Particle Open Boundary Condition On Surface *Surfacename*

Particle Spray

Scope

Particle Region

Summary

Block that defines data for a nozzle inflow.

begin Particle Spray *SprayName*

Arc_Angle *{=}* *ArcAngle*

Arc_Ref *{=}* *nRef1 nRef2 nRef3*

Center *{=}* *Xcenter1 Xcenter2 Xcenter3*

Diameter Number Represented Vector *{=}* *diamnumrepvec...*

Diameter *{=}* *Diameter*

Insertion Time *{=}* *InsertionTime*

Length Vector *{=}* *LengthVector1 LengthVector2 LengthVector3*

Length *{=}* *Length*

Mass Represented *{=}* *Mass_represented*

Mass_Flow_Rate *{=}* *Mdot*

Mass_Fraction *Species* *{=}* *y0*

Normal Vector *{=}* *NormalVector1 NormalVector2 NormalVector3*

Nozzle Radius *{=}* *Radius*

Number Representative Points *{=}* *NumRepPoints*

Number Represented Vector *{=}* *numrepvec...*

Number Represented *{=}* *Num_represented*

Particle Average_Diameter *{=}* *Velocity*

Particle Average_Velocity *{=}* *Velocity*

```

Particle Diameter Distribution Parameter VarName {=} Value

Particle Diameter Distribution Type {=} {constant | lognormal | normal_
↪| rosin_rammler | uniform}

Particle Log File {=} FieldName

Particle Velocity Distribution Parameter VarName {=} Value

Particle Velocity Distribution Type {=} {constant | lognormal | normal_
↪| rosin_rammler | uniform}

Porosity {=} Porosity

Spray_Angle {=} Angle

Spray_Angle_End {=} Angle

Spray_Angle_Start {=} Angle

Track Particles For Tabular Output

Temperature {=} Temperature

Width {=} Width

X_Velocity {=} X_velocity

Y_Velocity {=} Y_velocity

Z_Velocity {=} Z_velocity

Diameter Cutoff High {=} diamcutoffhigh

Diameter Cutoff Low {=} diamcutofflow

Particle Definition {=} Definition

end Particle Spray SprayName

```

Line Commands

Arc_Angle

Syntax

Arc_Angle {=} *ArcAngle*

Summary

Angle (in degrees) for circular wedge shaped spray

Parameter	Value	Default
{=}	{ = are is }	–
<i>ArcAngle</i>	real	–

Arc_Ref

Syntax

Arc_Ref {=} *nRef1 nRef2 nRef3*

Summary

Reference vector within spray plane to measure angle from for circular wedge shaped spray

Parameter	Value	Default
{=}	{ = are is }	–
<i>nRef</i>	real1 real2 real3	–

Center

Syntax

Center {=} *Xcenter1 Xcenter2 Xcenter3*

Summary

(x,y,z) coordinates of center of spray nozzle plane

Parameter	Value	Default
{=}	{ = are is }	–
<i>Xcenter</i>	real1 real2 real3	–

Diameter Number Represented Vector

Syntax

Diameter Number Represented Vector {=} *diamnumrepvec...*

Summary

Vector of diameters for function relating diameter of particle to number represented for fine control over number represented distributions

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>diamnumrepvec</i>	real. . .	–

Diameter

Syntax

Diameter *{=}* *Diameter*

Summary

Set initial diameter of particle(s)

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>Diameter</i>	real	–

Insertion Time

Syntax

Insertion Time *{=}* *InsertionTime*

Summary

Set time at which to insert particle; particle will be inserted at the first time step at or after this value.

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>InsertionTime</i>	real	–

Length Vector

Syntax

Length Vector *{=}* *LengthVector1* *LengthVector2* *LengthVector3*

Summary

Components (lx,ly,lz) of vector in the plane of the spray nozzle.

Description

This vector is necessary for a rectangular spray nozzle to define the direction in which the LENGTH parameter is measured. This vector must be orthogonal to the spray normal direction. A third vector defining the direction for the WIDTH parameter is computed internally to be orthogonal to both the NORMAL and LENGTH vectors.

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>LengthVector</i>	real1 real2 real3	–

Length

Syntax

Length *{=}* *Length*

Summary

Length of rectangular spray nozzle.

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>Length</i>	real	0.0

Mass Represented

Syntax

Mass Represented *{=}* *Mass_represented*

Summary

Set mass of physical particles represented by each particle node.

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>Mass_represented</i>	real	–

Mass_Flow_Rate

Syntax

Mass_Flow_Rate *{=}* *Mdot*

Summary

Mass flow rate of particles through the spray nozzle. The string entered can be a constant, or a function of time which will be evaluated using the stk expression evaluator. You must use ‘t’ as the time variable in this function.

If the function contains commas or spaces it should be enclosed in quotes (e.g. “max(t,1)”)

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>Mdot</i>	“string”	0.0

Mass_Fraction

Syntax

Mass_Fraction *Species* {=} *y0*

Summary

Specification of mass fraction to be used for computing particle mass initial condition.

Parameter	Value	Default
<i>Species</i>	string	–
{=}	{= are is}	–
<i>y0</i>	real	–

Normal Vector**Syntax**

Normal Vector {=} *NormalVector1 NormalVector2 NormalVector3*

Summary

Components (nx,ny,nz) of vector normal to spray nozzle plane

Parameter	Value	Default
{=}	{= are is}	–
<i>NormalVector</i>	real1 real2 real3	–

Nozzle Radius**Syntax**

Nozzle Radius {=} *Radius*

Summary

Radius of circular spray nozzle

Parameter	Value	Default
{=}	{= are is}	–
<i>Radius</i>	real	–

Number Representative Points**Syntax**

Number Representative Points {=} *NumRepPoints*

Summary

Number of possible insertion points used to represent this spray.

Parameter	Value	Default
{=}	{= are is}	–
<i>NumRepPoints</i>	integer	1000

Number Represented Vector

Syntax

Number Represented Vector {=} *numrepvec* . .

Summary

Vector of number represented for function relating diameter of particle to number represented for fine control over number represented distribution

Parameter	Value	Default
{=}	{= are is}	–
<i>numrepvec</i>	real. . .	–

Number Represented

Syntax

Number Represented {=} *Num_represented*

Summary

Set number of physical particles represented by each particle node.

Parameter	Value	Default
{=}	{= are is}	–
<i>Num_represented</i>	real	1

Particle Average_Diameter

Syntax

Particle Average_Diameter {=} *Velocity*

Summary

DEPRECATED: Mean diameter of particles entering through spray nozzle.

Parameter	Value	Default
{=}	{= are is}	–
<i>Velocity</i>	real	–

Particle Average_Velocity

Syntax

Particle Average_Velocity {=} *Velocity*

Summary

DEPRECATED: Mean velocity of particles entering through spray nozzle.

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>Velocity</i>	real	0.0

Particle Diameter Distribution Parameter

Syntax

Particle Diameter Distribution Parameter *VarName* *{=}* *Value*

Summary

Parameter for the probability distribution function for particle diameters.

Description

The number and type of parameters needed depends on the type of the probability distribution being used. The available distributions are described below.

CONSTANT

Required Parameters: VALUE

Always returns a constant value

UNIFORM

Required Parameters: MIN, MAX

Returns a random value uniformly distributed between the provided min and max values.

NORMAL

Required Parameters: MEAN, STDEV

Optional Parameters: MIN, MAX

Returns a random value normally distributed using the provided mean and standard deviation. If optional min or max values are provided then the PDF will be clipped to zero outside those bounds. The PDF for this distribution is:

$$P(x) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2\right)$$

LOGNORMAL

Required Parameters: MEAN and STDEV or LOGMEAN and LOGSTDEV or MEDIAN and SCATTER

Optional Parameters: MIN, MAX

Returns a random value with a log-normal distribution. If optional min or max values are provided then the PDF will be clipped to zero outside those bounds.

The PDF for this distribution is:

$$P(x) = \frac{1}{x\sigma\sqrt{2\pi}} \exp\left(-\frac{1}{2}\left(\frac{\ln(x) - \mu}{\sigma}\right)^2\right)$$

There are several ways to specify the parameters needed for this distribution (μ and σ). You can provide them directly (using LOGMEAN and LOGSTDEV), or you can provide non-log-scaled parameters and have them calculated.

Keep in mind that unlike a normal distribution, the peak of a lognormal distribution (its mode) is not equal to its mean or median. For a log-normal distribution the mean is

$$\text{Mean} = \mu_X = \exp\left(\mu + \frac{\sigma^2}{2}\right)$$

the median is

$$\text{Median} = \exp(\mu)$$

and the mode (peak of the distribution) is

$$\text{Mode} = \exp\left(\mu - \sigma^2\right)$$

If you provide a mean (μ_X , MEAN) and standard deviation (σ_X , STDEV), then the log-mean (μ) and log-standard deviation (σ) will be calculated using

$$\mu = \ln\left(\frac{\mu_X^2}{\sqrt{\mu_X^2 + \sigma_X^2}}\right)$$

$$\sigma = \sqrt{\ln\left(1 + \frac{\sigma_X^2}{\mu_X^2}\right)}$$

Alternately, if you provide a median (μ^* , MEDIAN) and scatter (σ^* , SCATTER), then log-mean (μ) and log-standard deviation (σ) will be calculated using

$$\mu = \ln(\mu^*)$$

$$\sigma = \ln(\sigma^*)$$

The median and scatter physical interpretation is best understood by the fact that about two thirds of the distribution will lie between μ^*/σ^* and $\mu^*\sigma^*$. Similarly, 95 percent of the distribution will lie between μ^*/σ^{*2} and $\mu^*\sigma^{*2}$. As such, the value for σ^* (SCATTER) must be a value greater than one since it is a multiplier of the median.

ROSIN_RAMMLER

Required Parameters: X, Q

Optional Parameters: MIN, MAX

Returns a random value with a Rosin-Rammler (or Weibull) distribution. If optional min or max values are provided then the PDF will be clipped to zero outside those bounds. The PDF for this distribution is:

$$P(x) = \frac{Q}{X} \left(\frac{x}{X} \right)^{Q-1} \exp \left(- \left(\frac{x}{X} \right)^Q \right)$$

The values provided for Q and X must both be greater than zero.

Parameter	Value	Default
<i>VarName</i>	string	–
<i>{=}</i>	{= are is}	–
<i>Value</i>	“string”	–

Particle Diameter Distribution Type

Syntax

Particle Diameter Distribution Type *{=}* *{constant | lognormal | normal | rosin_rammler | uniform}*

Summary

Probability distribution type for particle diameters

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>DistributionType</i>	{constant lognormal normal rosin_rammler uniform}	CONSTANT

Particle Log File

Syntax

Particle Log File *{=}* *FieldName*

Summary

Activates output to a file of a log of inserted particle nodes, including time injected, number represented, particle diameter, and particle mass

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>FieldName</i>	string	–

Particle Velocity Distribution Parameter

Syntax

Particle Velocity Distribution Parameter *VarName* {=} *Value*

Summary

Parameters for the probability distribution function for particle velocities.

Description

The number and type of parameters needed depends on the type of the probability distribution being used. The arguments here are the same as those used for the diameter distribution, consult that section for relevant details.

Parameter	Value	Default
<i>VarName</i>	string	–
{=}	{= are is}	–
<i>Value</i>	“string”	–

Particle Velocity Distribution Type

Syntax

Particle Velocity Distribution Type {=} {*constant* | *lognormal* | *normal* | *rosin_rammler* | *uniform*}

Summary

Probability distribution type for particle velocities

Parameter	Value	Default
{=}	{= are is}	–
<i>DistributionType</i>	{constant lognormal normal rosin_rammler uniform}	CONSTANT

Porosity

Syntax

Porosity {=} *Porosity*

Summary

Set initial porosity for particle(s).

Parameter	Value	Default
{=}	{= are is}	–
<i>Porosity</i>	real	–

Spray_Angle

Syntax

Spray_Angle {=} *Angle*

Summary

Maximum spray angle (degrees)

Parameter	Value	Default
<i>{=}</i>	{= are is }	–
<i>Angle</i>	real	–

Spray_Angle_End

Syntax

Spray_Angle_End *{=}* *Angle*

Summary

Maximum spray angle (degrees)

Parameter	Value	Default
<i>{=}</i>	{= are is }	–
<i>Angle</i>	real	0.0

Spray_Angle_Start

Syntax

Spray_Angle_Start *{=}* *Angle*

Summary

Minimum spray angle (degrees); if >0 results in a hollow cone spray.

Parameter	Value	Default
<i>{=}</i>	{= are is }	–
<i>Angle</i>	real	0.0

Track Particles For Tabular Output

Syntax

Track Particles For Tabular Output

Summary

All particles of this description are added to the tabular output

Description

Any particle matching this description will be output to the tabular data

Temperature

Syntax

Temperature *{=}* *Temperature*

Summary

Set initial temperature for particle(s).

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>Temperature</i>	real	–

Width

Syntax

Width *{=}* *Width*

Summary

Width of rectangular spray nozzle.

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>Width</i>	real	0.0

X_Velocity

Syntax

X_Velocity *{=}* *X_velocity*

Summary

Set initial x velocity for particle(s).

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>X_velocity</i>	real	–

Y_Velocity

Syntax

Y_Velocity *{=}* *Y_velocity*

Summary

Set initial y velocity for particle(s).

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>Y_velocity</i>	real	–

Z_Velocity

Syntax

Z_Velocity {=} *Z_velocity*

Summary

Set initial z velocity for particle(s).

Parameter	Value	Default
{=}	{= are is}	–
<i>Z_velocity</i>	real	–

Diameter Cutoff High**Syntax**

Diameter Cutoff High {=} *diamcutoffhigh*

Summary

Upper Cutoff for particle Diameters.

This command is DEPRECATED. Use the MAX parameter for the diameter distribution instead.

Parameter	Value	Default
{=}	{= are is}	–
<i>diamcutoffhigh</i>	real	1000000.0

Diameter Cutoff Low**Syntax**

Diameter Cutoff Low {=} *diamcutofflow*

Summary

Lower Cutoff for particle Diameters.

This command is DEPRECATED. Use the MIN parameter for the diameter distribution instead.

Parameter	Value	Default
{=}	{= are is}	–
<i>diamcutofflow</i>	real	0.0

Particle Definition**Syntax**

Particle Definition {=} *Definition*

Summary

Set particle definition to be used for this block

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>Definition</i>	string	–

Particle Tabular Output

Scope

Particle Region

Summary

Block that creates and defines options for tabular output of particle data.

Description

Particle data is output in tabulated columns, with one file per particle containing the time history of that particle, and one variable per column. Only particles inserted from an insertion source marked with the TRACK PARTICLES FOR TABULAR OUTPUT command line are tracked. IDs are assigned automatically using a hash code.

begin Particle Tabular Output *Name*

Filename Prefix *{=}* *FilenamePrefix*

Output Variable *VariableName* [Component *Comp*]

end Particle Tabular Output *Name*

Line Commands

Filename Prefix

Syntax

Filename Prefix *{=}* *FilenamePrefix*

Summary

Set filename prefix for output files.

Description

File names will be created for data output files by concatenating the prefix with the particle ID number, followed by the .dat extension, e.g.
particle_data_1.dat.

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>FilenamePrefix</i>	string	–

Output Variable

Syntax

Output Variable *VariableName* [Component *Comp*]

Summary

Designate a nodal particle variable for output.

Parameter	Value	Default
<i>VariableName</i>	string	–

Particle Wall Boundary Condition On Surface

Scope

Particle Region

Summary

Defines a wall boundary condition and its interaction with particles.

begin Particle Wall Boundary Condition On Surface *Surfacename*

Adhesion Model *{=}* *particle_adhesion_model...*

Particle Surface Interaction Type *{=}* *{combined | pass_through | ↪rebound | shatter | stick | undefined}*

Reflection Model *{=}* *particle_reflection_model...*

Particle_K_Pf_Crit *{=}* *particle_K_PF_crit*

Particle_Contact_Angle *{=}* *particle_contact_angle*

Particle_Maximum_Number_Fingers *{=}* *particle_maximum_number_fingers*

Particle_Random_Graze_Angle_Maximum *{=}* *particle_random_graze_angle_↪maximum*

Particle_Random_Graze_Angle_Minimum *{=}* *particle_random_graze_angle_↪minimum*

Particle_Splash_Parameter_A *{=}* *particle_splash_parameter_A*

Particle_Splash_Parameter_B *{=}* *particle_splash_parameter_B*

end Particle Wall Boundary Condition On Surface *Surfacename*

Line Commands

Adhesion Model

Syntax

Adhesion Model *{=}* *particle_adhesion_model...*

Summary

Specifies the adhesion model to use with the combined surface interaction model. Options are “RESUSPEND_WICHNER”, “STICK”, or “NONE”.

Each of these options is followed by a list of arguments specific to that model. For details on the available options, refer to the particle section in the Math Models chapter of the user manual.

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>particle_adhesion_model</i>	string...	–

Particle Surface Interaction Type

Syntax

Particle Surface Interaction Type *{=}* *{combined | pass_through | rebound | shatter | stick | undefined}*

Summary

Type of interaction between particles and surface. Use “COMBINED” to specify separate reflection and adhesion models. All other options are DEPRECATED and are internally replaced with the equivalent “COMBINED” model.

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>SurfaceInteraction-Type</i>	{ combined pass_through rebound shatter stick undefined }	–

Reflection Model

Syntax

Reflection Model *{=}* *particle_reflection_model...*

Summary

Specifies the reflection model to use with the combined surface interaction model. Options are “SHATTER_BROWN”, “REBOUND”, “REBOUND_YOON”, or “NONE”.

Each of these options is followed by a list of arguments specific to that model. For example, the shatter model would be “REFLECTION MODEL = SHATTER_BROWN K_crit = 58 contact_angle = 45”.

For details on the other options, refer to the particle section in the Math Models chapter of the user manual.

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>particle_reflection_model</i>	string. . .	–

Particle_K_Pf_Crit

Syntax

Particle_K_Pf_Crit *{=}* *particle_K_PF_crit*

Summary

THIS INPUT FORMAT IS DEPRECATED. USE THE “REFLECTION MODEL” and “ADHESION MODEL” FORMAT.

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>particle_K_PF_crit</i>	real	–

Particle_Contact_Angle

Syntax

Particle_Contact_Angle *{=}* *particle_contact_angle*

Summary

THIS INPUT FORMAT IS DEPRECATED. USE THE “REFLECTION MODEL” and “ADHESION MODEL” FORMAT.

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>particle_contact_angle</i>	real	–

Particle_Maximum_Number_Fingers

Syntax

Particle_Maximum_Number_Fingers *{=}* *particle_maximum_number_fingers*

Summary

THIS INPUT FORMAT IS DEPRECATED. USE THE “REFLECTION MODEL” and “ADHESION MODEL” FORMAT.

Parameter	Value	Default
<i>{=}</i>	{= are is }	–
<i>particle_maximum_number_fingers</i>	integer	–

Particle_Random_Graze_Angle_Maximum

Syntax

Particle_Random_Graze_Angle_Maximum *{=}*
particle_random_graze_angle_maximum

Summary

THIS INPUT FORMAT IS DEPRECATED. USE THE “REFLECTION MODEL” and “ADHESION MODEL” FORMAT.

Parameter	Value	Default
<i>{=}</i>	{= are is }	–
<i>particle_random_graze_angle_maximum</i>	real	–

Particle_Random_Graze_Angle_Minimum

Syntax

Particle_Random_Graze_Angle_Minimum *{=}*
particle_random_graze_angle_minimum

Summary

THIS INPUT FORMAT IS DEPRECATED. USE THE “REFLECTION MODEL” and “ADHESION MODEL” FORMAT.

Parameter	Value	Default
<i>{=}</i>	{= are is }	–
<i>particle_random_graze_angle_minimum</i>	real	–

Particle_Splash_Parameter_A

Syntax

Particle_Splash_Parameter_A *{=}* *particle_splash_parameter_A*

Summary

THIS INPUT FORMAT IS DEPRECATED. USE THE “REFLECTION MODEL” and “ADHESION MODEL” FORMAT.

Parameter	Value	Default
<i>{=}</i>	{= are is }	–
<i>particle_splash_parameter_A</i>	real	–

Particle_Splash_Parameter_B

Syntax

Particle_Splash_Parameter_B {=} *particle_splash_parameter_B*

Summary

THIS INPUT FORMAT IS DEPRECATED. USE THE “REFLECTION MODEL” and “ADHESION MODEL” FORMAT.

Parameter	Value	Default
{=}	{= are is}	–
<i>particle_splash_parameter_B</i>	real	–

Restart Data

Scope

Average Region, Fuego Region, Input_Output Region, Particle Region

Summary

Describes the data required to output and input restart data for the enclosing region.

begin Restart Data *Label*

Additional Steps {=} *List_of_steps...*

Additional Times {=} *List_of_times...*

At Step *n* {increment | interval} {=} *m*

At Time *Dt1* {increment | interval} {=} *Dt2*

At Wall Time *Dt1* {increment | interval} {=} *Dt2*

Component Separator Character {=} *Separator*

Cycle Count {=} *Count*

Database Name {=} *StreamName*

Database Type {=} {catalyst | catalyst_exodus | cgns | dof | dof_
→exodus | exodus | exodusii | exonull | generated | genesis | null |_
→parallel_exodus | textmesh}

Debug Dump

Decomposition Method {=} {block | cyclic | external | geom_kway | hsfc_
→| kway | kway_geom | linear | map | metis_sfc | random | rcb | rib |_
→variable}

Exists {=} {abort | add_suffix | append | overwrite}

File Cycle Count {=} Count

Input Database Name {=} StreamName

Optional

Output Database Name {=} StreamName

Output On Signal {=} {sigabrt | sigalrm | sigfpe | sighup | sigill |_
→sigint | sigkill | sigpipe | sigquit | sigsegv | sigterm | sigusr1 |_
→sigusr2}

Output Restart State {=} {off | on}

Overlay Count {=} Count

Overwrite {=} {false | no | off | on | true | yes}

Property *PropertyName* {=} *PropertyValue*

Restart {=} {auto}

Restart Time {=} Time

Start Time {=} Start_time

Synchronize Output

Shift To Start Time

Termination Time {=} Final_time

Timestep Adjustment Interval {=} Nsteps

Use Dynamic Topology Io

Use Output Scheduler *Timer_name*

end Restart Data *Label*

Line Commands

Additional Steps

Syntax

Additional Steps *{=}* *List_of_steps* . .

Summary

Additional simulation steps when output should occur.

Parameter	Value	Default
<i>{=}</i>	{= are is}	—
<i>List_of_steps</i>	integer. . .	—

Additional Times

Syntax

Additional Times *{=}* *List_of_times* . .

Summary

Additional simulation times when output should occur.

Parameter	Value	Default
<i>{=}</i>	{= are is}	—
<i>List_of_times</i>	real. . .	—

At Step

Syntax

At Step *n* *{increment | interval}* *{=}* *m*

Summary

Specify an output interval in terms of the internal iteration step count. The first step specifies the step count at the beginning of this interval and the second step specifies the output frequency to be used within this interval.

Parameter	Value	Default
<i>n</i>	integer	—
<i>Option</i>	{increment interval}	—
<i>{=}</i>	{= are is}	—
<i>m</i>	integer	—

At Time

Syntax

At Time *Dt1* {*increment* | *interval*} {=} *Dt2*

Summary

Specify an output interval in terms of the internal simulation time. The first time specifies the time at the beginning of this time interval and the second time specifies the output frequency to be used within this interval.

Parameter	Value	Default
<i>Dt1</i>	real	—
<i>Option</i>	{ <i>increment</i> <i>interval</i> }	—
<i>{=}</i>	{= are is}	—
<i>Dt2</i>	real	—

At Wall Time

Syntax

At Wall Time *Dt1* {*increment* | *interval*} {=} *Dt2*

Summary

Write a restart file at a specific wall time since the start of the run. Time string format allows s, m, h, d for seconds, minutes, hours, days

Parameter	Value	Default
<i>Dt1</i>	string	—
<i>Option</i>	{ <i>increment</i> <i>interval</i> }	—
<i>{=}</i>	{= are is}	—
<i>Dt2</i>	string	—

Component Separator Character

Syntax

Component Separator Character {=} *Separator*

Summary

The separator is the single character used to separate the output variable basename (e.g. “stress”) from the suffices (e.g. “xx”, “yy”) when displaying the names of the individual variable components. For example, the default separator is “_”, which results in names similar to “stress_xx”, “stress_yy”, ... “stress_zx”. To eliminate the separator, specify an empty string (“”) or NONE.

Parameter	Value	Default
<i>{=}</i>	{= is}	—
<i>Separator</i>	string	—

Cycle Count

Syntax

Cycle Count *{=}* *Count*

Summary

Specify the number of restart steps which will be written to the restart database before previously written steps are overwritten. For example, if the cycle count is 5 and restart is written every 0.1 seconds, the restart system will write 0.1, 0.2, 0.3, 0.4, 0.5 to the database. It will then overwrite the first step with data from time 0.6, the second with time 0.7. At time 0.8, the database would contain data at times 0.6, 0.7, 0.8, 0.4, 0.5. Note that time will not necessarily be monotonically increasing on a database that specifies the cycle count.

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>Count</i>	integer	–

Database Name

Syntax

Database Name *{=}* *StreamName*

Summary

The database containing the input and/or output restart data. If this analysis is being restarted, restart data will be read from this file. If the analysis is writing restart data, the data will be written to this file. It will be overwritten if it exists (after being read if applicable). If the filename begins with the ‘/’ character, it is an absolute path; otherwise, the path to the current directory will be prepended to the name. See also the ‘Input Database’ and ‘Output Database’ commands.

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>StreamName</i>	string	–

Database Type

Syntax

Database Type *{=}* *{catalyst | catalyst_exodus | cgns | dof | dof_exodus | exodus | exodusii | exonull | generated | genesis | null | parallel_exodus | textmesh}*

Summary

The database type/format used for the restart file.

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>Database Types</i>	{ catalyst catalyst_exodus cgns dof dof_exodus exodus exodusii exonull generated genesis null parallel_exodus textmesh }	–

Debug Dump

Syntax

Debug Dump

Summary

Specify whether the restart system will write the restart data immediately after reading the restart data if the run is restarting. The output data can be compared with the restart input data to determine whether they match.

Decomposition Method

Syntax

Decomposition Method *{=}* *{block | cyclic | external | geom_kway | hsf | kway | kway_geom | linear | map | metis_sfc | random | rcb | rib | variable}*

Summary

The decomposition algorithm to be used to partition elements to each processor in a parallel run.

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>Method</i>	{ block cyclic external geom_kway hsf kway kway_geom linear map metis_sfc random rcb rib variable }	–

Exists

Syntax

Exists *{=}* *{abort | add_suffix | append | overwrite}*

Summary

Specify the behavior when creating this database and there is an existing file with the same name. The default behavior is “OVERWRITE” which deletes the existing file and creates a new file of the same name. “APPEND” will (if possible) append the new data to the end of the existing file. “ABORT” will print an error message and end the analysis. “ADD_SUFFIX” will add a suffix to the file name and output to that file.

Parameter	Value	Default
<i>{=}</i>	{= is}	–
<i>Option2</i>	{abort add_suffix append overwrite}	–

File Cycle Count

Syntax

File Cycle Count *{=}* *Count*

Summary

Each restart dump will be written to a separate file suffixed with A,B, ... The count specifies how many separate files are used before the cycle repeats. For example, if “FILE CYCLE COUNT = 3” is specified, the restart dumps would be written to file-A.rs, file-B.rs, file-C.rs, file-A.rs, ... The maximum value for the cycle count is 26.

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>Count</i>	integer	–

Input Database Name

Syntax

Input Database Name *{=}* *StreamName*

Summary

The database containing the input restart data. If this analysis is being restarted, restart data will be read from this file. See also the ‘Database’ and ‘Output Database’ commands.

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>StreamName</i>	string	–

Optional

Syntax

Optional

Summary

The database will be read if it exists, but it is not an error if there is no restart database to read for this region during a restarted analysis.

Output Database Name

Syntax

Output Database Name *{=}* *StreamName*

Summary

The database containing the output restart data. If the analysis is writing restart data, the data will be written to this file. It will be overwritten if it exists. See also the 'Database' and 'Input Database' commands.

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>StreamName</i>	string	–

Output On Signal

Syntax

Output On Signal *{=}* *{sigabrt | sigalrm | sigfpe | sighup | sigill | sigint | sigkill | sigpipe | sigquit | sigsegv | sigterm | sigusr1 | sigusr2}*

Summary

When the specified signal is raised, the output stream associated with this block will be output.

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>Signals</i>	{sigabrt sigalrm sigfpe sighup sigill sigint sigkill sigpipe sigquit sigsegv sigterm sigusr1 sigusr2}	–

Output Restart State

Syntax

Output Restart State *{=}* *{off | on}*

Summary

Outputs the restarted state to the new restarted results file

Description

NOTE: This command must be placed at the Sierra scope of the input file. Allows the analyst to visualize the restarted state for debugging

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>Option</i>	{off on}	–

Overlay Count

Syntax

Overlay Count *{=}* *Count*

Summary

Specify the number of restart outputs which will be overlaid on top of the last written step. For example, if restarts are being output every 0.1 seconds and the overlay count is specified as 2, then restart will write times 0.1 to step 1 of the database. It will then write 0.2 and 0.3 also to step 1. It will then increment the database step and write 0.4 to step 2; overlay 0.5 and 0.6 on step 2. . . . At the end of the analysis, assuming it runs to completion, the database would have times 0.3, 0.6, 0.9, However, if there were a problem during the analysis, the last step on the database would contain an intermediate step.

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>Count</i>	integer	–

Overwrite

Syntax

Overwrite *{=}* *{false | no | off | on | true | yes}*

Summary

(DEPRECATED, Use EXISTS) Specify whether the restart database should be overwritten if it exists. The default behavior is to overwrite unless this command is specified in the restart block and either off, false, or no is specified.

Parameter	Value	Default
<i>{=}</i>	{ = is }	–
<i>Option2</i>	{ false no off on true yes }	–

Property

Syntax

Property *PropertyName* *{=}* *PropertyValue*

Summary

Define a database property named “PropertyName” with the value “PropertyValue”. If PropertyValue consists of all digits, it will define an integer property. If PropertyValue is “true” or “yes” or “false” or “no”, it will define a logical property; otherwise it will define a string property. If PropertyName consists of multiple strings, they will be concatenated together with “_” separating the individual words. Supported properties are typically database dependent; Current properties are:

- COMPRESSION_LEVEL = [0..9]
- COMPRESSION_SHUFFLE = true|false|on|off
- FILE_TYPE = netcdf4 (forces use of netcdf-4 hdf5-based file)

- INTEGER_SIZE_DB = 4|8
- INTEGER_SIZE_API = 4|8
- LOGGING = true|false|on|off
- MAX_NAME_LENGTH = value

Parameter	Value	Default
<i>PropertyName</i>	string	–
<i>{=}</i>	{ = are is }	–
<i>PropertyValue</i>	string	–

Restart

Syntax

Restart *{=}* *{auto}*

Summary

Specify automatic restart file read.

Description

NOTE: This command must be placed at the Sierra scope of the input file.

Specify that the analysis should be restarted from the last common time on all restart databases for each Region in the analysis. In addition to this line command, each Region in the analysis (strictly, only the region(s) that will be restarted) must have a restart block specifying the database to read the restart state data.

By default, use of this command will not cause output files (e.g., results, history, heartbeat, restart) to be overwritten. Instead output files will be written with the same basename and the suffix *-s0000**. Common visualization packages are written to handle this file organization gracefully in order for the user to view all results seamlessly.

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>{auto}</i>	{ auto automatic }	–

Restart Time

Syntax

Restart Time *{=}* *Time*

Summary

Specify restart file read at a specified time.

Description

NOTE: This command must be placed at the Sierra scope of the input file.

Specify the time that the analysis will be restarted. In addition to this line command, each Region in the analysis (strictly, only the region(s) that will be restarted) must have a restart block specifying the database to read the restart state data. The restart ‘time’ must be greater than zero and less than or equal to the termination time.

By default, use of this command will cause previous output files (e.g., results, history, heartbeat, restart) to be overwritten. If this command is chosen, the onus is placed on the user to ensure that previous output files are not overwritten.

Parameter	Value	Default
<i>{=}</i>	{= are is }	–
<i>Time</i>	real	–

Start Time

Syntax

Start Time *{=}* *Start_time*

Summary

Specify the time to start outputting results from this output request block. This time overrides all ‘at time’ and ‘at step’ specifications.

Parameter	Value	Default
<i>{=}</i>	{= are is }	–
<i>Start_time</i>	real	–

Synchronize Output

Syntax

Synchronize Output

Summary

In an analysis with multiple regions, it is sometimes desirable to synchronize the output of results data between the regions. This can be done by adding the SYNCHRONIZE OUTPUT command line to the results output block. If a results block has this set, then it will write output whenever a previous region writes output. The ordering of regions is based on the order in the input file, algorithmic considerations, or by solution control specifications.

Although the USE OUTPUT SCHEDULER command line can also synchronize output between regions, the SYNCHRONIZE OUTPUT command line will synchronize the output with regions where the output frequency is not under the direct control of the Sierra IO system. Examples of this are typically coupled

applications where one or more of the codes are not Sierra-based applications such as Alegra and CTH. A results block with SYNCHRONIZE OUTPUT specified will also synchronize its output with the output of the external code.

The SYNCHRONIZE OUTPUT command can be used with other output scheduling commands such as time-based or step-based output specifications.

Shift To Start Time

Syntax

Shift To Start Time

Summary

The shift to start time option allows a user to shift the restart time to the start time of the current region. An example use case would be if a restart time of 0.5 is specified, but the user would like to start the simulation at time 1.0.

Termination Time

Syntax

Termination Time *{=}* *Final_time*

Summary

Specify the time to stop outputting results from this output request block.

Parameter	Value	Default
<i>{=}</i>	{= are is }	–
<i>Final_time</i>	real	–

Timestep Adjustment Interval

Syntax

Timestep Adjustment Interval *{=}* *Nsteps*

Summary

Specify the number of steps to ‘look ahead’ and adjust the timestep to ensure that the specified output times or simulation end time will be hit ‘exactly’.

Parameter	Value	Default
<i>{=}</i>	{= are is }	–
<i>Nsteps</i>	integer	–

Use Dynamic Topology Io

Syntax

Use Dynamic Topology Io

Summary

Specify that the app use IO for dynamic topology modifications where the

output files are stored in a single database. Legacy file format for dynamically changing topology results in the creation of multiple files for each output on a mesh modification. This option leverages the ability of netCDF to create mesh groups within a single database and concatenate all mesh files into one. The names of each mesh group are of the form IOSS_MESH_GROUP-??? where ??? is the 1-based output index 1, 2, ..., 10, ..., 100, ... Please note that netCDF has a current limit of 65,536 groups

Use Output Scheduler

Syntax

Use Output Scheduler *Timer_name*

Summary

Associates a predefined output scheduler with this output block (results, restart, heartbeat, or history).

Parameter	Value	Default
<i>Timer_name</i>	string	–

Results Output

Scope

Average Region, Fuego Region, Input_Output Region, Particle Region

Summary

Describes the location and type of the output stream used for outputting results for the enclosing region.

begin Results Output *Label*

Additional Steps {=} *List_of_steps...*

Additional Times {=} *List_of_times...*

At Step *n* {increment | interval} {=} *m*

At Time *Dt1* {increment | interval} {=} *Dt2*

Auto Output {all | element | global | nodal} User Defined Variables [
 ↳In *UserOutputResultsList...*]

Auto Output {all | element | global | nodal} Variables

Component Separator Character {=} *Separator*

Database Name {=} *StreamName*

Database Type {=} {*catalyst* | *catalyst_exodus* | *cgns* | *dof* | *dof_*
↳*exodus* | *exodus* | *exodusii* | *exonull* | *generated* | *genesis* | *null* |
↳*parallel_exodus* | *textmesh*}

Edge *VariableList...*

Element *VariableList...*

Enable Large Ids

Exclude {=} [*ElementBlockList...*]

Exists {=} {*abort* | *add_suffix* | *append* | *overwrite*}

Face *VariableList...*

Flush Interval {=} *Option*

Global *VariableList...*

Include {=} [*ElementBlockList...*]

Nodal *VariableList...*

Node *VariableList...*

Nodeset *VariableList...*

Output Mesh {=} {*exposed surface* | *refined* | *unrefined*}

Output On Signal {=} {*sigabrt* | *sigalrm* | *sigfpe* | *sighup* | *sigill* |
↳*sigint* | *sigkill* | *sigpipe* | *sigquit* | *sigsegv* | *sigterm* | *sigusr1* |
↳*sigusr2*}

Overwrite {=} {*false* | *no* | *off* | *on* | *true* | *yes*}

Property *PropertyName* {=} *PropertyValue*

Sideset *VariableList...*

Start Time {=} *Start_time*

Surface *VariableList...*

Synchronize Output

Termination Time `{=}` *Final_time*

Timeseries Name `{=}` *filename*

Timestep Adjustment Interval `{=}` *Nsteps*

Title

Use Dynamic Topology Io

Use Output Scheduler *Timer_name*

begin Catalyst *Label*
end

end Results Output *Label*

Line Commands

Additional Steps

Syntax

Additional Steps `{=}` *List_of_steps...*

Summary

Additional simulation steps when output should occur.

Parameter	Value	Default
<code>{=}</code>	{= are is}	–
<i>List_of_steps</i>	integer...	–

Additional Times

Syntax

Additional Times `{=}` *List_of_times...*

Summary

Additional simulation times when output should occur.

Parameter	Value	Default
<code>{=}</code>	{= are is}	–
<i>List_of_times</i>	real...	–

At Step

Syntax

At Step *n* {*increment* | *interval*} {=} *m*

Summary

Specify an output interval in terms of the internal iteration step count. The first step specifies the step count at the beginning of this interval and the second step specifies the output frequency to be used within this interval.

Parameter	Value	Default
<i>n</i>	integer	—
<i>Option</i>	{ <i>increment</i> <i>interval</i> }	—
{=}	{= are is}	—
<i>m</i>	integer	—

At Time

Syntax

At Time *Dt1* {*increment* | *interval*} {=} *Dt2*

Summary

Specify an output interval in terms of the internal simulation time. The first time specifies the time at the beginning of this time interval and the second time specifies the output frequency to be used within this interval.

Parameter	Value	Default
<i>Dt1</i>	real	—
<i>Option</i>	{ <i>increment</i> <i>interval</i> }	—
{=}	{= are is}	—
<i>Dt2</i>	real	—

Auto Output

Syntax

Auto Output {*all* | *element* | *global* | *nodal*} User Defined Variables [In *UserOutputResultsList...*]

Summary

Allows users to automatically output all user output defined variables for the type requested.

Parameter	Value	Default
<i>auto_output_type_3</i>	{ <i>all</i> <i>element</i> <i>global</i> <i>nodal</i> }	—

Auto Output

Syntax

Auto Output *{all | element | global | nodal}* Variables

Summary

Allows users to automatically output all user output defined variables for the type requested.

Parameter	Value	Default
<i>auto_output_type_3</i>	{all element global nodal}	–

Component Separator Character

Syntax

Component Separator Character *{=} Separator*

Summary

The separator is the single character used to separate the output variable basename (e.g. “stress”) from the suffices (e.g. “xx”, “yy”) when displaying the names of the individual variable components. For example, the default separator is “_”, which results in names similar to “stress_xx”, “stress_yy”, ... “stress_zx”. To eliminate the separator, specify an empty string (“”) or NONE.

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>Separator</i>	string	–

Database Name

Syntax

Database Name *{=} StreamName*

Summary

The base name of the database containing the output results. If the filename begins with the ‘/’ character, it is an absolute path; otherwise, the path to the current directory will be prepended to the name. If this line is omitted, then a filename will be created from the basename of the input file with a “.e” suffix appended.

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>StreamName</i>	string	–

Database Type

Syntax

Database Type *{=} {catalyst | catalyst_exodus | cgns | dof | dof_exodus | exodus}*

| *exodusii* | *exonull* | *generated* | *genesis* | *null* | *parallel_exodus* | *textmesh*}

Summary

The database type/format to be used for the output results.

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>Database Type</i>	{ catalyst catalyst_exodus cgns dof dof_exodus exodus exodusii exonull generated genesis null parallel_exodus textmesh }	–

Edge

Syntax

Edge *VariableList*...

Summary

Define the variables that should be written to the results database. If “variable” is entered, then its name will be used on the output database. If “variable as db_name” is entered, then “db_name” will be the name used on the database for the internal variable “variable”. Multiple “variable” or “variable as db_name” entries are allowed on the same line. The entities that this variable are written to can also be limited or specified with “exclude list_of_entities” or “include list_of_entities”. Edge variables are not supported for all database types.

Parameter	Value	Default
<i>VariableList</i>	string...	–

Element

Syntax

Element *VariableList*...

Summary

Define the variables that should be written to the results database. If “variable” is entered, then its name will be used on the output database. If “variable as db_name” is entered, then “db_name” will be the name used on the database for the internal variable “variable”. Multiple “variable” or “variable as db_name” entries are allowed on the same line. The entities that this variable are written to can also be limited or specified with “exclude list_of_entities” or “include list_of_entities”

Parameter	Value	Default
<i>VariableList</i>	string...	–

Enable Large Ids

Syntax

Enable Large Ids

Summary

Enable 64 bit entity IDs for output

Exclude

Syntax

Exclude *{=}* [*ElementBlockList. . .*]

Summary

Specify that the results file will only contain a subset of the element blocks in the analysis model. The element_block_list lists only the blocks which will not be output to the results database.

Parameter	Value	Default
<i>{=}</i>	{= are is}	–

Exists

Syntax

Exists *{=}* *{abort | add_suffix | append | overwrite}*

Summary

Specify the behavior when creating this database and there is an existing file with the same name. The default behavior is “OVERWRITE” which deletes the existing file and creates a new file of the same name. “APPEND” will (if possible) append the new data to the end of the existing file. “ABORT” will print an error message and end the analysis. “ADD_SUFFIX” will add a -s???? suffix where the ???? is replaced by a sequential number starting at 0002.

Parameter	Value	Default
<i>{=}</i>	{= is}	–
<i>Option2</i>	{abort add_suffix append overwrite}	–

Face

Syntax

Face *VariableList. . .*

Summary

Define the variables that should be written to the results database. If “variable” is entered, then its name will be used on the output database. If “variable as db_name” is entered, then “db_name” will be the name used on the database for the internal variable “variable”. Multiple “variable” or “variable as db_name” entries are allowed on the same line. The entities that this variable are written to

can also be limited or specified with “exclude list_of_entities” or “include list_of_entities”. Face variables are not supported for all database types.

Parameter	Value	Default
<i>VariableList</i>	string. . .	–

Flush Interval

Syntax

Flush Interval *{=}* *Option*

Summary

The minimum time interval (in seconds) at which output will be explicitly flushed to disk. The default is 10 seconds.

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>Option</i>	integer	10

Global

Syntax

Global *VariableList. . .*

Summary

Define the global variables that should be written to the results database. If “variable” is entered, then its name will be used on the output database. If “variable as db_name” is entered, then “db_name” will be the name used on the database for the internal variable “variable”. Multiple “variable” or “variable as db_name” entries are allowed on the same line.

Parameter	Value	Default
<i>VariableList</i>	string. . .	–

Include

Syntax

Include *{=}* [*ElementBlockList. . .*]

Summary

Specify that the results file will only contain a subset of the element blocks in the analysis model. The element_block_list lists only the blocks which will be output to the results database.

Parameter	Value	Default
<i>{=}</i>	{= are is}	–

Nodal

Syntax

Nodal *VariableList...*

Summary

Define the nodal variables that should be written to the results database. If “variable” is entered, then its name will be used on the output database. If “variable as db_name” is entered, then “db_name” will be the name used on the database for the internal variable “variable”. Multiple “variable” or “variable as db_name” entries are allowed on the same line.

Parameter	Value	Default
<i>VariableList</i>	string...	–

Node

Syntax

Node *VariableList...*

Summary

Define the nodal variables that should be written to the results database. If “variable” is entered, then its name will be used on the output database. If “variable as db_name” is entered, then “db_name” will be the name used on the database for the internal variable “variable”. Multiple “variable” or “variable as db_name” entries are allowed on the same line.

Parameter	Value	Default
<i>VariableList</i>	string...	–

Nodeset

Syntax

Nodeset *VariableList...*

Summary

Define the variables that should be written to the results database. If “variable” is entered, then its name will be used on the output database. If “variable as db_name” is entered, then “db_name” will be the name used on the database for the internal variable “variable”. Multiple “variable” or “variable as db_name” entries are allowed on the same line. The entities that this variable are written to can also be limited or specified with “exclude list_of_entities” or “include list_of_entities”. Nodeset variables are not supported for all database types.

Parameter	Value	Default
<i>VariableList</i>	string...	–

Output Mesh

Syntax

Output Mesh *{=}* *{exposed surface | refined | unrefined}*

Summary

Use this command to turn on “unrefined” as the output mesh. The default behavior is “refined”, in which field variables are output on the current mesh, which may have been refined (either uniformly or adaptively) or had its topology altered in some way (e.g., dynamic load balancing) with respect to the original mesh read from the input file. By specifying “Output Mesh = unrefined”, all output variables are output only on the original mesh objects read from the input file.

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>OutputMesh</i>	{ exposed surface refined unrefined }	–

Output On Signal

Syntax

Output On Signal *{=}* *{sigabrt | sigalrm | sigfpe | sighup | sigill | sigint | sigkill | sigpipe | sigquit | sigsegv | sigterm | sigusr1 | sigusr2}*

Summary

When the specified signal is raised, the output stream associated with this block will be output.

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>Signals</i>	{ sigabrt sigalrm sigfpe sighup sigill sigint sigkill sigpipe sigquit sigsegv sigterm sigusr1 sigusr2 }	–

Overwrite

Syntax

Overwrite *{=}* *{false | no | off | on | true | yes}*

Summary

(DEPRECATED, Use EXISTS) Specify whether the database should be overwritten if it exists. The default behavior is to overwrite unless this command is specified in the output block and either off, false, or no is specified.

Parameter	Value	Default
<i>{=}</i>	{ = is }	–
<i>Option2</i>	{ false no off on true yes }	–

Property

Syntax

Property *PropertyName* *{=}* *PropertyValue*

Summary

Define a database property named “PropertyName” with the value “PropertyValue”. If PropertyValue consists of all digits, it will define an integer property. If PropertyValue is “true” or “yes” or “false” or “no”, it will define a logical property; otherwise it will define a string property. Supported properties are typically database dependent; Current properties are:

- COMPRESSION_LEVEL = [0..9] (off)
- COMPRESSION_SHUFFLE = true|false|on|off (off)
- FILE_TYPE = netcdf4 (forces use of netcdf-4 hdf5-based file) (netcdf3)
- INTEGER_SIZE_DB = 4|8 (4)
- INTEGER_SIZE_API = 4|8 (4)
- REAL_SIZE_DB = 4|8 (8 is default)
- LOGGING = true|false|on|off (off)
- MAX_NAME_LENGTH = value (32)

Parameter	Value	Default
<i>PropertyName</i>	string	–
<i>{=}</i>	{ = are is }	–
<i>PropertyValue</i>	string	–

Sideset

Syntax

Sideset *VariableList...*

Summary

Define the variables that should be written to the results database. If “variable” is entered, then its name will be used on the output database. If “variable as db_name” is entered, then “db_name” will be the name used on the database for the internal variable “variable”. Multiple “variable” or “variable as db_name” entries are allowed on the same line. The entities that this variable are written to

can also be limited or specified with “exclude list_of_entities” or “include list_of_entities”. Face variables are not supported for all database types.

Parameter	Value	Default
<i>VariableList</i>	string. . .	–

Start Time

Syntax

Start Time *{=}* *Start_time*

Summary

Specify the time to start outputting results from this output request block. This time overrides all ‘at time’ and ‘at step’ specifications.

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>Start_time</i>	real	–

Surface

Syntax

Surface *VariableList. . .*

Summary

Define the variables that should be written to the results database. If “variable” is entered, then its name will be used on the output database. If “variable as db_name” is entered, then “db_name” will be the name used on the database for the internal variable “variable”. Multiple “variable” or “variable as db_name” entries are allowed on the same line. The entities that this variable are written to can also be limited or specified with “exclude list_of_entities” or “include list_of_entities”. Face variables are not supported for all database types.

Parameter	Value	Default
<i>VariableList</i>	string. . .	–

Synchronize Output

Syntax

Synchronize Output

Summary

In an analysis with multiple regions, it is sometimes desirable to synchronize the output of results data between the regions. This can be done by adding the SYNCHRONIZE OUTPUT command line to the results output block. If a results block has this set, then it will write output whenever a previous region

writes output. The ordering of regions is based on the order in the input file, algorithmic considerations, or by solution control specifications.

Although the USE OUTPUT SCHEDULER command line can also synchronize output between regions, the SYNCHRONIZE OUTPUT command line will synchronize the output with regions where the output frequency is not under the direct control of the Sierra IO system. Examples of this are typically coupled applications where one or more of the codes are not Sierra-based applications such as Alegria and CTH. A results block with SYNCHRONIZE OUTPUT specified will also synchronize its output with the output of the external code.

The SYNCHRONIZE OUTPUT command can be used with other output scheduling commands such as time-based or step-based output specifications.

Termination Time

Syntax

Termination Time *{=}* *Final_time*

Summary

Specify the time to stop outputting results from this output request block.

Parameter	Value	Default
<i>{=}</i>	{= are is }	–
<i>Final_time</i>	real	–

Timeseries Name

Syntax

Timeseries Name *{=}* *filename*

Summary

Optionally specify a filename for a timeseries file that outputs the root database filename in the order that they are written. This is useful when running on large numbers of processors with many mesh-mods that cause simple disk operations to hang.

Parameter	Value	Default
<i>{=}</i>	{= are is }	–
<i>filename</i>	string	–

Timestep Adjustment Interval

Syntax

Timestep Adjustment Interval *{=}* *Nsteps*

Summary

Specify the number of steps to ‘look ahead’ and adjust the timestep to ensure

that the specified output times or simulation end time will be hit ‘exactly’.

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>Nsteps</i>	integer	–

Title

Syntax

Title

Summary

Specify the title to be used for this specific output block.

Use Dynamic Topology Io

Syntax

Use Dynamic Topology Io

Summary

Specify that the app use IO for dynamic topology modifications where the output files are stored in a single database. Legacy file format for dynamically changing topology results in the creation of multiple files for each output on a mesh modification. This option leverages the ability of netCDF to create mesh groups within a single database and concatenate all mesh files into one. The names of each mesh group are of the form IOSS_MESH_GROUP-??? where ??? is the 1-based output index 1, 2, ..., 10, ..., 100, ... Please note that netCDF has a current limit of 65,536 groups

Use Output Scheduler

Syntax

Use Output Scheduler *Timer_name*

Summary

Associates a predefined output scheduler with this output block (results, restart, heartbeat, or history).

Parameter	Value	Default
<i>Timer_name</i>	string	–

7.9 Average Region

Average Region

Scope

Fuego Procedure

Summary

Contains the commands needed to execute an analysis on this region.

begin Average Region *Regionname*

Use Finite Element Model *ModelName* [Model Coordinates Are *Nodal_↪variable_name*]

begin Heartbeat *Label*
end

begin Heartbeat Output *Label*
end

begin History Output *Label*
end

begin Restart Data *Label*
end

begin Results Output *Label*
end

begin Solution Options *OptionsName*
end

end Average Region *Regionname*

Line Commands

Use Finite Element Model

Syntax

Use Finite Element Model *ModelName* [Model Coordinates Are *Nodal_variable_name*]

Summary

Associates a predefined finite element model with this region.

Parameter	Value	Default
<i>ModelName</i>	string	–

History Output

Scope

Average Region, Fuego Region, Input_Output Region, Particle Region

Summary

Describes the location and type of the output stream used for outputting history for the enclosing region.

begin History Output *Label*

Additional Steps {=} *List_of_steps...*

Additional Times {=} *List_of_times...*

At Step *n* {increment | interval} {=} *m*

At Time *Dt1* {increment | interval} {=} *Dt2*

Auto Output {all | element | global | nodal} User Defined Variables [*␣*
↳ In *UserOutputHistoryList...*]

Database Name {=} *StreamName*

Database Type {=} {catalyst | catalyst_exodus | cgns | dof | dof_
↳ exodus | exodus | exodusii | exonull | generated | genesis | null | *␣*
↳ parallel_exodus | textmesh}

Element [*VariableList...*]

Exists {=} {abort | add_suffix | append | overwrite}

Face [*VariableList...*]

Flush Interval {=} *Option*

Global [*Variables...*]

Nodal [*VariableList...*]

Node [*VariableList...*]

Nodeset [*VariableList...*]

Output On Signal {=} {sigabrt | sigalrm | sigfpe | sighup | sigill |
↪sigint | sigkill | sigpipe | sigquit | sigsegv | sigterm | sigusr1 |
↪sigusr2}

Overwrite {=} {false | no | off | on | true | yes}

Property *PropertyName* {=} *PropertyValue*

Start Time {=} *Start_time*

Synchronize Output

Termination Time {=} *Final_time*

Timestep Adjustment Interval {=} *Nsteps*

Title

Use Dynamic Topology Io

Use Output Scheduler *Timer_name*

Variable {=} {edge | element | face | global | nodal | node} *Variable_*
↪*list...*

end History Output *Label*

Line Commands

Additional Steps

Syntax

Additional Steps {=} *List_of_steps...*

Summary

Additional simulation steps when output should occur.

Parameter	Value	Default
{=}	{= are is}	–
<i>List_of_steps</i>	integer...	–

Additional Times

Syntax

Additional Times *{=}* *List_of_times* . .

Summary

Additional simulation times when output should occur.

Parameter	Value	Default
<i>{=}</i>	{= are is}	—
<i>List_of_times</i>	real. . .	—

At Step

Syntax

At Step *n* *{increment | interval}* *{=}* *m*

Summary

Specify an output interval in terms of the internal iteration step count. The first step specifies the step count at the beginning of this interval and the second step specifies the output frequency to be used within this interval.

Parameter	Value	Default
<i>n</i>	integer	—
<i>Option</i>	{increment interval}	—
<i>{=}</i>	{= are is}	—
<i>m</i>	integer	—

At Time

Syntax

At Time *Dt1* *{increment | interval}* *{=}* *Dt2*

Summary

Specify an output interval in terms of the internal simulation time. The first time specifies the time at the beginning of this time interval and the second time specifies the output frequency to be used within this interval.

Parameter	Value	Default
<i>Dt1</i>	real	—
<i>Option</i>	{increment interval}	—
<i>{=}</i>	{= are is}	—
<i>Dt2</i>	real	—

Auto Output

Syntax

Auto Output {*all* | *element* | *global* | *nodal*} User Defined Variables [In *UserOutputHistoryList*. . .]

Summary

Allows users to automatically output all user output defined variables for the type requested.

Parameter	Value	Default
<i>auto_output_type_2</i>	{all element global nodal}	–

Database Name

Syntax

Database Name {=} *StreamName*

Summary

The base name of the database containing the output history. If the filename begins with the ‘/’ character, it is an absolute path; otherwise, the path to the current directory will be prepended to the name. If this line is omitted, then a filename will be created from the basename of the input file with a “.h” suffix appended.

Parameter	Value	Default
{=}	{= are is}	–
<i>StreamName</i>	string	–

Database Type

Syntax

Database Type {=} {*catalyst* | *catalyst_exodus* | *cgns* | *dof* | *dof_exodus* | *exodus* | *exodusii* | *exonull* | *generated* | *genesis* | *null* | *parallel_exodus* | *textmesh*}

Summary

The database type/format to be used for the output history.

Parameter	Value	Default
{=}	{= are is}	–
<i>DatabaseTypes</i>	{catalyst catalyst_exodus cgns dof dof_exodus exodus exodusii exonull generated genesis null parallel_exodus textmesh}	–

Element

Syntax

Element [*VariableList*. . .]

Summary

Define the element variables that should be written to the history database. The syntax is: “element {internal_name} at element {id} as {DBname}” or “element {internal_name} nearest location X, Y, Z as {DBname}”.

Where {internal_name} is the name of the variable in the Sierra application; and {DBname} is the name as it should appear on the history database.

Exists

Syntax

Exists {=} {abort | add_suffix | append | overwrite}

Summary

Specify the behavior when creating this database and there is an existing file with the same name. The default behavior is “OVERWRITE” which deletes the existing file and creates a new file of the same name. “APPEND” will (if possible) append the new data to the end of the existing file. “ABORT” will print an error message and end the analysis. “ADD_SUFFIX” will add a -s???? suffix where the ???? is replaced by a sequential number starting at 0002.

Parameter	Value	Default
{=}	{= is}	–
<i>Option2</i>	{abort add_suffix append overwrite}	–

Face

Syntax

Face [*VariableList...*]

Summary

Define the face variables that should be written to the history database. The syntax is: “face {internal_name} at face {id} as {DBname}” or “face {internal_name} nearest location X, Y, Z as {DBname}”.

Where {internal_name} is the name of the variable in the Sierra application; and {DBname} is the name as it should appear on the history database.

Flush Interval

Syntax

Flush Interval {=} *Option*

Summary

The minimum time interval (in seconds) at which output will be explicitly flushed to disk. The default is 10 seconds.

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>Option</i>	integer	10

Global

Syntax

Global [*Variables...*]

Summary

Define the global/reduction variables that should be written to the history database. The syntax is: “global {internal_name} as {DBname}”.

Where {internal_name} is the name of the variable in the Sierra application; and {DBname} is the name as it should appear on the history database.

Nodal

Syntax

Nodal [*VariableList...*]

Summary

Define the nodal variables that should be written to the history database. The syntax is: “nodal {internal_name} at node {id} as {DBname}” or “nodal {internal_name} nearest location X, Y, Z as {DBname}”.

Where {internal_name} is the name of the variable in the Sierra application; and {DBname} is the name as it should appear on the history database.

Node

Syntax

Node [*VariableList...*]

Summary

Define the nodal variables that should be written to the history database. The syntax is: “node {internal_name} at node {id} as {DBname}” or “node {internal_name} nearest location X, Y, Z as {DBname}”.

Where {internal_name} is the name of the variable in the Sierra application; and {DBname} is the name as it should appear on the history database.

Nodeset

Syntax

Nodeset [*VariableList...*]

Summary

Define the nodeset variables that should be written to the history database. The syntax is: “nodeset {internal_name} at node {id} as {DBname}” or “nodeset {internal_name} nearest location X, Y, Z as {DBname}”.

Where {internal_name} is the name of the variable in the Sierra application; and {DBname} is the name as it should appear on the history database.

Output On Signal

Syntax

Output On Signal *{=}* *{sigabrt | sigalrm | sigfpe | sighup | sigill | sigint | sigkill | sigpipe | sigquit | sigsegv | sigterm | sigusr1 | sigusr2}*

Summary

When the specified signal is raised, the output stream associated with this block will be output.

Parameter	Value	Default
<i>{=}</i>	{ = are is }	—
<i>Signals</i>	{ sigabrt sigalrm sigfpe sighup sigill sigint sigkill sigpipe sigquit sigsegv sigterm sigusr1 sigusr2 }	—

Overwrite

Syntax

Overwrite *{=}* *{false | no | off | on | true | yes}*

Summary

(DEPRECATED, Use EXISTS) Specify whether the database should be overwritten if it exists. The default behavior is to overwrite unless this command is specified in the output block and either off, false, or no is specified.

Parameter	Value	Default
<i>{=}</i>	{ = is }	—
<i>Option2</i>	{ false no off on true yes }	—

Property

Syntax

Property *PropertyName* *{=}* *PropertyValue*

Summary

Define a database property named “PropertyName” with the value “PropertyValue”. If PropertyValue consists of all digits, it will define an integer property. If PropertyValue is “true” or “yes” or “false” or “no”, it will define a logical property; otherwise it will define a string property. Supported properties are typically database dependent; Some history-related properties are:

- VARIABLE_NAME_CASE = upper|lower

- MAX_NAME_LENGTH = value (32)

Parameter	Value	Default
<i>PropertyName</i>	string	–
<i>{=}</i>	{ = are is }	–
<i>PropertyValue</i>	string	–

Start Time

Syntax

Start Time *{=}* *Start_time*

Summary

Specify the time to start outputting results from this output request block. This time overrides all ‘at time’ and ‘at step’ specifications.

