

NIST Special Publication 1019
Sixth Edition

Fire Dynamics Simulator

User's Guide

Kevin McGrattan
Simo Hostikka
Randall McDermott
Jason Floyd
Craig Weinschenk
Kristopher Overholt

<http://dx.doi.org/10.6028/NIST.SP.1019>



VTT Technical Research Centre of Finland

NIST
National Institute of
Standards and Technology
U.S. Department of Commerce

NIST Special Publication 1019
Sixth Edition

Fire Dynamics Simulator
User's Guide

Kevin McGrattan
Randall McDermott
Fire Research Division
Engineering Laboratory
Gaithersburg, Maryland, USA

Simo Hostikka
Aalto University
Espoo, Finland

Jason Floyd
JENSEN HUGHES
Baltimore, Maryland, USA

Craig Weinschenk
UL Firefighter Safety Research Institute
Columbia, Maryland, USA

Kristopher Overholt
Continuum Analytics
Austin, Texas, USA

<http://dx.doi.org/10.6028/NIST.SP.1019>

March 6, 2017
FDS Version 6.5.3
Revision: FDS6.5.3-807-ge3ed714-dirty



U.S. Department of Commerce
Penny Pritzker, Secretary

National Institute of Standards and Technology
Willie May, Under Secretary of Commerce for Standards and Technology and Director

Certain commercial entities, equipment, or materials may be identified in this document in order to describe an experimental procedure or concept adequately. Such identification is not intended to imply recommendation or endorsement by the National Institute of Standards and Technology, nor is it intended to imply that the entities, materials, or equipment are necessarily the best available for the purpose.

National Institute of Standards and Technology Special Publication 1019
Natl. Inst. Stand. Technol. Spec. Publ. 1019, 296 pages (October 2013)
CODEN: NSPUE2

FDS Developers

The Fire Dynamics Simulator and Smokeview are the products of an international collaborative effort led by the National Institute of Standards and Technology (NIST) and VTT Technical Research Centre of Finland. Its developers and contributors are listed below.

Principal Developers of FDS

Kevin McGrattan, NIST
Simo Hostikka, Aalto University
Randall McDermott, NIST
Jason Floyd, JENSEN HUGHES, Baltimore, Maryland, USA
Craig Weinschenk, UL Firefighter Safety Research Institute, Columbia, Maryland, USA
Kristopher Overholt, Continuum Analytics, Austin, Texas, USA

Principal Developer of Smokeview

Glenn Forney, NIST

Principal Developer of FDS+Evac

Timo Korhonen, VTT

Contributors

Daniel Haarhoff, Jülich Supercomputing Centre, Germany
Susan Kilian, hhpberlin, Germany
Vivien Lecoustre, University of Maryland, USA
Anna Matala, VTT
William Mell, U.S. Forest Service, Seattle, Washington, USA
Topi Sikanen, VTT
Ben Trettel, The University of Texas at Austin, USA
Julio Cesar Silva, National Council for Scientific and Technological Development, Brazil
Benjamin Ralph, University of Edinburgh, UK

About the Developers

Kevin McGrattan is a mathematician in the Fire Research Division of NIST. He received a bachelor of science degree from the School of Engineering and Applied Science of Columbia University in 1987 and a doctorate at the Courant Institute of New York University in 1991. He joined the NIST staff in 1992 and has since worked on the development of fire models, most notably the Fire Dynamics Simulator.

Simo Hostikka is an associate professor of fire safety engineering at Aalto University School of Engineering, since January 2014. Before joining Aalto, he worked as a Principal Scientist and Team Leader at VTT Technical Research Centre of Finland. He received a master of science (technology) degree in 1997 and a doctorate in 2008 from the Department of Engineering Physics and Mathematics of the Helsinki University of Technology. He is the principal developer of the radiation and solid phase sub-models within FDS.

Randall McDermott joined the Fire Research Division at NIST in 2008. He received a B.S. from the University of Tulsa in Chemical Engineering in 1994 and a Ph.D. from the University of Utah in 2005. His research interests include subgrid-scale models and numerical methods for large-eddy simulation, turbulent combustion, immersed boundary methods, and Lagrangian particle methods.

Jason Floyd is a Senior Engineer at JENSEN HUGHES, in Baltimore, Maryland. He received a bachelor of science and a doctorate in the Nuclear Engineering Program of the University of Maryland. After graduating, he was awarded a National Research Council Post-Doctoral Fellowship at the Building and Fire Research Laboratory of NIST. He is a principal developer of the combustion, control logic, and HVAC sub-models within FDS.

Craig Weinschenk is an engineer at Underwriters Laboratories Firefighter Safety Research Institute, in Columbia, Maryland. He worked in the Fire Research Division at NIST as a National Research Council Postdoctoral Research Associate in 2011. He received a B.S. from Rowan University in 2006 in Mechanical Engineering. He received an M.S. in 2007 and a doctorate in 2011 from The University of Texas at Austin in Mechanical Engineering. His research interests include numerical combustion, fire-structure interaction, and human factors research of fire-fighting tactics.

Kristopher Overholt is a software engineer at Continuum Analytics, developers of the Anaconda Python distribution. He received a B.S. in Fire Protection Engineering Technology from the University of Houston-Downtown in 2008, an M.S. in Fire Protection Engineering from Worcester Polytechnic Institute in 2010, and a Ph.D. in Civil Engineering from The University of Texas at Austin in 2013. He worked in the Fire Research Division at NIST from 2013 to 2015, where he was central to the development of the FDS continuous integration framework, Firebot. He also worked on aspects of FDS related to verification and validation and quality metrics. His research interests include inverse fire modeling problems, soot deposition in fires, and the use of fire models in forensic applications.

Glenn Forney is a computer scientist in the Fire Research Division of NIST. He received a bachelor of science degree in mathematics from Salisbury State College and a master of science and a doctorate in mathematics from Clemson University. He joined NIST in 1986 (then the National Bureau of Standards) and has since worked on developing tools that provide a better understanding of fire phenomena, most notably Smokeview, an advanced scientific software tool for visualizing Fire Dynamics Simulation data.

Timo Korhonen is a Senior Scientist at VTT Technical Research Centre of Finland. He received a master of science (technology) degree in 1992 and a doctorate in 1996 from the Department of Engineering Physics and Mathematics of the Helsinki University of Technology. He is the principal developer of the evacuation sub-model within FDS.

Daniel Haarhoff did his masters work at the Jülich Supercomputing Centre in Germany, graduating in 2015. His thesis is on providing and analyzing a hybrid parallelization of FDS. For this, he implemented OpenMP into FDS 6.

Susan Kilian is a mathematician with numerics and scientific computing expertise. She received her diploma from the University of Heidelberg and received her doctorate from the Technical University of Dortmund in 2002. Since 2007 she has been a research scientist for hhpberlin, a fire safety engineering firm located in Berlin, Germany. Her research interests include high performance computing and the development of efficient parallel solvers for the pressure Poisson equation.

Vivien Lecoustre is a Research Associate at the University of Maryland. He received a master of science in Aerospace Engineering from ENSMA (France) in 2005 and a doctorate in Mechanical Engineering from the University of Maryland in 2009. His research interests include radiation properties of fuels and numerical turbulent combustion.

Anna Matala is a Research Scientist at VTT Technical Research Centre of Finland and a Ph.D. candidate at Aalto University School of Science. She received her M.Sc. degree in Systems and Operations Research from Helsinki University of Technology in 2008. Her research concentrates on pyrolysis modelling and parameter estimation in fire simulations

William (Ruddy) Mell is an applied mathematician currently at the U.S. Forest Service in Seattle, Washington. He holds a B.S. degree from the University of Minnesota (1981) and doctorate from the University of Washington (1994). His research interests include the development of large-eddy simulation methods and sub-models applicable to the physics of large fires in buildings, vegetation, and the wildland-urban interface.

Topi Sikanen is a Research Scientist at VTT Technical Research Centre of Finland and a graduate student at Aalto University School of Science. He received his M.Sc. degree in Systems and Operations Research from Helsinki University of Technology in 2008. He works on the Lagrangian particle and liquid evaporation models.

Ben Trettel is a graduate student at The University of Texas at Austin. He received a B.S. in Mechanical Engineering in 2011 and an M.S. in Fire Protection Engineering in 2013, both from the University of Maryland. He develops models for the transport of Lagrangian particles for the Fire Dynamics Simulator.

Julio Cesar Silva is a Guest Researcher in the Fire Research Division of NIST from National Council for Scientific and Technological Development, Brazil. He received a M.Sc. in 2010 and a doctorate in 2014 from Federal University of Rio de Janeiro in Civil Engineering. His research interests include fire-structure interaction and he develops coupling strategies between FDS and finite-element codes.

Benjamin Ralph is a fire safety engineer and Ph.D. student at the BRE Centre for Fire Safety Engineering at University of Edinburgh, UK. He received his M.Eng. in Civil Engineering from the University of Southampton, UK in 2008 and his P.G.Dip. in Fire Safety Engineering from the University of Ulster, UK in 2014. He was a Guest Researcher in the Engineered Fire Safety Group at NIST in 2016. His research interests include coupled hybrid modeling and performance-based design in fire safety engineering. He is a developer of the HVAC sub-model - specifically the transient mass and energy transport solver.

Preface

This Guide describes how to use the Fire Dynamics Simulator (FDS). Because new features are added periodically, check the current version number on the inside front jacket of this manual.

Note that this Guide does not provide the background theory for FDS. A four volume set of companion documents, referred to collectively as the FDS Technical Reference Guide [?], contains details about the governing equations and numerical methods, model verification, experimental validation, and configuration management. The FDS User's Guide contains limited information on how to operate Smokeview, the companion visualization program for FDS. Its full capability is described in the Smokeview User's Guide [?].

Disclaimer

The US Department of Commerce makes no warranty, expressed or implied, to users of the Fire Dynamics Simulator (FDS), and accepts no responsibility for its use. Users of FDS assume sole responsibility under Federal law for determining the appropriateness of its use in any particular application; for any conclusions drawn from the results of its use; and for any actions taken or not taken as a result of analysis performed using these tools.

Users are warned that FDS is intended for use only by those competent in the fields of fluid dynamics, thermodynamics, heat transfer, combustion, and fire science, and is intended only to supplement the informed judgment of the qualified user. The software package is a computer model that may or may not have predictive capability when applied to a specific set of factual circumstances. Lack of accurate predictions by the model could lead to erroneous conclusions with regard to fire safety. All results should be evaluated by an informed user.

Throughout this document, the mention of computer hardware or commercial software does not constitute endorsement by NIST, nor does it indicate that the products are necessarily those best suited for the intended purpose.

Acknowledgments

The Fire Dynamics Simulator, in various forms, has been under development for almost 25 years. It was first released to the public in 2000. Since then, continued improvements have been made to the software based largely on feedback from its users. Included below are some who made important contributions related to the application of FDS.

- At NIST, Dan Madrzykowski, Doug Walton, Bob Vettori, Dave Stroup, Steve Kerber, Nelson Bryner, and Adam Barowy have used FDS and Smokeview as part of several investigations of fire fighter line of duty deaths. They have provided valuable information on the model's usability and accuracy when compared to large scale measurements made during fire reconstructions.
- Bryan Klein of Thunderhead Engineering assisted in adding cross-referencing functionality to this document, making it easier to view electronically. He also designed the on-line services for revision control, bug reporting, and general discussion of topics related to FDS.
- At VTT, Joonas Ryyänen implemented and documented the FED/FIC routine.
- The US Nuclear Regulatory Commission has provided financial support for the verification and validation of FDS, along with valuable insights into how fire models are used as part of probabilistic risk assessments of nuclear facilities. Special thanks to Mark Salley and Dave Stroup.
- The Society of Fire Protection Engineers (SFPE) sponsors a training course on the use of FDS and Smokeview. Chris Wood of ArupFire, Dave Sheppard of the US Bureau of Alcohol, Tobacco and Firearms (ATF), and Doug Carpenter of Combustion Science and Engineering developed the materials for the course, along with Morgan Hurley of the SFPE.
- David McGill of Seneca College, Ontario, Canada has conducted a remote-learning course on the use of FDS, and he has also maintained a web site that has provided valuable suggestions from users.
- Paul Hart of Swiss Re, GAP Services, and Pravinray Gandhi of Underwriters Laboratories provided useful suggestions about water droplet transport on solid objects.
- François Demouge of the Centre Scientifique et Technique du Bâtiment (CSTB) in France assisted with implementation of synthetic turbulence inflow boundary conditions.
- Max Gould, Summer Undergraduate Research Fellow, assisted in the testing and verification of non-standard boundary treatment methods.

Finally, on the following pages is a list of individuals and organizations who have volunteered their time and effort to “beta test” FDS and Smokeview prior to its official release. Their contribution is invaluable because there is simply no other way to test all of the various features of the model.

FDS 6 Beta Testers	
Mohammed Assal	CFD Algeria
Choon-Bum Choi	Building and Tunnel Technologies Inc. (BNTTEK), Korea
William J. Ferrante	Roosevelt Fire District, Hyde Park, New York, USA
Emanuele Gissi	Corpo nazionale dei Vigili del Fuoco, Italy
Timothy M. Groch	Engineering Planning and Management, Inc., Framingham, Massachusetts, USA
Georges Guigay	Mannvit Engineering, Iceland
Simon J. Ham	Fire Safety Engineering Consultants Limited, UK
Chris Lautenberger	Reax Engineering, Berkeley, California, USA
Tim McDonald	Endress Ingenieurgesellschaft mbH, Germany
Dave McGill	Seneca College, Ontario, Canada
Adrian Milford	Sereca Fire Consulting Ltd., British Columbia, Canada
Luca Nassi	National Fire Department, Italy
Stephen Olenick	Combustion Science and Engineering, Inc., Columbia, Maryland, USA
Natalie Ong	Arup Fire Singapore
Chris Salter	Hoare Lea and Partners, UK
Joakim Sandström	LTU/Brandskyddslaget, Sweden
Julio Cesar Silva	Federal University of Rio de Janeiro, Brazil
Boris Stock	BFT Cognos GmbH, Aachen, Germany
Csaba Szilagyi	OPTOMM Ltd., Budapest, Hungary
Giacomo Villi	Università di Padova, Italy
Andreas Vischer	Wijnveld//Ingenieure, Osnabrück, Germany
Christopher Wood	FireLink, LLC, Tewksbury, Massachusetts, USA

Contents

FDS Developers	i
About the Developers	iii
Preface	vii
Disclaimer	ix
Acknowledgments	xi
Contents	xiii
List of Figures	xxi
List of Tables	xxiii
I The Basics of FDS	1
1 Introduction	3
1.1 Features of FDS	3
1.2 What's New in FDS 6?	4
1.3 Changes to Input Parameters in FDS 6	6
1.4 A Note on Longer Run Times in FDS 6	9
2 Getting Started	11
2.1 How to Acquire FDS and Smokeview	11
2.2 Computer Hardware Requirements	11
2.3 Computer Operating System (OS) and Software Requirements	12
2.4 Installation Testing	12
3 Running FDS	13
3.1 A Brief Primer on Computer Hardware	13
3.2 Starting an FDS Calculation	14
3.2.1 Single or Multiple Mesh Simulation with OpenMP Parallel Processing	15
3.2.2 Multiple Mesh Simulation with MPI Parallel Processing	17
3.2.3 Using MPI and OpenMP Together	19
3.2.4 Efficiency of an MPI Calculation	19
3.2.5 Running Very Large Jobs	20
3.3 Monitoring Progress	21

4	User Support	23
4.1	The Version Number	23
4.2	Common Error Statements	24
4.3	Support Requests and Bug Tracking	26
II	Writing an FDS Input File	27
5	The Basic Structure of an Input File	29
5.1	Naming the Input File	29
5.2	Namelist Formatting	29
5.3	Input File Structure	30
6	Setting the Bounds of Time and Space	33
6.1	Naming the Job: The HEAD Namelist Group (Table 19.7)	33
6.2	Simulation Time: The TIME Namelist Group (Table 19.28)	33
6.2.1	Basics	33
6.2.2	Special Topic: Controlling the Time Step	34
6.2.3	Special Topic: Steady-State Applications	34
6.3	Computational Meshes: The MESH Namelist Group (Table 19.13)	35
6.3.1	Basics	35
6.3.2	Two-Dimensional and Axially-Symmetric Calculations	35
6.3.3	Multiple Meshes	35
6.3.4	Mesh Alignment	37
6.3.5	Mesh Stretching: The TRNX, TRNY and TRNZ Namelist Groups (Table 19.29)	39
6.3.6	Mesh Resolution	41
6.4	Miscellaneous Parameters: The MISC Namelist Group (Table 19.14)	42
6.4.1	Basics	42
6.4.2	Special Topic: Specifying a Force Field	42
6.4.3	Special Topic: Stopping and Restarting Calculations	43
6.4.4	Special Topic: Initializing a 3D Velocity Field	43
6.4.5	Special Topic: Turning off the Flow Field	44
6.4.6	Special Topic: Defying Gravity	44
6.4.7	Special Topic: The Baroclinic Vorticity	45
6.4.8	Special Topic: Large Eddy Simulation Parameters	45
6.4.9	Special Topic: Numerical Stability Parameters	46
6.4.10	Special Topic: Flux Limiters	49
6.5	Initial Conditions: The INIT Namelist Group (Table 19.10)	49
6.6	The Pressure Solver: The PRES Namelist Group (Table 19.18)	51
6.6.1	Parameters Related to the Solution of Poisson Equation for Pressure	51
6.6.2	Pressure Considerations in Long Tunnels	53
6.6.3	Parameters Related to the Background Pressure when Breaking Pressure Zones	55
6.7	Setting Limits: The CLIP Namelist Group (Table 19.2)	55
7	Building the Model	57
7.1	Bounding Surfaces: The SURF Namelist Group (Table 19.26)	57
7.2	Creating Obstructions: The OBST Namelist Group (Table 19.16)	57
7.2.1	Basics	57

7.2.2	Thin Obstructions	58
7.2.3	Overlapping Obstructions	58
7.2.4	Preventing Obstruction Removal	59
7.2.5	Transparent or Outlined Obstructions	59
7.2.6	Creating Holes in Obstructions: The HOLE Namelist Group (Table 19.8)	59
7.3	Applying Surface Properties: The VENT Namelist Group (Table 19.30)	60
7.3.1	Basics	60
7.3.2	Special Vents	61
7.3.3	Controlling Vents	63
7.3.4	Trouble-Shooting Vents	64
7.4	Coloring Obstructions, Vents, Surfaces and Meshes	64
7.4.1	Colors	64
7.4.2	Texture Maps	65
7.5	Repeated Objects: The MULT Namelist Group (Table 19.15)	65
8	Fire and Thermal Boundary Conditions	69
8.1	Basics	69
8.2	Surface Temperature and Heat Flux	70
8.2.1	Specified Solid Surface Temperature	70
8.2.2	Special Topic: Convective Heat Transfer Options	70
8.2.3	Special Topic: Adiabatic Surfaces	71
8.3	Heat Conduction in Solids	72
8.3.1	Structure of Solid Boundaries	72
8.3.2	Thermal Properties	73
8.3.3	Back Side Boundary Conditions	74
8.3.4	Initial and Back Side Temperature	74
8.3.5	Walls with Different Materials Front and Back	74
8.3.6	Special Topic: Specified Internal Heat Source	76
8.3.7	Special Topic: Non-Planar Walls and Targets	76
8.3.8	Special Topic: Solid Phase Numerical Gridding Issues	76
8.3.9	Solid Heat Transfer 3D (Beta)	77
8.4	Simple Pyrolysis Models	78
8.4.1	A Gas Burner with a Specified Heat Release Rate	78
8.4.2	Special Topic: A Radially-Spreading Fire	78
8.4.3	Solid Fuels that Burn at a Specified Rate	79
8.5	Complex Pyrolysis Models	80
8.5.1	Reaction Mechanism	80
8.5.2	Reaction Rates	81
8.5.3	Shrinking and swelling materials	84
8.5.4	Multiple Solid Phase Reactions	86
8.5.5	The Heat of Reaction	86
8.5.6	Special Topic: The “Threshold” Temperature	86
8.5.7	Liquid Fuels	87
8.5.8	Fuel Burnout	88
8.6	Testing Your Pyrolysis Model	90
8.6.1	Simulating the Cone Calorimeter	90
8.6.2	Simulating Bench-scale Measurements like the TGA, DSC, and MCC	93

9	Ventilation	95
9.1	Simple Vents, Fans and Heaters	95
9.1.1	Simple Supply and Exhaust Vents	95
9.1.2	Total Mass Flux	96
9.1.3	Heaters	96
9.1.4	Louvered Vents	97
9.1.5	Specified Normal Velocity Gradient	97
9.1.6	Species and Species Mass Flux Boundary Conditions	98
9.1.7	Tangential Velocity Boundary Conditions at Solid Surfaces	98
9.1.8	Synthetic Turbulence Inflow Boundary Conditions	99
9.1.9	Random Mass Flux Variation	100
9.2	HVAC Systems: The HVAC Namelist Group (Table 19.9)	101
9.2.1	HVAC Duct Parameters	102
9.2.2	HVAC Dampers	103
9.2.3	HVAC Node Parameters	104
9.2.4	HVAC Fan Parameters	105
9.2.5	HVAC Filter Parameters	107
9.2.6	HVAC Aircoil Parameters	108
9.2.7	Louvered HVAC Vents	109
9.2.8	HVAC Mass Transport	110
9.3	Pressure-Related Effects: The ZONE Namelist Group (Table 19.30)	110
9.3.1	Specifying Pressure Zones	111
9.3.2	Leaks	114
9.4	Pressure Boundary Conditions	116
9.5	Special Flow Profiles	119
10	Atmospheric Effects	121
10.1	Temperature Stratification	121
10.1.1	Stack Effect	121
10.2	Wind	122
10.2.1	Specifying a “Wall of Wind”	125
11	User-Specified Functions	127
11.1	Time-Dependent Functions	127
11.2	Temperature-Dependent Functions	128
11.3	Spatially-Dependent Velocity Profiles	129
12	Chemical Species	131
12.1	Specifying Primitive Species	131
12.1.1	Basics	132
12.1.2	Specifying Gas and Liquid Species Properties	133
12.1.3	Air	137
12.2	Specifying Lumped Species (Mixtures of Primitive Species)	138
12.2.1	Combining Lumped and Primitive Species	140
13	Combustion	141
13.1	Single-Step, Mixing-Controlled Combustion	141
13.1.1	Simple Chemistry Parameters	141

13.1.2	Heat of Combustion	143
13.1.3	Special Topic: Turbulent Combustion	145
13.1.4	Special Topic: Flame Extinction	145
13.2	Complex Stoichiometry	147
13.2.1	Complex Fuel Molecules	148
13.2.2	Multiple Chemical Reactions	149
13.2.3	Special Topic: Using the EQUATION input parameter	152
13.3	Finite Rate Combustion	153
13.4	Special Topic: Chemical Time Integration	155
13.5	Special Topic: Aerosol Deposition	155
13.5.1	Example Case: Soot Deposition from a Propane Flame	155
14	Radiation	157
14.1	Basic Radiation Parameters: The RADI Namelist Group	157
14.1.1	Radiative Fraction	157
14.1.2	Spatial and Temporal Resolution of the Radiation Transport Equation	159
14.2	Radiative Absorption and Scattering	159
14.2.1	RadCal Considerations	159
14.2.2	Radiative Absorption and Scattering by Particles	160
14.2.3	Wide Band Model	160
15	Particles and Droplets	161
15.1	Basics	161
15.2	Massless Particles	161
15.3	Liquid Droplets	162
15.3.1	Thermal Properties	162
15.3.2	Radiative Properties	162
15.3.3	Size Distribution	163
15.3.4	Secondary Breakup	165
15.3.5	Fuel Droplets	165
15.4	Solid Particles	167
15.4.1	Basic Geometry and Boundary Conditions	167
15.4.2	Drag	168
15.4.3	Splitting Particles	168
15.4.4	Gas Generating Particles	168
15.4.5	Vegetation	170
15.4.6	Firebrands	171
15.4.7	Porous Media	172
15.4.8	Screens	173
15.4.9	Electrical Cable Failure	174
15.5	Particle Insertion	175
15.5.1	Particles Introduced at a Solid Surface	176
15.5.2	Particles or Droplets Introduced at a Sprinkler or Nozzle	177
15.5.3	Particles or Droplets Introduced within a Volume	177
15.6	Special Topic: Suppression by Water	179
15.6.1	Velocity on Solid Surfaces	179
15.6.2	Reduction of the Burning Rate	180

16	Vegetation	183
16.1	Using Particles to Represent Vegetation	183
16.2	Conversion of Vegetation Parameters Commonly Used in Forestry to those Required by FDS	184
17	Devices and Control Logic	187
17.1	Device Location and Orientation: The DEVC Namelist Group (Table 19.5)	187
17.2	Device Output	188
17.3	Special Device Properties: The PROP Namelist Group (Table 19.20)	189
17.3.1	Sprinklers	189
17.3.2	Nozzles	192
17.3.3	Special Topic: Specified Entrainment (Velocity Patch)	194
17.3.4	Heat Detectors	194
17.3.5	Smoke Detectors	194
17.3.6	Beam Detection Systems	196
17.3.7	Aspiration Detection Systems	198
17.4	Basic Control Logic	199
17.4.1	Creating and Removing Obstructions	200
17.4.2	Activating and Deactivating Vents	201
17.5	Advanced Control Functions: The CTRL Namelist Group	201
17.5.1	Control Functions: ANY, ALL, ONLY, and AT_LEAST	203
17.5.2	Control Function: TIME_DELAY	203
17.5.3	Control Function: DEADBAND	204
17.5.4	Control Function: RESTART and KILL	204
17.5.5	Control Function: CUSTOM	205
17.5.6	Control Function: Math Operations	205
17.5.7	Control Function: PID Control Function	206
17.5.8	Combining Control Functions: A Pre-Action Sprinkler System	206
17.5.9	Combining Control Functions: A Dry Pipe Sprinkler System	207
17.5.10	Example Case: activate_vents	208
17.6	Controlling a RAMP	208
17.6.1	Changing the Independent variable	208
17.6.2	Freezing the Output Value, Example Case: hrr_freeze	208
17.7	Visualizing FDS Devices in Smokeview	210
17.7.1	Devices that Indicate Activation	210
17.7.2	Devices with Variable Properties	212
17.7.3	Objects that Represent Lagrangian Particles	214
18	Output	217
18.1	Output Control Parameters: The DUMP Namelist Group	217
18.2	Device Output: The DEVC Namelist Group	218
18.2.1	Single Point Output	218
18.2.2	Linear Array of Point Devices	219
18.2.3	Quantities at Certain Depth	220
18.2.4	Back Surface Temperature	221
18.3	Profiles of Quantities: The PROF Namelist Group	221
18.4	Animated Planar Slices: The SLCF Namelist Group	221
18.5	Animated Boundary Quantities: The BNDF Namelist Group	222
18.6	Animated Isosurfaces: The ISOF Namelist Group	223

18.7	Plot3D Static Data Dumps	223
18.8	SMOKE3D: Realistic Smoke and Fire	223
18.9	Particle Output Quantities	224
18.9.1	Liquid Droplets that are Attached to Solid Surfaces	224
18.9.2	Solid Particles on Solid Surfaces	225
18.9.3	Droplet and Particle Densities and Fluxes in the Gas Phase	225
18.9.4	Coloring Particles and Droplets in Smokeview	225
18.9.5	Detailed Properties of Solid Particles	226
18.10	Special Output Quantities	227
18.10.1	Heat Release Rate	227
18.10.2	Visibility and Obscuration	227
18.10.3	Layer Height and the Average Upper and Lower Layer Temperatures	229
18.10.4	Thermocouples	229
18.10.5	Heat Flux	230
18.10.6	Adiabatic Surface Temperature	231
18.10.7	Special Topic: Detailed Spray Properties	232
18.10.8	Useful Solid Phase Outputs	234
18.10.9	Fractional Effective Dose (FED) and Fractional Irritant Concentration (FIC)	235
18.10.10	Spatially-Integrated Outputs	236
18.10.11	Temporally-Integrated Outputs	240
18.10.12	Statistical Outputs	240
18.10.13	Wind and the Pressure Coefficient	241
18.10.14	Near-wall Grid Resolution	241
18.10.15	Dry Volume and Mass Fractions	242
18.10.16	Aerosol and Soot Concentration	242
18.10.17	Gas Velocity	242
18.10.18	Enthalpy	243
18.10.19	Computer Performance	243
18.10.20	Output File Precision	244
18.10.21	<i>A Posteriori</i> Mesh Quality Metrics	244
18.10.22	Extinction	248
18.11	Extracting Numbers from the Output Data Files	248
18.12	Summary of Frequently-Used Output Quantities	250
18.13	Summary of Infrequently-Used Output Quantities	254
18.14	Summary of HVAC Output Quantities	256
19	Alphabetical List of Input Parameters	257
19.1	BNDF (Boundary File Parameters)	258
19.2	CLIP (Clipping Parameters)	258
19.3	CSVF (Comma Separated Velocity Files)	258
19.4	CTRL (Control Function Parameters)	258
19.5	DEVC (Device Parameters)	259
19.6	DUMP (Output Parameters)	260
19.7	HEAD (Header Parameters)	261
19.8	HOLE (Obstruction Cutout Parameters)	262
19.9	HVAC (HVAC System Definition)	262
19.10	INIT (Initial Conditions)	263
19.11	ISOF (Isosurface Parameters)	264

19.12	MATL (Material Properties)	264
19.13	MESH (Mesh Parameters)	265
19.14	MISC (Miscellaneous Parameters)	266
19.15	MULT (Multiplier Function Parameters)	268
19.16	OBST (Obstruction Parameters)	268
19.17	PART (Lagrangian Particles/Droplets)	269
19.18	PRES (Pressure Solver Parameters)	270
19.19	PROF (Wall Profile Parameters)	271
19.20	PROP (Device Properties)	271
19.21	RADI (Radiation Parameters)	272
19.22	RAMP (Ramp Function Parameters)	273
19.23	REAC (Reaction Parameters)	273
19.24	SLCF (Slice File Parameters)	274
19.25	SPEC (Species Parameters)	275
19.26	SURF (Surface Properties)	276
19.27	TABL (Table Parameters)	278
19.28	TIME (Time Parameters)	278
19.29	TRNX, TRNY, TRNZ (MESH Transformations)	278
19.30	VENT (Vent Parameters)	279
19.31	ZONE (Pressure Zone Parameters)	279
III	FDS and Smokeview Development Tools	281
20	The FDS/Smokeview Repository	283
21	Compiling FDS	285
21.1	FDS Source Code	285
22	Output File Formats	287
22.1	Diagnostic Output	287
22.2	Heat Release Rate and Related Quantities	288
22.3	Device Output Data	288
22.4	Control Output Data	288
22.5	CPU Usage Data	289
22.6	Gas Mass Data	289
22.7	Slice Files	289
22.8	Plot3D Data	290
22.9	Boundary Files	290
22.10	Particle Data	291
22.11	Profile Files	292
22.12	3-D Smoke Files	292
22.13	Geometry, Isosurface Files	293
22.14	Geometry Data Files	294