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>Start_time</i>	real	–

Synchronize Output

Syntax

Synchronize Output

Summary

In an analysis with multiple regions, it is sometimes desirable to synchronize the output of results data between the regions. This can be done by adding the SYNCHRONIZE OUTPUT command line to the results output block. If a results block has this set, then it will write output whenever a previous region writes output. The ordering of regions is based on the order in the input file, algorithmic considerations, or by solution control specifications.

Although the USE OUTPUT SCHEDULER command line can also synchronize output between regions, the SYNCHRONIZE OUTPUT command line will synchronize the output with regions where the output frequency is not under the direct control of the Sierra IO system. Examples of this are typically coupled applications where one or more of the codes are not Sierra-based applications such as Alegria and CTH. A results block with SYNCHRONIZE OUTPUT specified will also synchronize its output with the output of the external code.

The SYNCHRONIZE OUTPUT command can be used with other output scheduling commands such as time-based or step-based output specifications.

Termination Time

Syntax

Termination Time *{=}* *Final_time*

Summary

Specify the time to stop outputting results from this output request block.

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>Final_time</i>	real	–

Timestep Adjustment Interval

Syntax

Timestep Adjustment Interval *{=}* *Nsteps*

Summary

Specify the number of steps to ‘look ahead’ and adjust the timestep to ensure that the specified output times or simulation end time will be hit ‘exactly’.

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>Nsteps</i>	integer	–

Title

Syntax

Title

Summary

Specify the title to be used for this specific output block.

Use Dynamic Topology Io

Syntax

Use Dynamic Topology Io

Summary

Specify that the app use IO for dynamic topology modifications where the output files are stored in a single database. Legacy file format for dynamically changing topology results in the creation of multiple files for each output on a mesh modification. This option leverages the ability of netCDF to create mesh groups within a single database and concatenate all mesh files into one. The names of each mesh group are of the form IOSS_MESH_GROUP-??? where ??? is the 1-based output index 1, 2, ..., 10, ..., 100, ... Please note that netCDF has a current limit of 65,536 groups

Use Output Scheduler

Syntax

Use Output Scheduler *Timer_name*

Summary

Associates a predefined output scheduler with this output block (results, restart, heartbeat, or history).

Parameter	Value	Default
<i>Timer_name</i>	string	—

Variable

Syntax

Variable *{=}* *{edge | element | face | global | nodal | node}* *Variable_list* . .

Summary

Define the variables that should be written to the history database. The syntax is: “variable = entity {internal_name} at entity {id} as {DBname}” or “variable = entity {internal_name} nearest location X, Y, Z as {DBname}” or “variable = entity {internal_name} at location X, Y, Z as {DBname}”.

Where {entity} is ‘node’, ‘element’, ‘face’, or ‘edge’; {internal_name} is the name of the variable in the Sierra application; and {DBname} is the name as it should appear on the history database.

Parameter	Value	Default
<i>{=}</i>	{ = are is }	—
<i>Option</i>	{ edge element face global nodal node }	—
<i>Variable_list</i>	string. . .	—

Restart Data

Scope

Average Region, Fuego Region, Input_Output Region, Particle Region

Summary

Describes the data required to output and input restart data for the enclosing region.

begin Restart Data *Label*

Additional Steps *{=}* *List_of_steps* . . .

Additional Times *{=}* *List_of_times* . . .

At Step *n* *{increment | interval}* *{=}* *m*

At Time *Dt1* *{increment | interval}* *{=}* *Dt2*

At Wall Time *Dt1* {increment | interval} {=} *Dt2*

Component Separator Character {=} *Separator*

Cycle Count {=} *Count*

Database Name {=} *StreamName*

Database Type {=} {catalyst | catalyst_exodus | cgns | dof | dof_
→exodus | exodus | exodusii | exonull | generated | genesis | null |
→parallel_exodus | textmesh}

Debug Dump

Decomposition Method {=} {block | cyclic | external | geom_kway | hsf_
→| kway | kway_geom | linear | map | metis_sfc | random | rcb | rib |
→variable}

Exists {=} {abort | add_suffix | append | overwrite}

File Cycle Count {=} *Count*

Input Database Name {=} *StreamName*

Optional

Output Database Name {=} *StreamName*

Output On Signal {=} {sigabrt | sigalrm | sigfpe | sighup | sigill |
→sigint | sigkill | sigpipe | sigquit | sigsegv | sigterm | sigusr1 |
→sigusr2}

Output Restart State {=} {off | on}

Overlay Count {=} *Count*

Overwrite {=} {false | no | off | on | true | yes}

Property *PropertyName* {=} *PropertyValue*

Restart {=} {auto}

Restart Time {=} *Time*

Start Time `{=}` *Start_time*

Synchronize Output

Shift To Start Time

Termination Time `{=}` *Final_time*

Timestep Adjustment Interval `{=}` *Nsteps*

Use Dynamic Topology Io

Use Output Scheduler *Timer_name*

end Restart Data *Label*

Line Commands

Additional Steps

Syntax

Additional Steps `{=}` *List_of_steps* . .

Summary

Additional simulation steps when output should occur.

Parameter	Value	Default
<code>{=}</code>	{= are is}	—
<i>List_of_steps</i>	integer. . .	—

Additional Times

Syntax

Additional Times `{=}` *List_of_times* . .

Summary

Additional simulation times when output should occur.

Parameter	Value	Default
<code>{=}</code>	{= are is}	—
<i>List_of_times</i>	real. . .	—

At Step

Syntax

At Step *n* {*increment* | *interval*} {=} *m*

Summary

Specify an output interval in terms of the internal iteration step count. The first step specifies the step count at the beginning of this interval and the second step specifies the output frequency to be used within this interval.

Parameter	Value	Default
<i>n</i>	integer	—
<i>Option</i>	{increment interval}	—
{=}	{= are is}	—
<i>m</i>	integer	—

At Time

Syntax

At Time *Dt1* {*increment* | *interval*} {=} *Dt2*

Summary

Specify an output interval in terms of the internal simulation time. The first time specifies the time at the beginning of this time interval and the second time specifies the output frequency to be used within this interval.

Parameter	Value	Default
<i>Dt1</i>	real	—
<i>Option</i>	{increment interval}	—
{=}	{= are is}	—
<i>Dt2</i>	real	—

At Wall Time

Syntax

At Wall Time *Dt1* {*increment* | *interval*} {=} *Dt2*

Summary

Write a restart file at a specific wall time since the start of the run. Time string format allows s, m, h, d for seconds, minutes, hours, days

Parameter	Value	Default
<i>Dt1</i>	string	—
<i>Option</i>	{increment interval}	—
{=}	{= are is}	—
<i>Dt2</i>	string	—

Component Separator Character

Syntax

Component Separator Character *{=}* *Separator*

Summary

The separator is the single character used to separate the output variable basename (e.g. “stress”) from the suffices (e.g. “xx”, “yy”) when displaying the names of the individual variable components. For example, the default separator is “_”, which results in names similar to “stress_xx”, “stress_yy”, ... “stress_zx”. To eliminate the separator, specify an empty string (“”) or NONE.

Parameter	Value	Default
<i>{=}</i>	{= is}	–
<i>Separator</i>	string	–

Cycle Count

Syntax

Cycle Count *{=}* *Count*

Summary

Specify the number of restart steps which will be written to the restart database before previously written steps are overwritten. For example, if the cycle count is 5 and restart is written every 0.1 seconds, the restart system will write 0.1, 0.2, 0.3, 0.4, 0.5 to the database. It will then overwrite the first step with data from time 0.6, the second with time 0.7. At time 0.8, the database would contain data at times 0.6, 0.7, 0.8, 0.4, 0.5. Note that time will not necessarily be monotonically increasing on a database that specifies the cycle count.

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>Count</i>	integer	–

Database Name

Syntax

Database Name *{=}* *StreamName*

Summary

The database containing the input and/or output restart data. If this analysis is being restarted, restart data will be read from this file. If the analysis is writing restart data, the data will be written to this file. It will be overwritten if it exists (after being read if applicable). If the filename begins with the ‘/’ character, it is an absolute path; otherwise, the path to the current directory will be prepended to the name. See also the ‘Input Database’ and ‘Output Database’ commands.

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>StreamName</i>	string	–

Database Type

Syntax

Database Type *{=}* { *catalyst* | *catalyst_exodus* | *cgns* | *dof* | *dof_exodus* | *exodus* | *exodusii* | *exonull* | *generated* | *genesis* | *null* | *parallel_exodus* | *textmesh* }

Summary

The database type/format used for the restart file.

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>Database Types</i>	{ <i>catalyst</i> <i>catalyst_exodus</i> <i>cgns</i> <i>dof</i> <i>dof_exodus</i> <i>exodus</i> <i>exodusii</i> <i>exonull</i> <i>generated</i> <i>genesis</i> <i>null</i> <i>parallel_exodus</i> <i>textmesh</i> }	–

Debug Dump

Syntax

Debug Dump

Summary

Specify whether the restart system will write the restart data immediately after reading the restart data if the run is restarting. The output data can be compared with the restart input data to determine whether they match.

Decomposition Method

Syntax

Decomposition Method *{=}* { *block* | *cyclic* | *external* | *geom_kway* | *hsfc* | *kway* | *kway_geom* | *linear* | *map* | *metis_sfc* | *random* | *rcb* | *rib* | *variable* }

Summary

The decomposition algorithm to be used to partition elements to each processor in a parallel run.

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>Method</i>	{ <i>block</i> <i>cyclic</i> <i>external</i> <i>geom_kway</i> <i>hsfc</i> <i>kway</i> <i>kway_geom</i> <i>linear</i> <i>map</i> <i>metis_sfc</i> <i>random</i> <i>rcb</i> <i>rib</i> <i>variable</i> }	–

Exists

Syntax

Exists *{=}* *{abort | add_suffix | append | overwrite}*

Summary

Specify the behavior when creating this database and there is an existing file with the same name. The default behavior is “OVERWRITE” which deletes the existing file and creates a new file of the same name. “APPEND” will (if possible) append the new data to the end of the existing file. “ABORT” will print an error message and end the analysis. “ADD_SUFFIX” will add a suffix to the file name and output to that file.

Parameter	Value	Default
<i>{=}</i>	{= is}	–
<i>Option2</i>	{abort add_suffix append overwrite}	–

File Cycle Count

Syntax

File Cycle Count *{=}* *Count*

Summary

Each restart dump will be written to a separate file suffixed with A,B, . . . The count specifies how many separate files are used before the cycle repeats. For example, if “FILE CYCLE COUNT = 3” is specified, the restart dumps would be written to file-A.rs, file-B.rs, file-C.rs, file-A.rs, . . . The maximum value for the cycle count is 26.

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>Count</i>	integer	–

Input Database Name

Syntax

Input Database Name *{=}* *StreamName*

Summary

The database containing the input restart data. If this analysis is being restarted, restart data will be read from this file. See also the ‘Database’ and ‘Output Database’ commands.

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>StreamName</i>	string	–

Optional

Syntax

Optional

Summary

The database will be read if it exists, but it is not an error if there is no restart database to read for this region during a restarted analysis.

Output Database Name

Syntax

Output Database Name *{=}* *StreamName*

Summary

The database containing the output restart data. If the analysis is writing restart data, the data will be written to this file. It will be overwritten if it exists. See also the 'Database' and 'Input Database' commands.

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>StreamName</i>	string	–

Output On Signal

Syntax

Output On Signal *{=}* *{sigabrt | sigalrm | sigfpe | sighup | sigill | sigint | sigkill | sigpipe | sigquit | sigsegv | sigterm | sigusr1 | sigusr2}*

Summary

When the specified signal is raised, the output stream associated with this block will be output.

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>Signals</i>	{ sigabrt sigalrm sigfpe sighup sigill sigint sigkill sigpipe sigquit sigsegv sigterm sigusr1 sigusr2 }	–

Output Restart State

Syntax

Output Restart State *{=}* *{off | on}*

Summary

Outputs the restarted state to the new restarted results file

Description

NOTE: This command must be placed at the Sierra scope of the input file.
Allows the analyst to visualize the restarted state for debugging

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>Option</i>	{ off on }	–

Overlay Count

Syntax

Overlay Count *{=}* *Count*

Summary

Specify the number of restart outputs which will be overlaid on top of the last written step. For example, if restarts are being output every 0.1 seconds and the overlay count is specified as 2, then restart will write times 0.1 to step 1 of the database. It will then write 0.2 and 0.3 also to step 1. It will then increment the database step and write 0.4 to step 2; overlay 0.5 and 0.6 on step 2. . . At the end of the analysis, assuming it runs to completion, the database would have times 0.3, 0.6, 0.9, . . . However, if there were a problem during the analysis, the last step on the database would contain an intermediate step.

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>Count</i>	integer	–

Overwrite

Syntax

Overwrite *{=}* *{false | no | off | on | true | yes}*

Summary

(DEPRECATED, Use EXISTS) Specify whether the restart database should be overwritten if it exists. The default behavior is to overwrite unless this command is specified in the restart block and either off, false, or no is specified.

Parameter	Value	Default
<i>{=}</i>	{ = is }	–
<i>Option2</i>	{ false no off on true yes }	–

Property

Syntax

Property *PropertyName* *{=}* *PropertyValue*

Summary

Define a database property named “PropertyName” with the value “PropertyValue”. If PropertyValue consists of all digits, it will define an integer property. If PropertyValue is “true” or “yes” or “false” or “no”, it will define a logical property; otherwise it will define a string property. If PropertyName consists of multiple strings, they will be concatenated together with “_” separating the individual words. Supported properties are typically database dependent; Current properties are:

- COMPRESSION_LEVEL = [0..9]
- COMPRESSION_SHUFFLE = true|false|on|off
- FILE_TYPE = netcdf4 (forces use of netcdf-4 hdf5-based file)
- INTEGER_SIZE_DB = 4|8
- INTEGER_SIZE_API = 4|8
- LOGGING = true|false|on|off
- MAX_NAME_LENGTH = value

Parameter	Value	Default
<i>PropertyName</i>	string	–
<i>{=}</i>	{ = are is }	–
<i>PropertyValue</i>	string	–

Restart

Syntax

Restart *{=}* *{auto}*

Summary

Specify automatic restart file read.

Description

NOTE: This command must be placed at the Sierra scope of the input file.

Specify that the analysis should be restarted from the last common time on all restart databases for each Region in the analysis. In addition to this line command, each Region in the analysis (strictly, only the region(s) that will be restarted) must have a restart block specifying the database to read the restart state data.

By default, use of this command will not cause output files (e.g., results, history, heartbeat, restart) to be overwritten. Instead output files will be written with the same basename and the suffix *-s000**. Common visualization packages are written to handle this file organization gracefully in order for the user to view all results seamlessly.

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>{auto}</i>	{ auto automatic }	–

Restart Time

Syntax

Restart Time *{=}* *Time*

Summary

Specify restart file read at a specified time.

Description

NOTE: This command must be placed at the Sierra scope of the input file.

Specify the time that the analysis will be restarted. In addition to this line command, each Region in the analysis (strictly, only the region(s) that will be restarted) must have a restart block specifying the database to read the restart state data. The restart ‘time’ must be greater than zero and less than or equal to the termination time.

By default, use of this command will cause previous output files (e.g., results, history, heartbeat, restart) to be overwritten. If this command is chosen, the onus is placed on the user to ensure that previous output files are not overwritten.

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>Time</i>	real	–

Start Time

Syntax

Start Time *{=}* *Start_time*

Summary

Specify the time to start outputting results from this output request block. This time overrides all ‘at time’ and ‘at step’ specifications.

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>Start_time</i>	real	–

Synchronize Output

Syntax

Synchronize Output

Summary

In an analysis with multiple regions, it is sometimes desirable to synchronize the output of results data between the regions. This can be done by adding the SYNCHRONIZE OUTPUT command line to the results output block. If a results block has this set, then it will write output whenever a previous region writes output. The ordering of regions is based on the order in the input file, algorithmic considerations, or by solution control specifications.

Although the USE OUTPUT SCHEDULER command line can also synchronize output between regions, the SYNCHRONIZE OUTPUT command line will synchronize the output with regions where the output frequency is not under the direct control of the Sierra IO system. Examples of this are typically coupled applications where one or more of the codes are not Sierra-based applications such as Alegra and CTH. A results block with SYNCHRONIZE OUTPUT specified will also synchronize its output with the output of the external code.

The SYNCHRONIZE OUTPUT command can be used with other output scheduling commands such as time-based or step-based output specifications.

Shift To Start Time

Syntax

Shift To Start Time

Summary

The shift to start time option allows a user to shift the restart time to the start time of the current region. An example use case would be if a restart time of 0.5 is specified, but the user would like to start the simulation at time 1.0.

Termination Time

Syntax

Termination Time {=} *Final_time*

Summary

Specify the time to stop outputting results from this output request block.

Parameter	Value	Default
{=}	{= are is }	–
<i>Final_time</i>	real	–

Timestep Adjustment Interval

Syntax

Timestep Adjustment Interval {=} *Nsteps*

Summary

Specify the number of steps to ‘look ahead’ and adjust the timestep to ensure that the specified output times or simulation end time will be hit ‘exactly’.

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>Nsteps</i>	integer	–

Use Dynamic Topology Io

Syntax

Use Dynamic Topology Io

Summary

Specify that the app use IO for dynamic topology modifications where the output files are stored in a single database. Legacy file format for dynamically changing topology results in the creation of multiple files for each output on a mesh modification. This option leverages the ability of netCDF to create mesh groups within a single database and concatenate all mesh files into one. The names of each mesh group are of the form IOSS_MESH_GROUP-??? where ??? is the 1-based output index 1, 2, ..., 10, ..., 100, ... Please note that netCDF has a current limit of 65,536 groups

Use Output Scheduler

Syntax

Use Output Scheduler *Timer_name*

Summary

Associates a predefined output scheduler with this output block (results, restart, heartbeat, or history).

Parameter	Value	Default
<i>Timer_name</i>	string	–

Results Output

Scope

Average Region, Fuego Region, Input_Output Region, Particle Region

Summary

Describes the location and type of the output stream used for outputting results for the enclosing region.

begin Results Output *Label*

Additional Steps *{=}* *List_of_steps...*

Additional Times *{=}* *List_of_times...*

At Step *n* {increment / interval} {=} *m*

At Time *Dt1* {increment / interval} {=} *Dt2*

Auto Output {all / element / global / nodal} User Defined Variables [*↪*
*↪*In *UserOutputResultsList...*]

Auto Output {all / element / global / nodal} Variables

Component Separator Character {=} *Separator*

Database Name {=} *StreamName*

Database Type {=} {catalyst / catalyst_exodus / cgns / dof / dof_
*↪*exodus / exodus / exodusii / exonull / generated / genesis / null / *↪*
*↪*parallel_exodus / textmesh}

Edge *VariableList...*

Element *VariableList...*

Enable Large Ids

Exclude {=} [*ElementBlockList...*]

Exists {=} {abort / add_suffix / append / overwrite}

Face *VariableList...*

Flush Interval {=} *Option*

Global *VariableList...*

Include {=} [*ElementBlockList...*]

Nodal *VariableList...*

Node *VariableList...*

Nodeset *VariableList...*

Output Mesh {=} {exposed surface / refined / unrefined}

Output On Signal {=} {sigabrt / sigalrm / sigfpe / sighup / sigill / *↪*
*↪*sigint / sigkill / sigpipe / sigquit / sigsegv / sigterm / sigusr1 / *↪*

↪ *sigusr2*}

Overwrite *{=}* *{false | no | off | on | true | yes}*

Property *PropertyName* *{=}* *PropertyValue*

Sideset *VariableList...*

Start Time *{=}* *Start_time*

Surface *VariableList...*

Synchronize Output

Termination Time *{=}* *Final_time*

Timeseries Name *{=}* *filename*

Timestep Adjustment Interval *{=}* *Nsteps*

Title

Use Dynamic Topology Io

Use Output Scheduler *Timer_name*

begin Catalyst *Label*

end

end Results Output *Label*

Line Commands

Additional Steps

Syntax

Additional Steps *{=}* *List_of_steps...*

Summary

Additional simulation steps when output should occur.

Parameter	Value	Default
<i>{=}</i>	<i>{= are is}</i>	–
<i>List_of_steps</i>	integer. . .	–

Additional Times

Syntax

Additional Times *{=}* *List_of_times* . .

Summary

Additional simulation times when output should occur.

Parameter	Value	Default
<i>{=}</i>	{= are is}	—
<i>List_of_times</i>	real. . .	—

At Step

Syntax

At Step *n* *{increment | interval}* *{=}* *m*

Summary

Specify an output interval in terms of the internal iteration step count. The first step specifies the step count at the beginning of this interval and the second step specifies the output frequency to be used within this interval.

Parameter	Value	Default
<i>n</i>	integer	—
<i>Option</i>	{increment interval}	—
<i>{=}</i>	{= are is}	—
<i>m</i>	integer	—

At Time

Syntax

At Time *Dt1* *{increment | interval}* *{=}* *Dt2*

Summary

Specify an output interval in terms of the internal simulation time. The first time specifies the time at the beginning of this time interval and the second time specifies the output frequency to be used within this interval.

Parameter	Value	Default
<i>Dt1</i>	real	—
<i>Option</i>	{increment interval}	—
<i>{=}</i>	{= are is}	—
<i>Dt2</i>	real	—

Auto Output

Syntax

Auto Output *{all | element | global | nodal}* User Defined Variables [In *UserOutputResultsList. . .*]

Summary

Allows users to automatically output all user output defined variables for the type requested.

Parameter	Value	Default
<i>auto_output_type_3</i>	{all element global nodal}	–

Auto Output

Syntax

Auto Output *{all | element | global | nodal}* Variables

Summary

Allows users to automatically output all user output defined variables for the type requested.

Parameter	Value	Default
<i>auto_output_type_3</i>	{all element global nodal}	–

Component Separator Character

Syntax

Component Separator Character *{=} Separator*

Summary

The separator is the single character used to separate the output variable basename (e.g. “stress”) from the suffices (e.g. “xx”, “yy”) when displaying the names of the individual variable components. For example, the default separator is “_”, which results in names similar to “stress_xx”, “stress_yy”, . . . “stress_zx”. To eliminate the separator, specify an empty string (“”) or NONE.

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>Separator</i>	string	–

Database Name

Syntax

Database Name *{=} StreamName*

Summary

The base name of the database containing the output results. If the filename begins with the ‘/’ character, it is an absolute path; otherwise, the path to the

current directory will be prepended to the name. If this line is omitted, then a filename will be created from the basename of the input file with a “.e” suffix appended.

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>StreamName</i>	string	–

Database Type

Syntax

Database Type *{=}* *{catalyst | catalyst_exodus | cgns | dof | dof_exodus | exodus | exodusii | exonull | generated | genesis | null | parallel_exodus | textmesh}*

Summary

The database type/format to be used for the output results.

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>Database Type</i>	{catalyst catalyst_exodus cgns dof dof_exodus exodus exodusii exonull generated genesis null parallel_exodus textmesh}	–

Edge

Syntax

Edge *VariableList. . .*

Summary

Define the variables that should be written to the results database. If “variable” is entered, then its name will be used on the output database. If “variable as db_name” is entered, then “db_name” will be the name used on the database for the internal variable “variable”. Multiple “variable” or “variable as db_name” entries are allowed on the same line. The entities that this variable are written to can also be limited or specified with “exclude list_of_entities” or “include list_of_entities”. Edge variables are not supported for all database types.

Parameter	Value	Default
<i>VariableList</i>	string. . .	–

Element

Syntax

Element *VariableList. . .*

Summary

Define the variables that should be written to the results database. If “variable” is entered, then its name will be used on the output database. If “variable as db_name” is entered, then “db_name” will be the name used on the database for the internal variable “variable”. Multiple “variable” or “variable as db_name” entries are allowed on the same line. The entities that this variable are written to can also be limited or specified with “exclude list_of_entities” or “include list_of_entities”

Parameter	Value	Default
<i>VariableList</i>	string. . .	–

Enable Large Ids

Syntax

Enable Large Ids

Summary

Enable 64 bit entity IDs for output

Exclude

Syntax

Exclude {=} [*ElementBlockList. . .*]

Summary

Specify that the results file will only contain a subset of the element blocks in the analysis model. The element_block_list lists only the blocks which will not be output to the results database.

Parameter	Value	Default
<i>{=}</i>	{= are is }	–

Exists

Syntax

Exists {=} {*abort* | *add_suffix* | *append* | *overwrite*}

Summary

Specify the behavior when creating this database and there is an existing file with the same name. The default behavior is “OVERWRITE” which deletes the existing file and creates a new file of the same name. “APPEND” will (if possible) append the new data to the end of the existing file. “ABORT” will print an error message and end the analysis. “ADD_SUFFIX” will add a -s???? suffix where the ??? is replaced by a sequential number starting at 0002.

Parameter	Value	Default
<i>{=}</i>	{= is }	–
<i>Option2</i>	{abort add_suffix append overwrite }	–

Face

Syntax

Face *VariableList...*

Summary

Define the variables that should be written to the results database. If “variable” is entered, then its name will be used on the output database. If “variable as db_name” is entered, then “db_name” will be the name used on the database for the internal variable “variable”. Multiple “variable” or “variable as db_name” entries are allowed on the same line. The entities that this variable are written to can also be limited or specified with “exclude list_of_entities” or “include list_of_entities”. Face variables are not supported for all database types.

Parameter	Value	Default
<i>VariableList</i>	string...	–

Flush Interval

Syntax

Flush Interval *{=} Option*

Summary

The minimum time interval (in seconds) at which output will be explicitly flushed to disk. The default is 10 seconds.

Parameter	Value	Default
<i>{=}</i>	{= are is }	–
<i>Option</i>	integer	10

Global

Syntax

Global *VariableList...*

Summary

Define the global variables that should be written to the results database. If “variable” is entered, then its name will be used on the output database. If “variable as db_name” is entered, then “db_name” will be the name used on the database for the internal variable “variable”. Multiple “variable” or “variable as db_name” entries are allowed on the same line.

Parameter	Value	Default
<i>VariableList</i>	string. . .	–

Include

Syntax

Include *{=}* [*ElementBlockList. . .*]

Summary

Specify that the results file will only contain a subset of the element blocks in the analysis model. The element_block_list lists only the blocks which will be output to the results database.

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–

Nodal

Syntax

Nodal *VariableList. . .*

Summary

Define the nodal variables that should be written to the results database. If “variable” is entered, then its name will be used on the output database. If “variable as db_name” is entered, then “db_name” will be the name used on the database for the internal variable “variable”. Multiple “variable” or “variable as db_name” entries are allowed on the same line.

Parameter	Value	Default
<i>VariableList</i>	string. . .	–

Node

Syntax

Node *VariableList. . .*

Summary

Define the nodal variables that should be written to the results database. If “variable” is entered, then its name will be used on the output database. If “variable as db_name” is entered, then “db_name” will be the name used on the database for the internal variable “variable”. Multiple “variable” or “variable as db_name” entries are allowed on the same line.

Parameter	Value	Default
<i>VariableList</i>	string. . .	–

Nodeset

Syntax

Nodeset *VariableList* . .

Summary

Define the variables that should be written to the results database. If “variable” is entered, then its name will be used on the output database. If “variable as db_name” is entered, then “db_name” will be the name used on the database for the internal variable “variable”. Multiple “variable” or “variable as db_name” entries are allowed on the same line. The entities that this variable are written to can also be limited or specified with “exclude list_of_entities” or “include list_of_entities”. Nodeset variables are not supported for all database types.

Parameter	Value	Default
<i>VariableList</i>	string. . .	–

Output Mesh

Syntax

Output Mesh *{=}* {*exposed surface* | *refined* | *unrefined*}

Summary

Use this command to turn on “unrefined” as the output mesh. The default behavior is “refined”, in which field variables are output on the current mesh, which may have been refined (either uniformly or adaptively) or had its topology altered in some way (e.g., dynamic load balancing) with respect to the original mesh read from the input file. By specifying “Output Mesh = unrefined”, all output variables are output only on the original mesh objects read from the input file.

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>OutputMesh</i>	{exposed surface refined unrefined}	–

Output On Signal

Syntax

Output On Signal *{=}* {*sigabrt* | *sigalrm* | *sigfpe* | *siglrm* | *sigill* | *sigint* | *sigkill* | *sigpipe* | *sigquit* | *sigsegv* | *sigterm* | *sigusr1* | *sigusr2*}

Summary

When the specified signal is raised, the output stream associated with this block will be output.

Pa- rame- ter	Value	De- fault
<i>{=}</i>	{ = are is }	–
<i>Signals</i>	{ sigabrt sigalrm sigfpe sighup sigill sigint sigkill sigpipe sigquit sigsegv sigterm sigusr1 sigusr2 }	–

Overwrite

Syntax

Overwrite *{=}* *{false | no | off | on | true | yes}*

Summary

(DEPRECATED, Use EXISTS) Specify whether the database should be overwritten if it exists. The default behavior is to overwrite unless this command is specified in the output block and either off, false, or no is specified.

Parameter	Value	Default
<i>{=}</i>	{ = is }	–
<i>Option2</i>	{ false no off on true yes }	–

Property

Syntax

Property *PropertyName* *{=}* *PropertyValue*

Summary

Define a database property named “PropertyName” with the value “PropertyValue”. If PropertyValue consists of all digits, it will define an integer property. If PropertyValue is “true” or “yes” or “false” or “no”, it will define a logical property; otherwise it will define a string property. Supported properties are typically database dependent; Current properties are:

- COMPRESSION_LEVEL = [0..9] (off)
- COMPRESSION_SHUFFLE = true|false|on|off (off)
- FILE_TYPE = netcdf4 (forces use of netcdf-4 hdf5-based file) (netcdf3)
- INTEGER_SIZE_DB = 4|8 (4)
- INTEGER_SIZE_API = 4|8 (4)
- REAL_SIZE_DB = 4|8 (8 is default)
- LOGGING = true|false|on|off (off)
- MAX_NAME_LENGTH = value (32)

Parameter	Value	Default
<i>PropertyName</i>	string	–
<i>{=}</i>	{ = are is }	–
<i>PropertyValue</i>	string	–

Sideset

Syntax

Sideset *VariableList. . .*

Summary

Define the variables that should be written to the results database. If “variable” is entered, then its name will be used on the output database. If “variable as db_name” is entered, then “db_name” will be the name used on the database for the internal variable “variable”. Multiple “variable” or “variable as db_name” entries are allowed on the same line. The entities that this variable are written to can also be limited or specified with “exclude list_of_entities” or “include list_of_entities”. Face variables are not supported for all database types.

Parameter	Value	Default
<i>VariableList</i>	string. . .	–

Start Time

Syntax

Start Time *{=} Start_time*

Summary

Specify the time to start outputting results from this output request block. This time overrides all ‘at time’ and ‘at step’ specifications.

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>Start_time</i>	real	–

Surface

Syntax

Surface *VariableList. . .*

Summary

Define the variables that should be written to the results database. If “variable” is entered, then its name will be used on the output database. If “variable as db_name” is entered, then “db_name” will be the name used on the database for the internal variable “variable”. Multiple “variable” or “variable as db_name” entries are allowed on the same line. The entities that this variable are written to

can also be limited or specified with “exclude list_of_entities” or “include list_of_entities”. Face variables are not supported for all database types.

Parameter	Value	Default
<i>VariableList</i>	string. . .	–

Synchronize Output

Syntax

Synchronize Output

Summary

In an analysis with multiple regions, it is sometimes desirable to synchronize the output of results data between the regions. This can be done by adding the SYNCHRONIZE OUTPUT command line to the results output block. If a results block has this set, then it will write output whenever a previous region writes output. The ordering of regions is based on the order in the input file, algorithmic considerations, or by solution control specifications.

Although the USE OUTPUT SCHEDULER command line can also synchronize output between regions, the SYNCHRONIZE OUTPUT command line will synchronize the output with regions where the output frequency is not under the direct control of the Sierra IO system. Examples of this are typically coupled applications where one or more of the codes are not Sierra-based applications such as Alegria and CTH. A results block with SYNCHRONIZE OUTPUT specified will also synchronize its output with the output of the external code.

The SYNCHRONIZE OUTPUT command can be used with other output scheduling commands such as time-based or step-based output specifications.

Termination Time

Syntax

Termination Time {=} *Final_time*

Summary

Specify the time to stop outputting results from this output request block.

Parameter	Value	Default
{=}	{= are is }	–
<i>Final_time</i>	real	–

Timeseries Name

Syntax

Timeseries Name {=} *filename*

Summary

Optionally specify a filename for a timeseries file that outputs the root database filename in the order that they are written. This is useful when running on large numbers of processors with many mesh-mods that cause simple disk operations to hang.

Parameter	Value	Default
<i>{=}</i>	{= are is }	–
<i>filename</i>	string	–

Timestep Adjustment Interval

Syntax

Timestep Adjustment Interval *{=}* *Nsteps*

Summary

Specify the number of steps to ‘look ahead’ and adjust the timestep to ensure that the specified output times or simulation end time will be hit ‘exactly’.

Parameter	Value	Default
<i>{=}</i>	{= are is }	–
<i>Nsteps</i>	integer	–

Title

Syntax

Title

Summary

Specify the title to be used for this specific output block.

Use Dynamic Topology Io

Syntax

Use Dynamic Topology Io

Summary

Specify that the app use IO for dynamic topology modifications where the output files are stored in a single database. Legacy file format for dynamically changing topology results in the creation of multiple files for each output on a mesh modification. This option leverages the ability of netCDF to create mesh groups within a single database and concatenate all mesh files into one. The names of each mesh group are of the form IOSS_MESH_GROUP-??? where ??? is the 1-based output index 1, 2, ..., 10, ..., 100, ... Please note that netCDF has a current limit of 65,536 groups

Use Output Scheduler

Syntax

Use Output Scheduler *Timer_name*

Summary

Associates a predefined output scheduler with this output block (results, restart, heartbeat, or history).

Parameter	Value	Default
<i>Timer_name</i>	string	–

Solution Options

Scope

Average Region, Fuego Region

Summary

Specify information regarding the governing equations to be solved.

begin Solution Options *OptionsName*

Activate Acoustic Compressibility Algorithm

Activate Equation {*conserved_enthalpy* | *continuity* | *edc_product* |
→*enthalpy* | *mixture_fraction* | *nuclei* | *progress_variable* | *scalar_*
→*variance* | *second_mixture_fraction* | *solid_volume_fraction* | *soot* |
→*species* | *temperature* | *turbulent_dissipation* | *turbulent_frequency*
→| *turbulent_helmholtz_function* | *turbulent_kinetic_energy* | *turbulent_*
→*v2* | *volume_of_fluid* | *x_momentum* | *x_solid_momentum* | *y_momentum* | *y_*
→*solid_momentum* | *z_momentum* | *z_solid_momentum*}

Activate Full Surface Cvfem Gradient Operator For Muscl Scheme

Activate Lighthill Tensor Postprocessing

Activate Species Enthalpy Calculations

Activate Viscous Dissipation Source Term

Compute Steady Solution Using Pseudo Transient Method

Coordinate System {=} {*2d* | *3d* | *xaxi* | *yaxi*}

First Order Upwind Factor {=} *Value* [For Equation {*conserved_enthalpy*
→| *continuity* | *edc_product* | *enthalpy* | *mixture_fraction* | *nuclei*
→| *progress_variable* | *scalar_variance* | *second_mixture_fraction* |

```

→solid_volume_fraction | soot | species | temperature | turbulent_
→dissipation | turbulent_frequency | turbulent_helmholtz_function |
→turbulent_kinetic_energy | turbulent_v2 | volume_of_fluid | x_momentum_
→| x_solid_momentum | y_momentum | y_solid_momentum | z_momentum | z_
→solid_momentum} ]

```

Fix Pressure To **FixedPressure** At A Single Node

```

Hybrid Upwind Factor {=} Value [ For Equation {conserved_enthalpy_
→| continuity | edc_product | enthalpy | mixture_fraction | nuclei_
→| progress_variable | scalar_variance | second_mixture_fraction |
→solid_volume_fraction | soot | species | temperature | turbulent_
→dissipation | turbulent_frequency | turbulent_helmholtz_function |
→turbulent_kinetic_energy | turbulent_v2 | volume_of_fluid | x_momentum_
→| x_solid_momentum | y_momentum | y_solid_momentum | z_momentum | z_
→solid_momentum} ]

```

```

Hybrid Upwind Method {=} {blending | tanh} [ For Equation {conserved_
→enthalpy | continuity | edc_product | enthalpy | mixture_fraction |
→nuclei | progress_variable | scalar_variance | second_mixture_fraction_
→| solid_volume_fraction | soot | species | temperature | turbulent_
→dissipation | turbulent_frequency | turbulent_helmholtz_function |
→turbulent_kinetic_energy | turbulent_v2 | volume_of_fluid | x_momentum_
→| x_solid_momentum | y_momentum | y_solid_momentum | z_momentum | z_
→solid_momentum} ]

```

```

Hybrid Upwind Shift {=} Value [ For Equation {conserved_enthalpy_
→| continuity | edc_product | enthalpy | mixture_fraction | nuclei_
→| progress_variable | scalar_variance | second_mixture_fraction |
→solid_volume_fraction | soot | species | temperature | turbulent_
→dissipation | turbulent_frequency | turbulent_helmholtz_function |
→turbulent_kinetic_energy | turbulent_v2 | volume_of_fluid | x_momentum_
→| x_solid_momentum | y_momentum | y_solid_momentum | z_momentum | z_
→solid_momentum} ]

```

```

Hybrid Upwind Width {=} Value [ For Equation {conserved_enthalpy_
→| continuity | edc_product | enthalpy | mixture_fraction | nuclei_
→| progress_variable | scalar_variance | second_mixture_fraction |
→solid_volume_fraction | soot | species | temperature | turbulent_
→dissipation | turbulent_frequency | turbulent_helmholtz_function |
→turbulent_kinetic_energy | turbulent_v2 | volume_of_fluid | x_momentum_
→| x_solid_momentum | y_momentum | y_solid_momentum | z_momentum | z_
→solid_momentum} ]

```

Include Continuity Residual Term [With Diagnostics]

Lighthill Tensor Smoothing Iterations `{=} Number`

Maximum Number Of Continuity_Momentum Nonlinear Iterations `{=} Number`

Maximum Number Of Energy_Species Nonlinear Iterations `{=} Number`

Maximum Number Of Gas_Solid_Momentum Nonlinear Iterations `{=} Number`

Maximum Number Of Kepsilon Nonlinear Iterations `{=} Number`

Maximum Number Of Komega Nonlinear Iterations `{=} Number`

Maximum Number Of Ksgs Nonlinear Iterations `{=} Number`

Maximum Number Of Mixture Fraction Nonlinear Iterations `{=} Number`

Maximum Number Of Nonlinear Iterations `{=} Number`

Maximum Number Of Solid Phase Nonlinear Iterations `{=} Number`

Maximum Number Of Soot Nuclei Nonlinear Iterations `{=} Number`

Maximum Number Of Species Nonlinear Iterations `{=} Number`

Maximum Number Of Species_Product Nonlinear Iterations `{=} Number`

Maximum Number Of V2F Nonlinear Iterations `{=} Number`

Maximum Wall Time `{=} WallTime` Hours

Minimum Number Of Nonlinear Iterations `{=} Number`

Nonlinear Residual Norm Tolerance `{=} Tolerance` [For Equation
→{conserved_enthalpy | continuity | edc_product | enthalpy | mixture_
→fraction | nuclei | progress_variable | scalar_variance | second_
→mixture_fraction | solid_volume_fraction | soot | species | temperature_
→| turbulent_dissipation | turbulent_frequency | turbulent_helmholtz_
→function | turbulent_kinetic_energy | turbulent_v2 | volume_of_fluid_
→| x_momentum | x_solid_momentum | y_momentum | y_solid_momentum | z_
→momentum | z_solid_momentum}]

Nonlinear Stabilization Method `{=} {commutation_error | none | _
→pointwise_residual_error}` [For Equation {conserved_enthalpy | _
→continuity | edc_product | enthalpy | mixture_fraction | nuclei | _

```

→progress_variable | scalar_variance | second_mixture_fraction | solid_
→volume_fraction | soot | species | temperature | turbulent_dissipation_
→| turbulent_frequency | turbulent_helmholtz_function | turbulent_
→kinetic_energy | turbulent_v2 | volume_of_fluid | x_momentum | x_
→solid_momentum | y_momentum | y_solid_momentum | z_momentum | z_solid_
→momentum} ]

```

Omit Density Time Derivative In Continuity Equation [For **OmitSteps**_
→Steps And Blend In Over **BlendSteps** Steps]

```

Output Nonlinear Residual Field For Equation {conserved_enthalpy_
→| continuity | edc_product | enthalpy | mixture_fraction | nuclei_
→| progress_variable | scalar_variance | second_mixture_fraction |
→solid_volume_fraction | soot | species | temperature | turbulent_
→dissipation | turbulent_frequency | turbulent_helmholtz_function |
→turbulent_kinetic_energy | turbulent_v2 | volume_of_fluid | x_momentum_
→| x_solid_momentum | y_momentum | y_solid_momentum | z_momentum | z_
→solid_momentum} As ResName [ On Output Block BlockName ]

```

Periodic Constant Momentum Body Source Term {=} **ConstSrc1 ConstSrc2**_
→**ConstSrc3**

Progress Variable Source Evaluation Time {=} {latest | presolve}

```

Projection Method {=} {fourth_order | second_order | stabilized |
→zeroth_order} Smoothing [ With {characteristic | momentum | timestep}_
→Scaling ]

```

Randomize Pressure

Skip Pressure Update If Continuity Solve Fails

```

Source Term Function {=} FuncStr For Equation {conserved_enthalpy_
→| continuity | edc_product | enthalpy | mixture_fraction | nuclei_
→| progress_variable | scalar_variance | second_mixture_fraction |
→solid_volume_fraction | soot | species | temperature | turbulent_
→dissipation | turbulent_frequency | turbulent_helmholtz_function |
→turbulent_kinetic_energy | turbulent_v2 | volume_of_fluid | x_momentum_
→| x_solid_momentum | y_momentum | y_solid_momentum | z_momentum | z_
→solid_momentum} [ VariableName ]

```

```

Source Term Subroutine {=} Subroutine For Equation {conserved_enthalpy_
→| continuity | edc_product | enthalpy | mixture_fraction | nuclei_
→| progress_variable | scalar_variance | second_mixture_fraction |
→solid_volume_fraction | soot | species | temperature | turbulent_

```

```

→dissipation | turbulent_frequency | turbulent_helmholtz_function |
→turbulent_kinetic_energy | turbulent_v2 | volume_of_fluid | x_momentum
→| x_solid_momentum | y_momentum | y_solid_momentum | z_momentum | z_
→solid_momentum} [ VariableName ]

```

Stop Simulation If Peak Velocity Exceeds **MaxVel**

```

Under Relax {conserved_enthalpy | continuity | edc_product | enthalpy
→| mixture_fraction | nuclei | progress_variable | scalar_variance |
→second_mixture_fraction | solid_volume_fraction | soot | species |
→temperature | turbulent_dissipation | turbulent_frequency | turbulent_
→helmholtz_function | turbulent_kinetic_energy | turbulent_v2 | volume_
→of_fluid | x_momentum | x_solid_momentum | y_momentum | y_solid_
→momentum | z_momentum | z_solid_momentum} By Urf [ With Implicit Term ]

```

Under Relax Momentum By **Urf**

Under Relax Pressure By **Urf**

Under Relax Solid_Momentum By **Urf**

Under Relax Temperature_Extraction By **Urf**

```

Upwind Limiter {=} {minmod | none | superbee | van_albada | van_
→leer} [ For Equation {conserved_enthalpy | continuity | edc_product
→| enthalpy | mixture_fraction | nuclei | progress_variable | scalar_
→variance | second_mixture_fraction | solid_volume_fraction | soot |
→species | temperature | turbulent_dissipation | turbulent_frequency
→| turbulent_helmholtz_function | turbulent_kinetic_energy | turbulent_
→v2 | volume_of_fluid | x_momentum | x_solid_momentum | y_momentum | y_
→solid_momentum | z_momentum | z_solid_momentum} ]

```

```

Upwind Method {=} {lps | muscl | upw} [ For Equation {conserved_
→enthalpy | continuity | edc_product | enthalpy | mixture_fraction |
→nuclei | progress_variable | scalar_variance | second_mixture_fraction
→| solid_volume_fraction | soot | species | temperature | turbulent_
→dissipation | turbulent_frequency | turbulent_helmholtz_function |
→turbulent_kinetic_energy | turbulent_v2 | volume_of_fluid | x_momentum
→| x_solid_momentum | y_momentum | y_solid_momentum | z_momentum | z_
→solid_momentum} ]

```

```

Use Equation Solver SolverName For Equation {conserved_enthalpy |
→continuity | edc_product | enthalpy | mixture_fraction | nuclei |
→progress_variable | scalar_variance | second_mixture_fraction | solid_
→volume_fraction | soot | species | temperature | turbulent_dissipation

```

```

→| turbulent_frequency | turbulent_helmholtz_function | turbulent_
→kinetic_energy | turbulent_v2 | volume_of_fluid | x_momentum | x_
→solid_momentum | y_momentum | y_solid_momentum | z_momentum | z_solid_
→momentum}

```

Use External Continuity Source

Use External Energy Source

Use External Mixture_Fraction Source

Use External Momentum Source

Use External Soot_Mass_Fraction Source

Use External Species Source

Use Lumped Velocity Density Interpolation

```

Use Radiation Source From External Region [ Using Classic_
→Linearization ]

```

Use Shifted Density Iteration

```

Use Skew Symmetric Central Operator [ For Equation {conserved_enthalpy_
→| continuity | edc_product | enthalpy | mixture_fraction | nuclei_
→| progress_variable | scalar_variance | second_mixture_fraction |
→solid_volume_fraction | soot | species | temperature | turbulent_
→dissipation | turbulent_frequency | turbulent_helmholtz_function |
→turbulent_kinetic_energy | turbulent_v2 | volume_of_fluid | x_momentum_
→| x_solid_momentum | y_momentum | y_solid_momentum | z_momentum | z_
→solid_momentum} ]

```

```

begin Acoustic Transfer Output DefinitionName
end

```

```

begin Beam Radiation Boundary Specification DefinitionName
end

```

```

begin Buoyancy Model Specification BuoyModelName
end

```

```

begin Edc Model Specification EdcSpecName
end

```

```

begin Mesh Motion Specification DefinitionName
end

begin Multiphase Model Specification DefinitionName
end

begin Point Source DefinitionName
end

begin Rad Transport Spectral Model Specification DefinitionName
end

begin Radiation Transport Equation Model Specification RadModelName
end

begin Time Integration Specification TimeIntSpecName
end

begin Turbulence Model Specification TurbSpecName
end

begin Vof Model Specification DefinitionName
end

end Solution Options OptionsName

```

Line Commands

Activate Acoustic Compressibility Algorithm

Syntax

Activate Acoustic Compressibility Algorithm

Summary

Variable thermodynamic pressure

Description

This option will allow for closed system pressurization either through heat-up or inflow of fluid.

The algorithm will add the substantial derivative of pressure, $\frac{\partial p}{\partial t} + u_j \frac{\partial p}{\partial x_j}$, to the laminar or turbulent enthalpy transport equation and to the laminar temperature transport equation. Additionally, the viscous work term $u_i \frac{\partial \tau_{ij}}{\partial x_j}$ will be added to the turbulent enthalpy equation. An implicit term in the continuity solve is added through the time density derivative. As such, Cantera support is required.

The convective terms within the continuity solve are neglected.

Caveats for this model:

- 1) The Cantera material model evaluator must be used.
- 2) The initial pressure and any boundary condition pressures must be specified with respect to the datum pressure.

If a zero datum pressure is specified, then all initial and boundary pressures will be in absolute units. If this is a coupled structural simulation, then the surface traction due to this pressure will need to be counteracted with a load on the “back side” that is equivalent to the ambient pressure in absolute units.

If a non-zero datum pressure is specified, then all initial and boundary pressures will be in relative units with respect to this datum. Pressure can then be thought of as a gauge pressure with respect to the datum. The “back side” load in structural simulations must be set accordingly. (For example, if the datum is set equal to the external ambient pressure, 1 atm, and the initial pressure is set to zero, then the initial surface traction force due to pressure will be zero and no “back side” load due to the ambient pressure should be specified.)

Activate Equation

Syntax

Activate Equation *{conserved_enthalpy | continuity | edc_product | enthalpy | mixture_fraction | nuclei | progress_variable | scalar_variance | second_mixture_fraction | solid_volume_fraction | soot | species | temperature | turbulent_dissipation | turbulent_frequency | turbulent_helmholtz_function | turbulent_kinetic_energy | turbulent_v2 | volume_of_fluid | x_momentum | x_solid_momentum | y_momentum | y_solid_momentum | z_momentum | z_solid_momentum}*

Summary

Activate the specified equation.

Parameter	Value	Default
<i>Equations</i>	{conserved_enthalpy continuity edc_product enthalpy mixture_fraction nuclei progress_variable scalar_variance second_mixture_fraction solid_volume_fraction soot species temperature turbulent_dissipation turbulent_frequency turbulent_helmholtz_function turbulent_kinetic_energy turbulent_v2 volume_of_fluid x_momentum x_solid_momentum y_momentum y_solid_momentum z_momentum z_solid_momentum}	–

Activate Full Surface Cvfem Gradient Operator For Muscl Scheme

Syntax

Activate Full Surface Cvfem Gradient Operator For Muscl Scheme

Summary

Use full stencil for gradient used in MUSCL convection operator

Description

The default gradient operator for the MUSCL scheme is the edge-based stencil. This option keeps integration points at the subcontrol surface points.

Activate Lighthill Tensor Postprocessing**Syntax**

Activate Lighthill Tensor Postprocessing

Summary

Postprocesses the nodal divergence of the Lighthill tensor

Description

This calculates the nodal divergence of the Lighthill tensor, used for acoustic analysis.

Activate Species Enthalpy Calculations**Syntax**

Activate Species Enthalpy Calculations

Summary

Enables calculation of species enthalpy

Description

This forces the calculation of species enthalpy, needed primarily for coupled Fuego-Aria problems.

Activate Viscous Dissipation Source Term**Syntax**

Activate Viscous Dissipation Source Term

Summary

Add viscous dissipation source term

Description

For low speed viscous dissipation effects, this source term will provide the viscous work source term in the static enthalpy equation. This source term is a subset of the full acoustically compressible source term option, however, the substantial pressure derivative is omitted.

Compute Steady Solution Using Pseudo Transient Method**Syntax**

Compute Steady Solution Using Pseudo Transient Method

Summary

Compute a steady-state solution using the pseudo-transient method (time march to steady solution).

Description

The solution will march forward in time until either the stopping time is reached or the steady convergence criterion is met. Convergence to steady state is detected when all equations meet their nonlinear residual norm tolerances after the first nonlinear iteration, since this will only occur as the solution stops changing between time steps. The nonlinear residual norm tolerances should be set small enough to prevent false positives. Also make sure the simulation time is set to be fairly large, to prevent a premature end to the simulation before convergence is achieved.

If you are using solution control, then you also need to test for a region parameter to stop the simulation. In your PARAMETERS FOR TRANSIENT solution control block, add the line (assuming your Fuego region is called `fuego_region`):

```
CONVERGED WHEN "fuego_region.REGION_STEADY_STATE == 1"
```

Coordinate System

Syntax

Coordinate System *{=}* {2d | 3d | xaxi | yaxi}

Summary

Specify the coordinate system.

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>CoordSys</i>	{2d 3d xaxi yaxi}	3D

First Order Upwind Factor

Syntax

First Order Upwind Factor *{=}* *Value* [For Equation {*conserved_enthalpy* | *continuity* | *edc_product* | *enthalpy* | *mixture_fraction* | *nuclei* | *progress_variable* | *scalar_variance* | *second_mixture_fraction* | *solid_volume_fraction* | *soot* | *species* | *temperature* | *turbulent_dissipation* | *turbulent_frequency* | *turbulent_helmholtz_function* | *turbulent_kinetic_energy* | *turbulent_v2* | *volume_of_fluid* | *x_momentum* | *x_solid_momentum* | *y_momentum* | *y_solid_momentum* | *z_momentum* | *z_solid_momentum*}]

Summary

First-order upwind factor, $0 < x < 1$

Description

This value specifies the explicit upwind blending between pure upwind and the

chosen convection operator, e.g., UPW*(firstOrderUpwind) + (1-firstOrderUpwind)*(blendedUpwindCentral).

where UPW is the pure first order upwind value and blendedUpwindCentral is a blend between the selected upwind method and central difference operator based on the local cell Peclet number (see Hybrid Upwind Factor line command). The value can be a time dependent string function.

Values for individual equation sets may be set using optional token. Using both (in either order):

- FIRST ORDER UPWIND FACTOR = String
- FIRST ORDER UPWIND FACTOR = String FOR EQUATION Equations

Will result in the particular equation set to specified value while all others set to general value.

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>Value</i>	“string”	1.0

Fix Pressure To

Syntax

Fix Pressure To *FixedPressure* At A Single Node

Summary

Sets a dirichlet for pressure at a single arbitrary node. This is required for a well posed pressure equation if none of the boundaries specify pressure (e.g. open).

Parameter	Value	Default
<i>FixedPressure</i>	real	0.0

Hybrid Upwind Factor

Syntax

Hybrid Upwind Factor *{=}* *Value* [For Equation *{conserved_enthalpy | continuity | edc_product | enthalpy | mixture_fraction | nuclei | progress_variable | scalar_variance | second_mixture_fraction | solid_volume_fraction | soot | species | temperature | turbulent_dissipation | turbulent_frequency | turbulent_helmholtz_function | turbulent_kinetic_energy | turbulent_v2 | volume_of_fluid | x_momentum | x_solid_momentum | y_momentum | y_solid_momentum | z_momentum | z_solid_momentum}*]

Summary

Hybrid upwinding factor.

Description

The upwind schemes are blended with a centered scheme. The HYBRID UPWIND FACTOR is a multiplier against the cell Peclet number used in the switching scheme (see First Order Upwind Factor line command).

- A HYBRID UPWIND FACTOR = 0.0 results in all centered.
- A HYBRID UPWIND FACTOR = 1.0 results in default hybrid.
- A HYBRID UPWIND FACTOR >> 1.0 results in all upwind.

Values for individual equation sets may be set using optional token. Using both (in either order):

- HYBRID UPWIND FACTOR = String
- HYBRID UPWIND FACTOR = String FOR EQUATION Equations

Will result in the particular equation set to specified value while all others set to general value. The value can be specified as a time dependent string function or a constant.

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>Value</i>	“string”	1.0

Hybrid Upwind Method

Syntax

Hybrid Upwind Method *{=}* *{blending | tanh}* [For Equation *{conserved_enthalpy | continuity | edc_product | enthalpy | mixture_fraction | nuclei | progress_variable | scalar_variance | second_mixture_fraction | solid_volume_fraction | soot | species | temperature | turbulent_dissipation | turbulent_frequency | turbulent_helmholtz_function | turbulent_kinetic_energy | turbulent_v2 | volume_of_fluid | x_momentum | x_solid_momentum | y_momentum | y_solid_momentum | z_momentum | z_solid_momentum}*]

Summary

Hybridizing method between central and upwind using Peclet number

Description

BLENDING and TANH approaches are currently supported.

Function χ determines the ratio between user-chosen upwind (χ) and central ($1-\chi$) operators. The need for an upwind operator is affected by cell-Peclet number.

BLENDING uses HYBRID UPWIND FACTOR (ζ) and the function is,

$$\chi = \frac{(\zeta Pe)^2}{5 + (\zeta Pe)^2}.$$

TAHH follows hyperbolic tangent profile between χ and Pe. It uses shift p (HYBRID UPWIND SHIFT) and width w (HYBRID UPWIND WIDTH) parameters as, $\chi = \frac{1}{2} [1 + \tanh(\frac{Pe-p}{w})]$. Tanh is centered ($\chi = 0.5$) when Peclet number is at the shifting factor p. Width determines how fast χ changes with Peclet number as follows:

- $\chi = 0.5$ at $Pe=p$.
- $\chi = 0.8808$ and 0.1192 at $Pe=p+w$ and $p-w$.
- $\chi = 0.9820$ and 0.0180 at $Pe=p+2w$ and $p-2w$.
- $\chi = 0.9975$ and 0.0025 at $Pe=p+3w$ and $p-3w$.
- $\chi = 0.9997$ and 0.0003 at $Pe=p+4w$ and $p-4w$.

TANH allows users to effectively remove upwind contribution for lower Pe. In the other extreme, one can enforce user-chosen upwind at all Pe if $p < 0.0$ and $w \ll 1.0$ (ex $p=-1.0$, $w=1e-10$).

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>HybridMethod</i>	{ blending tanh }	BLENDING

Hybrid Upwind Shift

Syntax

Hybrid Upwind Shift *{=}* *Value* [For Equation *{conserved_enthalpy | continuity | edc_product | enthalpy | mixture_fraction | nuclei | progress_variable | scalar_variance | second_mixture_fraction | solid_volume_fraction | soot | species | temperature | turbulent_dissipation | turbulent_frequency | turbulent_helmholtz_function | turbulent_kinetic_energy | turbulent_v2 | volume_of_fluid | x_momentum | x_solid_momentum | y_momentum | y_solid_momentum | z_momentum | z_solid_momentum}*]

Summary

Shifting factor for TANH hybrid approach. Can be specified as a time dependent string function or a constant.

Description

(see HYBRID UPWIND METHOD description)

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>Value</i>	“string”	0.0

Hybrid Upwind Width

Syntax

Hybrid Upwind Width *{=}* *Value* [For Equation *{conserved_enthalpy | continuity | edc_product | enthalpy | mixture_fraction | nuclei | progress_variable | scalar_variance | second_mixture_fraction | solid_volume_fraction | soot | species | temperature | turbulent_dissipation | turbulent_frequency | turbulent_helmholtz_function | turbulent_kinetic_energy | turbulent_v2 | volume_of_fluid | x_momentum | x_solid_momentum | y_momentum | y_solid_momentum | z_momentum | z_solid_momentum}*]

Summary

Width factor for TANH hybrid approach

Description

Minimum value for this parameter is 1e-10. (see HYBRID UPWIND METHOD description) Can be specified as a time dependent string function or a constant value.

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>Value</i>	“string”	1.0

Include Continuity Residual Term

Syntax

Include Continuity Residual Term [With Diagnostics]

Summary

Include the continuity residual term in transport equations

Description

Continuity is not exactly satisfied during the momentum solve for variable density flows since the mass flux is lagged while the density is updated with new properties. Including the continuity error in momentum can keep the momentum prediction better behaved. The residual should be on the order of the linear solver tolerance for other equations, but including the term can also make the other solves more robust to a bad continuity solve.

This term is always included when using VOF or a deforming mesh.

Lighthill Tensor Smoothing Iterations

Syntax

Lighthill Tensor Smoothing Iterations *{=}* *Number*

Summary

Number of smoothing iterations for the divergence of the Lighthill tensor.

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>Number</i>	integer	–

Maximum Number Of Continuity_Momentum Nonlinear Iterations

Syntax

Maximum Number Of Continuity_Momentum Nonlinear Iterations *{=}* *Number*

Summary

Maximum number of nonlinear iterations to take within the momentum/continuity solve.

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>Number</i>	integer	1

Maximum Number Of Energy_Species Nonlinear Iterations

Syntax

Maximum Number Of Energy_Species Nonlinear Iterations *{=}* *Number*

Summary

Maximum number of nonlinear iterations to take within the energy-species grouping.

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>Number</i>	integer	1

Maximum Number Of Gas_Solid_Momentum Nonlinear Iterations

Syntax

Maximum Number Of Gas_Solid_Momentum Nonlinear Iterations *{=}* *Number*

Summary

Maximum number of nonlinear iterations to take within the gas/solid momentum sets of equations.

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>Number</i>	integer	1

Maximum Number Of Kepsilon Nonlinear Iterations

Syntax

Maximum Number Of Kepsilon Nonlinear Iterations *{=}* *Number*

Summary

Maximum number of nonlinear iterations to take within the k-epsilon turbulence model equations grouping.

Parameter	Value	Default
<i>{=}</i>	{= are is }	–
<i>Number</i>	integer	1

Maximum Number Of Komega Nonlinear Iterations**Syntax**

Maximum Number Of Komega Nonlinear Iterations *{=}* *Number*

Summary

Maximum number of nonlinear iterations to take within the k-omega turbulence model equations grouping.

Parameter	Value	Default
<i>{=}</i>	{= are is }	–
<i>Number</i>	integer	1

Maximum Number Of Ksgs Nonlinear Iterations**Syntax**

Maximum Number Of Ksgs Nonlinear Iterations *{=}* *Number*

Summary

Maximum number of nonlinear iterations to take within the ksgs turbulence model equations grouping.

Parameter	Value	Default
<i>{=}</i>	{= are is }	–
<i>Number</i>	integer	1

Maximum Number Of Mixture Fraction Nonlinear Iterations**Syntax**

Maximum Number Of Mixture Fraction Nonlinear Iterations *{=}* *Number*

Summary

Maximum number of nonlinear iterations to take within the mixture fraction equations grouping.

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>Number</i>	integer	1

Maximum Number Of Nonlinear Iterations

Syntax

Maximum Number Of Nonlinear Iterations *{=}* *Number*

Summary

Maximum number of nonlinear iterations to take within the time step of the Fuego region.

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>Number</i>	integer	1

Maximum Number Of Solid Phase Nonlinear Iterations

Syntax

Maximum Number Of Solid Phase Nonlinear Iterations *{=}* *Number*

Summary

Maximum number of nonlinear iterations to take within the solid momentum/continuity sets of equations.

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>Number</i>	integer	1

Maximum Number Of Soot Nuclei Nonlinear Iterations

Syntax

Maximum Number Of Soot Nuclei Nonlinear Iterations *{=}* *Number*

Summary

Maximum number of nonlinear iterations to take within the soot nuclei equations grouping.

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>Number</i>	integer	1

Maximum Number Of Species Nonlinear Iterations

Syntax

Maximum Number Of Species Nonlinear Iterations {=} *Number*

Summary

Maximum number of nonlinear iterations for the species equations. If the EDC product transport feature is active, then the SPECIES_PRODUCT nonlinear iteration count should be set instead.

Parameter	Value	Default
{=}	{= are is}	–
<i>Number</i>	integer	1

Maximum Number Of Species_Product Nonlinear Iterations

Syntax

Maximum Number Of Species_Product Nonlinear Iterations {=} *Number*

Summary

Maximum number of nonlinear iterations to take within the Species/EDC_Product grouping. This is only used if the EDC model is active and the EDC product transport feature is being used.

Parameter	Value	Default
{=}	{= are is}	–
<i>Number</i>	integer	1

Maximum Number Of V2F Nonlinear Iterations

Syntax

Maximum Number Of V2F Nonlinear Iterations {=} *Number*

Summary

Maximum number of nonlinear iterations to take within the v2f turbulence model equations grouping.

Parameter	Value	Default
{=}	{= are is}	–
<i>Number</i>	integer	1

Maximum Wall Time

Syntax

Maximum Wall Time {=} *WallTime* Hours

Summary

Specify a maximum wall time to let the simulation end gracefully and output before slurm kills it.

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>WallTime</i>	real	Infinite

Minimum Number Of Nonlinear Iterations

Syntax

Minimum Number Of Nonlinear Iterations *{=}* *Number*

Summary

Minimum number of nonlinear iterations to take within the time step of the Fuego region.

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>Number</i>	integer	1

Nonlinear Residual Norm Tolerance

Syntax

Nonlinear Residual Norm Tolerance *{=}* *Tolerance* [For Equation *{conserved_enthalpy | continuity | edc_product | enthalpy | mixture_fraction | nuclei | progress_variable | scalar_variance | second_mixture_fraction | solid_volume_fraction | soot | species | temperature | turbulent_dissipation | turbulent_frequency | turbulent_helmholtz_function | turbulent_kinetic_energy | turbulent_v2 | volume_of_fluid | x_momentum | x_solid_momentum | y_momentum | y_solid_momentum | z_momentum | z_solid_momentum}*]

Summary

Nonlinear convergence tolerance within a time step in the Fuego region.

Values for individual equation sets may be set using the optional token. Using both (in either order):

```
NONLINEAR RESIDUAL NORM TOLERANCE = {Real}
NONLINEAR RESIDUAL NORM TOLERANCE = {Real} FOR EQUATION
↪{Equations}
```

Will result in the particular equation set to specified value while all others set to general value.

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>Tolerance</i>	real	1.0e-15

Nonlinear Stabilization Method

Syntax

Nonlinear Stabilization Method {=} {*commutation_error* | *none* | *pointwise_residual_error*} [For Equation {*conserved_enthalpy* | *continuity* | *edc_product* | *enthalpy* | *mixture_fraction* | *nuclei* | *progress_variable* | *scalar_variance* | *second_mixture_fraction* | *solid_volume_fraction* | *soot* | *species* | *temperature* | *turbulent_dissipation* | *turbulent_frequency* | *turbulent_helmholtz_function* | *turbulent_kinetic_energy* | *turbulent_v2* | *volume_of_fluid* | *x_momentum* | *x_solid_momentum* | *y_momentum* | *y_solid_momentum* | *z_momentum* | *z_solid_momentum*}]

Summary

Specify a artificial viscosity stabilization; default is NONE.

Description

Values for individual equation sets may be set using optional token. Using both (in either order):

- NSO METHOD = NSOMethod
- NSO METHOD = NSOMethod FOR EQUATION Equations

Will result in the particular equation set to specified value while all others set to general value.

Parameter	Value	Default
{=}	{ = are is }	–
<i>NSOMethod</i>	{ <i>commutation_error</i> <i>none</i> <i>pointwise_residual_error</i> }	NO_NSO

Omit Density Time Derivative In Continuity Equation

Syntax

Omit Density Time Derivative In Continuity Equation [For *OmitSteps* Steps And Blend In Over *BlendSteps* Steps]

Summary

Remove density time derivative in continuity equation

Description

Remove the density time derivative from the continuity equation. This feature is required for closed boundary flows with accumulation.

The optional arguments let you omit it for a certain number of timesteps at the start of the simulation, then gradually include it over a number of steps.

Output Nonlinear Residual Field For Equation

Syntax

Output Nonlinear Residual Field For Equation {*conserved_enthalpy* | *continuity* | *edc_product* | *enthalpy* | *mixture_fraction* | *nuclei* | *progress_variable* | *scalar_variance* | *second_mixture_fraction* | *solid_volume_fraction* | *soot* |

species | temperature | turbulent_dissipation | turbulent_frequency | turbulent_helmholtz_function | turbulent_kinetic_energy | turbulent_v2 | volume_of_fluid | x_momentum | x_solid_momentum | y_momentum | y_solid_momentum | z_momentum | z_solid_momentum } As *ResName* [On Output Block *BlockName*]

Summary

Generates output of nonlinear residuals for the requested equation.

Description

Provide nonlinear residual for output for specified equation. If the optional output block name is specified, then the residual will only be written to that output block.

Parameter	Value	Default
<i>Equations</i>	{conserved_enthalpy continuity edc_product enthalpy mixture_fraction nuclei progress_variable scalar_variance second_mixture_fraction solid_volume_fraction soot species temperature turbulent_dissipation turbulent_frequency turbulent_helmholtz_function turbulent_kinetic_energy turbulent_v2 volume_of_fluid x_momentum x_solid_momentum y_momentum y_solid_momentum z_momentum z_solid_momentum}	—
<i>ResName</i>	string	—

Periodic Constant Momentum Body Source Term

Syntax

Periodic Constant Momentum Body Source Term {=} *ConstSrc1 ConstSrc2 ConstSrc3*

Summary

Add constant body force due to periodic config

Description

For periodic BCS, commonly a constant body force is applied to drive the flow. This line command allows one to provide a constant body force in three dimensions. If more complex sources are needed, the user sub source term procedure is required.

Parameter	Value	Default
{=}	{= are is}	—
<i>ConstSrc</i>	real1 real2 real3	—

Progress Variable Source Evaluation Time

Syntax

Progress Variable Source Evaluation Time *{=}* *{latest | presolve}*

Summary

Evaluation point for sequences of interdependent progress variable source terms. Either the most recently nonlinear update is used in the order in which the progress variables are solved, or the progress variable source terms are evaluated together presolve.

Parameter	Value	Default
<i>{=}</i>	{= are is }	–
<i>EvalTime</i>	{latest presolve }	–

Projection Method

Syntax

Projection Method *{=}* *{fourth_order | second_order | stabilized | zeroth_order}*
Smoothing [With *{characteristic | momentum | timestep}* Scaling]

Summary

Specify choice of projection method.

Description

The smoothing choice may include zeroth, second, or fourth order. No smoothing (zeroth) may allow pressure-velocity decoupling.

The scaling term may be specified. This scaling term is related to the factorization approximation to the inverse of the momentum matrix.

Time step scaling may show results that are sensitive to the chosen simulation dt at coarse meshes. This error should vanish as the pressure field approached a linear shape, or refinement is performed. Note that characteristic scaling also has the same error, however, its manifestation is less obvious.

The stabilized option uses a fourth order smoothing term and characteristic scaling along with an additional dt stabilizing term.

In general, the stabilized and “fourth order smoothing” timestep scaling allows for larger time steps. Characteristic scaling seems to limit CFL to below unity, presumably due to stability loss during nodal projection, i.e., splitting error is $(I - \tau A)G(\Delta P)$.

“Momentum Scaling”, uses the diagonal of the momentum equation as the scaling term. While the leading order term with this method will be similar to the timestep scaling scheme, it can sometimes offer better stability since it also includes effects from the other terms in the momentum equation.

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>ProjectionMethod</i>	{ fourth_order second_order stabilized zeroth_order }	–

Randomize Pressure

Syntax

Randomize Pressure

Summary

Set a random pressure field for initial guess

Description

Randomize the initial guess to the linear solve for pressure. The randomization is imposed after the nonlinear residual is computed.

Skip Pressure Update If Continuity Solve Fails

Syntax

Skip Pressure Update If Continuity Solve Fails

Summary

Do not update the pressure field or mdot if the continuity solve fails

Description

If the continuity solve fails the resulting pressure delta may be large or non-physical. Activating this option skips the pressure update and mdot update when the solver fails. Repeated solver failures should be watched for in the log file.

This is a beta feature.

Source Term Function

Syntax

Source Term Function *{=}* *FuncStr* For Equation *{conserved_enthalpy | continuity | edc_product | enthalpy | mixture_fraction | nuclei | progress_variable | scalar_variance | second_mixture_fraction | solid_volume_fraction | soot | species | temperature | turbulent_dissipation | turbulent_frequency | turbulent_helmholtz_function | turbulent_kinetic_energy | turbulent_v2 | volume_of_fluid | x_momentum | x_solid_momentum | y_momentum | y_solid_momentum | z_momentum | z_solid_momentum}* [*VariableName*]

Summary

Source term string function to use for the given equation. Registered variables with aliases include time (t), spatial coordinates (x,y,z), velocity (u,v,w), density (rho), and pressure (p). Additionally, any valid global variable or nodal variable

can be used with its full name. Vector variables, like mass fraction, must be indexed numerically (e.g. “mass_fraction[3]”)

The function string must be enclosed in quotes if it has spaces or commas. For example: Source Term Function for x_momentum = “min(1, 0.1*t)”

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>Func-Str</i>	“string”	–
<i>Equation</i>	{conserved_enthalpy continuity edc_product enthalpy mixture_fraction nuclei progress_variable scalar_variance second_mixture_fraction solid_volume_fraction soot species temperature turbulent_dissipation turbulent_frequency turbulent_helmholtz_function turbulent_kinetic_energy turbulent_v2 volume_of_fluid x_momentum x_solid_momentum y_momentum y_solid_momentum z_momentum z_solid_momentum}	–

Source Term Subroutine

Syntax

Source Term Subroutine *{=}* *Subroutine* For Equation *{conserved_enthalpy | continuity | edc_product | enthalpy | mixture_fraction | nuclei | progress_variable | scalar_variance | second_mixture_fraction | solid_volume_fraction | soot | species | temperature | turbulent_dissipation | turbulent_frequency | turbulent_helmholtz_function | turbulent_kinetic_energy | turbulent_v2 | volume_of_fluid | x_momentum | x_solid_momentum | y_momentum | y_solid_momentum | z_momentum | z_solid_momentum}* [*VariableName*]

Summary

Source term user subroutine for the given equation. This is often useful in verification studies where one wishes to use a manufactured solution and must provide source terms for various governing equations.

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>Sub-routine</i>	string	–
<i>Equations</i>	{conserved_enthalpy continuity edc_product enthalpy mixture_fraction nuclei progress_variable scalar_variance second_mixture_fraction solid_volume_fraction soot species temperature turbulent_dissipation turbulent_frequency turbulent_helmholtz_function turbulent_kinetic_energy turbulent_v2 volume_of_fluid x_momentum x_solid_momentum y_momentum y_solid_momentum z_momentum z_solid_momentum}	–

Stop Simulation If Peak Velocity Exceeds

Syntax

Stop Simulation If Peak Velocity Exceeds *MaxVel*

Summary

Abort the simulation if velocities get too large.

Description

By default Fuego will continue time stepping as the simulation diverges and will go until velocities overflow or solvers start returning NaN or Inf.

If you want it to stop sooner than that, you can set a peak velocity magnitude to abort at.

Parameter	Value	Default
<i>MaxVel</i>	real	infinity

Under Relax

Syntax

Under Relax *{conserved_enthalpy | continuity | edc_product | enthalpy | mixture_fraction | nuclei | progress_variable | scalar_variance | second_mixture_fraction | solid_volume_fraction | soot | species | temperature | turbulent_dissipation | turbulent_frequency | turbulent_helmholtz_function | turbulent_kinetic_energy | turbulent_v2 | volume_of_fluid | x_momentum | x_solid_momentum | y_momentum | y_solid_momentum | z_momentum | z_solid_momentum}* By *Urf* [With Implicit Term]

Summary

Under relaxation factor for the given equation.

Description

Implicit relaxation is applied to the momentum equations. Explicit relaxation is applied to the pressure update. Transport equations are relaxed explicitly unless the “WITH IMPLICI TERM” option is used.

Under-relaxation can be a constant value or a function of time (t). If the function used has spaces or commas, it should be enclosed in quotes. The value will be internally clipped between 1e-6 and 1.

Parameter	Value	Default
<i>Equations</i>	{conserved_enthalpy continuity edc_product enthalpy mixture_fraction nuclei progress_variable scalar_variance second_mixture_fraction solid_volume_fraction soot species temperature turbulent_dissipation turbulent_frequency turbulent_helmholtz_function turbulent_kinetic_energy turbulent_v2 volume_of_fluid x_momentum x_solid_momentum y_momentum y_solid_momentum z_momentum z_solid_momentum}	–
<i>Urf</i>	“string”	1.0

Under Relax Momentum By

Syntax

Under Relax Momentum By *Urf*

Summary

Under relaxation factor for the momentum equations.

Under-relaxation can be a constant value or a function of time (t). If the function used has spaces or commas, it should be enclosed in quotes. The value will be internally clipped between 1e-6 and 1.

Parameter	Value	Default
<i>Urf</i>	“string”	–

Under Relax Pressure By

Syntax

Under Relax Pressure By *Urf*

Summary

Under relaxation factor for the pressure. This is equivalent to specifying an URF on continuity, and is provided for backward compatibility.

Under-relaxation can be a constant value or a function of time (t). If the function used has spaces or commas, it should be enclosed in quotes. The value will be

internally clipped between 1e-6 and 1.

Parameter	Value	Default
<i>Urf</i>	“string”	–

Under Relax Solid_Momentum By

Syntax

Under Relax Solid_Momentum By *Urf*

Summary

Under relaxation factor for the solid-phase momentum equations.

Under-relaxation can be a constant value or a function of time (t). If the function used has spaces or commas, it should be enclosed in quotes. The value will be internally clipped between 1e-6 and 1.

Parameter	Value	Default
<i>Urf</i>	“string”	–

Under Relax Temperature_Extraction By

Syntax

Under Relax Temperature_Extraction By *Urf*

Summary

Under relaxation factor for the temperature extraction from enthalpy

Description

Relax the temperature computed from the enthalpy. This gives a temperature that is not entirely consistent with the current state (composition and enthalpy), and will destroy time-accuracy unless sufficient Picard loops are taken. However, it may be useful for steady-state computations where species and energy equations are not being coupled strongly or solved accurately.

Under-relaxation can be a constant value or a function of time (t). If the function used has spaces or commas, it should be enclosed in quotes. The value will be internally clipped between 1e-6 and 1.

Parameter	Value	Default
<i>Urf</i>	“string”	–

Upwind Limiter

Syntax

Upwind Limiter {=} {*minmod* | *none* | *superbee* | *van_albada* | *van_leer*} [For Equation {*conserved_enthalpy* | *continuity* | *edc_product* | *enthalpy* |

mixture_fraction | nuclei | progress_variable | scalar_variance | second_mixture_fraction | solid_volume_fraction | soot | species | temperature | turbulent_dissipation | turbulent_frequency | turbulent_helmholtz_function | turbulent_kinetic_energy | turbulent_v2 | volume_of_fluid | x_momentum | x_solid_momentum | y_momentum | y_solid_momentum | z_momentum | z_solid_momentum}]

Summary

Specify a limiter for convection operator; default is SUPERBEE.

Description

Limiter functions are valid only for the MUSCL scheme.

Values for individual equation sets may be set using optional token. Using both (in either order):

- UPWIND LIMITER = UpwindLimiter
- UPWIND LIMITER = UpwindLimiter FOR EQUATION Equations

Will result in the particular equation set to specified value while all others set to general value.

Note: Rotational invariance of the code is not expected while using a limiter function.

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>UpwindLimiter</i>	{minmod none superbee van_albada van_leer}	SUPERBEE

Upwind Method

Syntax

Upwind Method *{=}* *{lps | muscl | upw}* [For Equation *{conserved_enthalpy | continuity | edc_product | enthalpy | mixture_fraction | nuclei | progress_variable | scalar_variance | second_mixture_fraction | solid_volume_fraction | soot | species | temperature | turbulent_dissipation | turbulent_frequency | turbulent_helmholtz_function | turbulent_kinetic_energy | turbulent_v2 | volume_of_fluid | x_momentum | x_solid_momentum | y_momentum | y_solid_momentum | z_momentum | z_solid_momentum}]*

Summary

Upwind method for convective terms

Description

All methods are hybrid in the sense that a centered scheme is blended based on the local Peclet number.

Values for individual equation sets may be set using optional token. Using both (in either order):

- UPWIND METHOD = UpwindMethod
- UPWIND METHOD = UpwindMethod FOR EQUATION Equations

Will result in the particular equation set to specified value while all others set to general value.

Parameter	Value	Default
<i>{=}</i>	{= are is }	–
<i>UpwindMethod</i>	{lps muscl upw }	LPS

Use Equation Solver

Syntax

Use Equation Solver *SolverName* For Equation *{conserved_enthalpy | continuity | edc_product | enthalpy | mixture_fraction | nuclei | progress_variable | scalar_variance | second_mixture_fraction | solid_volume_fraction | soot | species | temperature | turbulent_dissipation | turbulent_frequency | turbulent_helmholtz_function | turbulent_kinetic_energy | turbulent_v2 | volume_of_fluid | x_momentum | x_solid_momentum | y_momentum | y_solid_momentum | z_momentum | z_solid_momentum}*

Summary

Link an equation solver to an equation set.

Description

For example, if a solver block “scalar” was created using the Tpetra package, e.g., BEGIN TPETRA EQUATION SOLVER scalar and the equation set was the u-component of momentum then the line command would be as follows: USE EQUATION SOLVER scalar FOR EQUATION X-Momentum.

This command can be omitted, and a default solver will be assigned (either the HIGH_ASPECT_CONTINUITY or SCALAR_TRANSPORT preset solvers). The default continuity solver is GMRES with the MueLu preconditioner and the default scalar transport solver is GMRES with the SGS preconditioner.

Parameter	Value	Default
<i>Solver Name</i>	string	–
<i>Equations</i>	{conserved_enthalpy continuity edc_product enthalpy mixture_fraction nuclei progress_variable scalar_variance second_mixture_fraction solid_volume_fraction soot species temperature turbulent_dissipation turbulent_frequency turbulent_helmholtz_function turbulent_kinetic_energy turbulent_v2 volume_of_fluid x_momentum x_solid_momentum y_momentum y_solid_momentum z_momentum z_solid_momentum}	–

Use External Continuity Source

Syntax

Use External Continuity Source

Summary

Add external species source term from a transfer

Description

Add source terms to the continuity equation from a nodal field transferred to this region, e.g. from a Fuego particle region. The field continuity_source will be added to the RHS of the continuity equation; this fields should have units of rate-of-change of mass per volume, so that multiplication by the control volume gives the correct source term. The transfer operation should send to the variables at state “none”.

Use External Energy Source

Syntax

Use External Energy Source

Summary

Add external energy source term from a transfer

Description

Add source terms to the temperature or enthalpy equations from a nodal field transferred to this region, e.g. from a Fuego particle region. The field energy_source will be added to the RHS of the energy equation; this fields should have units of rate-of-change of energy per volume, so that multiplication by the control volume gives the correct source term. The transfer operation should send to the variables at state “none”.

Use External Mixture_Fraction Source

Syntax

Use External Mixture_Fraction Source

Summary

Add external species source term from a transfer

Description

Add source terms to the mixture fraction equation from a nodal field transferred to this region, e.g. from a Fuego particle region. The field `mixture_fraction_source` will be added to the RHS of the mixture fraction equation; this fields should have units of rate-of-change of mass per volume, so that multiplication by the control volume gives the correct source term. The transfer operation should send to the variables at state “none”.

Use External Momentum Source**Syntax**

Use External Momentum Source

Summary

Add external momentum source terms from a transfer

Description

Add source terms to the momentum equations from a nodal field transferred to this region, e.g. from a Fuego particle region. The fields `x_momentum_source`, `y_momentum_source`, and `z_momentum_source` will be added to the RHS of the momentum equations; these fields should have units of rate-of-change of momentum per volume, so that multiplication by the control volume gives the correct source term. The transfer operation should send to the variables at state “none”.

Use External Soot_Mass_Fraction Source**Syntax**

Use External Soot_Mass_Fraction Source

Summary

Add external soot source term from a transfer

Description

Add source terms to the soot mass fraction equation from a nodal field transferred to this region, e.g. from a Fuego particle region. The field `soot_mass_fraction_source` will be added to the RHS of the soot mass fraction equation; this fields should have units of rate-of-change of mass per volume, so that multiplication by the control volume gives the correct source term. The transfer operation should send to the variables at state “none”.

Use External Species Source**Syntax**

Use External Species Source

Summary

Add external species source term from a transfer

Description

Add source terms to the species equations from a nodal field transferred to this region, e.g. from a Fuego particle region. The vector field `species_source` will be added to the RHS of the species equation; this fields should have units of rate-of-change of mass of species *i* per volume, so that multiplication by the control volume gives the correct source term. The transfer operation should send to the variables at state “none”.

Use Lumped Velocity Density Interpolation**Syntax**

Use Lumped Velocity Density Interpolation

Summary

Interpolate the density-velocity product

Description

By default the continuity equation interpolates velocity and density separately to sub-control surfaces. This option interpolates the product of density times velocity instead.

Use Radiation Source From External Region**Syntax**

Use Radiation Source From External Region [Using Classic Linearization]

Summary

Add in a source term from participating-media radiation which comes from another region through a transfer.

The USING CLASSIC LINEARIZATION optional argument is no longer used or needed, and will be removed in a future release.

Use Shifted Density Iteration**Syntax**

Use Shifted Density Iteration

Summary

Use a lagged density in the momentum solve but an updated density in the velocity projection

Description

Use a lagged density for momentum solve relative to the velocity projection similar to <https://doi.org/10.1016/j.jcp.2012.01.027>

This is a beta feature.

Use Skew Symmetric Central Operator

Syntax

Use Skew Symmetric Central Operator [For Equation {*conserved_enthalpy* | *continuity* | *edc_product* | *enthalpy* | *mixture_fraction* | *nuclei* | *progress_variable* | *scalar_variance* | *second_mixture_fraction* | *solid_volume_fraction* | *soot* | *species* | *temperature* | *turbulent_dissipation* | *turbulent_frequency* | *turbulent_helmholtz_function* | *turbulent_kinetic_energy* | *turbulent_v2* | *volume_of_fluid* | *x_momentum* | *x_solid_momentum* | *y_momentum* | *y_solid_momentum* | *z_momentum* | *z_solid_momentum*}]

Summary

The blended central operator will be skew symmetric, default is **false**.

Description

The convection operator is always blended with pure central (see hybrid factor description). For the CVFEM methodology, there is a balance between stability and accuracy. Dotting the momentum equation with velocity and summing yields the kinetic energy equation. If the convection operator is skew symmetric, then this dot product leaves something that is perfectly zero. This means that there can be no generation of kinetic energy and simulations can remain stable.

The full CVFEM stencil (27-pt on a hex mesh) is not skew symmetric. Therefore, in cases where one uses pure central (by specifying a hybrid factor of unity) there can be issues - especially on coarse meshes.

7.10 Input Output Region

Input_Output Region

Scope

Fuego Procedure

Summary

Example:

```
BEGIN INPUT TRANSFER model_name
  USE FINITE ELEMENT MODEL fred
  START TIME          is 0
  OFFSET TIME         is 1
  PERIODICITY TIME    is 10
END INPUT TRANSFER model_name
```

begin Input_Output Region *Parameter_block_name*

Create Element Field *Field_name* Of Type {*asym_tensor_03* | *complex* | *full_tensor_22* | *full_tensor_36* | *integer* | *long_integer* | *matrix_22*}

```

→| matrix_33 | real | sym_tensor_21 | sym_tensor_31 | sym_tensor_33 |
→unsigned_integer | unsigned_integer_64 | vector_2d | vector_3d} And
→Dimension Dimension [ Value {=} Number... ]

```

```

Create Nodal Field Field_name Of Type {asym_tensor_03 | complex |
→full_tensor_22 | full_tensor_36 | integer | long_integer | matrix_22
→| matrix_33 | real | sym_tensor_21 | sym_tensor_31 | sym_tensor_33 |
→unsigned_integer | unsigned_integer_64 | vector_2d | vector_3d} And
→Dimension Dimension [ Value {=} Number... ]

```

```
Fixed Time [ {=} Fixed_time ]
```

```
Offset Time {=} Period_offset_time
```

```
Periodicity Time {=} Periodicity_time
```

```
Start Time {=} Start_time
```

```
Time Interpolation Method {=} {closest | linear}
```

```
Timestep Adjustment Interval {=} Nsteps
```

```

Use Finite Element Model ModelName [ Model Coordinates Are Nodal_
→variable_name ]

```

```

begin Heartbeat Label
end

```

```

begin Heartbeat Output Label
end

```

```

begin History Output Label
end

```

```

begin Restart Data Label
end

```

```

begin Results Output Label
end

```

```
end Input_Output Region Parameter_block_name
```

Line Commands

Create Element Field

Syntax

Create Element Field *Field_name* Of Type {*asym_tensor_03* | *complex* | *full_tensor_22* | *full_tensor_36* | *integer* | *long_integer* | *matrix_22* | *matrix_33* | *real* | *sym_tensor_21* | *sym_tensor_31* | *sym_tensor_33* | *unsigned_integer* | *unsigned_integer_64* | *vector_2d* | *vector_3d*} And Dimension *Dimension* [*Value* {=} *Number...*]

Summary

Creates a Element Field name field_name on the region.

Parameter	Value	Default
<i>Field_name</i>	string	—
<i>Option</i>	{ <i>asym_tensor_03</i> <i>complex</i> <i>full_tensor_22</i> <i>full_tensor_36</i> <i>integer</i> <i>long_integer</i> <i>matrix_22</i> <i>matrix_33</i> <i>real</i> <i>sym_tensor_21</i> <i>sym_tensor_31</i> <i>sym_tensor_33</i> <i>unsigned_integer</i> <i>unsigned_integer_64</i> <i>vector_2d</i> <i>vector_3d</i> }	—
<i>Dimension</i>	integer	—

Create Nodal Field

Syntax

Create Nodal Field *Field_name* Of Type {*asym_tensor_03* | *complex* | *full_tensor_22* | *full_tensor_36* | *integer* | *long_integer* | *matrix_22* | *matrix_33* | *real* | *sym_tensor_21* | *sym_tensor_31* | *sym_tensor_33* | *unsigned_integer* | *unsigned_integer_64* | *vector_2d* | *vector_3d*} And Dimension *Dimension* [*Value* {=} *Number...*]

Summary

Creates a Nodal Field name field_name on the region.

Parameter	Value	Default
<i>Field</i>	<i>string</i>	—
<i>Option</i>	{asym_tensor_03 complex full_tensor_22 full_tensor_36 integer long_integer matrix_22 matrix_33 real sym_tensor_21 sym_tensor_31 sym_tensor_33 unsigned_integer unsigned_integer_64 vector_2d vector_3d}	—
<i>Dimension</i>	integer	—

Fixed Time

Syntax

Fixed Time [{=} *Fixed_time*]

Summary

The line specifies that the database will be read for a single, fixed time. Specifying the actual time is optional. If the time is not specified, the final time plane in the database will be read.

NOTE:

- This option takes precedence over the periodic specifications given by START TIME, PERIODICITY TIME, and OFFSET TIME.

```

if FIXED TIME is specified then
    if FIXED TIME value is given then (eg., FIXED TIME is 1.)
        DATABASE TIME = FIXED TIME
    else (eg., FIXED TIME)
        DATABASE TIME = last time in database
else
    if PERIODICITY TIME greater than 0 then
        if APPLICATION TIME less than or equal to START TIME
        then
            DATABASE TIME = APPLICATION TIME
        else
            DATABASE TIME = START TIME +
            (APPLICATION TIME - START TIME) modulo PERIODICITY
        TIME
    else
        DATABASE TIME = APPLICATION TIME
        now add OFFSET TIME to the computed DATABASE TIME

```

Offset Time

Syntax

Offset Time {=} *Period_offset_time*

Summary

This value is added to the application time to determine what database time slice to input. If OFFSET TIME were 15 than at application time 0 database time slice 15 would be read from the file and used for the initial values. At application time 1, database time slice 16 would be read.

NOTES:

- The OFFSET TIME is added in after the START TIME and PERIODICITY TIME are used.
- The FIXED TIME option takes precedence over this option.

```
if FIXED TIME is specified then
  if FIXED TIME value is given then (eg., FIXED TIME is 1.)
    DATABASE TIME = FIXED TIME
  else (eg., FIXED TIME)
    DATABASE TIME = last time in database
else
  if PERIODICITY TIME greater than 0 then
    if APPLICATION TIME less than or equal to START TIME
    then
      DATABASE TIME = APPLICATION TIME
    else
      DATABASE TIME = START TIME +
        (APPLICATION TIME - START TIME) modulo PERIODICITY
    TIME
  else
    DATABASE TIME = APPLICATION TIME
  now add OFFSET TIME to the computed DATABASE TIME
```

Parameter	Value	Default
{=}	{= are is}	–
<i>Period_offset_time</i>	real	–

Periodicity Time

Syntax

Periodicity Time {=} *Periodicity_time*

Summary

START TIME and PERIODICITY TIME taken together give the time frame from the input database to use to initialize the application values. If START TIME is 25 and PERIODICITY TIME is 10, then time slices from 25 to 35 will

be used over and over again as the application time runs from 0 to whatever. In general

DATABASE TIME **is** (APPLICATION TIME - START TIME) modulo **PERIODICITY TIME**

after the application time reaches the START TIME.

NOTES:

- The OFFSET TIME is added in after the START TIME and PERIODICITY TIME are used.
- The FIXED TIME option takes precedence over this option.

```
if FIXED TIME is specified then
  if FIXED TIME value is given then (eg., FIXED TIME is 1.)
    DATABASE TIME = FIXED TIME
  else (eg., FIXED TIME)
    DATABASE TIME = last time in database
else
  if PERIODICITY TIME greater than 0 then
    if APPLICATION TIME less than or equal to START TIME
    then
      DATABASE TIME = APPLICATION TIME
    else
      DATABASE TIME = START TIME +
        (APPLICATION TIME - START TIME) modulo PERIODICITY
    TIME
  else
    DATABASE TIME = APPLICATION TIME
  now add OFFSET TIME to the computed DATABASE TIME
```

Parameter	Value	Default
{=}	{= are is}	–
<i>Periodicity_time</i>	real	–

Start Time

Syntax

Start Time **{=}** *Start_time*

Summary

The time in which to start applying PERIODICITY TIME. If PERIODICITY TIME is not specified then START TIME is ignored.

NOTES:

- The OFFSET TIME is added in after the START TIME and PERIODICITY TIME are used.
- The FIXED TIME option takes precedence over this option.

```

if FIXED TIME is specified then
  if FIXED TIME value is given then (eg., FIXED TIME is 1.)
    DATABASE TIME = FIXED TIME
  else (eg., FIXED TIME)
    DATABASE TIME = last time in database
else
  if PERIODICITY TIME greater than 0 then
    if APPLICATION TIME less than or equal to START TIME
    then
      DATABASE TIME = APPLICATION TIME
    else
      DATABASE TIME = START TIME +
        (APPLICATION TIME - START TIME) modulo PERIODICITY
    TIME
  else
    DATABASE TIME = APPLICATION TIME
    now add OFFSET TIME to the computed DATABASE TIME

```

Parameter	Value	Default
<i>{=}</i>	{= are is}	—
<i>Start_time</i>	real	—

Time Interpolation Method

Syntax

Time Interpolation Method *{=}* *{closest | linear}*

Summary

This line specifies how interpolation in time in the database will be handled. If linear (the default option) is specified, quantities at a given point are linearly interpolated from the bounding known time points. If the closest option is selected, then the closest known time point will be taken.

Parameter	Value	Default
<i>{=}</i>	{= are is}	—
<i>Method</i>	{closest linear}	—

Timestep Adjustment Interval

Syntax

Timestep Adjustment Interval *{=}* *Nsteps*

Summary

Specify the number of steps to ‘look ahead’ and adjust the timestep to ensure that the specified output times or simulation end time will be hit ‘exactly’.

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>Nsteps</i>	integer	–

Use Finite Element Model

Syntax

Use Finite Element Model *ModelName* [Model Coordinates Are *Nodal_variable_name*]

Summary

Associates a predefined finite element model with this region.

Parameter	Value	Default
<i>ModelName</i>	string	–

Results Output

Scope

Average Region, Fuego Region, Input_Output Region, Particle Region

Summary

Describes the location and type of the output stream used for outputting results for the enclosing region.

begin Results Output *Label*

Additional Steps *{=}* *List_of_steps...*

Additional Times *{=}* *List_of_times...*

At Step *n* {*increment* | *interval*} *{=}* *m*

At Time *Dt1* {*increment* | *interval*} *{=}* *Dt2*

Auto Output {*all* | *element* | *global* | *nodal*} User Defined Variables [*↪*In *UserOutputResultsList...*]

Auto Output {*all* | *element* | *global* | *nodal*} Variables

Component Separator Character *{=}* *Separator*

Database Name {=} *StreamName*

Database Type {=} {*catalyst* | *catalyst_exodus* | *cgns* | *dof* | *dof_*
↳*exodus* | *exodus* | *exodusii* | *exonull* | *generated* | *genesis* | *null* |
↳*parallel_exodus* | *textmesh*}

Edge *VariableList...*

Element *VariableList...*

Enable Large Ids

Exclude {=} [*ElementBlockList...*]

Exists {=} {*abort* | *add_suffix* | *append* | *overwrite*}

Face *VariableList...*

Flush Interval {=} *Option*

Global *VariableList...*

Include {=} [*ElementBlockList...*]

Nodal *VariableList...*

Node *VariableList...*

Nodeset *VariableList...*

Output Mesh {=} {*exposed surface* | *refined* | *unrefined*}

Output On Signal {=} {*sigabrt* | *sigalrm* | *sigfpe* | *sighup* | *sigill* |
↳*sigint* | *sigkill* | *sigpipe* | *sigquit* | *sigsegv* | *sigterm* | *sigusr1* |
↳*sigusr2*}

Overwrite {=} {*false* | *no* | *off* | *on* | *true* | *yes*}

Property *PropertyName* {=} *PropertyValue*

Sideset *VariableList...*

Start Time {=} *Start_time*

Surface *VariableList...*

Synchronize Output

Termination Time *{=} Final_time*

Timeseries Name *{=} filename*

Timestep Adjustment Interval *{=} Nsteps*

Title

Use Dynamic Topology Io

Use Output Scheduler *Timer_name*

begin Catalyst *Label*
end

end Results Output *Label*

Line Commands

Additional Steps

Syntax

Additional Steps *{=} List_of_steps. . .*

Summary

Additional simulation steps when output should occur.

Parameter	Value	Default
<i>{=}</i>	{= are is }	—
<i>List_of_steps</i>	integer. . .	—

Additional Times

Syntax

Additional Times *{=} List_of_times. . .*

Summary

Additional simulation times when output should occur.

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>List_of_times</i>	real. . .	–

At Step

Syntax

At Step *n* *{increment | interval}* *{=}* *m*

Summary

Specify an output interval in terms of the internal iteration step count. The first step specifies the step count at the beginning of this interval and the second step specifies the output frequency to be used within this interval.

Parameter	Value	Default
<i>n</i>	integer	–
<i>Option</i>	{increment interval}	–
<i>{=}</i>	{= are is}	–
<i>m</i>	integer	–

At Time

Syntax

At Time *Dt1* *{increment | interval}* *{=}* *Dt2*

Summary

Specify an output interval in terms of the internal simulation time. The first time specifies the time at the beginning of this time interval and the second time specifies the output frequency to be used within this interval.

Parameter	Value	Default
<i>Dt1</i>	real	–
<i>Option</i>	{increment interval}	–
<i>{=}</i>	{= are is}	–
<i>Dt2</i>	real	–

Auto Output

Syntax

Auto Output *{all | element | global | nodal}* User Defined Variables [In *UserOutputResultsList. . .*]

Summary

Allows users to automatically output all user output defined variables for the type requested.

Parameter	Value	Default
<i>auto_output_type_3</i>	{all element global nodal}	–

Auto Output

Syntax

Auto Output *{all | element | global | nodal}* Variables

Summary

Allows users to automatically output all user output defined variables for the type requested.

Parameter	Value	Default
<i>auto_output_type_3</i>	{all element global nodal}	–

Component Separator Character

Syntax

Component Separator Character *{=} Separator*

Summary

The separator is the single character used to separate the output variable basename (e.g. “stress”) from the suffices (e.g. “xx”, “yy”) when displaying the names of the individual variable components. For example, the default separator is “_”, which results in names similar to “stress_xx”, “stress_yy”, . . . “stress_zx”. To eliminate the separator, specify an empty string (“”) or NONE.

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>Separator</i>	string	–

Database Name

Syntax

Database Name *{=} StreamName*

Summary

The base name of the database containing the output results. If the filename begins with the ‘/’ character, it is an absolute path; otherwise, the path to the current directory will be prepended to the name. If this line is omitted, then a filename will be created from the basename of the input file with a “.e” suffix appended.

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>StreamName</i>	string	–

Database Type

Syntax

Database Type *{=}* *{catalyst | catalyst_exodus | cgns | dof | dof_exodus | exodus | exodusii | exonull | generated | genesis | null | parallel_exodus | textmesh}*

Summary

The database type/format to be used for the output results.

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>Database Type</i>	{catalyst catalyst_exodus cgns dof dof_exodus exodus exodusii exonull generated genesis null parallel_exodus textmesh}	–

Edge

Syntax

Edge *VariableList...*

Summary

Define the variables that should be written to the results database. If “variable” is entered, then its name will be used on the output database. If “variable as db_name” is entered, then “db_name” will be the name used on the database for the internal variable “variable”. Multiple “variable” or “variable as db_name” entries are allowed on the same line. The entities that this variable are written to can also be limited or specified with “exclude list_of_entities” or “include list_of_entities”. Edge variables are not supported for all database types.

Parameter	Value	Default
<i>VariableList</i>	string...	–

Element

Syntax

Element *VariableList...*

Summary

Define the variables that should be written to the results database. If “variable” is entered, then its name will be used on the output database. If “variable as db_name” is entered, then “db_name” will be the name used on the database for the internal variable “variable”. Multiple “variable” or “variable as db_name” entries are allowed on the same line. The entities that this variable are written to can also be limited or specified with “exclude list_of_entities” or “include list_of_entities”

Parameter	Value	Default
<i>VariableList</i>	string. . .	–

Enable Large Ids

Syntax

Enable Large Ids

Summary

Enable 64 bit entity IDs for output

Exclude

Syntax

Exclude *{=}* [*ElementBlockList. . .*]

Summary

Specify that the results file will only contain a subset of the element blocks in the analysis model. The *element_block_list* lists only the blocks which will not be output to the results database.

Parameter	Value	Default
<i>{=}</i>	{= are is}	–

Exists

Syntax

Exists *{=}* *{abort | add_suffix | append | overwrite}*

Summary

Specify the behavior when creating this database and there is an existing file with the same name. The default behavior is “OVERWRITE” which deletes the existing file and creates a new file of the same name. “APPEND” will (if possible) append the new data to the end of the existing file. “ABORT” will print an error message and end the analysis. “ADD_SUFFIX” will add a -s???? suffix where the ???? is replaced by a sequential number starting at 0002.

Parameter	Value	Default
<i>{=}</i>	{= is}	–
<i>Option2</i>	{abort add_suffix append overwrite}	–

Face

Syntax

Face *VariableList. . .*

Summary

Define the variables that should be written to the results database. If “variable” is entered, then its name will be used on the output database. If “variable as db_name” is entered, then “db_name” will be the name used on the database for the internal variable “variable”. Multiple “variable” or “variable as db_name” entries are allowed on the same line. The entities that this variable are written to can also be limited or specified with “exclude list_of_entities” or “include list_of_entities”. Face variables are not supported for all database types.

Parameter	Value	Default
<i>VariableList</i>	string. . .	–

Flush Interval

Syntax

Flush Interval *{=}* *Option*

Summary

The minimum time interval (in seconds) at which output will be explicitly flushed to disk. The default is 10 seconds.

Parameter	Value	Default
<i>{=}</i>	{= are is }	–
<i>Option</i>	integer	10

Global

Syntax

Global *VariableList. . .*

Summary

Define the global variables that should be written to the results database. If “variable” is entered, then its name will be used on the output database. If “variable as db_name” is entered, then “db_name” will be the name used on the database for the internal variable “variable”. Multiple “variable” or “variable as db_name” entries are allowed on the same line.

Parameter	Value	Default
<i>VariableList</i>	string. . .	–

Include

Syntax

Include *{=}* [*ElementBlockList. . .*]

Summary

Specify that the results file will only contain a subset of the element blocks in the analysis model. The `element_block_list` lists only the blocks which will be output to the results database.

Parameter	Value	Default
<code>/=</code>	{ = are is }	–

Nodal

Syntax

Nodal *VariableList. . .*

Summary

Define the nodal variables that should be written to the results database. If “variable” is entered, then its name will be used on the output database. If “variable as db_name” is entered, then “db_name” will be the name used on the database for the internal variable “variable”. Multiple “variable” or “variable as db_name” entries are allowed on the same line.

Parameter	Value	Default
<i>VariableList</i>	string. . .	–

Node

Syntax

Node *VariableList. . .*

Summary

Define the nodal variables that should be written to the results database. If “variable” is entered, then its name will be used on the output database. If “variable as db_name” is entered, then “db_name” will be the name used on the database for the internal variable “variable”. Multiple “variable” or “variable as db_name” entries are allowed on the same line.

Parameter	Value	Default
<i>VariableList</i>	string. . .	–

Nodeset

Syntax

Nodeset *VariableList. . .*

Summary

Define the variables that should be written to the results database. If “variable” is entered, then its name will be used on the output database. If “variable as

db_name” is entered, then “db_name” will be the name used on the database for the internal variable “variable”. Multiple “variable” or “variable as db_name” entries are allowed on the same line. The entities that this variable are written to can also be limited or specified with “exclude list_of_entities” or “include list_of_entities”. Nodeset variables are not supported for all database types.

Parameter	Value	Default
<i>VariableList</i>	string. . .	–

Output Mesh

Syntax

Output Mesh *{=} {exposed surface | refined | unrefined}*

Summary

Use this command to turn on “unrefined” as the output mesh. The default behavior is “refined”, in which field variables are output on the current mesh, which may have been refined (either uniformly or adaptively) or had its topology altered in some way (e.g., dynamic load balancing) with respect to the original mesh read from the input file. By specifying “Output Mesh = unrefined”, all output variables are output only on the original mesh objects read from the input file.

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>OutputMesh</i>	{ exposed surface refined unrefined }	–

Output On Signal

Syntax

Output On Signal *{=} {sigabrt | sigalrm | sigfpe | sighup | sigill | sigint | sigkill | sigpipe | sigquit | sigsegv | sigterm | sigusr1 | sigusr2}*

Summary

When the specified signal is raised, the output stream associated with this block will be output.

Pa- rame- ter	Value	De- fault
<i>{=}</i>	{ = are is }	–
<i>Signals</i>	{ sigabrt sigalrm sigfpe sighup sigill sigint sigkill sigpipe sigquit sigsegv sigterm sigusr1 sigusr2 }	–

Overwrite

Syntax

Overwrite *{=}* *{false | no | off | on | true | yes}*

Summary

(DEPRECATED, Use EXISTS) Specify whether the database should be overwritten if it exists. The default behavior is to overwrite unless this command is specified in the output block and either off, false, or no is specified.

Parameter	Value	Default
<i>{=}</i>	{ = is }	–
<i>Option2</i>	{ false no off on true yes }	–

Property

Syntax

Property *PropertyName* *{=}* *PropertyValue*

Summary

Define a database property named “PropertyName” with the value “PropertyValue”. If PropertyValue consists of all digits, it will define an integer property. If PropertyValue is “true” or “yes” or “false” or “no”, it will define a logical property; otherwise it will define a string property. Supported properties are typically database dependent; Current properties are:

- COMPRESSION_LEVEL = [0..9] (off)
- COMPRESSION_SHUFFLE = true|false|on|off (off)
- FILE_TYPE = netcdf4 (forces use of netcdf-4 hdf5-based file) (netcdf3)
- INTEGER_SIZE_DB = 4|8 (4)
- INTEGER_SIZE_API = 4|8 (4)
- REAL_SIZE_DB = 4|8 (8 is default)
- LOGGING = true|false|on|off (off)
- MAX_NAME_LENGTH = value (32)

Parameter	Value	Default
<i>PropertyName</i>	string	–
<i>{=}</i>	{ = are is }	–
<i>PropertyValue</i>	string	–

Sideset

Syntax

Sideset *VariableList...*

Summary

Define the variables that should be written to the results database. If “variable” is entered, then its name will be used on the output database. If “variable as db_name” is entered, then “db_name” will be the name used on the database for the internal variable “variable”. Multiple “variable” or “variable as db_name” entries are allowed on the same line. The entities that this variable are written to can also be limited or specified with “exclude list_of_entities” or “include list_of_entities”. Face variables are not supported for all database types.

Parameter	Value	Default
<i>VariableList</i>	string. . .	–

Start Time

Syntax

Start Time {=} *Start_time*

Summary

Specify the time to start outputting results from this output request block. This time overrides all ‘at time’ and ‘at step’ specifications.

Parameter	Value	Default
{=}	{= are is }	–
<i>Start_time</i>	real	–

Surface

Syntax

Surface *VariableList. . .*

Summary

Define the variables that should be written to the results database. If “variable” is entered, then its name will be used on the output database. If “variable as db_name” is entered, then “db_name” will be the name used on the database for the internal variable “variable”. Multiple “variable” or “variable as db_name” entries are allowed on the same line. The entities that this variable are written to can also be limited or specified with “exclude list_of_entities” or “include list_of_entities”. Face variables are not supported for all database types.

Parameter	Value	Default
<i>VariableList</i>	string. . .	–

Synchronize Output

Syntax

Synchronize Output

Summary

In an analysis with multiple regions, it is sometimes desirable to synchronize the output of results data between the regions. This can be done by adding the SYNCHRONIZE OUTPUT command line to the results output block. If a results block has this set, then it will write output whenever a previous region writes output. The ordering of regions is based on the order in the input file, algorithmic considerations, or by solution control specifications.

Although the USE OUTPUT SCHEDULER command line can also synchronize output between regions, the SYNCHRONIZE OUTPUT command line will synchronize the output with regions where the output frequency is not under the direct control of the Sierra IO system. Examples of this are typically coupled applications where one or more of the codes are not Sierra-based applications such as Alegria and CTH. A results block with SYNCHRONIZE OUTPUT specified will also synchronize its output with the output of the external code.

The SYNCHRONIZE OUTPUT command can be used with other output scheduling commands such as time-based or step-based output specifications.

Termination Time

Syntax

Termination Time *{=}* *Final_time*

Summary

Specify the time to stop outputting results from this output request block.

Parameter	Value	Default
<i>{=}</i>	{= are is }	–
<i>Final_time</i>	real	–

Timeseries Name

Syntax

Timeseries Name *{=}* *filename*

Summary

Optionally specify a filename for a timeseries file that outputs the root database filename in the order that they are written. This is useful when running on large numbers of processors with many mesh-mods that cause simple disk operations to hang.

Parameter	Value	Default
<i>{=}</i>	{= are is }	–
<i>filename</i>	string	–

Timestep Adjustment Interval

Syntax

Timestep Adjustment Interval *{=} Nsteps*

Summary

Specify the number of steps to ‘look ahead’ and adjust the timestep to ensure that the specified output times or simulation end time will be hit ‘exactly’.

Parameter	Value	Default
<i>{=}</i>	{= are is }	–
<i>Nsteps</i>	integer	–

Title**Syntax**

Title

Summary

Specify the title to be used for this specific output block.

Use Dynamic Topology Io**Syntax**

Use Dynamic Topology Io

Summary

Specify that the app use IO for dynamic topology modifications where the output files are stored in a single database. Legacy file format for dynamically changing topology results in the creation of multiple files for each output on a mesh modification. This option leverages the ability of netCDF to create mesh groups within a single database and concatenate all mesh files into one. The names of each mesh group are of the form IOSS_MESH_GROUP-??? where ??? is the 1-based output index 1, 2, ..., 10, ..., 100, ... Please note that netCDF has a current limit of 65,536 groups

Use Output Scheduler**Syntax**

Use Output Scheduler *Timer_name*

Summary

Associates a predefined output scheduler with this output block (results, restart, heartbeat, or history).

Parameter	Value	Default
<i>Timer_name</i>	string	–

7.11 Initial Conditions

Initial Condition Block

Scope

Fuego Region

Summary

Allows the specification of piecewise constant initial conditions on any combination of volumes, surfaces and nodes.

begin Initial Condition Block *BlockName*

```
{contact_angle | edc_product | gas_volume_fraction | mixture_fraction_  
→| pressure | progress_variable | scalar_variance | second_mixture_  
→fraction | solid_volume_fraction | soot_mass_fraction | soot_nuclei_  
→mass_fraction | temperature | turbulent_dissipation | turbulent_  
→frequency | turbulent_helmholtz_function | turbulent_kinetic_energy_  
→| turbulent_v2 | volume_of_fluid | x_solid_velocity | x_velocity | y_  
→solid_velocity | y_velocity | z_solid_velocity | z_velocity} {=} Value
```

```
Copy Nodal File Variable ExodusName To Region Field AfgoName [ At Time_  
→InterpTime ]
```

```
External Field For VariableName {=} ExtFieldName [ Of Size Value ]
```

```
External Field For Progress_Variable ProgressVariableName {=}_  
→ExtFieldName
```

```
Function For {contact_angle | edc_product | gas_volume_fraction |_  
→mixture_fraction | pressure | progress_variable | scalar_variance |_  
→second_mixture_fraction | solid_volume_fraction | soot_mass_fraction_  
→| soot_nuclei_mass_fraction | temperature | turbulent_dissipation |_  
→turbulent_frequency | turbulent_helmholtz_function | turbulent_kinetic_  
→energy | turbulent_v2 | volume_of_fluid | x_solid_velocity | x_velocity_  
→| y_solid_velocity | y_velocity | z_solid_velocity | z_velocity} {=}_  
→FuncName In The {t | x | y | z} Direction
```

```
Function For Mass_Fraction {=} FuncName In The {t | x | y | z}_  
→Direction
```

```
Function For Mole_Fraction {=} FuncName In The {t | x | y | z}_  
→Direction
```

Ignite

```

    Inline Function For {contact_angle | edc_product | gas_volume_fraction |
    ↳| mixture_fraction | pressure | progress_variable | scalar_variance |
    ↳second_mixture_fraction | solid_volume_fraction | soot_mass_fraction |
    ↳| soot_nuclei_mass_fraction | temperature | turbulent_dissipation |
    ↳turbulent_frequency | turbulent_helmholtz_function | turbulent_kinetic_
    ↳energy | turbulent_v2 | volume_of_fluid | x_solid_velocity | x_velocity |
    ↳| y_solid_velocity | y_velocity | z_solid_velocity | z_velocity} {=}
    ↳FuncStr

```

Integer Data For Subroutine *SubName* {=} *Values...*

Mass_Fraction *Species* {=} *y0*

Mole_Fraction *Species* {=} *y0*

Progress_Variable *ProgressVariableName* {=} *Value*

Real Data For Subroutine *SubName* {=} *Values...*

Solid Object {=} *t0*

```

    Subroutine For {contact_angle | edc_product | gas_volume_fraction |
    ↳mixture_fraction | pressure | progress_variable | scalar_variance |
    ↳second_mixture_fraction | solid_volume_fraction | soot_mass_fraction |
    ↳| soot_nuclei_mass_fraction | temperature | turbulent_dissipation |
    ↳turbulent_frequency | turbulent_helmholtz_function | turbulent_kinetic_
    ↳energy | turbulent_v2 | volume_of_fluid | x_solid_velocity | x_velocity |
    ↳| y_solid_velocity | y_velocity | z_solid_velocity | z_velocity} {=}
    ↳SubName

```

Subroutine For Mass_Fraction {=} *Sub*

Subroutine For Mole_Fraction {=} *Sub*

Update Enthalpy And Density

```

    Vof Shape For {contact_angle | edc_product | gas_volume_fraction |
    ↳mixture_fraction | pressure | progress_variable | scalar_variance |
    ↳second_mixture_fraction | solid_volume_fraction | soot_mass_fraction |
    ↳| soot_nuclei_mass_fraction | temperature | turbulent_dissipation |
    ↳turbulent_frequency | turbulent_helmholtz_function | turbulent_kinetic_
    ↳energy | turbulent_v2 | volume_of_fluid | x_solid_velocity | x_velocity |
    ↳| y_solid_velocity | y_velocity | z_solid_velocity | z_velocity} {=}
    ↳ShapeName ShapeArgs...

```

Volume {=} *Blocks...*

end Initial Condition Block *BlockName*

Line Commands

Primitivevariable

Syntax

Primitivevariable {*contact_angle* | *edc_product* | *gas_volume_fraction* | *mixture_fraction* | *pressure* | *progress_variable* | *scalar_variance* | *second_mixture_fraction* | *solid_volume_fraction* | *soot_mass_fraction* | *soot_nuclei_mass_fraction* | *temperature* | *turbulent_dissipation* | *turbulent_frequency* | *turbulent_helmholtz_function* | *turbulent_kinetic_energy* | *turbulent_v2* | *volume_of_fluid* | *x_solid_velocity* | *x_velocity* | *y_solid_velocity* | *y_velocity* | *z_solid_velocity* | *z_velocity*} {=} *Value*

Summary

Assign initial condition to the specified variable. Specified value can be a constant or a string function of time, space, and global variables.

The function string must be enclosed in quotes if it contains spaces or commas.
For example: *x_velocity* = “min(1, 0.1*x + y)”

Parameter	Value	Default
<i>Primitive-Variable</i>	{ <i>contact_angle</i> <i>edc_product</i> <i>gas_volume_fraction</i> <i>mixture_fraction</i> <i>pressure</i> <i>progress_variable</i> <i>scalar_variance</i> <i>second_mixture_fraction</i> <i>solid_volume_fraction</i> <i>soot_mass_fraction</i> <i>soot_nuclei_mass_fraction</i> <i>temperature</i> <i>turbulent_dissipation</i> <i>turbulent_frequency</i> <i>turbulent_helmholtz_function</i> <i>turbulent_kinetic_energy</i> <i>turbulent_v2</i> <i>volume_of_fluid</i> <i>x_solid_velocity</i> <i>x_velocity</i> <i>y_solid_velocity</i> <i>y_velocity</i> <i>z_solid_velocity</i> <i>z_velocity</i> }	–
{=}	{= are is}	–
<i>Value</i>	“string”	–

Copy Nodal File Variable

Syntax

Copy Nodal File Variable *ExodusName* To Region Field *AfgoName* [At Time *InterpTime*]

Summary

Solid geometry to which this initial condition applies.

Parameter	Value	Default
<i>ExodusName</i>	string	–
<i>AfgoName</i>	string	–

External Field For

Syntax

External Field For *VariableName* {=} *ExtFieldName* [Of Size *Value*]

Summary

Name of the field that is to be transferred in.

Parameter	Value	Default
<i>VariableName</i>	string	–
{=}	{ = are is }	–
<i>ExtFieldName</i>	string	–

External Field For Progress_Variable

Syntax

External Field For Progress_Variable *ProgressVariableName* {=} *ExtFieldName*

Summary

Name of the progress variable field that is to be transferred in.

Parameter	Value	Default
<i>ProgressVariableName</i>	string	–
{=}	{ = are is }	–
<i>ExtFieldName</i>	string	–

Function For

Syntax

Function For {*contact_angle* | *edc_product* | *gas_volume_fraction* | *mixture_fraction* | *pressure* | *progress_variable* | *scalar_variance* | *second_mixture_fraction* | *solid_volume_fraction* | *soot_mass_fraction* | *soot_nuclei_mass_fraction* | *temperature* | *turbulent_dissipation* | *turbulent_frequency* | *turbulent_helmholtz_function* | *turbulent_kinetic_energy* | *turbulent_v2* | *volume_of_fluid* | *x_solid_velocity* | *x_velocity* | *y_solid_velocity* | *y_velocity* | *z_solid_velocity* | *z_velocity*} {=} *FuncName* In The {*t* | *x* | *y* | *z*} Direction

Summary

Name of function with which to set the initial condition for the given variable in the given direction.

Parameter	Value	Default
<i>Primitive-Variable</i>	{contact_angle edc_product gas_volume_fraction mixture_fraction pressure progress_variable scalar_variance second_mixture_fraction solid_volume_fraction soot_mass_fraction soot_nuclei_mass_fraction temperature turbulent_dissipation turbulent_frequency turbulent_helmholtz_function turbulent_kinetic_energy turbulent_v2 volume_of_fluid x_solid_velocity x_velocity y_solid_velocity y_velocity z_solid_velocity z_velocity}	–
<i>{=}</i>	{= are is}	–
<i>FuncName</i>	string	–
<i>Direction</i>	{t x y z}	–

Function For Mass_Fraction

Syntax

Function For Mass_Fraction *{=}* *FuncName* In The *{t | x | y | z}* Direction

Summary

Names of functions with which to set the initial mass fractions for the given species in the given direction.

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>FuncName</i>	string	–
<i>Direction</i>	{t x y z}	–

Function For Mole_Fraction

Syntax

Function For Mole_Fraction *{=}* *FuncName* In The *{t | x | y | z}* Direction

Summary

Name of the mole fraction function to use for all species in the given direction.
Note that this must be a Multicolumn function

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>FuncName</i>	string	–
<i>Direction</i>	{t x y z}	–

Ignite

Syntax

Ignite

Summary

Flag this block for combustion ignition

Inline Function For

Syntax

Inline Function For *{contact_angle | edc_product | gas_volume_fraction | mixture_fraction | pressure | progress_variable | scalar_variance | second_mixture_fraction | solid_volume_fraction | soot_mass_fraction | soot_nuclei_mass_fraction | temperature | turbulent_dissipation | turbulent_frequency | turbulent_helmholtz_function | turbulent_kinetic_energy | turbulent_v2 | volume_of_fluid | x_solid_velocity | x_velocity | y_solid_velocity | y_velocity | z_solid_velocity | z_velocity}* {=} *FuncStr*

Summary

warning{This command is deprecated. Regular ICs now support using string functions directly}

Parameter	Value	Default
<i>Primitive-Variable</i>	{contact_angle edc_product gas_volume_fraction mixture_fraction pressure progress_variable scalar_variance second_mixture_fraction solid_volume_fraction soot_mass_fraction soot_nuclei_mass_fraction temperature turbulent_dissipation turbulent_frequency turbulent_helmholtz_function turbulent_kinetic_energy turbulent_v2 volume_of_fluid x_solid_velocity x_velocity y_solid_velocity y_velocity z_solid_velocity z_velocity}	—
<i>{=}</i>	{= are is}	—
<i>FuncStr</i>	“string”	—

Integer Data For Subroutine

Syntax

Integer Data For Subroutine *SubName* {=} *Values...*

Summary

List of integer data values to be passed down in to the user subroutine. These values may be changed by the user subroutine.