List of Figures

3.1	OpenMP timing study	15
3.2	MPI scaling study	20
6.1	An example of a multiple-mesh geometry.	36
6.2	Rules governing the alignment of meshes	38
6.3	Piecewise-linear mesh transformation	39
6.4	Polynomial mesh transformation	39
6.5	Snapshot of the <code>helium_2d_isothermal</code> test case	46
6.6	Results of the <code>duct_flow</code> test case	52
6.7	Results of the <code>dancing_eddies</code> test cases	53
6.8	Reduction of pressure iterations in the <code>dancing_eddies</code> test cases	54
6.9	Convergence test for the <code>tunnel_demo</code> test case	55
7.1	Results of the <code>circular_burner</code> test case	63
7.2	An example of the multiplier function	67
8.1	Simple demonstration of the pyrolysis model	83
8.2	Results of the <code>couch</code> test case	85
8.3	A more complicated demonstration of the pyrolysis model	86
8.4	Results of the <code>water_ice_water</code> test case	87
8.5	Results of the <code>box_burn_away</code> test cases	91
8.6	Sample results of a <code>tga_analysis</code>	94
9.1	Results of the <code>volume_flow</code> test cases	96
9.2	The <code>tangential_velocity</code> test case	97
9.3	Synthetic Eddy Method vent profiles	100
9.4	An example of simplifying a complex duct	104
9.5	Example of fan curves	107
9.6	Example of a jet fan	108
9.7	Results of the <code>HVAC_aircoil</code> case	110
9.8	Results of the <code>pressure_rise</code> test case	112
9.9	Results of the <code>zone_break</code> test cases	113
9.10	Results of the <code>zone_shape</code> test case	114
9.11	Results of the <code>door_crack</code> test case	117
9.12	Snapshots of the <code>pressure_boundary</code> test case	118
9.13	Results of the <code>parabolic_profile</code> test case	119
9.14	Boundary layer profile	120
10.1	Results of the <code>stack_effect</code> test case	123

10.2	Results of the <code>wind_example</code> test case	124
12.1	Results of the <code>gas_filling</code> test case	132
13.1	Results of the <code>pvc_combustion</code> test case	150
13.2	HRR for <code>energy_budget_adiabatic_two_fuels</code> test case	152
13.3	Wall soot deposition for the <code>propane_flame_deposition</code> test case	156
14.1	Results of the <code>ramp_chi_r</code> test case	159
15.1	Droplet size distributions	164
15.2	Results of the <code>spray_burner</code> test case	166
15.3	Examples of particle splitting	169
15.4	Example of specified gas mass production from particles	170
15.5	Example of burning vegetation	172
15.6	Mass generation of firebrands in the <code>dragon_5a</code> test case	173
15.7	Results of the <code>particle_flux</code> test case	176
15.8	Results of the <code>bucket_test_3</code> case	179
15.9	Example of water cascading over solid obstructions	180
15.10	Results of the <code>e_coefficient</code> test case	181
17.1	Results of the <code>bucket_test_2</code> case	191
17.2	Results of the <code>flow_rate</code> test case	193
17.3	Results of the <code>beam_detector</code> test case	197
17.4	Results of the <code>aspiration_detector</code> test case	199
17.5	Results of the <code>control_test_2</code> case	207
17.6	Snapshots of the <code>activate_vents</code> test case	208
17.7	Example of freezing the output of a RAMP	209
18.1	Results of the <code>bucket_test_1</code> case	225
18.2	Results of the <code>bucket_test_4</code> case	226
18.3	Results of the <code>hallways</code> test case	228
18.4	Results of the <code>adiabatic_surface_temperature</code> test case	232
18.5	Examples of the measure of turbulence resolution	245
18.6	Haar mother wavelet	246
18.7	Haar wavelet transforms on four typical signals	247

List of Tables

1.1	List of changes to input parameters for FDS 6	7
1.2	List of changes to input parameters for FDS 6 (continued)	8
5.1	Namelist Group Reference Table	32
6.1	Turbulence model options	46
6.2	Flux limiter options	49
7.1	A sample of color definitions	66
11.1	Parameters used to control time-dependence	129
12.1	Optional gas and liquid species	134
14.1	Default Radiative Fraction based on FUEL	158
17.1	Suggested values for smoke detector model	195
17.2	Control function types	202
17.3	Single frame static objects	210
17.4	Dual frame static objects	211
17.4	Dual frame static objects (continued)	212
17.5	Dynamic Smokeview objects	212
17.5	Dynamic Smokeview objects (continued)	213
17.5	Dynamic Smokeview objects (continued)	214
17.6	Dynamic Smokeview objects for Lagrangian particles	214
17.6	Dynamic Smokeview objects for Lagrangian particles (continued)	215
18.1	Output quantities available for PDPA	233
18.2	Coefficients used for the computation of irritant effects of gases	236
18.3	Frequently used output quantities	251
18.4	Infrequently used output quantities	254
18.5	HVAC output quantities	256
19.1	Boundary file parameters (BNDF namelist group)	258
19.2	Clipping parameters (CLIP namelist group)	258
19.3	Comma separated velocity files (CSVF namelist group)	258
19.4	Control function parameters (CTRL namelist group)	258
19.5	Device parameters (DEVC namelist group)	259
19.6	Output control parameters (DUMP namelist group)	260
19.7	Header parameters (HEAD namelist group)	261
19.8	Obstruction cutout parameters (HOLE namelist group)	262

19.9	HVAC parameters (HVAC namelist group)	262
19.10	Initial conditions (INIT namelist group)	263
19.11	Isosurface parameters (ISOF namelist group)	264
19.12	Material properties (MATL namelist group)	264
19.13	Mesh parameters (MESH namelist group)	265
19.14	Miscellaneous parameters (MISC namelist group)	266
19.15	Multiplier function parameters (MULT namelist group)	268
19.16	Obstruction parameters (OBST namelist group)	268
19.17	Lagrangian particles (PART namelist group)	269
19.18	Pressure solver parameters (PRES namelist group)	270
19.19	Wall profile parameters (PROF namelist group)	271
19.20	Device properties (PROP namelist group)	271
19.21	Radiation parameters (RADI namelist group)	272
19.22	Ramp function parameters (RAMP namelist group)	273
19.23	Reaction parameters (REAC namelist group)	273
19.24	Slice file parameters (SLCF namelist group)	274
19.25	Species parameters (SPEC namelist group)	275
19.26	Surface properties (SURF namelist group)	276
19.27	Table parameters (TABL namelist group)	278
19.28	Time parameters (TIME namelist group)	278
19.29	MESH transformation parameters (TRN* namelist groups)	279
19.30	Vent parameters (VENT namelist group)	279
19.31	Pressure zone parameters (ZONE namelist group)	280
21.1	FDS source code files	286

Part I

The Basics of FDS

Chapter 1

Introduction

The software described in this document, Fire Dynamics Simulator (FDS), is a computational fluid dynamics (CFD) model of fire-driven fluid flow. FDS solves numerically a form of the Navier-Stokes equations appropriate for low-speed ($Ma < 0.3$), thermally-driven flow with an emphasis on smoke and heat transport from fires. The formulation of the equations and the numerical algorithm are contained in the FDS Technical Reference Guide [?]. Verification and Validation of the model are discussed in the FDS Verification [?] and Validation [?] Guides.

Smokeyview is a separate visualization program that is used to display the results of an FDS simulation. A detailed description of Smokeyview is found in a separate User's Guide [?].

1.1 Features of FDS

The first version of FDS was publicly released in February 2000. To date, about half of the applications of the model have been for design of smoke handling systems and sprinkler/detector activation studies. The other half consist of residential and industrial fire reconstructions. Throughout its development, FDS has been aimed at solving practical fire problems in fire protection engineering, while at the same time providing a tool to study fundamental fire dynamics and combustion.

Hydrodynamic Model FDS solves numerically a form of the Navier-Stokes equations appropriate for low-speed, thermally-driven flow with an emphasis on smoke and heat transport from fires. The core algorithm is an explicit predictor-corrector scheme, second order accurate in space and time. Turbulence is treated by means of Large Eddy Simulation (LES). It is possible to perform a Direct Numerical Simulation (DNS) if the underlying numerical mesh is fine enough. LES is the default mode of operation.

Combustion Model For most applications, FDS uses a single step, mixing-controlled chemical reaction which uses three lumped species (a species representing a group of species). These lumped species are air, fuel, and products. By default the last two lumped species are explicitly computed. Options are available to include multiple reactions and reactions that are not necessarily mixing-controlled.

Radiation Transport Radiative heat transfer is included in the model via the solution of the radiation transport equation for a gray gas, and in some limited cases using a wide band model. The equation is solved using a technique similar to finite volume methods for convective transport, thus the name given to it is the Finite Volume Method (FVM). Using approximately 100 discrete angles, the finite volume solver requires about 20 % of the total CPU time of a calculation, a modest cost given the complexity of radiation heat transfer. The absorption coefficients of the gas-soot mixtures are computed using the RadCal narrow-band model [?]. Liquid droplets can absorb and scatter thermal radiation. This is important in

cases involving mist sprinklers, but also plays a role in all sprinkler cases. The absorption and scattering coefficients are based on Mie theory.

Geometry FDS approximates the governing equations on a rectilinear mesh. Rectangular obstructions are forced to conform with the underlying mesh.

Multiple Meshes This is a term used to describe the use of more than one rectangular mesh in a calculation. It is possible to prescribe more than one rectangular mesh to handle cases where the computational domain is not easily embedded within a single mesh.

Parallel Processing FDS employs OpenMP [?], a programming interface that exploits multiple processing units on a single computer. For clusters of computers, FDS employs Message Passing Interface (MPI) [?]. Details can be found in Section 3.2.2.

Boundary Conditions All solid surfaces are assigned thermal boundary conditions, plus information about the burning behavior of the material. Heat and mass transfer to and from solid surfaces is usually handled with empirical correlations, although it is possible to compute directly the heat and mass transfer when performing a Direct Numerical Simulation (DNS).

1.2 What's New in FDS 6?

Many of the changes in FDS 6 are improvements to the various sub-models that do not affect the basic structure or parameters of the input file. Most of the changes listed below do not require additional input parameters beyond those used in FDS 5.

Hydrodynamics and Turbulence

- Conservative, total variation diminishing (TVD) scalar transport is implemented: Superbee (LES default) and CHARM (DNS default). These schemes prevent over-shoots and under-shoots in species concentrations and temperature.
- Improved models for the turbulent viscosity are implemented: Deardorff (default), Dynamic Smagorinsky, and Vreman. These models provide more dynamic range to the flow field for coarse resolution and converge to the correct solution at fine resolution.
- The conservative form of the sensible enthalpy equation is satisfied by construction in the FDS 6 formulation, eliminating temperature anomalies and energy conservation errors due to numerical mixing.
- The baroclinic torque is included by default.
- Improvements are made to the wall functions for momentum and heat flux. An optional wall heat flux model accounts for variable Prandtl number fluids.
- Jarrin's Synthetic Eddy Method (SEM) is implemented for turbulent boundary conditions at vents.

Species and Combustion

- Custom species mixtures ("lumped species") can be defined with the input group SPEC.

- Turbulent combustion is handled with a new partially-stirred batch reactor model. At the subgrid level, species exist in one of two states: unmixed or mixed. The degree of mixing evolves over the FDS time step by the interaction by exchange with the mean (IEM) mixing model. Chemical kinetics may be considered infinitely fast or obey an Arrhenius rate law.
- It is now possible to transport, produce, and consume product species such as CO and soot. Chemical mechanisms must be provided by the user and may include reversible reactions.
- It is now possible to deposit aerosol species onto surfaces.
- There are an increased number of predefined species that now include liquid properties.

Lagrangian Particles

- The functionality of Lagrangian particles has expanded to include the same heat transfer and pyrolysis models that apply to solid walls. In other words, you can now assign a set of surface properties to planar, cylindrical, or spherical particles much like you would for a solid surface.
- More alternatives and user-defined option are available for the liquid droplet size distribution.
- You can specify the radiative properties of the liquid droplets.
- Drag effects of thin porous media (i.e., window screens) can be simulated using planes of particles.

Solid Phase Heat Transfer and Pyrolysis

- The basic 1-D heat transfer and pyrolysis model for solid surfaces remains the same, but there has been a change in several of the input parameters to expand functionality and readability of the input file.
- The pyrolysis model allows for the surface to shrink or swell, based on the specified material densities.

HVAC

- Filters, louvered vents, and heating/cooling capability has been added for HVAC systems.
- HVAC is now functional with MPI.

Radiation

- RadCal database has been extended to include additional fuel species.
- In cells with heat release, the emission term is based on a corrected σT^4 such that when this term is integrated over the flame volume the specified radiative fraction (default 0.35) is recovered. This differs from FDS 5 and earlier where the radiative fraction times the heat release rate was applied locally as the emission term.

Multi-Mesh Computations

- By default, FDS now iterates pressure and velocity at mesh and solid boundaries. You can control the error tolerance and maximum number of iterations via parameters on the `PRES` line.

Control Functions

- CTRL functions have been extended to include math operations.
- The evaluation of RAMPs and DEVCS can be stopped, freezing their value, based upon the activation of a device or control function.

Devices and Output

- Multiple pipe networks can be specified for sprinklers for reduction of flow rate based on the number of operating heads.
- The numerical value of a control function can be output with a DEVC.
- A line of devices can be specified using a number of POINTS on one DEVC line.
- Statistical outputs for RMS, covariance, and correlation coefficient are available.

1.3 Changes to Input Parameters in FDS 6

This section describes the changes in the input parameters between FDS version 5 and version 6. Table 1.1 lists in alphabetical order parameters from FDS 5 that have changed. Note that this table does not list new parameters in FDS 6.

There has been a limited amount of backward compatibility programmed into FDS 6. In other words, several commonly used parameters and conventions from previous versions still work, but you are encouraged to gradually modify your input files to conform to the new conventions. Gradually, obsolescent features will be removed. Some of the more notable changes in FDS 6 are:

- If you want to model a fire, you *must* include a REAC line with a specified FUEL. See Chapter 13 for details.
- The output quantity 'MIXTURE_FRACTION' has been replaced with 'MIXTURE FRACTION' and is only usable if there is a single REAC input of the form $A + B \rightarrow C$ and INITIAL_UNMIXED_FRACTION=0.
- There is no longer a STATE_FILE because there is no longer a simple mixture fraction model.
- PRESSURE_CORRECTION has been eliminated. See Section 6.6 for ways to improve the performance of the pressure solver.
- Species mass and volume fraction outputs are no longer invoked using QUANTITY='species name'. Use QUANTITY='MASS FRACTION' or QUANTITY='VOLUME FRACTION' along with SPEC_ID instead. Also note that all predefined species (Table 12.1) are now referenced with all uppercase letters.
- The output quantity 'SOOT VOLUME FRACTION' is now 'AEROSOL VOLUME FRACTION' along with SPEC_ID to identify the name of the species.

Table 1.1: Changes to input parameters, FDS version 5 to 6.

Namelist	FDS 5 Parameter	Namelist	FDS 6 Parameter	Notes
BNDF	RECOUNT_DRIP		Eliminated	
CLIP	MAXIMUM_MASS_FRACTION		Eliminated	Section 6.7
CLIP	MINIMUM_MASS_FRACTION		Eliminated	Section 6.7
DUMP	MAXIMUM_DROPLETS	DUMP	MAXIMUM_PARTICLES	Section 18.1
DUMP	STATE_FILE		Eliminated	
INIT	NUMBER_INITIAL_DROPLETS	INIT	N_PARTICLES, N_PARTICLES_PER_CELL	Section 15.5.3
MATL	NU_FUEL	MATL	NU_SPEC + SPEC_ID	Section 8.5
MATL	NU_GAS	MATL	NU_SPEC + SPEC_ID	Section 8.5
MATL	NU_RESIDUE	MATL	NU_MATL	Section 8.5
MATL	NU_WATER	MATL	NU_SPEC + SPEC_ID	Section 8.5
MATL	RESIDUE	MATL	MATL_ID	Section 8.5
MISC	BACKGROUND_SPECIES	SPEC	BACKGROUND=.TRUE.	Section 12
MISC	CONDUCTIVITY	SPEC	CONDUCTIVITY	Section 12
MISC	CO_PRODUCTION		New procedure	Section 13.3
MISC	CSMAG	MISC	C_SMAGORINSKY	Section 6.4.8
MISC	MW	SPEC	MW	Section 12
MISC	PRESSURE_CORRECTION	PRES	VELOCITY_TOLERANCE	Section 6.6
MISC	RADIATION	RADI	RADIATION	Section 14.1
MISC	EVAC_SURF_DEFAULT	SURF	EVAC_DEFAULT	Section 7.1
MISC	SURF_DEFAULT	SURF	DEFAULT	Section 7.1
MISC	VISCOSITY	SPEC	VISCOSITY	Section 12
OBST	SAWTOOTH	SURF	Eliminated	Section 9.1.7
PART	FUEL	PART	SPEC_ID=' [FUEL] '	Section 15.3.5
PART	HEAT_OF_VAPORIZATION	SPEC	HEAT_OF_VAPORIZATION	Section 15.3.1
PART	H_V_REFERENCE_TEMPERATURE	SPEC	H_V_REFERENCE_TEMPERATURE	Section 15.3.1
PART	MELTING_TEMPERATURE	SPEC	MELTING_TEMPERATURE	Section 15.3.1
PART	NUMBER_INITIAL_DROPLETS	INIT	N_PARTICLES	Section 15.5.3
PART	PARTICLES_PER_SECOND	PROP	PARTICLES_PER_SECOND	Section 15.5.2

Table 1.2: Changes to input parameters, FDS version 5 to 6 (continued).

Paramlist	FDS 5 Parameter	Paramlist	FDS 6 Parameter	Notes
PART	SPECIFIC_HEAT	SPEC	SPECIFIC_HEAT_LIQUID	Section 15.3.1
PART	VAPORIZATION_TEMPERATURE	SPEC	VAPORIZATION_TEMPERATURE	Section 15.3.1
PART	WATER	PART	SPEC_ID='WATER VAPOR'	Section 15.1
PROP	CABLE_DIAMETER		New procedure	Section 15.4.9
PROP	CABLE_FAILURE_TEMPERATURE		New procedure	Section 15.4.9
PROP	CABLE_JACKET_THICKNESS		New procedure	Section 15.4.9
PROP	CABLE_MASS_PER_LENGTH		New procedure	Section 15.4.9
PROP	CONDUIT_DIAMETER		New procedure	Section 15.4.9
PROP	CONDUIT_THICKNESS		New procedure	Section 15.4.9
PROP	DROPLETS_PER_SECOND	PROP	PARTICLES_PER_SECOND	Section 15.5.2
PROP	DROPLET_VELOCITY	PROP	PARTICLE_VELOCITY	Section 17.3.1
PROP	DT_INSERT		New procedure	Section 15.5
RADI	CH4_BANDS		Eliminated	
RADI	RADIATIVE_FRACTION	REAC	RADIATIVE_FRACTION	Section 14.1.1
REAC	BOF	REAC	A	Section 13.3
REAC	ID	REAC	FUEL	Section 13.1.1
REAC	MASS_EXTINCTION_COEFFICIENT	SPEC	MASS_EXTINCTION_COEFFICIENT	Section 18.10.2
REAC	MAXIMUM_VISIBILITY	MISC	MAXIMUM_VISIBILITY	Section 18.10.2
REAC	OXIDIZER		New procedure	Section 13.2
REAC	VISIBILITY_FACTOR	MISC	VISIBILITY_FACTOR	Section 18.10.2
SPEC	ABSORBING	SPEC	RADCAL_ID	Section 12.1.2
SURF	H_FIXED	SURF	HEAT_TRANSFER_COEFFICIENT	Section 8.2.2
SURF	POROUS		New procedure	Section 9.2.4
SURF	VOLUME_FLUX	SURF	VOLUME_FLOW	Section 9.1.6
TIME	TWEIN	TIME	T_END	Section 6.2
VENT	MASS_FRACTION		Eliminated	

1.4 A Note on Longer Run Times in FDS 6

A number of changes made in FDS 6 are aimed at improving the robustness and accuracy of the simulations. However, these improvements come at increased cost in both CPU time and memory usage. Some of this increased cost is offset by increasingly faster computers and improved parallel processing. In particular, starting with FDS 6.1.0, the default released version of FDS will employ OpenMP [?] by default. OpenMP is a programming interface that enables FDS to exploit multiple processing units on a given computer. Most Windows-based personal computers now come with multi-core processors, but past versions of FDS could only exploit a single core for a given calculation. With this new release and the increasingly faster processors available on the market, FDS 6 ought to maintain and eventually surpass the computing speed of past versions.

Listed below are suggested ways to decrease CPU time, but these options should be considered very carefully. The default parameter settings are designed to address a wide range of fire scenarios, but there are scenarios for which approximations used in past versions of FDS may still be appropriate. The best way to determine if one or more of these time-saving assumptions is appropriate, run identical simulations with and without the assumption to determine if the difference in results is acceptable.

1. The improved turbulence model in FDS 6 has been found to produce comparable results to older versions of FDS using slightly less refined numerical grids. Section 6.3.6 introduces a dimensionless parameter, $D^*/\delta x$, that indicates the number of grid cells of dimension δx that span the characteristic width of the fire, D^* . A grid resolution study should be performed to determine the loss of accuracy caused by a reduced value of $D^*/\delta x$.
2. One reason for the increased CPU cost of FDS 6 is the more precise treatment of gas species properties. Previous versions of FDS assumed that the specific heat of a gas species is solely dependent on its molecular weight, and that the ratio of specific heats, c_p/c_v , is equal to 1.4, a value appropriate for a diatomic gas like nitrogen, N_2 . Section 12.1.2 provides more details. FDS 6 now assumes that gas species are temperature-dependent, and this assumption increases the cost of the calculation in a number of different routines, in particular the calculation of the divergence. If you set `CONSTANT_SPECIFIC_HEAT_RATIO=.TRUE.` together with `STRATIFICATION=.FALSE.` on the `MISC` line, you can once again assume that the gas species are all diatomic. For scenarios where the overall compartment temperature does not approach flashover conditions, this assumption might be appropriate.
3. In situations where you are simulating a relatively small fire in a relatively large space and you are not interested in heat fluxes to surrounding structures, it might be reasonable to turn off the radiation transport calculation by setting `RADIATION` equal to `.FALSE.` on the `RADI` line. FDS will still assume that a fixed fraction of the fire's energy is radiated away, only now the energy is simply removed from the calculation.
4. In situations where the heat transfer conditions are stationary or change only gradually, you can reduce the cost of the radiation solution by reducing the temporal resolution. More details in Section 14.1.2. A sensitivity study should be performed to determine the loss of accuracy.

Chapter 2

Getting Started

FDS is a computer program that solves equations that describe the evolution of fire. It is a Fortran program that reads input parameters from a text file, computes a numerical solution to the governing equations, and writes user-specified output data to files. Smokeview is a companion program that reads FDS output files and produces animations on the computer screen. Smokeview has a simple menu-driven interface. FDS does not. However, there are various third-party programs that have been developed to generate the text file containing the input parameters needed by FDS.

This guide describes how to obtain FDS and Smokeview and how to use FDS. A separate document [?] describes how to use Smokeview.

2.1 How to Acquire FDS and Smokeview

Detailed instructions on how to download executables, manuals, source-code and related utilities, can be found at the project home page:

<https://pages.nist.gov/fds-smv/>

The typical FDS/Smokeview distribution consists of an installation package or compressed archive, which is available for MS Windows, Mac OS X, and Linux.

If you ever want to keep an older version of FDS and Smokeview, copy the installation directory to some other place so that it is not overwritten during the updated installation.

2.2 Computer Hardware Requirements

The only hard requirement to run the compiled versions of FDS and Smokeview is a 64 bit Windows, Linux, or Mac OS X operating system. The single computer or compute cluster ought to have fast processors (CPUs), and at least 4 GB RAM per processor. The CPU speed will determine how long the computation will take to finish, while the amount of RAM will determine how many mesh cells can be held in memory. A large hard drive is required to store the output of the calculations. It is not unusual for the output of a single calculation to consume more than 1 GB of storage space.

Most computers purchased within the past few years are adequate for running Smokeview with the caveat that additional memory (RAM) should be purchased to bring the memory size up to at least 2 GB. This is so the computer can display results without “swapping” to disk. For Smokeview it is also important to obtain a fast graphics card for the PC used to display the results of the FDS computations.

The MPI version of FDS requires shared disk access to each computer where cases will be run. On Windows systems this involves a domain network with the ability to share folders. On a Linux or Mac OS X system this involves NFS cross mounted files systems with ssh keys setup for passwordless login. For Multi-Mesh calculations, the MPI version of FDS can operate over standard 100 Mb/s networks. A gigabit (1000 Mb/s) network will further reduce network communication times improving data transfer rates between instances of MPI FDS running the parallel cases.

2.3 Computer Operating System (OS) and Software Requirements

The goal of making FDS and Smokeview publicly available has been to enable practicing engineers to perform fairly sophisticated simulations at a reasonable cost. Thus, FDS and Smokeview have been designed for computers running Microsoft Windows, Mac OS X, and various implementations of Unix/Linux.

MS Windows An installation package is available for Windows operating system. It is not recommended to run FDS/Smokeview under any version of MS Windows released prior to Windows 7.

Mac OS X Pre-compiled executables are installed into a user selected directory using an installation script. Mac OS X 10.4.x or better is recommended. You can always download the latest version of FDS source and compile FDS for other versions of OS X (See Appendix 21 for details).

Linux Pre-compiled executables are installed into a user selected directory using an installation script. If the pre-compiled FDS executable does not work (usually because of library incompatibilities), the FDS Fortran source code can be downloaded and compiled (See Appendix 21 for details). If Smokeview does not work on the Linux workstation, you can use the Windows version to view FDS output.

Unix There are no pre-compiled versions of FDS for the various flavors of Unix. However, the advice for Linux applies equally as well to Unix.

2.4 Installation Testing

If you are running FDS under a quality assurance plan that requires installation testing, a test procedure is provided in Appendix B of the FDS Verification Guide [?]. This guide can be obtained from the FDS-SMV website.

Chapter 3

Running FDS

Each FDS simulation is controlled by a single text-based input file, typically given a name that helps identify the particular case, and ending with the file extension `.fds`. This input file can be written directly with a text editor or with the help of a third-party graphical user interface (GUI). The simulation is started directly via the command prompt or through the GUI. The creation of an input file is covered in detail in Part II. This chapter describes how the simulation is run once the input file is written.

If you are new to FDS and Smokeview, it is strongly suggested that you start with an existing input file, run it as is, and then make the appropriate changes to the file for your desired scenario. By running a sample case, you become familiar with the procedure, learn how to use Smokeview, and ensure that your computer is up to the task before embarking on learning how to create new input files.

Sample input files are included as part of the standard installation. A good case for a first time user is located in the subfolder called `Fires` within the folder called `Examples`. Find the file called `simple_test.fds` and copy it to a folder on your computer that is not within the installation folder. The reason for doing this is to avoid cluttering up the installation folder with a lot of output files. Follow the instructions in Section 3.2.1 to run this simple single mesh case. The simulation should only take a few minutes. Once the simulation is completed, use Smokeview to examine the output. In this way, you will quickly learn the basics of running and analyzing simulations.

3.1 A Brief Primer on Computer Hardware

The following brief definitions were copied from the Indiana University Knowledge Base, which was developed with support from (U.S) National Science Foundation grant OCI-1053575.

Cores: Recent developments in computational architecture can lead to confusion concerning what a micro-processor is. Since the advent of multi-core technology, such as dual-cores and quad-cores, the term “processor” has been used to describe a logical execution unit or a physical chip. A multi-core chip may have several cores. With the advent of multi-core technology, the term “processor” has become context-sensitive, and is largely ambiguous when describing large multi-core systems. Essentially a core comprises a logical execution unit containing an L1 cache and functional units. Cores are able to independently execute programs or threads. Supercomputers are listed as having thousands of cores.

Chips: A chip refers to a physical integrated circuit (IC) on a computer. A chip in the context of this document refers to an execution unit that can be single- or multi-core technology.

Sockets: The socket refers to a physical connector on a computer motherboard that accepts a single physical chip. Many motherboards can have multiple sockets that can in turn accept multi-core chips.

Processes: A process is an independent program running on a computer. A process has a full stack of memory associated for its own use, and does not depend on another process for execution. MPI (Message Passing Interface) processes are true processes because they can run on independent machines or the same machine.

Threads: A thread is essentially a process that does not have a full stack of memory associated for it. The thread is tied to a parent process, and is merely an offshoot of execution. Typically thread processes must run on the same computer, but can execute simultaneously on separate cores of the same node. OpenMP parallelism uses threads for child processes.

Hyper-threading: Hyper-threading is an Intel technology that originally preceded multi-core systems, and was used to make a single core appear logically as multiple cores on the same chip. Intel abandoned hyper-threading briefly during the advent of multi-core processors but reintroduced the technology in 2008. Since then, Intel has used it extensively to improve the performance of parallel computations in its multi-core processors. Hyper-threading improves performance by sharing the computational workload between multiple cores whenever possible, allowing the operating system to schedule more than one process at a time.

N-ways: Multi-core compute nodes can be described by the number of execution units, or cores. A computer with 8 cores would be described as an 8-way node. This machine can have 8 independent processes running simultaneously. A 32-core system would be called a 32-way node.

Processors: As explained above, a processor could describe either a single execution core or a single physical multi-core chip. The context of use will define the meaning of the term.