Parameter	Value	Default
<i>SubName</i>	string	–
<i>{=}</i>	{= are is}	–
<i>Values</i>	integer. . .	–

Mass_Fraction

Syntax

Mass_Fraction *Species* *{=}* *y0*

Summary

Set the initial mass fraction. Specified value can be a constant or a string function of time, space, and global variables.

Parameter	Value	Default
<i>Species</i>	string	–
<i>{=}</i>	{= are is}	–
<i>y0</i>	“string”	–

Mole_Fraction

Syntax

Mole_Fraction *Species* *{=}* *y0*

Summary

Set the initial mole fractions to a constant value.

Parameter	Value	Default
<i>Species</i>	string	–
<i>{=}</i>	{= are is}	–
<i>y0</i>	“string”	–

Progress_Variable

Syntax

Progress_Variable *ProgressVariableName* *{=}* *Value*

Summary

Set the initial progress variable. Specified value can be a constant or a string function of time, space, and global variables.

Parameter	Value	Default
<i>ProgressVariableName</i>	string	–
<i>{=}</i>	{= are is}	–
<i>Value</i>	“string”	–

Real Data For Subroutine

Syntax

Real Data For Subroutine *SubName* {=} *Values* . .

Summary

List of real data values to be passed down in to the user subroutine. These values may be changed by the user subroutine.

Parameter	Value	Default
<i>SubName</i>	string	–
{=}	{= are is}	–
<i>Values</i>	real. . .	–

Solid Object

Syntax

Solid Object {=} *t0*

Summary

Solid geometry to which this initial condition applies.

Parameter	Value	Default
{=}	{= are is}	–
<i>t0</i>	string	–

Subroutine For

Syntax

Subroutine For {*contact_angle* | *edc_product* | *gas_volume_fraction* | *mixture_fraction* | *pressure* | *progress_variable* | *scalar_variance* | *second_mixture_fraction* | *solid_volume_fraction* | *soot_mass_fraction* | *soot_nuclei_mass_fraction* | *temperature* | *turbulent_dissipation* | *turbulent_frequency* | *turbulent_helmholtz_function* | *turbulent_kinetic_energy* | *turbulent_v2* | *volume_of_fluid* | *x_solid_velocity* | *x_velocity* | *y_solid_velocity* | *y_velocity* | *z_solid_velocity* | *z_velocity*} {=} *SubName*

Summary

Set the initial condition on the specified variable using a subroutine

Parameter	Value	Default
<i>Primitive-Variable</i>	{contact_angle edc_product gas_volume_fraction mixture_fraction pressure progress_variable scalar_variance second_mixture_fraction solid_volume_fraction soot_mass_fraction soot_nuclei_mass_fraction temperature turbulent_dissipation turbulent_frequency turbulent_helmholtz_function turbulent_kinetic_energy turbulent_v2 volume_of_fluid x_solid_velocity x_velocity y_solid_velocity y_velocity z_solid_velocity z_velocity}	–
<i>{=}</i>	{= are is}	–
<i>Sub-Name</i>	string	–

Subroutine For Mass_Fraction

Syntax

Subroutine For Mass_Fraction *{=}* *Sub*

Summary

Set the initial mass fractions using a subroutine.

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>Sub</i>	string	–

Subroutine For Mole_Fraction

Syntax

Subroutine For Mole_Fraction *{=}* *Sub*

Summary

Set the initial mole fractions using a subroutine.

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>Sub</i>	string	–

Update Enthalpy And Density

Syntax

Update Enthalpy And Density

Summary

Flag this block for updating enthalpy after replacing T or Y

Vof Shape For

Syntax

Vof Shape For {*contact_angle* | *edc_product* | *gas_volume_fraction* | *mixture_fraction* | *pressure* | *progress_variable* | *scalar_variance* | *second_mixture_fraction* | *solid_volume_fraction* | *soot_mass_fraction* | *soot_nuclei_mass_fraction* | *temperature* | *turbulent_dissipation* | *turbulent_frequency* | *turbulent_helmholtz_function* | *turbulent_kinetic_energy* | *turbulent_v2* | *volume_of_fluid* | *x_solid_velocity* | *x_velocity* | *y_solid_velocity* | *y_velocity* | *z_solid_velocity* | *z_velocity*} {=} *ShapeName* *ShapeArgs*...

Summary

Set the smooth VOF initial conditions using basic shapes. Multiple shapes can be specified, and where they intersect will be a union operation.

Parameter	Value	Default
<i>Primitive-Variable</i>	{ <i>contact_angle</i> <i>edc_product</i> <i>gas_volume_fraction</i> <i>mixture_fraction</i> <i>pressure</i> <i>progress_variable</i> <i>scalar_variance</i> <i>second_mixture_fraction</i> <i>solid_volume_fraction</i> <i>soot_mass_fraction</i> <i>soot_nuclei_mass_fraction</i> <i>temperature</i> <i>turbulent_dissipation</i> <i>turbulent_frequency</i> <i>turbulent_helmholtz_function</i> <i>turbulent_kinetic_energy</i> <i>turbulent_v2</i> <i>volume_of_fluid</i> <i>x_solid_velocity</i> <i>x_velocity</i> <i>y_solid_velocity</i> <i>y_velocity</i> <i>z_solid_velocity</i> <i>z_velocity</i> }	—
{=}	{= are is}	—
<i>ShapeName</i>	string	—
<i>ShapeArgs</i>	string...	—

Volume

Syntax

Volume {=} *Blocks*...

Summary

Element Block name(s) to which this initial condition applies.

For multiple blocks, you can either use multiple VOLUME = X lines or specify all the blocks in one line as VOLUME = X Y Z.

Parameter	Value	Default
{=}	{= are is}	—
<i>Blocks</i>	string...	—

7.12 Boundary Conditions

Fixed Boundary Condition On Surface

Scope

Fuego Region

Summary

Defines a fixed boundary condition on a named surface of the mesh.

begin Fixed Boundary Condition On Surface *Surfacename*

```
{contact_angle | edc_product | gas_volume_fraction | mixture_fraction |  
→| pressure | progress_variable | scalar_variance | second_mixture_  
→fraction | solid_volume_fraction | soot_mass_fraction | soot_nuclei_  
→mass_fraction | temperature | turbulent_dissipation | turbulent_  
→frequency | turbulent_helmholtz_function | turbulent_kinetic_energy_  
→| turbulent_v2 | volume_of_fluid | x_solid_velocity | x_velocity | y_  
→solid_velocity | y_velocity | z_solid_velocity | z_velocity} {=} Value
```

Emissivity Spectral File Name {=} *Name*

```
External Field For VariableName [{of} Species] {=} ExtFieldName [ Of_  
→Size Value With Multiplier {=} Multiplier ]
```

Emissivity {=} *value*

```
Function For {contact_angle | edc_product | gas_volume_fraction |  
→mixture_fraction | pressure | progress_variable | scalar_variance |  
→second_mixture_fraction | solid_volume_fraction | soot_mass_fraction_  
→| soot_nuclei_mass_fraction | temperature | turbulent_dissipation |  
→turbulent_frequency | turbulent_helmholtz_function | turbulent_kinetic_  
→energy | turbulent_v2 | volume_of_fluid | x_solid_velocity | x_velocity_  
→| y_solid_velocity | y_velocity | z_solid_velocity | z_velocity} {=}_  
→FuncName [ In The {t | x | y | z} Direction ]
```

```
Function For Emissivity {=} functionName [ In The {t | x | y | z}_  
→Direction ]
```

```
Function For Mass_Fraction {=} FuncName In The {t | x | y | z}_  
→Direction
```

```
Function For Mole_Fraction {=} FuncName In The {t | x | y | z}_  
→Direction
```

Function For Radiation Boundary Temperature {=} *functionName*

Function For Radiation Environment Temperature {=} *functionName*

Function For Transmissivity {=} *functionName* [In The {t | x | y | z}
→Direction]

Function For Transparent Band Emissivity {=} *functionName* [In The {t
→| x | y | z} Direction]

Inline Function For {contact_angle | edc_product | gas_volume_fraction
→| mixture_fraction | pressure | progress_variable | scalar_variance |
→second_mixture_fraction | solid_volume_fraction | soot_mass_fraction
→| soot_nuclei_mass_fraction | temperature | turbulent_dissipation |
→turbulent_frequency | turbulent_helmholtz_function | turbulent_kinetic
→energy | turbulent_v2 | volume_of_fluid | x_solid_velocity | x_velocity
→| y_solid_velocity | y_velocity | z_solid_velocity | z_velocity} {=}
→FuncStr

Integer Data For Subroutine *SubName* {=} *Values...*

Mass_Fraction *Species* {=} *Mass fraction*

Mole_Fraction *Species* {=} *Mole fraction*

Postprocess *FluxType* Flux Of {conserved_enthalpy | continuity | edc_
→product | enthalpy | mixture_fraction | nuclei | progress_variable
→| scalar_variance | second_mixture_fraction | solid_volume_fraction
→| soot | species | temperature | turbulent_dissipation | turbulent_
→frequency | turbulent_helmholtz_function | turbulent_kinetic_energy
→| turbulent_v2 | volume_of_fluid | x_momentum | x_solid_momentum |
→y_momentum | y_solid_momentum | z_momentum | z_solid_momentum} [As
→aliases...]

Progress_Variable *ProgressVariableName* {=} *Value*

Real Data For Subroutine *SubName* {=} *Values...*

Radiation Boundary Temperature {=} *value*

Radiation Boundary Temperature Field {=} *FieldName*

Radiation Environment Temperature {=} *value*

Radiation Environment Temperature Field {=} *FieldName*

```

Subroutine For {contact_angle | edc_product | gas_volume_fraction |
↪mixture_fraction | pressure | progress_variable | scalar_variance |
↪second_mixture_fraction | solid_volume_fraction | soot_mass_fraction |
↪| soot_nuclei_mass_fraction | temperature | turbulent_dissipation |
↪turbulent_frequency | turbulent_helmholtz_function | turbulent_kinetic_
↪energy | turbulent_v2 | volume_of_fluid | x_solid_velocity | x_velocity |
↪| y_solid_velocity | y_velocity | z_solid_velocity | z_velocity} {=}
↪Subroutine

```

```

Subroutine For Emissivity {=} subroutineName

```

```

Subroutine For Mass_Fraction {=} Subroutine

```

```

Subroutine For Mole_Fraction {=} Subroutine

```

```

Subroutine For Radiation Boundary Temperature {=} subroutineName

```

```

Subroutine For Radiation Environment Temperature {=} subroutineName

```

```

Subroutine For Transmissivity {=} subroutineName

```

```

Subroutine For Transparent Band Emissivity {=} subroutineName

```

```

Transparent Band Emissivity {=} value

```

```

Transmissivity {=} value

```

```

end Fixed Boundary Condition On Surface Surfacename

```

Line Commands

Primitivevariable

Syntax

```

Primitivevariable {contact_angle | edc_product | gas_volume_fraction |
mixture_fraction | pressure | progress_variable | scalar_variance |
second_mixture_fraction | solid_volume_fraction | soot_mass_fraction |
soot_nuclei_mass_fraction | temperature | turbulent_dissipation |
turbulent_frequency | turbulent_helmholtz_function | turbulent_kinetic_energy |
turbulent_v2 | volume_of_fluid | x_solid_velocity | x_velocity | y_solid_velocity |
y_velocity | z_solid_velocity | z_velocity} {=} Value

```

Summary

Value for the specified variable (in consistent units). Value can be a constant or a

string function of position (x,y,z), time (t), and any global variable.

The function string must be enclosed in quotes if it has spaces or commas. For example: x_velocity = “min(1, 0.1*t)”

Parameter	Value	Default
<i>Primitive-Variable</i>	{contact_angle edc_product gas_volume_fraction mixture_fraction pressure progress_variable scalar_variance second_mixture_fraction solid_volume_fraction soot_mass_fraction soot_nuclei_mass_fraction temperature turbulent_dissipation turbulent_frequency turbulent_helmholtz_function turbulent_kinetic_energy turbulent_v2 volume_of_fluid x_solid_velocity x_velocity y_solid_velocity y_velocity z_solid_velocity z_velocity}	–
<i>{=}</i>	{= are is}	–
<i>Value</i>	“string”	–

Emissivity Spectral File Name

Syntax

Emissivity Spectral File Name *{=}* *Name*

Summary

Specify the file name for defining spectral radiation properties on a surface.

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>Name</i>	string	–

External Field For

Syntax

External Field For *VariableName* [*{of}* *Species*] *{=}* *ExtFieldName* [Of Size *Value* With Multiplier *{=}* *Multiplier*]

Summary

Name of the field that is to be transferred in. For species equation, specify variable as “mass fraction of” followed by the species name for transfers associated with separated individual species field names.

Parameter	Value	Default
<i>VariableName</i>	string	–
<i>Species</i>	string	–
<i>{=}</i>	{= are is}	–
<i>ExtFieldName</i>	string	–

Emissivity

Syntax

Emissivity **{=}** *value*

Summary

Constant value for emissivity variable.

Parameter	Value	Default
{=}	{= are is}	–
<i>value</i>	“string”	–

Function For

Syntax

Function For *{contact_angle | edc_product | gas_volume_fraction | mixture_fraction | pressure | progress_variable | scalar_variance | second_mixture_fraction | solid_volume_fraction | soot_mass_fraction | soot_nuclei_mass_fraction | temperature | turbulent_dissipation | turbulent_frequency | turbulent_helmholtz_function | turbulent_kinetic_energy | turbulent_v2 | volume_of_fluid | x_solid_velocity | x_velocity | y_solid_velocity | y_velocity | z_solid_velocity | z_velocity}* **{=}** *FuncName* [In The *{t | x | y | z}* Direction]

Summary

Name of function to use for the given variable, in the specified direction.

Parameter	Value	Default
<i>Primitive-Variable</i>	{contact_angle edc_product gas_volume_fraction mixture_fraction pressure progress_variable scalar_variance second_mixture_fraction solid_volume_fraction soot_mass_fraction soot_nuclei_mass_fraction temperature turbulent_dissipation turbulent_frequency turbulent_helmholtz_function turbulent_kinetic_energy turbulent_v2 volume_of_fluid x_solid_velocity x_velocity y_solid_velocity y_velocity z_solid_velocity z_velocity}	–
{=}	{= are is}	–
<i>FuncName</i>	string	–

Function For Emissivity

Syntax

Function For Emissivity **{=}** *functionName* [In The *{t | x | y | z}* Direction]

Summary

Name of the function to use for the emissivity in the given direction (default direction = time).

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>functionName</i>	string	–

Function For Mass_Fraction

Syntax

Function For Mass_Fraction *{=}* *FuncName* In The *{t | x | y | z}* Direction

Summary

Name of the mass fraction function to use for all species in the given direction.
Note that this must be a Multicolumn function

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>FuncName</i>	string	–
<i>Direction</i>	{ t x y z }	–

Function For Mole_Fraction

Syntax

Function For Mole_Fraction *{=}* *FuncName* In The *{t | x | y | z}* Direction

Summary

Name of the mass fraction function to use for all species in the given direction.
Note that this must be a Multicolumn function

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>FuncName</i>	string	–
<i>Direction</i>	{ t x y z }	–

Function For Radiation Boundary Temperature

Syntax

Function For Radiation Boundary Temperature *{=}* *functionName*

Summary

Name of function to use for the radiation_boundary_temperature variable

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>functionName</i>	string	–

Function For Radiation Environment Temperature

Syntax

Function For Radiation Environment Temperature *{=}* *functionName*

Summary

Name of function to use for the radiation_environment_temperature variable

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>functionName</i>	string	–

Function For Transmissivity

Syntax

Function For Transmissivity *{=}* *functionName* [In The *{t | x | y | z}* Direction]

Summary

Name of function to use for the transmissivity variable

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>functionName</i>	string	–

Function For Transparent Band Emissivity

Syntax

Function For Transparent Band Emissivity *{=}* *functionName* [In The *{t | x | y | z}* Direction]

Summary

Name of function to use for the transparent emissivity band for spectral emissivity on surfaces.

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>functionName</i>	string	–

Inline Function For

Syntax

Inline Function For *{contact_angle | edc_product | gas_volume_fraction |*

*mixture_fraction | pressure | progress_variable | scalar_variance |
second_mixture_fraction | solid_volume_fraction | soot_mass_fraction |
soot_nuclei_mass_fraction | temperature | turbulent_dissipation |
turbulent_frequency | turbulent_helmholtz_function | turbulent_kinetic_energy |
turbulent_v2 | volume_of_fluid | x_solid_velocity | x_velocity | y_solid_velocity |
y_velocity | z_solid_velocity | z_velocity} {=} **FuncStr***

Summary

warning{This command is deprecated. Regular BCs now support using string functions directly}

Parameter	Value	Default
<i>Primitive-Variable</i>	{contact_angle edc_product gas_volume_fraction mixture_fraction pressure progress_variable scalar_variance second_mixture_fraction solid_volume_fraction soot_mass_fraction soot_nuclei_mass_fraction temperature turbulent_dissipation turbulent_frequency turbulent_helmholtz_function turbulent_kinetic_energy turbulent_v2 volume_of_fluid x_solid_velocity x_velocity y_solid_velocity y_velocity z_solid_velocity z_velocity}	—
<i>{=}</i>	{= are is}	—
<i>FuncStr</i>	“string”	—

Integer Data For Subroutine

Syntax

Integer Data For Subroutine *SubName {=} Values...*

Summary

List of integer data values to be passed down in to the user subroutine. These values may be changed by the user subroutine.

Parameter	Value	Default
<i>SubName</i>	string	—
<i>{=}</i>	{= are is}	—
<i>Values</i>	integer...	—

Mass_Fraction

Syntax

Mass_Fraction *Species {=} Mass fraction*

Summary

Value for the mass fraction of selected species. Can be a constant value or a

string function of space (x,y,z), time (t), and any global variable.

Parameter	Value	Default
<i>Species</i>	string	–
<i>{=}</i>	{= are is}	–
<i>Mass fraction</i>	“string”	–

Mole_Fraction

Syntax

Mole_Fraction *Species* *{=}* *Mole fraction*

Summary

Value for the mole fraction of selected species. Can be a constant value or a string function of space (x,y,z), time (t), and any global variable.

Parameter	Value	Default
<i>Species</i>	string	–
<i>{=}</i>	{= are is}	–
<i>Mole fraction</i>	“string”	–

Postprocess

Syntax

Postprocess *FluxType* Flux Of *{conserved_enthalpy | continuity | edc_product | enthalpy | mixture_fraction | nuclei | progress_variable | scalar_variance | second_mixture_fraction | solid_volume_fraction | soot | species | temperature | turbulent_dissipation | turbulent_frequency | turbulent_helmholtz_function | turbulent_kinetic_energy | turbulent_v2 | volume_of_fluid | x_momentum | x_solid_momentum | y_momentum | y_solid_momentum | z_momentum | z_solid_momentum}* [As *aliases...*]

Summary

Enable nodal flux post-processing for a given equation. Flux types can be “total”, “advective”, or “diffusive”. Integrated fluxes will also be output to global variables for post-processing or use in other string functions.

By default the variables will be named based on the equation and sideset they are applied to as bc_FluxType_EquationName_flux_SurfaceName. However, if you want to assign a more compact or descriptive name you can provide it with

```
POSTPROCESS TOTAL FLUX OF Continuity AS mdot1
```

When post-processing enthalpy, the average temperature is also output as a global variable. If you provide only one alias the enthalpy flux uses your alias and the average temperature is automatically named. If you provide two aliases the first is used for enthalpy and the second is used for temperature.

```
POSTPROCESS TOTAL FLUX OF ENTHALPY AS hFlux
POSTPROCESS TOTAL FLUX OF ENTHALPY AS hFlux TavG
```

When post-processing species, the post-processor is run for each species that is solved for. This means that there is no post-processor run on the last species (which is determined by a fractional balance). If you provide aliases for the species post-processor you should provide one for each post-processed species. For example, in a problem with O2, CO2, and N2 (in that order) you could use:

```
POSTPROCESS TOTAL FLUX OF SPECIES AS mdotO2 mdotCO2
```

Parameter	Value	Default
<i>Flux Type</i>	string	—
<i>equation</i>	{conserved_enthalpy continuity edc_product enthalpy mixture_fraction nuclei progress_variable scalar_variance second_mixture_fraction solid_volume_fraction soot species temperature turbulent_dissipation turbulent_frequency turbulent_helmholtz_function turbulent_kinetic_energy turbulent_v2 volume_of_fluid x_momentum x_solid_momentum y_momentum y_solid_momentum z_momentum z_solid_momentum}	—

Progress_Variable

Syntax

Progress_Variable *ProgressVariableName* {=} *Value*

Summary

Constant value for the progress variable of selected scalars.

Parameter	Value	Default
<i>ProgressVariableName</i>	string	—
{=}	{= are is}	—
<i>Value</i>	real	—

Real Data For Subroutine

Syntax

Real Data For Subroutine *SubName* {=} *Values...*

Summary

List of real data values to be passed down in to the user subroutine. These values may be changed by the user subroutine.

Parameter	Value	Default
<i>SubName</i>	string	–
<i>{=}</i>	{= are is}	–
<i>Values</i>	real. . .	–

Radiation Boundary Temperature

Syntax

Radiation Boundary Temperature *{=}* *value*

Summary

Constant value for radiation_boundary_temperature variable.

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>value</i>	“string”	–

Radiation Boundary Temperature Field

Syntax

Radiation Boundary Temperature Field *{=}* *FieldName*

Summary

Name of the field to use for the radiation boundary temperature.

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>FieldName</i>	string	–

Radiation Environment Temperature

Syntax

Radiation Environment Temperature *{=}* *value*

Summary

Constant value for radiation_environment_temperature variable.

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>value</i>	“string”	–

Radiation Environment Temperature Field

Syntax

Radiation Environment Temperature Field *{=}* *FieldName*

Summary

Name of the field to use for the radiation environment temperature.

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>FieldName</i>	string	–

Subroutine For

Syntax

Subroutine For *{contact_angle | edc_product | gas_volume_fraction | mixture_fraction | pressure | progress_variable | scalar_variance | second_mixture_fraction | solid_volume_fraction | soot_mass_fraction | soot_nuclei_mass_fraction | temperature | turbulent_dissipation | turbulent_frequency | turbulent_helmholtz_function | turbulent_kinetic_energy | turbulent_v2 | volume_of_fluid | x_solid_velocity | x_velocity | y_solid_velocity | y_velocity | z_solid_velocity | z_velocity}* *{=}* *Subroutine*

Summary

Name of the subroutine to use for this variable.

Parameter	Value	Default
<i>Primitive-Variable</i>	{contact_angle edc_product gas_volume_fraction mixture_fraction pressure progress_variable scalar_variance second_mixture_fraction solid_volume_fraction soot_mass_fraction soot_nuclei_mass_fraction temperature turbulent_dissipation turbulent_frequency turbulent_helmholtz_function turbulent_kinetic_energy turbulent_v2 volume_of_fluid x_solid_velocity x_velocity y_solid_velocity y_velocity z_solid_velocity z_velocity}	–
<i>{=}</i>	{= are is}	–
<i>Subroutine</i>	string	–

Subroutine For Emissivity

Syntax

Subroutine For Emissivity *{=}* *subroutineName*

Summary

Name of the user subroutine to use for the emissivity variable.

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>subroutineName</i>	string	–

Subroutine For Mass_Fraction

Syntax

Subroutine For Mass_Fraction *{=}* *Subroutine*

Summary

Name of the subroutine to use for the mass fraction. ALL species mass fractions must be assigned by this subroutine.

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>Subroutine</i>	string	–

Subroutine For Mole_Fraction

Syntax

Subroutine For Mole_Fraction *{=}* *Subroutine*

Summary

Name of the subroutine to use for the mole fractions. ALL species mole fractions must be assigned by this subroutine.

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>Subroutine</i>	string	–

Subroutine For Radiation Boundary Temperature

Syntax

Subroutine For Radiation Boundary Temperature *{=}* *subroutineName*

Summary

Name of the user subroutine to use for the radiation_boundary_temperature variable.

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>subroutineName</i>	string	–

Subroutine For Radiation Environment Temperature

Syntax

Subroutine For Radiation Environment Temperature *{=} subroutineName*

Summary

Name of the user subroutine to use for the radiation_environment_temperature variable.

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>subroutineName</i>	string	–

Subroutine For Transmissivity**Syntax**

Subroutine For Transmissivity *{=} subroutineName*

Summary

Name of the user subroutine to use for the transmissivity variable.

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>subroutineName</i>	string	–

Subroutine For Transparent Band Emissivity**Syntax**

Subroutine For Transparent Band Emissivity *{=} subroutineName*

Summary

Name of the user subroutine to use for the transparent emissivity band for spectral emissivity on surfaces.

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>subroutineName</i>	string	–

Transparent Band Emissivity**Syntax**

Transparent Band Emissivity *{=} value*

Summary

Constant value to use for the emissivity in the transparent band for spectral surfaces.

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>value</i>	“string”	–

Transmissivity

Syntax

Transmissivity *{=}* *value*

Summary

Constant value for transmissivity variable.

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>value</i>	“string”	–

Heat Flux Boundary Condition On Surface

Scope

Fuego Region

Summary

Defines a heat flux boundary condition for heat transfer on a named surface of the mesh.

Description

From Section *Wall Functions; Enthalpy Transport*, In practice, the heat flux boundary condition block is to be defined on an already defined wall boundary condition block (without temperature specification). In this manner, multiple boundary conditions are painted on a particular sideset.

If one desires to post-process heat flux on a surface, one should remember that post-processed heat flux (both in the log file and Exodus output) is the reconstruction from solution variables and any turbulent wall model that may be specified. As such, the post-processed heat flux is dependent on mesh quality and has been shown to converge to the correct value at a zeroth order rate. Thus, caution should be exercised when evaluating post-processed heat flux values.

begin Heat Flux Boundary Condition On Surface *Surfacename*

```
External Field For VariableName [{of} Species] {=} ExtFieldName [ of_
↪Size Value With Multiplier {=} Multiplier ]
```

Function For Heat_Flux {=} *functionName* In The {t | x | y | z}_
→Direction

Heat_Flux {=} *value*

Integer Data For Subroutine *SubName* {=} *Values...*

Postprocess *FluxType* Flux Of {conserved_enthalpy | continuity | edc_
→product | enthalpy | mixture_fraction | nuclei | progress_variable_
→| scalar_variance | second_mixture_fraction | solid_volume_fraction_
→| soot | species | temperature | turbulent_dissipation | turbulent_
→frequency | turbulent_helmholtz_function | turbulent_kinetic_energy_
→| turbulent_v2 | volume_of_fluid | x_momentum | x_solid_momentum |_
→y_momentum | y_solid_momentum | z_momentum | z_solid_momentum} [As_
→*aliases...*]

Real Data For Subroutine *SubName* {=} *Values...*

Subroutine For Heat_Flux {=} *subroutineName*

Law_Of_Wall_Roughness_Parameter {=} *Law of the Wall Roughness parameter*

end Heat Flux Boundary Condition On Surface *Surfacename*

Line Commands

External Field For

Syntax

External Field For *VariableName* [{of} *Species*] {=} *ExtFieldName* [Of Size
Value With Multiplier {=} *Multiplier*]

Summary

Name of the field that is to be transferred in. For species equation, specify variable as “mass fraction of” followed by the species name for transfers associated with separated individual species field names.

Parameter	Value	Default
<i>VariableName</i>	string	–
<i>Species</i>	string	–
{=}	{ = are is }	–
<i>ExtFieldName</i>	string	–

Function For Heat_Flux

Syntax

Function For Heat_Flux {=} *functionName* In The {t | x | y | z} Direction

Summary

Name of the function to use for the heat flux in the given direction.

Parameter	Value	Default
{=}	{ = are is }	–
<i>functionName</i>	string	–
<i>Direction</i>	{ t x y z }	–

Heat_Flux

Syntax

Heat_Flux {=} *value*

Summary

Constant value for the heat flux through the surface.

Description

This value is positive for heat into the surface.

Parameter	Value	Default
{=}	{ = are is }	–
<i>value</i>	“string”	–

Integer Data For Subroutine

Syntax

Integer Data For Subroutine *SubName* {=} *Values* . .

Summary

List of integer data values to be passed down in to the user subroutine. These values may be changed by the user subroutine.

Parameter	Value	Default
<i>SubName</i>	string	–
{=}	{ = are is }	–
<i>Values</i>	integer. . .	–

Postprocess

Syntax

Postprocess *FluxType* Flux Of {*conserved_enthalpy* | *continuity* | *edc_product* | *enthalpy* | *mixture_fraction* | *nuclei* | *progress_variable* | *scalar_variance* | *second_mixture_fraction* | *solid_volume_fraction* | *soot* | *species* | *temperature* | *turbulent_dissipation* | *turbulent_frequency* | *turbulent_helmholtz_function* |

turbulent_kinetic_energy | *turbulent_v2* | *volume_of_fluid* | *x_momentum* | *x_solid_momentum* | *y_momentum* | *y_solid_momentum* | *z_momentum* | *z_solid_momentum* } [As *aliases...*]

Summary

Enable nodal flux post-processing for a given equation. Flux types can be “total”, “advective”, or “diffusive”. Integrated fluxes will also be output to global variables for post-processing or use in other string functions.

By default the variables will be named based on the equation and sideset they are applied to as `bc_FluxType_EquationName_flux_SurfaceName`. However, if you want to assign a more compact or descriptive name you can provide it with

```
POSTPROCESS TOTAL FLUX OF Continuity AS mdot1
```

When post-processing enthalpy, the average temperature is also output as a global variable. If you provide only one alias the enthalpy flux uses your alias and the average temperature is automatically named. If you provide two aliases the first is used for enthalpy and the second is used for temperature.

```
POSTPROCESS TOTAL FLUX OF ENTHALPY AS hFlux
POSTPROCESS TOTAL FLUX OF ENTHALPY AS hFlux Tavg
```

When post-processing species, the post-processor is run for each species that is solved for. This means that there is no post-processor run on the last species (which is determined by a fractional balance). If you provide aliases for the species post-processor you should provide one for each post-processed species. For example, in a problem with O2, CO2, and N2 (in that order) you could use:

```
POSTPROCESS TOTAL FLUX OF SPECIES AS mdotO2 mdotCO2
```

Parameter	Value	Default
<i>FluxType</i>	string	—
<i>equation</i>	{conserved_enthalpy continuity edc_product enthalpy mixture_fraction nuclei progress_variable scalar_variance second_mixture_fraction solid_volume_fraction soot species temperature turbulent_dissipation turbulent_frequency turbulent_helmholtz_function turbulent_kinetic_energy turbulent_v2 volume_of_fluid x_momentum x_solid_momentum y_momentum y_solid_momentum z_momentum z_solid_momentum }	—

Real Data For Subroutine

Syntax

Real Data For Subroutine *SubName* {=} *Values* . .

Summary

List of real data values to be passed down in to the user subroutine. These values may be changed by the user subroutine.

Parameter	Value	Default
<i>SubName</i>	string	–
{=}	{= are is}	–
<i>Values</i>	real. . .	–

Subroutine For Heat_Flux

Syntax

Subroutine For Heat_Flux {=} *subroutineName*

Summary

Name of the subroutine to use for the heat flux.

Parameter	Value	Default
{=}	{= are is}	–
<i>subroutineName</i>	string	–

Law_Of_Wall_Roughness_Parameter

Syntax

Law_Of_Wall_Roughness_Parameter {=} *Law of the Wall Roughness parameter*

Summary

Specify dimensionless roughness factor to be used in the law-of-the-wall formulation.

Parameter	Value	Default
{=}	{= are is}	–
<i>Law of the Wall Roughness parameter</i>	real	9.8

Inflow Boundary Condition On Surface

Scope

Fuego Region

Summary

Defines an inflow boundary condition on a named surface of the mesh.

begin Inflow Boundary Condition On Surface *Surfacename*

*{contact_angle | edc_product | gas_volume_fraction | mixture_fraction |
→| pressure | progress_variable | scalar_variance | second_mixture_
→fraction | solid_volume_fraction | soot_mass_fraction | soot_nuclei_
→mass_fraction | temperature | turbulent_dissipation | turbulent_
→frequency | turbulent_helmholtz_function | turbulent_kinetic_energy_
→| turbulent_v2 | volume_of_fluid | x_solid_velocity | x_velocity | y_
→solid_velocity | y_velocity | z_solid_velocity | z_velocity} {=} Value*

Emissivity Spectral File Name *{=} Name*

External Field For *VariableName* [*{of} Species*] *{=} ExtFieldName* [*Of_
→Size Value* With Multiplier *{=} Multiplier*]

Emissivity *{=} value*

Function For *{contact_angle | edc_product | gas_volume_fraction |
→mixture_fraction | pressure | progress_variable | scalar_variance |
→second_mixture_fraction | solid_volume_fraction | soot_mass_fraction_
→| soot_nuclei_mass_fraction | temperature | turbulent_dissipation |
→turbulent_frequency | turbulent_helmholtz_function | turbulent_kinetic_
→energy | turbulent_v2 | volume_of_fluid | x_solid_velocity | x_velocity_
→| y_solid_velocity | y_velocity | z_solid_velocity | z_velocity} {=}_
→FuncName* [In The *{t | x | y | z}* Direction]

Function For Emissivity *{=} functionName* [In The *{t | x | y | z}*_
→Direction]

Function For Mass_Fraction *{=} FuncName* In The *{t | x | y | z}*_
→Direction

Function For Mole_Fraction *{=} FuncName* In The *{t | x | y | z}*_
→Direction

Function For Radiation Boundary Temperature *{=} functionName*

Function For Radiation Environment Temperature *{=} functionName*

Function For Transmissivity *{=} functionName* [In The *{t | x | y | z}*_
→Direction]

Function For Transparent Band Emissivity *{=} functionName* [In The *{t_
→| x | y | z}* Direction]

Inline Function For {contact_angle | edc_product | gas_volume_fraction |
 ↳ mixture_fraction | pressure | progress_variable | scalar_variance |
 ↳ second_mixture_fraction | solid_volume_fraction | soot_mass_fraction |
 ↳ soot_nuclei_mass_fraction | temperature | turbulent_dissipation |
 ↳ turbulent_frequency | turbulent_helmholtz_function | turbulent_kinetic_
 ↳ energy | turbulent_v2 | volume_of_fluid | x_solid_velocity | x_velocity |
 ↳ y_solid_velocity | y_velocity | z_solid_velocity | z_velocity} {=} **FuncStr**

Integer Data For Subroutine **SubName** {=} **Values...**

Mass_Fraction **Species** {=} **Mass fraction**

Mole_Fraction **Species** {=} **Mole fraction**

Omit Diffusion Terms

Post Process Delta Area

Post Process Delta Mass Flow Rate

Post Process Delta Pressure

Post Process Delta Thrust

Postprocess **FluxType** Flux Of {conserved_enthalpy | continuity | edc_
 ↳ product | enthalpy | mixture_fraction | nuclei | progress_variable |
 ↳ scalar_variance | second_mixture_fraction | solid_volume_fraction |
 ↳ soot | species | temperature | turbulent_dissipation | turbulent_
 ↳ frequency | turbulent_helmholtz_function | turbulent_kinetic_energy |
 ↳ turbulent_v2 | volume_of_fluid | x_momentum | x_solid_momentum |
 ↳ y_momentum | y_solid_momentum | z_momentum | z_solid_momentum} [As_
 ↳ **aliases...**]

Project Nodes

Progress_Variable **ProgressVariableName** {=} **Value**

Real Data For Subroutine **SubName** {=} **Values...**

Radiation Boundary Temperature {=} **value**

Radiation Boundary Temperature Field {=} **FieldName**

Radiation Environment Temperature {=} **value**

Radiation Environment Temperature Field {=} *FieldName*

Subroutine For {*contact_angle* | *edc_product* | *gas_volume_fraction* |
↳*mixture_fraction* | *pressure* | *progress_variable* | *scalar_variance* |
↳*second_mixture_fraction* | *solid_volume_fraction* | *soot_mass_fraction* |
↳*soot_nuclei_mass_fraction* | *temperature* | *turbulent_dissipation* |
↳*turbulent_frequency* | *turbulent_helmholtz_function* | *turbulent_kinetic_*
↳*energy* | *turbulent_v2* | *volume_of_fluid* | *x_solid_velocity* | *x_velocity* |
↳*y_solid_velocity* | *y_velocity* | *z_solid_velocity* | *z_velocity*} {=} |
↳*Subroutine*

Subroutine For Emissivity {=} *subroutineName*

Subroutine For Mass_Fraction {=} *Subroutine*

Subroutine For Mole_Fraction {=} *Subroutine*

Subroutine For Radiation Boundary Temperature {=} *subroutineName*

Subroutine For Radiation Environment Temperature {=} *subroutineName*

Subroutine For Transmissivity {=} *subroutineName*

Subroutine For Transparent Band Emissivity {=} *subroutineName*

Transparent Band Emissivity {=} *value*

Transmissivity {=} *value*

Use Fluxes

Weak Dirichlet

end Inflow Boundary Condition On Surface *Surfacename*

Line Commands

Primitivevariable

Syntax

Primitivevariable {*contact_angle* | *edc_product* | *gas_volume_fraction* | *mixture_fraction* | *pressure* | *progress_variable* | *scalar_variance* | *second_mixture_fraction* | *solid_volume_fraction* | *soot_mass_fraction* | *soot_nuclei_mass_fraction* | *temperature* | *turbulent_dissipation* | *turbulent_frequency* | *turbulent_helmholtz_function* | *turbulent_kinetic_energy* | *turbulent_v2* | *volume_of_fluid* | *x_solid_velocity* | *x_velocity* | *y_solid_velocity* | *y_velocity* | *z_solid_velocity* | *z_velocity*} {=} *Value*

Summary

Value for the specified variable (in consistent units). Value can be a constant or a string function of position (x,y,z), time (t), and any global variable.

The function string must be enclosed in quotes if it has spaces or commas. For example: *x_velocity* = “min(1, 0.1*t)”

Parameter	Value	Default
<i>Primitivevariable</i>	{ <i>contact_angle</i> <i>edc_product</i> <i>gas_volume_fraction</i> <i>mixture_fraction</i> <i>pressure</i> <i>progress_variable</i> <i>scalar_variance</i> <i>second_mixture_fraction</i> <i>solid_volume_fraction</i> <i>soot_mass_fraction</i> <i>soot_nuclei_mass_fraction</i> <i>temperature</i> <i>turbulent_dissipation</i> <i>turbulent_frequency</i> <i>turbulent_helmholtz_function</i> <i>turbulent_kinetic_energy</i> <i>turbulent_v2</i> <i>volume_of_fluid</i> <i>x_solid_velocity</i> <i>x_velocity</i> <i>y_solid_velocity</i> <i>y_velocity</i> <i>z_solid_velocity</i> <i>z_velocity</i> }	–
{=}	{= are is}	–
<i>Value</i>	“string”	–

Emissivity Spectral File Name

Syntax

Emissivity Spectral File Name {=} *Name*

Summary

Specify the file name for defining spectral radiation properties on a surface.

Parameter	Value	Default
{=}	{= are is}	–
<i>Name</i>	string	–

External Field For

Syntax

External Field For *VariableName* [*{of}* *Species*] *{=}* *ExtFieldName* [Of Size *Value* With Multiplier *{=}* *Multiplier*]

Summary

Name of the field that is to be transferred in. For species equation, specify variable as “mass fraction of” followed by the species name for transfers associated with separated individual species field names.

Parameter	Value	Default
<i>VariableName</i>	string	–
<i>Species</i>	string	–
<i>{=}</i>	{ = are is }	–
<i>ExtFieldName</i>	string	–

Emissivity

Syntax

Emissivity *{=}* *value*

Summary

Constant value for emissivity variable.

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>value</i>	“string”	–

Function For

Syntax

Function For *{contact_angle | edc_product | gas_volume_fraction | mixture_fraction | pressure | progress_variable | scalar_variance | second_mixture_fraction | solid_volume_fraction | soot_mass_fraction | soot_nuclei_mass_fraction | temperature | turbulent_dissipation | turbulent_frequency | turbulent_helmholtz_function | turbulent_kinetic_energy | turbulent_v2 | volume_of_fluid | x_solid_velocity | x_velocity | y_solid_velocity | y_velocity | z_solid_velocity | z_velocity}* *{=}* *FuncName* [In The *{t | x | y | z}* Direction]

Summary

Name of function to use for the given variable, in the specified direction.

Parameter	Value	Default
<i>Primitive-Variable</i>	{contact_angle edc_product gas_volume_fraction mixture_fraction pressure progress_variable scalar_variance second_mixture_fraction solid_volume_fraction soot_mass_fraction soot_nuclei_mass_fraction temperature turbulent_dissipation turbulent_frequency turbulent_helmholtz_function turbulent_kinetic_energy turbulent_v2 volume_of_fluid x_solid_velocity x_velocity y_solid_velocity y_velocity z_solid_velocity z_velocity}	–
<i>{=}</i>	{= are is}	–
<i>FuncName</i>	string	–

Function For Emissivity

Syntax

Function For Emissivity *{=}* *funcName* [In The *{t | x | y | z}* Direction]

Summary

Name of the function to use for the emissivity in the given direction (default direction = time).

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>funcName</i>	string	–

Function For Mass_Fraction

Syntax

Function For Mass_Fraction *{=}* *FuncName* In The *{t | x | y | z}* Direction

Summary

Name of the mass fraction function to use for all species in the given direction. Note that this must be a Multicolumn function

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>FuncName</i>	string	–
<i>Direction</i>	{t x y z}	–

Function For Mole_Fraction

Syntax

Function For Mole_Fraction *{=}* *FuncName* In The *{t | x | y | z}* Direction

Summary

Name of the mass fraction function to use for all species in the given direction.
Note that this must be a Multicolumn function

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>FuncName</i>	string	–
<i>Direction</i>	{ t x y z }	–

Function For Radiation Boundary Temperature

Syntax

Function For Radiation Boundary Temperature *{=}* *functionName*

Summary

Name of function to use for the radiation_boundary_temperature variable

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>functionName</i>	string	–

Function For Radiation Environment Temperature

Syntax

Function For Radiation Environment Temperature *{=}* *functionName*

Summary

Name of function to use for the radiation_environment_temperature variable

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>functionName</i>	string	–

Function For Transmissivity

Syntax

Function For Transmissivity *{=}* *functionName* [In The *{t | x | y | z}* Direction]

Summary

Name of function to use for the transmissivity variable

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>functionName</i>	string	–

Function For Transparent Band Emissivity

Syntax

Function For Transparent Band Emissivity *{=}* *functionName* [In The *{t | x | y | z}* Direction]

Summary

Name of function to use for the transparent emissivity band for spectral emissivity on surfaces.

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>functionName</i>	string	–

Inline Function For

Syntax

Inline Function For *{contact_angle | edc_product | gas_volume_fraction | mixture_fraction | pressure | progress_variable | scalar_variance | second_mixture_fraction | solid_volume_fraction | soot_mass_fraction | soot_nuclei_mass_fraction | temperature | turbulent_dissipation | turbulent_frequency | turbulent_helmholtz_function | turbulent_kinetic_energy | turbulent_v2 | volume_of_fluid | x_solid_velocity | x_velocity | y_solid_velocity | y_velocity | z_solid_velocity | z_velocity}* *{=}* *FuncStr*

Summary

warning{This command is deprecated. Regular BCs now support using string functions directly}

Parameter	Value	Default
<i>Primitive-Variable</i>	{contact_angle edc_product gas_volume_fraction mixture_fraction pressure progress_variable scalar_variance second_mixture_fraction solid_volume_fraction soot_mass_fraction soot_nuclei_mass_fraction temperature turbulent_dissipation turbulent_frequency turbulent_helmholtz_function turbulent_kinetic_energy turbulent_v2 volume_of_fluid x_solid_velocity x_velocity y_solid_velocity y_velocity z_solid_velocity z_velocity}	–
<i>{=}</i>	{ = are is }	–
<i>FuncStr</i>	“string”	–

Integer Data For Subroutine

Syntax

Integer Data For Subroutine *SubName* *{=}* *Values...*

Summary

List of integer data values to be passed down in to the user subroutine. These values may be changed by the user subroutine.

Parameter	Value	Default
<i>SubName</i>	string	–
<i>{=}</i>	{ = are is }	–
<i>Values</i>	integer. . .	–

Mass_Fraction

Syntax

Mass_Fraction *Species {=} Mass fraction*

Summary

Value for the mass fraction of selected species. Can be a constant value or a string function of space (x,y,z), time (t), and any global variable.

Parameter	Value	Default
<i>Species</i>	string	–
<i>{=}</i>	{ = are is }	–
<i>Mass fraction</i>	“string”	–

Mole_Fraction

Syntax

Mole_Fraction *Species {=} Mole fraction*

Summary

Value for the mole fraction of selected species. Can be a constant value or a string function of space (x,y,z), time (t), and any global variable.

Parameter	Value	Default
<i>Species</i>	string	–
<i>{=}</i>	{ = are is }	–
<i>Mole fraction</i>	“string”	–

Omit Diffusion Terms

Syntax

Omit Diffusion Terms

Summary

Deactivate the diffusion terms on an open boundary.

Description

It may be preferable to omit the diffusion terms on open open boundaries, but

retain them (default) on confined open boundaries such as duct flow. This is only applied to the momentum equation.

Post Process Delta Area

Syntax

Post Process Delta Area

Summary

Request post processing of delta area on this mesh part.

Description

Area change can be post-processed on this mesh part for laminar or turbulent flows. This option calculates the absolute value of area change after each iteration of a time step, writes it to the log and stores the last value for a given step. The value can be accessed by solution control and used as a criterion in the evaluator. To access this value use `region_name.MaxDeltaArea(n)` where for area change in the x-direction $n=1$, area change in the y-direction $n=2$, and area change in the z-direction $n=3$.

Note that `MaxDeltaArea(1)` will return the maximum area change in x-direction if the request to post process this value was made on more than one boundary in a given region.

Post Process Delta Mass Flow Rate

Syntax

Post Process Delta Mass Flow Rate

Summary

Request post processing of delta mass flow rate on this mesh part.

Description

Mass flow rate change can be post-processed on this mesh part for laminar or turbulent flows. This option calculates the absolute value of mass flow rate change after each iteration of a time step, writes it to the log and stores the last value for a given step. The value can be accessed by solution control and used as a criterion in the evaluator. To access this variable use `region_name.MaxDeltaMassFlowRate(0)`.

Note that `MaxDeltaMassFlowRate(0)` will return the maximum mass flow rate change if the request to post process this value was made on more than one boundary in a given region.

Post Process Delta Pressure

Syntax

Post Process Delta Pressure

Summary

Request post processing of delta pressure on this mesh part.

Description

Pressure change can be post-processed on this mesh part for laminar or turbulent flows. This option calculates the absolute value of pressure change after each iteration of a time step, writes it to the log and stores the last value for a given step. The value can be accessed by solution control and used as a criterion in the evaluator. To access this value use `region_name.MaxDeltaPressure(n)` where for pressure change in the x-direction $n=1$, pressure change in the y-direction $n=2$, and pressure change in the z-direction $n=3$.

Note that `MaxDeltaPressure(1)` will return the maximum pressure change in x-direction if the request to post process this value was made on more than one boundary in a given region.

Post Process Delta Thrust

Syntax

Post Process Delta Thrust

Summary

Request post processing of delta thrust on this mesh part.

Description

Thrust change can be post-processed on this mesh part for laminar or turbulent flows. This option calculates the absolute value of thrust change after each iteration of a time step, writes it to the log and stores the last value for a given step. The value can be accessed by solution control and used as a criterion in the evaluator. To access this value use `region_name.MaxDeltaThrust(n)` where for thrust change in the x-direction $n=1$, thrust change in the y-direction $n=2$, and thrust change in the z-direction $n=3$.

Note that `MaxDeltaThrust(1)` will return the maximum thrust change in x-direction if the request to post process this value was made on more than one boundary in a given region.

Postprocess

Syntax

Postprocess *FluxType* Flux Of {*conserved_enthalpy* | *continuity* | *edc_product* | *enthalpy* | *mixture_fraction* | *nuclei* | *progress_variable* | *scalar_variance* | *second_mixture_fraction* | *solid_volume_fraction* | *soot* | *species* | *temperature* | *turbulent_dissipation* | *turbulent_frequency* | *turbulent_helmholtz_function* | *turbulent_kinetic_energy* | *turbulent_v2* | *volume_of_fluid* | *x_momentum* | *x_solid_momentum* | *y_momentum* | *y_solid_momentum* | *z_momentum* | *z_solid_momentum*} [As *aliases...*]

Summary

Enable nodal flux post-processing for a given equation. Flux types can be “total”, “advective”, or “diffusive”. Integrated fluxes will also be output to global variables for post-processing or use in other string functions.

By default the variables will be named based on the equation and sideset they are applied to as bc_FluxType_EquationName_flux_SurfaceName. However, if you want to assign a more compact or descriptive name you can provide it with

```
POSTPROCESS TOTAL FLUX OF Continuity AS mdot1
```

When post-processing enthalpy, the average temperature is also output as a global variable. If you provide only one alias the enthalpy flux uses your alias and the average temperature is automatically named. If you provide two aliases the first is used for enthalpy and the second is used for temperature.

```
POSTPROCESS TOTAL FLUX OF ENTHALPY AS hFlux
POSTPROCESS TOTAL FLUX OF ENTHALPY AS hFlux Tavg
```

When post-processing species, the post-processor is run for each species that is solved for. This means that there is no post-processor run on the last species (which is determined by a fractional balance). If you provide aliases for the species post-processor you should provide one for each post-processed species. For example, in a problem with O2, CO2, and N2 (in that order) you could use:

```
POSTPROCESS TOTAL FLUX OF SPECIES AS mdotO2 mdotCO2
```

Parameter	Value	Default
<i>Flux Type</i>	string	—
<i>equation</i>	{conserved_enthalpy continuity edc_product enthalpy mixture_fraction nuclei progress_variable scalar_variance second_mixture_fraction solid_volume_fraction soot species temperature turbulent_dissipation turbulent_frequency turbulent_helmholtz_function turbulent_kinetic_energy turbulent_v2 volume_of_fluid x_momentum x_solid_momentum y_momentum y_solid_momentum z_momentum z_solid_momentum}	—

Project Nodes

Syntax

Project Nodes

Summary

Allow for nodal projection on the particular mesh part.

Description

The nodal projection of velocity is defaulted to occur on open boundary conditions only. Wall and inflow BCs are defaulted to not be projected, even on

flux inflow (i.e., USE FLUXES) and turbulent wall nodes. This option flags the particular mesh part to be included in the nodes to be projected.

Progress_Variable

Syntax

Progress_Variable *ProgressVariableName* {=} *Value*

Summary

Constant value for the progress variable of selected scalars.

Parameter	Value	Default
<i>ProgressVariableName</i>	string	—
{=}	{= are is}	—
<i>Value</i>	real	—

Real Data For Subroutine

Syntax

Real Data For Subroutine *SubName* {=} *Values* . .

Summary

List of real data values to be passed down in to the user subroutine. These values may be changed by the user subroutine.

Parameter	Value	Default
<i>SubName</i>	string	—
{=}	{= are is}	—
<i>Values</i>	real. . .	—

Radiation Boundary Temperature

Syntax

Radiation Boundary Temperature {=} *value*

Summary

Constant value for radiation_boundary_temperature variable.

Parameter	Value	Default
{=}	{= are is}	—
<i>value</i>	“string”	—

Radiation Boundary Temperature Field

Syntax

Radiation Boundary Temperature Field {=} *FieldName*

Summary

Name of the field to use for the radiation boundary temperature.

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>FieldName</i>	string	–

Radiation Environment Temperature

Syntax

Radiation Environment Temperature *{=}* *value*

Summary

Constant value for radiation_environment_temperature variable.

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>value</i>	“string”	–

Radiation Environment Temperature Field

Syntax

Radiation Environment Temperature Field *{=}* *FieldName*

Summary

Name of the field to use for the radiation environment temperature.

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>FieldName</i>	string	–

Subroutine For

Syntax

Subroutine For *{contact_angle | edc_product | gas_volume_fraction | mixture_fraction | pressure | progress_variable | scalar_variance | second_mixture_fraction | solid_volume_fraction | soot_mass_fraction | soot_nuclei_mass_fraction | temperature | turbulent_dissipation | turbulent_frequency | turbulent_helmholtz_function | turbulent_kinetic_energy | turbulent_v2 | volume_of_fluid | x_solid_velocity | x_velocity | y_solid_velocity | y_velocity | z_solid_velocity | z_velocity}* *{=}* *Subroutine*

Summary

Name of the subroutine to use for this variable.

Parameter	Value	Default
<i>Primitive-Variable</i>	{contact_angle edc_product gas_volume_fraction mixture_fraction pressure progress_variable scalar_variance second_mixture_fraction solid_volume_fraction soot_mass_fraction soot_nuclei_mass_fraction temperature turbulent_dissipation turbulent_frequency turbulent_helmholtz_function turbulent_kinetic_energy turbulent_v2 volume_of_fluid x_solid_velocity x_velocity y_solid_velocity y_velocity z_solid_velocity z_velocity}	–
<i>{=}</i>	{= are is}	–
<i>Subroutine</i>	string	–

Subroutine For Emissivity

Syntax

Subroutine For Emissivity *{=}* *subroutineName*

Summary

Name of the user subroutine to use for the emissivity variable.

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>subroutineName</i>	string	–

Subroutine For Mass_Fraction

Syntax

Subroutine For Mass_Fraction *{=}* *Subroutine*

Summary

Name of the subroutine to use for the mass fraction. ALL species mass fractions must be assigned by this subroutine.

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>Subroutine</i>	string	–

Subroutine For Mole_Fraction

Syntax

Subroutine For Mole_Fraction *{=}* *Subroutine*

Summary

Name of the subroutine to use for the mole fractions. ALL species mole fractions must be assigned by this subroutine.

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>Subroutine</i>	string	–

Subroutine For Radiation Boundary Temperature

Syntax

Subroutine For Radiation Boundary Temperature *{=}* *subroutineName*

Summary

Name of the user subroutine to use for the radiation_boundary_temperature variable.

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>subroutineName</i>	string	–

Subroutine For Radiation Environment Temperature

Syntax

Subroutine For Radiation Environment Temperature *{=}* *subroutineName*

Summary

Name of the user subroutine to use for the radiation_environment_temperature variable.

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>subroutineName</i>	string	–

Subroutine For Transmissivity

Syntax

Subroutine For Transmissivity *{=}* *subroutineName*

Summary

Name of the user subroutine to use for the transmissivity variable.

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>subroutineName</i>	string	–

Subroutine For Transparent Band Emissivity

Syntax

Subroutine For Transparent Band Emissivity *{=}* *subroutineName*

Summary

Name of the user subroutine to use for the transparent emissivity band for spectral emissivity on surfaces.

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>subroutineName</i>	string	–

Transparent Band Emissivity

Syntax

Transparent Band Emissivity *{=}* *value*

Summary

Constant value to use for the emissivity in the transparent band for spectral surfaces.

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>value</i>	“string”	–

Transmissivity

Syntax

Transmissivity *{=}* *value*

Summary

Constant value for transmissivity variable.

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>value</i>	“string”	–

Use Fluxes

Syntax

Use Fluxes

Summary

Use fluxes at a velocity-specified inflow boundary instead of Dirichlet boundary conditions.

Weak Dirichlet

Syntax

Weak Dirichlet

Summary

Use weak implementation for dirichlet condition

Mass Flux Boundary Condition On Surface

Scope

Fuego Region

Summary

Defines a boundary condition with a specified mass flux on a named surface of the mesh.