3.2 Starting an FDS Calculation

FDS can be run on a single computer, using one or more cores, or it can be run on multiple computers. Starting with FDS version 6.2.0, for each supported operating system (Windows, Linux, Mac OS X) there is a single executable file called `fds` (with an `.exe` file extension on Windows). Previous releases of FDS contained two executables, one that ran on a single processor and one that ran on multiple processors. Starting with FDS 6.2.0, these two executables have been combined into one, and it can run either in serial or parallel mode.

There are two ways that FDS can be run in parallel; that is, exploit multiple cores on a single computer or multiple processors/cores distributed over multiple computers on a network or compute cluster. The first way is OpenMP (Open Multi-Processing) [?] which allows a single computer to run a single or multiple mesh FDS simulation on multiple cores. The use of OpenMP does not require the computational domain to be broken up into multiple meshes, and it will still work with cases that have multiple meshes defined. The second way to run FDS in parallel is by way of MPI (Message Passing Interface). Here, the computational domain must be divided into multiple meshes and typically each mesh is assigned its own *process*. These processes can be limited to a single computer, or they can be distributed over a network.

MPI and OpenMP can also be used together. For example, 4 MPI processes can be assigned to 4 different computers, and each MPI process can be supported by, say, 8 OpenMP threads, assuming each computer has 8 cores. Most of the speed up is achieved by the MPI. For a reasonably fast network, you can expect 4 MPI processes to speed up the computation time by a factor of about 0.9 times 4. The OpenMP can provide an extra factor up to about 2, regardless of the number of cores used beyond about 4.

3.2.1 Single or Multiple Mesh Simulation with OpenMP Parallel Processing

If your simulation involves only one mesh, you can only run it on one computer, but you can exploit its multiple processors or cores using OpenMP. When you install FDS, it will query your computer to determine the number of available cores. By default, FDS will use approximately half of the available cores¹ on a single computer. This is done for two reasons: (1) so as not to take over your entire machine when you run a simulation, and (2) because using all cores for a single simulation may not minimize the run time. OpenMP works best when exploiting multiple (logical) cores associated with a single (physical) processor or “socket”. For example, if your computer has two processors, each with 4 cores, it may not be worthwhile to use all 8 cores in an OpenMP simulation. You need to experiment with your own machine to determine the strategy that is best for you. To change the number of cores that are available for a given FDS simulation, you can set an environment variable called `OMP_NUM_THREADS`. The way to do this depends on the operating system and will be explained below.

When the job is started, FDS will print the number of cores that will be used for that job. Note that this setting only applies until you log out or restart your machine. To set the default value of available cores upon startup, the `OMP_NUM_THREADS` environment variable can also be set in the startup configuration scripts on the machine. Refer to the documentation for your operating system for more information on how to configure environment variables upon startup.

To confirm the speedup for the OpenMP version of the code, a series of test cases are run for two problem sizes (64^3 and 128^3) varying the number of OpenMP threads. The setup is a simple channel flow carrying two extra species to mimic the scalar transport performed in typical fire problems (see the `Timing_Benchmarks/openmp_test` series in the FDS verification suite). The results are shown in Fig. 3.1. Generally, users can expect a factor of 2 speedup using 4 cores (default setting).

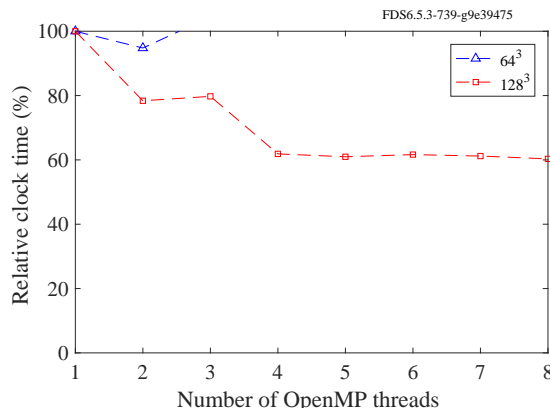


Figure 3.1: Benchmark timing comparison for the OpenMP test cases. The computer that ran these jobs has 2 (physical) sockets, and each socket has 4 (logical) cores. This explains the decrease in efficiency beyond 4 OpenMP threads.

Details of how to run the simulation are included below for the different operating systems as follows:

¹To determine the number of cores used by OpenMP, just type `fds` at the command prompt.

MS Windows

Open up a Command Prompt window (click Start, then Run, then type “cmd”), and change directories (“cd”) to where the input file for the case is located. Decide how many cores you want to devote to the simulation. For example, if you have 4 cores available, type the following at the command prompt:

```
set OMP_NUM_THREADS=4
```

Then run the simulation by typing:

```
fds job_name.fds
```

The progress of the simulation is indicated by diagnostic output that is written out onto the screen. Detailed diagnostic information is automatically written to a file `job_name.out`. Screen output can be redirected to a file via the alternative command:

```
fds job_name.fds > job_name.err
```

Note that it is also possible to associate the `.fds` extension with the FDS executable directly, thereby making FDS run by double-clicking on the input file. Be careful not to accidentally double-click on the input file when trying to edit it. This action will cause previously generated output files to be over-written.

Mac OS X, Unix, Linux

The installer for Mac OS X, Unix, and Linux versions of FDS sets the `PATH` variable allowing one to invoke FDS without a full path reference to the executable. To specify 4 OpenMP threads on a Mac OS X or Linux computer running a Bash shell, type the following at the command prompt:

```
export OMP_NUM_THREADS=4
```

To run FDS from the command line type:

```
fds job_name.fds
```

The input parameters are read from the file `job_name.fds`, and error statements and other diagnostics are written out to the screen. To run the job in the background:

```
fds job_name.fds >& job_name.err &
```

Note that in the latter case, the screen output is stored in the file `job_name.err` and the detailed diagnostics are saved automatically in a file `job_name.out`. It is preferable to run jobs in the background so as to free the console for other uses. To see output while an FDS job is running type:

```
tail -f job_name.err
```

or

```
tail -f job_name.out
```

3.2.2 Multiple Mesh Simulation with MPI Parallel Processing

FDS uses MPI (Message-Passing Interface) [?] to allow multiple computers, or multiple cores on one computer, to run a single multi-mesh FDS job. The main idea is that you must break up the FDS domain into multiple meshes, and then the flow field in each mesh is computed as a different *process*. Note the subtle difference between these terms – a *process* does not have the same meaning as a *processor*. The *process* can be thought of as a “task” that you would see in the Windows Task Manager or by executing the “top” command on a Linux/Unix machine. The *processor* refers to the computer hardware. A single *processor* may run multiple *processes*, for example. The computation on a given FDS mesh is thought of as an individual *process*, and MPI handles the transfer of information between these *processes*. Usually, each mesh is assigned its own *process* in an MPI calculation, although it is also possible to assign multiple meshes to a single *process*. In this way, large meshes can be computed on dedicated *processors*, while smaller meshes can be clustered together in a single *process* running on a single *processor*, without the need for MPI message passing between themselves.

Also note that FDS refers to its meshes by the numbers 1, 2, 3, and so on, whereas MPI refers to its processes by the numbers 0, 1, 2, and so on. Thus, Mesh 1 is assigned to Process 0; Mesh 2 to Process 1, and so on. You do not explicitly number the meshes or the processes yourself, but error statements from FDS or from MPI might refer to the meshes or processes by number. As an example, if a FDS case with five meshes, the first printout (usually to the screen unless otherwise directed) is:

```
Mesh 1 is assigned to MPI Process 0
Mesh 2 is assigned to MPI Process 1
Mesh 3 is assigned to MPI Process 2
Mesh 4 is assigned to MPI Process 3
Mesh 5 is assigned to MPI Process 4
```

This means that 5 MPI processes (numbered 0 to 4) have started and that each mesh is being handled by its own process. The processes may be on the same or different computers. Each computer has its own memory (RAM), but each individual MPI process has its own independent memory, even if the processes are on the same computer.

There are different implementations of MPI, much like there are different Fortran and C compilers. Each implementation is essentially a library of subroutines called from FDS that transfer data from one process to another across a fast network. The format of the subroutine calls has been widely accepted in the community, allowing different vendors and organizations the freedom to develop better software while working within an open framework. For Mac OS X and Linux, we use Open MPI, an open source implementation that is developed and maintained by a consortium of academic, research, and industry partners (www.openmpi.org). For Windows, we use Intel MPI ².

Intel MPI for Windows

The files needed to run an MPI calculation across a Windows domain network are bundled in with the FDS download. There is no need to install Intel MPI or its redistributable libraries. The following procedure is intended for a Windows domain network; that is, a network where user accounts are centrally managed such that any user can log in to any machine using the same credentials.

To run the MPI version of FDS, you need to have administrator privileges on the machine that launches the job, but not necessarily on the machines that run the job. We have designed the FDS installation script to set up the necessary firewall exceptions and system environment variables automatically so that you need

²Prior to FDS version 6.1.2, the Windows version of FDS used MPICH, a free implementation of MPI developed by Argonne National Laboratory. The MPICH developers have announced that they are no longer supporting the Windows version.

only install FDS as you normally would to make this work. You do not have to download and install the MPI software separately. Everything you need is in the FDS installation package.

First, uninstall the FDS-SMV package on all of the machines you plan to use. The new version will overwrite all existing versions unless they are moved or renamed.

If you wish to run FDS on more than one core on a single machine, then type the following command:

```
mpiexec -n 4 fds job_name.fds
```

where 4 indicates the number of processes that will be created. Note that as a default each process will use 4 OpenMP threads. It is advisable to set the environmental variable `OMP_NUM_THREADS` so that the total number of threads ($n \times \text{OMP_NUM_THREADS}$) is less than or equal to the total number of cores on the machine

If you wish to run FDS on more than one machine, the following steps should be taken:

1. Open up a command prompt by right-clicking on the command prompt icon and choosing “Run as administrator”. Type the following command:

```
mpiexec -hosts 2 <my_machine> 1 <other_machine> 1 test_mpi
```

If this command returns a “Hello World” message from your machine and the other machine on your network, proceed to the next step. If this command fails, check that you can “see” the other machine by pinging it, and check that the other machine can “see” your machine as well.

2. Share (with both read and write privilege) a working directory on your machine. Do not put this directory within the Program Files path. Share the working directory with everybody so that all other machines can see it. Note how this directory is defined on the other machines. Sometimes it is `\\<my_machine>\<my_shared_directory>` and sometimes it is defined via the numerical IP address, like `\\129.6.129.87\<my_shared_directory>`. The definition depends on the way your domain name server (DNS) works. In any case, do not leave blank spaces within any directory or file names. We have found that blanks create all sorts of trouble. Unless you are a DOS/Windows expert, avoid them.
3. Within the command prompt, `cd` to the working directory. Find or create within the working directory a relatively small, simple, two mesh FDS input file. At the command prompt, type:

```
mpiexec -hosts 2 m1 1 m2 1 -wdir \\...\\... fds job_name.fds
```

where m1 and m2 are the names of two computers on your network.

4. If successful, you should see the usual FDS printout indicating the processes being assigned to the machines. If unsuccessful, try running the case on your own machine:

```
mpiexec -hosts 1 m1 2 -wdir \\...\\... fds job_name.fds
```

If this is not successful, check with your network administrator or monitor the FDS help forums for advice.

Open MPI for Linux and Mac OS X

The release versions of FDS for Linux and Mac OSX are built with Open MPI, an open source MPI implementation that is developed and maintained by a consortium of academic, research, and industry partners. With Open MPI, FDS is run using the command:

```
mpirun -np 5 fds -hostfile my_hosts.txt job_name.fds
```

where the 5 indicates that 5 processes are to be used. In this case, the executable `fds` is located in the working directory, but you can also provide the full path name to the installation directory. The file `my_hosts.txt` might look like this:

```
comp1 slots=2
comp2 slots=1
comp3 slots=2
```

where `slots` indicate the number of available cores on that computer.

To make the process run in the background, use the command:

```
mpirun ... >& job_name.err &
```

The file `job_name.err` contains what is normally printed out to the screen.

3.2.3 Using MPI and OpenMP Together

Because it more efficiently divides the computation, MPI is the better choice for multiple mesh simulations. However, it is possible to combine MPI and OpenMP in the same simulation. If you have multiple computers at your disposal, and each computer has multiple cores, you can assign one MPI process to each computer, and use multiple cores on each computer to speed up the processing of a given mesh using OpenMP. For example, on a Windows Domain Network, if you have two computers that are available and each computer has four cores, you can run a two mesh simulation as follows:

```
mpiexec -hosts 2 m1 1 m2 1 -wdir \\...\\... -env OMP_NUM_THREADS 4 fds job\_name.fds
```

The use of OpenMP in this instance will probably speed the calculation by a factor of 2, but now you will be using 8 cores rather than 2. It might be better to divide the computational domain into 8 meshes and simply use MPI to process them. This all depends on your particular OS, hardware, network traffic, and so on. You should choose a good test case and try different meshing and parallel processing strategies to see what is best for you.

3.2.4 Efficiency of an MPI Calculation

At the end of a calculation, FDS prints out a file called `CHID_cpu.csv` that records the amount of CPU time that each MPI process spends in the major routines. For example, the column header `VELO` stands for all the subroutines related to computing the flow velocity; `MASS` stands for all the subroutines related to computing the species mass fractions and density. The column header `MAIN` represents all of the CPU time that is not explicitly accounted for; that is, time spend in the main control loop. Ideally, this ought to be a few percent of the overall CPU time usage.

There are two basic approaches to assessing the efficiency or scalability of an MPI (parallel) computation. The first is known as “weak scaling,” in which the amount of work done by each MPI process stays the same and additional processes are added to solve a larger problem. For example, if you are simulating the flow of air over a patch of terrain, and you keep adding more and more meshes of the same physical and numerical dimension, assigning each new mesh to its own MPI process, so as to simulate a larger and larger patch of terrain, then you would expect that the overall time of the simulation would not increase significantly with each additional mesh. The efficiency of such a calculation is given by the following expression:

$$E_w = \frac{t_1}{t_N} \quad (3.1)$$

where t_1 is the CPU time for the case with 1 mesh (MPI process), and t_N is the CPU time for the case with N meshes (MPI processes). The left hand plot of Fig. 3.2 shows the results of a weak scaling study of FDS. Meshes with dimension 50 by 50 by 50 are lined up side by side, ranging from 1 to 288 meshes. Ideally, the CPU time ought to be about the same for all cases, because each MPI process is doing the same amount of work. Only mesh to mesh communication should lead to inefficiencies. However, notice in the figure that the efficiency of the 1, 2, 4, and 8 mesh cases is greater than those with more MPI processes. The reason for this is that on most compute clusters, each node has multiple cores, and typically jobs run faster when a node is less than completely full. These test cases were run at NIST, where there is a compute cluster with 8 cores per node, and one with 12 cores per node.

The second way to assess MPI efficiency is known as “strong scaling.” Here, you simulate a given scenario on a single mesh, and then you divide the mesh so that the cell size and the overall number of cells does not change. Ideally, if you divide a given mesh into two and run the case with two MPI processes instead of one, you would expect your computation time to decrease by a factor of two. But as you increase the number of MPI processes, you increase the amount of communication required among the processes. You also increase the overall number of boundary cells to compute, even though the overall number of gas phase cells remains the same. The efficiency of such a set of calculations is given by:

$$E_s = \frac{t_1}{N t_N} \quad (3.2)$$

In the strong study demonstrated here, a single mesh of dimension 180 by 160 by 80 is divided into a range of smaller meshes, with the smallest partitioning being 288 meshes of dimension 20 by 20 by 20. The resulting decrease in the CPU time of the entire calculation and the major subroutines is shown in the right hand plot of Fig. 3.2. Ideally, the CPU time should be inversely proportional to the number of meshes (MPI processes); that is, the relative CPU times ought to follow the black dotted lines. The one notable exception to this rule is for “COMM” or COMMunications. This curve represents the time spent in communicating information across the network.

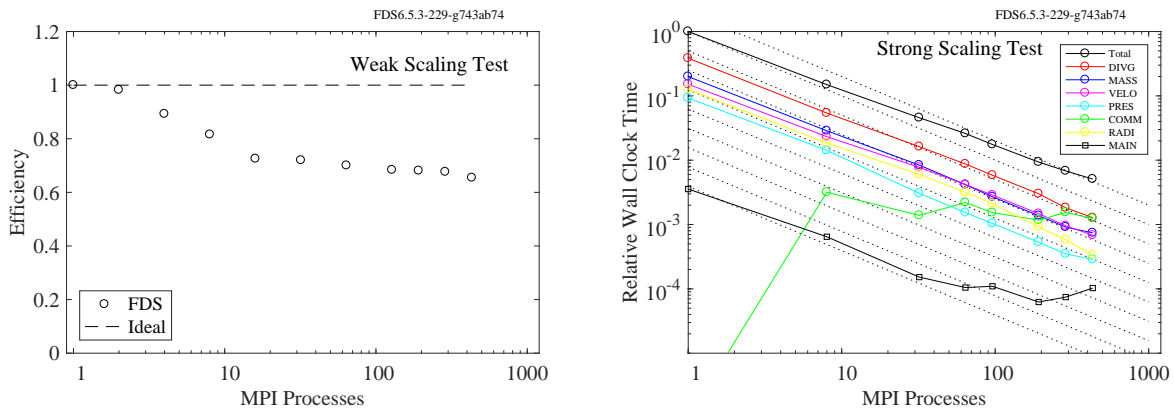


Figure 3.2: Example of a weak (left) and strong (right) scaling study.

3.2.5 Running Very Large Jobs

Most FDS simulations reported in the literature use one to several dozen meshes, and MPI is the method of choice to parallelize these jobs. Usually the meshes are mapped to MPI processes in a one to one manner and the meshes contain a comparable number of grid cells. However, it is possible to run FDS jobs that

involve thousands of meshes. In 2016, the FDS developers at NIST were given access to the Oak Ridge Leadership Computing Facility at Oak Ridge National Laboratory in Tennessee. The facility provides users access to compute clusters with very large numbers of processors connected via a high speed network. FDS simulations were performed using up to approximately 10,000 MPI processes. If you have access to facilities such as this one, here are a few pointers:

1. Use MPI only. OpenMP will probably not speed up the run time appreciably, and it will consume cores that could be put to better use running more MPI processes.
2. For jobs using thousands of meshes/processes, add the parameter `SHARED_FILE_SYSTEM=.FALSE.` to the `MISC` line. This directs FDS to break up the Smokeview (`.smv`) file according to MPI process. This will greatly speed up the preliminary part of the simulation because the Smokeview file is written serially, not in parallel. For a modest number of meshes, this serial write is not a problem, but for thousands of meshes, the initialization routines can take hours. When the job completes, you can reconstruct the Smokeview file by appending the numbered files, `CHID_n.smv`, to the main Smokeview file, `CHID.smv`.
3. Set `DT_CPU` to some convenient time interval on the `DUMP` line. This parameter directs FDS to periodically write out a file (`CHID_cpu.csv`) that records the wall clock time that each MPI process consumes in the major subroutines. This can help you determine if any of the MPI processes spend an inordinate amount of time idling.
4. Run your job for a short amount of time to estimate the time required for the full job. Most large compute clusters will limit you to a certain amount of wall clock time, after which your job is simply stopped. If you have to use the restart feature in FDS, practice first with a short job to make sure that the job can be continued properly.
5. Do a strong scaling study for your particular case. That is, run the job a fixed number of time steps with the least number of meshes that can fit within the machine's memory. Then divide the mesh by factors of 2, 4, or 8 until you reach a point where the increased number of meshes/processes does not provide a significant speed up.

3.3 Monitoring Progress

Diagnostics for a given calculation are written into a file called `CHID.out`. The current simulation time and time step is written here, so you can see how far along the program has progressed. At any time during a calculation, Smokeview can be run and the progress can be checked visually.

By default, the diagnostics in the `CHID.out` file are verbose. When running large MPI jobs it may be advantageous to quiet this output, which is all written by MPI process 0. To do this, add

```
&DUMP SUPPRESS_DIAGNOSTICS=.TRUE. /
```

Be aware the output file will not monitor mesh boundary velocity errors in this case; it will echo only the simulation time and time step. You could still output a `BNDF` of `QUANTITY='VELOCITY_ERROR'`, if necessary.

To stop a calculation before its scheduled time, either kill the process, or preferably create a file in the same directory as the output files called `CHID.stop`. The existence of this file stops the program gracefully, causing it to dump out the latest flow variables for viewing in Smokeview.

Since calculations can be hours or days long, there is a restart feature in FDS. Details of how to use this feature are given in Section 6.4.3. Briefly, specify at the beginning of calculation how often a “restart” file

should be saved. Should something happen to disrupt the calculation, like a power outage, the calculation can be restarted from the time the last restart file was saved.

It is also possible to control the stop time and the dumping of restart files by using control functions as described in [Section 17.5](#).

Chapter 4

User Support

It is not unusual over the course of a project to run into various problems, some related to FDS, some related to your computer. FDS is an CPU and memory intensive calculation that can push your computer's processor and memory to its limits. In fact, there are no hardwired bounds within FDS that prevent you from starting a calculation that is too large for your hardware. Even if your machine has adequate memory (RAM), you can still easily set up calculations that can require weeks or months to complete. It is difficult to predict at the start of a simulation just how long and how much memory will be required. Learn how to monitor the resource usage of your computer. Start with small calculations and build your way up.

Although many features in FDS are fairly mature, there are many that are not. FDS is used for practical engineering applications, but also for research in fire and combustion. As you become more familiar with the software, you will inevitably run into areas that are of current research interest. Indeed, burning a roomful of ordinary furniture is one of the most challenging applications of the model. So be patient, and learn to dissect a given scenario into its constitutive parts. For example, do not attempt to simulate a fire spreading through an entire floor of a building unless you have simulated the burning of the various combustibles with relatively small calculations.

Along with the FDS User's Guide, there are resources available on the Internet. These resources include an "Issue Tracker" for reporting bugs and requesting new features, a "Discussion Group" for clarifying questions and discussing more general topics rather than just specific problems, and "Wiki Pages" that provide supplementary information about FDS-SMV development, third-party tools, and other resources. Before using these on-line resources, it is important to first try to solve your own problems by performing simple test calculations or debugging your input file. The next few sections provide a list of error statements and suggestions on how to solve problems.

4.1 The Version Number

If you encounter problems with FDS, it is crucial that you submit, along with a description of the problem, the FDS version number. Each release of FDS comes with a version number like 5.2.6, where the first number is the *major* release, the second is the *minor* release, and the third is the *maintenance* release. Major releases occur every few years, and as the name implies significantly change the functionality of the model. Minor releases occur every few months, and may cause minor changes in functionality. Release notes can help you decide whether the changes should effect the type of applications that you typically do. Maintenance releases are just bug fixes, and should not affect code functionality. To get the version number, just type the executable at the command prompt without an input file, and the relevant information will appear, along with a date of compilation (useful to you) and a so-called Git hash tag (useful to us). The Git hash tag refers to the GitHub repository number of the source code. It allows us to go back in time and

recover the exact source code files that were used to build that executable.

Get in the habit of checking the version number of your executable, periodically checking for new releases which might already have addressed your problem, and telling us what version you are using if you report a problem.

4.2 Common Error Statements

An FDS calculation may end before the specified time limit. Following is a list of common error statements and how to diagnose the problems:

Input File Errors: The most common errors in FDS are due to mis-typed input statements. These errors result in the immediate halting of the program and a statement like, “ERROR: Problem with the HEAD line.” For these errors, check the line in the input file named in the error statement. Make sure the parameter names are spelled correctly. Make sure that a / (forward slash) is put at the end of each namelist entry. Make sure that the right type of information is being provided for each parameter, like whether one real number is expected, or several integers, or whatever. Make sure there are no non-ASCII characters being used, as can sometimes happen when text is cut and pasted from other applications or word-processing software. Make sure zeros are zeros and O’s are O’s. Make sure l’s are not !’s. Make sure apostrophes are used to designate character strings. Make sure the text file on a Unix/Linux machine was not created on a Windows machine, and *vice versa*. Make sure that all the parameters listed are still being used – new versions of FDS often drop or change parameters forcing you to re-examine old input files.

Numerical Instability Errors: It is possible that during an FDS calculation the flow velocity at some location in the domain can increase due to numerical error causing the time step size to decrease to a point¹ where logic in the code decides that the results are unphysical and stops the calculation with an error message in the file `CHID.out`. In these cases, FDS ends by dumping out one final Plot3D file giving you a hint as to where the error is occurring within the computational domain. Usually, a numerical instability can be identified by fictitiously large velocity vectors emanating from a small region within the domain. Common causes of such instabilities are:

- mesh cells that have an aspect ratio larger than 2 to 1
- high speed flow through a small opening
- a sudden change in the heat release rate
- the removal or creation of an obstruction, like the opening or closing of a door
- a high (>100 g/mol) molecular weight fuel molecule that is not in the FDS database, Table 12.1. In such cases, reduce the molecular weight but maintain the atom ratios.
- long, sealed tunnels, in which pressure fluctuations can cause spurious numerical artifacts. See Section 6.6 for details.

There are various ways to solve the problem, depending on the situation. Try to diagnose and fix the problem before reporting it. It is difficult for anyone but the originator of the input file to diagnose the problem.

¹By default, the calculation is stopped when the time step drops below 0.0001 of the initial time step. This factor can be changed via the `TIME` line by specifying the `LIMITING_DT_RATIO`.

Inadequate Computer Resources: The calculation might be using more RAM than the machine has (you will see an error message like “ERROR: Memory allocation failed for ZZ in the routine INIT”) , or the output files could have used up all the available disk space. In these situations, the computer may or may not produce an intelligible error message. Sometimes the computer is just unresponsive. It is your responsibility to ensure that the computer has adequate resources to do the calculation. Remember, there is no limit to how big or how long FDS calculations can be – it depends on the resources of the computer. For any new simulation, try running the case with a modest-sized mesh, and gradually make refinements until the computer can no longer handle it. Then back off somewhat on the size of the calculation so that the computer can comfortably run the case. Trying to run with 90 % to 100 % of computer resources is risky. In fact, for a typical 32 bit Windows PC with 4 GB RAM, only 2 GB will be available to FDS, based on user feedback. If you want to run bigger cases, consider buying a computer with a 64 bit operating system or break up the calculation into multiple meshes and use the MPI version of FDS. If you are using a Linux/Unix machine, make sure that the stacksize is unlimited, which will allow FDS to access as much of the RAM as possible. Changing the stacksize limit differs with each shell type, so it is best to do an on-line search to find out how to ensure that your stacksize is unlimited.

Run-Time Errors: An error occurs either within the computer operating system or the FDS program. An error message is printed out by the operating system of the computer onto the screen or into the diagnostic output file. This message is most often unintelligible to most people, including the programmers, although occasionally one might get a small clue if there is mention of a specific problem, like “stack overflow,” “divide by zero,” or “file write error, unit=...” Sometimes the error message simply refers to a “Segmentation Fault.” These errors may be caused by a bug in FDS, for example if a number is divided by zero, or an array is used before it is allocated, or any number of other problems. Before reporting the error to the Issue Tracker, try to systematically simplify the input file until the error goes away. This process usually brings to light some feature of the calculation responsible for the problem and helps in the debugging.

File Writing Errors: Occasionally, especially on Windows machines, FDS fails because it is not permitted to write to a file. A typical error statement reads:

```
forrtl: severe (47): write to READONLY file, unit 8598, file C:\Users\...\
```

The unit, in this case 8598, is just a number that FDS has associated with one of the output files. If this error occurs just after the start of the calculation, you can try adding the phrase

```
FLUSH_FILE_BUFFERS=.FALSE.
```

on the `DUMP` line of the input file (see Section 18.1). This will prevent FDS from attempting to flush the contents of the internal buffers, something it does to make it possible to view the FDS output in Smokeview during the FDS simulation. On some Windows machines, you might encounter security settings that prevent command line programs such as FDS from writing to system folders that contain program files. In this case, try to rerun the case in a non-system folder (i.e., a location within your home directory).

Poisson Initialization: Sometimes at the very start of a calculation, an error appears stating that there is a problem with the “Poisson initialization.” The equation for pressure in FDS is known as the Poisson equation. The Poisson solver consists of large system of linear equations that must be initialized at the start of the calculation. Most often, an error in the initialization step is due to a mesh `IJK` dimension being less than 4 (except in the case of a two-dimensional calculation). It is also possible that something is fundamentally wrong with the coordinates of the computational domain. Diagnose the problem by checking the `MESH` lines in the input file.

4.3 Support Requests and Bug Tracking

Because FDS development is on-going, problems will inevitably occur with various routines and features. The developers need to know if a certain feature is not working, and reporting problems is encouraged. However, the problem must be clearly identified. The best way to do this is to simplify the input file as much as possible so that the bug can be diagnosed (i.e., create and submit a minimal working example). Also, limit the bug reports to those features that clearly do not work. Physical problems such as fires that do not ignite, flames that do not spread, etc., may be related to the mesh resolution or scenario formulation, and you need to investigate the problem first before reporting it. If an error message originates from the operating system as opposed to FDS, first investigate some of the more obvious possibilities, such as memory size, disk space, etc.

If that does not solve the problem, report the problem with as much information about the error message and circumstances related to the problem. The input file should be simplified as much as possible so that the bug occurs early in the calculation. Attach the simplified input file if necessary, following the instructions provided at the web site. In this way, the developers can quickly run the problematic input file and hopefully diagnose the problem.

Note: Reports of specific bugs, problems, feature requests, and enhancements should be posted to the Issue Tracker and not the Discussion Group.

Part II

Writing an FDS Input File

Chapter 5

The Basic Structure of an Input File

5.1 Naming the Input File

The operation of FDS is based on a single ASCII¹ text file containing parameters organized into *namelist*² groups. The input file provides FDS with all of the necessary information to describe the scenario. The input file is saved with a name such as `job_name.fds`, where `job_name` is any character string that helps to identify the simulation. If this same string is repeated under the `HEAD` namelist group within the input file, then all of the output files associated with the calculation will then have this common prefix name.

There should be no blank spaces in the job name. Instead use the underscore character to represent a space. Using an underscore characters instead of a space also applies to the general practice of naming directories on your system.

Be aware that FDS will simply over-write the output files of a given case if its assigned name is the same. This is convenient when developing an input file because you save on disk space. Just be careful not to overwrite a calculation that you want to keep.

5.2 Namelist Formatting

Parameters are specified within the input file by using *namelist* formatted records. Each namelist record begins with the ampersand character, `&`, followed immediately by the name of the namelist group, then a comma-delimited list of the input parameters, and finally a forward slash, `/`. For example, the line

```
&DUMP NFRAMES=1800, DT_HRR=10., DT_DEVC=10., DT_PROF=30. /
```

sets various values of parameters contained in the `DUMP` namelist group. The meanings of these various parameters will be explained in subsequent chapters. The namelist records can span multiple lines in the input file, but just be sure to end the record with a slash or else the data will not be understood. Do not add anything to a namelist line other than the parameters and values appropriate for that group. Otherwise, FDS will stop immediately upon execution.

Parameters within a namelist record can be separated by either commas, spaces, or line breaks. It is recommended that you use commas or line breaks, and never use tab stops because they are not explicitly defined in the namelist data structure. Comments and notes can be written into the file so long as nothing comes before the ampersand except a space and nothing comes between the ampersand and the slash except appropriate parameters corresponding to that particular namelist group.

¹ ASCII – American Standard Code for Information Interchange. There are 256 characters that make up the standard ASCII text.

² A *namelist* is a Fortran input record.

The parameters in the input file can be integers, reals, character strings, or logical parameters. A logical parameter is either `.TRUE.` or `.FALSE.` – the periods are a Fortran convention. Character strings that are listed in this User’s Guide must be copied exactly as written – the code is case sensitive and underscores *do* matter. The maximum length of most character input parameters is 60.

Most of the input parameters are simply real or integer scalars, like `DT=0.02`, but sometimes the inputs are multidimensional arrays. For example, when describing a particular solid surface, you need to express the mass fractions of multiple materials that are to be found in multiple layers. The input array `MATL_MASS_FRACTION(IL, IC)` is intended to convey to FDS the mass fraction of component `IC` of layer `IL`. For example, if the mass fraction of the second material of the third layer is 0.5, then write

```
MATL_MASS_FRACTION(3,2)=0.5
```

To enter more than one mass fraction, use this notation:

```
MATL_MASS_FRACTION(1,1:3)=0.5,0.4,0.1
```

which means that the first three materials of layer 1 have mass fractions of 0.5, 0.4, and 0.1, respectively. The notation `1:3` means array element 1 through 3, inclusive.

Note that character strings can be enclosed either by single or double quotation marks. Be careful not to create the input file by pasting text from something other than a simple text editor, in which case the punctuation marks may not transfer properly into the text file.

Some text file encodings may not work on all systems. If file reading errors occur and no typographical errors can be found in the input file, try saving the input file using a different encoding. For example, the text file editor Notepad works fine on a Windows PC, but a file edited in Notepad may not work on Linux or Mac OS X because of the difference in line endings between Windows and Unix/Linux operating systems. The editor Wordpad typically works better, but try a simple case first.

5.3 Input File Structure

In general, the namelist records can be entered in any order in the input file, but it is a good idea to organize them in some systematic way. Typically, general information is listed near the top of the input file, and detailed information, like obstructions, devices, and so on, are listed below. FDS scans the entire input file each time it processes a particular namelist group. With some text editors, it has been noticed that the last line of the file is often not read by FDS because of the presence of an “end of file” character. To ensure that FDS reads the entire input file, add

```
&TAIL /
```

as the last line at the end of the input file. This completes the file from `&HEAD` to `&TAIL`. FDS does not even look for this last line. It just forces the “end of file” character past relevant input.

Another general rule of thumb when writing input files is to only add parameters that make change from the default value. That way, you can more easily distinguish between what you want and what FDS wants. Add comments liberally to the file, so long as these comments do not fall within the namelist records.

The general structure of an input file is shown below, with many lines of the original validation input file³ removed for clarity.

```
&HEAD CHID='WTC_05', TITLE='WTC Phase 1, Test 5' /
```

³The actual input file, `WTC_05.fds`, is part of the FDS Validation Suite

```

&MESH IJK=90,36,38, XB=-1.0,8.0,-1.8,1.8,0.0,3.82 /
&TIME T_END=5400. /
&MISC TMPA=20. /
&DUMP NFRAMES=1800, DT_HRR=10., DT_DEVC=10., DT_PROF=30. /

&REAC FUEL      = 'N-HEPTANE'
      FYI      = 'Heptane, C_7 H_16'
      C        = 7.
      H        = 16.
      CO_YIELD = 0.008
      SOOT_YIELD = 0.015 /

&OBST XB= 3.5, 4.5,-1.0, 1.0, 0.0, 0.0, SURF_ID='STEEL FLANGE' /  Fire Pan
...
&SURF ID        = 'STEEL FLANGE'
      COLOR     = 'BLACK'
      MATL_ID   = 'STEEL'
      BACKING   = 'EXPOSED'
      THICKNESS = 0.0063 /
...
&VENT MB='XMIN', SURF_ID='OPEN' /
...
&SLCF PBY=0.0, QUANTITY='TEMPERATURE', VECTOR=.TRUE. /
...
&BNDF QUANTITY='GAUGE HEAT FLUX' /
...
&DEVC XYZ=6.04,0.28,3.65, QUANTITY='VOLUME FRACTION', SPEC_ID='OXYGEN', ID='EO2_FDS' /
...
&TAIL / End of file.

```

It is recommended that when looking at a new scenario, first select a pre-written input file that resembles the case, make the necessary changes, then run the case at fairly low mesh resolution to determine if the geometry is set up correctly. It is best to start off with a relatively simple file that captures the main features of the problem without getting tied down with too much detail that might mask a fundamental flaw in the calculation. Initial calculations ought to be meshed coarsely so that the run times are less than an hour and corrections can easily be made without wasting too much time. As you learn how to write input files, you will continually run and re-run your case as you add in complexity.

Table 5.1 provides a quick reference to all the namelist parameters and where you can find the reference to where it is introduced in the document and the table containing all of the keywords for each group.

Table 5.1: Namelist Group Reference Table

Group Name	Namelist Group Description	Reference Section	Parameter Table
BNDF	Boundary File Output	18.5	19.1
CLIP	Clipping Parameters	6.7	19.2
CSVF	Velocity Input File	6.4.4	19.3
CTRL	Control Function Parameters	17.5	19.4
DEVC	Device Parameters	17.1	19.5
DUMP	Output Parameters	18.1	19.6
HEAD	Input File Header	6.1	19.7
HOLE	Obstruction Cutout	7.2.6	19.8
HVAC	Heating, Vent., Air Cond.	9.2	19.9
INIT	Initial Condition	6.5	19.10
ISOF	Isosurface File Output	18.6	19.11
MATL	Material Property	8.3	19.12
MESH	Mesh Parameters	6.3	19.13
MISC	Miscellaneous	6.4	19.14
MULT	Multiplier Parameters	7.5	19.15
OBST	Obstruction	7.2	19.16
PART	Lagrangian Particle	15	19.17
PRES	Pressure Solver Parameters	6.6	19.18
PROF	Profile Output	18.3	19.19
PROP	Device Property	17.3	19.20
RADI	Radiation	14.1	19.21
RAMP	Ramp Profile	11	19.22
REAC	Reaction Parameters	13	19.23
SLCF	Slice File Output	18.4	19.24
SPEC	Species Parameters	12	19.25
SURF	Surface Properties	7.1	19.26
TABL	Tabulated Particle Data	17.3.1	19.27
TIME	Simulation Time	6.2	19.28
TRNX	Mesh Stretching	6.3.5	19.29
VENT	Vent Parameters	7.3	19.30
ZONE	Pressure Zone Parameters	9.3	19.31

Chapter 6

Setting the Bounds of Time and Space

This chapter describes global input parameters that affect the general scope of the simulation, like the simulation time and the size and extent of the computational domain. Essentially, these parameters establish the spatial and temporal coordinate systems that are used by all other components of the simulation, which is why these parameters are usually listed at the top of the input file and why they are described here first.

6.1 Naming the Job: The `HEAD` Namelist Group (Table 19.7)

The first thing to do when setting up an input file is to give the job a name. The name of the job is important because often a project involves numerous simulations in which case the names of the individual simulations should be meaningful and help to organize the project. The namelist group `HEAD` contains two parameters, as in this example:

```
&HEAD CHID='WTC_05', TITLE='WTC Phase 1, Test 5' /
```

`CHID` is a string of 30 characters or less used to tag the output files. If, for example, `CHID='WTC_05'`, it is convenient to name the input data file `WTC_05.fds` so that the input file can be associated with the output files. No periods or spaces are allowed in `CHID` because the output files are tagged with suffixes that are meaningful to certain computer operating systems. If `CHID` is not specified, then it will be set to the name of the input file minus everything at and beyond the first period.

`TITLE` is a string of 60 characters or less that describes the simulation. It is simply a descriptive text that is passed to various output files.

6.2 Simulation Time: The `TIME` Namelist Group (Table 19.28)

`TIME` is the name of a group of parameters that define the time duration of the simulation and the initial time step used to advance the solution of the discretized equations.

6.2.1 Basics

Usually, only the duration of the simulation is required on this line, via the parameter `T_END`. The default is 1 s. For example, the following line will instruct FDS to run the simulation for 5400 seconds.

```
&TIME T_END=5400. /
```

If `T_END` is set to zero, only the set-up work is performed, allowing you to quickly check the geometry in Smokeview.

If you want the time line to start at a number other than zero, you can use the parameter `T_BEGIN` to specify the time written to file for the first time step. This would be useful for matching time lines of experimental data or video recordings.

Time-based RAMPs are evaluated using the actual time if the RAMP activation time is the same as `T_BEGIN`; otherwise, they are evaluated using the time from when the RAMP activates. Therefore, if you are setting `T_BEGIN` in order to test a time-based CTRL or DEVC that is ultimately linked to a RAMP, then you should set `T_BEGIN` to be slightly less than the time the RAMP will activate. For example if you are testing a VENT that is to open at 10 s whose SURF_ID uses a RAMP, `T_BEGIN` should be set slightly less than 10 s.

6.2.2 Special Topic: Controlling the Time Step

The initial time step size can be specified with `DT`. This parameter is normally set automatically by dividing the size of a mesh cell by the characteristic velocity of the flow. During the calculation, the time step is adjusted so that the CFL (Courant, Friedrichs, Lewy) condition is satisfied. The default value of `DT` is $5(\delta x \delta y \delta z)^{\frac{1}{3}} / \sqrt{gH}$ s, where δx , δy , and δz are the dimensions of the smallest mesh cell, H is the height of the computational domain, and g is the acceleration of gravity. Note that by default the time step is never allowed to increase above its initial value. To allow this to happen, set `RESTRICT_TIME_STEP=.FALSE.`

If something sudden is to happen right at the start of a simulation, like a sprinkler activation, it is a good idea to set the initial time step to avoid a numerical instability caused by too large a time step. Experiment with different values of `DT` by monitoring the initial time step sizes recorded in the output file `job_name.out`.

Finally, if you want to prevent FDS from automatically changing the time step, set `LOCK_TIME_STEP` equal to `.TRUE.` on the `TIME` line, in which case the specified time step, `DT`, will not be adjusted. This parameter is intended for diagnostic purposes only, for example, timing program execution. It can lead to numerical instabilities if the initial time step is set too high.

6.2.3 Special Topic: Steady-State Applications

Occasionally, there are applications in which only the steady-state solution (in a time-averaged sense) is desired. However, the time necessary to heat the walls to steady-state can make the cost of the calculation prohibitive. In these situations, if you specify a `TIME_SHRINK_FACTOR` of, say, 10, the specific heats of the various materials is reduced by a factor of 10, speeding up the heating of these materials roughly by 10. An example of an application where this parameter is handy is a validation experiment where a steady heat source warms up a compartment to a nearly equilibrium state at which point time-averaged flow quantities are measured.

Note that when `TIME_SHRINK_FACTOR` is used a device with `QUANTITY='TIME'` or a device or control function with a `DELAY` will have those values adjusted by the value of `TIME_SHRINK_FACTOR`. For example if a 10 s `DELAY` is specified for a CTRL input with a `TIME_SHRINK_FACTOR` of 10, then FDS will adjust the `DELAY` to 1 s.

6.3 Computational Meshes: The MESH Namelist Group (Table 19.13)

All FDS calculations must be performed within a domain that is made up of rectilinear volumes called *meshes*. Each mesh is divided into rectangular *cells*, the number of which depends on the desired resolution of the flow dynamics. MESH is the namelist group that defines the computational domain.

6.3.1 Basics

A mesh is a single right parallelepiped, i.e., a box. The coordinate system within a mesh conforms to the right hand rule. The origin point of a mesh is defined by the first, third and fifth values of the real number sextuplet, XB, and the opposite corner is defined by the second, fourth and sixth values. For example,

```
&MESH IJK=10,20,30, XB=0.0,1.0,0.0,2.0,0.0,3.0 /
```

defines a mesh that spans the volume starting at the origin and extending 1 m in the positive x direction, 2 m in the positive y direction, and 3 m in the positive z direction. The mesh is subdivided into uniform cells via the parameter IJK. In this example, the mesh is divided into 10 cm cubes. It is best if the mesh cells resemble cubes; that is, the length, width and height of the cells ought to be roughly the same. If it is desired that the mesh cells in a particular direction not be uniform in size, then the namelist groups TRNX, TRNY and/or TRNZ may be used to alter the uniformity of the mesh (See Section 6.3.5).

Any obstructions or vents that extend beyond the boundary of the mesh are cut off at the boundary. There is no penalty for defining objects outside of the mesh, and these objects will not appear in Smokeview.

The pressure solver in FDS employs Fast Fourier Transforms (FFTs) in the y and z directions, and this algorithm works most efficiently if the number of cells in these directions (the J and K of IJK) can be factored into low primes, like 2, 3, and 5. The number of cells in the x direction (the I in IJK) is not affected by this restriction because the pressure solver does not use an FFT in the x direction. However, since the pressure solver uses less than 10 % of the total CPU time, the gains in using low prime dimensions are usually negligible. Experiment with different mesh dimensions to ensure that those that are ultimately used do not unduly slow down the calculation.

6.3.2 Two-Dimensional and Axially-Symmetric Calculations

The governing equations solved in FDS are written in terms of a three dimensional Cartesian coordinate system. However, a two dimensional Cartesian or two dimensional cylindrical (axially-symmetric) calculation can be performed by setting the J in the IJK triplet to 1 on the MESH line. For axial symmetry, add CYLINDRICAL=.TRUE. to the MESH line, and the coordinate x is then interpreted as the radial coordinate r . No boundary conditions should be set at the planes $y = YMIN = XB(3)$ or $y = YMAX = XB(4)$, nor at $r = XMIN = XB(1)$ in an axially-symmetric calculation in which $r = XB(1) = 0$. For better visualizations, the difference between $XB(4)$ and $XB(3)$ should be small so that the Smokeview rendering appears to be in 2-D. An example of an axially-symmetric helium plume is given in Section 6.4.7.

6.3.3 Multiple Meshes

The term “multiple meshes” means that the computational domain consists of more than one computational mesh, usually connected although this is not required. If more than one mesh is used, there should be a MESH line for each. The order in which these lines are entered in the input file matters. In general, the meshes should be entered from finest to coarsest. FDS assumes that a mesh listed first in the input file has precedence over a mesh listed second if the two meshes overlap. Meshes can overlap, abut, or not touch

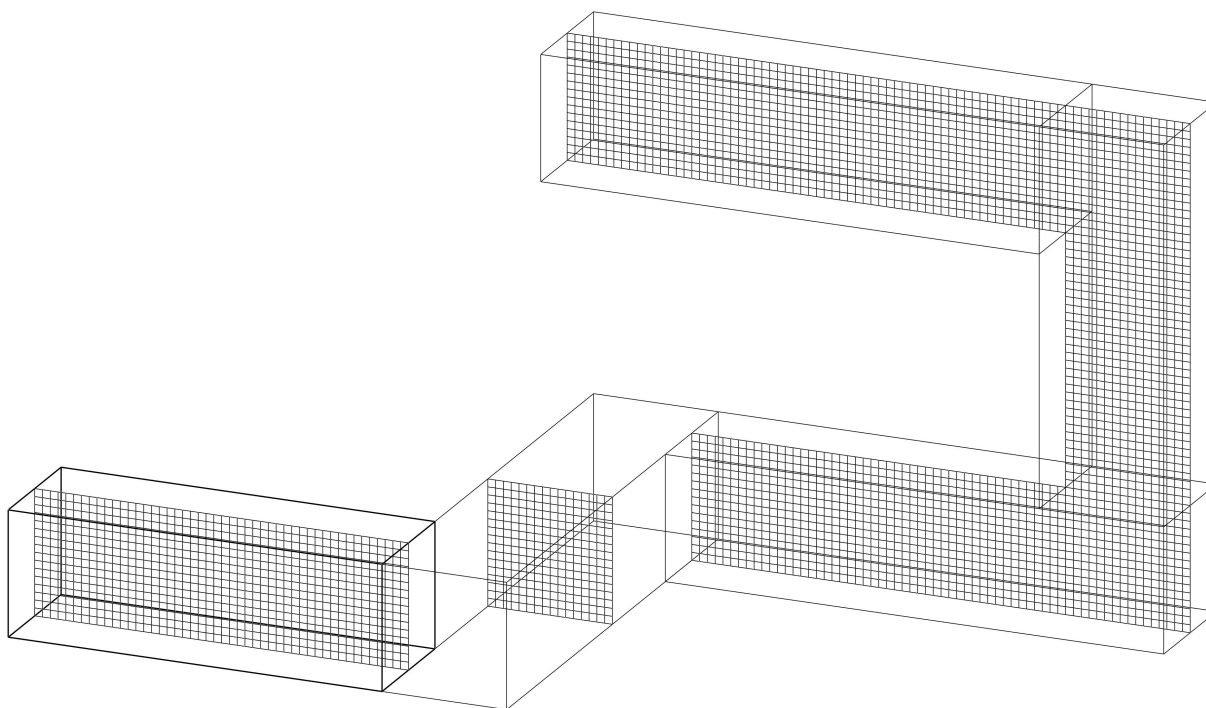


Figure 6.1: An example of a multiple-mesh geometry.

at all. In the last case, essentially two separate calculations are performed with no communication at all between them. Obstructions and vents are entered in terms of the overall coordinate system and need not apply to any one particular mesh. Each mesh checks the coordinates of all the geometric entities and decides whether or not they are to be included.

To run FDS in parallel using MPI (Message Passing Interface), you **must** break up the computational domain into multiple meshes so that the workload can be divided among the computers. In general, it is better to run multiple mesh cases with the MPI version of FDS if you have the computers available, but be aware that two computers will not necessarily finish the job in half the time as one. For the MPI version to work well, there has to be a comparable number of cells in each mesh, or otherwise most of the computers will sit idle waiting for the one with the largest mesh to finish processing each time step. You can use multiple meshes even when running the non-MPI version of FDS, in which case one CPU will serially process each mesh, one by one.

Usually in a MPI calculation, each mesh is assigned its own process, and each process its own processor. However, it is possible to assign more than one mesh to a single process, and it is possible to assign more than one process to a single processor. Consider a case that involves six meshes:

```
&MESH ID='mesh1', IJK=..., XB=..., MPI_PROCESS=0 /
&MESH ID='mesh2', IJK=..., XB=..., MPI_PROCESS=1 /
&MESH ID='mesh3', IJK=..., XB=..., MPI_PROCESS=1 /
&MESH ID='mesh4', IJK=..., XB=..., MPI_PROCESS=2 /
&MESH ID='mesh5', IJK=..., XB=..., MPI_PROCESS=3 /
&MESH ID='mesh6', IJK=..., XB=..., MPI_PROCESS=3 /
```

The parameter `MPI_PROCESS` instructs FDS to assign that particular mesh to the given process. In this case, only four processes are to be started, numbered 0 through 3. Note that the processes need to be invoked in

ascending order, starting with 0. Why would you do this? Suppose you only have four processors available for this job. By starting only four processes instead of six, you can save time because ‘mesh2’ and ‘mesh3’ can communicate directly with each other without having to transmit data using MPI calls over the network. Same goes for ‘mesh5’ and ‘mesh6’. In essence, it is as if these mesh pairs are neighbors and need not send mail to each other via the postal system. The letters can just be walked next door.

Additionally to the mesh assignment to individual MPI processes, the number of OpenMP threads for each MPI process may be specified. The parameter `N_THREADS` instructs FDS to set this number of threads and must be consistent for all meshes on the same MPI process, but may vary between them:

```
&MESH ID='mesh1', IJK=..., XB=..., MPI_PROCESS=0 /
&MESH ID='mesh2', IJK=..., XB=..., MPI_PROCESS=1, N_THREADS=2 /
&MESH ID='mesh3', IJK=..., XB=..., MPI_PROCESS=1, N_THREADS=2 /
&MESH ID='mesh4', IJK=..., XB=..., MPI_PROCESS=2, N_THREADS=4 /
&MESH ID='mesh5', IJK=..., XB=..., MPI_PROCESS=3 /
&MESH ID='mesh6', IJK=..., XB=..., MPI_PROCESS=3 /
```

Notes: An unspecified value will result in the default number of threads (see 3.2.1) on the MPI process. Some cluster systems do not support an heterogenous distribution of resources, here computational cores, per MPI process. Therefore, the number of OpenMP threads may change, but not the amount of cores assigned to the MPI process.

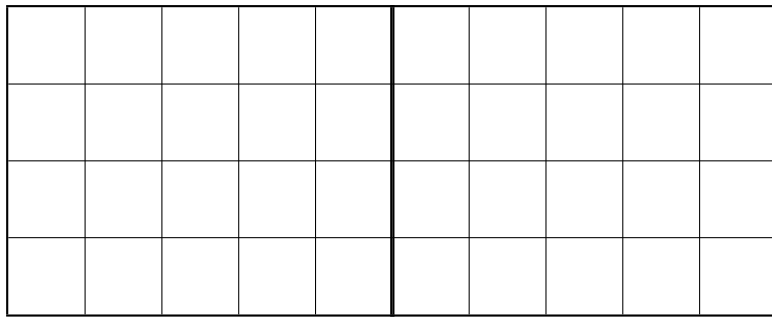
For cases involving many meshes, you might want to assign them colors using either the character string `COLOR` or the integer triplet `RGB`. You may also want to consider using the multiplying feature to easily create a 3-D array of meshes. See Section 7.5 for details.

Some parallel computing environments do not have a centralized file system, in which case FDS must write the output files for each process to a separate disk. If your computing cluster does not have a `SHARED_FILE_SYSTEM`, then set this parameter to `.FALSE.` on the `MISC` line. This parameter is also handy for MPI jobs involving hundreds or thousands of processes, in which case writing a single Smokeview file is time-consuming. When `SHARED_FILE_SYSTEM` is set to `.FALSE.`, the file that Smokeview reads is broken into pieces, one for each MPI process. By doing this, you avoid serially writing to the Smokeview file. One other useful parameter for larger MPI jobs is called `VERBOSE` on the `MISC` line. This logical parameter suppresses information that is printed to the diagnostic output files. By default, its value is `.TRUE.` for MPI jobs involving 50 or less processes, and `.FALSE.` for larger jobs.

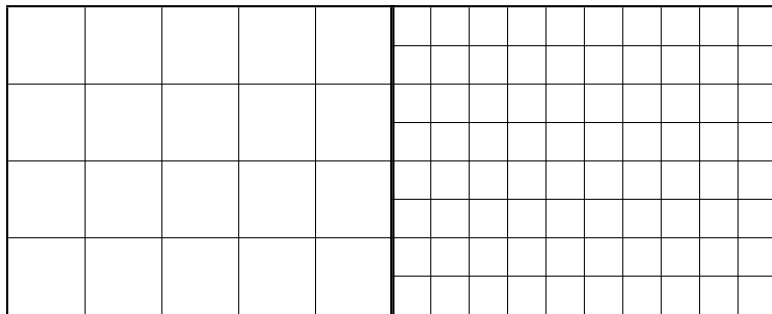
6.3.4 Mesh Alignment

Whether the calculation is to be run using MPI or not, the rules of prescribing multiple meshes are similar, with some issues to keep in mind. The most important rule of mesh alignment is that abutting cells ought to have the same cross sectional area, or integral ratios, as shown in Fig. 6.2. The following rules of thumb should also be followed when setting up a multiple mesh calculation:

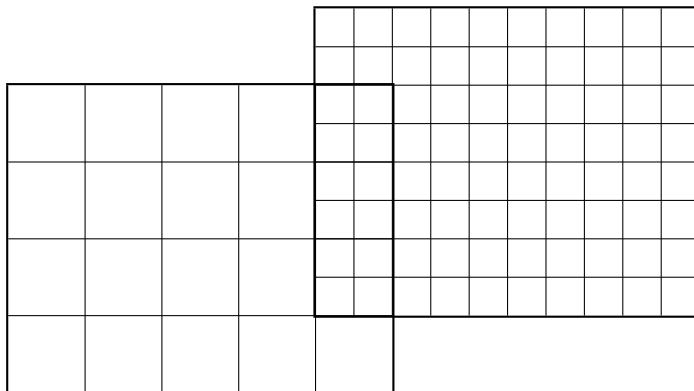
- Avoid putting mesh boundaries where critical action is expected, especially fire. Sometimes fire spread from mesh to mesh cannot be avoided, but if at all possible try to keep mesh interfaces relatively free of complicated phenomena since the exchange of information across mesh boundaries is not yet as accurate as cell to cell exchanges within one mesh.
- In general, there is little advantage to overlapping meshes because information is only exchanged at exterior boundaries. This means that a mesh that is completely embedded within another receives information at its exterior boundary, but the larger mesh receives no information from the mesh embedded within. Essentially, the larger, usually coarser, mesh is doing its own simulation of the scenario and



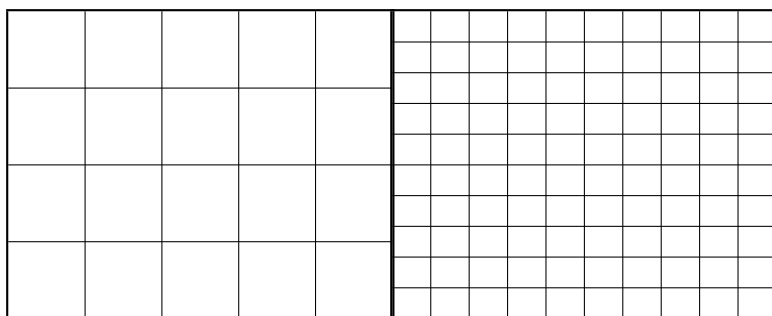
This is the ideal kind of mesh to mesh alignment.



This is allowed so long as there are an integral number of fine cells abutting each coarse cell.



This is allowed, but of questionable value.



This is not allowed.

Figure 6.2: Rules governing the alignment of meshes.

is not affected by the smaller, usually finer, mesh embedded within it. Details within the fine mesh, especially related to fire growth and spread, may not be picked up by the coarse mesh. In such cases, it is preferable to isolate the detailed fire behavior within one mesh, and position coarser meshes at the exterior boundary of the fine mesh. Then the fine and coarse meshes mutually exchange information.

- Be careful when using the shortcut convention of declaring an entire face of the domain to be an `OPEN` vent. Every mesh takes on this attribute. See Section 7.3 for more details.
- If a planar obstruction is close to where two meshes abut, make sure that each mesh “sees” the obstruction. If the obstruction is even a millimeter outside of one of the meshes, that mesh does not account for it, in which case information is not transferred properly between meshes.

Accuracy of the Multiple Mesh Calculation

Experiment with different mesh configurations using relatively coarse mesh cells to ensure that information is being transferred properly from mesh to mesh. There are two issues of concern. First, does it appear that the flow is being badly affected by the mesh boundary? If so, try to move the mesh boundaries away from areas of activity. Second, is there too much of a jump in cell size from one mesh to another? If so, consider whether the loss of information moving from a fine to a coarse mesh is tolerable.

6.3.5 Mesh Stretching: The `TRNX`, `TRNY` and `TRNZ` Namelist Groups (Table 19.29)

By default the mesh cells that fill the computational domain are uniform in size. However, it is possible to specify that the cells be non-uniform in one or two of the three coordinate directions. For a given co-

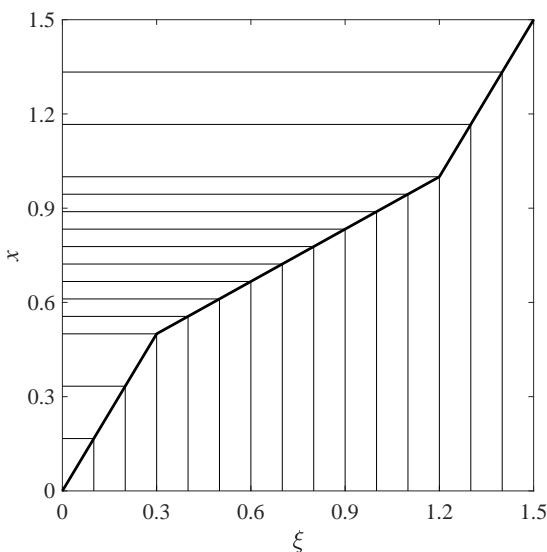


Figure 6.3: Piecewise-linear mesh transformation.

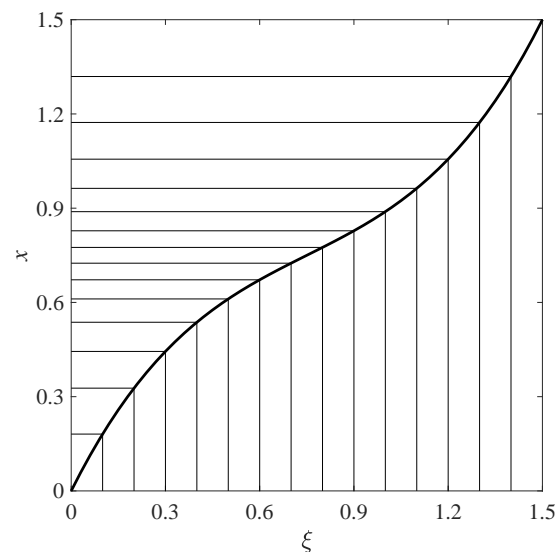


Figure 6.4: Polynomial mesh transformation.

ordinate direction, x , y or z , a function can be prescribed that transforms the uniformly-spaced mesh to a non-uniformly spaced mesh. **Be careful with mesh transformations!** If you shrink cells in one region you must stretch cells somewhere else. When one or two coordinate directions are transformed, the aspect ratio of the mesh cells in the 3D mesh will vary. To be on the safe side, transformations that alter the aspect ratio

of cells beyond 2 or 3 should be avoided. Keep in mind that the large eddy simulation technique is based on the assumption that the numerical mesh should be fine enough to allow the formation of eddies that are responsible for the mixing. In general, eddy formation is limited by the largest dimension of a mesh cell, thus shrinking the mesh resolution in one or two directions may not necessarily lead to a better simulation if the third dimension is large. Transformations, in general, reduce the efficiency of the computation, with two coordinate transformations impairing efficiency more than a transformation in one coordinate direction. Experiment with different meshing strategies to see how much of a penalty you will pay.