Description

The user can specify a mass flow rate either into or out of the domain. The mass flux should obey the sign convention: positive for flux OUT OF the domain, negative for flux INTO the domain. The specified quantity is the total flow rate through the surface, in units of mass per time (note that this is not mass flow per area). When flow is out of the domain (positive mass flux), no variables other than the mass flux need to be specified. When flow into the domain (negative mass flux), the user must provide all of the variables needed for an inflow boundary condition, including (for turbulent flow) the quantities “turbulence_intensity” and “characteristic_length”. The variables provided by the user are used to compute the state of the incoming fluid (e.g. enthalpy). In addition, by computing the density at the specified state, the velocity at the inlet can be computed (since mass flux is the product of density, velocity and area). This velocity is assumed to be in the direction normal to the inlet surface. Any other velocity specified by the user is ignored.

begin Mass Flux Boundary Condition On Surface *Surfacename*

```
{contact_angle | edc_product | gas_volume_fraction | mixture_fraction_  
→| pressure | progress_variable | scalar_variance | second_mixture_  
→fraction | solid_volume_fraction | soot_mass_fraction | soot_nuclei_  
→mass_fraction | temperature | turbulent_dissipation | turbulent_  
→frequency | turbulent_helmholtz_function | turbulent_kinetic_energy_  
→| turbulent_v2 | volume_of_fluid | x_solid_velocity | x_velocity | y_  
→solid_velocity | y_velocity | z_solid_velocity | z_velocity} {=} Value
```

Characteristic_Length {=} *Characteristic length*

Emissivity Spectral File Name {=} *Name*

External Field For *VariableName* [{of} *Species*] [=] *ExtFieldName* [Of
→Size *Value* With Multiplier [=] *Multiplier*]

Emissivity [=] *value*

Enthalpy [=] *value*

Function For {*contact_angle* | *edc_product* | *gas_volume_fraction* |
→*mixture_fraction* | *pressure* | *progress_variable* | *scalar_variance* |
→*second_mixture_fraction* | *solid_volume_fraction* | *soot_mass_fraction* |
→*soot_nuclei_mass_fraction* | *temperature* | *turbulent_dissipation* |
→*turbulent_frequency* | *turbulent_helmholtz_function* | *turbulent_kinetic_*
→*energy* | *turbulent_v2* | *volume_of_fluid* | *x_solid_velocity* | *x_velocity* |
→*y_solid_velocity* | *y_velocity* | *z_solid_velocity* | *z_velocity*} [=]
→*FuncName* [In The {*t* | *x* | *y* | *z*} Direction]

Function For Emissivity [=] *functionName* [In The {*t* | *x* | *y* | *z*}
→Direction]

Function For Mass_Fraction [=] *FuncName* In The {*t* | *x* | *y* | *z*}
→Direction

Function For Mole_Fraction [=] *FuncName* In The {*t* | *x* | *y* | *z*}
→Direction

Function For Radiation Boundary Temperature [=] *functionName*

Function For Radiation Environment Temperature [=] *functionName*

Function For Transmissivity [=] *functionName* [In The {*t* | *x* | *y* | *z*}
→Direction]

Function For Transparent Band Emissivity [=] *functionName* [In The {*t* |
→*x* | *y* | *z*} Direction]

Include Condensed Phase Convection

Inline Function For {*contact_angle* | *edc_product* | *gas_volume_fraction* |
→*mixture_fraction* | *pressure* | *progress_variable* | *scalar_variance* |
→*second_mixture_fraction* | *solid_volume_fraction* | *soot_mass_fraction* |
→*soot_nuclei_mass_fraction* | *temperature* | *turbulent_dissipation* |
→*turbulent_frequency* | *turbulent_helmholtz_function* | *turbulent_kinetic_*
→*energy* | *turbulent_v2* | *volume_of_fluid* | *x_solid_velocity* | *x_velocity* |
→*y_solid_velocity* | *y_velocity* | *z_solid_velocity* | *z_velocity*} [=]
→*FuncStr*

Integer Data For Subroutine *SubName* [=] *Values...*

Mass Flux [=] *Mass_flux*

Mass_Fraction *Species* [=] *Mass fraction*

Mole_Fraction *Species* [=] *Mole fraction*

Porosity Field [=] *porosity_field_name*

Porous Condensed Temperature Field [=] *porous_condensed_temp_field_name*

Porous Flux Field For {*conserved_enthalpy* | *continuity* | *edc_product* |
→ *enthalpy* | *mixture_fraction* | *nuclei* | *progress_variable* | *scalar* |
→ *variance* | *second_mixture_fraction* | *solid_volume_fraction* | *soot* |
→ *species* | *temperature* | *turbulent_dissipation* | *turbulent_frequency* |
→ *turbulent_helmholtz_function* | *turbulent_kinetic_energy* | *turbulent* |
→ *v2* | *volume_of_fluid* | *x_momentum* | *x_solid_momentum* | *y_momentum* |
→ *y_solid_momentum* | *z_momentum* | *z_solid_momentum*} [{*of*} *Species*] [=] *porous_flux_field_name*

Porous Penalty Field For {*conserved_enthalpy* | *continuity* | *edc* |
→ *product* | *enthalpy* | *mixture_fraction* | *nuclei* | *progress_variable* |
→ *scalar_variance* | *second_mixture_fraction* | *solid_volume_fraction* |
→ *soot* | *species* | *temperature* | *turbulent_dissipation* | *turbulent* |
→ *frequency* | *turbulent_helmholtz_function* | *turbulent_kinetic_energy* |
→ *turbulent_v2* | *volume_of_fluid* | *x_momentum* | *x_solid_momentum* | *y* |
→ *momentum* | *y_solid_momentum* | *z_momentum* | *z_solid_momentum*} [{*of*} *Species*] [=] *porous_penalty_field_name*

Porous Target Value Field For {*conserved_enthalpy* | *continuity* | *edc* |
→ *product* | *enthalpy* | *mixture_fraction* | *nuclei* | *progress_variable* |
→ *scalar_variance* | *second_mixture_fraction* | *solid_volume_fraction* |
→ *soot* | *species* | *temperature* | *turbulent_dissipation* | *turbulent* |
→ *frequency* | *turbulent_helmholtz_function* | *turbulent_kinetic_energy* |
→ *turbulent_v2* | *volume_of_fluid* | *x_momentum* | *x_solid_momentum* | *y* |
→ *momentum* | *y_solid_momentum* | *z_momentum* | *z_solid_momentum*} [{*of*} *Species*] [=] *porous_target_field_name*

Porous Volumetric Heat Transfer Coefficient [=] *porous_vol_xfer_field_name*

Post Process Delta Area

Post Process Delta Mass Flow Rate

Post Process Delta Pressure

Post Process Delta Thrust

Postprocess *FluxType* Flux Of {*conserved_enthalpy* | *continuity* | *edc_*
→*product* | *enthalpy* | *mixture_fraction* | *nuclei* | *progress_variable*_
→| *scalar_variance* | *second_mixture_fraction* | *solid_volume_fraction*_
→| *soot* | *species* | *temperature* | *turbulent_dissipation* | *turbulent*_
→*frequency* | *turbulent_helmholtz_function* | *turbulent_kinetic_energy*_
→| *turbulent_v2* | *volume_of_fluid* | *x_momentum* | *x_solid_momentum* |_
→*y_momentum* | *y_solid_momentum* | *z_momentum* | *z_solid_momentum*} [As_
→*aliases...*]

Progress_Variable *ProgressVariableName* {=} *Value*

Real Data For Subroutine *SubName* {=} *Values...*

Radiation Boundary Temperature {=} *value*

Radiation Boundary Temperature Field {=} *FieldName*

Radiation Environment Temperature {=} *value*

Radiation Environment Temperature Field {=} *FieldName*

Specific Surface Area Field {=} *ssa_field_name*

Subroutine For {*contact_angle* | *edc_product* | *gas_volume_fraction* |_
→*mixture_fraction* | *pressure* | *progress_variable* | *scalar_variance* |_
→*second_mixture_fraction* | *solid_volume_fraction* | *soot_mass_fraction*_
→| *soot_nuclei_mass_fraction* | *temperature* | *turbulent_dissipation* |_
→*turbulent_frequency* | *turbulent_helmholtz_function* | *turbulent_kinetic*_
→*energy* | *turbulent_v2* | *volume_of_fluid* | *x_solid_velocity* | *x_velocity*_
→| *y_solid_velocity* | *y_velocity* | *z_solid_velocity* | *z_velocity*} {=}_
→*Subroutine*

Subroutine For Emissivity {=} *subroutineName*

Subroutine For Mass_Fraction {=} *Subroutine*

Subroutine For Mole_Fraction {=} *Subroutine*

Subroutine For Radiation Boundary Temperature {=} *subroutineName*

Subroutine For Radiation Environment Temperature {=} *subroutineName*

Subroutine For Transmissivity {=} *subroutineName*

Subroutine For Transparent Band Emissivity {=} *subroutineName*

Transparent Band Emissivity {=} *value*

Transmissivity {=} *value*

Turbulence_Intensity {=} *Turbulent intensity*

Use Porous Coupling For {*conserved_enthalpy* | *continuity* | *edc_product_*
→*enthalpy* | *mixture_fraction* | *nuclei* | *progress_variable* | *scalar_*
→*variance* | *second_mixture_fraction* | *solid_volume_fraction* | *soot* |
→*species* | *temperature* | *turbulent_dissipation* | *turbulent_frequency* |
→*turbulent_helmholtz_function* | *turbulent_kinetic_energy* | *turbulent_*
→*v2* | *volume_of_fluid* | *x_momentum* | *x_solid_momentum* | *y_momentum* | *y_*
→*solid_momentum* | *z_momentum* | *z_solid_momentum*}

end Mass Flux Boundary Condition On Surface *Surfacename*

Line Commands

Primitivevariable

Syntax

Primitivevariable {*contact_angle* | *edc_product* | *gas_volume_fraction* |
mixture_fraction | *pressure* | *progress_variable* | *scalar_variance* |
second_mixture_fraction | *solid_volume_fraction* | *soot_mass_fraction* |
soot_nuclei_mass_fraction | *temperature* | *turbulent_dissipation* |
turbulent_frequency | *turbulent_helmholtz_function* | *turbulent_kinetic_energy* |
turbulent_v2 | *volume_of_fluid* | *x_solid_velocity* | *x_velocity* | *y_solid_velocity* |
y_velocity | *z_solid_velocity* | *z_velocity*} {=} *Value*

Summary

Value for the specified variable (in consistent units). Value can be a constant or a string function of position (x,y,z), time (t), and any global variable.

The function string must be enclosed in quotes if it has spaces or commas. For example: *x_velocity* = "min(1, 0.1*t)"

Parameter	Value	Default
<i>Primitive-Variable</i>	{contact_angle edc_product gas_volume_fraction mixture_fraction pressure progress_variable scalar_variance second_mixture_fraction solid_volume_fraction soot_mass_fraction soot_nuclei_mass_fraction temperature turbulent_dissipation turbulent_frequency turbulent_helmholtz_function turbulent_kinetic_energy turbulent_v2 volume_of_fluid x_solid_velocity x_velocity y_solid_velocity y_velocity z_solid_velocity z_velocity}	–
<i>{=}</i>	{= are is}	–
<i>Value</i>	“string”	–

Characteristic_Length

Syntax

Characteristic_Length *{=}* *Characteristic length*

Summary

Specify characteristic length for possible entrainment on open pressure specified boundary.

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>Characteristic length</i>	real	1.0

Emissivity Spectral File Name

Syntax

Emissivity Spectral File Name *{=}* *Name*

Summary

Specify the file name for defining spectral radiation properties on a surface.

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>Name</i>	string	–

External Field For

Syntax

External Field For *VariableName* [*/of*] *Species*] *{=}* *ExtFieldName* [Of Size *Value* With Multiplier *{=}* *Multiplier*]

Summary

Name of the field that is to be transferred in. For species equation, specify

variable as “mass fraction of” followed by the species name for transfers associated with separated individual species field names.

Parameter	Value	Default
<i>VariableName</i>	string	–
<i>Species</i>	string	–
<i>{=}</i>	{ = are is }	–
<i>ExtFieldName</i>	string	–

Emissivity

Syntax

Emissivity *{=}* *value*

Summary

Constant value for emissivity variable.

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>value</i>	“string”	–

Enthalpy

Syntax

Enthalpy *{=}* *value*

Summary

Value for inflowing enthalpy on mass flux BC.

Description

Value for inflowing enthalpy on mass flux BC. Normally it is best to specify temperature and let Fuego calculate enthalpy, but when coupling mass BCs together it is more natural and less error prone to specify enthalpy directly. The value can be either a constant or a string function of time and global variables.

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>value</i>	“string”	–

Function For

Syntax

Function For *{contact_angle | edc_product | gas_volume_fraction | mixture_fraction | pressure | progress_variable | scalar_variance | second_mixture_fraction | solid_volume_fraction | soot_mass_fraction | soot_nuclei_mass_fraction | temperature | turbulent_dissipation |*

turbulent_frequency | turbulent_helmholtz_function | turbulent_kinetic_energy | turbulent_v2 | volume_of_fluid | x_solid_velocity | x_velocity | y_solid_velocity | y_velocity | z_solid_velocity | z_velocity {=} *FuncName* [In The {*t | x | y | z*} Direction]

Summary

Name of function to use for the given variable, in the specified direction.

Parameter	Value	Default
<i>Primitive-Variable</i>	{contact_angle edc_product gas_volume_fraction mixture_fraction pressure progress_variable scalar_variance second_mixture_fraction solid_volume_fraction soot_mass_fraction soot_nuclei_mass_fraction temperature turbulent_dissipation turbulent_frequency turbulent_helmholtz_function turbulent_kinetic_energy turbulent_v2 volume_of_fluid x_solid_velocity x_velocity y_solid_velocity y_velocity z_solid_velocity z_velocity}	–
{=}	{= are is}	–
<i>FuncName</i>	string	–

Function For Emissivity

Syntax

Function For Emissivity {=} *functionName* [In The {*t | x | y | z*} Direction]

Summary

Name of the function to use for the emissivity in the given direction (default direction = time).

Parameter	Value	Default
{=}	{= are is}	–
<i>functionName</i>	string	–

Function For Mass_Fraction

Syntax

Function For Mass_Fraction {=} *FuncName* In The {*t | x | y | z*} Direction

Summary

Name of the mass fraction function to use for all species in the given direction. Note that this must be a Multicolumn function

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>FuncName</i>	string	–
<i>Direction</i>	{t x y z}	–

Function For Mole_Fraction

Syntax

Function For Mole_Fraction *{=}* *FuncName* In The *{t | x | y | z}* Direction

Summary

Name of the mass fraction function to use for all species in the given direction.
Note that this must be a Multicolumn function

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>FuncName</i>	string	–
<i>Direction</i>	{t x y z}	–

Function For Radiation Boundary Temperature

Syntax

Function For Radiation Boundary Temperature *{=}* *functionName*

Summary

Name of function to use for the radiation_boundary_temperature variable

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>functionName</i>	string	–

Function For Radiation Environment Temperature

Syntax

Function For Radiation Environment Temperature *{=}* *functionName*

Summary

Name of function to use for the radiation_environment_temperature variable

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>functionName</i>	string	–

Function For Transmissivity

Syntax

Function For Transmissivity *{=}* *functionName* [In The *{t | x | y | z}* Direction]

Summary

Name of function to use for the transmissivity variable

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>functionName</i>	string	–

Function For Transparent Band Emissivity

Syntax

Function For Transparent Band Emissivity *{=}* *functionName* [In The *{t | x | y | z}* Direction]

Summary

Name of function to use for the transparent emissivity band for spectral emissivity on surfaces.

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>functionName</i>	string	–

Include Condensed Phase Convection

Syntax

Include Condensed Phase Convection

Summary

Include condensed phase convection contribution to enthalpy flux. Requires specifying condensed temperature, porosity, and volumetric heat transfer coefficient fields that are transferred from the porous region.

Inline Function For

Syntax

Inline Function For *{contact_angle | edc_product | gas_volume_fraction | mixture_fraction | pressure | progress_variable | scalar_variance | second_mixture_fraction | solid_volume_fraction | soot_mass_fraction | soot_nuclei_mass_fraction | temperature | turbulent_dissipation | turbulent_frequency | turbulent_helmholtz_function | turbulent_kinetic_energy | turbulent_v2 | volume_of_fluid | x_solid_velocity | x_velocity | y_solid_velocity | y_velocity | z_solid_velocity | z_velocity}* *{=}* *FuncStr*

Summary

warning{This command is deprecated. Regular BCs now support using string functions directly}

Parameter	Value	Default
<i>Primitive-Variable</i>	{contact_angle edc_product gas_volume_fraction mixture_fraction pressure progress_variable scalar_variance second_mixture_fraction solid_volume_fraction soot_mass_fraction soot_nuclei_mass_fraction temperature turbulent_dissipation turbulent_frequency turbulent_helmholtz_function turbulent_kinetic_energy turbulent_v2 volume_of_fluid x_solid_velocity x_velocity y_solid_velocity y_velocity z_solid_velocity z_velocity}	–
<i>{=}</i>	{= are is}	–
<i>Func-Str</i>	“string”	–

Integer Data For Subroutine

Syntax

Integer Data For Subroutine *SubName* *{=}* *Values* . .

Summary

List of integer data values to be passed down in to the user subroutine. These values may be changed by the user subroutine.

Parameter	Value	Default
<i>SubName</i>	string	–
<i>{=}</i>	{= are is}	–
<i>Values</i>	integer. . .	–

Mass Flux

Syntax

Mass Flux *{=}* *Mass_flux*

Summary

Value for the specified mass flux in consistent units. The value can be a constant or a function of time and global variables.

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>Mass_flux</i>	“string”	–

Mass_Fraction

Syntax

Mass_Fraction *Species* *{=}* *Mass fraction*

Summary

Value for the mass fraction of selected species. Can be a constant value or a string function of space (x,y,z), time (t), and any global variable.

Parameter	Value	Default
<i>Species</i>	string	–
<i>{=}</i>	{= are is}	–
<i>Mass fraction</i>	“string”	–

Mole_Fraction

Syntax

Mole_Fraction *Species {=} Mole fraction*

Summary

Value for the mole fraction of selected species. Can be a constant value or a string function of space (x,y,z), time (t), and any global variable.

Parameter	Value	Default
<i>Species</i>	string	–
<i>{=}</i>	{= are is}	–
<i>Mole fraction</i>	“string”	–

Porosity Field

Syntax

Porosity Field *{=} porosity_field_name*

Summary

Specify name of the external field containing the porosity transferred from the porous region.

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>porosity_field_name</i>	string	–

Porous Condensed Temperature Field

Syntax

Porous Condensed Temperature Field *{=} porous_condensed_temp_field_name*

Summary

Specify name of the external field containing the condensed phase temperature transferred from the porous region.

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>porous_condensed_temp_field_name</i>	string	–

Porous Flux Field For

Syntax

Porous Flux Field For *{conserved_enthalpy | continuity | edc_product | enthalpy | mixture_fraction | nuclei | progress_variable | scalar_variance | second_mixture_fraction | solid_volume_fraction | soot | species | temperature | turbulent_dissipation | turbulent_frequency | turbulent_helmholtz_function | turbulent_kinetic_energy | turbulent_v2 | volume_of_fluid | x_momentum | x_solid_momentum | y_momentum | y_solid_momentum | z_momentum | z_solid_momentum}* [*of*] *Species* *{=}* *porous_flux_field_name*

Summary

Specify name of the external field containing the flux transferred from the porous region for the specified equation. For continuity this should be the mass flux (rho*u) field, for enthalpy or species this should be the diffusive flux field.

Parameter	Value	Default
<i>equation</i>	{conserved_enthalpy continuity edc_product enthalpy mixture_fraction nuclei progress_variable scalar_variance second_mixture_fraction solid_volume_fraction soot species temperature turbulent_dissipation turbulent_frequency turbulent_helmholtz_function turbulent_kinetic_energy turbulent_v2 volume_of_fluid x_momentum x_solid_momentum y_momentum y_solid_momentum z_momentum z_solid_momentum}	–
<i>Species</i>	string	–
<i>{=}</i>	{ = are is }	–
<i>porous_flux_field_name</i>	string	–

Porous Penalty Field For

Syntax

Porous Penalty Field For *{conserved_enthalpy | continuity | edc_product | enthalpy | mixture_fraction | nuclei | progress_variable | scalar_variance | second_mixture_fraction | solid_volume_fraction | soot | species | temperature | turbulent_dissipation | turbulent_frequency | turbulent_helmholtz_function | turbulent_kinetic_energy | turbulent_v2 | volume_of_fluid | x_momentum | x_solid_momentum | y_momentum | y_solid_momentum | z_momentum | z_solid_momentum}* [*of*] *Species* *{=}* *porous_penalty_field_name*

Summary

Specify name of the external field containing the penalty parameter transferred from the porous region for the specified equation.

Parameter	Value	Default
<i>equation</i>	{conserved_enthalpy continuity edc_product enthalpy mixture_fraction nuclei progress_variable scalar_variance second_mixture_fraction solid_volume_fraction soot species temperature turbulent_dissipation turbulent_frequency turbulent_helmholtz_function turbulent_kinetic_energy turbulent_v2 volume_of_fluid x_momentum x_solid_momentum y_momentum y_solid_momentum z_momentum z_solid_momentum}	—
<i>Species</i>	string	—
<i>{=}</i>	{= are is}	—
<i>porous_string</i>	<i>penalty_field_name</i>	—

Porous Target Value Field For

Syntax

Porous Target Value Field For *{conserved_enthalpy | continuity | edc_product | enthalpy | mixture_fraction | nuclei | progress_variable | scalar_variance | second_mixture_fraction | solid_volume_fraction | soot | species | temperature | turbulent_dissipation | turbulent_frequency | turbulent_helmholtz_function | turbulent_kinetic_energy | turbulent_v2 | volume_of_fluid | x_momentum | x_solid_momentum | y_momentum | y_solid_momentum | z_momentum | z_solid_momentum}* [*of*] *Species* *{=}* *porous_target_field_name*

Summary

Specify name of the external field containing the target value transferred from the porous region for the specified equation. For continuity this should be pressure, enthalpy it should be enthalpy, and species it should be mass fraction.

Parameter	Value	Default
<i>equation</i>	{conserved_enthalpy continuity edc_product enthalpy mixture_fraction nuclei progress_variable scalar_variance second_mixture_fraction solid_volume_fraction soot species temperature turbulent_dissipation turbulent_frequency turbulent_helmholtz_function turbulent_kinetic_energy turbulent_v2 volume_of_fluid x_momentum x_solid_momentum y_momentum y_solid_momentum z_momentum z_solid_momentum}	—
<i>Species</i>	string	—
<i>{=}</i>	{= are is}	—
<i>porous_string</i>	<i>target_field_name</i>	—

Porous Volumetric Heat Transfer Coefficient

Syntax

Porous Volumetric Heat Transfer Coefficient *{=} porous_vol_xfer_field_name*

Summary

Specify name of the external field containing the volumetric heat transfer coefficient transferred from the porous region.

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>porous_vol_xfer_field_name</i>	string	–

Post Process Delta Area

Syntax

Post Process Delta Area

Summary

Request post processing of delta area on this mesh part.

Description

Area change can be post-processed on this mesh part for laminar or turbulent flows. This option calculates the absolute value of area change after each iteration of a time step, writes it to the log and stores the last value for a given step. The value can be accessed by solution control and used as a criterion in the evaluator. To access this value use `region_name.MaxDeltaArea(n)` where for area change in the x-direction $n=1$, area change in the y-direction $n=2$, and area change in the z-direction $n=3$.

Note that `MaxDeltaArea(1)` will return the maximum area change in x-direction if the request to post process this value was made on more than one boundary in a given region.

Post Process Delta Mass Flow Rate

Syntax

Post Process Delta Mass Flow Rate

Summary

Request post processing of delta mass flow rate on this mesh part.

Description

Mass flow rate change can be post-processed on this mesh part for laminar or turbulent flows. This option calculates the absolute value of mass flow rate change after each iteration of a time step, writes it to the log and stores the last value for a given step. The value can be accessed by solution control and used as a criterion in the evaluator. To access this variable use `region_name.MaxDeltaMassFlowRate(0)`.

Note that `MaxDeltaMassFlowRate(0)` will return the maximum mass flow rate change if the request to post process this value was made on more than one boundary in a given region.

Post Process Delta Pressure

Syntax

Post Process Delta Pressure

Summary

Request post processing of delta pressure on this mesh part.

Description

Pressure change can be post-processed on this mesh part for laminar or turbulent flows. This option calculates the absolute value of pressure change after each iteration of a time step, writes it to the log and stores the last value for a given step. The value can be accessed by solution control and used as a criterion in the evaluator. To access this value use `region_name.MaxDeltaPressure(n)` where for pressure change in the x-direction `n=1`, pressure change in the y-direction `n=2`, and pressure change in the z-direction `n=3`.

Note that `MaxDeltaPressure(1)` will return the maximum pressure change in x-direction if the request to post process this value was made on more than one boundary in a given region.

Post Process Delta Thrust

Syntax

Post Process Delta Thrust

Summary

Request post processing of delta thrust on this mesh part.

Description

Thrust change can be post-processed on this mesh part for laminar or turbulent flows. This option calculates the absolute value of thrust change after each iteration of a time step, writes it to the log and stores the last value for a given step. The value can be accessed by solution control and used as a criterion in the evaluator. To access this value use `region_name.MaxDeltaThrust(n)` where for thrust change in the x-direction `n=1`, thrust change in the y-direction `n=2`, and thrust change in the z-direction `n=3`.

Note that `MaxDeltaThrust(1)` will return the maximum thrust change in x-direction if the request to post process this value was made on more than one boundary in a given region.

Postprocess

Syntax

Postprocess *FluxType* Flux Of *{conserved_enthalpy | continuity | edc_product | enthalpy | mixture_fraction | nuclei | progress_variable | scalar_variance |*

second_mixture_fraction | solid_volume_fraction | soot | species | temperature | turbulent_dissipation | turbulent_frequency | turbulent_helmholtz_function | turbulent_kinetic_energy | turbulent_v2 | volume_of_fluid | x_momentum | x_solid_momentum | y_momentum | y_solid_momentum | z_momentum | z_solid_momentum [As *aliases...*]

Summary

Enable nodal flux post-processing for a given equation. Flux types can be “total”, “advective”, or “diffusive”. Integrated fluxes will also be output to global variables for post-processing or use in other string functions.

By default the variables will be named based on the equation and sideset they are applied to as bc_FluxType_EquationName_flux_SurfaceName. However, if you want to assign a more compact or descriptive name you can provide it with

```
POSTPROCESS TOTAL FLUX OF Continuity AS mdot1
```

When post-processing enthalpy, the average temperature is also output as a global variable. If you provide only one alias the enthalpy flux uses your alias and the average temperature is automatically named. If you provide two aliases the first is used for enthalpy and the second is used for temperature.

```
POSTPROCESS TOTAL FLUX OF ENTHALPY AS hFlux
POSTPROCESS TOTAL FLUX OF ENTHALPY AS hFlux Tavg
```

When post-processing species, the post-processor is run for each species that is solved for. This means that there is no post-processor run on the last species (which is determined by a fractional balance). If you provide aliases for the species post-processor you should provide one for each post-processed species. For example, in a problem with O2, CO2, and N2 (in that order) you could use:

```
POSTPROCESS TOTAL FLUX OF SPECIES AS mdotO2 mdotCO2
```

Parameter	Value	Default
<i>Flux Type</i>	string	—
<i>equation</i>	{conserved_enthalpy continuity edc_product enthalpy mixture_fraction nuclei progress_variable scalar_variance second_mixture_fraction solid_volume_fraction soot species temperature turbulent_dissipation turbulent_frequency turbulent_helmholtz_function turbulent_kinetic_energy turbulent_v2 volume_of_fluid x_momentum x_solid_momentum y_momentum y_solid_momentum z_momentum z_solid_momentum}	—

Progress_Variable

Syntax

Progress_Variable *ProgressVariableName* {=} *Value*

Summary

Constant value for the progress variable of selected scalars.

Parameter	Value	Default
<i>ProgressVariableName</i>	string	—
{=}	{= are is}	—
<i>Value</i>	real	—

Real Data For Subroutine

Syntax

Real Data For Subroutine *SubName* {=} *Values* . .

Summary

List of real data values to be passed down in to the user subroutine. These values may be changed by the user subroutine.

Parameter	Value	Default
<i>SubName</i>	string	—
{=}	{= are is}	—
<i>Values</i>	real. . .	—

Radiation Boundary Temperature

Syntax

Radiation Boundary Temperature {=} *value*

Summary

Constant value for radiation_boundary_temperature variable.

Parameter	Value	Default
{=}	{= are is}	—
<i>value</i>	“string”	—

Radiation Boundary Temperature Field

Syntax

Radiation Boundary Temperature Field {=} *FieldName*

Summary

Name of the field to use for the radiation boundary temperature.

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>FieldName</i>	string	–

Radiation Environment Temperature

Syntax

Radiation Environment Temperature *{=}* *value*

Summary

Constant value for radiation_environment_temperature variable.

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>value</i>	“string”	–

Radiation Environment Temperature Field

Syntax

Radiation Environment Temperature Field *{=}* *FieldName*

Summary

Name of the field to use for the radiation environment temperature.

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>FieldName</i>	string	–

Specific Surface Area Field

Syntax

Specific Surface Area Field *{=}* *ssa_field_name*

Summary

Specify name of the external field containing the specific surface area transferred from the porous region.

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>ssa_field_name</i>	string	–

Subroutine For

Syntax

Subroutine For *{contact_angle | edc_product | gas_volume_fraction | mixture_fraction | pressure | progress_variable | scalar_variance |*

second_mixture_fraction | solid_volume_fraction | soot_mass_fraction | soot_nuclei_mass_fraction | temperature | turbulent_dissipation | turbulent_frequency | turbulent_helmholtz_function | turbulent_kinetic_energy | turbulent_v2 | volume_of_fluid | x_solid_velocity | x_velocity | y_solid_velocity | y_velocity | z_solid_velocity | z_velocity} {=} Subroutine

Summary

Name of the subroutine to use for this variable.

Parameter	Value	Default
<i>Primitive-Variable</i>	{contact_angle edc_product gas_volume_fraction mixture_fraction pressure progress_variable scalar_variance second_mixture_fraction solid_volume_fraction soot_mass_fraction soot_nuclei_mass_fraction temperature turbulent_dissipation turbulent_frequency turbulent_helmholtz_function turbulent_kinetic_energy turbulent_v2 volume_of_fluid x_solid_velocity x_velocity y_solid_velocity y_velocity z_solid_velocity z_velocity}	—
<i>{=}</i>	{= are is}	—
<i>Subroutine</i>	string	—

Subroutine For Emissivity

Syntax

Subroutine For Emissivity *{=} subroutineName*

Summary

Name of the user subroutine to use for the emissivity variable.

Parameter	Value	Default
<i>{=}</i>	{= are is}	—
<i>subroutineName</i>	string	—

Subroutine For Mass_Fraction

Syntax

Subroutine For Mass_Fraction *{=} Subroutine*

Summary

Name of the subroutine to use for the mass fraction. ALL species mass fractions must be assigned by this subroutine.

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>Subroutine</i>	string	–

Subroutine For Mole_Fraction

Syntax

Subroutine For Mole_Fraction *{=}* *Subroutine*

Summary

Name of the subroutine to use for the mole fractions. ALL species mole fractions must be assigned by this subroutine.

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>Subroutine</i>	string	–

Subroutine For Radiation Boundary Temperature

Syntax

Subroutine For Radiation Boundary Temperature *{=}* *subroutineName*

Summary

Name of the user subroutine to use for the radiation_boundary_temperature variable.

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>subroutineName</i>	string	–

Subroutine For Radiation Environment Temperature

Syntax

Subroutine For Radiation Environment Temperature *{=}* *subroutineName*

Summary

Name of the user subroutine to use for the radiation_environment_temperature variable.

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>subroutineName</i>	string	–

Subroutine For Transmissivity

Syntax

Subroutine For Transmissivity *{=}* *subroutineName*

Summary

Name of the user subroutine to use for the transmissivity variable.

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>subroutineName</i>	string	–

Subroutine For Transparent Band Emissivity**Syntax**

Subroutine For Transparent Band Emissivity *{=}* *subroutineName*

Summary

Name of the user subroutine to use for the transparent emissivity band for spectral emissivity on surfaces.

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>subroutineName</i>	string	–

Transparent Band Emissivity**Syntax**

Transparent Band Emissivity *{=}* *value*

Summary

Constant value to use for the emissivity in the transparent band for spectral surfaces.

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>value</i>	“string”	–

Transmissivity**Syntax**

Transmissivity *{=}* *value*

Summary

Constant value for transmissivity variable.

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>value</i>	“string”	–

Turbulence_Intensity

Syntax

Turbulence_Intensity {=} *Turbulent intensity*

Summary

Specify turbulent intensity for use in the calculation for the entrainment value of turbulent kinetic energy when there is possible entrainment on open pressure specified boundary.

Parameter	Value	Default
{=}	{= are is}	–
<i>Turbulent intensity</i>	real	–

Use Porous Coupling For

Syntax

Use Porous Coupling For {*conserved_enthalpy* | *continuity* | *edc_product* | *enthalpy* | *mixture_fraction* | *nuclei* | *progress_variable* | *scalar_variance* | *second_mixture_fraction* | *solid_volume_fraction* | *soot* | *species* | *temperature* | *turbulent_dissipation* | *turbulent_frequency* | *turbulent_helmholtz_function* | *turbulent_kinetic_energy* | *turbulent_v2* | *volume_of_fluid* | *x_momentum* | *x_solid_momentum* | *y_momentum* | *y_solid_momentum* | *z_momentum* | *z_solid_momentum*}

Summary

Enable porous fluid coupling algorithm for specified equation.

Parameter	Value	Default
<i>equation</i>	{ <i>conserved_enthalpy</i> <i>continuity</i> <i>edc_product</i> <i>enthalpy</i> <i>mixture_fraction</i> <i>nuclei</i> <i>progress_variable</i> <i>scalar_variance</i> <i>second_mixture_fraction</i> <i>solid_volume_fraction</i> <i>soot</i> <i>species</i> <i>temperature</i> <i>turbulent_dissipation</i> <i>turbulent_frequency</i> <i>turbulent_helmholtz_function</i> <i>turbulent_kinetic_energy</i> <i>turbulent_v2</i> <i>volume_of_fluid</i> <i>x_momentum</i> <i>x_solid_momentum</i> <i>y_momentum</i> <i>y_solid_momentum</i> <i>z_momentum</i> <i>z_solid_momentum</i> }	–

Open Boundary Condition On Surface

Scope

Fuego Region

Summary

Defines an open (specified pressure) boundary condition on a named surface of the mesh. Optionally, specify far-field entrainment values for scalars. Any primitive variable line command below is used to specify these far-field entrainment values. One may also specify a total pressure in which case, the dynamic component will be removed for use in the pressure gradient and mass flux routine. Lastly, for turbulent flows, if the turbulence quantities are not set, then one must specify at least a characteristic length or also specify a turbulence intensity (10% assumed). These values are used to determine an entrainment value for turbulence quantities.

begin Open Boundary Condition On Surface *Surfacename*

```
{contact_angle | edc_product | gas_volume_fraction | mixture_fraction |  
→| pressure | progress_variable | scalar_variance | second_mixture_  
→fraction | solid_volume_fraction | soot_mass_fraction | soot_nuclei_  
→mass_fraction | temperature | turbulent_dissipation | turbulent_  
→frequency | turbulent_helmholtz_function | turbulent_kinetic_energy |  
→| turbulent_v2 | volume_of_fluid | x_solid_velocity | x_velocity | y_  
→solid_velocity | y_velocity | z_solid_velocity | z_velocity} {=} Value
```

Apply Pressure Penalty [Of *PenaltyValue*]

Characteristic_Length {=} *Characteristic length*

Emissivity Spectral File Name {=} *Name*

Entrain Local Values

```
External Field For VariableName [{of} Species] {=} ExtFieldName [ Of_  
→Size Value With Multiplier {=} Multiplier ]
```

Emissivity {=} *value*

Flow Must Exit Domain

```
Function For {contact_angle | edc_product | gas_volume_fraction |  
→mixture_fraction | pressure | progress_variable | scalar_variance |  
→second_mixture_fraction | solid_volume_fraction | soot_mass_fraction |  
→| soot_nuclei_mass_fraction | temperature | turbulent_dissipation |
```

↳turbulent_frequency | turbulent_helmholtz_function | turbulent_kinetic_
 ↳energy | turbulent_v2 | volume_of_fluid | x_solid_velocity | x_velocity_
 ↳| y_solid_velocity | y_velocity | z_solid_velocity | z_velocity} {=}_
 ↳FuncName [In The {t | x | y | z} Direction]

Function For Emissivity {=} functionName [In The {t | x | y | z}_
 ↳Direction]

Function For Mass_Fraction {=} FuncName In The {t | x | y | z}_
 ↳Direction

Function For Mole_Fraction {=} FuncName In The {t | x | y | z}_
 ↳Direction

Function For Radiation Boundary Temperature {=} functionName

Function For Radiation Environment Temperature {=} functionName

Function For Transmissivity {=} functionName [In The {t | x | y | z}_
 ↳Direction]

Function For Transparent Band Emissivity {=} functionName [In The {t_
 ↳| x | y | z} Direction]

Function For Total_Pressure {=} Pressure function In The {t | x | y |_
 ↳z} Direction

Inline Function For {contact_angle | edc_product | gas_volume_fraction_
 ↳| mixture_fraction | pressure | progress_variable | scalar_variance |_
 ↳second_mixture_fraction | solid_volume_fraction | soot_mass_fraction_
 ↳| soot_nuclei_mass_fraction | temperature | turbulent_dissipation |_
 ↳turbulent_frequency | turbulent_helmholtz_function | turbulent_kinetic_
 ↳energy | turbulent_v2 | volume_of_fluid | x_solid_velocity | x_velocity_
 ↳| y_solid_velocity | y_velocity | z_solid_velocity | z_velocity} {=}_
 ↳FuncStr

Integer Data For Subroutine SubName {=} Values...

Momentum Entrainment {=} {computed | specified | tangential}

Mass_Fraction Species {=} Mass fraction

Mole_Fraction Species {=} Mole fraction

Omit Diffusion Terms

Post Process Delta Area

Post Process Delta Mass Flow Rate

Post Process Delta Pressure

Post Process Delta Thrust

Postprocess **FluxType** Flux Of {conserved_enthalpy | continuity | edc_
→product | enthalpy | mixture_fraction | nuclei | progress_variable_
→| scalar_variance | second_mixture_fraction | solid_volume_fraction_
→| soot | species | temperature | turbulent_dissipation | turbulent_
→frequency | turbulent_helmholtz_function | turbulent_kinetic_energy_
→| turbulent_v2 | volume_of_fluid | x_momentum | x_solid_momentum |_
→y_momentum | y_solid_momentum | z_momentum | z_solid_momentum} [As_
→aliases...]

Progress_Variable **ProgressVariableName** {=} **Value**

Real Data For Subroutine **SubName** {=} **Values...**

Radiation Boundary Temperature {=} **value**

Radiation Boundary Temperature Field {=} **FieldName**

Radiation Environment Temperature {=} **value**

Radiation Environment Temperature Field {=} **FieldName**

Sponge Layer Blending Function {=} {heaviside | linear}

Sponge Layer Thickness {=} **Value**

Sponge Layer Viscosity {=} **Value**

Sponge Layer Viscosity Type {=} {absolute | multiple}

Subroutine For {contact_angle | edc_product | gas_volume_fraction |_
→mixture_fraction | pressure | progress_variable | scalar_variance |_
→second_mixture_fraction | solid_volume_fraction | soot_mass_fraction_
→| soot_nuclei_mass_fraction | temperature | turbulent_dissipation |_
→turbulent_frequency | turbulent_helmholtz_function | turbulent_kinetic_
→energy | turbulent_v2 | volume_of_fluid | x_solid_velocity | x_velocity_
→| y_solid_velocity | y_velocity | z_solid_velocity | z_velocity} {=}_

↪Subroutine

Subroutine For Emissivity {=} *subroutineName*

Subroutine For Mass_Fraction {=} *Subroutine*

Subroutine For Mole_Fraction {=} *Subroutine*

Subroutine For Radiation Boundary Temperature {=} *subroutineName*

Subroutine For Radiation Environment Temperature {=} *subroutineName*

Subroutine For Transmissivity {=} *subroutineName*

Subroutine For Transparent Band Emissivity {=} *subroutineName*

Subroutine For Total_Pressure {=} *Subroutine*

Transparent Band Emissivity {=} *value*

Total_Pressure {=} *Pressure*

Transmissivity {=} *value*

Turbulence_Intensity {=} *Turbulent intensity*

Use Sponge Layer

end Open Boundary Condition On Surface *Surfacename*

Line Commands

Primitivevariable

Syntax

Primitivevariable {*contact_angle* | *edc_product* | *gas_volume_fraction* | *mixture_fraction* | *pressure* | *progress_variable* | *scalar_variance* | *second_mixture_fraction* | *solid_volume_fraction* | *soot_mass_fraction* | *soot_nuclei_mass_fraction* | *temperature* | *turbulent_dissipation* | *turbulent_frequency* | *turbulent_helmholtz_function* | *turbulent_kinetic_energy* | *turbulent_v2* | *volume_of_fluid* | *x_solid_velocity* | *x_velocity* | *y_solid_velocity* | *y_velocity* | *z_solid_velocity* | *z_velocity*} {=} *Value*

Summary

Value for the specified variable (in consistent units). Value can be a constant or a

string function of position (x,y,z), time (t), and any global variable.

The function string must be enclosed in quotes if it has spaces or commas. For example: x_velocity = “min(1, 0.1*t)”

Parameter	Value	Default
<i>Primitive-Variable</i>	{contact_angle edc_product gas_volume_fraction mixture_fraction pressure progress_variable scalar_variance second_mixture_fraction solid_volume_fraction soot_mass_fraction soot_nuclei_mass_fraction temperature turbulent_dissipation turbulent_frequency turbulent_helmholtz_function turbulent_kinetic_energy turbulent_v2 volume_of_fluid x_solid_velocity x_velocity y_solid_velocity y_velocity z_solid_velocity z_velocity}	–
<i>{=}</i>	{= are is}	–
<i>Value</i>	“string”	–

Apply Pressure Penalty

Syntax

Apply Pressure Penalty [Of *PenaltyValue*]

Summary

Apply a penalty term on the open boundary pressure.

Description

Apply a DG-like penalty term to the open boundary pressure. This can improve stability for tets and non-orthogonal meshes on open boundaries.

Characteristic_Length

Syntax

Characteristic_Length *{=}* *Characteristic length*

Summary

Specify characteristic length for possible entrainment on open pressure specified boundary.

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>Characteristic length</i>	real	1.0

Emissivity Spectral File Name

Syntax

Emissivity Spectral File Name *{=}* *Name*

Summary

Specify the file name for defining spectral radiation properties on a surface.

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>Name</i>	string	–

Entrain Local Values

Syntax

Entrain Local Values

Summary

If the flow re-circulates at the open, entrain the local values rather than specified far-field values. This is applied to scalars at the open (not to velocity or pressure) and if activated will entrain the local scalar value from the previous time step rather than the user-specified far-field value.

External Field For

Syntax

External Field For *VariableName* [*{of}* *Species*] *{=}* *ExtFieldName* [Of Size *Value* With Multiplier *{=}* *Multiplier*]

Summary

Name of the field that is to be transferred in. For species equation, specify variable as “mass fraction of” followed by the species name for transfers associated with separated individual species field names.

Parameter	Value	Default
<i>VariableName</i>	string	–
<i>Species</i>	string	–
<i>{=}</i>	{ = are is }	–
<i>ExtFieldName</i>	string	–

Emissivity

Syntax

Emissivity *{=}* *value*

Summary

Constant value for emissivity variable.

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>value</i>	“string”	–

Flow Must Exit Domain

Syntax

Flow Must Exit Domain

Summary

warning{This command is deprecated and will no longer have any effect. It will be completely removed in a future version of Fuego.}

Function For

Syntax

Function For *{contact_angle | edc_product | gas_volume_fraction | mixture_fraction | pressure | progress_variable | scalar_variance | second_mixture_fraction | solid_volume_fraction | soot_mass_fraction | soot_nuclei_mass_fraction | temperature | turbulent_dissipation | turbulent_frequency | turbulent_helmholtz_function | turbulent_kinetic_energy | turbulent_v2 | volume_of_fluid | x_solid_velocity | x_velocity | y_solid_velocity | y_velocity | z_solid_velocity | z_velocity}* *{=}* *FuncName* [In The *{t | x | y | z}* Direction]

Summary

Name of function to use for the given variable, in the specified direction.

Parameter	Value	Default
<i>Primitive-Variable</i>	{contact_angle edc_product gas_volume_fraction mixture_fraction pressure progress_variable scalar_variance second_mixture_fraction solid_volume_fraction soot_mass_fraction soot_nuclei_mass_fraction temperature turbulent_dissipation turbulent_frequency turbulent_helmholtz_function turbulent_kinetic_energy turbulent_v2 volume_of_fluid x_solid_velocity x_velocity y_solid_velocity y_velocity z_solid_velocity z_velocity}	—
<i>{=}</i>	{= are is}	—
<i>FuncName</i>	string	—

Function For Emissivity

Syntax

Function For Emissivity *{=}* *functionName* [In The *{t | x | y | z}* Direction]

Summary

Name of the function to use for the emissivity in the given direction (default direction = time).

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>functionName</i>	string	–

Function For Mass_Fraction

Syntax

Function For Mass_Fraction *{=}* *FuncName* In The *{t | x | y | z}* Direction

Summary

Name of the mass fraction function to use for all species in the given direction.
Note that this must be a Multicolumn function

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>FuncName</i>	string	–
<i>Direction</i>	{ t x y z }	–

Function For Mole_Fraction

Syntax

Function For Mole_Fraction *{=}* *FuncName* In The *{t | x | y | z}* Direction

Summary

Name of the mass fraction function to use for all species in the given direction.
Note that this must be a Multicolumn function

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>FuncName</i>	string	–
<i>Direction</i>	{ t x y z }	–

Function For Radiation Boundary Temperature

Syntax

Function For Radiation Boundary Temperature *{=}* *functionName*

Summary

Name of function to use for the radiation_boundary_temperature variable

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>functionName</i>	string	–

Function For Radiation Environment Temperature

Syntax

Function For Radiation Environment Temperature $\{=\}$ *functionName*

Summary

Name of function to use for the radiation_environment_temperature variable

Parameter	Value	Default
$\{=\}$	{ = are is }	–
<i>functionName</i>	string	–

Function For Transmissivity**Syntax**

Function For Transmissivity $\{=\}$ *functionName* [In The $\{t | x | y | z\}$ Direction]

Summary

Name of function to use for the transmissivity variable

Parameter	Value	Default
$\{=\}$	{ = are is }	–
<i>functionName</i>	string	–

Function For Transparent Band Emissivity**Syntax**

Function For Transparent Band Emissivity $\{=\}$ *functionName* [In The $\{t | x | y | z\}$ Direction]

Summary

Name of function to use for the transparent emissivity band for spectral emissivity on surfaces.

Parameter	Value	Default
$\{=\}$	{ = are is }	–
<i>functionName</i>	string	–

Function For Total_Pressure**Syntax**

Function For Total_Pressure $\{=\}$ *Pressure function* In The $\{t | x | y | z\}$ Direction

Summary

total pressure function name

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>Pressure function</i>	string	–
<i>Direction</i>	{t x y z}	–

Inline Function For

Syntax

Inline Function For *{contact_angle | edc_product | gas_volume_fraction | mixture_fraction | pressure | progress_variable | scalar_variance | second_mixture_fraction | solid_volume_fraction | soot_mass_fraction | soot_nuclei_mass_fraction | temperature | turbulent_dissipation | turbulent_frequency | turbulent_helmholtz_function | turbulent_kinetic_energy | turbulent_v2 | volume_of_fluid | x_solid_velocity | x_velocity | y_solid_velocity | y_velocity | z_solid_velocity | z_velocity}* *{=}* *FuncStr*

Summary

warning{This command is deprecated. Regular BCs now support using string functions directly}

Parameter	Value	Default
<i>Primitive-Variable</i>	{contact_angle edc_product gas_volume_fraction mixture_fraction pressure progress_variable scalar_variance second_mixture_fraction solid_volume_fraction soot_mass_fraction soot_nuclei_mass_fraction temperature turbulent_dissipation turbulent_frequency turbulent_helmholtz_function turbulent_kinetic_energy turbulent_v2 volume_of_fluid x_solid_velocity x_velocity y_solid_velocity y_velocity z_solid_velocity z_velocity}	–
<i>{=}</i>	{= are is}	–
<i>FuncStr</i>	“string”	–

Integer Data For Subroutine

Syntax

Integer Data For Subroutine *SubName* *{=}* *Values...*

Summary

List of integer data values to be passed down in to the user subroutine. These values may be changed by the user subroutine.

Parameter	Value	Default
<i>SubName</i>	string	–
<i>{=}</i>	{= are is}	–
<i>Values</i>	integer. . .	–

Momentum Entrainment

Syntax

Momentum Entrainment *{=}* *{computed | specified | tangential}*

Summary

Select how momentum is entrained at the open BC.

Description

The available options are COMPUTED, TANGENTIAL, or SPECIFIED (default).

The default momentum entrainment (SPECIFIED) entrains the user-specified velocities at the open when flow enters the domain.

The computed momentum entrainment (COMPUTED) calculates a normal velocity component using the local mass flux and density, and a tangential component from the user specification on the boundary. This can have stability issues at high vorticity, but often gives better behavior for open BCs that are primarily inflowing or entraining.

It may be preferable to omit the normal component of entrained momentum at an open BC that is primarily exiting the domain (TANGENTIAL) or to simply entrain the user-specified velocities (SPECIFIED).

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>EntrainmentMethod</i>	{computed specified tangential}	–

Mass_Fraction

Syntax

Mass_Fraction *Species* *{=}* *Mass fraction*

Summary

Value for the mass fraction of selected species. Can be a constant value or a string function of space (x,y,z), time (t), and any global variable.

Parameter	Value	Default
<i>Species</i>	string	–
<i>{=}</i>	{= are is}	–
<i>Mass fraction</i>	“string”	–

Mole_Fraction

Syntax

Mole_Fraction *Species {=} Mole fraction*

Summary

Value for the mole fraction of selected species. Can be a constant value or a string function of space (x,y,z), time (t), and any global variable.

Parameter	Value	Default
<i>Species</i>	string	–
<i>{=}</i>	{= are is}	–
<i>Mole fraction</i>	“string”	–

Omit Diffusion Terms

Syntax

Omit Diffusion Terms

Summary

Deactivate the diffusion terms on an open boundary.

Description

It may be preferable to omit the diffusion terms on open open boundaries, but retain them (default) on confined open boundaries such as duct flow. This is only applied to the momentum equation.

Post Process Delta Area

Syntax

Post Process Delta Area

Summary

Request post processing of delta area on this mesh part.

Description

Area change can be post-processed on this mesh part for laminar or turbulent flows. This option calculates the absolute value of area change after each iteration of a time step, writes it to the log and stores the last value for a given step. The value can be accessed by solution control and used as a criterion in the evaluator. To access this value use `region_name.MaxDeltaArea(n)` where for area change in the x-direction n=1, area change in the y-direction n=2, and area change in the z-direction n=3.

Note that `MaxDeltaArea(1)` will return the maximum area change in x-direction if the request to post process this value was made on more than one boundary in a given region.

Post Process Delta Mass Flow Rate

Syntax

Post Process Delta Mass Flow Rate

Summary

Request post processing of delta mass flow rate on this mesh part.

Description

Mass flow rate change can be post-processed on this mesh part for laminar or turbulent flows. This option calculates the absolute value of mass flow rate change after each iteration of a time step, writes it to the log and stores the last value for a given step. The value can be accessed by solution control and used as a criterion in the evaluator. To access this variable use `region_name.MaxDeltaMassFlowRate(0)`.

Note that `MaxDeltaMassFlowRate(0)` will return the maximum mass flow rate change if the request to post process this value was made on more than one boundary in a given region.

Post Process Delta Pressure**Syntax**

Post Process Delta Pressure

Summary

Request post processing of delta pressure on this mesh part.

Description

Pressure change can be post-processed on this mesh part for laminar or turbulent flows. This option calculates the absolute value of pressure change after each iteration of a time step, writes it to the log and stores the last value for a given step. The value can be accessed by solution control and used as a criterion in the evaluator. To access this value use `region_name.MaxDeltaPressure(n)` where for pressure change in the x-direction $n=1$, pressure change in the y-direction $n=2$, and pressure change in the z-direction $n=3$.

Note that `MaxDeltaPressure(1)` will return the maximum pressure change in x-direction if the request to post process this value was made on more than one boundary in a given region.

Post Process Delta Thrust**Syntax**

Post Process Delta Thrust

Summary

Request post processing of delta thrust on this mesh part.

Description

Thrust change can be post-processed on this mesh part for laminar or turbulent flows. This option calculates the absolute value of thrust change after each iteration of a time step, writes it to the log and stores the last value for a given

step. The value can be accessed by solution control and used as a criterion in the evaluator. To access this value use `region_name.MaxDeltaThrust(n)` where for thrust change in the x-direction $n=1$, thrust change in the y-direction $n=2$, and thrust change in the z-direction $n=3$.

Note that `MaxDeltaThrust(1)` will return the maximum thrust change in x-direction if the request to post process this value was made on more than one boundary in a given region.

Postprocess

Syntax

Postprocess *FluxType* Flux Of {*conserved_enthalpy* | *continuity* | *edc_product* | *enthalpy* | *mixture_fraction* | *nuclei* | *progress_variable* | *scalar_variance* | *second_mixture_fraction* | *solid_volume_fraction* | *soot* | *species* | *temperature* | *turbulent_dissipation* | *turbulent_frequency* | *turbulent_helmholtz_function* | *turbulent_kinetic_energy* | *turbulent_v2* | *volume_of_fluid* | *x_momentum* | *x_solid_momentum* | *y_momentum* | *y_solid_momentum* | *z_momentum* | *z_solid_momentum*} [As *aliases...*]

Summary

Enable nodal flux post-processing for a given equation. Flux types can be “total”, “advective”, or “diffusive”. Integrated fluxes will also be output to global variables for post-processing or use in other string functions.

By default the variables will be named based on the equation and sideset they are applied to as `bc_FluxType_EquationName_flux_SurfaceName`. However, if you want to assign a more compact or descriptive name you can provide it with

```
POSTPROCESS TOTAL FLUX OF Continuity AS mdot1
```

When post-processing enthalpy, the average temperature is also output as a global variable. If you provide only one alias the enthalpy flux uses your alias and the average temperature is automatically named. If you provide two aliases the first is used for enthalpy and the second is used for temperature.

```
POSTPROCESS TOTAL FLUX OF ENTHALPY AS hFlux
POSTPROCESS TOTAL FLUX OF ENTHALPY AS hFlux Tavg
```

When post-processing species, the post-processor is run for each species that is solved for. This means that there is no post-processor run on the last species (which is determined by a fractional balance). If you provide aliases for the species post-processor you should provide one for each post-processed species. For example, in a problem with O2, CO2, and N2 (in that order) you could use:

```
POSTPROCESS TOTAL FLUX OF SPECIES AS mdotO2 mdotCO2
```

Parameter	Value	Default
<i>Flux- Type</i>	string	–
<i>equation</i>	{conserved_enthalpy continuity edc_product enthalpy mixture_fraction nuclei progress_variable scalar_variance second_mixture_fraction solid_volume_fraction soot species temperature turbulent_dissipation turbulent_frequency turbulent_helmholtz_function turbulent_kinetic_energy turbulent_v2 volume_of_fluid x_momentum x_solid_momentum y_momentum y_solid_momentum z_momentum z_solid_momentum}	–

Progress_Variable

Syntax

Progress_Variable *ProgressVariableName* {=} *Value*

Summary

Constant value for the progress variable of selected scalars.

Parameter	Value	Default
<i>ProgressVariableName</i>	string	–
{=}	{= are is}	–
<i>Value</i>	real	–

Real Data For Subroutine

Syntax

Real Data For Subroutine *SubName* {=} *Values...*

Summary

List of real data values to be passed down in to the user subroutine. These values may be changed by the user subroutine.

Parameter	Value	Default
<i>SubName</i>	string	–
{=}	{= are is}	–
<i>Values</i>	real. . .	–

Radiation Boundary Temperature

Syntax

Radiation Boundary Temperature {=} *value*

Summary

Constant value for radiation_boundary_temperature variable.

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>value</i>	“string”	–

Radiation Boundary Temperature Field

Syntax

Radiation Boundary Temperature Field *{=}* *FieldName*

Summary

Name of the field to use for the radiation boundary temperature.

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>FieldName</i>	string	–

Radiation Environment Temperature

Syntax

Radiation Environment Temperature *{=}* *value*

Summary

Constant value for radiation_environment_temperature variable.

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>value</i>	“string”	–

Radiation Environment Temperature Field

Syntax

Radiation Environment Temperature Field *{=}* *FieldName*

Summary

Name of the field to use for the radiation environment temperature.

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>FieldName</i>	string	–

Sponge Layer Blending Function

Syntax

Sponge Layer Blending Function **{=}** *{heaviside | linear}*

Summary

Options are Heaviside or Linear. Linear gradually applies the sponge forcing across the layer, and Heaviside applies the full sponge forcing across the entire sponge layer. Default is heaviside.

Parameter	Value	Default
{=}	{= are is}	–
<i>Value</i>	{heaviside linear}	–

Sponge Layer Thickness

Syntax

Sponge Layer Thickness **{=}** *Value*

Summary

Distance the sponge layer will propagate into the domain normal to the open boundary condition's surface

Parameter	Value	Default
{=}	{= are is}	–
<i>Value</i>	real	–

Sponge Layer Viscosity

Syntax

Sponge Layer Viscosity **{=}** *Value*

Summary

Value of the viscosity applied within the sponge layer

Parameter	Value	Default
{=}	{= are is}	–
<i>Value</i>	real	–

Sponge Layer Viscosity Type

Syntax

Sponge Layer Viscosity Type **{=}** *{absolute | multiple}*

Summary

Options are multiple or absolute. The Multiple option uses the product of the molecular viscosity specified sponge layer viscosity to calculate the forcing term. The Absolute option just uses the specified sponge layer viscosity for calculating the forcing term. Multiple is the default

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>Value</i>	{ absolute multiple }	–

Subroutine For

Syntax

Subroutine For *{contact_angle | edc_product | gas_volume_fraction | mixture_fraction | pressure | progress_variable | scalar_variance | second_mixture_fraction | solid_volume_fraction | soot_mass_fraction | soot_nuclei_mass_fraction | temperature | turbulent_dissipation | turbulent_frequency | turbulent_helmholtz_function | turbulent_kinetic_energy | turbulent_v2 | volume_of_fluid | x_solid_velocity | x_velocity | y_solid_velocity | y_velocity | z_solid_velocity | z_velocity}* *{=}* **Subroutine**

Summary

Name of the subroutine to use for this variable.

Parameter	Value	Default
<i>Primitive-Variable</i>	{contact_angle edc_product gas_volume_fraction mixture_fraction pressure progress_variable scalar_variance second_mixture_fraction solid_volume_fraction soot_mass_fraction soot_nuclei_mass_fraction temperature turbulent_dissipation turbulent_frequency turbulent_helmholtz_function turbulent_kinetic_energy turbulent_v2 volume_of_fluid x_solid_velocity x_velocity y_solid_velocity y_velocity z_solid_velocity z_velocity}	–
<i>{=}</i>	{ = are is }	–
<i>Subroutine</i>	string	–

Subroutine For Emissivity

Syntax

Subroutine For Emissivity *{=}* **subroutineName**

Summary

Name of the user subroutine to use for the emissivity variable.

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>subroutineName</i>	string	–

Subroutine For Mass_Fraction

Syntax

Subroutine For Mass_Fraction {=} *Subroutine*

Summary

Name of the subroutine to use for the mass fraction. ALL species mass fractions must be assigned by this subroutine.

Parameter	Value	Default
{=}	{= are is}	–
<i>Subroutine</i>	string	–

Subroutine For Mole_Fraction

Syntax

Subroutine For Mole_Fraction {=} *Subroutine*

Summary

Name of the subroutine to use for the mole fractions. ALL species mole fractions must be assigned by this subroutine.

Parameter	Value	Default
{=}	{= are is}	–
<i>Subroutine</i>	string	–

Subroutine For Radiation Boundary Temperature

Syntax

Subroutine For Radiation Boundary Temperature {=} *subroutineName*

Summary

Name of the user subroutine to use for the radiation_boundary_temperature variable.

Parameter	Value	Default
{=}	{= are is}	–
<i>subroutineName</i>	string	–

Subroutine For Radiation Environment Temperature

Syntax

Subroutine For Radiation Environment Temperature {=} *subroutineName*

Summary

Name of the user subroutine to use for the radiation_environment_temperature variable.

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>subroutineName</i>	string	–

Subroutine For Transmissivity

Syntax

Subroutine For Transmissivity *{=}* *subroutineName*

Summary

Name of the user subroutine to use for the transmissivity variable.

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>subroutineName</i>	string	–

Subroutine For Transparent Band Emissivity

Syntax

Subroutine For Transparent Band Emissivity *{=}* *subroutineName*

Summary

Name of the user subroutine to use for the transparent emissivity band for spectral emissivity on surfaces.

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>subroutineName</i>	string	–

Subroutine For Total_Pressure

Syntax

Subroutine For Total_Pressure *{=}* *Subroutine*

Summary

Name of the subroutine to use for pressure.

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>Subroutine</i>	string	–

Transparent Band Emissivity

Syntax

Transparent Band Emissivity *{=}* *value*

Summary

Constant value to use for the emissivity in the transparent band for spectral surfaces.

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>value</i>	“string”	–

Total_Pressure

Syntax

Total_Pressure *{=}* *Pressure*

Summary

Constant value for the total pressure.

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>Pressure</i>	real	–

Transmissivity

Syntax

Transmissivity *{=}* *value*

Summary

Constant value for transmissivity variable.

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>value</i>	“string”	–

Turbulence_Intensity

Syntax

Turbulence_Intensity *{=}* *Turbulent intensity*

Summary

Specify turbulent intensity for use in the calculation for the entrainment value of turbulent kinetic energy when there is possible entrainment on open pressure specified boundary.

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>Turbulent intensity</i>	real	–

Use Sponge Layer

Syntax

Use Sponge Layer

Summary

Indicator that a sponge layer should be implemented next to this open boundary condition

Symmetry Boundary Condition On Surface

Scope

Fuego Region

Summary

Defines a symmetry plane boundary condition on a named surface of the mesh.

begin Symmetry Boundary Condition On Surface *Surfacename*

Emissivity Spectral File Name *{=} Name*

Emissivity *{=} value*

Function For Emissivity *{=} functionName* [In The *{t | x | y | z}* ↵
↵Direction]

Function For Radiation Boundary Temperature *{=} functionName*

Function For Radiation Environment Temperature *{=} functionName*

Function For Transmissivity *{=} functionName* [In The *{t | x | y | z}* ↵
↵Direction]

Function For Transparent Band Emissivity *{=} functionName* [In The *{t ↵*
↵*| x | y | z}* Direction]

Radiation Boundary Temperature *{=} value*

Radiation Boundary Temperature Field *{=} fieldName*

Radiation Environment Temperature *{=} value*

Radiation Environment Temperature Field *{=} fieldName*

Subroutine For Emissivity *{=} subroutineName*

Subroutine For Radiation Boundary Temperature {=} *subroutineName*

Subroutine For Radiation Environment Temperature {=} *subroutineName*

Subroutine For Transmissivity {=} *subroutineName*

Subroutine For Transparent Band Emissivity {=} *subroutineName*

Transparent Band Emissivity {=} *value*

Transmissivity {=} *value*

end Symmetry Boundary Condition On Surface *Surfacename*

Line Commands

Emissivity Spectral File Name

Syntax

Emissivity Spectral File Name {=} *Name*

Summary

Specify the file name for defining spectral radiation properties on a surface.

Parameter	Value	Default
{=}	{= are is}	–
<i>Name</i>	string	–

Emissivity

Syntax

Emissivity {=} *value*

Summary

Constant value for emissivity variable.

Parameter	Value	Default
{=}	{= are is}	–
<i>value</i>	“string”	–

Function For Emissivity

Syntax

Function For Emissivity {=} *functionName* [In The {t | x | y | z} Direction]

Summary

Name of the function to use for the emissivity in the given direction (default direction = time).

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>functionName</i>	string	–

Function For Radiation Boundary Temperature

Syntax

Function For Radiation Boundary Temperature *{=}* *functionName*

Summary

Name of function to use for the radiation_boundary_temperature variable

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>functionName</i>	string	–

Function For Radiation Environment Temperature

Syntax

Function For Radiation Environment Temperature *{=}* *functionName*

Summary

Name of function to use for the radiation_environment_temperature variable

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>functionName</i>	string	–

Function For Transmissivity

Syntax

Function For Transmissivity *{=}* *functionName* [In The *{t | x | y | z}* Direction]

Summary

Name of function to use for the transmissivity variable

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>functionName</i>	string	–

Function For Transparent Band Emissivity

Syntax

Function For Transparent Band Emissivity *{=}* *functionName* [In The *t* | *x* | *y* | *z* Direction]

Summary

Name of function to use for the transparent emissivity band for spectral emissivity on surfaces.

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>functionName</i>	string	–

Radiation Boundary Temperature

Syntax

Radiation Boundary Temperature *{=}* *value*

Summary

Constant value for radiation_boundary_temperature variable.

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>value</i>	“string”	–

Radiation Boundary Temperature Field

Syntax

Radiation Boundary Temperature Field *{=}* *FieldName*

Summary

Name of the field to use for the radiation boundary temperature.

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>FieldName</i>	string	–

Radiation Environment Temperature

Syntax

Radiation Environment Temperature *{=}* *value*

Summary

Constant value for radiation_environment_temperature variable.

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>value</i>	“string”	–

Radiation Environment Temperature Field

Syntax

Radiation Environment Temperature Field *{=}* *FieldName*

Summary

Name of the field to use for the radiation environment temperature.

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>FieldName</i>	string	–

Subroutine For Emissivity

Syntax

Subroutine For Emissivity *{=}* *subroutineName*

Summary

Name of the user subroutine to use for the emissivity variable.

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>subroutineName</i>	string	–

Subroutine For Radiation Boundary Temperature

Syntax

Subroutine For Radiation Boundary Temperature *{=}* *subroutineName*

Summary

Name of the user subroutine to use for the radiation_boundary_temperature variable.

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>subroutineName</i>	string	–

Subroutine For Radiation Environment Temperature

Syntax

Subroutine For Radiation Environment Temperature *{=}* *subroutineName*

Summary

Name of the user subroutine to use for the radiation_environment_temperature variable.

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>subroutineName</i>	string	–

Subroutine For Transmissivity

Syntax

Subroutine For Transmissivity *{=}* *subroutineName*

Summary

Name of the user subroutine to use for the transmissivity variable.

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>subroutineName</i>	string	–

Subroutine For Transparent Band Emissivity

Syntax

Subroutine For Transparent Band Emissivity *{=}* *subroutineName*

Summary

Name of the user subroutine to use for the transparent emissivity band for spectral emissivity on surfaces.

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>subroutineName</i>	string	–

Transparent Band Emissivity

Syntax

Transparent Band Emissivity *{=}* *value*

Summary

Constant value to use for the emissivity in the transparent band for spectral surfaces.

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>value</i>	“string”	–

Transmissivity

Syntax

Transmissivity *{=}* *value*

Summary

Constant value for transmissivity variable.

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>value</i>	“string”	–

Wall Boundary Condition On Surface

Scope

Fuego Region

Summary

Defines a wall boundary condition on a named surface of the mesh.

begin Wall Boundary Condition On Surface *Surfacename*

```
{contact_angle | edc_product | gas_volume_fraction | mixture_fraction |  
→| pressure | progress_variable | scalar_variance | second_mixture_  
→fraction | solid_volume_fraction | soot_mass_fraction | soot_nuclei_  
→mass_fraction | temperature | turbulent_dissipation | turbulent_  
→frequency | turbulent_helmholtz_function | turbulent_kinetic_energy |  
→| turbulent_v2 | volume_of_fluid | x_solid_velocity | x_velocity | y_  
→solid_velocity | y_velocity | z_solid_velocity | z_velocity} {=} Value
```

```
Activate Surface Flux Balance Temperature Algorithm [ With_  
→BacksideParamFlux1[ BacksideParamFlux2] ]
```

```
Backside {dx | h | k | too} {=} BackVal
```

```
Calculate Convection Coefficient Using Tref {=} Tref ([Dt_Min {=} Dt_min  
→DTmin] | [{clip}])
```

```
Emissivity Spectral File Name {=} Name
```

```
Enforce Zero Flux For Turbulence Wall Bc
```

```
External Field For VariableName [{of} Species] {=} ExtFieldName [ Of_  
→Size Value With Multiplier {=} Multiplier ]
```

```
Emissivity {=} value
```

```
Function For {contact_angle | edc_product | gas_volume_fraction |  
→mixture_fraction | pressure | progress_variable | scalar_variance |
```

```

→second_mixture_fraction | solid_volume_fraction | soot_mass_fraction_
→| soot_nuclei_mass_fraction | temperature | turbulent_dissipation |_
→turbulent_frequency | turbulent_helmholtz_function | turbulent_kinetic_
→energy | turbulent_v2 | volume_of_fluid | x_solid_velocity | x_velocity_
→| y_solid_velocity | y_velocity | z_solid_velocity | z_velocity} {=}
→FuncName [ In The {t | x | y | z} Direction ]

```

```

Function For Emissivity {=} functionName [ In The {t | x | y | z}_
→Direction ]

```

```

Function For Mass_Fraction {=} FuncName In The {t | x | y | z}_
→Direction

```

```

Function For Mole_Fraction {=} FuncName In The {t | x | y | z}_
→Direction

```

```

Function For Radiation Boundary Temperature {=} functionName

```

```

Function For Radiation Environment Temperature {=} functionName

```

```

Function For Transmissivity {=} functionName [ In The {t | x | y | z}_
→Direction ]

```

```

Function For Transparent Band Emissivity {=} functionName [ In The {t_
→| x | y | z} Direction ]

```

```

Function For Wall Temperature {=} functionName [ In The {t | x | y | z}
→ Direction ]

```

```

Inline Function For {contact_angle | edc_product | gas_volume_fraction_
→| mixture_fraction | pressure | progress_variable | scalar_variance |_
→second_mixture_fraction | solid_volume_fraction | soot_mass_fraction_
→| soot_nuclei_mass_fraction | temperature | turbulent_dissipation |_
→turbulent_frequency | turbulent_helmholtz_function | turbulent_kinetic_
→energy | turbulent_v2 | volume_of_fluid | x_solid_velocity | x_velocity_
→| y_solid_velocity | y_velocity | z_solid_velocity | z_velocity} {=}
→FuncStr

```

```

Integer Data For Subroutine SubName {=} Values...

```

```

Interface Boundary

```

```

Mass_Fraction Species {=} Mass fraction

```

```

Mole_Fraction Species {=} Mole fraction

```

Post Process Delta Area

Post Process Delta Heat Flux

Post Process Delta Pressure

Post Process Delta Viscous Stress

Post Process Yplus

Postprocess **FluxType** Flux Of {*conserved_enthalpy* | *continuity* | *edc_*
→*product* | *enthalpy* | *mixture_fraction* | *nuclei* | *progress_variable*_
→| *scalar_variance* | *second_mixture_fraction* | *solid_volume_fraction*_
→| *soot* | *species* | *temperature* | *turbulent_dissipation* | *turbulent*_
→*frequency* | *turbulent_helmholtz_function* | *turbulent_kinetic_energy*_
→| *turbulent_v2* | *volume_of_fluid* | *x_momentum* | *x_solid_momentum* |_
→*y_momentum* | *y_solid_momentum* | *z_momentum* | *z_solid_momentum*} [As_
→*aliases...*]

Project Nodes

Progress_Variable **ProgressVariableName** {=} **Value**

Real Data For Subroutine **SubName** {=} **Values...**

Radiation Boundary Temperature {=} **value**

Radiation Boundary Temperature Field {=} **FieldName**

Radiation Environment Temperature {=} **value**

Radiation Environment Temperature Field {=} **FieldName**

Subroutine For {*contact_angle* | *edc_product* | *gas_volume_fraction* |_
→*mixture_fraction* | *pressure* | *progress_variable* | *scalar_variance* |_
→*second_mixture_fraction* | *solid_volume_fraction* | *soot_mass_fraction*_
→| *soot_nuclei_mass_fraction* | *temperature* | *turbulent_dissipation* |_
→*turbulent_frequency* | *turbulent_helmholtz_function* | *turbulent_kinetic*_
→*energy* | *turbulent_v2* | *volume_of_fluid* | *x_solid_velocity* | *x_velocity*_
→| *y_solid_velocity* | *y_velocity* | *z_solid_velocity* | *z_velocity*} {=}_
→**Subroutine**

Subroutine For Emissivity {=} **subroutineName**

Subroutine For Mass_Fraction {=} *Subroutine*

Subroutine For Mole_Fraction {=} *Subroutine*

Subroutine For Radiation Boundary Temperature {=} *subroutineName*

Subroutine For Radiation Environment Temperature {=} *subroutineName*

Subroutine For Transmissivity {=} *subroutineName*

Subroutine For Transparent Band Emissivity {=} *subroutineName*

Subroutine For Wall Temperature {=} *subroutineName*

Transparent Band Emissivity {=} *value*

Transmissivity {=} *value*

Use Equilibrium Production Model

Use Neumann Condition For Ksgs

Velocity Is Relative To Mesh

Wall Temperature {=} *value*

Weak Dirichlet

Law_Of_Wall_Roughness_Parameter {=} *Law of the Wall Roughness parameter*

Projected Distance For Exchange Location {=} *value*

end Wall Boundary Condition On Surface *Surfacename*

Line Commands

Primitivevariable

Syntax

Primitivevariable {*contact_angle* | *edc_product* | *gas_volume_fraction* | *mixture_fraction* | *pressure* | *progress_variable* | *scalar_variance* | *second_mixture_fraction* | *solid_volume_fraction* | *soot_mass_fraction* | *soot_nuclei_mass_fraction* | *temperature* | *turbulent_dissipation* | *turbulent_frequency* | *turbulent_helmholtz_function* | *turbulent_kinetic_energy* |

*turbulent_v2 | volume_of_fluid | x_solid_velocity | x_velocity | y_solid_velocity | y_velocity | z_solid_velocity | z_velocity} {=} **Value***

Summary

Value for the specified variable (in consistent units). Value can be a constant or a string function of position (x,y,z), time (t), and any global variable.

The function string must be enclosed in quotes if it has spaces or commas. For example: x_velocity = “min(1, 0.1*t)”

Parameter	Value	Default
<i>Primitive-Variable</i>	{contact_angle edc_product gas_volume_fraction mixture_fraction pressure progress_variable scalar_variance second_mixture_fraction solid_volume_fraction soot_mass_fraction soot_nuclei_mass_fraction temperature turbulent_dissipation turbulent_frequency turbulent_helmholtz_function turbulent_kinetic_energy turbulent_v2 volume_of_fluid x_solid_velocity x_velocity y_solid_velocity y_velocity z_solid_velocity z_velocity}	–
<i>{=}</i>	{= are is}	–
<i>Value</i>	“string”	–

Activate Surface Flux Balance Temperature Algorithm

Syntax

Activate Surface Flux Balance Temperature Algorithm [With *BacksideParamFlux1* [*BacksideParamFlux2*]]

Summary

Compute a surface balance equation for nodal temperature.

Description

To be done

Backside

Syntax

Backside *{dx | h | k | too} {=} BackVal*

Summary

Specify h, T, k and dx.

Description

To be done

Parameter	Value	Default
<i>BacksideParam</i>	{dx h k too}	–
<i>{=}</i>	{= are is}	–
<i>BackVal</i>	real	–

Calculate Convection Coefficient

Syntax

Calculate Convection Coefficient Using Tref *{=}* *Tref* ([Dt_Min *{=}* *DTmin*] | [*clip*])

Summary

Calculates the convection coefficient on the wall using a user-specified value for the reference temperature. This calculates $h = Q / (T_{wall} - T_{ref})$.

T_{ref} can be a constant, or a string function of time and global variables.

By default, h is set to 0 if $T_{wall} - T_{ref} < 1 \times 10^{-6}$ however a different tolerance can be supplied with ΔT_{min} .

The optional ‘clip’ argument will clip the calculated value of h to be non-negative.

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>Tref</i>	“string”	–
<i>{=}</i>	{= are is}	–
<i>DTmin</i>	real	–
<i>Clip</i>	{clip}	–

Emissivity Spectral File Name

Syntax

Emissivity Spectral File Name *{=}* *Name*

Summary

Specify the file name for defining spectral radiation properties on a surface.

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>Name</i>	string	–

Enforce Zero Flux For Turbulence Wall Bc

Syntax

Enforce Zero Flux For Turbulence Wall Bc

Summary

warning{This command is deprecated and has no effect}

External Field For

Syntax

External Field For *VariableName* [*{of}* *Species*] *{=}* *ExtFieldName* [Of Size *Value* With Multiplier *{=}* *Multiplier*]

Summary

Name of the field that is to be transferred in. For species equation, specify variable as “mass fraction of” followed by the species name for transfers associated with separated individual species field names.

Parameter	Value	Default
<i>VariableName</i>	string	–
<i>Species</i>	string	–
<i>{=}</i>	{ = are is }	–
<i>ExtFieldName</i>	string	–

Emissivity

Syntax

Emissivity *{=}* *value*

Summary

Constant value for emissivity variable.

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>value</i>	“string”	–

Function For

Syntax

Function For *{contact_angle | edc_product | gas_volume_fraction | mixture_fraction | pressure | progress_variable | scalar_variance | second_mixture_fraction | solid_volume_fraction | soot_mass_fraction | soot_nuclei_mass_fraction | temperature | turbulent_dissipation | turbulent_frequency | turbulent_helmholtz_function | turbulent_kinetic_energy | turbulent_v2 | volume_of_fluid | x_solid_velocity | x_velocity | y_solid_velocity | y_velocity | z_solid_velocity | z_velocity}* *{=}* *FuncName* [In The *{t | x | y | z}* Direction]

Summary

Name of function to use for the given variable, in the specified direction.

Parameter	Value	Default
<i>Primitive-Variable</i>	{contact_angle edc_product gas_volume_fraction mixture_fraction pressure progress_variable scalar_variance second_mixture_fraction solid_volume_fraction soot_mass_fraction soot_nuclei_mass_fraction temperature turbulent_dissipation turbulent_frequency turbulent_helmholtz_function turbulent_kinetic_energy turbulent_v2 volume_of_fluid x_solid_velocity x_velocity y_solid_velocity y_velocity z_solid_velocity z_velocity}	–
<i>{=}</i>	{= are is}	–
<i>FuncName</i>	string	–

Function For Emissivity

Syntax

Function For Emissivity *{=}* *funcName* [In The *{t | x | y | z}* Direction]

Summary

Name of the function to use for the emissivity in the given direction (default direction = time).

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>funcName</i>	string	–

Function For Mass_Fraction

Syntax

Function For Mass_Fraction *{=}* *FuncName* In The *{t | x | y | z}* Direction

Summary

Name of the mass fraction function to use for all species in the given direction. Note that this must be a Multicolumn function

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>FuncName</i>	string	–
<i>Direction</i>	{t x y z}	–

Function For Mole_Fraction

Syntax

Function For Mole_Fraction *{=}* *FuncName* In The *{t | x | y | z}* Direction

Summary

Name of the mass fraction function to use for all species in the given direction.
Note that this must be a Multicolumn function

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>FuncName</i>	string	–
<i>Direction</i>	{ t x y z }	–

Function For Radiation Boundary Temperature

Syntax

Function For Radiation Boundary Temperature *{=}* *functionName*

Summary

Name of function to use for the radiation_boundary_temperature variable

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>functionName</i>	string	–

Function For Radiation Environment Temperature

Syntax

Function For Radiation Environment Temperature *{=}* *functionName*

Summary

Name of function to use for the radiation_environment_temperature variable

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>functionName</i>	string	–

Function For Transmissivity

Syntax

Function For Transmissivity *{=}* *functionName* [In The *{t | x | y | z}* Direction]

Summary

Name of function to use for the transmissivity variable

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>functionName</i>	string	–

Function For Transparent Band Emissivity

Syntax

Function For Transparent Band Emissivity **{=}** *functionName* [In The *{t | x | y | z}* Direction]

Summary

Name of function to use for the transparent emissivity band for spectral emissivity on surfaces.

Parameter	Value	Default
{=}	{ = are is }	–
<i>functionName</i>	string	–

Function For Wall Temperature

Syntax

Function For Wall Temperature **{=}** *functionName* [In The *{t | x | y | z}* Direction]

Summary

Name of function to use for the wall_temperature variable, in the specified direction.

Parameter	Value	Default
{=}	{ = are is }	–
<i>functionName</i>	string	–

Inline Function For

Syntax

Inline Function For *{contact_angle | edc_product | gas_volume_fraction | mixture_fraction | pressure | progress_variable | scalar_variance | second_mixture_fraction | solid_volume_fraction | soot_mass_fraction | soot_nuclei_mass_fraction | temperature | turbulent_dissipation | turbulent_frequency | turbulent_helmholtz_function | turbulent_kinetic_energy | turbulent_v2 | volume_of_fluid | x_solid_velocity | x_velocity | y_solid_velocity | y_velocity | z_solid_velocity | z_velocity}* **{=}** *FuncStr*

Summary

warning{This command is deprecated. Regular BCs now support using string functions directly}

Parameter	Value	Default
<i>Primitive-Variable</i>	{contact_angle edc_product gas_volume_fraction mixture_fraction pressure progress_variable scalar_variance second_mixture_fraction solid_volume_fraction soot_mass_fraction soot_nuclei_mass_fraction temperature turbulent_dissipation turbulent_frequency turbulent_helmholtz_function turbulent_kinetic_energy turbulent_v2 volume_of_fluid x_solid_velocity x_velocity y_solid_velocity y_velocity z_solid_velocity z_velocity}	–
<i>{=}</i>	{= are is}	–
<i>Func-Str</i>	“string”	–

Integer Data For Subroutine

Syntax

Integer Data For Subroutine *SubName* *{=}* *Values* . .

Summary

List of integer data values to be passed down in to the user subroutine. These values may be changed by the user subroutine.

Parameter	Value	Default
<i>SubName</i>	string	–
<i>{=}</i>	{= are is}	–
<i>Values</i>	integer. . .	–

Interface Boundary

Syntax

Interface Boundary

Summary

Mark this fluids boundary as an interface between two physical regions. Data will be interpolated across this boundary.

Mass_Fraction

Syntax

Mass_Fraction *Species* *{=}* *Mass fraction*

Summary

Value for the mass fraction of selected species. Can be a constant value or a string function of space (x,y,z), time (t), and any global variable.

Parameter	Value	Default
<i>Species</i>	string	–
<i>{=}</i>	{= are is}	–
<i>Mass fraction</i>	“string”	–

Mole_Fraction

Syntax

Mole_Fraction *Species {=} Mole fraction*

Summary

Value for the mole fraction of selected species. Can be a constant value or a string function of space (x,y,z), time (t), and any global variable.

Parameter	Value	Default
<i>Species</i>	string	–
<i>{=}</i>	{= are is}	–
<i>Mole fraction</i>	“string”	–

Post Process Delta Area

Syntax

Post Process Delta Area

Summary

Request post processing of delta area on this mesh part.

Description

Area change can be post-processed on this mesh part for laminar or turbulent flows. This option calculates the absolute value of area change after each iteration of a time step, writes it to the log and stores the last value for a given step. The value can be accessed by solution control and used as a criterion in the evaluator. To access this value use `region_name.MaxDeltaArea(n)` where for area change in the x-direction n=1, area change in the y-direction n=2, and area change in the z-direction n=3.

Note that `MaxDeltaArea(1)` will return the maximum area change in x-direction if the request to post process this value was made on more than one boundary in a given region.

Post Process Delta Heat Flux

Syntax

Post Process Delta Heat Flux

Summary

Request post processing of delta heat flux on this mesh part.

Description

Heat flux change can be post-processed on this mesh part for laminar or turbulent flows. This option calculates the absolute value of heat flux change after each iteration of a time step, writes it to the log and stores the last value for a given step. The value can be accessed by solution control and used as a criterion in the evaluator. To access this variable use `region_name.MaxDeltaHeatFlux(0)`

Note that `MaxDeltaHeatFlux(0)` will return the maximum heat flux change if the request to post process this value was made on more than one boundary in a given region.

Post Process Delta Pressure**Syntax**

Post Process Delta Pressure

Summary

Request post processing of delta pressure on this mesh part.

Description

Pressure change can be post-processed on this mesh part for laminar or turbulent flows. This option calculates the absolute value of pressure change after each iteration of a time step, writes it to the log and stores the last value for a given step. The value can be accessed by solution control and used as a criterion in the evaluator. To access this value use `region_name.MaxDeltaPressure(n)` where for pressure change in the x-direction $n=1$, pressure change in the y-direction $n=2$, and pressure change in the z-direction $n=3$.

Note that `MaxDeltaPressure(1)` will return the maximum pressure change in x-direction if the request to post process this value was made on more than one boundary in a given region.

Post Process Delta Viscous Stress**Syntax**

Post Process Delta Viscous Stress

Summary

Request post processing of delta viscous stress on this mesh part.

Description

Viscous stress change can be post-processed on this mesh part for laminar or turbulent flows. This option calculates the absolute value of viscous stress change after each iteration of a time step, writes it to the log and stores the last value for a given step. The value can be accessed by solution control and used as a criterion in the evaluator. To access this value use `region_name.MaxDeltaViscousStress(n)` where for viscous stress change in the x-direction $n=1$, viscous stress change in the y-direction $n=2$, and viscous stress change in the z-direction $n=3$.

Note that `MaxDeltaViscousStress(1)` will return the maximum viscous stress change in x-direction if the request to post process this value was made on more than one boundary in a given region.

Post Process Yplus

Syntax

Post Process Yplus

Summary

Request post processing of yplus on this mesh part.

Description

The normalized distance to the wall (y^+) can be post-processed on any wall (laminar or turbulent). Yplus is calculated in a manner consistent with how it is calculated in the solution procedure. Max/min y^+ values are provided as output at every wall. This option computes and stores the entire y^+ field along the wall. Since this option computes y^+ using the most recent values for all solution variables, the max/min written to the log may differ from the max/min of this field. However, these will become identical at convergence.

Note that this line command does not automatically output this field to a results file. For that, one must use the standard output line command for a nodal field using the string name, yplus.

Postprocess

Syntax

Postprocess *FluxType* Flux Of {*conserved_enthalpy* | *continuity* | *edc_product* | *enthalpy* | *mixture_fraction* | *nuclei* | *progress_variable* | *scalar_variance* | *second_mixture_fraction* | *solid_volume_fraction* | *soot* | *species* | *temperature* | *turbulent_dissipation* | *turbulent_frequency* | *turbulent_helmholtz_function* | *turbulent_kinetic_energy* | *turbulent_v2* | *volume_of_fluid* | *x_momentum* | *x_solid_momentum* | *y_momentum* | *y_solid_momentum* | *z_momentum* | *z_solid_momentum*} [As *aliases...*]

Summary

Enable nodal flux post-processing for a given equation. Flux types can be “total”, “advective”, or “diffusive”. Integrated fluxes will also be output to global variables for post-processing or use in other string functions.

By default the variables will be named based on the equation and sideset they are applied to as `bc_FluxType_EquationName_flux_SurfaceName`. However, if you want to assign a more compact or descriptive name you can provide it with

```
POSTPROCESS TOTAL FLUX OF Continuity AS mdot1
```

When post-processing enthalpy, the average temperature is also output as a global variable. If you provide only one alias the enthalpy flux uses your alias

and the average temperature is automatically named. If you provide two aliases the first is used for enthalpy and the second is used for temperature.

```
POSTPROCESS TOTAL FLUX OF ENTHALPY AS hFlux
POSTPROCESS TOTAL FLUX OF ENTHALPY AS hFlux Tavg
```

When post-processing species, the post-processor is run for each species that is solved for. This means that there is no post-processor run on the last species (which is determined by a fractional balance). If you provide aliases for the species post-processor you should provide one for each post-processed species. For example, in a problem with O2, CO2, and N2 (in that order) you could use:

```
POSTPROCESS TOTAL FLUX OF SPECIES AS mdotO2 mdotCO2
```

Parameter	Value	Default
<i>Flux Type</i>	string	—
<i>equation</i>	{conserved_enthalpy continuity edc_product enthalpy mixture_fraction nuclei progress_variable scalar_variance second_mixture_fraction solid_volume_fraction soot species temperature turbulent_dissipation turbulent_frequency turbulent_helmholtz_function turbulent_kinetic_energy turbulent_v2 volume_of_fluid x_momentum x_solid_momentum y_momentum y_solid_momentum z_momentum z_solid_momentum}	—

Project Nodes

Syntax

Project Nodes

Summary

Allow for nodal projection on the particular mesh part.

Description

The nodal projection of velocity is defaulted to occur on open boundary conditions only. Wall and inflow BCs are defaulted to not be projected, even on flux inflow (i.e., USE FLUXES) and turbulent wall nodes. This option flags the particular mesh part to be included in the nodes to be projected.

Progress_Variable

Syntax

Progress_Variable *ProgressVariableName* [=] *Value*

Summary

Constant value for the progress variable of selected scalars.

Parameter	Value	Default
<i>ProgressVariableName</i>	string	—
<i>{=}</i>	{= are is}	—
<i>Value</i>	real	—

Real Data For Subroutine

Syntax

Real Data For Subroutine *SubName* *{=}* *Values* . .

Summary

List of real data values to be passed down in to the user subroutine. These values may be changed by the user subroutine.

Parameter	Value	Default
<i>SubName</i>	string	—
<i>{=}</i>	{= are is}	—
<i>Values</i>	real. . .	—

Radiation Boundary Temperature

Syntax

Radiation Boundary Temperature *{=}* *value*

Summary

Constant value for radiation_boundary_temperature variable.

Parameter	Value	Default
<i>{=}</i>	{= are is}	—
<i>value</i>	“string”	—

Radiation Boundary Temperature Field

Syntax

Radiation Boundary Temperature Field *{=}* *FieldName*

Summary

Name of the field to use for the radiation boundary temperature.

Parameter	Value	Default
<i>{=}</i>	{= are is}	—
<i>FieldName</i>	string	—

Radiation Environment Temperature

Syntax

Radiation Environment Temperature *{=}* *value*

Summary

Constant value for radiation_environment_temperature variable.

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>value</i>	“string”	–

Radiation Environment Temperature Field

Syntax

Radiation Environment Temperature Field *{=}* *FieldName*

Summary

Name of the field to use for the radiation environment temperature.

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>FieldName</i>	string	–

Subroutine For

Syntax

Subroutine For *{contact_angle | edc_product | gas_volume_fraction | mixture_fraction | pressure | progress_variable | scalar_variance | second_mixture_fraction | solid_volume_fraction | soot_mass_fraction | soot_nuclei_mass_fraction | temperature | turbulent_dissipation | turbulent_frequency | turbulent_helmholtz_function | turbulent_kinetic_energy | turbulent_v2 | volume_of_fluid | x_solid_velocity | x_velocity | y_solid_velocity | y_velocity | z_solid_velocity | z_velocity}* *{=}* *Subroutine*

Summary

Name of the subroutine to use for this variable.

Parameter	Value	Default
<i>Primitive-Variable</i>	{contact_angle edc_product gas_volume_fraction mixture_fraction pressure progress_variable scalar_variance second_mixture_fraction solid_volume_fraction soot_mass_fraction soot_nuclei_mass_fraction temperature turbulent_dissipation turbulent_frequency turbulent_helmholtz_function turbulent_kinetic_energy turbulent_v2 volume_of_fluid x_solid_velocity x_velocity y_solid_velocity y_velocity z_solid_velocity z_velocity}	–
<i>{=}</i>	{= are is}	–
<i>Subroutine</i>	string	–

Subroutine For Emissivity

Syntax

Subroutine For Emissivity *{=} subroutineName*

Summary

Name of the user subroutine to use for the emissivity variable.

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>subroutineName</i>	string	–

Subroutine For Mass_Fraction

Syntax

Subroutine For Mass_Fraction *{=} Subroutine*

Summary

Name of the subroutine to use for the mass fraction. ALL species mass fractions must be assigned by this subroutine.

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>Subroutine</i>	string	–

Subroutine For Mole_Fraction

Syntax

Subroutine For Mole_Fraction *{=} Subroutine*

Summary

Name of the subroutine to use for the mole fractions. ALL species mole fractions must be assigned by this subroutine.

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>Subroutine</i>	string	–

Subroutine For Radiation Boundary Temperature

Syntax

Subroutine For Radiation Boundary Temperature *{=}* *subroutineName*

Summary

Name of the user subroutine to use for the radiation_boundary_temperature variable.

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>subroutineName</i>	string	–

Subroutine For Radiation Environment Temperature

Syntax

Subroutine For Radiation Environment Temperature *{=}* *subroutineName*

Summary

Name of the user subroutine to use for the radiation_environment_temperature variable.

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>subroutineName</i>	string	–

Subroutine For Transmissivity

Syntax

Subroutine For Transmissivity *{=}* *subroutineName*

Summary

Name of the user subroutine to use for the transmissivity variable.

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>subroutineName</i>	string	–

Subroutine For Transparent Band Emissivity

Syntax

Subroutine For Transparent Band Emissivity *{=}* *subroutineName*

Summary

Name of the user subroutine to use for the transparent emissivity band for spectral emissivity on surfaces.

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>subroutineName</i>	string	–

Subroutine For Wall Temperature

Syntax

Subroutine For Wall Temperature *{=}* *subroutineName*

Summary

Name of the user subroutine to use for the wall_temperature variable.

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>subroutineName</i>	string	–

Transparent Band Emissivity

Syntax

Transparent Band Emissivity *{=}* *value*

Summary

Constant value to use for the emissivity in the transparent band for spectral surfaces.

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>value</i>	“string”	–

Transmissivity

Syntax

Transmissivity *{=}* *value*

Summary

Constant value for transmissivity variable.

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>value</i>	“string”	–

Use Equilibrium Production Model

Syntax

Use Equilibrium Production Model

Summary

Change near-wall turbulence production.

Description

This option specifies that the near wall turb_ke production term is given by:

$$\frac{\tau_w^2}{\kappa \sqrt[4]{C_\mu} \rho y_p \sqrt{k}}$$

. Modifications are made for the KW and SST model. This model is suggested when using the standard KW and SST model when a Dirichlet condition is not placed on the wall node of turbulence kinetic energy.

Use Neumann Condition For Ksgs

Syntax

Use Neumann Condition For Ksgs

Summary

Apply a Neumann bc for the one-equation turbulent SGS equation.

Description

This option specifies that the normal component of the subgrid scale gradient is zero at the wall. The standard SGS production and dissipation term are used. This method is preferred over the USE EQUILIBRIUM PRODUCTION MODEL option that is most suitable for RANS. Moreover, the user should remove the OMIT NEAR WALL TURBULENT KE TRANSPORT EQUATION option from solution options and ensure that DETERMINE UTAU VIA NONLINEAR LAW OF THE WALL ITERATION has been activated.

Velocity Is Relative To Mesh

Syntax

Velocity Is Relative To Mesh

Summary

Indicates that the specified velocity is relative to the mesh motion. Applicable only for wall BCs.

Wall Temperature

Syntax

Wall Temperature *{=}* *value*

Summary

Value for wall_temperature variable. Can be a constant or a string function of space (x,y,z), time (t) or global variables.

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>value</i>	“string”	–

Weak Dirichlet**Syntax**

Weak Dirichlet

Summary

Use weak implementation for dirichlet condition

Law_Of_Wall_Roughness_Parameter**Syntax**

Law_Of_Wall_Roughness_Parameter *{=}* *Law of the Wall Roughness parameter*

Summary

Specify dimensionless roughness factor to be used in the law-of-the-wall formulation.

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>Law of the Wall Roughness parameter</i>	real	9.8

Projected Distance For Exchange Location**Syntax**

Projected Distance For Exchange Location *{=}* *value*

Summary

Specify projected distance for the exchange-based model; generally roughly three elements.

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>value</i>	real	0.0

Wall Mass Inject Boundary Condition On Surface

Scope

Fuego Region

Summary

Defines a wall with mass injection boundary condition on a named surface of the mesh. This boundary condition will use all flux conditions for all fields other than turbulent k.e.

begin Wall Mass Inject Boundary Condition On Surface *Surfacename*

```
{contact_angle | edc_product | gas_volume_fraction | mixture_fraction |  
→| pressure | progress_variable | scalar_variance | second_mixture_  
→fraction | solid_volume_fraction | soot_mass_fraction | soot_nuclei_  
→mass_fraction | temperature | turbulent_dissipation | turbulent_  
→frequency | turbulent_helmholtz_function | turbulent_kinetic_energy_  
→| turbulent_v2 | volume_of_fluid | x_solid_velocity | x_velocity | y_  
→solid_velocity | y_velocity | z_solid_velocity | z_velocity} {=} Value
```

```
Calculate Convection Coefficient Using Tref {=} Tref ([Dt_Min {=} Dtmin]  
→Dtmin] | [{clip}])
```

```
Emissivity Spectral File Name {=} Name
```

```
External Field For VariableName [{of} Species] {=} ExtFieldName [ Of_  
→Size Value With Multiplier {=} Multiplier ]
```

```
External Field For Extracted PropertyName [{of} Species] {=} ExtFieldName
```

```
External Field For Injected PropertyName [{of} Species] {=} ExtFieldName
```

```
Extracted PropertyName [{of} Species] {=} propValue
```

```
Emissivity {=} value
```

```
Function For {contact_angle | edc_product | gas_volume_fraction |   
→mixture_fraction | pressure | progress_variable | scalar_variance |   
→second_mixture_fraction | solid_volume_fraction | soot_mass_fraction |   
→| soot_nuclei_mass_fraction | temperature | turbulent_dissipation |   
→turbulent_frequency | turbulent_helmholtz_function | turbulent_kinetic_  
→energy | turbulent_v2 | volume_of_fluid | x_solid_velocity | x_velocity |   
→| y_solid_velocity | y_velocity | z_solid_velocity | z_velocity} {=} Value
```

→ *FuncName* [In The {t | x | y | z} Direction]

Function For Extracted *PropertyName* [{of} *Species*] {=} *FuncName* [In_
→ The {t | x | y | z} Direction]

Function For Emissivity {=} *functionName* [In The {t | x | y | z}_
→ Direction]

Function For Injected *PropertyName* [{of} *Species*] {=} *FuncName* [In_
→ The {t | x | y | z} Direction]

Function For Injected_Temperature {=} *functionName* [In The {t | x | y_
→ | z} Direction]

Function For Mass Flux {=} *functionName* [In The {t | x | y | z}_
→ Direction]

Function For Mass_Fraction {=} *FuncName* In The {t | x | y | z}_
→ Direction

Function For Mole_Fraction {=} *FuncName* In The {t | x | y | z}_
→ Direction

Function For Radiation Boundary Temperature {=} *functionName*

Function For Radiation Environment Temperature {=} *functionName*

Function For Transmissivity {=} *functionName* [In The {t | x | y | z}_
→ Direction]

Function For Transparent Band Emissivity {=} *functionName* [In The {t_
→ | x | y | z} Direction]

Function For Wall Temperature {=} *functionName* [In The {t | x | y | z}
→ Direction]

Injected *PropertyName* [{of} *Species*] {=} *propValue*

Injected_Temperature {=} *temperatureValue*

Inline Function For {contact_angle | edc_product | gas_volume_fraction_
→ | mixture_fraction | pressure | progress_variable | scalar_variance |_
→ second_mixture_fraction | solid_volume_fraction | soot_mass_fraction_
→ | soot_nuclei_mass_fraction | temperature | turbulent_dissipation |_
→ turbulent_frequency | turbulent_helmholtz_function | turbulent_kinetic_

↳energy | turbulent_v2 | volume_of_fluid | x_solid_velocity | x_velocity_␣
 ↳| y_solid_velocity | y_velocity | z_solid_velocity | z_velocity} {=}␣
 ↳FuncStr

Integer Data For Subroutine SubName {=} Values...

Interface Boundary

Mass Flux {=} fluxValue

Mass_Fraction Species {=} Mass fraction

Mole_Fraction Species {=} Mole fraction

Pool {absorption_coefficient | boil_flash_temperature_difference |␣
 ↳density | evaporation_temperature | heat_of_evaporation | ignition_time_␣
 ↳| initial_height | initial_temperature | minimum_mass_flux | minimum_␣
 ↳mass_injection_rate | percent_sensible | specific_heat} {=} The pool_␣
 ↳values

Post Process Yplus

Postprocess FluxType Flux Of {conserved_enthalpy | continuity | edc_␣
 ↳product | enthalpy | mixture_fraction | nuclei | progress_variable_␣
 ↳| scalar_variance | second_mixture_fraction | solid_volume_fraction_␣
 ↳| soot | species | temperature | turbulent_dissipation | turbulent_␣
 ↳frequency | turbulent_helmholtz_function | turbulent_kinetic_energy_␣
 ↳| turbulent_v2 | volume_of_fluid | x_momentum | x_solid_momentum |␣
 ↳y_momentum | y_solid_momentum | z_momentum | z_solid_momentum} [As_␣
 ↳aliases...]

Project Nodes

Progress_Variable ProgressVariableName {=} Value

Real Data For Subroutine SubName {=} Values...

Radiation Boundary Temperature {=} value

Radiation Boundary Temperature Field {=} FieldName

Radiation Environment Temperature {=} value

Radiation Environment Temperature Field {=} FieldName

```

Subroutine For {contact_angle | edc_product | gas_volume_fraction |
↪mixture_fraction | pressure | progress_variable | scalar_variance |
↪second_mixture_fraction | solid_volume_fraction | soot_mass_fraction
↪| soot_nuclei_mass_fraction | temperature | turbulent_dissipation |
↪turbulent_frequency | turbulent_helmholtz_function | turbulent_kinetic_
↪energy | turbulent_v2 | volume_of_fluid | x_solid_velocity | x_velocity
↪| y_solid_velocity | y_velocity | z_solid_velocity | z_velocity} {=}
↪Subroutine

```

```

Subroutine For Emissivity {=} subroutineName

```

```

Subroutine For Mass_Fraction {=} Subroutine

```

```

Subroutine For Mole_Fraction {=} Subroutine

```

```

Subroutine For Radiation Boundary Temperature {=} subroutineName

```

```

Subroutine For Radiation Environment Temperature {=} subroutineName

```

```

Subroutine For Transmissivity {=} subroutineName

```

```

Subroutine For Transparent Band Emissivity {=} subroutineName

```

```

Subroutine For Wall Temperature {=} subroutineName

```

```

Transparent Band Emissivity {=} value

```

```

Transmissivity {=} value

```

```

Use Equilibrium Production Model

```

```

Use Law Of Wall Blowing Correction

```

```

Use Neumann Condition For Ksgs

```

```

Wall Temperature {=} value

```

```

Law_Of_Wall_Blowing_Parameter {=} Law of Wall Blowing parameter

```

```

Law_Of_Wall_Roughness_Parameter {=} Law of the Wall Roughness parameter

```

```

Projected Distance For Exchange Location {=} value

```

```

end Wall Mass Inject Boundary Condition On Surface Surfacename

```

Line Commands

Primitivevariable

Syntax

Primitivevariable {*contact_angle* | *edc_product* | *gas_volume_fraction* | *mixture_fraction* | *pressure* | *progress_variable* | *scalar_variance* | *second_mixture_fraction* | *solid_volume_fraction* | *soot_mass_fraction* | *soot_nuclei_mass_fraction* | *temperature* | *turbulent_dissipation* | *turbulent_frequency* | *turbulent_helmholtz_function* | *turbulent_kinetic_energy* | *turbulent_v2* | *volume_of_fluid* | *x_solid_velocity* | *x_velocity* | *y_solid_velocity* | *y_velocity* | *z_solid_velocity* | *z_velocity*} {=} *Value*

Summary

Value for the specified variable (in consistent units). Value can be a constant or a string function of position (x,y,z), time (t), and any global variable.

The function string must be enclosed in quotes if it has spaces or commas. For example: *x_velocity* = “min(1, 0.1*t)”

Parameter	Value	Default
<i>Primitive-Variable</i>	{ <i>contact_angle</i> <i>edc_product</i> <i>gas_volume_fraction</i> <i>mixture_fraction</i> <i>pressure</i> <i>progress_variable</i> <i>scalar_variance</i> <i>second_mixture_fraction</i> <i>solid_volume_fraction</i> <i>soot_mass_fraction</i> <i>soot_nuclei_mass_fraction</i> <i>temperature</i> <i>turbulent_dissipation</i> <i>turbulent_frequency</i> <i>turbulent_helmholtz_function</i> <i>turbulent_kinetic_energy</i> <i>turbulent_v2</i> <i>volume_of_fluid</i> <i>x_solid_velocity</i> <i>x_velocity</i> <i>y_solid_velocity</i> <i>y_velocity</i> <i>z_solid_velocity</i> <i>z_velocity</i> }	–
{=}	{= are is}	–
<i>Value</i>	“string”	–

Calculate Convection Coefficient

Syntax

Calculate Convection Coefficient Using Tref {=} *Tref* ([Dt_Min {=} *DTmin*] | [*clip*])

Summary

Calculates the convection coefficient on the wall using a user-specified value for the reference temperature. This calculates $h = Q / (T_{wall} - T_{ref})$.

T_{ref} can be a constant, or a string function of time and global variables.

By default, h is set to 0 if $T_{wall} - T_{ref} < 1 \times 10^{-6}$ however a different tolerance can be supplied with ΔT_{min} .

The optional ‘clip’ argument will clip the calculated value of h to be non-negative.

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>Tref</i>	“string”	–
<i>{=}</i>	{ = are is }	–
<i>DTmin</i>	real	–
<i>Clip</i>	{ clip }	–

Emissivity Spectral File Name

Syntax

Emissivity Spectral File Name *{=}* *Name*

Summary

Specify the file name for defining spectral radiation properties on a surface.

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>Name</i>	string	–

External Field For

Syntax

External Field For *VariableName* [*{of}* *Species*] *{=}* *ExtFieldName* [Of Size *Value* With Multiplier *{=}* *Multiplier*]

Summary

Name of the field that is to be transferred in. For species equation, specify variable as “mass fraction of” followed by the species name for transfers associated with separated individual species field names.

Parameter	Value	Default
<i>VariableName</i>	string	–
<i>Species</i>	string	–
<i>{=}</i>	{ = are is }	–
<i>ExtFieldName</i>	string	–

External Field For Extracted

Syntax

External Field For Extracted *PropertyName* [*{of}* *Species*] *{=}* *ExtFieldName*

Summary

Use an external field for this property of the extracted stream.

Parameter	Value	Default
<i>PropertyName</i>	string	–
<i>Species</i>	string	–
<i>{=}</i>	{ = are is }	–
<i>ExtFieldName</i>	string	–

External Field For Injected

Syntax

External Field For Injected *PropertyName* [*{of}* *Species*] *{=}* *ExtFieldName*

Summary

Use an external field for this property of the injected stream.

Parameter	Value	Default
<i>PropertyName</i>	string	–
<i>Species</i>	string	–
<i>{=}</i>	{ = are is }	–
<i>ExtFieldName</i>	string	–

Extracted

Syntax

Extracted *PropertyName* [*{of}* *Species*] *{=}* *propValue*

Summary

Property of the extracted stream. Value can be a constant or a string function of time, space, and global variables.

Parameter	Value	Default
<i>PropertyName</i>	string	–
<i>Species</i>	string	–
<i>{=}</i>	{ = are is }	–
<i>propValue</i>	“string”	–

Emissivity

Syntax

Emissivity *{=}* *value*

Summary

Constant value for emissivity variable.

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>value</i>	“string”	–

Function For

Syntax

Function For {*contact_angle* | *edc_product* | *gas_volume_fraction* | *mixture_fraction* | *pressure* | *progress_variable* | *scalar_variance* | *second_mixture_fraction* | *solid_volume_fraction* | *soot_mass_fraction* | *soot_nuclei_mass_fraction* | *temperature* | *turbulent_dissipation* | *turbulent_frequency* | *turbulent_helmholtz_function* | *turbulent_kinetic_energy* | *turbulent_v2* | *volume_of_fluid* | *x_solid_velocity* | *x_velocity* | *y_solid_velocity* | *y_velocity* | *z_solid_velocity* | *z_velocity*} {=} *FuncName* [In The {*t* | *x* | *y* | *z*} Direction]

Summary

Name of function to use for the given variable, in the specified direction.

Parameter	Value	Default
<i>Primitive-Variable</i>	{ <i>contact_angle</i> <i>edc_product</i> <i>gas_volume_fraction</i> <i>mixture_fraction</i> <i>pressure</i> <i>progress_variable</i> <i>scalar_variance</i> <i>second_mixture_fraction</i> <i>solid_volume_fraction</i> <i>soot_mass_fraction</i> <i>soot_nuclei_mass_fraction</i> <i>temperature</i> <i>turbulent_dissipation</i> <i>turbulent_frequency</i> <i>turbulent_helmholtz_function</i> <i>turbulent_kinetic_energy</i> <i>turbulent_v2</i> <i>volume_of_fluid</i> <i>x_solid_velocity</i> <i>x_velocity</i> <i>y_solid_velocity</i> <i>y_velocity</i> <i>z_solid_velocity</i> <i>z_velocity</i> }	—
{=}	{= are is}	—
<i>FuncName</i>	string	—

Function For Extracted

Syntax

Function For Extracted *PropertyName* [{*of*} *Species*] {=} *FuncName* [In The {*t* | *x* | *y* | *z*} Direction]

Summary

Name of function to use for the given extracted property, in the specified direction.

Parameter	Value	Default
<i>PropertyName</i>	string	—
<i>Species</i>	string	—
{=}	{= are is}	—
<i>FuncName</i>	string	—

Function For Emissivity

Syntax

Function For Emissivity *{=}* *functionName* [In The *{t | x | y | z}* Direction]

Summary

Name of the function to use for the emissivity in the given direction (default direction = time).

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>functionName</i>	string	–

Function For Injected

Syntax

Function For Injected *PropertyName* [*{of}* *Species*] *{=}* *FuncName* [In The *{t | x | y | z}* Direction]

Summary

Name of function to use for the given injected property, in the specified direction.

Parameter	Value	Default
<i>PropertyName</i>	string	–
<i>Species</i>	string	–
<i>{=}</i>	{ = are is }	–
<i>FuncName</i>	string	–

Function For Injected_Temperature

Syntax

Function For Injected_Temperature *{=}* *functionName* [In The *{t | x | y | z}* Direction]

Summary

Name of function to use for the injected temperature, in the specified direction.



Warning: Prefer using `function` for injected temperature = `some_fcn`

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>functionName</i>	string	–

Function For Mass Flux

Syntax

Function For Mass Flux {=} *functionName* [In The {t | x | y | z} Direction]

Summary

Name of function to use for mass flux, in the specified direction.



Warning: Prefer using function for injected mass_flux
= X

Parameter	Value	Default
{=}	{= are is}	–
<i>functionName</i>	string	–

Function For Mass_Fraction

Syntax

Function For Mass_Fraction {=} *FuncName* In The {t | x | y | z} Direction

Summary

Name of the mass fraction function to use for all species in the given direction.

Note that this must be a Multicolumn function

Parameter	Value	Default
{=}	{= are is}	–
<i>FuncName</i>	string	–
<i>Direction</i>	{t x y z}	–

Function For Mole_Fraction

Syntax

Function For Mole_Fraction {=} *FuncName* In The {t | x | y | z} Direction

Summary

Name of the mass fraction function to use for all species in the given direction.

Note that this must be a Multicolumn function

Parameter	Value	Default
{=}	{= are is}	–
<i>FuncName</i>	string	–
<i>Direction</i>	{t x y z}	–

Function For Radiation Boundary Temperature

Syntax

Function For Radiation Boundary Temperature *{=}* *functionName*

Summary

Name of function to use for the radiation_boundary_temperature variable

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>functionName</i>	string	–

Function For Radiation Environment Temperature**Syntax**

Function For Radiation Environment Temperature *{=}* *functionName*

Summary

Name of function to use for the radiation_environment_temperature variable

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>functionName</i>	string	–

Function For Transmissivity**Syntax**

Function For Transmissivity *{=}* *functionName* [In The *{t | x | y | z}* Direction]

Summary

Name of function to use for the transmissivity variable

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>functionName</i>	string	–

Function For Transparent Band Emissivity**Syntax**

Function For Transparent Band Emissivity *{=}* *functionName* [In The *{t | x | y | z}* Direction]

Summary

Name of function to use for the transparent emissivity band for spectral emissivity on surfaces.

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>functionName</i>	string	–

Function For Wall Temperature

Syntax

Function For Wall Temperature *{=}* *functionName* [In The *{t | x | y | z}* Direction]

Summary

Name of function to use for the wall_temperature variable, in the specified direction.

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>functionName</i>	string	–

Injected

Syntax

Injected *PropertyName* [*{of}* *Species*] *{=}* *propValue*

Summary

Property of the injected stream. Value can be a constant or a string function of time, space, and global variables.

Parameter	Value	Default
<i>PropertyName</i>	string	–
<i>Species</i>	string	–
<i>{=}</i>	{ = are is }	–
<i>propValue</i>	“string”	–

Injected_Temperature

Syntax

Injected_Temperature *{=}* *temperatureValue*

Summary

Injected temperature of mass injection wall BC with specified flux. Value can be a constant or a string function of time, space, and global variables.



Warning: Prefer using injected temperature = X

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>temperatureValue</i>	“string”	–

Inline Function For

Syntax

Inline Function For {*contact_angle* | *edc_product* | *gas_volume_fraction* | *mixture_fraction* | *pressure* | *progress_variable* | *scalar_variance* | *second_mixture_fraction* | *solid_volume_fraction* | *soot_mass_fraction* | *soot_nuclei_mass_fraction* | *temperature* | *turbulent_dissipation* | *turbulent_frequency* | *turbulent_helmholtz_function* | *turbulent_kinetic_energy* | *turbulent_v2* | *volume_of_fluid* | *x_solid_velocity* | *x_velocity* | *y_solid_velocity* | *y_velocity* | *z_solid_velocity* | *z_velocity*} {=} *FuncStr*

Summary

warning{This command is deprecated. Regular BCs now support using string functions directly}

Parameter	Value	Default
<i>Primitive-Variable</i>	{ <i>contact_angle</i> <i>edc_product</i> <i>gas_volume_fraction</i> <i>mixture_fraction</i> <i>pressure</i> <i>progress_variable</i> <i>scalar_variance</i> <i>second_mixture_fraction</i> <i>solid_volume_fraction</i> <i>soot_mass_fraction</i> <i>soot_nuclei_mass_fraction</i> <i>temperature</i> <i>turbulent_dissipation</i> <i>turbulent_frequency</i> <i>turbulent_helmholtz_function</i> <i>turbulent_kinetic_energy</i> <i>turbulent_v2</i> <i>volume_of_fluid</i> <i>x_solid_velocity</i> <i>x_velocity</i> <i>y_solid_velocity</i> <i>y_velocity</i> <i>z_solid_velocity</i> <i>z_velocity</i> }	–
{=}	{= are is}	–
<i>FuncStr</i>	“string”	–

Integer Data For Subroutine

Syntax

Integer Data For Subroutine *SubName* {=} *Values*. . .

Summary

List of integer data values to be passed down in to the user subroutine. These values may be changed by the user subroutine.

Parameter	Value	Default
<i>SubName</i>	string	–
{=}	{= are is}	–
<i>Values</i>	integer. . .	–

Interface Boundary

Syntax

Interface Boundary

Summary

Mark this fluids boundary as an interface between two physical regions. Data will be interpolated across this boundary.

Mass Flux

Syntax

Mass Flux *{=}* *fluxValue*

Summary

Mass injection rate (mass/area-time). Value can be a constant or a string function of time, space, and global variables.



Warning: Prefer using injected `mass_flux = X`

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>fluxValue</i>	“string”	–

Mass_Fraction

Syntax

Mass_Fraction *Species* *{=}* *Mass fraction*

Summary

Value for the mass fraction of selected species. Can be a constant value or a string function of space (x,y,z), time (t), and any global variable.

Parameter	Value	Default
<i>Species</i>	string	–
<i>{=}</i>	{ = are is }	–
<i>Mass fraction</i>	“string”	–

Mole_Fraction

Syntax

Mole_Fraction *Species* *{=}* *Mole fraction*

Summary

Value for the mole fraction of selected species. Can be a constant value or a string function of space (x,y,z), time (t), and any global variable.

Parameter	Value	Default
<i>Species</i>	string	–
<i>{=}</i>	{= are is}	–
<i>Mole fraction</i>	“string”	–

Pool

Syntax

Pool {*absorption_coefficient* | *boil_flash_temperature_difference* | *density* | *evaporation_temperature* | *heat_of_evaporation* | *ignition_time* | *initial_height* | *initial_temperature* | *minimum_mass_flux* | *minimum_mass_injection_rate* | *percent_sensible* | *specific_heat*} {=} *The pool values*

Summary

Specify parameters for pool evaporation model.

Description

Specify a number of pool BC parameters. Details about some of the parameters are listed below:

Minimum_Mass_Injection_Rate This command is DEPRECATED. Formerly this would limit the minimum mass flow rate (mass/time) per node to this value when the time is less than the ignition time. This has been replaced by the Minimum_Mass_Flux command, which is in mesh-independent mass/area-time units.

Minimum_Mass_Flux Specify a minimum mass flux to apply to the pool while time is less than the specified ignition time. This defaults to 1e-3 g/(cm²-s) (converted to problem units).

Ignition_Time Specify the time before which the minimum mass flux should be applied. This defaults to 1 second.

Evaporation_Temperature Pool evaporation temperature; default provided as 450 K (converted to user units).

Absorption_Coefficient Absorption coefficient for pool; default provided as 1.0.

Boil_Flash_Temperature_Difference Temperature difference between boil and flash temperature (ΔT_{flash}); default provided as 125 K.

Percent_Sensible Maximum fraction of energy that is applied to pool heat-up ($f_{sensible,max}$); default provided as 0.7. This value should be between 0 and 1. The fraction of energy that goes into heating up the pool ($f_{sensible}$) instead of phase change is defined as:

$$f_{sensible} = \min(f_{sensible,max}, (T_{evap} - T_{pool})/\Delta T_{flash})$$

Setting this fraction to a value less than 1 ensures that even at pool temperatures much lower than the boiling point some fraction of the energy goes into phase change.

Heat_Of_Evaporation Pool heat of evaporation; default provided as 2.8e9 erg/g (converted to user units).

Specific_Heat Pool specific heat; default provided as 24.7e6 erg/g-k (converted to user units).

Density Pool density; default provided as 0.808 g/cm³ (converted to user units).

Parameter	Value	Default
<i>PoolBC</i>	{absorption_coefficient boil_flash_temperature_difference density evaporation_temperature heat_of_evaporation ignition_time initial_height initial_temperature minimum_mass_flux minimum_mass_injection_rate percent_sensible specific_heat}	—
<i>{=}</i>	{= are is}	—
<i>The pool values</i>	real	—

Post Process Yplus

Syntax

Post Process Yplus

Summary

Request post processing of yplus on this mesh part.

Description

The normalized distance to the wall (y^+) can be post-processed on any wall (laminar or turbulent). Yplus is calculated in a manner consistent with how it is calculated in the solution procedure. Max/min y^+ values are provided as output at every wall. This option computes and stores the entire y^+ field along the wall. Since this option computes y^+ using the most recent values for all solution variables, the max/min written to the log may differ from the max/min of this field. However, these will become identical at convergence.

Note that this line command does not automatically output this field to a results file. For that, one must use the standard output line command for a nodal field using the string name, **yplus**.

Postprocess

Syntax

Postprocess *FluxType* Flux Of {*conserved_enthalpy* | *continuity* | *edc_product* | *enthalpy* | *mixture_fraction* | *nuclei* | *progress_variable* | *scalar_variance* | *second_mixture_fraction* | *solid_volume_fraction* | *soot* | *species* | *temperature* | *turbulent_dissipation* | *turbulent_frequency* | *turbulent_helmholtz_function* | *turbulent_kinetic_energy* | *turbulent_v2* | *volume_of_fluid* | *x_momentum* | *x_solid_momentum* | *y_momentum* | *y_solid_momentum* | *z_momentum* | *z_solid_momentum*} [As *aliases...*]

Summary

Enable nodal flux post-processing for a given equation. Flux types can be “total”, “advective”, or “diffusive”. Integrated fluxes will also be output to global variables for post-processing or use in other string functions.

By default the variables will be named based on the equation and sideset they are applied to as `bc_FluxType_EquationName_flux_SurfaceName`. However, if you want to assign a more compact or descriptive name you can provide it with

```
POSTPROCESS TOTAL FLUX OF Continuity AS mdot1
```

When post-processing enthalpy, the average temperature is also output as a global variable. If you provide only one alias the enthalpy flux uses your alias and the average temperature is automatically named. If you provide two aliases the first is used for enthalpy and the second is used for temperature.

```
POSTPROCESS TOTAL FLUX OF ENTHALPY AS hFlux
POSTPROCESS TOTAL FLUX OF ENTHALPY AS hFlux Tavg
```

When post-processing species, the post-processor is run for each species that is solved for. This means that there is no post-processor run on the last species (which is determined by a fractional balance). If you provide aliases for the species post-processor you should provide one for each post-processed species. For example, in a problem with O2, CO2, and N2 (in that order) you could use:

```
POSTPROCESS TOTAL FLUX OF SPECIES AS mdotO2 mdotCO2
```

Parameter	Value	Default
<i>FluxType</i>	string	–
<i>equation</i>	{conserved_enthalpy continuity edc_product enthalpy mixture_fraction nuclei progress_variable scalar_variance second_mixture_fraction solid_volume_fraction soot species temperature turbulent_dissipation turbulent_frequency turbulent_helmholtz_function turbulent_kinetic_energy turbulent_v2 volume_of_fluid x_momentum x_solid_momentum y_momentum y_solid_momentum z_momentum z_solid_momentum}	–

Project Nodes

Syntax

Project Nodes

Summary

Allow for nodal projection on the particular mesh part.

Description

The nodal projection of velocity is defaulted to occur on open boundary conditions only. Wall and inflow BCs are defaulted to not be projected, even on flux inflow (i.e., USE FLUXES) and turbulent wall nodes. This option flags the particular mesh part to be included in the nodes to be projected.

Progress_Variable

Syntax

Progress_Variable *ProgressVariableName* {=} *Value*

Summary

Constant value for the progress variable of selected scalars.

Parameter	Value	Default
<i>ProgressVariableName</i>	string	–
{=}	{= are is}	–
<i>Value</i>	real	–

Real Data For Subroutine

Syntax

Real Data For Subroutine *SubName* {=} *Values...*

Summary

List of real data values to be passed down in to the user subroutine. These values

may be changed by the user subroutine.

Parameter	Value	Default
<i>SubName</i>	string	—
<i>{=}</i>	{= are is}	—
<i>Values</i>	real. . .	—

Radiation Boundary Temperature

Syntax

Radiation Boundary Temperature *{=}* *value*

Summary

Constant value for radiation_boundary_temperature variable.

Parameter	Value	Default
<i>{=}</i>	{= are is}	—
<i>value</i>	“string”	—

Radiation Boundary Temperature Field

Syntax

Radiation Boundary Temperature Field *{=}* *FieldName*

Summary

Name of the field to use for the radiation boundary temperature.

Parameter	Value	Default
<i>{=}</i>	{= are is}	—
<i>FieldName</i>	string	—

Radiation Environment Temperature

Syntax

Radiation Environment Temperature *{=}* *value*

Summary

Constant value for radiation_environment_temperature variable.

Parameter	Value	Default
<i>{=}</i>	{= are is}	—
<i>value</i>	“string”	—

Radiation Environment Temperature Field

Syntax

Radiation Environment Temperature Field *{=}* *FieldName*

Summary

Name of the field to use for the radiation environment temperature.

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>FieldName</i>	string	–

Subroutine For

Syntax

Subroutine For *{contact_angle | edc_product | gas_volume_fraction | mixture_fraction | pressure | progress_variable | scalar_variance | second_mixture_fraction | solid_volume_fraction | soot_mass_fraction | soot_nuclei_mass_fraction | temperature | turbulent_dissipation | turbulent_frequency | turbulent_helmholtz_function | turbulent_kinetic_energy | turbulent_v2 | volume_of_fluid | x_solid_velocity | x_velocity | y_solid_velocity | y_velocity | z_solid_velocity | z_velocity}* *{=}* *Subroutine*

Summary

Name of the subroutine to use for this variable.