Here is an example of how to do a mesh transformation. Suppose your mesh is defined

```
&MESH IJK=15,10,20, XB=0.0,1.5,1.2,2.2,3.2,5.2 /
```

and you want to alter the uniform spacing in the x direction. First, refer to the figures above. You need to define a function $x = f(\xi)$ that maps the uniformly-spaced *Computational Coordinate* (CC) $0 \leq \xi \leq 1.5$ to the *Physical Coordinate* (PC) $0 \leq x \leq 1.5$. The function has three mandatory constraints: it must be monotonic (always increasing), it must map $\xi = 0$ to $x = 0$, and it must map $\xi = 1.5$ to $x = 1.5$. The default transformation function is $f(\xi) = \xi$ for a uniform mesh, but you need not do anything in this case.

Two types of transformation functions are allowed. The first, and simplest, is a piecewise-linear function. Figure 6.3 gives an example of a piecewise-linear transformation. The graph indicates how 15 uniformly spaced mesh cells along the horizontal axis are transformed into 15 non-uniformly spaced cells along the vertical axis. In this case, the function is made up of straight line segments connecting points (CC,PC), in increasing order, as specified by the following lines in the input file:

```
&TRNX CC=0.30, PC=0.50, MESH_NUMBER=2 /
&TRNX CC=1.20, PC=1.00, MESH_NUMBER=2 /
```

The integer MESH_NUMBER indicates which mesh to apply the transformation. If you want the transformation to be applied to all meshes, set MESH_NUMBER to 0. The parameter CC refers to the Computational Coordinate, ξ , located on the horizontal axis; PC is the Physical Coordinate, x , located on the vertical axis. The slopes of the line segments in the plot indicate whether the mesh is being stretched (slopes greater than 1) or shrunk (slopes less than 1). The tricky part about this process is that you usually have a desired shrinking/stretching strategy for the Physical Coordinate on the vertical axis, and must work backwards to determine what the corresponding points should be for the Computational Coordinate on the horizontal axis. Note that the above transformation is applied to the second mesh in a multiple mesh job.

The second type of transformation is a polynomial function whose constraints are of the form

$$\frac{d^n f(\text{CC})}{d\xi^n} = \text{PC}$$

Figure 6.4 gives an example of a polynomial transformation, for which the parameters are specified (assuming that this is the third mesh):

```
&TRNX IDERIV=0, CC=0.75, PC=0.75, MESH_NUMBER=3 /
&TRNX IDERIV=1, CC=0.75, PC=0.50, MESH_NUMBER=3 /
```

which correspond to the constraints $f(0.75) = 0.75$ and $\frac{df}{d\xi}(0.75) = 0.5$, or, in words, the function maps 0.75 into 0.75 and the slope of the function at $\xi = 0.75$ is 0.5. The transform function must also pass through the points (0,0) and (1.5,1.5), meaning that FDS must compute the coefficients for the cubic polynomial $f(\xi) = c_0 + c_1 \xi + c_2 \xi^2 + c_3 \xi^3$. More constraints on the function lead to higher order polynomial functions, so be careful about too many constraints which could lead to non-monotonic functions. The monotonicity of the function is checked by the program and an error message is produced if it is not monotonic.

Do not specify either linear transformation points or `IDERIV=0` points at coordinate values corresponding to the mesh boundaries. This is done automatically by FDS.

6.3.6 Mesh Resolution

A common question asked by new FDS users is, “What should my grid spacing be?” The answer is not easy because it depends considerably on what you are trying to accomplish. In general, you should build an FDS input file using a relatively coarse mesh, and then gradually refine the mesh until you do not see appreciable differences in your results. This is referred to as a mesh sensitivity study.

For simulations involving buoyant plumes, a measure of how well the flow field is resolved is given by the non-dimensional expression $D^*/\delta x$, where D^* is a characteristic fire diameter

$$D^* = \left(\frac{\dot{Q}}{\rho_\infty c_p T_\infty \sqrt{g}} \right)^{\frac{2}{5}} \quad (6.1)$$

and δx is the nominal size of a mesh cell¹. The quantity, \dot{Q} , is the total heat release rate of the fire. If it changes over time, you should consider the corresponding change in resolution. The quantity $D^*/\delta x$ can be thought of as the number of computational cells spanning the characteristic (not necessarily the physical) diameter of the fire. The more cells spanning the fire, the better the resolution of the calculation. It is better to assess the quality of the mesh in terms of this non-dimensional parameter, rather than an absolute mesh cell size. For example, a cell size of 10 cm may be “adequate,” in some sense, for evaluating the spread of smoke and heat through a building from a sizable fire, but may not be appropriate to study a very small, smoldering source.

The FDS Validation Guide [?] contains a table of the values of $D^*/\delta x$ used in the simulation of the validation experiments. The table is near the end of the chapter that describes all the experiments. These values range over two orders of magnitude and were chosen based on a grid resolution study and the particular attributes of the given fire scenario. It would be inappropriate to take any of these values as an “acceptable” minimum.

There are a number of special output quantities that provide local measures of grid resolution. See Section 18.10.21 for details.

¹The characteristic fire diameter is related to the characteristic fire size via the relation $Q^* = (D^*/D)^{5/2}$, where D is the physical diameter of the fire.

6.4 Miscellaneous Parameters: The MISC Namelist Group (Table 19.14)

MISC is the namelist group of global miscellaneous input parameters. It contains parameters that do not logically fit into any other category.

6.4.1 Basics

Only one MISC line should be entered in the data file. For example, the input line

```
&MISC TMPA=25. /
```

sets the ambient temperature at 25 °C. The MISC parameters vary in scope and degree of importance. Here is a partial list of MISCellaneous parameters. Others are described where necessary throughout this guide.

DNS A logical parameter that, if .TRUE., directs FDS to perform a Direct Numerical Simulation, as opposed to the default Large Eddy Simulation (LES). This feature is appropriate only for simulations that use mesh cells that are on the order of a millimeter or less in size, or for diagnostic purposes.

GVEC The 3 components of gravity, in m/s². The default is GVEC=0, 0, -9.81.

NOISE FDS initializes the flow field with a very small amount of “noise” to prevent the development of a perfectly symmetric flow when the boundary and initial conditions are perfectly symmetric. To turn this off, set NOISE=.FALSE. To control the amount of noise, set NOISE_VELOCITY. Its default value is 0.005 m/s.

OVERWRITE If .FALSE. FDS checks for the existence of CHID.out and stops execution if it exists.

P_INF Background pressure (at the ground) in Pa. The default is 101325 Pa.

TMPA Ambient temperature, the temperature of everything at the start of the simulation. The default is 20 °C.

6.4.2 Special Topic: Specifying a Force Field

Similar to the MEAN_FORCING feature, you may specify a constant and uniform force per unit volume by setting FORCE_VECTOR(1:3) on MISC. This is useful, for example, in specifying a mean pressure drop in a duct. In the absence of other forces, the force vector F_i affects the momentum equation by

$$\frac{\partial u_i}{\partial t} = -F_i/\rho \quad (6.2)$$

This feature is typically used together with periodic boundaries. To specify a mean pressure drop of 0.01 Pa/m in the x direction for a periodic duct, for example, use

```
&MISC FORCE_VECTOR(1)=0.01 /  
&VENT MB='XMIN', SURF_ID='PERIODIC' /  
&VENT MB='XMAX', SURF_ID='PERIODIC' /
```

You may apply a time ramp to FORCE_VECTOR(1), for example, via RAMP_FVX_T on MISC.

6.4.3 Special Topic: Stopping and Restarting Calculations

An important `MISC` parameter is called `RESTART`. Normally, a simulation consists of a sequence of events starting from ambient conditions. However, there are occasions when you might want to stop a calculation, make a few limited adjustments, and then restart the calculation from that point in time. To do this, first bring the calculation to a halt gracefully by creating a file called `CHID.stop` in the directory where the output files are located. Remember that FDS is case-sensitive. The file name must be exactly the same as the `CHID` and ‘stop’ should be lower case. FDS checks for the existence of this file at each time step, and if it finds it, gracefully shuts down the calculation after first creating a final `Plot3D` file and a file (or files in the case of a multiple mesh job) called `CHID.restart` (or `CHID_nn.restart`). To restart a job, the file(s) `CHID.restart` should exist in the working directory, and the phrase `RESTART=.TRUE.` needs to be added to the `MISC` line of the input data file. For example, suppose that the job whose `CHID` is “plume” is halted by creating a dummy file called `plume.stop` in the directory where all the output files are being created. To restart this job from where it left off, add `RESTART=.TRUE.` to the `MISC` line of the input file `plume.fds`, or whatever you have chosen to name the input file. The existence of a restart file with the same `CHID` as the original job tells FDS to continue saving the new data in the same files as the old². If `RESTART_CHID` is also specified on the `MISC` line, then FDS will look for old output files tagged with this string instead of using the specified `CHID` on the `HEAD` line. In this case, the new output files will be tagged with `CHID`, and the old output files will not be altered. When running the restarted job, the diagnostic output of the restarted job is appended to output files from the original job.

There may be times when you want to save restart files periodically during a run as insurance against power outages or system crashes. If this is the case, at the start of the original run set `DT_RESTART=50.` on the `DUMP` line to save restart files every 50 s, for example. The default for `DT_RESTART` is 1000000, meaning no restart files are created unless you gracefully stop a job by creating a dummy file called `CHID.stop`. It is also possible to use the new control function feature (see Section 17.5) to stop a calculation or dump a restart file when the computation reaches some measurable condition such as a first sprinkler activation.

Between job stops and restarts, major changes cannot be made in the calculation like adding or removing vents and obstructions. The changes are limited to those parameters that do not instantly alter the existing flow field. Since the restart capability has been used infrequently by the developers, it should be considered a fragile construct. Examine the output to ensure that no sudden or unexpected events occur during the stop and restart.

6.4.4 Special Topic: Initializing a 3D Velocity Field

It may be useful to start a calculation from an established flow field. Usually this can be accomplished with the normal restart functionality. But in some circumstances restart may be fragile, or you may want to specify a profile throughout the entire domain. For such situations we have added the ability to read the velocity field information from a comma-separated value (.csv) file. You have the option of creating the velocity file using FDS or creating your own. To generate the velocity initialization file with FDS, in the input file add a `DUMP` line with a `UVW_TIMER` (time in seconds). The timer will accept up to 10 values, and will write velocity files for each mesh and each timer index to the working directory. For example, if you want to write the simulation velocity field at 10 minutes into the run, add the following:

```
&DUMP Uvw_TIMER(1)=600 /
```

FDS will then write `CHID_uvw_nn.csv` for each time index and mesh. The format for this file is

²By default, when a job is restarted, the spreadsheet output files will be appended at the time the job was restarted, not the time the job was stopped. If you want the output files to be appended without clipping off any existing data, even though some duplicate output will be left over, then set `CLIP_RESTART_FILES` to `.FALSE.` on the `DUMP` line.

```

WRITE(LU_UVW) IMIN, IMAX, JMIN, JMAX, KMIN, KMAX
DO K=KMIN, KMAX
  DO J=JMIN, JMAX
    DO I=IMIN, IMAX
      WRITE(LU_UVW, *) U(I, J, K), ' ', V(I, J, K), ' ', W(I, J, K)
    ENDDO
  ENDDO
ENDDO

```

You may read in the 3-D velocity field using a CSVF line. For example:

```
&CSVF UVWFILE='my_velocity_field.csv' /
```

If multiple meshes are involved, it is assumed that the CSVF lines are provided in the input file in the same order as the meshes. For two meshes you might have the following:

```

&CSVF UVWFILE='CHID_t001_m001.csv' /
&CSVF UVWFILE='CHID_t001_m002.csv' /
&MISC PROJECTION=.TRUE. /

```

It is recommended that you also specify `PROJECTION=.TRUE.` on `MISC` if the specified velocity field does not satisfy the divergence constraint to machine precision on a staggered grid (this may not even be true if an analytical solution to the Navier-Stokes equations is sampled at the staggered velocity component locations).

6.4.5 Special Topic: Turning off the Flow Field

For certain types of diagnostic tests, it is useful to turn off the velocity field and exercise some other aspect of the model, like radiation of particle transport. To do this, set `FREEZE_VELOCITY=.TRUE.` on the `MISC` line.

6.4.6 Special Topic: Defying Gravity

By default, gravity points in the negative z direction, or more simply, downward. However, to change the direction of gravity to model a sloping roof or tunnel, for example, specify the gravity vector on the `MISC` line with a triplet of numbers of the form `GVEC=0., 0., -9.81`, with units of m/s^2 . This is the default, but it can be changed to be any direction.

There are a few special applications where you might want to vary the gravity vector as a function of time or as a function of the first spatial coordinate, x . For example, on board space craft, small motions can cause temporal changes in the normally zero level of gravity, an effect known as “g-jitter.” More commonly, in tunnel fire simulations, it is sometimes convenient to change the direction of gravity to mimic the change in slope. The slope of the tunnel might change as you travel through it; thus, you can tell FDS where to redirect gravity. For either a spatially or temporally varying direction and/or magnitude of gravity, do the following. First, on the `MISC` line, set the three components of gravity, `GVEC`, to some “base” state like `GVEC=1., 1., 1.`, which gives you the flexibility to vary all three components. Next, designate “ramps” for the individual components, `RAMP_GX`, `RAMP_GY`, and `RAMP_GZ`, all of which are specified on the `MISC` line. There is more discussion of `RAMPs` in Section 11, but for now you can use the following as a simple template to follow:

```

&MISC GVEC=1., 0., 1., RAMP_GX='x-ramp', RAMP_GZ='z-ramp' /

&RAMP ID='x-ramp', X= 0., F=0.0 /
&RAMP ID='x-ramp', X= 50., F=0.0 /

```



```

&RAMP ID='x-ramp', X= 51., F=-0.49 /
&RAMP ID='x-ramp', X=100., F=-0.49 /

&RAMP ID='z-ramp', X= 0., F=-9.81 /
&RAMP ID='z-ramp', X= 50., F=-9.81 /
&RAMP ID='z-ramp', X= 51., F=-9.80 /
&RAMP ID='z-ramp', X=100., F=-9.80 /

```

Note that both the x and z components of gravity are functions of x . FDS has been programmed to only allow variation in the x coordinate. Note also that F is just a multiplier of the “base” gravity vector components, given by `GVEC`. This is why using the number 1 is convenient – it allows you to specify the gravity components on the `RAMP` lines directly. The effect of these lines is to model the first 50 m of a tunnel without a slope, but the second 50 m with a 5 % slope upwards. Note that the angle from vertical of the gravity vector due to a 5 % slope is $\tan^{-1} 0.05 = 2.86^\circ$ and that 0.49 and 9.80 are equal to the magnitude of the gravity vector, 9.81 m/s^2 , multiplied by the sine and cosine of 2.86° , respectively. To check your math, the sum of the squares of the gravity components ought to equal 9.81. Notice in this case that the y direction has been left out because there is no y variation in the gravity vector. To vary the direction and/or magnitude of gravity in time, follow the same procedure but replace the x in the `RAMP` lines with a T .

6.4.7 Special Topic: The Baroclinic Vorticity

The pressure term in the momentum transport equation solved by FDS is decomposed as follows:

$$\frac{1}{\rho} \nabla \tilde{p} = \nabla \left(\frac{\tilde{p}}{\rho} \right) - \tilde{p} \nabla \left(\frac{1}{\rho} \right) \quad (6.3)$$

The pressure term is written like this so that a separable elliptic partial differential equation can be solved for the “total” pressure, $H \equiv |\mathbf{u}|^2/2 + \tilde{p}/\rho$, using a direct solver. The second term is calculated based on the pressure field from the previous time step, a slight approximation necessary to render the pressure equation separable. This term is sometimes referred to as the baroclinic torque, and it is responsible for generating vorticity due to the non-alignment of pressure and density gradients. In versions of FDS prior to 6, the inclusion of the baroclinic torque term was found to sometimes cause numerical instabilities. If it is suspected that the term is responsible for numerical problems, it can be removed by setting `BAROCLINIC=.FALSE.` on the `MISC` line. For example, in the simple helium plume test case below, neglecting the baroclinic torque changes the puffing behavior noticeably. In other applications, however, its effect is less significant. For further discussion of its effect, see Ref. [?].

Example Case: Flowfields/helium_2d_isothermal

This case demonstrates the use of baroclinic correction for an axially-symmetric helium plume. Note that the governing equations solved in FDS are written in terms of a three dimensional Cartesian coordinate system. However, a two dimensional Cartesian or two dimensional cylindrical (axially-symmetric) calculation can be performed by setting the number of cells in the y direction to 1. An example of an axially-symmetric helium plume is shown in Fig. 6.5.

6.4.8 Special Topic: Large Eddy Simulation Parameters

By default FDS uses the Deardorff [?, ?] turbulent viscosity,

$$(\mu_{\text{LES}}/\rho) = C_v \Delta \sqrt{k_{\text{sgs}}} \quad (6.4)$$

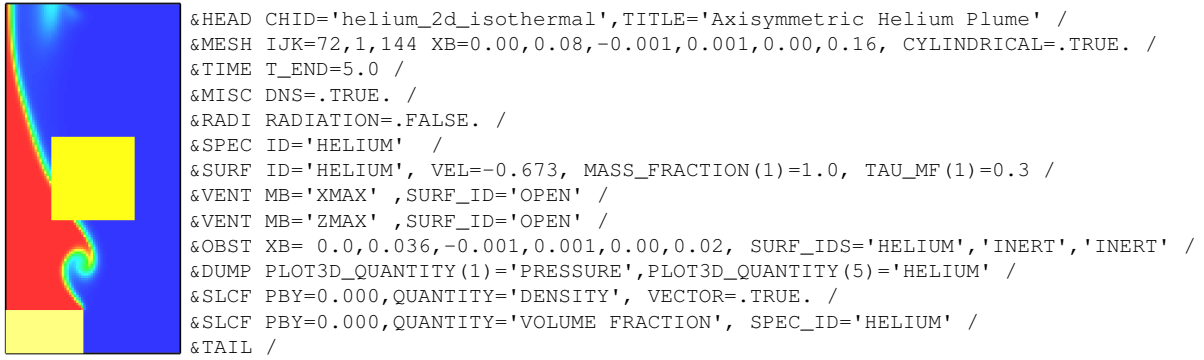


Figure 6.5: Simulation of a helium plume.

where $C_v = 0.1$ and the subgrid scale (sgs) kinetic energy is taken from an algebraic relationship based on scale similarity (see the FDS Technical Reference Guide [?]). The LES filter width is taken as the geometric mean of the local mesh spacing in each direction, $\Delta = (\delta x \delta y \delta z)^{(1/3)}$.

Options for the `TURBULENCE_MODEL` on the `MISC` line are listed in Table 6.1. Note that the model used in FDS versions 1-5 is 'CONSTANT SMAGORINSKY'. The thermal conductivity and material diffusivity are related to the turbulent viscosity by:

$$k_{LES} = \frac{\mu_{LES} c_p}{Pr_t} \quad ; \quad (\rho D)_{LES} = \frac{\mu_{LES}}{Sc_t} \quad (6.5)$$

The turbulent Prandtl number Pr_t and the turbulent Schmidt number Sc_t are assumed to be constant for a given scenario. Although it is not recommended for most calculations, you can modify $Pr_t = 0.5$, and $Sc_t = 0.5$ via the parameters `PR`, and `SC` on the `MISC` line. A more detailed discussion of these parameters is given in the FDS Technical Reference Guide [?].

Table 6.1: Turbulence model options.

TURBULENCE_MODEL	Description	Coefficient(s)
'CONSTANT SMAGORINSKY'	Constant coefficient Smagorinsky model [?]	C_SMAGORINSKY
'DYNAMIC SMAGORINSKY'	Dynamic Smagorinsky model [?, ?]	None
'DEARDORFF'	Deardorff model [?, ?]	C_DEARDORFF
'VREMAN'	Vreman's eddy viscosity model [?]	C_VREMAN
'RNG'	Renormalization group eddy viscosity model [?]	C_RNG, C_RNG_CUTOFF

6.4.9 Special Topic: Numerical Stability Parameters

FDS uses an explicit time advancement scheme; thus, the time step plays an important role in maintaining numerical stability and accuracy. Below we examine the constraints on the time step necessary for stability in the presence of advection, diffusion, and expansion of the velocity and scalar fields. In addition, there are additional constraints that ensure accuracy of various algorithms.

The Courant-Friedrichs-Lewy (CFL) Constraint

The well-known CFL constraint given by

$$\text{CFL} = \delta t \frac{\|\mathbf{u}\|}{\delta x} < 1 \quad (6.6)$$

places a restriction on the time step due to the advection velocity. The limits for the CFL are set by `CFL_MIN` (default 0.8) and `CFL_MAX` (default 1) on `MISC`. Physically, the constraint says that a fluid element should not traverse more than one cell within a time step. For LES, this constraint has the added advantage of keeping the implicit temporal and spatial filters consistent with each other. In other words, in order to resolve an eddy of size δx , the time step needs to obey the CFL constraint. If one were to employ an implicit scheme for purpose of taking time steps say 10 times larger than the CFL limit, the smallest resolvable turbulent motions would then be roughly 10 times the grid spacing, which would severely limit the benefit of using LES. In most cases, if you want the simulation to run faster, a better strategy is to coarsen the grid resolution while keeping the CFL close to 1.

The exact CFL needed to maintain stability depends on the order (as well as other properties) of the time integration scheme and the choice of velocity norm. Three choices for velocity norm are available in FDS (set on `MISC`):

`CFL_VELOCITY_NORM=0` (LES default, least restrictive, corresponds to L_∞ norm of velocity vector)

$$\frac{\|\mathbf{u}\|}{\delta x} = \max \left(\frac{|u|}{\delta x}, \frac{|v|}{\delta y}, \frac{|w|}{\delta z} \right) \quad (6.7)$$

`CFL_VELOCITY_NORM=1` (DNS default, most restrictive, corresponds to L_1 norm of velocity vector)

$$\frac{\|\mathbf{u}\|}{\delta x} = \frac{|u|}{\delta x} + \frac{|v|}{\delta y} + \frac{|w|}{\delta z} \quad (6.8)$$

`CFL_VELOCITY_NORM=2` (L_2 norm of velocity vector)

$$\frac{\|\mathbf{u}\|}{\delta x} = \sqrt{\left(\frac{u}{\delta x}\right)^2 + \left(\frac{v}{\delta y}\right)^2 + \left(\frac{w}{\delta z}\right)^2} \quad (6.9)$$

The Von Neumann Constraint

The Von Neumann constraint is given by

$$\text{VN} \equiv \delta t \max \left[\frac{\mu}{\rho}, D_\alpha \right] \left(\frac{1}{\delta x^2} + \frac{1}{\delta y^2} + \frac{1}{\delta z^2} \right) < \frac{1}{2} \quad (6.10)$$

The Von Neumann stability check is invoked by setting `CHECK_VN=.TRUE.` on the `MISC` line (for DNS, `CHECK_VN=.TRUE.` by default). The limits for VN may be adjusted using `VN_MIN` (default 0.4) and `VN_MAX` (default 0.5) on `MISC`. We can understand this constraint in a couple of different ways. First, we could consider the model for the diffusion velocity of species α in direction i , $V_{\alpha,i} Y_\alpha = -D_\alpha \partial Y_\alpha / \partial x_i$, and we would then see that VN is simply a CFL constraint due to diffusive transport.

We can also think of VN in terms of a total variation diminishing (TVD) constraint. That is, if we have variation (curvature) in the scalar field, we do not want to create spurious oscillations that can lead to an instability by overshooting the smoothing step. Consider the following explicit update of the heat equation for u in 1-D. Here subscripts indicate grid indices and v is the diffusivity.

$$u_i^{n+1} = u_i^n + \frac{\delta t v}{\delta x^2} (u_{i-1}^n - 2u_i^n + u_{i+1}^n) \quad (6.11)$$

Very simply, notice that if $\delta t v / \delta x^2 = 1/2$ then $u_i^{n+1} = (u_{i-1}^n + u_{i+1}^n)/2$. If the time step is any larger we overshoot the straight line connecting neighboring cell values. Of course, this restriction is only guaranteed to be TVD if the u field is “smooth”; otherwise, the neighboring cell values may be shifted in the opposite direction. Unfortunately, in LES there is no such guarantee and so the VN constraint can be particularly devilish in generating instabilities. For this reason, some practitioners like to employ implicit methods for the diffusive terms.

Realizable Mass Density Constraint

In an explicit Euler update of the continuity equation, if the time increment is too large the computational cell may be totally drained of mass, which, of course, is not physical. The constraint $\rho^{n+1} > 0$ therefore leads to the following restriction on the time step:

$$\delta t < \frac{\rho^n}{\bar{\mathbf{u}}^n \cdot \nabla \rho^n + \rho^n \nabla \cdot \mathbf{u}^n} \quad (6.12)$$

We can argue that the case we are most concerned with is when ρ^n is near zero. A reasonable approximation to (6.12) then becomes

$$\delta t < \frac{\rho}{\bar{u}_i \left(\frac{\rho-0}{\delta x_i} \right) + \rho \nabla \cdot \mathbf{u}} = \left[\frac{\bar{u}_i}{\delta x_i} + \nabla \cdot \mathbf{u} \right]^{-1} \quad (6.13)$$

Eq. (6.13) basically adds the effect of thermal expansion to the CFL constraint and provides a reason to prefer `CFL_VELOCITY_NORM=1` as the basis for the time step restriction. To be clear, the CFL constraint is now given by

$$\text{CFL} = \delta t \left(\frac{\|\mathbf{u}\|}{\delta x} + |\nabla \cdot \mathbf{u}| \right) \quad (6.14)$$

Stability of particle transport

The movement of Lagrangian particles over the course of a time step is calculated using an analytical solution and remains stable regardless of the time step used by the flow solver. However, if the particle moves over the width of several grid cells in a single time step, the momentum transferred between the particle and the gas cannot be allocated properly to all of the affected cells. To overcome this problem, FDS subdivides the gas phase time step based on each particle’s velocity. For example, if the particle travels across two cells in a single gas phase time step, then its trajectory is calculated by subdividing the time step into two.

In some cases with extremely fast particles, however, the stability of the overall flow behavior may require setting an additional parameter that limits the time step of the flow solver according to the speed of the fastest particle in the simulation. The actual value of the constraint is set using `PARTICLE_CFL_MAX` on the `MISC` line. A value of 1 (default) means that the fastest moving particle can move a distance of one grid cell during the time step. Because very fast nozzle velocities can cause extremely small time steps and hence very long run times, the `PARTICLE_CFL` constraint is set to `.FALSE.` by default. Setting `PARTICLE_CFL` to `.TRUE.` on the `MISC` line activates this constraint.

One additional parameter might be useful in some special cases. The numerical transport scheme for particles is first order accurate in time. If you want to improve the accuracy of the scheme, set `SECOND_ORDER_PARTICLE_TRANSPORT` to `.TRUE.` on the `PART` line. There is an increased computational cost incurred by this option, so check whether or not the extra cost is worth the extra accuracy. For most applications, given the relatively small time step of the gas phase solver, and the subdivision of the particle trajectory calculation to enforce a local particle CFL constraint, the improved accuracy is not worth the additional cost.

Heat Transfer Constraint

Note that the heat transfer coefficient, h , has units of $\text{W}/(\text{m}^2 \cdot \text{K})$. Thus, a velocity scale may be formed from $h/(\rho c_p)$. Anytime we have a velocity scale to resolve we have a CFL-type stability restriction. Therefore, the heat transfer stability check loops over all wall cells to ensure $\delta t \leq \delta x \rho c_p / h$. This check may be invoked by setting `CHECK_HT=.TRUE.` on the `MISC` line. It is `.FALSE.` by default.

Adjusting the Time Step

At the end of the first part of the explicit predictor-corrector time update, the time step is checked to ensure that it is within the appropriate stability bounds. If it is not, it is adjusted up or down by 10 % (or until it is within limits) and the predictor part of the time step is re-run. Resetting the stability parameters is not recommended except in very special circumstances, as they can lead to simulations failing due to numerical instabilities. If you want to prevent FDS from automatically changing the time step, set `LOCK_TIME_STEP` to `.TRUE.` on the `TIME` line, in which case the specified time step, `DT`, will not be adjusted. This parameter is intended for diagnostic purposes only, for example, timing program execution. It can lead to numerical instabilities if the initial time step is set too high.

6.4.10 Special Topic: Flux Limiters

FDS employs *total variation diminishing* (TVD) schemes for scalar transport. The default for LES is Superbee [?], so chosen because this scheme does the best job preserving the scalar variance in highly turbulent flows with coarse grid resolution. The default scheme for DNS is CHARM [?] because the gradient steepening used in Superbee forces a stair step pattern at high resolution, while CHARM is convergent. A few other schemes (including Godunov and central differencing) are included for completeness; more details can be found in the Tech Guide [?]. Table 6.2 below shows the integer codes which may be used to invoke the various limiter schemes.

```
&MISC FLUX_LIMITER=1 / ! invoke Godunov (first-order upwind scheme)
```

Table 6.2: Flux limiter options.

Scheme	FLUX_LIMITER
Central differencing	0
Godunov	1
Superbee (LES default)	2
MINMOD	3
CHARM (DNS default)	4
MP5	5

6.5 Initial Conditions: The `INIT` Namelist Group (Table 19.10)

Usually, an FDS simulation begins at time $t = 0$ with ambient conditions. The air temperature is assumed constant with height, and the density and pressure decrease with height (the z direction). This decrease is not noticed in most building scale calculations, but it is important in large outdoor simulations. There are

some scenarios for which it is convenient to change the ambient conditions within some rectangular region of the domain using the namelist keyword `INIT`.

Note that the rectangular regions defined by `INIT` may overlap, but in such cases, it is the second of the overlapping regions that takes precedence, including default conditions. That is, it is possible to overwrite the initial conditions explicitly specified by the first `INIT` line with the default initial conditions implied by the second `INIT` line.