Parameter	Value	Default
<i>Primitive-Variable</i>	{contact_angle edc_product gas_volume_fraction mixture_fraction pressure progress_variable scalar_variance second_mixture_fraction solid_volume_fraction soot_mass_fraction soot_nuclei_mass_fraction temperature turbulent_dissipation turbulent_frequency turbulent_helmholtz_function turbulent_kinetic_energy turbulent_v2 volume_of_fluid x_solid_velocity x_velocity y_solid_velocity y_velocity z_solid_velocity z_velocity}	–
<i>{=}</i>	{= are is}	–
<i>Subroutine</i>	string	–

Subroutine For Emissivity

Syntax

Subroutine For Emissivity *{=}* *subroutineName*

Summary

Name of the user subroutine to use for the emissivity variable.

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>subroutineName</i>	string	–

Subroutine For Mass_Fraction

Syntax

Subroutine For Mass_Fraction *{=}* *Subroutine*

Summary

Name of the subroutine to use for the mass fraction. ALL species mass fractions must be assigned by this subroutine.

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>Subroutine</i>	string	–

Subroutine For Mole_Fraction

Syntax

Subroutine For Mole_Fraction *{=}* *Subroutine*

Summary

Name of the subroutine to use for the mole fractions. ALL species mole fractions must be assigned by this subroutine.

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>Subroutine</i>	string	–

Subroutine For Radiation Boundary Temperature

Syntax

Subroutine For Radiation Boundary Temperature *{=}* *subroutineName*

Summary

Name of the user subroutine to use for the radiation_boundary_temperature variable.

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>subroutineName</i>	string	–

Subroutine For Radiation Environment Temperature

Syntax

Subroutine For Radiation Environment Temperature *{=} subroutineName*

Summary

Name of the user subroutine to use for the radiation_environment_temperature variable.

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>subroutineName</i>	string	–

Subroutine For Transmissivity**Syntax**

Subroutine For Transmissivity *{=} subroutineName*

Summary

Name of the user subroutine to use for the transmissivity variable.

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>subroutineName</i>	string	–

Subroutine For Transparent Band Emissivity**Syntax**

Subroutine For Transparent Band Emissivity *{=} subroutineName*

Summary

Name of the user subroutine to use for the transparent emissivity band for spectral emissivity on surfaces.

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>subroutineName</i>	string	–

Subroutine For Wall Temperature**Syntax**

Subroutine For Wall Temperature *{=} subroutineName*

Summary

Name of the user subroutine to use for the wall_temperature variable.

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>subroutineName</i>	string	–

Transparent Band Emissivity

Syntax

Transparent Band Emissivity {=} *value*

Summary

Constant value to use for the emissivity in the transparent band for spectral surfaces.

Parameter	Value	Default
{=}	{= are is }	–
<i>value</i>	“string”	–

Transmissivity

Syntax

Transmissivity {=} *value*

Summary

Constant value for transmissivity variable.

Parameter	Value	Default
{=}	{= are is }	–
<i>value</i>	“string”	–

Use Equilibrium Production Model

Syntax

Use Equilibrium Production Model

Summary

Change near-wall turbulence production.

Description

This option specifies that the near wall turb_ke production term is given by:

$$\frac{\tau_w^2}{\kappa^4 \sqrt{C_\mu} \rho y_p \sqrt{k}}$$

. Modifications are made for the KW and SST model. This model is suggested when using the standard KW and SST model when a Dirichlet condition is not placed on the wall node of turbulence kinetic energy.

Use Law Of Wall Blowing Correction

Syntax

Use Law Of Wall Blowing Correction

Summary

Modify wall friction velocity LOW extraction to include the effect of mass injection. The constant is generally roughly ten.

Description

This option allows for modification of the standard law of the wall by blowing effects.

Use Neumann Condition For Ksgs

Syntax

Use Neumann Condition For Ksgs

Summary

Apply a Neumann bc for the one-equation turbulent SGS equation.

Description

This option specifies that the normal component of the subgrid scale gradient is zero at the wall. The standard SGS production and dissipation term are used. This method is preferred over the USE EQUILIBRIUM PRODUCTION MODEL option that is most suitable for RANS. Moreover, the user should remove the OMIT NEAR WALL TURBULENT KE TRANSPORT EQUATION option from solution options and ensure that DETERMINE UTAU VIA NONLINEAR LAW OF THE WALL ITERATION has been activated.

Wall Temperature

Syntax

Wall Temperature {=} *value*

Summary

Value for wall_temperature variable. Can be a constant or a string function of space (x,y,z), time (t) or global variables.

Parameter	Value	Default
{=}	{= are is}	–
<i>value</i>	“string”	–

Law_Of_Wall_Blowing_Parameter

Syntax

Law_Of_Wall_Blowing_Parameter {=} *Law of Wall Blowing parameter*

Summary

Specify mass injection blowing correction; usually roughly ten.

Parameter	Value	Default
{=}	{= are is}	–
<i>Law of Wall Blowing parameter</i>	real	10.0

Law_Of_Wall_Roughness_Parameter

Syntax

Law_Of_Wall_Roughness_Parameter {=} *Law of the Wall Roughness parameter*

Summary

Specify dimensionless roughness factor to be used in the law-of-the-wall formulation.

Parameter	Value	Default
{=}	{= are is}	–
<i>Law of the Wall Roughness parameter</i>	real	9.8

Projected Distance For Exchange Location

Syntax

Projected Distance For Exchange Location {=} *value*

Summary

Specify projected distance for the exchange-based model; generally roughly three elements.

Parameter	Value	Default
{=}	{= are is}	–
<i>value</i>	real	0.0

Post Process On Surface

Scope

Fuego Region

Summary

Defines a wall boundary condition on a named surface of the mesh.

begin Post Process On Surface *Surfacename*

Post Processing File Name {=} *Name*

Reference Temperature {=} *Temperature*

end Post Process On Surface *Surfacename*

Line Commands

Post Processing File Name

Syntax

Post Processing File Name **{=}** *Name*

Summary

Specify the base file name for post-processing surface data.

Parameter	Value	Default
{=}	{= are is}	–
<i>Name</i>	string	–

Reference Temperature

Syntax

Reference Temperature **{=}** *Temperature*

Summary

Constant value for the reference temperature in consistent units.

Parameter	Value	Default
{=}	{= are is}	–
<i>Temperature</i>	real	–

Composite Boundary Condition On Surface

Scope

Fuego Region

Summary

Defines a thermally-decomposing composite material wall boundary condition that outgasses flammable species when heated sufficiently.

begin Composite Boundary Condition On Surface *Surfacename*

```
{contact_angle | edc_product | gas_volume_fraction | mixture_fraction_  
↪| pressure | progress_variable | scalar_variance | second_mixture_  
↪fraction | solid_volume_fraction | soot_mass_fraction | soot_nuclei_  
↪mass_fraction | temperature | turbulent_dissipation | turbulent_  
↪frequency | turbulent_helmholtz_function | turbulent_kinetic_energy_  
↪| turbulent_v2 | volume_of_fluid | x_solid_velocity | x_velocity | y_  
↪solid_velocity | y_velocity | z_solid_velocity | z_velocity} {=} Value
```

Activate Backside Convection

Activate Backside Radiation

Activate Particle Deposition

Calculate Convection Coefficient Using Tref {=} *Tref* ([Dt_Min {=} *Dtmin*] | [{clip}])

Composite {backside_convection_coefficient | backside_reference_
 → temperature | initial_temperature | max_nonlinear_iterations | max_
 → thickness | min_nonlinear_iterations | nodes | residual_tolerance |
 → subcycle_safety_factor | thickness} {=} *value*

Composite Initial_Mass_Fraction *Species* {=} *value*

Composite Material {=} *material_name*

Emissivity Spectral File Name {=} *Name*

External Field For *VariableName* [{of} *Species*] {=} *ExtFieldName* [Of_
 → Size *Value* With Multiplier {=} *Multiplier*]

External Field For Composite {backside_convection_coefficient |
 → backside_reference_temperature | initial_temperature | max_nonlinear_
 → iterations | max_thickness | min_nonlinear_iterations | nodes_
 → | residual_tolerance | subcycle_safety_factor | thickness} {=}
 → *ExtFieldName*

Emissivity {=} *value*

Function For {contact_angle | edc_product | gas_volume_fraction |
 → mixture_fraction | pressure | progress_variable | scalar_variance |
 → second_mixture_fraction | solid_volume_fraction | soot_mass_fraction_
 → | soot_nuclei_mass_fraction | temperature | turbulent_dissipation |
 → turbulent_frequency | turbulent_helmholtz_function | turbulent_kinetic_
 → energy | turbulent_v2 | volume_of_fluid | x_solid_velocity | x_velocity_
 → | y_solid_velocity | y_velocity | z_solid_velocity | z_velocity} {=}
 → *FuncName* [In The {t | x | y | z} Direction]

Function For Composite {backside_convection_coefficient | backside_
 → reference_temperature | initial_temperature | max_nonlinear_iterations_
 → | max_thickness | min_nonlinear_iterations | nodes | residual_tolerance_
 → | subcycle_safety_factor | thickness} {=} *functionName* [In The {t | x_
 → | y | z} Direction]

Function For Composite Initial_Mass_Fraction {=} *funcName* In The {t |
→x | y | z} Direction

Function For Emissivity {=} *functionName* [In The {t | x | y | z}_
→Direction]

Function For Mass_Fraction {=} *FuncName* In The {t | x | y | z}_
→Direction

Function For Mole_Fraction {=} *FuncName* In The {t | x | y | z}_
→Direction

Function For Radiation Boundary Temperature {=} *functionName*

Function For Radiation Environment Temperature {=} *functionName*

Function For Transmissivity {=} *functionName* [In The {t | x | y | z}_
→Direction]

Function For Transparent Band Emissivity {=} *functionName* [In The {t_
→| x | y | z} Direction]

Function For Wall Temperature {=} *functionName* [In The {t | x | y | z}
→ Direction]

Inline Function For {contact_angle | edc_product | gas_volume_fraction_
→| mixture_fraction | pressure | progress_variable | scalar_variance |_
→second_mixture_fraction | solid_volume_fraction | soot_mass_fraction_
→| soot_nuclei_mass_fraction | temperature | turbulent_dissipation |_
→turbulent_frequency | turbulent_helmholtz_function | turbulent_kinetic_
→energy | turbulent_v2 | volume_of_fluid | x_solid_velocity | x_velocity_
→| y_solid_velocity | y_velocity | z_solid_velocity | z_velocity} {=}_
→*FuncStr*

Integer Data For Subroutine *SubName* {=} *Values...*

Interface Boundary

Mass_Fraction *Species* {=} *Mass fraction*

Mole_Fraction *Species* {=} *Mole fraction*

Particle Species {=} *species_name...*

Post Process Yplus

```
Postprocess FluxType Flux Of {conserved_enthalpy | continuity | edc_  
→product | enthalpy | mixture_fraction | nuclei | progress_variable_  
→| scalar_variance | second_mixture_fraction | solid_volume_fraction_  
→| soot | species | temperature | turbulent_dissipation | turbulent_  
→frequency | turbulent_helmholtz_function | turbulent_kinetic_energy_  
→| turbulent_v2 | volume_of_fluid | x_momentum | x_solid_momentum |_  
→y_momentum | y_solid_momentum | z_momentum | z_solid_momentum} [ As_  
→aliases... ]
```

Project Nodes

Progress_Variable **ProgressVariableName** {=} **Value**

Real Data For Subroutine **SubName** {=} **Values...**

Radiation Boundary Temperature {=} **value**

Radiation Boundary Temperature Field {=} **FieldName**

Radiation Environment Temperature {=} **value**

Radiation Environment Temperature Field {=} **FieldName**

```
Subroutine For {contact_angle | edc_product | gas_volume_fraction |_  
→mixture_fraction | pressure | progress_variable | scalar_variance |_  
→second_mixture_fraction | solid_volume_fraction | soot_mass_fraction_  
→| soot_nuclei_mass_fraction | temperature | turbulent_dissipation |_  
→turbulent_frequency | turbulent_helmholtz_function | turbulent_kinetic_  
→energy | turbulent_v2 | volume_of_fluid | x_solid_velocity | x_velocity_  
→| y_solid_velocity | y_velocity | z_solid_velocity | z_velocity} {=}_  
→Subroutine
```

```
Subroutine For Composite {backside_convection_coefficient | backside_  
→reference_temperature | initial_temperature | max_nonlinear_iterations_  
→| max_thickness | min_nonlinear_iterations | nodes | residual_tolerance_  
→| subcycle_safety_factor | thickness} {=} subroutineName
```

Subroutine For Emissivity {=} **subroutineName**

Subroutine For Mass_Fraction {=} **Subroutine**

Subroutine For Mole_Fraction {=} **Subroutine**

Subroutine For Radiation Boundary Temperature {=} *subroutineName*

Subroutine For Radiation Environment Temperature {=} *subroutineName*

Subroutine For Transmissivity {=} *subroutineName*

Subroutine For Transparent Band Emissivity {=} *subroutineName*

Subroutine For Wall Temperature {=} *subroutineName*

Transparent Band Emissivity {=} *value*

Transmissivity {=} *value*

Use Equilibrium Production Model

Wall Temperature {=} *value*

Law_Of_Wall_Roughness_Parameter {=} *Law of the Wall Roughness parameter*

end Composite Boundary Condition On Surface *Surfacename*

Line Commands

Primitivevariable

Syntax

Primitivevariable {*contact_angle* | *edc_product* | *gas_volume_fraction* | *mixture_fraction* | *pressure* | *progress_variable* | *scalar_variance* | *second_mixture_fraction* | *solid_volume_fraction* | *soot_mass_fraction* | *soot_nuclei_mass_fraction* | *temperature* | *turbulent_dissipation* | *turbulent_frequency* | *turbulent_helmholtz_function* | *turbulent_kinetic_energy* | *turbulent_v2* | *volume_of_fluid* | *x_solid_velocity* | *x_velocity* | *y_solid_velocity* | *y_velocity* | *z_solid_velocity* | *z_velocity*} {=} *Value*

Summary

Value for the specified variable (in consistent units). Value can be a constant or a string function of position (x,y,z), time (t), and any global variable.

The function string must be enclosed in quotes if it has spaces or commas. For example: *x_velocity* = "min(1, 0.1*t)"

Parameter	Value	Default
<i>Primitive-Variable</i>	{contact_angle edc_product gas_volume_fraction mixture_fraction pressure progress_variable scalar_variance second_mixture_fraction solid_volume_fraction soot_mass_fraction soot_nuclei_mass_fraction temperature turbulent_dissipation turbulent_frequency turbulent_helmholtz_function turbulent_kinetic_energy turbulent_v2 volume_of_fluid x_solid_velocity x_velocity y_solid_velocity y_velocity z_solid_velocity z_velocity}	–
<i>{=}</i>	{= are is}	–
<i>Value</i>	“string”	–

Activate Backside Convection

Syntax

Activate Backside Convection

Summary

Turn on model for convective heat transfer on the back side of the composite material. A convection coefficient and reference temperature must also be specified.

Activate Backside Radiation

Syntax

Activate Backside Radiation

Summary

Turn on model for radiative heat transfer on the back side of the composite material. A reference temperature must also be specified.

Activate Particle Deposition

Syntax

Activate Particle Deposition

Summary

Turn on deposition of material from the accumulation of particles on the composite material. A species name for the particles must also be supplied. Currently only one species is supported.

Calculate Convection Coefficient

Syntax

Calculate Convection Coefficient Using Tref *{=}* *Tref* ([Dt_Min *{=}* *DTmin*] | [*clip*])

Summary

Calculates the convection coefficient on the wall using a user-specified value for the reference temperature. This calculates $h = Q/(T_{wall} - T_{ref})$.

T_{ref} can be a constant, or a string function of time and global variables.

By default, h is set to 0 if $T_{wall} - T_{ref} < 1 \times 10^{-6}$ however a different tolerance can be supplied with ΔT_{min} .

The optional 'clip' argument will clip the calculated value of h to be non-negative.

Parameter	Value	Default
<i>{=}</i>	{= are is}	—
<i>Tref</i>	"string"	—
<i>{=}</i>	{= are is}	—
<i>DTmin</i>	real	—
<i>Clip</i>	{clip}	—

Composite

Syntax

Composite *{backside_convection_coefficient | backside_reference_temperature | initial_temperature | max_nonlinear_iterations | max_thickness | min_nonlinear_iterations | nodes | residual_tolerance | subcycle_safety_factor | thickness} {=} value*

Summary

Parameters for composite fire model.

Parameter	Value	Default
<i>Composite-Variable</i>	{backside_convection_coefficient backside_reference_temperature initial_temperature max_nonlinear_iterations max_thickness min_nonlinear_iterations nodes residual_tolerance subcycle_safety_factor thickness}	—
<i>{=}</i>	{= are is}	—
<i>value</i>	real	—

Composite Initial_Mass_Fraction

Syntax

Composite Initial_Mass_Fraction *Species {=} value*

Summary

Constant value for the initial mass fraction of selected species in the composite.

Parameter	Value	Default
<i>Species</i>	string	–
<i>{=}</i>	{= are is}	–
<i>value</i>	“string”	–

Composite Material

Syntax

Composite Material *{=}* *material_name*

Summary

Name of the Fuego material to use for the composite BC.

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>material_name</i>	string	–

Emissivity Spectral File Name

Syntax

Emissivity Spectral File Name *{=}* *Name*

Summary

Specify the file name for defining spectral radiation properties on a surface.

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>Name</i>	string	–

External Field For

Syntax

External Field For *VariableName* [*{of}* *Species*] *{=}* *ExtFieldName* [Of Size *Value* With Multiplier *{=}* *Multiplier*]

Summary

Name of the field that is to be transferred in. For species equation, specify variable as “mass fraction of” followed by the species name for transfers associated with separated individual species field names.

Parameter	Value	Default
<i>VariableName</i>	string	–
<i>Species</i>	string	–
<i>{=}</i>	{= are is}	–
<i>ExtFieldName</i>	string	–

External Field For Composite

Syntax

External Field For Composite *{backside_convection_coefficient | backside_reference_temperature | initial_temperature | max_nonlinear_iterations | max_thickness | min_nonlinear_iterations | nodes | residual_tolerance | subcycle_safety_factor | thickness} {=} ExtFieldName*

Summary

Name of an external field to use for composite variable.

Parameter	Value	Default
<i>Composite-Variable</i>	{backside_convection_coefficient backside_reference_temperature initial_temperature max_nonlinear_iterations max_thickness min_nonlinear_iterations nodes residual_tolerance subcycle_safety_factor thickness}	–
<i>{=}</i>	{= are is}	–
<i>ExtFieldName</i>	string	–

Emissivity

Syntax

Emissivity *{=} value*

Summary

Constant value for emissivity variable.

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>value</i>	“string”	–

Function For

Syntax

Function For *{contact_angle | edc_product | gas_volume_fraction | mixture_fraction | pressure | progress_variable | scalar_variance | second_mixture_fraction | solid_volume_fraction | soot_mass_fraction | soot_nuclei_mass_fraction | temperature | turbulent_dissipation | turbulent_frequency | turbulent_helmholtz_function | turbulent_kinetic_energy | turbulent_v2 | volume_of_fluid | x_solid_velocity | x_velocity | y_solid_velocity | y_velocity | z_solid_velocity | z_velocity} {=} FuncName [In The {t | x | y | z} Direction]*

Summary

Name of function to use for the given variable, in the specified direction.

Parameter	Value	Default
<i>Primitive-Variable</i>	{contact_angle edc_product gas_volume_fraction mixture_fraction pressure progress_variable scalar_variance second_mixture_fraction solid_volume_fraction soot_mass_fraction soot_nuclei_mass_fraction temperature turbulent_dissipation turbulent_frequency turbulent_helmholtz_function turbulent_kinetic_energy turbulent_v2 volume_of_fluid x_solid_velocity x_velocity y_solid_velocity y_velocity z_solid_velocity z_velocity}	–
<i>{=}</i>	{= are is}	–
<i>Function-Name</i>	string	–

Function For Composite

Syntax

Function For Composite *{backside_convection_coefficient | backside_reference_temperature | initial_temperature | max_nonlinear_iterations | max_thickness | min_nonlinear_iterations | nodes | residual_tolerance | subcycle_safety_factor | thickness}* *{=}* *functionName* [In The *{t | x | y | z}* Direction]

Summary

Name of function to use for the given composite fire variable, in the specified direction.

Parameter	Value	Default
<i>Composite-Variable</i>	{backside_convection_coefficient backside_reference_temperature initial_temperature max_nonlinear_iterations max_thickness min_nonlinear_iterations nodes residual_tolerance subcycle_safety_factor thickness}	–
<i>{=}</i>	{= are is}	–
<i>function-Name</i>	string	–

Function For Composite Initial_Mass_Fraction

Syntax

Function For Composite Initial_Mass_Fraction *{=}* *funcName* In The *{t | x | y | z}* Direction

Summary

Name of the mass fraction function to use for all species in the composite in a given direction. Note that this must be a Multicolumn function

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>funcName</i>	string	–
<i>Direction</i>	{ t x y z }	–

Function For Emissivity

Syntax

Function For Emissivity *{=}* *functionName* [In The *{t | x | y | z}* Direction]

Summary

Name of the function to use for the emissivity in the given direction (default direction = time).

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>functionName</i>	string	–

Function For Mass_Fraction

Syntax

Function For Mass_Fraction *{=}* *FuncName* In The *{t | x | y | z}* Direction

Summary

Name of the mass fraction function to use for all species in the given direction. Note that this must be a Multicolumn function

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>FuncName</i>	string	–
<i>Direction</i>	{ t x y z }	–

Function For Mole_Fraction

Syntax

Function For Mole_Fraction *{=}* *FuncName* In The *{t | x | y | z}* Direction

Summary

Name of the mass fraction function to use for all species in the given direction. Note that this must be a Multicolumn function

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>FuncName</i>	string	–
<i>Direction</i>	{ t x y z }	–

Function For Radiation Boundary Temperature

Syntax

Function For Radiation Boundary Temperature *{=}* *functionName*

Summary

Name of function to use for the radiation_boundary_temperature variable

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>functionName</i>	string	–

Function For Radiation Environment Temperature

Syntax

Function For Radiation Environment Temperature *{=}* *functionName*

Summary

Name of function to use for the radiation_environment_temperature variable

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>functionName</i>	string	–

Function For Transmissivity

Syntax

Function For Transmissivity *{=}* *functionName* [In The *{t | x | y | z}* Direction]

Summary

Name of function to use for the transmissivity variable

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>functionName</i>	string	–

Function For Transparent Band Emissivity

Syntax

Function For Transparent Band Emissivity *{=}* *functionName* [In The *{t | x | y | z}* Direction]

Summary

Name of function to use for the transparent emissivity band for spectral emissivity on surfaces.

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>functionName</i>	string	–

Function For Wall Temperature

Syntax

Function For Wall Temperature *{=}* *functionName* [In The *{t | x | y | z}* Direction]

Summary

Name of function to use for the wall_temperature variable, in the specified direction.

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>functionName</i>	string	–

Inline Function For

Syntax

Inline Function For *{contact_angle | edc_product | gas_volume_fraction | mixture_fraction | pressure | progress_variable | scalar_variance | second_mixture_fraction | solid_volume_fraction | soot_mass_fraction | soot_nuclei_mass_fraction | temperature | turbulent_dissipation | turbulent_frequency | turbulent_helmholtz_function | turbulent_kinetic_energy | turbulent_v2 | volume_of_fluid | x_solid_velocity | x_velocity | y_solid_velocity | y_velocity | z_solid_velocity | z_velocity}* *{=}* *FuncStr*

Summary

warning{This command is deprecated. Regular BCs now support using string functions directly}

Parameter	Value	Default
<i>Primitive-Variable</i>	{contact_angle edc_product gas_volume_fraction mixture_fraction pressure progress_variable scalar_variance second_mixture_fraction solid_volume_fraction soot_mass_fraction soot_nuclei_mass_fraction temperature turbulent_dissipation turbulent_frequency turbulent_helmholtz_function turbulent_kinetic_energy turbulent_v2 volume_of_fluid x_solid_velocity x_velocity y_solid_velocity y_velocity z_solid_velocity z_velocity}	–
<i>{=}</i>	{= are is}	–
<i>Func-Str</i>	“string”	–

Integer Data For Subroutine

Syntax

Integer Data For Subroutine *SubName* *{=}* *Values* . .

Summary

List of integer data values to be passed down in to the user subroutine. These values may be changed by the user subroutine.

Parameter	Value	Default
<i>SubName</i>	string	–
<i>{=}</i>	{= are is}	–
<i>Values</i>	integer. . .	–

Interface Boundary

Syntax

Interface Boundary

Summary

Mark this fluids boundary as an interface between two physical regions. Data will be interpolated across this boundary.

Mass_Fraction

Syntax

Mass_Fraction *Species* *{=}* *Mass fraction*

Summary

Value for the mass fraction of selected species. Can be a constant value or a string function of space (x,y,z), time (t), and any global variable.

Parameter	Value	Default
<i>Species</i>	string	–
<i>{=}</i>	{ = are is }	–
<i>Mass fraction</i>	“string”	–

Mole_Fraction

Syntax

Mole_Fraction *Species {=} Mole fraction*

Summary

Value for the mole fraction of selected species. Can be a constant value or a string function of space (x,y,z), time (t), and any global variable.

Parameter	Value	Default
<i>Species</i>	string	–
<i>{=}</i>	{ = are is }	–
<i>Mole fraction</i>	“string”	–

Particle Species

Syntax

Particle Species *{=} species_name. . .*

Summary

Names of the particle species that can accumulate on the composite BC (this must be the complete list of species in the particle definition, listed in the same order as defined in the particle definition).

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>species_name</i>	string. . .	–

Post Process Yplus

Syntax

Post Process Yplus

Summary

Request post processing of yplus on this mesh part.

Description

The normalized distance to the wall (y^+) can be post-processed on any wall (laminar or turbulent). Yplus is calculated in a manner consistent with how it is calculated in the solution procedure. Max/min y^+ values are provided as output at every wall. This option computes and stores the entire y^+ field along the wall.

Since this option computes y^+ using the most recent values for all solution variables, the max/min written to the log may differ from the max/min of this field. However, these will become identical at convergence.

Note that this line command does not automatically output this field to a results file. For that, one must use the standard output line command for a nodal field using the string name, `yplus`.

Postprocess

Syntax

Postprocess *FluxType* Flux Of {*conserved_enthalpy* | *continuity* | *edc_product* | *enthalpy* | *mixture_fraction* | *nuclei* | *progress_variable* | *scalar_variance* | *second_mixture_fraction* | *solid_volume_fraction* | *soot* | *species* | *temperature* | *turbulent_dissipation* | *turbulent_frequency* | *turbulent_helmholtz_function* | *turbulent_kinetic_energy* | *turbulent_v2* | *volume_of_fluid* | *x_momentum* | *x_solid_momentum* | *y_momentum* | *y_solid_momentum* | *z_momentum* | *z_solid_momentum*} [As *aliases...*]

Summary

Enable nodal flux post-processing for a given equation. Flux types can be “total”, “advective”, or “diffusive”. Integrated fluxes will also be output to global variables for post-processing or use in other string functions.

By default the variables will be named based on the equation and sideset they are applied to as `bc_FluxType_EquationName_flux_SurfaceName`. However, if you want to assign a more compact or descriptive name you can provide it with

```
POSTPROCESS TOTAL FLUX OF Continuity AS mdot1
```

When post-processing enthalpy, the average temperature is also output as a global variable. If you provide only one alias the enthalpy flux uses your alias and the average temperature is automatically named. If you provide two aliases the first is used for enthalpy and the second is used for temperature.

```
POSTPROCESS TOTAL FLUX OF ENTHALPY AS hFlux
POSTPROCESS TOTAL FLUX OF ENTHALPY AS hFlux Tavg
```

When post-processing species, the post-processor is run for each species that is solved for. This means that there is no post-processor run on the last species (which is determined by a fractional balance). If you provide aliases for the species post-processor you should provide one for each post-processed species. For example, in a problem with O2, CO2, and N2 (in that order) you could use:

```
POSTPROCESS TOTAL FLUX OF SPECIES AS mdotO2 mdotCO2
```

Parameter	Value	Default
<i>FluxType</i>	string	–
<i>equation</i>	{conserved_enthalpy continuity edc_product enthalpy mixture_fraction nuclei progress_variable scalar_variance second_mixture_fraction solid_volume_fraction soot species temperature turbulent_dissipation turbulent_frequency turbulent_helmholtz_function turbulent_kinetic_energy turbulent_v2 volume_of_fluid x_momentum x_solid_momentum y_momentum y_solid_momentum z_momentum z_solid_momentum}	–

Project Nodes

Syntax

Project Nodes

Summary

Allow for nodal projection on the particular mesh part.

Description

The nodal projection of velocity is defaulted to occur on open boundary conditions only. Wall and inflow BCs are defaulted to not be projected, even on flux inflow (i.e., USE FLUXES) and turbulent wall nodes. This option flags the particular mesh part to be included in the nodes to be projected.

Progress_Variable

Syntax

Progress_Variable *ProgressVariableName* {=} *Value*

Summary

Constant value for the progress variable of selected scalars.

Parameter	Value	Default
<i>ProgressVariableName</i>	string	–
{=}	{= are is}	–
<i>Value</i>	real	–

Real Data For Subroutine

Syntax

Real Data For Subroutine *SubName* {=} *Values...*

Summary

List of real data values to be passed down in to the user subroutine. These values

may be changed by the user subroutine.

Parameter	Value	Default
<i>SubName</i>	string	—
<i>{=}</i>	{= are is}	—
<i>Values</i>	real. . .	—

Radiation Boundary Temperature

Syntax

Radiation Boundary Temperature *{=}* *value*

Summary

Constant value for radiation_boundary_temperature variable.

Parameter	Value	Default
<i>{=}</i>	{= are is}	—
<i>value</i>	“string”	—

Radiation Boundary Temperature Field

Syntax

Radiation Boundary Temperature Field *{=}* *FieldName*

Summary

Name of the field to use for the radiation boundary temperature.

Parameter	Value	Default
<i>{=}</i>	{= are is}	—
<i>FieldName</i>	string	—

Radiation Environment Temperature

Syntax

Radiation Environment Temperature *{=}* *value*

Summary

Constant value for radiation_environment_temperature variable.

Parameter	Value	Default
<i>{=}</i>	{= are is}	—
<i>value</i>	“string”	—

Radiation Environment Temperature Field

Syntax

Radiation Environment Temperature Field *{=}* *FieldName*

Summary

Name of the field to use for the radiation environment temperature.

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>FieldName</i>	string	–

Subroutine For

Syntax

Subroutine For *{contact_angle | edc_product | gas_volume_fraction | mixture_fraction | pressure | progress_variable | scalar_variance | second_mixture_fraction | solid_volume_fraction | soot_mass_fraction | soot_nuclei_mass_fraction | temperature | turbulent_dissipation | turbulent_frequency | turbulent_helmholtz_function | turbulent_kinetic_energy | turbulent_v2 | volume_of_fluid | x_solid_velocity | x_velocity | y_solid_velocity | y_velocity | z_solid_velocity | z_velocity}* *{=}* *Subroutine*

Summary

Name of the subroutine to use for this variable.

Parameter	Value	Default
<i>Primitive-Variable</i>	{contact_angle edc_product gas_volume_fraction mixture_fraction pressure progress_variable scalar_variance second_mixture_fraction solid_volume_fraction soot_mass_fraction soot_nuclei_mass_fraction temperature turbulent_dissipation turbulent_frequency turbulent_helmholtz_function turbulent_kinetic_energy turbulent_v2 volume_of_fluid x_solid_velocity x_velocity y_solid_velocity y_velocity z_solid_velocity z_velocity}	–
<i>{=}</i>	{= are is}	–
<i>Subroutine</i>	string	–

Subroutine For Composite

Syntax

Subroutine For Composite *{backside_convection_coefficient | backside_reference_temperature | initial_temperature | max_nonlinear_iterations | max_thickness | min_nonlinear_iterations | nodes | residual_tolerance | subcycle_safety_factor | thickness}* *{=}* *subroutineName*

Summary

Name of the user subroutine to use for the given composite fire variable.

Parameter	Value	Default
<i>Composite-Variable</i>	{backside_convection_coefficient backside_reference_temperature initial_temperature max_nonlinear_iterations max_thickness min_nonlinear_iterations nodes residual_tolerance subcycle_safety_factor thickness}	–
<i>{=}</i>	{= are is}	–
<i>sub-routine-Name</i>	string	–

Subroutine For Emissivity

Syntax

Subroutine For Emissivity *{=}* *subroutineName*

Summary

Name of the user subroutine to use for the emissivity variable.

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>subroutineName</i>	string	–

Subroutine For Mass_Fraction

Syntax

Subroutine For Mass_Fraction *{=}* *Subroutine*

Summary

Name of the subroutine to use for the mass fraction. ALL species mass fractions must be assigned by this subroutine.

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>Subroutine</i>	string	–

Subroutine For Mole_Fraction

Syntax

Subroutine For Mole_Fraction *{=}* *Subroutine*

Summary

Name of the subroutine to use for the mole fractions. ALL species mole fractions must be assigned by this subroutine.

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>Subroutine</i>	string	–

Subroutine For Radiation Boundary Temperature

Syntax

Subroutine For Radiation Boundary Temperature *{=}* *subroutineName*

Summary

Name of the user subroutine to use for the radiation_boundary_temperature variable.

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>subroutineName</i>	string	–

Subroutine For Radiation Environment Temperature

Syntax

Subroutine For Radiation Environment Temperature *{=}* *subroutineName*

Summary

Name of the user subroutine to use for the radiation_environment_temperature variable.

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>subroutineName</i>	string	–

Subroutine For Transmissivity

Syntax

Subroutine For Transmissivity *{=}* *subroutineName*

Summary

Name of the user subroutine to use for the transmissivity variable.

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>subroutineName</i>	string	–

Subroutine For Transparent Band Emissivity

Syntax

Subroutine For Transparent Band Emissivity *{=}* *subroutineName*

Summary

Name of the user subroutine to use for the transparent emissivity band for spectral emissivity on surfaces.

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>subroutineName</i>	string	–

Subroutine For Wall Temperature

Syntax

Subroutine For Wall Temperature *{=}* *subroutineName*

Summary

Name of the user subroutine to use for the wall_temperature variable.

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>subroutineName</i>	string	–

Transparent Band Emissivity

Syntax

Transparent Band Emissivity *{=}* *value*

Summary

Constant value to use for the emissivity in the transparent band for spectral surfaces.

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>value</i>	“string”	–

Transmissivity

Syntax

Transmissivity *{=}* *value*

Summary

Constant value for transmissivity variable.

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>value</i>	“string”	–

Use Equilibrium Production Model

Syntax

Use Equilibrium Production Model

Summary

Change near-wall turbulence production.

Description

This option specifies that the near wall turb_ke production term is given by:

$$\frac{\tau_w^2}{\kappa \sqrt[4]{C_\mu} \rho y_p \sqrt{k}}$$

. Modifications are made for the KW and SST model. This model is suggested when using the standard KW and SST model when a Dirichlet condition is not placed on the wall node of turbulence kinetic energy.

Wall Temperature

Syntax

Wall Temperature *{=}* *value*

Summary

Value for wall_temperature variable. Can be a constant or a string function of space (x,y,z), time (t) or global variables.

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>value</i>	“string”	–

Law_Of_Wall_Roughness_Parameter

Syntax

Law_Of_Wall_Roughness_Parameter *{=}* *Law of the Wall Roughness parameter*

Summary

Specify dimensionless roughness factor to be used in the law-of-the-wall formulation.

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>Law of the Wall Roughness parameter</i>	real	9.8

Composite Interface Boundary Condition On Surface

Scope

Fuego Region

Summary

Defines a wall interface boundary condition that outgasses flammable species when heated sufficiently. This boundary condition may be treated as a porous composite material wall where the heat flux, mass transfer and pressure are applied from a coupled region via a transfer. This boundary enforces Dirichlet-Robin coupling.

Sign Convention: If transferring a field for `composite_mass_rate` or `composite_species_mass_rate`, the expected sign convention is positive for flows into the Fuego domain. If the transferred field has the opposite convention use the multiplier option with a value of -1.

begin Composite Interface Boundary Condition On Surface *Surfacename*

```
{contact_angle | edc_product | gas_volume_fraction | mixture_fraction |  
→| pressure | progress_variable | scalar_variance | second_mixture_  
→fraction | solid_volume_fraction | soot_mass_fraction | soot_nuclei_  
→mass_fraction | temperature | turbulent_dissipation | turbulent_  
→frequency | turbulent_helmholtz_function | turbulent_kinetic_energy_  
→| turbulent_v2 | volume_of_fluid | x_solid_velocity | x_velocity | y_  
→solid_velocity | y_velocity | z_solid_velocity | z_velocity} {=} Value
```

```
Calculate Convection Coefficient Using Tref {=} Tref ([Dt_Min {=} DTmin] | [{clip}])
```

```
Composite Initial_Mass_Fraction Species {=} value
```

```
Emissivity Spectral File Name {=} Name
```

```
External Field For VariableName [{of} Species] {=} ExtFieldName [ Of_  
→Size Value With Multiplier {=} Multiplier ]
```

```
Emissivity {=} value
```

```
Function For {contact_angle | edc_product | gas_volume_fraction |   
→mixture_fraction | pressure | progress_variable | scalar_variance |   
→second_mixture_fraction | solid_volume_fraction | soot_mass_fraction_  
→| soot_nuclei_mass_fraction | temperature | turbulent_dissipation |   
→turbulent_frequency | turbulent_helmholtz_function | turbulent_kinetic_  
→energy | turbulent_v2 | volume_of_fluid | x_solid_velocity | x_velocity_  
→| y_solid_velocity | y_velocity | z_solid_velocity | z_velocity} {=} Value
```

```

→FuncName [ In The {t | x | y | z} Direction ]

Function For Composite Initial_Mass_Fraction {=} funcName In The {t |
→x | y | z} Direction

Function For Emissivity {=} functionName [ In The {t | x | y | z}
→Direction ]

Function For Mass_Fraction {=} FuncName In The {t | x | y | z}
→Direction

Function For Mole_Fraction {=} FuncName In The {t | x | y | z}
→Direction

Function For Radiation Boundary Temperature {=} functionName

Function For Radiation Environment Temperature {=} functionName

Function For Transmissivity {=} functionName [ In The {t | x | y | z}
→Direction ]

Function For Transparent Band Emissivity {=} functionName [ In The {t
→| x | y | z} Direction ]

Function For Wall Temperature {=} functionName [ In The {t | x | y | z}
→ Direction ]

Inline Function For {contact_angle | edc_product | gas_volume_fraction
→| mixture_fraction | pressure | progress_variable | scalar_variance |
→second_mixture_fraction | solid_volume_fraction | soot_mass_fraction
→| soot_nuclei_mass_fraction | temperature | turbulent_dissipation |
→turbulent_frequency | turbulent_helmholtz_function | turbulent_kinetic_
→energy | turbulent_v2 | volume_of_fluid | x_solid_velocity | x_velocity
→| y_solid_velocity | y_velocity | z_solid_velocity | z_velocity} {=}
→FuncStr

Integer Data For Subroutine SubName {=} Values...

Interface Boundary

Mass_Fraction Species {=} Mass fraction

Mole_Fraction Species {=} Mole fraction

Post Process Yplus

```

```

Postprocess FluxType Flux Of {conserved_enthalpy | continuity | edc_
↳product | enthalpy | mixture_fraction | nuclei | progress_variable_
↳| scalar_variance | second_mixture_fraction | solid_volume_fraction_
↳| soot | species | temperature | turbulent_dissipation | turbulent_
↳frequency | turbulent_helmholtz_function | turbulent_kinetic_energy_
↳| turbulent_v2 | volume_of_fluid | x_momentum | x_solid_momentum |_
↳y_momentum | y_solid_momentum | z_momentum | z_solid_momentum} [ As_
↳aliases... ]

```

Project Nodes

Progress_Variable *ProgressVariableName* {=} *Value*

Real Data For Subroutine *SubName* {=} *Values...*

Radiation Boundary Temperature {=} *value*

Radiation Boundary Temperature Field {=} *FieldName*

Radiation Environment Temperature {=} *value*

Radiation Environment Temperature Field {=} *FieldName*

```

Subroutine For {contact_angle | edc_product | gas_volume_fraction |_
↳mixture_fraction | pressure | progress_variable | scalar_variance |_
↳second_mixture_fraction | solid_volume_fraction | soot_mass_fraction_
↳| soot_nuclei_mass_fraction | temperature | turbulent_dissipation |_
↳turbulent_frequency | turbulent_helmholtz_function | turbulent_kinetic_
↳energy | turbulent_v2 | volume_of_fluid | x_solid_velocity | x_velocity_
↳| y_solid_velocity | y_velocity | z_solid_velocity | z_velocity} {=}_
↳Subroutine

```

Subroutine For Composite Initial_Mass_Fraction {=} *subName*

Subroutine For Emissivity {=} *subroutineName*

Subroutine For Mass_Fraction {=} *Subroutine*

Subroutine For Mole_Fraction {=} *Subroutine*

Subroutine For Radiation Boundary Temperature {=} *subroutineName*

Subroutine For Radiation Environment Temperature {=} *subroutineName*

Subroutine For Transmissivity {=} *subroutineName*

Subroutine For Transparent Band Emissivity {=} *subroutineName*

Subroutine For Wall Temperature {=} *subroutineName*

Transparent Band Emissivity {=} *value*

Transmissivity {=} *value*

Use Equilibrium Production Model

Wall Temperature {=} *value*

Law_Of_Wall_Roughness_Parameter {=} *Law of the Wall Roughness parameter*

end Composite Interface Boundary Condition On Surface *Surfacename*

Line Commands

Primitivevariable

Syntax

Primitivevariable {*contact_angle* | *edc_product* | *gas_volume_fraction* | *mixture_fraction* | *pressure* | *progress_variable* | *scalar_variance* | *second_mixture_fraction* | *solid_volume_fraction* | *soot_mass_fraction* | *soot_nuclei_mass_fraction* | *temperature* | *turbulent_dissipation* | *turbulent_frequency* | *turbulent_helmholtz_function* | *turbulent_kinetic_energy* | *turbulent_v2* | *volume_of_fluid* | *x_solid_velocity* | *x_velocity* | *y_solid_velocity* | *y_velocity* | *z_solid_velocity* | *z_velocity*} {=} *Value*

Summary

Value for the specified variable (in consistent units). Value can be a constant or a string function of position (x,y,z), time (t), and any global variable.

The function string must be enclosed in quotes if it has spaces or commas. For example: *x_velocity* = "min(1, 0.1*t)"

Parameter	Value	Default
<i>Primitive-Variable</i>	{contact_angle edc_product gas_volume_fraction mixture_fraction pressure progress_variable scalar_variance second_mixture_fraction solid_volume_fraction soot_mass_fraction soot_nuclei_mass_fraction temperature turbulent_dissipation turbulent_frequency turbulent_helmholtz_function turbulent_kinetic_energy turbulent_v2 volume_of_fluid x_solid_velocity x_velocity y_solid_velocity y_velocity z_solid_velocity z_velocity}	–
<i>{=}</i>	{= are is}	–
<i>Value</i>	“string”	–

Calculate Convection Coefficient

Syntax

Calculate Convection Coefficient Using Tref *{=}* *Tref* ([Dt_Min *{=}* *DTmin*] | [*clip*])

Summary

Calculates the convection coefficient on the wall using a user-specified value for the reference temperature. This calculates $h = Q / (T_{wall} - T_{ref})$.

T_{ref} can be a constant, or a string function of time and global variables.

By default, h is set to 0 if $T_{wall} - T_{ref} < 1 \times 10^{-6}$ however a different tolerance can be supplied with ΔT_{min} .

The optional ‘clip’ argument will clip the calculated value of h to be non-negative.

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>Tref</i>	“string”	–
<i>{=}</i>	{= are is}	–
<i>DTmin</i>	real	–
<i>Clip</i>	{clip}	–

Composite Initial_Mass_Fraction

Syntax

Composite Initial_Mass_Fraction *Species* *{=}* *value*

Summary

Constant value for the initial mass fraction of selected species in the composite.

Parameter	Value	Default
<i>Species</i>	string	–
<i>{=}</i>	{ = are is }	–
<i>value</i>	“string”	–

Emissivity Spectral File Name

Syntax

Emissivity Spectral File Name *{=}* *Name*

Summary

Specify the file name for defining spectral radiation properties on a surface.

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>Name</i>	string	–

External Field For

Syntax

External Field For *VariableName* [*{of}* *Species*] *{=}* *ExtFieldName* [Of Size *Value* With Multiplier *{=}* *Multiplier*]

Summary

Name of the field that is to be transferred in. For species equation, specify variable as “mass fraction of” followed by the species name for transfers associated with separated individual species field names.

Parameter	Value	Default
<i>VariableName</i>	string	–
<i>Species</i>	string	–
<i>{=}</i>	{ = are is }	–
<i>ExtFieldName</i>	string	–

Emissivity

Syntax

Emissivity *{=}* *value*

Summary

Constant value for emissivity variable.

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>value</i>	“string”	–

Function For

Syntax

Function For *{contact_angle | edc_product | gas_volume_fraction | mixture_fraction | pressure | progress_variable | scalar_variance | second_mixture_fraction | solid_volume_fraction | soot_mass_fraction | soot_nuclei_mass_fraction | temperature | turbulent_dissipation | turbulent_frequency | turbulent_helmholtz_function | turbulent_kinetic_energy | turbulent_v2 | volume_of_fluid | x_solid_velocity | x_velocity | y_solid_velocity | y_velocity | z_solid_velocity | z_velocity}* *{=}* *FuncName* [In The *{t | x | y | z}* Direction]

Summary

Name of function to use for the given variable, in the specified direction.

Parameter	Value	Default
<i>Primitive-Variable</i>	{contact_angle edc_product gas_volume_fraction mixture_fraction pressure progress_variable scalar_variance second_mixture_fraction solid_volume_fraction soot_mass_fraction soot_nuclei_mass_fraction temperature turbulent_dissipation turbulent_frequency turbulent_helmholtz_function turbulent_kinetic_energy turbulent_v2 volume_of_fluid x_solid_velocity x_velocity y_solid_velocity y_velocity z_solid_velocity z_velocity}	—
<i>{=}</i>	{= are is}	—
<i>FuncName</i>	string	—

Function For Composite Initial_Mass_Fraction

Syntax

Function For Composite Initial_Mass_Fraction *{=}* *funcName* In The *{t | x | y | z}* Direction

Summary

Name of the mass fraction function to use for all species in the composite in a given direction. Note that this must be a Multicolumn function

Parameter	Value	Default
<i>{=}</i>	{= are is}	—
<i>funcName</i>	string	—
<i>Direction</i>	{t x y z}	—

Function For Emissivity

Syntax

Function For Emissivity {=} *functionName* [In The {t | x | y | z} Direction]

Summary

Name of the function to use for the emissivity in the given direction (default direction = time).

Parameter	Value	Default
{=}	{= are is}	–
<i>functionName</i>	string	–

Function For Mass_Fraction

Syntax

Function For Mass_Fraction {=} *FuncName* In The {t | x | y | z} Direction

Summary

Name of the mass fraction function to use for all species in the given direction.
Note that this must be a Multicolumn function

Parameter	Value	Default
{=}	{= are is}	–
<i>FuncName</i>	string	–
<i>Direction</i>	{t x y z}	–

Function For Mole_Fraction

Syntax

Function For Mole_Fraction {=} *FuncName* In The {t | x | y | z} Direction

Summary

Name of the mass fraction function to use for all species in the given direction.
Note that this must be a Multicolumn function

Parameter	Value	Default
{=}	{= are is}	–
<i>FuncName</i>	string	–
<i>Direction</i>	{t x y z}	–

Function For Radiation Boundary Temperature

Syntax

Function For Radiation Boundary Temperature {=} *functionName*

Summary

Name of function to use for the radiation_boundary_temperature variable

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>functionName</i>	string	–

Function For Radiation Environment Temperature

Syntax

Function For Radiation Environment Temperature *{=}* *functionName*

Summary

Name of function to use for the radiation_environment_temperature variable

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>functionName</i>	string	–

Function For Transmissivity

Syntax

Function For Transmissivity *{=}* *functionName* [In The *{t | x | y | z}* Direction]

Summary

Name of function to use for the transmissivity variable

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>functionName</i>	string	–

Function For Transparent Band Emissivity

Syntax

Function For Transparent Band Emissivity *{=}* *functionName* [In The *{t | x | y | z}* Direction]

Summary

Name of function to use for the transparent emissivity band for spectral emissivity on surfaces.

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>functionName</i>	string	–

Function For Wall Temperature

Syntax

Function For Wall Temperature *{=}* *functionName* [In The *{t | x | y | z}* Direction]

Summary

Name of function to use for the wall_temperature variable, in the specified direction.

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>functionName</i>	string	–

Inline Function For

Syntax

Inline Function For *{contact_angle | edc_product | gas_volume_fraction | mixture_fraction | pressure | progress_variable | scalar_variance | second_mixture_fraction | solid_volume_fraction | soot_mass_fraction | soot_nuclei_mass_fraction | temperature | turbulent_dissipation | turbulent_frequency | turbulent_helmholtz_function | turbulent_kinetic_energy | turbulent_v2 | volume_of_fluid | x_solid_velocity | x_velocity | y_solid_velocity | y_velocity | z_solid_velocity | z_velocity}* *{=}* *FuncStr*

Summary

warning{This command is deprecated. Regular BCs now support using string functions directly}

Parameter	Value	Default
<i>Primitive-Variable</i>	{contact_angle edc_product gas_volume_fraction mixture_fraction pressure progress_variable scalar_variance second_mixture_fraction solid_volume_fraction soot_mass_fraction soot_nuclei_mass_fraction temperature turbulent_dissipation turbulent_frequency turbulent_helmholtz_function turbulent_kinetic_energy turbulent_v2 volume_of_fluid x_solid_velocity x_velocity y_solid_velocity y_velocity z_solid_velocity z_velocity}	–
<i>{=}</i>	{ = are is }	–
<i>FuncStr</i>	“string”	–

Integer Data For Subroutine

Syntax

Integer Data For Subroutine *SubName* *{=}* *Values...*

Summary

List of integer data values to be passed down in to the user subroutine. These values may be changed by the user subroutine.

Parameter	Value	Default
<i>SubName</i>	string	—
<i>{=}</i>	{ = are is }	—
<i>Values</i>	integer. . .	—

Interface Boundary

Syntax

Interface Boundary

Summary

Mark this fluids boundary as an interface between two physical regions. Data will be interpolated across this boundary.

Mass_Fraction

Syntax

Mass_Fraction *Species {=} Mass fraction*

Summary

Value for the mass fraction of selected species. Can be a constant value or a string function of space (x,y,z), time (t), and any global variable.

Parameter	Value	Default
<i>Species</i>	string	—
<i>{=}</i>	{ = are is }	—
<i>Mass fraction</i>	“string”	—

Mole_Fraction

Syntax

Mole_Fraction *Species {=} Mole fraction*

Summary

Value for the mole fraction of selected species. Can be a constant value or a string function of space (x,y,z), time (t), and any global variable.

Parameter	Value	Default
<i>Species</i>	string	—
<i>{=}</i>	{ = are is }	—
<i>Mole fraction</i>	“string”	—

Post Process Yplus

Syntax

Post Process Yplus

Summary

Request post processing of yplus on this mesh part.

Description

The normalized distance to the wall (y^+) can be post-processed on any wall (laminar or turbulent). Yplus is calculated in a manner consistent with how it is calculated in the solution procedure. Max/min y^+ values are provided as output at every wall. This option computes and stores the entire y^+ field along the wall. Since this option computes y^+ using the most recent values for all solution variables, the max/min written to the log may differ from the max/min of this field. However, these will become identical at convergence.

Note that this line command does not automatically output this field to a results file. For that, one must use the standard output line command for a nodal field using the string name, yplus.

Postprocess

Syntax

Postprocess *FluxType* Flux Of {*conserved_enthalpy* | *continuity* | *edc_product* | *enthalpy* | *mixture_fraction* | *nuclei* | *progress_variable* | *scalar_variance* | *second_mixture_fraction* | *solid_volume_fraction* | *soot* | *species* | *temperature* | *turbulent_dissipation* | *turbulent_frequency* | *turbulent_helmholtz_function* | *turbulent_kinetic_energy* | *turbulent_v2* | *volume_of_fluid* | *x_momentum* | *x_solid_momentum* | *y_momentum* | *y_solid_momentum* | *z_momentum* | *z_solid_momentum*} [As *aliases...*]

Summary

Enable nodal flux post-processing for a given equation. Flux types can be “total”, “advective”, or “diffusive”. Integrated fluxes will also be output to global variables for post-processing or use in other string functions.

By default the variables will be named based on the equation and sideset they are applied to as bc_FluxType_EquationName_flux_SurfaceName. However, if you want to assign a more compact or descriptive name you can provide it with

```
POSTPROCESS TOTAL FLUX OF Continuity AS mdot1
```

When post-processing enthalpy, the average temperature is also output as a global variable. If you provide only one alias the enthalpy flux uses your alias and the average temperature is automatically named. If you provide two aliases the first is used for enthalpy and the second is used for temperature.

```
POSTPROCESS TOTAL FLUX OF ENTHALPY AS hFlux
POSTPROCESS TOTAL FLUX OF ENTHALPY AS hFlux Tavg
```

When post-processing species, the post-processor is run for each species that is solved for. This means that there is no post-processor run on the last species (which is determined by a fractional balance). If you provide aliases for the species post-processor you should provide one for each post-processed species. For example, in a problem with O2, CO2, and N2 (in that order) you could use:

POSTPROCESS TOTAL FLUX OF SPECIES AS mdotO2 mdotCO2

Parameter	Value	Default
<i>Flux Type</i>	string	—
<i>equation</i>	{conserved_enthalpy continuity edc_product enthalpy mixture_fraction nuclei progress_variable scalar_variance second_mixture_fraction solid_volume_fraction soot species temperature turbulent_dissipation turbulent_frequency turbulent_helmholtz_function turbulent_kinetic_energy turbulent_v2 volume_of_fluid x_momentum x_solid_momentum y_momentum y_solid_momentum z_momentum z_solid_momentum}	—

Project Nodes

Syntax

Project Nodes

Summary

Allow for nodal projection on the particular mesh part.

Description

The nodal projection of velocity is defaulted to occur on open boundary conditions only. Wall and inflow BCs are defaulted to not be projected, even on flux inflow (i.e., USE FLUXES) and turbulent wall nodes. This option flags the particular mesh part to be included in the nodes to be projected.

Progress_Variable

Syntax

Progress_Variable *ProgressVariableName* {=} *Value*

Summary

Constant value for the progress variable of selected scalars.

Parameter	Value	Default
<i>ProgressVariableName</i>	string	—
{=}	{= are is}	—
<i>Value</i>	real	—

Real Data For Subroutine

Syntax

Real Data For Subroutine *SubName* {=} *Values. . .*

Summary

List of real data values to be passed down in to the user subroutine. These values may be changed by the user subroutine.

Parameter	Value	Default
<i>SubName</i>	string	–
{=}	{= are is}	–
<i>Values</i>	real. . .	–

Radiation Boundary Temperature

Syntax

Radiation Boundary Temperature {=} *value*

Summary

Constant value for radiation_boundary_temperature variable.

Parameter	Value	Default
{=}	{= are is}	–
<i>value</i>	“string”	–

Radiation Boundary Temperature Field

Syntax

Radiation Boundary Temperature Field {=} *FieldName*

Summary

Name of the field to use for the radiation boundary temperature.

Parameter	Value	Default
{=}	{= are is}	–
<i>FieldName</i>	string	–

Radiation Environment Temperature

Syntax

Radiation Environment Temperature {=} *value*

Summary

Constant value for radiation_environment_temperature variable.

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>value</i>	“string”	–

Radiation Environment Temperature Field

Syntax

Radiation Environment Temperature Field *{=}* *FieldName*

Summary

Name of the field to use for the radiation environment temperature.

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>FieldName</i>	string	–

Subroutine For

Syntax

Subroutine For *{contact_angle | edc_product | gas_volume_fraction | mixture_fraction | pressure | progress_variable | scalar_variance | second_mixture_fraction | solid_volume_fraction | soot_mass_fraction | soot_nuclei_mass_fraction | temperature | turbulent_dissipation | turbulent_frequency | turbulent_helmholtz_function | turbulent_kinetic_energy | turbulent_v2 | volume_of_fluid | x_solid_velocity | x_velocity | y_solid_velocity | y_velocity | z_solid_velocity | z_velocity}* *{=}* *Subroutine*

Summary

Name of the subroutine to use for this variable.

Parameter	Value	Default
<i>Primitive-Variable</i>	{contact_angle edc_product gas_volume_fraction mixture_fraction pressure progress_variable scalar_variance second_mixture_fraction solid_volume_fraction soot_mass_fraction soot_nuclei_mass_fraction temperature turbulent_dissipation turbulent_frequency turbulent_helmholtz_function turbulent_kinetic_energy turbulent_v2 volume_of_fluid x_solid_velocity x_velocity y_solid_velocity y_velocity z_solid_velocity z_velocity}	–
<i>{=}</i>	{= are is}	–
<i>Subroutine</i>	string	–

Subroutine For Composite Initial_Mass_Fraction

Syntax

Subroutine For Composite Initial_Mass_Fraction *{=}* *subName*

Summary

Name of the subroutine to use for the mass fraction. ALL species mass fractions must be assigned by this subroutine.

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>subName</i>	string	–

Subroutine For Emissivity

Syntax

Subroutine For Emissivity *{=}* *subroutineName*

Summary

Name of the user subroutine to use for the emissivity variable.

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>subroutineName</i>	string	–

Subroutine For Mass_Fraction

Syntax

Subroutine For Mass_Fraction *{=}* *Subroutine*

Summary

Name of the subroutine to use for the mass fraction. ALL species mass fractions must be assigned by this subroutine.

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>Subroutine</i>	string	–

Subroutine For Mole_Fraction

Syntax

Subroutine For Mole_Fraction *{=}* *Subroutine*

Summary

Name of the subroutine to use for the mole fractions. ALL species mole fractions must be assigned by this subroutine.

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>Subroutine</i>	string	–

Subroutine For Radiation Boundary Temperature

Syntax

Subroutine For Radiation Boundary Temperature *{=}* *subroutineName*

Summary

Name of the user subroutine to use for the radiation_boundary_temperature variable.

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>subroutineName</i>	string	–

Subroutine For Radiation Environment Temperature

Syntax

Subroutine For Radiation Environment Temperature *{=}* *subroutineName*

Summary

Name of the user subroutine to use for the radiation_environment_temperature variable.

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>subroutineName</i>	string	–

Subroutine For Transmissivity

Syntax

Subroutine For Transmissivity *{=}* *subroutineName*

Summary

Name of the user subroutine to use for the transmissivity variable.

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>subroutineName</i>	string	–

Subroutine For Transparent Band Emissivity

Syntax

Subroutine For Transparent Band Emissivity *{=}* *subroutineName*

Summary

Name of the user subroutine to use for the transparent emissivity band for spectral emissivity on surfaces.

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>subroutineName</i>	string	–

Subroutine For Wall Temperature

Syntax

Subroutine For Wall Temperature *{=}* *subroutineName*

Summary

Name of the user subroutine to use for the wall_temperature variable.

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>subroutineName</i>	string	–

Transparent Band Emissivity

Syntax

Transparent Band Emissivity *{=}* *value*

Summary

Constant value to use for the emissivity in the transparent band for spectral surfaces.

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>value</i>	“string”	–

Transmissivity

Syntax

Transmissivity *{=}* *value*

Summary

Constant value for transmissivity variable.

Parameter	Value	Default
<i>{=}</i>	{ = are is }	–
<i>value</i>	“string”	–

Use Equilibrium Production Model

Syntax

Use Equilibrium Production Model

Summary

Change near-wall turbulence production.