Species

Species concentrations can be initialized using pairs of `SPEC_ID(N)` and `MASS_FRACTION(N)` where `N` is an ordinal index starting from 1. Note that `N` is not necessarily indicative of the order in which the species are listed in the input file. Also note that the background species cannot be specified when using `MASS_FRACTION`. The mass fraction of the background species will be set to account for any mass fraction not specified with other species. Make sure that you specify all species (components of `MASS_FRACTION(N)`) on the same `INIT` line for example,

```
&INIT XB=0.0,0.1,0.0,0.025,0.0,0.1,  
      MASS_FRACTION(1)=0.21, SPEC_ID(1)='OXYGEN',  
      MASS_FRACTION(2)=0.06, SPEC_ID(2)='PROPANE' /
```

Here, within the region whose bounds are given by the sextuplet `XB`, the initial mass fractions of oxygen and propane will be initialized to 0.21 and 0.06, respectively. You must specify the gas species using the `SPEC` namelist group. See Section 12 for details.

Temperature

To modify the local initial temperature, add lines of the form,

```
&INIT XB=0.0,0.1,0.0,0.025,0.0,0.1, TEMPERATURE=60. /
```

This indicates that the temperature shall be 60 °C instead of the ambient within the bounds given by `XB`. The `INIT` construct may be useful in examining the influence of stack effect in a building, where the temperature is different inside and outside. If you wanted to initialize both temperature and species in the same volume, both quantities would use the same `INIT` line,

```
&INIT XB=0.0,0.1,0.0,0.025,0.0,0.1,  
      MASS_FRACTION(1)=0.21, SPEC_ID(1)='OXYGEN',  
      MASS_FRACTION(2)=0.06, SPEC_ID(2)='PROPANE',  
      TEMPERATURE=60. /
```

Density

When specifying an initial density it is important to recognize the order in which FDS solves the governing equations. In the following example, initial species mass fractions, temperature, and density are all initialized in the same volume.

```
&INIT XB=0.0,0.1,0.0,0.025,0.0,0.1,  
      MASS_FRACTION(1)=0.21, SPEC_ID(1)='OXYGEN',  
      MASS_FRACTION(2)=0.06, SPEC_ID(2)='PROPANE',  
      TEMPERATURE=60., DENSITY=1.13 /
```

This example is a case where we have over-defined the problem. Since the temperature is computed from the equation of state using the specified density, the specified temperature will not, in general, satisfy the equation of state, and FDS will overwrite the specified temperature.

Heat Release Rate Per Unit Volume (HRRPUV)

The `INIT` line may also be used to specify a volumetric heat source term. For example,

```
&INIT XB=0.0,0.1,0.0,0.025,0.0,0.1, HRRPUV=1000. /
```

indicates that the region bounded by `XB` shall generate 1000 kW/m³. This feature is mainly useful for diagnostics, or to model a fire in a very simple way.

6.6 The Pressure Solver: The `PRES` Namelist Group (Table 19.18)

6.6.1 Parameters Related to the Solution of Poisson Equation for Pressure

FDS uses a low-Mach number formulation of the Navier-Stokes equations. One of the consequences of this is that the speed of sound is assumed infinite, and that the pressure throughout the computational domain is affected, instantaneously, by local changes in the flow field. A simple example of this is when air is pushed through a tunnel. If the tunnel has forced flow at one end and an opening at the other, the volume flow at the opening is the same as that which is forced at the other end. Without any heat addition, the air is assumed incompressible. Information is passed through the tunnel instantaneously in the model via a solution of a linear system of equations for the pressure. For a single mesh, the solution of this Poisson equation for the pressure is very accurate. However, for multiple meshes, there is potentially a delay in information passing throughout the domain because the Poisson equation is solved on each individual mesh, without any influence from the larger computational domain. The details of the numerical approach can be found in the FDS Technical Reference Guide.

Another limitation of the pressure solver is that at solid surfaces that are not part of the boundary of the computational domain, the pressure solver enforces a no-flux boundary condition. However, it is not perfect, and it is possible to have a non-zero normal velocity at a solid surface. For most applications, this velocity is so small that it has a negligible effect on the solution.

If either the error in the normal component of the velocity at a mesh interface or at a solid boundary is large, you can reduce it by making more than the default number of calls to the pressure solver at each time step. To do so, specify `VELOCITY_TOLERANCE` on the `PRES` line to be the maximum allowable normal velocity component on the solid boundary or the largest error at a mesh interface. It is in units of m/s. If you set this, experiment with different values, and monitor the number of pressure iterations required at each time step to achieve your desired tolerance. The default value is $\delta x/2$, where δx is the characteristic grid cell size. The number of iterations are written out to the file `CHID.out`. If you use a value that is too small, the CPU time required might be prohibitive. The maximum number of iterations for each half of the time step is given by `MAX_PRESSURE_ITERATIONS`, also on the `PRES` line. Its default value is 10.

By default, FDS will suspend the pressure iterations if the error does not drop below `ITERATION_SUSPEND_FACTOR` (0.95, by default) of its previous value. If you do not want FDS to suspend the pressure iteration, set `SUSPEND_PRESSURE_ITERATIONS` to `.FALSE.` on the `PRES` line. This will be done automatically if you change the default values of `VELOCITY_TOLERANCE` or `MAX_PRESSURE_ITERATIONS`.

If `.TRUE.`, the parameter `CHECK_POISSON` tells FDS to check that the left-hand and right-hand sides of the Poisson equation for H are equivalent (see the FDS Tech Guide [?]). The error is printed to the

CHID.out file.

Example Case: Pressure_Solver/duct_flow

To demonstrate how to improve the accuracy of the pressure solver, consider the flow of air through a square duct that crosses several meshes. In the sample input file, `duct_flow.fds`, air is pushed through a 1 m² duct at 1 m/s. With no thermal expansion, the volume flow into the duct ought to equal the volume flow out of the duct. Figure 6.6 displays the computed inflow and outflow as a function of time, and the number of pressure iterations required. The outflow does not match the inflow exactly because of inaccuracies at the solid and mesh boundaries. The `VELOCITY_TOLERANCE` has been set to 0.001 m/s with `MAX_PRESSURE_ITERATIONS` set to 1000 and the grid cell size is 0.2 m.

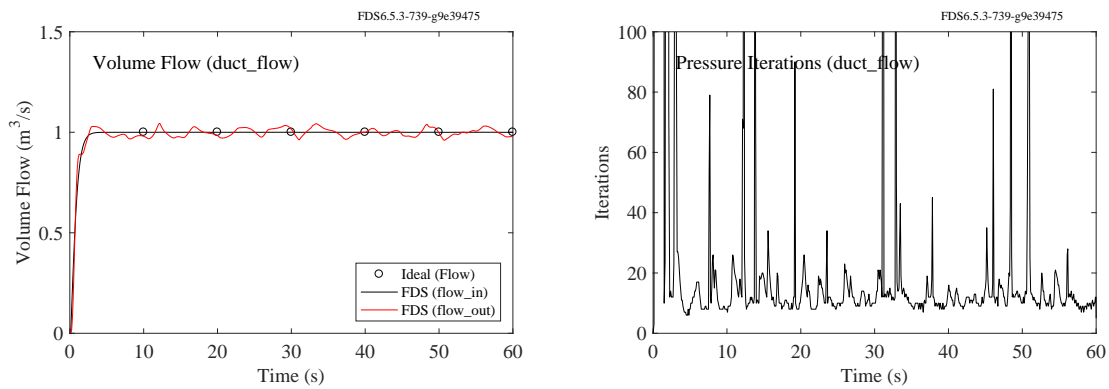


Figure 6.6: (Left) Volume flow into and out of a square duct. (Right) The number of pressure iterations as a function of time.

Example Case: Pressure_Solver/dancing_eddies

In this example, air is pushed through a 30 cm long, two-dimensional channel at 0.5 m/s. A plate obstruction normal to the flow creates a Karman vortex street. The computational domain is divided into 4 meshes. Two simulations are performed, one in which the `VELOCITY_TOLERANCE` is set to a relatively small value, and one in which it is set to the default value. Figure 6.7 shows the downstream pressure histories for these two cases compared to a simulation that uses only one mesh. The case with the tighter tolerance produces a better match to the single mesh solution, but at a higher computational cost.

One strategy for reducing the computational cost associated with a tightened tolerance on the normal velocity at mesh boundaries is to overlap the meshes. The idea is described in detail for a non-structured mesh in Ref. [?]. The basic idea is that a slight overlap of meshes can lead to faster convergence of the pressure solver towards the desired level of accuracy. In the `dancing_eddies` test case, the case called `dancing_eddies_tight` is rerun as `dancing_eddies_tight_overlap` where each mesh is extended an additional 10 grid cells in the longitudinal direction. This increases the work by approximately a factor of 85/75, but it reduces the number of iterations. Figure 6.8 shows the reduction in the number of pressure iterations (left) and the overall reduction in CPU time (right).

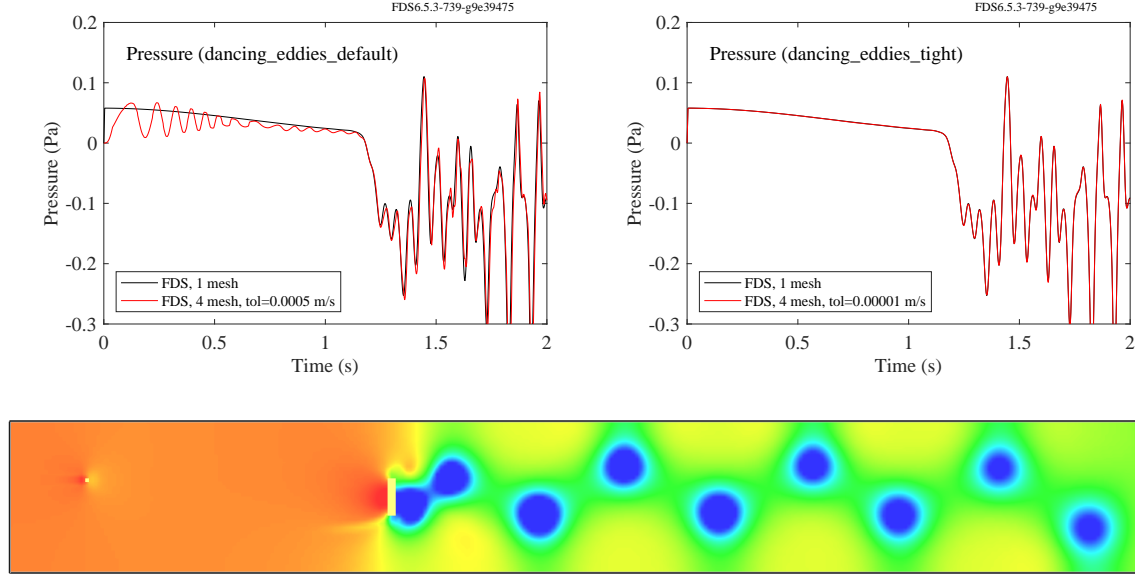


Figure 6.7: (Top) Comparison of pressure traces in the channel for two different settings of `VELOCITY_TOLERANCE`, the default value on the left and a tighter tolerance on the right. (Bottom) A contour plot of the pressure after 2 s with the default tolerance.

Special Case: True Periodic Boundaries

In Section 7.3.2, we discuss how to set periodic boundaries. By default, in order to handle the most general case of a periodic domain with multiple meshes (imagine a periodic channel divided into several meshes), FDS treats the pressure boundary condition as what we call an “interpolated boundary.” This means that the matrix for the pressure Poisson equation is arranged for Dirichlet boundary conditions (the value of the solution is specified at the boundary). This can lead to small errors in the solution and sometimes it is desirable to use the true periodic matrix for the Poisson equation. But with the current solver (Fishpak) this is only possible for a single mesh. If you want to implement true periodic boundaries for a single mesh case, set the appropriate `FISHPAK_BC` value to zero on the `PRES` line. For example,

```
&PRES FISHPAK_BC(1:3)=0,0,0 /
```

6.6.2 Pressure Considerations in Long Tunnels

A common application of FDS is tunnel fires, but simulations of fires in long, relatively tight tunnels can have spurious fluctuations in the pressure field that can lead to numerical instabilities. The root cause of the problem is the way that FDS solves the equation for the pressure, \tilde{p} (the tilde indicates that this is the *perturbation* pressure, or pressure above background). To solve for \tilde{p} , an elliptic PDE is formed by taking the divergence of the momentum equation, which contains the term $\nabla \tilde{p} / \rho$. Ideally, this PDE would be of the form:

$$\nabla \cdot \left(\frac{1}{\rho^n} \right) \nabla \tilde{p}^n = \dots \quad (6.15)$$

where n indicates the current time step. However, this equation cannot be solved efficiently because the matrix of the linear system of equations that is formed when discretizing it does not have constant coefficients

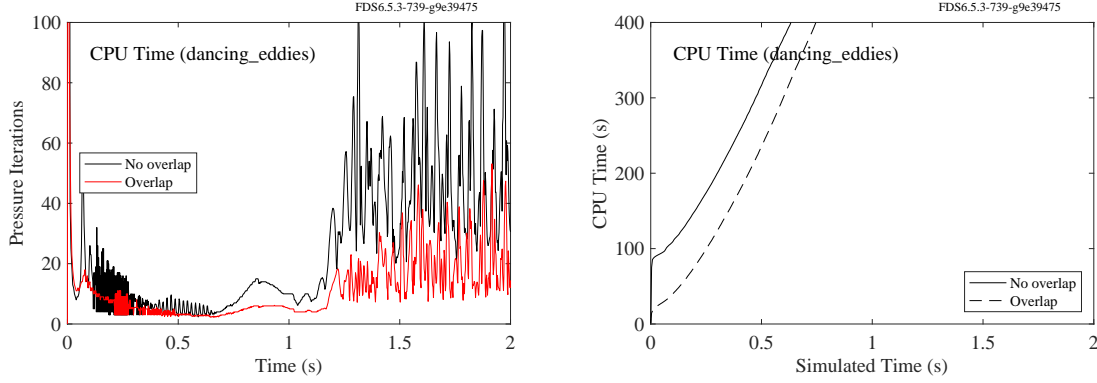


Figure 6.8: The number of pressure iterations (left) and the total CPU time (right) for overlapped and non-overlapped meshes.

because the density, ρ^n , changes with each new time step, n . To get around this problem, the pressure term in the momentum equation is decomposed as follows:

$$\frac{1}{\rho} \nabla \tilde{p} = \nabla \left(\frac{\tilde{p}}{\rho} \right) - \tilde{p} \nabla \left(\frac{1}{\rho} \right) \quad (6.16)$$

which leads to a Poisson equation that can be solved efficiently:

$$\nabla^2 \left(\frac{\tilde{p}^{n,k}}{\rho^n} \right) = \nabla \cdot \tilde{p}^{n,k-1} \nabla \left(\frac{1}{\rho^n} \right) + \dots \quad (6.17)$$

The superscript k indicates that this equation is solved multiple times, and k indicates the iteration. With each iteration, the old and new values of \tilde{p} are driven closer together. The iterations continue³ until the maximum value of the following expression in discretized form:

$$\varepsilon^k = \max_{ijk} \left| \nabla \cdot \left(\frac{1}{\rho^n} \right) \nabla \tilde{p}^{n,k} - \nabla^2 \left(\frac{\tilde{p}^{n,k}}{\rho^n} \right) + \nabla \cdot \tilde{p}^{n,k-1} \nabla \left(\frac{1}{\rho^n} \right) \right| \quad (6.18)$$

drops below a specified tolerance. By default, the `PRESSURE_TOLERANCE` is $20/\delta x^2$ where δx is the characteristic grid cell size. If a numerical instability occurs in a simulation involving a tunnel, you could try reducing this value on the `PRES` line to alleviate the mismatch between old and new pressure fields.

A simple test case, `Pressure_Solver/tunnel_demo.fds`, demonstrates some of the issues discussed in this section. An 8 MW fire is situated in a 4 m by 4 m by 128 m long tunnel that is sloped 10° , closed at the lower end and open at the upper end. The tunnel is divided into 8 meshes, all the same size, with a uniform grid of 20 cm. The case runs for a relatively short amount of time, and a diagnostic file⁴ containing information about the velocity and pressure errors is printed out. This file is generated by setting `VELOCITY_ERROR_FILE` to `.TRUE.` on the `DUMP` line. The name of the file is `tunnel_demo_pressit.csv`. Column 1 of the file is the time step iteration. Column 2 is the pressure iteration within the time step. Column 3 is the total number of pressure iterations for the entire simulation. Columns 4-7 contain the mesh and cell indices where the maximum velocity error occurs, which is listed

³Keep in mind that the pressure equation iterations continue until three criteria are satisfied. The first deals with the decomposition of the pressure term, the second deals with the normal component of velocity at internal solid surfaces, and the third deals with the mismatch of normal velocity components at mesh interfaces.

⁴Caution: the velocity error file can be quite large. Use it only for relatively short simulations only.

in column 8. Columns 9-12 contain the mesh and cell indices where the maximum pressure error occurs, which is listed in column 13.

Figure 6.9 shows the velocity and pressure errors over a short span of the simulation. For each quantity, there is a default error tolerance which may or may not be reached depending on whether or not the `MAX_PRESSURE_ITERATIONS` (default 10) has been reached, or whether or not the error is reduced by at least 25 % over the previous iteration. In some situations, the error decreases slowly, and these criteria have been added to avoid excessive iterations that lead to little improvement in accuracy. The errors shown in Fig. 6.9 are relatively large because fires in long, closed enclosures like tunnels can generate large fluctuations in pressure that challenge both the multiple mesh capability and the pressure solving algorithm discussed above.

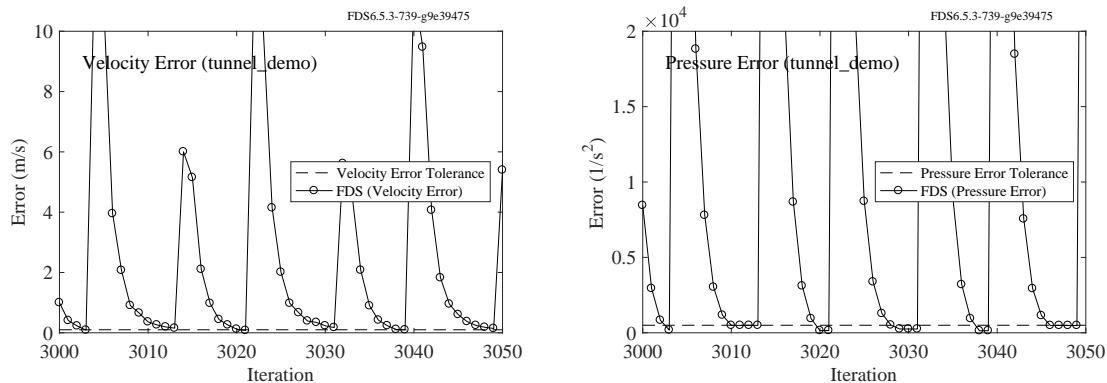


Figure 6.9: Reduction in velocity and pressure error due to iteration of the pressure solver.

6.6.3 Parameters Related to the Background Pressure when Breaking Pressure Zones

There are two parameters on the `PRES` line that control iterative procedures related to the coupling of velocity and pressure. One is called `RELAXATION_FACTOR` and its default value is 1. When there is an error in the normal component of velocity at a solid boundary, this parameter dictates that the correction be applied in 1 time step. If its value were 0.5, the correction would be applied in 2 time steps.

A similar parameter is the `PRESSURE_RELAX_TIME`. It controls the rate at which the pressures in adjacent compartments are brought into equilibrium following a breach. Its default value is 1 s, meaning that equilibrium is achieved in roughly a second.

6.7 Setting Limits: The `CLIP` Namelist Group (Table 19.2)

The algorithms in FDS are designed to work within a certain range of values for density, temperature and mass fraction. To prevent unphysical results, there are bounds placed on these variables to prevent a single spurious value from causing a numerical instability. It also prevents out of range errors from calls to temperature-dependent look-up tables. By default, FDS determines the lowest and highest values of the variables based on your input, but it is not possible in all cases to anticipate just how low or high a given value might be. Thus, on rare occasions you might need to set upper or lower bounds on the density, temperature, or species mass fractions. Temperature and density bounds are input under the namelist group called `CLIP`. The parameters are listed in Table 19.2. You only need to set these values if you notice that one of them appears to be “cut off” when examining the results in Smokeview. For typical fire scenarios, you need

not set these values, but if you anticipate relatively low or high values in an unusual case, take a look at the calculation results to determine if a change in the bounds is needed.

It is possible for the species mass fractions to dip slightly below zero or increase slightly above 1. To force the species mass fractions to remain strictly between 0 and 1, set `CLIP_MASS_FRACTION` to `.TRUE.` on the `MISC` line.

Chapter 7

Building the Model

A considerable amount of work in setting up a calculation lies in specifying the geometry of the space to be modeled and applying boundary conditions to the solid surfaces. The geometry is described in terms of rectangular obstructions that can heat up, burn, conduct heat, etc.; and vents from which air or fuel can be either injected into, or drawn from, the flow domain. A boundary condition needs to be assigned to each obstruction and vent describing its thermal properties. A fire is just one type of boundary condition. This chapter describes how to build the model.

7.1 Bounding Surfaces: The SURF Namelist Group (Table 19.26)

Before describing how to build up the geometry, it is first necessary to explain how to describe what these bounding surfaces consist of. SURF is the namelist group that defines the structure of all solid surfaces or openings within or bounding the flow domain. Boundary conditions for obstructions and vents are prescribed by referencing the appropriate SURF line(s) whose parameters are described in this section.

The default boundary condition for all solid surfaces is that of a smooth inert wall with the temperature fixed at `TMPA`, and is referred to as 'INERT'. If only this boundary condition is needed, there is no need to add any SURF lines to the input file. If additional boundary conditions are desired, they are to be listed one boundary condition at a time. Each SURF line consists of an identification string `ID='...'` to allow references to it by an obstruction or vent. Thus, on each OBST and VENT line that are to be described below, the character string `SURF_ID='...'` indicates the ID of the SURF line containing the desired boundary condition parameters. If a particular SURF line is to be applied as the default boundary condition, set `DEFAULT=.TRUE.` on the SURF line.

7.2 Creating Obstructions: The OBST Namelist Group (Table 19.16)

The namelist group OBST contains parameters used to define obstructions. The entire geometry of the model is made up entirely of rectangular solids, each one introduced on a single line in the input file.

7.2.1 Basics

Each OBST line contains the coordinates of a rectangular solid within the flow domain. This solid is defined by two points (x_1, y_1, z_1) and (x_2, y_2, z_2) that are entered on the OBST line in terms of the real sextuplet XB. In addition to the coordinates, the boundary conditions for the obstruction can be specified with the parameter SURF_ID, which designates which SURF line (Section 7.1) to apply at the surface of the obstruction. If the obstruction has different properties for its top, sides and bottom, do not specify only one SURF_ID. Instead,

use `SURF_IDS`, an array of three character strings specifying the boundary condition IDs for the top, sides and bottom of the obstruction, respectively. If the default boundary condition is desired, then `SURF_ID` or `SURF_IDS` need not be set. However, if at least one of the surface conditions for an obstruction is the inert default, it can be referred to as `'INERT'`, but it does not have to be explicitly defined. For example:

```
&SURF ID='FIRE', HRRPUA=1000.0 /
&OBST XB=2.3,4.5,1.3,4.8,0.0,9.2, SURF_IDS='FIRE','INERT','INERT' /
```

puts a fire on top of the obstruction. This is a simple way of prescribing a burner.

In addition to `SURF_ID` and `SURF_IDS`, you can also use the sextuplet `SURF_ID6` as follows:

```
&OBST XB=2.3,4.5,1.3,4.8,0.0,9.2,
      SURF_ID6='FIRE','INERT','HOT','COLD','BLOW','INERT' /
```

where the six surface descriptors refer to the planes $x = 2.3$, $x = 4.5$, $y = 1.3$, $y = 4.8$, $z = 0.0$, and $z = 9.2$, respectively. Note that `SURF_ID6` should not be used on the same `OBST` line as `SURF_ID` or `SURF_IDS`.

Obstructions may be created or removed during a simulation. See Section 17.4.1 for details.

7.2.2 Thin Obstructions

Obstructions can have zero thickness. Often, thin sheets, like a window, form a barrier, but if the numerical mesh is coarse relative to the thickness of the barrier, the obstruction might be unnecessarily large if it is assumed to be one layer of mesh cells thick. All faces of an obstruction are shifted to the closest mesh cell. If the obstruction is very thin, the two faces may be approximated on the same cell face. FDS and Smokeview render this obstruction as a thin sheet, but it is allowed to have thermally thick boundary conditions. This feature is fragile, especially in terms of burning and blowing gas. A thin sheet obstruction can only have one velocity vector on its face, thus a gas cannot be injected reliably from a thin obstruction because whatever is pushed from one side is necessarily pulled from the other. For full functionality, the obstruction should be specified to be at least one mesh cell thick. Thin sheet obstructions work fine as flow barriers, but other features are fragile and should be used with caution. To prevent FDS from allowing thin sheet obstructions, set `THICKEN_OBSTRUCTIONS=.TRUE.` on the `MISC` line, or `THICKEN=.TRUE.` on each `OBST` line for which the thin sheet assumption is not allowed.

Obstructions that are too small relative to the underlying numerical mesh are rejected. Be careful when testing cases on coarse meshes.

7.2.3 Overlapping Obstructions

If the faces of two obstructions overlap each other, FDS will choose the surface properties of the obstruction that is specified second in the input file. If you do not want this, add `OVERLAY=.FALSE.` to the `OBST` line of the second obstruction, in which case the surface properties of the first obstruction will be applied. The default value of `OVERLAY` is `.TRUE.`

When obstructions overlap, Smokeview renders both obstructions independently of each other, often leading to an unsightly cross-hatching of the two surface colors where there is an overlap. A simple remedy for this is to “shrink” the obstruction you do not wish to take precedence by slightly by adjusting its coordinates (`XB`) accordingly. Then, in Smokeview, toggle the “q” key to show the obstructions as you specified them, rather than as FDS rendered them.

7.2.4 Preventing Obstruction Removal

Obstructions can be protected from the `HOLE` punching feature. Sometimes it is convenient to create a door or window using a `HOLE`. For example, suppose a `HOLE` is punched in a wall to represent a door or window. An obstruction can be defined to fill this hole (presumably to be removed or colored differently or whatever) so long as the phrase `PERMIT_HOLE=.FALSE.` is included on the `OBST` line. In general, any obstruction can be made impenetrable to a `HOLE` using this phrase. By default, `PERMIT_HOLE=.TRUE.`, meaning that an obstruction is assumed to be penetrable unless otherwise directed. Note that if a penetrable obstruction and an impenetrable obstruction overlap, the obstruction with `PERMIT_HOLE=.FALSE.` should be listed first.

If the obstruction is not to be removed or rejected for any reason, set `REMOVABLE=.FALSE.` This is sometimes needed to stop FDS from removing the obstruction if it is embedded within another, like a door within a wall.

In rare cases, you might not want to allow a `VENT` to be attached to a particular obstruction, in which case set `ALLOW_VENT=.FALSE.`

7.2.5 Transparent or Outlined Obstructions

Obstructions can be made semi-transparent by assigning a `TRANSPARENCY` on the `OBST` line. This real parameter ranges from 0 to 1, with 0 being fully transparent. The parameter should always be set along with either `COLOR` or an RGB triplet. It can also be specified on the appropriate `SURF` line, along with a color indicator. If you want the obstruction to be invisible, set `COLOR='INVISIBLE'`.

Obstructions are typically drawn as solids in Smokeview. To draw an outline representation, set `OUTLINE` equal to `.TRUE.`

7.2.6 Creating Holes in Obstructions: The `HOLE` Namelist Group (Table 19.8)

The `HOLE` namelist group defines parameters that carve a hole out of an existing obstruction or set of obstructions. To do this, add lines of the form

```
&HOLE XB=2.0,4.5,1.9,4.8,0.0,9.2 /
```

Any solid mesh cells within the volume $2.0 < x < 4.5$, $1.9 < y < 4.8$, $0.0 < z < 9.2$ are removed. Obstructions intersecting the volume are broken up into smaller blocks. If the hole represents a door or window, a good rule of thumb is to punch more than enough to create the hole. This ensures that the hole is created through the entire obstruction. For example, if the `OBST` line denotes a wall 0.1 m thick:

```
&OBST XB=1.0,1.1,0.0,5.0,0.0,3.0 /
```

and you want to create a door, add this:

```
&HOLE XB=0.99,1.11,2.0,3.0,0.0,2.0 /
```

The extra centimeter added to the x coordinates of the hole make it clear that the hole is to punch through the entire obstruction.

When a `HOLE` is created, the affected obstruction(s) are either rejected, or created or removed at pre-determined times. See Section 17.4.1 for details. To allow a hole to be controlled with either the `CTRL` or `DEVC` namelist groups, you will need to add the `CTRL_ID` or `DEVC_ID` parameter respectively, to the `HOLE` line. When the state of the `HOLE` evaluates to `.FALSE.`, an obstruction will be placed in the `HOLE`. By default the obstruction filling the `HOLE` will take the color of the surrounding `OBST` that the `HOLE` was punched through. To make the obstruction filling the `HOLE` a different color than the original obstruction, set the

COLOR or integer triplet RGB on the HOLE line (see Section 7.4). If you want the obstruction filling the HOLE to be invisible, then set COLOR='INVISIBLE'. Additionally, you may use the keyword TRANSPARENCY, real number from 0 to 1, to make the obstruction filling the HOLE transparent. See Section 17.4.1 for an example.

If an obstruction is not to be punctured by a HOLE, add PERMIT_HOLE=.FALSE. to the OBST line. Note that a HOLE has no effect on a VENT or a mesh boundary. It only applies to OBSTstructions.

It is a good idea to inspect the geometry by running either a setup job (T_END=0 on the TIME line) or a short-time job to test the operation of devices and control functions.

7.3 Applying Surface Properties: The VENT Namelist Group (Table 19.30)

Whereas the OBST group is used to specify obstructions within the computational domain, the VENT group (Table 19.30) is used to prescribe planes adjacent to obstructions or external walls. Note that the label VENT is used for historical reasons – this group of parameters has evolved well beyond its initial role as simply allowing for air to be blown into, or sucked out of, the computational domain.

7.3.1 Basics

The vents are chosen in a similar manner to the obstructions, with the sextuplet XB denoting a plane abutting a solid surface. Two of the six coordinates must be the same, denoting a plane as opposed to a solid. Note that only one VENT may be specified for any given wall cell. If additional VENT lines are specified for a given wall cell, FDS will output a warning message and ignore redundant VENT lines.

The term “VENT” is somewhat misleading. Taken literally, a VENT can be used to model components of the ventilation system in a building, like a diffuser or a return. In these cases, the VENT coordinates form a plane on a solid surface forming the boundary of the duct. No holes need to be created through the solid; it is assumed that air is pushed out of or sucked into duct work within the wall. Less literally, a VENT is used simply as a means of applying a particular boundary condition to a rectangular patch on a solid surface. A fire, for example, is usually created by first generating a solid obstruction via an OBST line, and then specifying a VENT somewhere on one of the faces of the solid with a SURF_ID with the characteristics of the thermal and combustion properties of the fuel. For example, the lines

```
&OBST XB=0.0,5.0,2.0,3.0,0.0,4.0, SURF_ID='big block' /  
&VENT XB=1.0,2.0,2.0,2.0,1.0,3.0, SURF_ID='hot patch' /
```

specify a large obstruction (with the properties given elsewhere in the file under the name 'big block') with a “patch” applied to one of its faces with alternative properties under the name 'hot patch'. This latter surface property need not actually be a “vent,” like a supply or return duct, but rather just a patch with different boundary conditions than those assumed for the obstruction. Note that the surface properties of a VENT over-ride those of the underlying obstruction.

A VENT must always be attached to a solid obstruction. See Section 9.1 for instructions on specifying different types of fans that allow gases to flow through.

An easy way to specify an entire external wall is to replace XB with MB (Mesh Boundary), a character string whose value is one of the following: 'XMAX', 'XMIN', 'YMAX', 'YMIN', 'ZMAX' or 'ZMIN' denoting the planes $x = XMAX$, $x = XMIN$, $y = YMAX$, $y = YMIN$, $z = ZMAX$ or $z = ZMIN$, respectively. Like an obstruction, the boundary condition index of a vent is specified with SURF_ID, indicating which of the listed SURF lines to apply. If the default boundary condition is desired, then SURF_ID need not be set.

Be careful when using the MB shortcut when doing a multiple mesh simulation; that is, when more than one rectangular mesh is used. The plane designated by the character string MB may be mistakenly applied

to more than one mesh, possibly leading to confusion about whether a plane is a solid wall or an open boundary. Check the geometry in Smokeview to assure that the `VENTs` are properly specified. Use color as much as possible to double-check the set-up. More detail on color in Section 7.4 and Table 7.1. Also, the parameter `OUTLINE=.TRUE.` on the `VENT` line causes the `VENT` to be drawn as an outline in Smokeview.

7.3.2 Special Vents

There are three reserved `SURF_ID`'s that may be applied to a `VENT` – '`OPEN`', '`MIRROR`', and '`PERIODIC`'. The term *reserved* means that these `SURF_IDS` should not be explicitly defined by you. Their properties are predefined.

Open Vents

The first special `VENT` is invoked by the parameter `SURF_ID='OPEN'`. This is used only if the `VENT` is applied to the exterior boundary of the computational domain, where it denotes a passive opening to the outside. By default, FDS assumes that the exterior boundary of the computational domain (the `XBs` on the `MESH` line) is a solid wall. To create a totally or partially open domain, use `OPEN` vents on the exterior mesh boundaries. It is sometimes convenient to specify doors or windows that open out to the exterior of the computational domain by simply specifying it to be `OPEN`. However, keep in mind that the pressure boundary condition on such an opening is imperfect, and it is recommended that if the flow through the doorway or window is important, you should extend the domain a few meters rather than use an `OPEN` boundary. You would still have to use the `OPEN` boundary to open up one or more sides of the computational domain, but these openings would be far enough away from the modeled door or window that they would not affect the flow pattern.

By default, it is assumed that ambient conditions exist beyond the '`OPEN`' vent. However, in some cases, you may want to alter this assumption, for example, the temperature. If you assume a temperature other than ambient, specify `TMP_EXTERIOR` along with `SURF_ID='OPEN'`. You can modify the time history of this parameter using a ramp function, `TMP_EXTERIOR_RAMP`. Use this option cautiously – in many situations if you want to describe the exterior of a building, it is better to include the exterior explicitly in your calculation because the flow in and out of the doors and windows will be more naturally captured. See Section 10.1.1 for more details. If you want to specify a non-ambient pressure at the `OPEN` boundary, see Section 9.4.

The `OPEN` pressure boundary condition is most stable for flows that are predominantly normal to the vent, either mostly in or mostly out. This is because the prescribed pressure at an `OPEN` boundary is ill-conditioned (a small perturbation to the input may lead to large change in the output) if the flow is parallel to the vent. Suppose, for example, that an outdoor flow is 10 m/s in the x direction and ± 0.001 m/s in the z direction with an `OPEN` top boundary. The kinetic energy of this flow is roughly $k = 50 \text{ m}^2/\text{s}^2$. When the vertical velocity is positive (+0.001 m/s) then the prescribed boundary condition for the stagnation pressure is set to $H = k = 50 \text{ m}^2/\text{s}^2$. But when the vertical velocity is negative (-0.001 m/s) then $H = 0$ (see [?]). For this reason, `OPEN` vents should be used with care in outdoor applications. See Section 10.2 for an alternative approach.

Vents to the outside of the computational domain (`OPEN` vents) *can* be opened or closed during a simulation. It is best done by creating or removing a thin obstruction that covers the `OPEN VENT`. See Section 17.4.2 for details.

Mirror Vents

A `VENT` with `SURF_ID='MIRROR'` denotes a symmetry plane. Usually, a `MIRROR` spans an entire face of the computational domain, essentially doubling the size of the domain with the `MIRROR` acting as a plane

of symmetry. The flow on the opposite side of the `MIRROR` is exactly reversed¹. From a numerical point of view, a `MIRROR` is a no-flux, free-slip boundary. As with `OPEN`, a `MIRROR` can only be prescribed at an exterior boundary of the computational domain. Often, `OPEN` or `MIRROR VENTS` are prescribed along an entire side of the computational domain, in which case the “MB” notation is handy.

In conventional RANS (Reynolds-Averaged Navier-Stokes) models, symmetry boundaries are often used as a way of saving on computation time. However, because FDS is an LES (Large Eddy Simulation) model, the use of symmetry boundaries should be considered carefully. The reason for this is that an LES model does not compute a time-averaged solution of the N-S equations. In other words, for a RANS model, a fire plume is represented as an axially-symmetric flow field because that is what you would expect if you time-averaged the actual flow field over a sufficient amount of time. Thus, for a RANS model, a symmetry boundary along the plume centerline is appropriate. In an LES model, however, there is no time-averaging built into the equations, and there is no time-averaged, symmetric solution. Putting a `MIRROR` boundary along the centerline of a fire plume will change its dynamics entirely. It will produce something very much like the flow field of a fire that is adjacent to a vertical wall. For this reason, a `MIRROR` boundary condition is not recommended along the centerline of a turbulent fire plume. If the fire or burner is very small, and the flow is laminar, then the `MIRROR` boundary condition makes sense. In fact, in 2-D calculations, `MIRROR` boundary conditions are employed in the third coordinate direction (this is done automatically, you need not specify it explicitly).

Periodic Vents

A `VENT` with `SURF_ID='PERIODIC'` may be used in combination with another periodic vent on the boundary of the domain in any of the three coordinate directions. As an example, consider the following (valid for a single mesh case):

```
&VENT MB='XMIN', SURF_ID='PERIODIC' /
&VENT MB='XMAX', SURF_ID='PERIODIC' /
```

In this example, the entire `XMIN` boundary is periodic with the `XMAX` boundary.

For multi-mesh cases with `PERIODIC` boundaries the `VENT` planes must be specified using `PBX`, `PBY`, or `PBZ`. If $x_{\min} = 0$ and $x_{\max} = 1$, for example, use

```
&VENT PBX=0, SURF_ID='PERIODIC' /
&VENT PBX=1, SURF_ID='PERIODIC' /
```

Also, you must explicitly tell FDS that two non-connected meshes need to share information. To do this, use the parameter `PERIODIC_MESH_IDS(1:3)` on the appropriate `MESH` lines. For example, suppose you have three meshes side by side. Mesh 1 and Mesh 3 have periodic boundaries, but they do not physically touch each other. Provide this information as follows:

```
&MESH ID='mesh1', IJK=..., XB=..., PERIODIC_MESH_IDS(1)='mesh3' /
&MESH ID='mesh2', IJK=..., XB=... /
&MESH ID='mesh3', IJK=..., XB=..., PERIODIC_MESH_IDS(1)='mesh1' /
```

Note that `PERIODIC_MESH_IDS` is an array of three character strings because it is possible to have three periodic boundaries, one in each coordinate direction. In cases where the number of meshes is not large, say a few dozen, you can set `PROCESS_ALL_MESHES` to `.TRUE.` on the `MISC` line, in which case FDS will store the geometry of all meshes for all MPI processes, and in which case you need not specify

¹Note that the mirror image of a scene is *not* shown in Smokeview.

PERIODIC_MESH_IDS.

For additional information related to periodic boundaries and the pressure solver see Section 6.6.1.

Periodic vents may not be used to connect offset vents or vents in different coordinate directions. For such cases, you must employ HVAC capabilities (see Section 9.2).

Circular Vents

Circular or semi-circular vents may be specified as the intersection of a rectangle with coordinates XB and a circle with center XYZ and radius RADIUS. The rectangular surface cells that are assigned the corresponding SURF_ID will be those whose centroid falls within the intersection. In the example case called Fires/circular_burner.fds, the following two lines create a circular vent that is 1 m in diameter and flows propane gas at a rate of 0.02 kg/m²/s:

```
&SURF ID='BURNER', MASS_FLUX(1)=0.02, SPEC_ID(1)='PROPANE', TAU_MF(1)=0.01 /
&VENT XB=-0.6,0.6,-0.6,0.6,0,0, XYZ=0,0,0, RADIUS=0.5, SURF_ID='BURNER',
      SPREAD_RATE=0.05 /
```

The XB coordinates designate the orientation of the vent. In this case, the extent of the area specified by XB is large enough to contain the entire circle. Note also in this example that the parameter SPREAD_RATE causes the fire to spread outward at a rate of 0.05 m/s. The mass flux of propane through the vent is plotted in Fig. 7.1. Notice that the mass flux increases following a “t-squared” profile. This is what is expected of a fire which spreads radially at a linear rate. In this case, the fire reaches the RADIUS of the circle in 10 s, as expected. Note also that the parameter TAU_MF indicates that the fuel should ramp up quickly once the flame front reaches a given grid cell. In other words, TAU_MF controls the local ramp-up of fuel; the SPREAD_RATE controls the global ramp-up. Following the ramp-up, the fuel flows at a rate equal to the area of the circle times the mass flux of fuel per unit area. Even if the circle is crudely resolved on a coarse grid, the fuel flow rate will be adjusted to produce the desired value governed by the circular vent.

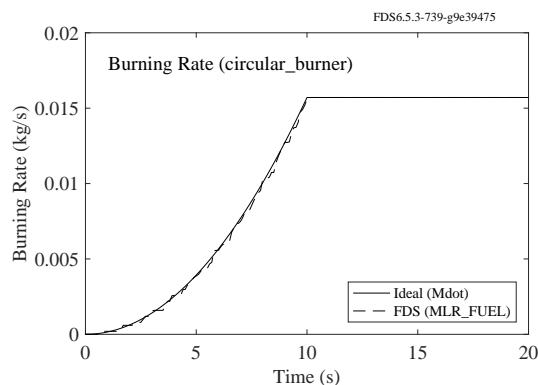


Figure 7.1: Results of the circular_burner test case.

7.3.3 Controlling Vents

VENT functionality can be controlled in some cases using “devices” and “controls,” specified via a DEVC_ID or a CTRL_ID. See Section 17.4.2 for details.

7.3.4 Trouble-Shooting Vents

Unlike most of the entries in the input file, the order that you specify `VENTS` can be important. There might be situations where it is convenient to position one `VENT` atop another. For example, suppose you want to designate the ceiling of a compartment to have a particular set of surface properties, and you designate the entire ceiling to have the appropriate `SURF_ID`. Then, you want to designate a smaller patch on the ceiling to have another set of surface properties, like an air supply. In this case, you must designate the supply `VENT` *first* because for that area of the ceiling, FDS will ignore the ceiling properties and apply the supply properties. FDS processes the first `VENT`, not the second as it did in versions prior to FDS 5. Now, the rule for `VENTS` is “first come, first served.” Keep in mind, however, that the second `VENT` is not rejected entirely – only where there is overlap. FDS will also print out a warning to the screen (or to standard error) saying which `VENT` has priority.

Smokeview can help identify where two `VENTS` overlap, assuming each has a unique `COLOR`. Because Smokeview draws `VENTS` on top of each other, areas of overlap will have a grainy, awkward appearance that changes pattern as you move the scene. In situations where you desire the overlap for the sake of convenience, you might want to slightly adjust the coordinates of the preferred `VENT` so that it is slightly offset from the solid surface. Make the offset less than about a tenth of a cell dimension so that FDS snaps it to its desired location. Then, by toggling the “q” key in Smokeview, you can eliminate the grainy color overlap by showing the `VENT` exactly where you specified it, as opposed to where FDS repositioned it. This trick also works where the faces of two obstructions overlap.

If an error message appears requesting that the orientation of a vent be specified, first check to make sure that the vent is a plane. If the vent is a plane, then the orientation can be forced by specifying the parameter `IOR`. If the normal direction of the `VENT` is in the positive x direction, set `IOR=1`. If the normal direction is in the negative x direction, set `IOR=-1`. For the y and z direction, use the number 2 and 3, respectively. Setting `IOR` may sometimes solve the problem, but it is more likely that if there is an error message about orientation, then the `VENT` is buried within a solid obstruction, in which case the program cannot determine the direction in which the `VENT` is facing.

7.4 Coloring Obstructions, Vents, Surfaces and Meshes

It is useful when visualizing the results of a simulation to assign to objects a meaningful color or pattern. There are two ways to do this in FDS. You can either assign a single color, or you can assign a texture map, which is essentially an image of a your choosing.

7.4.1 Colors

Colors for many items within FDS can be prescribed in two ways; a triplet of integer color values, `RGB`, or a character string, `COLOR`. The three `RGB` integers range from 0 to 255, indicating the amount of Red, Green and Blue that make up the color. If you define the `COLOR` by name, it is important that you type the name *exactly* as it is listed in the color tables. Color parameters can be specified on a `SURF` line, in which case all surfaces of that type will have that color, or color parameters can be applied directly to obstructions or vents. For example, the lines:

```
&SURF ID='UPHOLSTERY', ..., RGB=0,255,0 /  
&OBST XB=..., COLOR='BLUE' /
```

will color all `UPHOLSTERY` green and this particular obstruction blue. Table 7.1 provides a small sampling

of RGB values and COLOR names for a variety of colors². It is highly recommended that colors be assigned to surfaces via the SURF line because as the geometries of FDS simulations become more complex, it is very useful to use color as a spot check to determine if the desired surface properties have been assigned throughout the room or building under study.

Obstructions and vents may be colored individually, over-riding the color designated by the SURF line. The special case COLOR='INVISIBLE' causes the vent or obstruction not to be drawn by Smokeview. Another special case COLOR='RAINBOW' causes the color of the vent, obstruction or mesh to be randomly selected from the full range of RGB values; this can be useful if you are using the MULT namelist group and want to differentiate between the multiplied obstruction, vent or mesh.

7.4.2 Texture Maps

There are various ways of prescribing the color of various objects within the computational domain, but there is also a way of pasting images onto the obstructions for the purpose of making the Smokeview images more realistic. This technique is known as “texture mapping.” For example, to apply a wood paneling image to a wall, add to the SURF line defining the physical properties of the paneling the line:

```
&SURF ID='wood paneling',..., TEXTURE_MAP='paneling.jpg', TEXTURE_WIDTH=1.,
    TEXTURE_HEIGHT=2. /
```

Assuming that a JPEG file called paneling.jpg exists in the working directory, Smokeview should read it and display the image wherever the paneling is used. Note that the image does not appear when Smokeview is first invoked. It is an option controlled by the Show/Hide menu. The parameters TEXTURE_WIDTH and TEXTURE_HEIGHT are the physical dimensions of the image. In this case, the JPEG image is of a 1 m wide by 2 m high piece of paneling. Smokeview replicates the image as often as necessary to make it appear that the paneling is applied where desired. Consider carefully how the image repeats itself when applied in a scene. If the image has no obvious pattern, there is no problem with the image being repeated. If the image has an obvious direction, the real triplet TEXTURE_ORIGIN should be added to the VENT or OBST line to which a texture map should be applied. For example,

```
&OBST XB=1.,2.,3.,4.,5.,7., SURF_ID='wood paneling', TEXTURE_ORIGIN=1.,3.,5. /
```

applies paneling to an obstruction whose dimensions are 1 m by 1 m by 2 m, such that the image of the paneling is positioned at the point (1,3,5). The default value of TEXTURE_ORIGIN is (0,0,0), and the global default can be changed by added a TEXTURE_ORIGIN statement to the MISC line.























































7.5 Repeated Objects: The MULT Namelist Group (Table 19.15)

Sometimes obstructions, holes and vents are repeated over and over in the input file. This can be tedious to create and make the input file hard to read. However, if a particular set of objects repeats itself in a regular pattern, you can use a utility known as a multiplier. If you want to repeat an obstruction, for example, create a line in the input file as follows:

```
&MULT ID='m1', DX=1.2, DY=2.4, I_LOWER=-2, I_UPPER=3, J_LOWER=0, J_UPPER=5 /
&OBST XB=x1,x2,y1,y2,z1,z2, MULT_ID='m1' /
```

²A complete listing of all 500+ colors can be found by searching the FDS source code file data.f90.

Table 7.1: A sample of color definitions.

Name		R	G	B	Name		R	G	B
AQUAMARINE		127	255	212	MAROON		128	0	0
ANTIQUE WHITE		250	235	215	MELON		227	168	105
BEIGE		245	245	220	MIDNIGHT BLUE		25	25	112
BLACK		0	0	0	MINT		189	252	201
BLUE		0	0	255	NAVY		0	0	128
BLUE VIOLET		138	43	226	OLIVE		128	128	0
BRICK		156	102	31	OLIVE DRAB		107	142	35
BROWN		165	42	42	ORANGE		255	128	0
BURNT SIENNA		138	54	15	ORANGE RED		255	69	0
BURNT UMBER		138	51	36	ORCHID		218	112	214
CADET BLUE		95	158	160	PINK		255	192	203
CHOCOLATE		210	105	30	POWDER BLUE		176	224	230
COBALT		61	89	171	PURPLE		128	0	128
CORAL		255	127	80	RASPBERRY		135	38	87
CYAN		0	255	255	RED		255	0	0
DIMGRAY		105	105	105	ROYAL BLUE		65	105	225
EMERALD GREEN		0	201	87	SALMON		250	128	114
FIREBRICK		178	34	34	SANDY BROWN		244	164	96
FLESH		255	125	64	SEA GREEN		84	255	159
FOREST GREEN		34	139	34	SEPIA		94	38	18
GOLD		255	215	0	SIENNA		160	82	45
GOLDENROD		218	165	32	SILVER		192	192	192
GRAY		128	128	128	SKY BLUE		135	206	235
GREEN		0	255	0	SLATEBLUE		106	90	205
GREEN YELLOW		173	255	47	SLATE GRAY		112	128	144
HONEYDEW		240	255	240	SPRING GREEN		0	255	127
HOT PINK		255	105	180	STEEL BLUE		70	130	180
INDIAN RED		205	92	92	TAN		210	180	140
INDIGO		75	0	130	TEAL		0	128	128
IVORY		255	255	240	THISTLE		216	191	216
IVORY BLACK		41	36	33	TOMATO		255	99	71
KELLY GREEN		0	128	0	TURQUOISE		64	224	208
KHAKI		240	230	140	VIOLET		238	130	238
LAVENDER		230	230	250	VIOLET RED		208	32	144
LIME GREEN		50	205	50	WHITE		255	255	255
MAGENTA		255	0	255	YELLOW		255	255	0

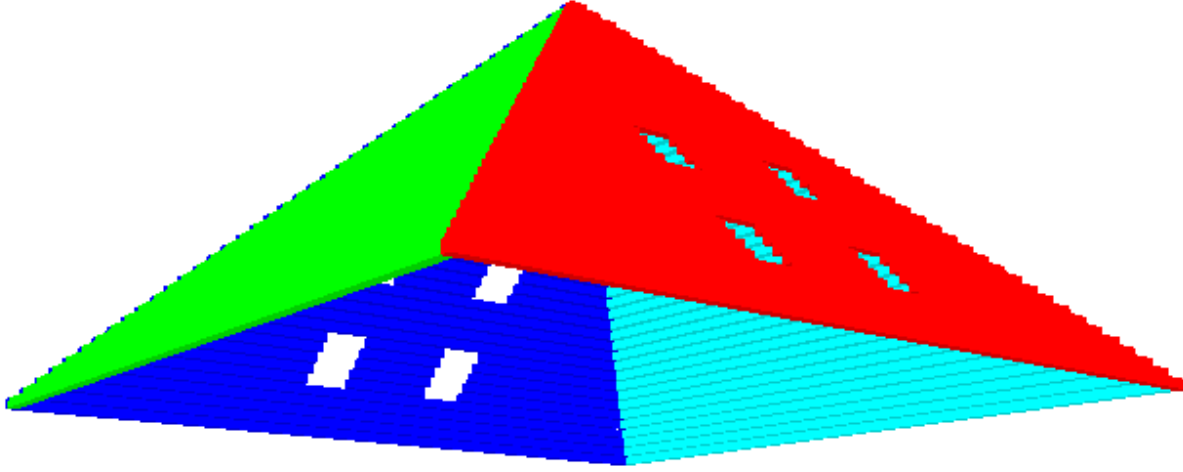


Figure 7.2: An example of the multiplier function.

This has the effect of making an array of obstructions according to the following formulae:

$$\begin{aligned}
 x1' &= x1 + DX0 + i \, DX & ; \quad I_LOWER \leq i \leq I_UPPER \\
 x2' &= x2 + DX0 + i \, DX & ; \quad I_LOWER \leq i \leq I_UPPER \\
 y1' &= y1 + DY0 + j \, DY & ; \quad J_LOWER \leq j \leq J_UPPER \\
 y2' &= y2 + DY0 + j \, DY & ; \quad J_LOWER \leq j \leq J_UPPER \\
 z1' &= z1 + DZ0 + k \, DZ & ; \quad K_LOWER \leq k \leq K_UPPER \\
 z2' &= z2 + DZ0 + k \, DZ & ; \quad K_LOWER \leq k \leq K_UPPER
 \end{aligned}$$

In situations where the position of the obstruction needs shifting prior to the multiplication, use the parameters $DX0$, $DY0$, and $DZ0$.

A variation of this idea is to replace the parameters, DX , DY , and DZ , with a sextuplet called DXB . The six entries in DXB increment the respective values of the obstruction coordinates given by XB . For example, the x coordinates are transformed as follows:

$$\begin{aligned}
 x1' &= x1 + DX0 + n \, DXB(1) & ; \quad N_LOWER \leq n \leq N_UPPER \\
 x2' &= x2 + DX0 + n \, DXB(2) & ; \quad N_LOWER \leq n \leq N_UPPER
 \end{aligned}$$

Notice that we use `N_LOWER` and `N_UPPER` to denote the range of `N`. This more flexible input scheme allows you to create, for example, a slanted roof in which the individual roof segments shorten as they ascend to the top. This feature is demonstrated by the following short input file that creates a hollowed out pyramid using the four perimeter obstructions that form the outline of its base:

```
&HEAD CHID='pyramid', TITLE='Simple demo of multiplier function' /
&MESH IJK=100,100,100, XB=0.0,1.0,0.0,1.0,0.0,1.0 /
&TIME T_END=0. /
&MULT ID='south', DXB=0.01,-.01,0.01,0.01,0.01,0.01, N_LOWER=0, N_UPPER=39 /
&MULT ID='north', DXB=0.01,-.01,-.01,-.01,0.01,0.01, N_LOWER=0, N_UPPER=39 /
&MULT ID='east', DXB=-.01,-.01,0.01,-.01,0.01,0.01, N_LOWER=0, N_UPPER=39 /
&MULT ID='west', DXB=0.01,0.01,0.01,-.01,0.01,0.01, N_LOWER=0, N_UPPER=39 /
&OBST XB=0.10,0.90,0.10,0.11,0.10,0.11, MULT_ID='south', COLOR='RED' /
&OBST XB=0.10,0.90,0.89,0.90,0.10,0.11, MULT_ID='north', COLOR='BLUE' /
&OBST XB=0.10,0.11,0.11,0.89,0.10,0.11, MULT_ID='west', COLOR='GREEN' /
&OBST XB=0.89,0.90,0.11,0.89,0.10,0.11, MULT_ID='east', COLOR='CYAN' /
&MULT ID='holes', DX=0.15, DZ=0.1, I_UPPER=1, K_UPPER=1 /
&HOLE XB=0.40,0.45,0.00,1.00,0.15,0.20, MULT_ID='holes' /
&TAIL /
```

The end result of this input file is to create a pyramid by repeating long, rectangular obstructions at the base of each face in a stair-step pattern. Note in this case the use of `N_LOWER` and `N_UPPER` which automatically cause FDS to repeat the obstructions in sequence rather than as an array.

Note that the `MULTIPLICATION` functionality works for `MESH`, `OBST`, `HOLE`, `VENT`, and `INIT` lines. For a `MESH`, it only applies to the bounds (`XB`) of the mesh, not the number of cells.

Note also that if a `COLOR` is specified on a line that includes a `MULT_ID`, this color will be applied to all replicates of the object. However, you can set `COLOR='RAINBOW'` which will instruct FDS to randomly choose a color for each replicate object.

Chapter 8

Fire and Thermal Boundary Conditions

This chapter describes how to specify the thermal properties of solid objects. This is the most challenging part of setting up the simulation. Why? First, for both real and simulated fires, the growth of the fire is very sensitive to the thermal properties of the surrounding materials. Second, even if all the material properties are known to some degree, the physical phenomena of interest may not be simulated properly due to limitations in the model algorithms or resolution of the numerical mesh. It is your responsibility to supply the thermal properties of the materials, and then assess the performance of the model to ensure that the phenomena of interest are being captured.

8.1 Basics

By default, the outer boundary of the computational domain is assumed to be a solid boundary that is maintained at ambient temperature. The same is true for any obstructions that are added to the scene. To specify the properties of solids, use the namelist group `SURF` (Section 7.1). Solids are assumed to consist of layers that can be made of different materials. The properties of each material required are designated via the `MATL` namelist group (Section 8.3). These properties indicate how rapidly the materials heat up, and how they burn. Each `MATL` entry in the input file must have an `ID`, or name, so that they may be associated with a particular `SURF` via the parameter `MATL_ID`. For example, the input file entries:

```
&MATL ID='BRICK', CONDUCTIVITY=0.69, SPECIFIC_HEAT=0.84, DENSITY=1600. /  
&SURF ID='BRICK WALL', MATL_ID='BRICK', COLOR='RED', BACKING='EXPOSED',  
      THICKNESS=0.20 /  
&OBST XB=0.1,5.0,1.0,1.2,0.0,1.0, SURF_ID='BRICK WALL' /
```

define a brick wall that is 4.9 m long, 1 m high, and 20 cm thick. Note that the thickness of the wall indicated by the `OBST` line is independent of the `THICKNESS` specified by the `SURF` line. The `OBST` line defines the geometry of the obstruction (i.e., how the obstruction is seen by the flow solver). The `SURF` line defines the heat transfer characteristics of the obstruction (i.e., how the obstruction is seen by the 1D solid phase solver). This allows an obstruction to snap to the local grid but still have the heat transfer solution reflect the actual thickness.

8.2 Surface Temperature and Heat Flux

This section describes how to specify simple thermal boundary conditions. These are often used when there is little or no information about the properties of the solid materials. If the properties of the materials are known, it is better to specify these properties and let the model compute the heat flux to, and temperature of, the walls and other solid surfaces.

8.2.1 Specified Solid Surface Temperature

Usually, the thermal properties of a solid boundary are specified via the `MATL` namelist group, which is in turn invoked by the `SURF` entry via the character string `MATL_ID`. However, sometimes it is convenient to specify a fixed temperature boundary condition, in which case set `TMP_FRONT` to be the surface temperature in units of °C:

```
&SURF ID='HOT WALL', COLOR='RED', TMP_FRONT=200. /
```

Note that there is no need to specify a `MATL_ID` or `THICKNESS`. Because the wall is to be maintained at the given temperature, there is no need to say anything about its material composition or thickness.

8.2.2 Special Topic: Convective Heat Transfer Options

This section is labeled as a special topic because normally you do not need to modify the convective heat transfer model in FDS. However, there are special cases for which the default model may not be adequate, and this section describes some options.

Default Convective Heat Transfer Model

In an LES calculation, the convective heat transfer coefficient, h , is based on a combination of natural and forced convection correlations:

$$\dot{q}_c'' = h(T_g - T_w) \quad \text{W/m}^2 \quad ; \quad h = \max \left[C|T_g - T_w|^{\frac{1}{3}}, \frac{k}{L} \text{Nu} \right] \quad \text{W}/(\text{m}^2 \cdot \text{K}) \quad (8.1)$$

where C is a empirical coefficient for natural convection (1.52 for a horizontal plate and 1.31 for a vertical plane or cylinder) [?], L is a characteristic length related to the size of the physical obstruction, and k is the thermal conductivity of the gas. The Nusselt number (Nu) depends on the geometric and flow characteristics. For many flow regimes, it has the form:

$$\text{Nu} = C_1 + C_2 \text{Re}^n \text{Pr}^m \quad ; \quad \text{Re} = \frac{\rho |\mathbf{u}| L}{\mu} \quad ; \quad \text{Pr} = 0.7 \quad (8.2)$$

For planar surfaces, the default values are $C_1 = 0$, $C_2 = 0.037$, $n = 0.8$, $m = 0.33$, and $L = 1$ m. For cylindrical surfaces, the default values are $C_1 = 0$, $C_2 = 0.683$, $n = 0.466$, $m = 0.33$, and $L = D$, the diameter of the cylinder. For spherical surfaces, the default values are $C_1 = 2$, $C_2 = 0.6$, $n = 0.5$, $m = 0.33$, and $L = D$, the diameter of the sphere. Note that for a sphere, the coefficient for natural convection, C , is assumed to be zero. It is possible to change these values for a particular application, but it is not possible to find a set of parameters that is appropriate for the wide variety of scenarios considered. Various correlations for planes, cylinders, and spheres can be found in Refs. [?, ?].