Description

This option specifies that the near wall turb_ke production term is given by:

$$\frac{\tau_w^2}{\kappa \sqrt[4]{C_\mu} \rho y_p \sqrt{k}}$$

. Modifications are made for the KW and SST model. This model is suggested when using the standard KW and SST model when a Dirichlet condition is not placed on the wall node of turbulence kinetic energy.

Wall Temperature

Syntax

Wall Temperature *{=}* *value*

Summary

Value for wall_temperature variable. Can be a constant or a string function of space (x,y,z), time (t) or global variables.

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>value</i>	“string”	–

Law_Of_Wall_Roughness_Parameter

Syntax

Law_Of_Wall_Roughness_Parameter *{=}* *Law of the Wall Roughness parameter*

Summary

Specify dimensionless roughness factor to be used in the law-of-the-wall formulation.

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>Law of the Wall Roughness parameter</i>	real	9.8

Periodic Boundary Condition On Surface

Scope

Fuego Region

Summary

Defines a periodic boundary condition on a named surface of the mesh.

begin Periodic Boundary Condition On Surface *Surfacename*

Paired With *Value*

Search Tolerance *{=}* *Value*

end Periodic Boundary Condition On Surface *Surfacename*

Line Commands

Paired With

Syntax

Paired With *Value*

Summary

List which sideset to pair this periodic BC to. The listed sideset should not have a boundary condition applied to it.

Parameter	Value	Default
<i>Value</i>	string	–

Search Tolerance

Syntax

Search Tolerance *{=}* *Value*

Summary

Search tolerance to use when matching periodic BC nodes.

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>Value</i>	real	–

Nonconformal Boundary Condition On Surface

Scope

Fuego Region

Summary

Defines a non-conformal boundary condition on a named surface of the mesh.

begin Nonconformal Boundary Condition On Surface *Surfacename*

```
{contact_angle | edc_product | gas_volume_fraction | mixture_fraction_  
→| pressure | progress_variable | scalar_variance | second_mixture_  
→fraction | solid_volume_fraction | soot_mass_fraction | soot_nuclei_  
→mass_fraction | temperature | turbulent_dissipation | turbulent_  
→frequency | turbulent_helmholtz_function | turbulent_kinetic_energy_  
→| turbulent_v2 | volume_of_fluid | x_solid_velocity | x_velocity | y_  
→solid_velocity | y_velocity | z_solid_velocity | z_velocity} {=} Value
```

Enforce Zero Flux For Turbulence Wall Bc

Normal Treatment {=} {*keep_both* | *use_average* | *use_current*}

Omit Pressure Stabilization Term

Opposing Sides {=} *SideNames...*

Post Process Yplus

Pressure Penalty Multiplier {=} *PressurePenaltyMult*

Search Expansion Factor {=} *SearchExpansion*

Search Method {=} {*boost_rtree* | *kdtree*}

Search Tolerance {=} *SearchTol*

Use Equilibrium Production Model

Law_Of_Wall_Roughness_Parameter {=} *Law of the Wall Roughness parameter*

end Nonconformal Boundary Condition On Surface *Surfacename*

Line Commands

Primitivevariable

Syntax

Primitivevariable {*contact_angle* | *edc_product* | *gas_volume_fraction* | *mixture_fraction* | *pressure* | *progress_variable* | *scalar_variance* | *second_mixture_fraction* | *solid_volume_fraction* | *soot_mass_fraction* | *soot_nuclei_mass_fraction* | *temperature* | *turbulent_dissipation* | *turbulent_frequency* | *turbulent_helmholtz_function* | *turbulent_kinetic_energy* | *turbulent_v2* | *volume_of_fluid* | *x_solid_velocity* | *x_velocity* | *y_solid_velocity* | *y_velocity* | *z_solid_velocity* | *z_velocity*} {=} *Value*

Summary

Value for the specified variable (in consistent units). Value can be a constant or a string function of position (x,y,z), time (t), and any global variable.

The function string must be enclosed in quotes if it has spaces or commas. For example: x_velocity = “min(1, 0.1*t)”

Parameter	Value	Default
<i>Primitivevariable</i>	{ <i>contact_angle</i> <i>edc_product</i> <i>gas_volume_fraction</i> <i>mixture_fraction</i> <i>pressure</i> <i>progress_variable</i> <i>scalar_variance</i> <i>second_mixture_fraction</i> <i>solid_volume_fraction</i> <i>soot_mass_fraction</i> <i>soot_nuclei_mass_fraction</i> <i>temperature</i> <i>turbulent_dissipation</i> <i>turbulent_frequency</i> <i>turbulent_helmholtz_function</i> <i>turbulent_kinetic_energy</i> <i>turbulent_v2</i> <i>volume_of_fluid</i> <i>x_solid_velocity</i> <i>x_velocity</i> <i>y_solid_velocity</i> <i>y_velocity</i> <i>z_solid_velocity</i> <i>z_velocity</i> }	–
{=}	{= are is}	–
<i>Value</i>	“string”	–

Enforce Zero Flux For Turbulence Wall Bc

Syntax

Enforce Zero Flux For Turbulence Wall Bc

Summary

warning{This command is deprecated and has no effect}

Normal Treatment

Syntax

Normal Treatment {=} {*keep_both* | *use_average* | *use_current*}

Summary

Select which search method to use. Options are USE_CURRENT (default),

USE_AVERAGE, and KEEP_BOTH. You can specify this in just one non-conformal BC and it will apply to all. Specifying different treatments in different non-conformal BCs is currently not supported and will generate an error.

In general, USE_CURRENT will provide the cleanest behavior, with the least amount of numerical noise at the interface. For very complex, moving and deforming geometries you may sometimes gain stability with the USE_AVERAGE treatment. KEEP_BOTH is generally not recommended, and is included for diagnostic purposes only.

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>NormalMethod</i>	{keep_both use_average use_current}	–

Omit Pressure Stabilization Term

Syntax

Omit Pressure Stabilization Term

Summary

Omits the pressure stabilization term at the non-conformal interface. This option can improve stability, particularly for difficult meshes, but may adversely affect convergence rates.

Opposing Sides

Syntax

Opposing Sides *{=}* *SideNames*. . .

Summary

List of opposing sides for non-conformal BC specification.

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>SideNames</i>	string. . .	–

Post Process Yplus

Syntax

Post Process Yplus

Summary

Request post processing of yplus on this mesh part.

Description

The normalized distance to the wall (y^+) can be post-processed on any wall (laminar or turbulent). Yplus is calculated in a manner consistent with how it is

calculated in the solution procedure. Max/min y^+ values are provided as output at every wall. This option computes and stores the entire y^+ field along the wall. Since this option computes y^+ using the most recent values for all solution variables, the max/min written to the log may differ from the max/min of this field. However, these will become identical at convergence.

Note that this line command does not automatically output this field to a results file. For that, one must use the standard output line command for a nodal field using the string name, `yplus`.

Pressure Penalty Multiplier

Syntax

Pressure Penalty Multiplier `{=}` *PressurePenaltyMult*

Summary

Multiplier to increase the penalty factor on discontinuous pressures at the non-conformal boundaries. Default = 1.0.

Parameter	Value	Default
<code>{=}</code>	{ = are is }	–
<i>PressurePenaltyMult</i>	real	–

Search Expansion Factor

Syntax

Search Expansion Factor `{=}` *SearchExpansion*

Summary

Expansion factor for the search tolerance in terms of face size. Setting a value larger than 0 will expand the stencil so the linear system has to be re-initialized fewer times.

Warning: This feature is under development and may incur a significant performance penalty at this time.

Parameter	Value	Default
<code>{=}</code>	{ = are is }	–
<i>SearchExpansion</i>	real	–

Search Method

Syntax

Search Method `{=}` *{boost_rtree | kdtree}*

Summary

warning{This command is deprecated, KDTree is the only supported option}

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>SearchMethod</i>	{boost_rtree kdtree}	–

Search Tolerance

Syntax

Search Tolerance *{=}* *SearchTol*

Summary

Search tolerance for the non-conformal side matching search. This is in real distance units and should be set larger than the expected gaps between sidesets due to faceting or geometry mismatches. If omitted, Fuego calculates this automatically as a tenth of the square root of the largest face area on the non-conformal BC.

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>SearchTol</i>	real	–

Use Equilibrium Production Model

Syntax

Use Equilibrium Production Model

Summary

Change near-wall turbulence production.

Description

This option specifies that the near wall turb_ke production term is given by:

$$\frac{\tau_w^2}{\kappa^4 \sqrt{C_\mu} \rho y_p \sqrt{k}}$$

. Modifications are made for the KW and SST model. This model is suggested when using the standard KW and SST model when a Dirichlet condition is not placed on the wall node of turbulence kinetic energy.

Law_Of_Wall_Roughness_Parameter

Syntax

Law_Of_Wall_Roughness_Parameter *{=}* *Law of the Wall Roughness parameter*

Summary

Specify dimensionless roughness factor to be used in the law-of-the-wall formulation.

Parameter	Value	Default
<i>{=}</i>	{= are is}	–
<i>Law of the Wall Roughness parameter</i>	real	9.8

References

- [1] Tieszen, S. R., A. R. Lopez, C. D. Moen, T. Y. Chu, V. F. Nicolette, W. Gill, S. P. Burns, and W. C. Moffatt. "SIERRA/Fuego and SIERRA/Syrinx Verification and Validation Plan, Version 2.0". internal report, Sandia National Laboratories, 2001.
- [2] Rehm, R. G. and H. R. Baum. "The Equations of Motion for Thermally Driven Buoyant Flows". *Journal of Research of the National Bureau of Standards*, \bf 83:279, 1978.
- [3] Paolucci, S. "On the Filtering of Sound Waves from the Navier-Stokes Equations". Technical Report SAND Report 82-8257, Sandia National Laboratories, Livermore, CA, December 1982.
- [4] Majda, A. and J. Sethian. "The Derivation and Numerical Solution of the Equations for Zero Mach Number Combustion". *Combustion Science and Technology*, \bf 42:185-205, 1985.
- [5] Merkle, C. L. and Y. H. Choi. "Computation of Compressible Flows at Very Low Mach Numbers". AIAA Paper 86-0351, AIAA 24th Aerospace Sciences Meeting, Reno, NV, January 1986.
- [6] Bird, R. B., W. E. Stewart and E. N. Lightfoot. *Transport Phenomena*. John Wiley and Sons, 1960.
- [7] Burns, S. P. "Turbulence Radiation Interaction Modeling in Combustion Simulations". Technical Report, Sandia National Laboratories, Albuquerque, NM, 1999.
- [8] Modest, M. F. *Radiation Heat Transfer*. McGraw Hill Book Company, New York, 1993.
- [9] Siegel R. and J. R. Howell. *Thermal Radiation Heat Transfer, 3rd ed.* Hemisphere Publishing, Washington, D.C., 1992.
- [10] Tennekes, H. and J. L. Lumley. *A First Course in Turbulence*. MIT Press, Cambridge, 1972.
- [11] Libby, P. A. and F. A. Williams. *Turbulent Reacting Flows, Fundamental Aspects, Topics in Applied Physics, V. 44*. Springer-Verlag., 1980.
- [12] Kuo, K. K. *Principles of Combustion*. John Wiley and Sons, 1986.
- [13] Wilcox, D. C. *Turbulence Modeling for CFD*. DCW Industries, 2nd edition, 1998.
- [14] Moin, P., K. Squires, W. Cabot, and S. Lee. A dynamic subgrid-scale model for compressible turbulence and scalar transport. *Phys. Fluids A*, 3(11):2746-2757, 1991.
- [15] Erlebacher, G., M. Y. Hussaini, C. G. Speziale, and T. A. Zang. Toward the large-eddy simulation of compressible turbulent flows. *J. Fluid Mech.*, 238:155-185, 1992.
- [16] Gran, I. M. C. Melaaen, and B. F. Magnussen. "Numerical Simulation of Local Extinction Effects in Turbulent Combustor Flows of Methane and Air". In *25th Symposium on Combustion*, 1283-1291. The Combustion Institute, 1994.
- [17] Tieszen, S. R., S. P. Domino, A. R. Black. Validation of a simple turbulence model suitable for closure of temporally-filtered Navier-Stokes equations using a helium plume. Technical

Report SAND Report 2005–3210, Sandia National Laboratories, Albuquerque, NM, June 2005.

- [18] Launder, G. E. and B. I. Sharma. Application of the energy dissipation model of turbulence to the calculation of flow near a spinning disc. *Letters in Heat and Mass Transfer*, \bf 1(2):131–138, 1974.
- [19] A. Mathur and S. He. "Performance and implementation of the Launder–Sharma low-Reynolds number turbulence model". *Computers and Fluids*, \bf 79:134–139, 2013.
- [20] Papageorgakis, G. C. and D. N. Assanis. Comparison of linear and nonlinear RNG-based k-epsilon models for incompressible turbulent flows. *Numerical Heat Transfer, Part B*, 35(1):1–22, 1999.
- [21] Durbin, P. A. "Near-Wall Turbulence Closure Modeling Without Damping Functions". *Theoretical and Computational Fluid Dynamics*, \bf 3:1–13, 1991.
- [22] Wilcox, D. C. Formulation of the k-omega turbulence model revisited. 45th AIAA Aerospace Sciences Meeting and Exhibit, 2007.
- [23] Menter, F. R., Kuntz, M. and R. Langtry. "Ten years of industrial experience with the SST turbulence model". *Turb, Heat and Mass Trans*, 2003.
- [24] Smagorinsky, J. General circulation experiments with the primitive equations. i. the basic experiment. *Monthly Weather Review*, 91:99–164, 1963.
- [25] Rogallo, R. S. and P. Moin. Numerical simulation of turbulent flows. *Annual Review of Fluid Mechanics*, 16:99–137, 1984.
- [26] Germano, M., U. Piomelli, P. Moin, and W. H. Cabot. A dynamic subgrid-scale eddy viscosity model. *Physics of Fluids A*, 3(7):1760–1765, July 1991.
- [27] Germano, M. Turbulence: the filtering approach. *Journal of Fluid Mechanics*, 238:325–336, 1992.
- [28] Ghosal, S., T. S. Lund, P. Moin, and K. Akselvoll. A dynamic localization model for large-eddy simulation of turbulent flow. *Journal of Fluid Mechanics*, 286:229–255, 1995.
- [29] Lilly, D. K. A proposed modification of the germano subgrid-scale closure model. *Physics of Fluids A*, 4(3):633–635, March 1992.
- [30] Masahide Inagaki. A new wall-damping function for large eddy simulation employing kolmogorov velocity scale. *International Journal of Heat and Fluid Flow*, 32(1):26–40, 2011. URL: <https://www.sciencedirect.com/science/article/pii/S0142727X10001268>, doi:<https://doi.org/10.1016/j.ijheatfluidflow.2010.07.001>.
- [31] Kim, W. W. and S. Menon. "Application of the localized dynamic subgrid-scale model to turbulent wall-bounded flows". 35th AIAA Aerospace Sciences Meeting and Exhibit, 1997.
- [32] Nicolette, V. F. and S. R. Tieszen. "Effect of Turbulent Kinetic Energy Source Terms on Pool Fire Simulations with the \$k\$-epsilon \$Model". Technical Report, Internal Memorandum to Distribution, Official Use Only, Sandia National Laboratories, 2000.

- [33] Rodi W. *Turbulence Models and their Applications in Hydrolics - A State of the Art Review*. Publication of the International Association for Hydrolic Research, Delf, Netherlands, 1984.
- [34] de Ris, J. L. Mechanism of buoyant turbulent diffusion flames. *Procedia Engineering*, 62:13–27, 2013.
- [35] Jones, W. P. and B. E. Launder. “The Prediction of Laminarization with a Two-Equation Model of Turbulence”. *International Journal of Heat and Mass Transfer*, \bf 15:301–314, 1972.
- [36] White, F. M. *Viscous Fluid Flow*. McGraw-Hill, Inc., 2nd ed., 1991.
- [37] J. Larsson, S. Kawai, J. Bodart, and I. Bermejo-Moreno. Large eddy simulation with modeled wall-stress: recent progress and future directions. *Mechanical Engineering Reviews*, 3(1):15–00418–15–00418, 2016.
- [38] Sondak, D. L. and R. H. Pletcher. “Application of wall functions to generalized nonorthogonal curvilinear coordinate systems”. *AIAA Journal*, \bf 33(1):33–41, January 1995.
- [39] Elkaim, D., M. Reggio, and R. Camarero. “Control Volume Finite-Element Solution of a Confined Turbulent Diffusion Flame”. *Numerical Heat Transfer, Part A*, \bf 23(3):259–279, 1993.
- [40] Launder, B. E. and D. B. Spalding. “The Numerical Computation of Turbulent Flows”. *Computer Methods in Applied Mechanics and Engineering*, \bf 3:269–289, 1974.
- [41] Versteeg, H. K. and W. Malalasekera. *An Introduction to Computational Fluid Dynamics*. Longman Group LTD, 1995.
- [42] Jayatilke, C. L. V. “The Influence of Prandtl Number and Surface Roughness on the Resistance of Laminar Sub-Layer to Momentum and Heat Transfer”. *Progress in Heat and Mass Transfer*, 1969.
- [43] V.I. Kornilov. Current state and prospects of researches on the control of turbulent boundary layer by air blowing. *Progress in Aerospace Sciences*, 76:1–23, 2015. URL: <https://www.sciencedirect.com/science/article/pii/S0376042115000329>, doi:<https://doi.org/10.1016/j.paerosci.2015.05.001>.
- [44] Magnussen, B. F., G. H. Hjertager, J. G. Olsen, and D. Bhaduri. “Effect of Turbulent Structure and Local Concentrations on Soot Formation and Combustion in C₂H₂ Diffusion Flames”. In *Seventeenth Symposium (International) on Combustion*, 1383–1393. The Combustion Institute, Pittsburgh, 1979.
- [45] Magnussen, B. F. “On the Structure of Turbulence and a Generalised Eddy Dissipation Concept for Chemical Reactions in Turbulent Flow”. 9th AIAA Sc. Meeting, St. Louis, 1981.
- [46] Byggstøl, S. and B. F. Magnussen. “A Model for Extinction in Turbulent Flows”. In et al.

- Bradbury, editor, *4th Symposium on Turbulent Shear Flow*, 381–395. Springer-Verlag, Berlin, 1983.
- [47] Magnussen, B. F. “Heat Transfer in Gas Turbine Combustors – A Discussion of Combustion, Heat and Mass Transfer in Gas Turbine Combustors”. In *Conference Proceedings no. 390, Advisory Group for Aerospace Research and Development (AGARD)*. 1985.
 - [48] Lilleheie, N. I., I. Ertesvåg, T. Bjorge, S. Byggstøl, and B. F. Magnussen. “Modeling and Chemical Reactions, Review of Turbulence and Combustion Models”. Technical Report, The Foundation for Scientific and Industrial Research, Norwegian Institute of Technology, SINTEF Report STF15 A89024, July 1989.
 - [49] Gran, I. and B. F. Magnussen. “A Numerical Study of a Bluff-body Stabilized Diffusion Flame. Part 2: Influence of Combustion Modeling and Finite Rate Chemistry”. *Combustion Science and Technology*, 119:191–217, 1996.
 - [50] Tieszen, S. R., V. F. Nicolette, L. A. Gritz, J. K. Holen, D. Murray, and J. L. Moya. “Vortical Structures In Pool Fires: Observation, Speculation, and Simulation”. Technical Report, SAND96-2607, Sandia National Laboratories, Albuquerque, NM, November 1996.
 - [51] Strehlow, R. A. *Combustion Fundamentals*. McGraw-Hill, New York, 1984.
 - [52] Ertesvåg, I. S. and B. F. Magnussen. “The Eddy-dissipation turbulence energy cascade model”. Technical Report, Department of Applied Mechanics, Thermodynamics and Fluid Dynamics, The Norwegian University of Science and Technology, Trondheim, Norway (In Preparation), 1997.
 - [53] Holen, J., B. Lakså, B. F. Magnussen, and B. E. Vembe. “KAMELEON-II-Fire Theory Manual, A Description of the Mathematical Models, Numerical Methods, and Solution Procedures”. Technical Report, The Foundation for Scientific and Industrial Research, Norwegian Institute of Technology, Trondheim, Norway, SINTEF Report STF15 F94070, November 1994.
 - [54] N. Peters. Laminar diffusion flamelet models in non-premixed turbulent combustion. *Progress in Energy and Combustion Science*, 10:319–339, 1984.
 - [55] IM Aksit and JB Moss. A hybrid scalar model for sooting turbulent flames. *Combustion and flame*, 145(1-2):231–244, 2006.
 - [56] Danny Messig, Franziska Hunger, Jens Keller, and Christian Hasse. Evaluation of radiation modeling approaches for non-premixed flamelets considering a laminar methane air flame. *Combustion and Flame*, 160(2):251–264, 2013.
 - [57] T. Poinso and D. Veynante. *Theoretical and Numerical Combustion*. R. T. Edwards, Inc., Philadelphia, PA, 2005.
 - [58] C. D. Pierce and P. Moin. A dynamic model for subgrid-scale variance and dissipation rate of a conserved scalar. *Physics of Fluids*, 3041–3044.

- [59] R. W. Bilger. The structure of turbulent nonpremixed flames. In *Proceedings of the 22nd Symposium (International) on Combustion*, pages 475–488. The Combustion Institute, Pittsburg, PA, 1988.
- [60] G. Erlebacher, M. Y. Hussaini, C. G. Speziale, and T. A. Zang. Toward the large eddy simulation of compressible turbulent flows. ICASE Report 87-20, NASA Langley Research Center, Hampton, VA, 1987. Also available as \em NASA CR 178273.
- [61] S. Ghosal, T.S. Lund, P. Moin, and K. Akselvoll. A dynamic localization model for large-eddy simulation of turbulent flow. *Journal of Fluid Mechanics*, 286:229–255, 1995.
- [62] Parente, A. and Malik, M. R. and Contino, F. and Cuoci, A. and Dally, B. B. “Extension of the Eddy Dissipation Concept for turbulence/chemistry interactions to MILD combustion”. *Fuel*, \bf 163:98–111, 2016.
- [63] Magnussen, B. F. and G. H. Hjertager. “On Mathematical Modeling of Turbulent Combustion with Special Emphasis on Soot Formation and Combustion”. In *Sixteenth Symposium (International) on Combustion*, 719–729. The Combustion Institute, Pittsburgh, 1977.
- [64] Tesner, P. A., T. D. Snegiriova, and V. G. Knorre. “Kinetics of dispersed carbon formation”. *Combustion and Flame*, \bf 17:253–260, 1971.
- [65] Haynes, B. S. and H. G. Wagner. “Soot Formation”. *Progress in Energy and Combustion Science*, \bf 7:229–273, 1981.
- [66] Tesner, P. A., E. I. Tsygankova, L. P. Guilazetdinov, V. P. Zuyev, and G. V. Loshakova. “The formation of soot from aromatic hydrocarbons in diffusion flames of hydrocarbon-hydrogen mixtures”. *Combustion and Flame*, \bf 17:279–285, 1971.
- [67] Tezduyar, T. E. “Stabilized Finite Element Formulations for Incompressible Flow Computations”. In editor, *Advances in Applied Mechanics*, volume \bf 28, pages 1–44. Academic Press, Inc., 1992.
- [68] Felske, J. D. and C. L. Tien. “Calculation of the Emissivity of Luminous Flames”. *Combustion Science and Technology*, \bf 7:23–31, 1973.
- [69] Felske, J. D. and T. T. Charalampopoulos. “Gray Gas Weighting Coefficients for Arbitrary Gas-Soot Mixtures”. *International Journal of Heat and Mass Transfer*, \bf 25(12):1849–1855, 1982.
- [70] Abramowitz, M. and I. A. Stegun. *Handbook of Mathematical Functions*. National Bureau of Standards, 1964.
- [71] Leckner, B. “Spectral and Total Emissivity of Water Vapor and Carbon Dioxide”. *Combustion and Flame*, \bf 19:33–48, 1972.
- [72] Martinez, M. J. and P. I. Hopkins. “Modeling Subsurface Multiphase Transport of JP8 During a Fuel Spill Fire”. Technical Report, SAND2000-2464, Sandia National Laboratories, Albuquerque, NM, October 2000.

- [73] Saito, K. G. Tashtoush, C. Cremers, and L. A. Gritzo. "Flame Spread over JP8 Aircraft Fuel". to appear in *Combustion Science and Technology*, 1997.
- [74] Gritzo, L. A., E. A. Boucheron, and D. Murray. "Fuel Temperature Distribution and Burning Rate in Large Pool Fires". NIST Annual Conference on Fire Research, Gaithersburg, MD, October 1996.
- [75] Gritzo, L. A., V. F. Nicolette, S. R. Tieszen, and J. L. Moya. "Heat Transfer to the Fuel Surface in Large Pool Fires". In S. H. Chan, editor, *Transport Phenomenon in Combustion*, 701–712. Taylor and Francis, 1996.
- [76] Blinov, V. I. and G. N. Khudiakov. "Diffusion Burning of Liquids". Technical Report, English Translation: U.S. Army Engineering Research and Development Labs, Fort Belvoir, VA, Report AERDL-T-1490-A, 1961.
- [77] Mansfield, J. M. and L. J. Linley. "Measurement and Statistical Analysis of Flame Temperatures from Large Fuel Spill Fires". Technical Report, NWC TP 7061, Naval Air Warfare Center, China Lake, CA, 93555, 1991.
- [78] Cline, D. D. and L. N. Koenig. "The Transient Growth of an Unconfined Pool Fire". *Fire Technology*, \bf 19(3):149–162, 1983.
- [79] Magnoli, D. E. "A Model for Fuel Fire Duration and Application to the B-1B Bomber". Technical Report, Lawrence Livermore National Laboratory, Livermore, CA, UCRL-ID-112576, 1992.
- [80] S. P. Domino. Towards verification of sliding mesh algorithms for complex applications using mms. *Proceedings of the 2010 Summer Program, Center for Turbulence Research*, 2010.
- [81] M. Discacciati, A. Quarteroni, and A. Valli. Robin-Robin domain decomposition methods for the Stokes-Darcy coupling. *SIAM Journal of Numerical Analysis*, 45(3):1246–1268, 2007.
- [82] G. S. Beavers and D. D. Joseph. Boundary conditions at a naturally permeable wall. *J. Fluid Mech.*, 30:197–207, 1967.
- [83] P. G. Saffman. On the boundary condition at the surface of a porous medium. *Studies in Applied Mathematics*, 50(2):93–101, 1971.
- [84] R. H. Davis and H. A. Stone. Flow through beds of porous particles. *Chemical Engineering Science*, 48(23):3993–4005, 1993.
- [85] Mencinger, J. and Zun, I. A plic-vof method suited for adaptive moving grids. *Journal of Computational Physics*, \bf 230:644–663, 2011.
- [86] Martinez, J. and Chesneau, X. and Zeghmami, B. A new curvature technique calculation for surface tension contribution in plic-vof method. *Computational Mechanics*, \bf 37:182–193, 2006.
- [87] Sun, M. Accuracy improvement of plic-vof volume-tracking method using the equation of surface. *Advances in Pure Mathematics*, \bf 3:219–225, 2013.

- [88] Huang, M. and Wu, L. and Chen, B. A piecewise linear interface-capturing volume-of-fluid method based on unstructured grids. *Numerical Heat Transfer, Part B: Fundamentals*, \bf 61:412–437, 2012.
- [89] Lee, H. and Rhee, S. A dynamic interface compression method for vof simulations of high-speed planing watercraft. *Journal of Mechanical Science and Technology*, \bf 29:1849–1857, 2015.
- [90] Raeini, A. and Blunt, M. and Bijeljic, B. Modelling two-phase flow in porous media at the pore scale using the volume-of-fluid method. *Journal of Computational Physics*, \bf 231:5653–5668, 2012.
- [91] Jofre, L. and Lehmkuhl, O. and Castro, J. and Olivia, A. A plic-vof implementation on parallel 3d unstructured meshes. In Lisbon Portugal, editor, *European Conference on Computational Fluid Dynamics*. 2010.
- [92] Hardt, S. and Wondra, F. Evaporation model for interfacial flows based on a continuum-field representation of the source terms. *Journal of Computational Physics*, \bf 227:5781–5895, 2008.
- [93] Francois, M. M. and Cummins, S. J. and Dendy, E. D. and Kothe, D. B. and Sicilian, J. M. and Williams, M. W. A balanced-force algorithm for continuous and sharp interfacial surface tension models within a volume tracking framework. *Journal of Computational Physics*, \bf 213:141–173, 2006.
- [94] Lin, S. and Yan, J. and Kats, D. and Wagner, G. A volume-conserving balanced-force level set method on unstructured meshes using a control volume finite element formulation. *Journal of Computational Physics*, \bf 380:119–142, 2019.
- [95] Law, C. K. “Recent Advances in Droplet Vaporization and Combustion”. *Prog. Energy Combust Sci.*, \bf 8(3):171–201, 1982.
- [96] Sirignano, W. A. “Fuel Droplet Vaporization and Spray Combustion Theory”. *Prog. Energy Combust Sci.*, \bf 9(4):291–322, 1983.
- [97] Faeth, G. M. “Evaporation and combustion of sprays”. *Prog. Energy Combust Sci.*, \bf 9:1–76, 1983.
- [98] Amsden, A. A., et al. “KIVA: A Computer Program for Two-and Three-Dimensional Fluid Flows with Chemical Reactions and Fuel Sprays”. 1985.
- [99] Faeth, G.M. “Mixing, Transport and Combustion in Sprays”. *Prog. Energy Combust Sci.*, \bf 13:293–345, 1987.
- [100] Amsden, A.A., P.O'Rourke, and T.D. Butler. “KIVA-II: A Computer Program for Chemically Reactive Flows with Sprays”. 1989.
- [101] Crowe, C., M. Sommerfeld, and Y. Tsuji. “*Multiphase flows with droplets and particles*”. CRC Press, New York, NY, 1998.
- [102] Sommerfeld, M. and H.H. Qiu. “Experimental studies of spray evaporation in turbulent flow”. *International Journal of Heat and Fluid Flow*, \bf 19(1):10–22, 1998.

- [103] Shaw, R.A. "Particle-turbulence interactions in atmospheric clouds". *Annual Review of Fluid Mechanics*, \bf 35:183–227, 2003.
- [104] Holen, J., M. Brostrom, and B.F. Magnussen. "Finite Difference Calculation of Pool Fire". *Proc. Combust. Instit.*, \bf 23:1677–1683, 1990.
- [105] Yoon, S.S., et al. "Numerical modeling and experimental measurements of a high speed solid-cone water spray for use in fire suppression applications". *International Journal of Multiphase Flow*, \bf 30(11):1369–88, 2004.
- [106] DesJardin, P.E. and L.A. Gritzo. "A Dilute Spray Model for Fire Simulations: Formulation, Usage and Benchmark Problems". "*Sandia Technical Report*", 2002.
- [107] Maxey, M.R. and J.J. Riley. "Equation of motion for a small rigid sphere in a nonuniform flow". *Phys. Fluids*, \bf 26(4):883–889, 1983.
- [108] Williams, F.A. "Spray Combustion and Atomization". *Physics of Fluids*, \bf 1(6):541–545, 1958.
- [109] O'Rourke, P.J. and A.A. Amsden. "The TAB Method for Numerical Calculation of Spray Droplet Breakup". *SAE Technical Paper 872089*, 1987.
- [110] Patankar, S.V. "*Numerical Heat Transfer and Fluid Flow*". Taylor and Francis, 1980.
- [111] Gosman, A.D. and E. Ioannides. "Aspects of computer simulation of liquid-fueled combustors". *AIAA Paper 81-0323*, 1981.
- [112] Shuen, J.S., L.D. Chen, and G.M. Faeth. "Evaluation of a stochastic model of particle dispersion in a turbulent round jet". *AIChE Journal*, \bf 29(1):167–70, 1983.
- [113] Zhou, Q. and S.C. Yao. "Group modeling of impacting spray dynamics". *International Journal of Heat and Mass Transfer*, \bf 35(1):121–9, 1992.
- [114] Dukowicz, J.K. "A Particle-Fluid Numerical Model for Liquid Sprays". *Journal of Computational Physics*, \bf 35(2):229–253, 1980.
- [115] Jones, W.P. and B.E. Launder. "Prediction of Laminarization with a 2-Equation Model of Turbulence". *Int. J. Heat Mass Transfer*, \bf 15:301, 1972.
- [116] Yuen, M.C. and L.W. Chen. "On Drag of Evaporating Liquid Droplets". *Combust. Sci. Technol.*, \bf 14(4-5-6):147–154, 1976.
- [117] Taylor, G.I. "Diffusion by continuous movement". *Proc. London Math. Soc.*, \bf 20:196–211, 1921.
- [118] Snyder, W.H. and J.L. Lumley. "Some measurements of particle velocity autocorrelation functions in a turbulent flow". *Journal of Fluid Mechanics*, \bf 48:41, 1971.
- [119] Abramzon, B. and W.A. Sirignano. "Droplet vaporization model for spray combustion calculations". *International Journal of Heat and Mass Transfer*, \bf 32(9):1605–18, 1989.
- [120] Modest, M.F. "*Radiative Heat Transfer*". Academic Press, second ed. edition, 2003.
- [121] Lefebvre, A.H. "*Atomization And Sprays*". Hemisphere, 1988.

- [122] Johns, L.E. and R.B. Beckmann. "Mechanism of dispersed-phase mass transfer in viscous single-drop extraction systems". *Amer. Instit. Chem. Eng. J.*, \bf 12(1):10–16, 1966.
- [123] Sirignano, W.A. "*Fluid Dynamics and Transport of Droplets and Sprays*". Cambridge University Press, New York, NY, 1999.
- [124] van de Hulst, H.C. "*Light Scattering by Small Particles*". Dover, New York, 1981.
- [125] Yoon, S.S., et al. "Numerical Modeling and Experimental Measurements of Water Spray Impact and Transport over a Cylinder". *Int. J. Multiphase Flow*, 2005.
- [126] Erikson, W.W. and W. Gill. "Analytic Model for Propellant Fire Heat Transfer with Deposition". *JANNAF 40th CS, 28th APS, 22nd PSHS and 4th MSS Joint Meeting*, 2005.
- [127] O'Rourke, P.J. "Statistical properties and numerical implementation of a model for droplet dispersion in a turbulent gas". *Journal of Computational Physics*, \bf 83(2):345–60, 1989.
- [128] Zienkiewicz, O. C. and R. L. Taylor. *The Finite Element Method*, \rm 4th ed., Vol. 1. McGraw-Hill, 1989.
- [129] Zienkiewicz, O. C. and R. L. Taylor. *The Finite Element Method*, \rm 4th ed., Vol. 2. McGraw-Hill, 1991.
- [130] Gresho, P. M, S. T. Chan, R. L. Lee, and C. D. Upson. "A Modified Finite Element Method for Solving the Time-Dependent, Incompressible Navier-Stokes Equations. Part I: Theory". *International Journal for Numerical Methods in Fluids*, \bf 4:557–598, 1984.
- [131] Patankar, S. V. "Recent Developments in Computational Heat Transfer". *Journal of Heat Transfer*, \bf 110(4B):1037–1045, 1988.
- [132] Shyy, W. "Elements of Pressure-Based Computational Algorithms for Complex Fluid Flow and Heat Transfer". In Hartnett, J. P. and T. F. Irvine, editor, *Advances in Heat Transfer*, volume \bf 24, pages 191–275. Academic Press, Inc., 1994.
- [133] Ferziger, J. H. and M. Perić. *Computational Methods for Fluid Dynamics*. Springer-Verlag, 1996.
- [134] Baliga, B. R. and S. V. Patankar. "A New Finite-Element Formulation for Convection-Diffusion Problems". *Numerical Heat Transfer*, \bf 3(4):393–409, 1980.
- [135] Baliga, B. R. "*A Control Volume Based Finite Element Method for Convective Heat and Mass Transfer*". PhD thesis, University of Minnesota, Minneapolis, MN, 1978.
- [136] Baliga, B. R. and S. V. Patankar. "A Control Volume Finite-Element Method for Two-Dimensional Fluid Flow and Heat Transfer". *Numerical Heat Transfer*, \bf 6(3):245–261, 1983.
- [137] Schneider, G. E. and M. Zedan. A Control Volume Based Finite Element Formulation of the Heat Conduction Equation. In H. E. Collicott and P. E. Bauer, editor, *Spacecraft Thermal Control, Design and Operation*, volume \bf 86, pages 305–327. Progress in Astronautics and Aeronautics, 1983.

- [138] Schneider, G. E. and M. J. Raw. "A Skewed, Positive Influence Coefficient Upwinding Procedure for Control-Volume-Based Finite-Element Convection-Diffusion Computation". *Numerical Heat Transfer*, \bf 9(1):1–26, 1986.
- [139] Swaminathan, C. R. and V. R. Voller. "Streamline Upwind Scheme for Control-Volume Finite Elements, Part I. Formulations". *Numerical Heat Transfer, Part B*, \bf 22(1):95–107, 1992.
- [140] Patankar, S. V. *Numerical Heat Transfer and Fluid Flow*. Hemisphere, 1980.
- [141] none. Tascflow theory documentation. Technical Report, Advanced Scientific Computing, Ltd., Waterloo, Ontario, 1995.
- [142] Winslow, A. M. "Numerical Solution of the Quasilinear Poisson Equation in a Nonuniform Triangle Mesh". *Journal of Computational Physics*, \bf 1(2):149–172, 1966.
- [143] Ramadhyani, S. and S. V. Patankar. "Solution of the Poisson Equation: Comparison of the Galerkin and Control-Volume Methods". *International Journal for Numerical Methods in Engineering*, \bf 15:1395–1418, 1980.
- [144] Baliga, B. R., T. T. Pham and S. V. Patankar. "Solution of Some Two-Dimensional Incompressible Fluid Flow and Heat Transfer Problems Using a Control Volume Finite-Element Method". *Numerical Heat Transfer*, \bf 6(3):263–282, 1983.
- [145] Prakash, C. and S. V. Patankar. "A Control Volume-Based Finite-Element Method for Solving the Navier-Stokes Equations Using Equal-Order Velocity-Pressure Interpolation". *Numerical Heat Transfer*, \bf 8(3):259–280, 1985.
- [146] Ramadhyani, S. and S. V. Patankar. "Solution of the Convection-Diffusion Equation by a Finite-Element Method Using Quadrilateral Elements". *Numerical Heat Transfer*, \bf 8(5):595–612, 1985.
- [147] Raithby, G. D. "Skew Upstream Differencing Schemes for Problems Involving Fluid Flow". *Computer Methods in Applied Mechanics and Engineering*, \bf 9:153–164, 1976.
- [148] Raw, M. J. "A New Control-Volume-Based Finite Element Procedure for the Numerical Solution of the Fluid Flow and Scalar Transport Equations". PhD thesis, University of Waterloo, Ontario, Canada, 1985.
- [149] LeDain-Muir, B. and B. R. Baliga. "Solution of Three-Dimensional Convection-Diffusion Problems Using Tetrahedral Elements and Flow-Oriented Upwind Interpolation Functions". *Numerical Heat Transfer*, \bf 9(2):143–162, 1986.
- [150] Prakash, C. "An Improved Control Volume Finite-Element Method for Heat and Mass Transfer, and for Fluid Flow Using Equal-Order Velocity-Pressure Interpolation". *Numerical Heat Transfer*, \bf 9(3):253–276, 1986.
- [151] Schneider, G. E. and M. J. Raw. "Control Volume Finite-Element Method for Heat Transfer and Fluid Flow Using Colocated Variables–1. Computational Procedure". *Numerical Heat Transfer*, \bf 11(4):363–390, 1987.

- [152] Schneider, G. E. and M. J. Raw. "Control Volume Finite-Element Method for Heat Transfer and Fluid Flow Using Colocated Variables–2. Application and Validation". *Numerical Heat Transfer*, \bf 11(4):391–400, 1987.
- [153] Schneider, G. E. "Preliminary Results of a Novel Fluid Flow Prediction Procedure Applied to Axi-Symmetric Problems". AIAA Paper 87-1639, 22nd Thermophysics Conference, Honolulu, HA, June 1987.
- [154] Prakash, C. "Examination of the Upwind (Donor-Cell) Formulation in Control Volume Finite Element Methods for Fluid Flow and Heat Transfer". *Numerical Heat Transfer*, \bf 11(4):401–416, 1987.
- [155] Hookey, N. A., B. R. Baliga and C. Prakash. "Evaluation and Enhancements of Some Control Volume Finite-Element Methods–1. Convection-Diffusion Problems". *Numerical Heat Transfer*, \bf 14(3):255–272, 1988.
- [156] Hookey, N. A. and B. R. Baliga. "Evaluation and Enhancements of Some Control Volume Finite-Element Methods–1. Incompressible Fluid Flow Problems". *Numerical Heat Transfer*, \bf 14(3):273–293, 1988.
- [157] van Doormaal, J. P. and G. D. Raithby. "Enhancements of the SIMPLE Method for Predicting Incompressible Fluid Flows". *Numerical Heat Transfer*, \bf 7:147–163, 1984.
- [158] Schneider, G. E. Elliptic Systems: Finite-Element Method I. In Minkowycz, W. J., E. M. Sparrow, G. E. Schneider and R. H. Pletcher, editor, *Handbook of Numerical Heat Transfer*, chapter 10, pages 379–420. John Wiley & Sons, Inc., 1988.
- [159] Baliga, B. R. and S. V. Patankar. Elliptic Systems: Finite-Element Method II. In Minkowycz, W. J., E. M. Sparrow, G. E. Schneider and R. H. Pletcher, editor, *Handbook of Numerical Heat Transfer*, chapter 11, pages 421–462. John Wiley & Sons, Inc., 1988.
- [160] Swaminathan, C. R. and V. R. Voller. "Streamline Upwind Scheme for Control-Volume Finite Elements, Part II. Implementaion and Comparison with the SUPG Finite-Element Scheme". *Numerical Heat Transfer, Part B*, \bf 22(1):109–124, 1992.
- [161] Brooks, A. N. and T. J. R. Hughes. "Streamline Upwind/Petrov-Galerkin Formulations for Convection Dominated Flows with Particular Emphasis on the Incompressible Navier-Stokes Equations". *Computer Methods in Applied Mechanics and Engineering*, \bf 32:199–259, 1982.
- [162] Baliga, B. R. and H. J. Saabas. "Control-Volume Finite Element Methods for Incompressible Fluid Flow". Invited Keynote Lecture on Fluid Mechanics, III Portuguese Conference on Computational Mechanics, Coimbra, Portugal, September 1992.
- [163] Naterer, G. F. and G. E. Schneider. "Physical Influences of Integration Point Equations On a Control-Volume-Based Finite Element Method for Compressible Flows". AIAA Paper 92-0365, 30th Aerospace Sciences Meeting, Reno, NV, January 1992.
- [164] Swaminathan, C. R., V. R. Voller and S. V. Patankar. "A Streamline Upwind Control Volume Finite Element Method for Modeling Fluid Flow and Heat Transfer Problems". *Finite Elements in Analysis and Design*, \bf 13(2-3):169–184, 1993.

- [165] Saabas, H. J. and B. R. Baliga. "Co-Located Equal-Order Control-Volume Finite-Element Method for Multidimensional, Incompressible, Fluid Flow—Part I: Formulation". *Numerical Heat Transfer, Part B*, \bf 26(4):381–407, 1994.
- [166] Saabas, H. J. and B. R. Baliga. "Co-Located Equal-Order Control-Volume Finite-Element Method for Multidimensional, Incompressible, Fluid Flow—Part II: Verification". *Numerical Heat Transfer, Part B*, \bf 26(4):409–424, 1994.
- [167] Masson, C., H. J. Saabas and B. R. Baliga. "Co-Located Equal-Order Control-Volume Finite Element Method for Two-Dimensional Axisymmetric Incompressible Fluid Flow". *Internation Journal for Numerical Methods in Fluids*, \bf 18(1):1–26, 1994.
- [168] Masson, C. and B. R. Baliga. "A Control Volume Finite-Element Method for Dilute Gas-Solid Particle Flows". *Computers and Fluids*, \bf 23(8):1073–1096, 1994.
- [169] Karimian, S. M. H. and G. E. Schneider. "Numerical Solution of Two-Dimensional Incompressible Navier-Stokes Equations: Treatment of Velocity-Pressure Coupling". AIAA Paper 94-2359, 25th AIAA Fluid Dynamics Conference, Colorado Springs, CO, June 1994.
- [170] Karimian, S. M. H. and G. E. Schneider. "Pressure-Based Computational Method for Compressible and Incompressible Flows". *Journal of Thermophysics and Heat Transfer*, \bf 8(2):267–274, 1994.
- [171] Deng, G. B., J. Piquet, P. Queutey and M. Visonneau. "Incompressible Flow Calculations With a Consistent Physical Interpolation Finite Volume Approach". *Computers and Fluids*, \bf 23(8):1029–1047, 1994.
- [172] Rhie, C. M. and W. L. Chow. "Numerical Study of the Turbulent Flow Past an Airfoil with Trailing Edge Separation". *AIAA Journal*, \bf 21(11):1525–1532, November 1983.
- [173] Costa, V. A. F., L. A. Oliveira and A. R. Figueiredo. "A Control-Volume Based Finite Element Method for Three-Dimensional Incompressible Turbulent Fluid Flow, Heat Transfer, and Related Phenomena". *International Journal for Numerical Methods in Fluids*, \bf 21(7):591–613, 1995.
- [174] Karimian, S. M. H. and G. E. Schneider. "Application of a Control-Volume-Based Finite-Element Formulation to the Shock Tube Problem (TN)". *AIAA Journal*, \bf 33(1):165–, 1995.
- [175] Karimian, S. M. H. and G. E. Schneider. "Pressure-Based Control-Volume Finite Element Method for Flow at All Speeds". *AIAA Journal*, \bf 33(9):1611–1618, September 1995.
- [176] Padra, C. and A. Larreteguy. "A-Posteriori Error Estimator for the Control-Volume Finite-Element Method as Applied to Convection-Diffusion Problem". *Numerical Heat Transfer, Part B*, \bf 27(1):63–80, 1995.
- [177] Larreteguy, A. E. "An Equal-Order Control-Volume Finite-Element Method for Fluid Flow in Arbitrary Triangulations". *Numerical Heat Transfer, Part B*, \bf 28:401–413, 1995.
- [178] Harms, T. M., T. W. von Backström and J. P. du Plessis. "Simplified Control-Volume Finite-Element Method". *Numerical Heat Transfer, Part B*, \bf 30(2):179–194, 1996.

- [179] Comini, G., S. Del Giudice, and C. Nonino. "Energy Balances in CVFEM and GFEM Formulations of Convection-Type Problems". *International Journal for Numerical Methods in Engineering*, \bf 39(13):2249–2263, 1996.
- [180] Neises, J. and I. Steinbach. "Finite Element Integration for the Control Volume Method". *Communications in Numerical Methods in Engineering*, \bf 12(9):543–556, 1996.
- [181] Völker, S., T. Burton and S. P. Vanka. "Finite-Volume Multigrid Calculation of Natural-Convection Flows on Unstructured Grids". *Numerical Heat Transfer, Part B*, \bf 30(1):1–22, 1996.
- [182] Botta, N. and D. Hempel. "Finite-Volume Projection Method for Incompressible Flows on Triangular Grids". *Hamburger Beiträge zur Angewandten Mathematik, Reihe A*, Preprint 110, September 1996.
- [183] Darbandi, M. and G. E. Schneider. "Momentum Variable Procedure for Solving Compressible and Incompressible Flows". *AIAA Journal*, \bf 35(12):1801–1805, 1997.
- [184] B. R. Baliga. "Control-Volume Finite Element Methods for Fluid Flow and Heat Transfer". In Minkowycz, W. J. and E. M. Sparrow, editor, *Advances in Numerical Heat Transfer, Volume 1*, pages 97–135. Taylor and Francis, 1997.
- [185] O'Rourke, P. J., and M. S. Sahota. "A Variable Explicit/Implicit Numerical Method for Calculating Advection on Unstructured Meshes". *Journal of Computational Physics*, \bf 143:312–345, 1998.
- [186] Gresho, P. M. and R. L. Sani. *Incompressible Flow and the Finite Element Method*. John Wiley and Sons, 1998.
- [187] Venditti, D. A. and B. R. Baliga. "An h-Adaptive Strategy for CVFEM Simulations of Viscous Incompressible Flow". Proc. 6th Annual Conference of the Computational Fluid Dynamics Society of Canada (CFD 98), pp. VIII-65–VIII-70, Quebec City, Canada, June 7-9, 1998.
- [188] Reyes, M., J. Rincon, J. and J. Damia. "Simulation of Turbulent Flow in Irregular Geometries Using a Control-Volume Finite-Element Method". *Numerical Heat Transfer, Part B*, \bf 39(1):79–90, 2001.
- [189] Campos Silva, J. B. and L. F. M. de Moura. "A Control-Volume Finite-Element Method (CVFEM) for Unsteady, Incompressible, Viscous Fluid Flows". *Numerical Heat Transfer, Part B*, \bf 40(1):61–82, 2001.
- [190] Zhao, Y., J. Tai, and F. Ahmed . "Simulation of Micro Flows with Moving Boundaries Using High-Order Upwind FV Method on Unstructured Grids". *Computational Mechanics*, \bf 28:66–75, 2002.
- [191] Kettleborough, C. F., S. R. Husain, and C. Prakash. "Solution of Fluid Flow Problems with the Vorticity-Streamfunction Formulation and the Control Volume Based Finite-Element Method". *Numerical Heat Transfer, Part B*, \bf 16(1):31–58, 1989.

- [192] Choudhury, S. and R. A. Nicolaides. "Discretization of Incompressible Vorticity-Velocity Equations on Triangular Meshes". *International Journal for Numerical Methods in Fluids*, \bf 11(6):823–, 1990.
- [193] Krakov, M. S. "Control Volume Finite-Element Method for Navier-Stokes Equations in Vortex-Streamfunction Formulation". *Numerical Heat Transfer, Part B*, \bf 21(2):125–145, 1992.
- [194] Elkaim, D., M. Reggio, and R. Camarero. "Simulating Two-Dimensional Turbulent Flow by Using the k-epsilon Model and the Vorticity-Streamfunction Formulation". *International Journal for Numerical Methods in Fluids*, \bf 14(8):961–980, 1992.
- [195] Elkaim, D., F. McKenty, M. Reggio, and R. Camarero. "Control Volume Finite Element Solution of Confined Turbulent Swirling Flows". *International Journal for Numerical Methods in Fluids*, \bf 19(2):135–152, 1994.
- [196] Banaszek, J. A. "A Conservative Finite Element Method for Heat Conduction Problems". *International Journal for Numerical Methods in Engineering*, \bf 20:2033–2050, 1984.
- [197] Blackwell, B. F. "Numerical Prediction of One-Dimensional Ablation Using a Finite Control Volume Procedure with Exponential Differencing". *Numerical Heat Transfer*, \bf 14:17–34, 1988.
- [198] Abboud, J. B. and H. Hardisty. "Control-Volume Energy-Balance FE Formulations of the 8-Node Hexahedron Element". *Communications in Applied Numerical Methods*, \bf 7:141–153, 1991.
- [199] Blackwell, B. F. and R. E. Hogan. "Numerical Solution of Axisymmetric Heat Conduction Problems Using Finite Control Volume Technique". *Journal of Thermophysics and Heat Transfer*, \bf 7:462–471, 1993.
- [200] Blackwell, B. F. and R. E. Hogan. "One-Dimensional Ablation Using Landau Transformation and Finite Control Volume Procedure". *Journal of Thermophysics and Heat Transfer*, \bf 8(2):282–287, 1994.
- [201] Ferguson, W. J. and I. W. Turner. "Control Volume Finite Element Model of Mechano-Sorptive Creep in Timber". *Numerical Heat Transfer, Part A*, \bf 29(2):147–164, 1996.
- [202] Ferguson, W. J. and I. W. Turner. "A Control Volume Finite Element Numerical Simulation of the Drying of Spruce". *Journal of Computational Physics*, \bf 125(1):59–70, 1996.
- [203] Ferguson, W. J. "A Control Volume Finite Element Numerical Solution of Creep Problems". *International Journal for Numerical Methods in Engineering*, \bf 40(18):3463–, 1997.
- [204] Forsyth, P. A. "Control Volume Finite Element Approach to NAPL Groundwater Contamination". *SIAM Journal on Scientific and Statistical Computing*, \bf 12(5):1029–1057, 1991.
- [205] Letniowski, F. W. and P. A. Forsyth. "A Control Volume Finite Element Method for

- Three-Dimensional NAPL Groundwater Contamination". *International Journal for Numerical Methods in Fluids*, \bf 13(8):955–970, 1991.
- [206] Fung, L. S. K., A. D. Hiebert, and L. X. Nghiem. "Reservoir Simulation With a Control-Volume Finite Element Method". *SPE Reservoir Engineering*, \bf 7(3):349–, 1992.
- [207] Durlofsky, L. J. "Accuracy of Mixed and Control Volume Finite Element Approximations to Darcy Velocity and Related Quantities". *Water Resources Research*, \bf 30(4):965–973, 1994.
- [208] Eymard, R. and F. Sonier. "Mathematical and Numerical Properties of Control-Volume, Finite-Element Scheme for Reservoir Simulation". *SPE Reservoir Engineering*, \bf 9(4):283–, 1994.
- [209] Fung, L. S. K., L. Buchanan, and R. Sharma. "Hybrid-CVFE Method for Flexible-Grid Reservoir Simulation". *SPE Reservoir Engineering*, \bf 9(3):188–194, 1994.
- [210] Jones, J. E., Z. Cai, S. F. McCormick, and T. F. Russell. "Control-Volume Mixed Finite Element Methods". Technical Report TR-97-16, ICASE, Langley, VA, February 1997.
- [211] Gottardi, G. and M. Venuttelli. "A Control-Volume Finite-Element Model for Two-Dimensional Overland Flow". *Advances in Water Resources*, \bf 16(5):277–, 1993.
- [212] Di Giammarco, P., E. Todini and P. Lamberti. "A Conservative Finite Elements Approach to Overland Flow: The Control Volume Finite Element Formulation". *Journal of Hydrology*, \bf 175(1-4):267–291, 1996.
- [213] Oñate, E., M. Cervera, and O. C. Zienkiewicz. "A Finite Volume Method for Structural Mechanics". *Internation Journal for Numerical Methods in Engineering*, \bf 37:181–201, 1994.
- [214] Bailey, C. and M. Cross. "A Finite Volume Procedure to Solve Elastic Solid Mechanics Problems in Three Dimensions on an Unstructured Mesh". *Internation Journal for Numerical Methods in Engineering*, \bf 38:1757–1776, 1995.
- [215] Majumdar, S. "Role of Under-relaxation in Momentum Interpolation for Calculation of Flow with Non-staggered Grids". *Numerical Heat Transfer*, \bf 13:125–132, 1988.
- [216] Perić, M., R. Kessler, and G. Scheuerer. "Comparison of Finite-Volume Numerical Methods with Staggered and Colocated Grids". *Computers and Fluids*, \bf 16(4):389–403, 1988.
- [217] Papageorgakopoulos, J. G., Arampatzis, D., Assimacopoulos, and N. C. Markatos. "Enhancement of the momentum interpolation method on non-structured grids". *Int. J. Numer. Meth. Fluids*, \bf 33:1–22, 2000.
- [218] Codina, R. "Pressure stability in fractional step finite element methods for incompressible flows". *J. Comp. Phys.*, \bf 170:112–140, 2001.
- [219] Soto, O. R., F. Löhner, and J. Cebal. "An implicit monolithic time accurate finite element scheme for incompressible flow problems". *AIAA-2001-2616*, 2001.

- [220] Almgren, A. S., J. B. Bell, and W. Y. Crutchfield. "Approximate projection methods: part I. inviscid analysis". *SIAM J. Sci. Comp.*, \bf 22:1139–1159, 2000.
- [221] Codina, R. and S. Badia. "On some pressure segregation methods of fractional-step type for the finite element approximation of incompressible flow problems". *Comp. Methods. Appl. Mech. Engr.*, \bf 170:112–140, 2005.
- [222] Dukowicz, J. K. and A. S. Dvinsky. "Approximate Factorization as a High Order Splitting for the Implicit Incompressible Flow Equations". *Journal of Computational Physics*, \bf 102:336–347, 1992.
- [223] Strikwerda, J. C. and Y. S. Lee. "The accuracy of the fractional step method". *SIAM J. Numer. Anal.*, \bf 37:37–48, 1999.
- [224] Perot, J. B. "An Analysis of the Fractional Step Method". *Journal of Computational Physics*, \bf 108:51–58, 1993.
- [225] Chorin, A. J. "Numerical Solution of the Navier-Stokes Equations". *Mathematics of Computation*, \bf 22(104):745–762, 1968.
- [226] Kim, D. and H. Choi. "A second-order time-accurate finite volume method for unsteady incompressible flow on hybrid unstructured grids". *J. Comp. Phys.*, \bf 162:411–428, 2000.
- [227] Kim, J. and P. Moin. "Application of a Fractional Step Method to Incompressible Navier-Stokes Equations". *Journal of Computational Physics*, \bf 59(2):308–323, 1985.
- [228] Brown, D. L., R. Cortez, and M. Minion. "Accurate projection method for the incompressible Navier-Stokes equations". *J. Comp. Phys.*, \bf 168:464–499, 2001.
- [229] Domino et. al. "Fuego Verification Manual". Sandia National Laboratories, <http://scico.sandia.gov/fuego>, 2007.
- [230] Ober, C. C. and J. N. Shadid. "Studies on the accuracy of time-integration methods for the radiation-diffusion equations". *J. Comp. Phys.*, \bf 195:743–772, 2004.
- [231] Hirsch, C. *Numerical Computation of Internal and External Flows, Volume 2*. John Wiley & Sons, 1990.
- [232] Jameson, A. "Artificial Diffusion, Upwind Biasing, Limiters and Their Effect on Accuracy and Multigrid Convergence in Transonic and Hypersonic Flows". AIAA Paper 93-3359, 11th AIAA Computational Fluid Dynamics Conference, Orlando, FL, July 1993.
- [233] Christie, I. and C. Hall. "The Maximum Principle for Bilinear Elements". *International Journal for Numerical Methods in Engineering*, \bf 20(3):549–553, 1984.
- [234] Yeap, C. F. and J. A. Pearce. "A Unified Subroutine for the Solution of 2D and 3D Axisymmetric Diffusion Equation". *Advances in Engineering Software*, \bf 11(3):118–127, 1989.
- [235] Blackwell, B. F., R. J. Cochran, and R. E. Hogan. "A Formal Method for Computing Thermal Conductors for Arbitrary, Complex Geometries". ASME-HTD Vol. 311, pp. 31–42, 30th National Heat Transfer Conference, Portland, OR, August 1995.

- [236] Gresho, P. M, and R. L. Lee. "Don't Suppress the Wiggles—They're Telling You Something". *Computers and Fluids*, \bf 9(2):223–253, 1981.
- [237] Flanagan, D. P. and T. Belytschko. "A Uniform Strain Hexahedron and Quadrilateral with Orthogonal Hourglass Control". *International Journal for Numerical Methods in Engineering*, \bf 17:679–706, 1981.
- [238] Aftosmis, M. "Upwind Method for Simulation of Viscous Flow on Adaptively Refined Meshes". *AIAA Journal*, \bf 32(2):268–277, 1994.
- [239] Kallinderis, Y. G. and J. R. Baron. "Adaptive Methods for a New Navier-Stokes Algorithm". *AIAA Journal*, \bf 27(1):37–43, 1989.
- [240] Kallinderis, Y. and P. Vijayan. "Adaptive Refinement-Coarsening Scheme for Three-Dimensional Unstructured Meshes". *AIAA Journal*, \bf 31(8):1440–1447, 1993.
- [241] Mavriplis, D. J. "Adaptive Meshing Techniques for Viscous Flow Calculations on Mixed Element Unstructured Meshes". *International Journal for Numerical Methods in Fluids*, \bf 34(2):93–111, 2000.
- [242] S. Gordan and B. J. McBride. Computer program for calculation of complex chemical equilibrium compositions and applications I. analysis. Technical Report Ref. Publication 1311, NASA, October 1994.
- [243] B. J. McBride and S. Gordan. Computer program for calculation of complex chemical equilibrium compositions and applications II. users manual and program description. Technical Report Ref. Publication 1311, NASA, June 1996.



Sandia
National
Laboratories

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology & Engineering Solutions of Sandia LLC, a wholly owned subsidiary of Honeywell International Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.