You can change the values of the empirical coefficient for natural convection, C , by specifying `C_HORIZONTAL` and `C_VERTICAL` on the `SURF` line. The length scale, L , is specified by `CONVECTION_LENGTH_SCALE` on the `SURF` line. You can change the empirical coefficients for the forced

convection model by using `C_FORCED_CONSTANT`, `C_FORCED_RE`, `C_FORCED_RE_EXP`, and `C_FORCED_PR_EXP` for the constants C_1 , C_2 , n , and m in the Nusselt number correlation, all of which are input on the `SURF` line.

Logarithmic Law of the Wall

Near-wall treatments, such as wall models or wall functions, aim to mimic the sudden change from molecular to turbulent transport close to the walls using algebraic formulations without the need of resolving the otherwise computationally expensive region of flow-field. The main theory follows dimensional analysis based on the idea that shear at the wall is constant. Accordingly, the non-dimensional velocity u^+ is calculated using a wall function [?].

By analogy, we define the non-dimensional temperature $T^+ \equiv (T_g - T_w)/T_\tau$, where T_g is the gas temperature of the first off-wall grid cell and T_τ is defined with the wall heat flux, \dot{q}_w'' , as $T_\tau = \dot{q}_w''/\rho_w u_\tau c_p$. The local heat transfer coefficient is then obtained from

$$h = \frac{\dot{q}_w''}{T_g - T_w} = \frac{\rho_w c_p u_\tau}{T^+} \quad (8.3)$$

Refer to the FDS Tech Guide [?] for further details of the formulation. To specify this heat transfer model for a particular surface, set `HEAT_TRANSFER_MODEL` equal to 'LOGLAW' on the `SURF` line.

Specified Convective Heat Transfer Coefficient

If you want to specify the convective heat transfer coefficient, you can set it to a constant using `HEAT_TRANSFER_COEFFICIENT` on the `SURF` line in units of $W/(m^2 \cdot K)$.

Specifying the Heat Flux at a Solid Surface

Instead of altering the convective heat transfer coefficient, you may specify a fixed heat flux directly. Two methods are available to do this. The first is to specify a `NET_HEAT_FLUX` in units of kW/m^2 . When this is specified FDS will compute the surface temperature required to ensure that the combined radiative and convective heat flux from the surface is equal to the `NET_HEAT_FLUX`. The second method is to specify separately the `CONVECTIVE_HEAT_FLUX`, in units of kW/m^2 , and the radiative heat flux. The radiative heat flux is specified by setting both `TMP_FRONT` and `EMISSIVITY` appropriately on the `SURF` line. Note that if you wish there to be only a convective heat flux from a surface, then the `EMISSIVITY` should be set to zero. If `NET_HEAT_FLUX` or `CONVECTIVE_HEAT_FLUX` is positive, the wall heats up the surrounding gases. If `NET_HEAT_FLUX` or `CONVECTIVE_HEAT_FLUX` is negative, the wall cools the surrounding gases.

8.2.3 Special Topic: Adiabatic Surfaces

For some special applications, it is often desired that a solid surface be adiabatic, that is, there is no net heat transfer (radiative and convective) from the gas to the solid. For this case, all that must be prescribed on the `SURF` line is `ADIABATIC=.TRUE.`, and nothing else. FDS will compute a wall temperature so that the sum of the net convective and radiative heat flux is zero. Specifying a surface as `ADIABATIC` will result in FDS defining `NET_HEAT_FLUX=0` and `EMISSIVITY=1`.

No solid surface is truly adiabatic; thus, the specification of an adiabatic boundary condition should be used for diagnostic purposes only.

8.3 Heat Conduction in Solids

Specified temperature or heat flux boundary conditions are easy to apply, but only of limited usefulness in real fire scenarios. In most cases, walls, ceilings and floors are made up of several layers of lining materials. The `MATL` namelist group is used to define the properties of the materials that make up boundary solid surfaces. A solid boundary can consist of multiple layers¹ of different materials, and each layer can consist of multiple material components.

8.3.1 Structure of Solid Boundaries

Material layers and components are specified on the `SURF` line via the array called `MATL_ID(IL, IC)`. The argument `IL` is an integer indicating the layer index, starting at 1, the layer at the exterior boundary. The argument `IC` is an integer indicating the component index. For example, `MATL_ID(2, 3) = 'BRICK'` indicates that the third material component of the second layer is `BRICK`. In practice, the materials are often listed as in the following example:

```
&MATL ID          = 'INSULATOR'
  CONDUCTIVITY    = 0.041
  SPECIFIC_HEAT   = 2.09
  DENSITY         = 229. /

&SURF ID          = 'BRICK WALL'
  MATL_ID         = 'BRICK', 'INSULATOR'
  COLOR           = 'RED'
  BACKING         = 'EXPOSED'
  THICKNESS       = 0.20, 0.10 /
```

Without arguments, the parameter `MATL_ID` is assumed to be a list of the materials in multiple layers, with each layer consisting of only a single material component.

When a set of `SURF` parameters is applied to the face of an `OBST`, the first `MATL_ID` defines the first layer of solid material. The other `MATL_ID`s are applied in succession. If `BACKING = 'EXPOSED'`, the last `MATL_ID` is applied to the opposite face of the `OBST`, assuming that the `OBST` is zero or one grid cells thick. If the `OBST` is thicker than one grid cell, then `BACKING = 'EXPOSED'` is not defined and it will be treated as if the condition `BACKING = 'VOID'` was set. If in the example above, `BRICK WALL` was applied to the entire `OBST` using `SURF_ID`, then when doing a heat transfer calculation from the $+x$ face to the $-x$ face, `FDS` would consider the `OBST` to be `BRICK` followed by `INSULATOR` and the same for a heat transfer calculation from the $-x$ face to the $+x$ face. To avoid this, specify a second `SURF` that has the reverse `MATL_ID` and use `SURF_ID6` to apply the two `SURF` definitions to opposite faces of the `OBST`.

Mixtures of solid materials within the same layer can be defined using the `MATL_MASS_FRACTION` keyword. This parameter has the same two indices as the `MATL_ID` keyword. For example, if the brick layer contains some additional water, the input could look like this:

```
&MATL ID          = 'WATER'
  CONDUCTIVITY    = 0.60
  SPECIFIC_HEAT   = 4.19
  DENSITY         = 1000. /

&SURF ID          = 'BRICK WALL'
  MATL_ID(1, 1:2) = 'BRICK', 'WATER'
  MATL_MASS_FRACTION(1, 1:2) = 0.95, 0.05
```

¹The maximum number of material layers is 20. The maximum number of material components is 20.

```

MATL_ID(2,1)      = 'INSULATOR'
COLOR             = 'RED'
BACKING           = 'EXPOSED'
THICKNESS         = 0.20,0.10 / <--- for layers 1 and 2

```

In this example, the first layer of material, Layer 1, is composed of a mixture of brick and water. This is given by the `MATL_ID` array which specifies Component 1 of Layer 1 to be brick, and Component 2 of Layer 1 to be water. The mass fraction of each is specified via `MATL_MASS_FRACTION`. In this case, brick is 95 %, by mass, of Layer 1, and water is 5 %.

It is important to notice that the components of the solid mixtures are treated as pure substances with no voids. The density of the mixture is

$$\rho = \left(\sum_i \frac{Y_i}{\rho_i} \right)^{-1} \quad (8.4)$$

where Y_i are the material mass fractions and ρ_i are the material bulk densities defined on the `MATL` lines. In the example above, the resulting density of the wall would be about 1553 kg/m³. The fact that the wall density is smaller than the density of pure brick may be confusing, but can be explained easily. If the wall can contain water, the whole volume of the wall can not be pure brick. Instead there are voids (pores) that are filled with water. If the water is taken away, there is only about 1476 kg/m³ of brick left. To have a density of 1600 kg/m³ for a partially void wall, a higher density should be used for the pure brick.

8.3.2 Thermal Properties

For any solid material, specify its thermal `CONDUCTIVITY` (W/(m·K)), `DENSITY` (kg/m³), `SPECIFIC_HEAT` (kJ/(kg·K)), and `EMISSIVITY` (0.9 by default). Both `CONDUCTIVITY` and `SPECIFIC_HEAT` can be functions of temperature. `DENSITY` and `EMISSIVITY` cannot. Temperature-dependence is specified using the `RAMP` convention. As an example, consider marinite, a wall material suitable for high temperature applications:

```

&MATL ID          = 'MARINITE'
  EMISSIVITY       = 0.8
  DENSITY          = 737.
  SPECIFIC_HEAT_RAMP = 'c_ramp'
  CONDUCTIVITY_RAMP = 'k_ramp' /
&RAMP ID='k_ramp', T= 24., F=0.13 /
&RAMP ID='k_ramp', T=149., F=0.12 /
&RAMP ID='k_ramp', T=538., F=0.12 /
&RAMP ID='c_ramp', T= 93., F=1.172 /
&RAMP ID='c_ramp', T=205., F=1.255 /
&RAMP ID='c_ramp', T=316., F=1.339 /
&RAMP ID='c_ramp', T=425., F=1.423 /

```

Notice that with temperature-dependent quantities, the `RAMP` parameter `T` means Temperature, and `F` is the value of either the specific heat or conductivity. In this case, neither `CONDUCTIVITY` nor `SPECIFIC_HEAT` is given on the `MATL` line, but rather the `RAMP` names.

The solid material can be given an `ABSORPTION_COEFFICIENT` (1/m) that allows the radiation to penetrate and absorb into the solid. Correspondingly, the emission of the material is based on the internal temperatures, not just the surface.

8.3.3 Back Side Boundary Conditions

The layers of a solid boundary are listed in order from the surface. By default, if the obstruction is less than or equal to one cell thick, then the innermost layer will be exposed to the air temperature on the back side. If the obstruction is on the boundary of the domain or is more than one cell thick, it is assumed to back up to an air gap at ambient temperature. For example a thin steel plate (i.e. thickness less than or equal to the grid) would use the FDS predicted temperatures on either side of the plate for predicting heat transfer.

There are other back side boundary conditions that can be applied. One is to assume that the wall backs up to an insulated material in which case no heat is lost to the backing material. The expression `BACKING='INSULATED'` on the `SURF` line prevents any heat loss from the back side of the material. Use of this condition means that you do not have to specify properties of the inner insulating material because it is assumed to be perfectly insulated.

If the wall is assumed to back up to the room on the other side of the wall and you want FDS to calculate the heat transfer through the wall into the space behind the wall, the attribute `BACKING='EXPOSED'` should be listed on the `SURF` line. This feature only works if the wall is less than or equal to one mesh cell thick, and if there is a non-zero volume of computational domain on the other side of the wall. Obviously, if the wall is an external boundary of the domain, the heat is lost to an ambient temperature void. The same happens if the back side gas cell cannot be found (in which case, the wall would not be one cell thick). This is the default boundary conditions.

If the wall is assumed to always back up to the ambient, then the attribute `BACKING='VOID'` should be set.

The back side emissivity of the surface can be controlled by specifying `EMISSIVITY_BACK` on the `SURF` line. If not specified, the back side emissivity will be calculated during the simulations as a mass-weighted sum of the `MATL` emissivities.

8.3.4 Initial and Back Side Temperature

By default, the initial temperature of the solid material is set to ambient (`TMPA` on the `MISC` line). Use `TMP_INNER` on the `SURF` line to specify a different initial temperature of the solid. The layers of the surface can have different initial temperatures. Also, the back side temperature boundary condition of a solid can be set using the parameter `TMP_BACK` on the `SURF` line. `TMP_BACK` is not the actual back side surface temperature, but rather the gas temperature to which the back side surface is exposed. This parameter has no meaning for surfaces with `BACKING='EXPOSED'` or `BACKING='INSULATED'`.

As an alternative to `TMP_INNER` one can also use `RAMP_T_I` to specify the name of a `RAMP` containing a depth vs. temperature profile for the surface.

Note that the parameters `TMP_INNER` and `TMP_BACK` are only meaningful for solids with specified `THICKNESS` and material properties (via the `MATL_ID` keyword).

8.3.5 Walls with Different Materials Front and Back

If you have an `OBST` that is one cell thick with gas cells on both sides (i.e., the obstruction is not at the edge of the domain) and you apply the attribute `BACKING='EXPOSED'`, then FDS calculates the heat conduction through the entire `THICKNESS`, and it uses the gas phase temperature and heat flux on the front and back sides for boundary conditions. A redundant calculation is performed on the opposite side of the obstruction. FDS always applies a `SURF` to an obstruction by having the first layer be the exposed surface of the face and the last layer as the opposite face. Take for example the `SURF` definition below and assume that the grid spacing is 10 cm. On the -x side of the `OBST`, layer 1 will be `MATERIAL A`, layer 2 will be `MATERIAL B`, and layer 3 will be the last `MATERIAL A`. On the +x side the `SURF` will be applied in the same manner.

```

&OBST XB=0.1,0.2,...., SURF_ID='SYMMETRIC'/
&SURF ID                = 'SYMMETRIC'
      COLOR              = 'ANTIQUÉ WHITE'
      BACKING            = 'EXPOSED'
      MATL_ID(1:3,1)    = 'MATERIAL A','MATERIAL B','MATERIAL A'
      THICKNESS(1:3)    = 0.1,0.2,0.1 /

```

For example, take the SURF definition below and assume that the grid spacing is 10 cm. On the -x side of the OBST, layer 1 will be MATERIAL A, layer 2 will be MATERIAL B, and layer 3 will be MATERIAL C. On the +x side, the SURF will be applied in the same manner, layer 1 will be MATERIAL A, layer 2 will be MATERIAL B, and layer 3 will be MATERIAL C. This means that both sides of the OBST will compute heat transfer assuming MATERIAL A is the first layer.

```

&OBST XB=0.1,0.2,...., SURF_ID='NON-SYMMETRIC'/
&SURF ID                = 'NON-SYMMETRIC'
      COLOR              = 'ANTIQUÉ WHITE'
      BACKING            = 'EXPOSED'
      MATL_ID(1:3,1)    = 'MATERIAL A','MATERIAL B','MATERIAL C'
      THICKNESS(1:3)    = 0.1,0.2,0.1 /

```

Therefore, if you apply the attribute BACKING='EXPOSED' on a SURF line that is applied to a zero or one-cell thick obstruction, you should be careful of how you specify multiple layers. If the layering is symmetric, the same SURF line can be applied to both sides. However, if the layering is not symmetric, you must create two separate SURF lines and apply one to each side. For example, a hollow box column that is made of steel and covered on the outside by a layer of insulation material and a layer of plastic on top of the insulation material, would have to be described with two SURF lines like the following:

```

&SURF ID                = 'COLUMN EXTERIOR'
      COLOR              = 'ANTIQUÉ WHITE'
      BACKING            = 'EXPOSED'
      MATL_ID(1:3,1)    = 'PLASTIC','INSULATION','STEEL'
      THICKNESS(1:3)    = 0.002,0.036,0.0063 /

&SURF ID                = 'COLUMN INTERIOR'
      COLOR              = 'BLACK'
      BACKING            = 'EXPOSED'
      MATL_ID(1:3,1)    = 'STEEL','INSULATION','PLASTIC'
      THICKNESS(1:3)    = 0.0063,0.036,0.002 /

```

If, in addition, the insulation material and plastic are combustible, and their burning properties are specified on the appropriate MATL lines, then you need to indicate which side of the column would generate the fuel vapor. In this case, the steel is impermeable; thus you should add the parameter LAYER_DIVIDE=2.0 to the SURF line labeled 'COLUMN EXTERIOR' to indicate that fuel vapors formed by the heating of the two first layers ('PLASTIC' and 'INSULATION') are to be driven out of that surface. You need to also specify LAYER_DIVIDE=0.0 on the SURF line labeled 'COLUMN INTERIOR' to indicate that no fuel vapors are to be driven into the interior of the column. In fact, values from 0.0 to 1.0 would work equally because the material 'STEEL' would not generate any fuel vapors.

By default, LAYER_DIVIDE is 0.5 times the number of layers for surfaces with EXPOSED backing, and equal to the number of layers for other surfaces.

8.3.6 Special Topic: Specified Internal Heat Source

The condensed phase heat conduction equation has a source term that describes the internal sources and sinks of energy. There are three types of sources that contribute to this term: heats of reaction for the pyrolysis (see Section 8.5), internal absorption and emission of radiation (see Section 8.5.7), and the source specified by the user. An example of the case where specified heat source could be needed is the heating of electrical cables due to internal current.

You can specify the internal source term for each layer of the surface using `INTERNAL_HEAT_SOURCE` on the `SURF` line. Its units are kW/m^3 and the default value is zero. In the example below, the cylindrical surface describing a cable consists of an outer plastic layer and inner core of metal. The metal core is heated with a power of 300 kW/m^3 .

```
&SURF ID = 'Cable'
      THICKNESS = 0.002,0.008
      MATL_ID(1,1) = 'PLASTIC'
      MATL_ID(2,1) = 'METAL'
      GEOMETRY = 'CYLINDRICAL'
      LENGTH = 0.1
      INTERNAL_HEAT_SOURCE = 0.,300. /
```

8.3.7 Special Topic: Non-Planar Walls and Targets

All obstructions in FDS are assumed to conform to the rectilinear mesh, and all bounding surfaces are assumed to be flat planes. However, many objects, like cables, pipes, and ducts, are not flat. Even though these objects have to be represented in FDS as “boxes,” you can still perform the internal heat transfer calculation as if the object were really cylindrical or spherical. For example, the input lines:

```
&OBST XB=0.0,5.0,1.1,1.2,3.4,3.5, SURF_ID='CABLE' /
&SURF ID='CABLE', MATL_ID='PVC', GEOMETRY='CYLINDRICAL', THICKNESS=0.01 /
```

can be used to model a power cable that is 5 m long, cylindrical in cross section, 2 cm in diameter. The heat transfer calculation is still one-dimensional; that is, it is assumed that there is a uniform heat flux all about the object. This can be somewhat confusing because the cable is represented as an obstruction of square cross section, with a separate heat transfer calculation performed at each face, and no communication among the four faces. Obviously, this is not an ideal way to do solid phase heat transfer, but it does provide a reasonable bounding surface temperature for the gas phase calculation. More detailed assessment of a cable would require a two or three-dimensional heat conduction calculation, which is not included in FDS. Use `GEOMETRY='SPHERICAL'` to describe a spherical object.

8.3.8 Special Topic: Solid Phase Numerical Gridding Issues

To compute the temperature and reactions inside the solids, FDS solves the one-dimensional heat transfer equation numerically. The size of the mesh cells on the surface of the solid is automatically chosen using a rule that makes the cell size smaller than the square root of the material diffusivity ($k/\rho c$). By default, the solid mesh cells increase towards the middle of the material layer and are smallest on the layer boundaries.

The default parameters are usually appropriate for simple heat transfer calculations but sometimes the use of pyrolysis reactions makes the temperatures and burning rate fluctuate. Adjustments may also be needed in case of extremely transient heat transfer situations. The numerical accuracy and stability of the solid phase solution may be improved by one of the following methods:

Make the mesh density more uniform inside the material by setting `STRETCH_FACTOR(NL)=1.` on the `SURF` line. This will generate a perfectly uniform mesh for layer number `NL`. (This happens automatically if the layer contains one or more reacting materials.) Values between 1 and 2 give different levels of stretching. Note that `STRETCH_FACTOR` needs to be specified for all the layers.

Make the mesh cells smaller by setting `CELL_SIZE_FACTOR` less than 1.0. For example, a value of 0.5 makes the mesh cells half the size. The scaling always applies to all layers.

Improve the time resolution by setting `WALL_INCREMENT=1` on the `TIME` line. This forces the solid phase temperatures to be solved on every time step.

Limit the number of cells in a layer by setting `N_LAYER_CELLS_MAX`. This array input has a default of 1000.

If all the material components of the surface are reacting, and the pyrolysis reactions have no solid residue, the thickness of the surface is going to shrink when the surface reacts. Each of the shrinking layers will vanish from the computation when its thickness gets smaller than a prescribed limiting value. This value can be set on a `SURF` line via `MINIMUM_LAYER_THICKNESS` keyword, defaulting to 1×10^{-6} m. When all the material of a shrinking surface is consumed but `BURN_AWAY` is not prescribed, the surface temperature is set to `TMP_BACK`, convective heat flux to zero and burning rate to zero.

See Section 8.6 for ways to check and improve the accuracy of the solid phase calculation.

8.3.9 Solid Heat Transfer 3D (Beta)

The conduction model described in the previous sections does not account for lateral heat transfer within a solid. Further, if a solid is immersed within a gas phase region it must be only one cell thick in order to communicate with neighboring gas phase regions for surface boundary conditions. These issues are addressed by the solid heat transfer 3D (`HT3D`) method, currently in beta testing in FDS.

`HT3D` solves the 3D heat equation within an `OBST` region of the domain. You may connect multiple `OBST` regions. The thermal properties of the material are uniform within a cell and are taken from a `MATL` tied to the `OBST` line. If you want layers of material, you must use a different `OBST` for each layer, and each layer must be at least one cell thick. The model uses the underlying gas phase grid resolution and explicit time integration that subcycles if necessary to achieve a stable solution to the heat equation. The advantage of using the `OBST` framework is that things like parallel capabilities and output methods (`DEVC` and `SLCF`) are automatically available without additional code requirements. `HT3D` is not currently hooked up to the pyrolysis model.

The method is invoked by adding `HT3D=.TRUE.` to an `OBST` line. The `OBST` line must then also have a `MATL_ID` associated. If two-way coupling with the gas phase is desired, then the `SURF` associated with the `OBST` should also have `HT3D=.TRUE.`. Note that if the FDS time step is too large then the accuracy of two-way coupling can be poor. You may need to set `DT` on the `TIME` line to be on the same order as the Von Neumann time step restriction based on the thermal diffusivity, $\alpha = k/(\rho c)$, of the solid (see also Sec. 6.4.9),

$$\delta t < \left[2\alpha \left(\frac{1}{\delta x^2} + \frac{1}{\delta y^2} + \frac{1}{\delta z^2} \right) \right]^{-1} \quad (8.5)$$

Below is an example input file section:

```
&MATL ID='steel', .../
&SURF ID='s1', HT3D=.TRUE./
&OBST XB=..., HT3D=.TRUE., MATL_ID='steel', SURF_ID='s1'/
```

8.4 Simple Pyrolysis Models

FDS has several approaches for describing the pyrolysis of solids and liquids. The approach to take depends largely on the availability of material properties and the appropriateness of the underlying pyrolysis model. Note that all pyrolysis models in FDS require that you explicitly define the gas phase reaction. See Chapter 13 for details.

8.4.1 A Gas Burner with a Specified Heat Release Rate

Solids and liquid fuels can be modeled by specifying their relevant properties via the `MATL` namelist group. However, if you simply want to specify a fire of a given heat release rate (HRR), you need not specify any material properties. A specified fire is basically modeled as the ejection of gaseous fuel from a solid surface or vent. This is essentially a burner, with a specified Heat Release Rate Per Unit Area, `HRRPUA`, in units of kW/m^2 . For example

```
&SURF ID='FIRE', HRRPUA=500. /
```

applies 500 kW/m^2 to any surface with the attribute `SURF_ID='FIRE'`. See the discussion of time-dependent quantities in Chapter 11 to learn how to ramp the heat release rate up and down.

An alternative to `HRRPUA` with the exact same functionality is `MLRPUA`, except this parameter specifies the Mass Loss Rate of fuel gas Per Unit Area in $\text{kg/(m}^2 \cdot \text{s)}$. Do not specify both `HRRPUA` and `MLRPUA` on the same `SURF` line. Neither of them can be used if the model contains multiple reactions.

8.4.2 Special Topic: A Radially-Spreading Fire

Sometimes it is desired that a fire spread radially at some specified rate. Rather than trying to obtain material properties to directly model the ignition and spread of the fire, you can specify the fire spread rate directly. First, you need to add a `SURF` line with a specified heat release rate, `HRRPUA`, and an optional time history parameter, `RAMP_Q` or `TAU_Q` (see Section 11.1). Then, you must specify `XYZ` and `SPREAD_RATE` on either a `VENT` or the same `SURF` line. The fire is directed to start at the point `XYZ` and spread radially at a rate of `SPREAD_RATE` (m/s). The optional ramp-up of the HRR begins at the time when the fire arrives at a given point. For example, the lines

```
&SURF ID='FIRE', HRRPUA=500.0, RAMP_Q='fireramp' /
&RAMP ID='fireramp', T= 0.0, F=0.0 /
&RAMP ID='fireramp', T= 1.0, F=1.0 /
&RAMP ID='fireramp', T=30.0, F=1.0 /
&RAMP ID='fireramp', T=31.0, F=0.0 /
&VENT XB=0.0,5.0,1.5,9.5,0.0,0.0, SURF_ID='FIRE', XYZ=1.5,4.0,0.0, SPREAD_RATE=0.03 /
```

create a rectangular area via the `VENT` line on which the fire starts at the point (1.5,4.0,0.0) and spreads outwards at a rate of 0.03 m/s. Each surface cell burns for 30 s as the fire spreads outward, creating a widening ring of fire. Note that the `RAMP_Q` is used to turn the burning on and off to simulate the consumption of fuel as the fire spreads radially. It should not be used to mimic a t -squared fire growth rate – the whole point of the exercise is to mimic this curve in a more natural way. Eventually, the fire goes out as the ring grows past the boundary of the rectangle. Some trial and error is probably required to find the `SPREAD_RATE` that leads to a desired time history of the heat release rate.

If you desire that the fire spread over an area that is not confined to a flat plane, specify `XYZ` and `SPREAD_RATE` on the `SURF` line directly and then apply that `SURF` line to the obstructions or particles over which you want the fire to spread. This technique can be useful for simulating the spread of fire through

a cluttered space when the detailed properties of the materials are unknown, or when the uncertainties associated with modeling the pyrolysis of the solid fuels directly are too great.

If the starting time of the simulation, `T_BEGIN`, is not zero, be aware that the default start time of the radially spreading fire is `T_BEGIN`, not zero. This is also true of `TAU_Q`, but it is not true of `RAMP_Q`. Because this might be confusing, if you start the calculation at a time other than zero, do a quick test to ensure that the ramps or fire spread behave as expected.

8.4.3 Solid Fuels that Burn at a Specified Rate

Real objects, like furnishings, office equipment, and so on, are often difficult to describe via the `SURF` and `MATL` parameters. Sometimes the only information about a given object is its bulk thermal properties, its “ignition” temperature, and its subsequent burning rate as a function of time from ignition. For this situation, add lines similar to the following:

```
&MATL ID              = 'stuff'
      CONDUCTIVITY     = 0.1
      SPECIFIC_HEAT    = 1.0
      DENSITY          = 900.0 /

&SURF ID              = 'my surface'
      COLOR            = 'GREEN'
      MATL_ID          = 'stuff'
      HRRPUA           = 1000.
      IGNITION_TEMPERATURE = 500.
      RAMP_Q           = 'fire_ramp'
      THICKNESS        = 0.01 /

&RAMP ID='fire_ramp', T= 0.0, F=0.0 /
&RAMP ID='fire_ramp', T= 10.0, F=1.0 /
&RAMP ID='fire_ramp', T=310.0, F=1.0 /
&RAMP ID='fire_ramp', T=320.0, F=0.0 /
```

An object with surface properties defined by ‘`my surface`’ shall burn at a rate of 1000 kW/m² after a linear ramp-up of 10 s following its “ignition” when its surface temperature reaches 500 °C. Burning shall continue for 5 min, and then ramp-down in 10 s. Note that the time `T` in the `RAMP` means time from ignition, not the time from the beginning of the simulation. Note also that now the “ignition temperature” is a surface property, not material property.

After the surface has ignited, the heat transfer into the solid is still calculated, but there is no coupling between the burning rate and the surface temperature. As a result, the surface temperature may increase too much. To account for the energy loss due to the vaporization of the solid fuel, `HEAT_OF_VAPORIZATION` can be specified for the surface. For example, when using the lines below, the net heat flux at the material surface is reduced by a factor 1000 kJ/kg times the instantaneous burning rate.

```
&SURF ID              = 'my surface'
      COLOR            = 'GREEN'
      MATL_ID          = 'stuff'
      HRRPUA           = 1000.
      IGNITION_TEMPERATURE = 500.
      HEAT_OF_VAPORIZATION = 1000.
      RAMP_Q           = 'fire_ramp'
      THICKNESS        = 0.01 /
```

The parameters `HRRPUA`, `IGNITION_TEMPERATURE`, and `HEAT_OF_VAPORIZATION` are all telling FDS

that you want to control the burning rate yourself, but you still want to simulate the heating up and “ignition” of the fuel. When these parameters appear on the `SURF` line, they are acting in concert. If `HRRPUA` appears alone, the surface will begin burning at the start of the simulation, like a piloted burner. The addition of an `IGNITION_TEMPERATURE` delays burning until your specified temperature is reached. The addition of `HEAT_OF_VAPORIZATION` tells FDS to account for the energy used to vaporize the fuel. For any of these options, if a `MATL` line is invoked by a `SURF` line containing a specified `HRRPUA`, then that `MATL` ought to have only thermal properties. The `MATL` line should have no reaction parameters, product yields, and so on, like those described in the previous sections. By specifying `HRRPUA`, you are controlling the burning rate rather than letting the material pyrolyze based on the conditions of the surrounding environment.

8.5 Complex Pyrolysis Models

This section describes the parameters that describe the reactions that occur within solid materials when they are burning. It is strongly recommended before reading this section that you read some background material on solid phase pyrolysis, for example “Thermal Decomposition of Polymers,” by Hirschler and Morgan, or “Flaming Ignition of Solid Fuels,” by Torero, both of which are in the 4th edition of the *SFPE Handbook of Fire Protection Engineering*.

8.5.1 Reaction Mechanism

A solid surface in FDS may consist of multiple layers with multiple material components per layer. The material components are described via `MATL` lines and are specified on the `SURF` line that describes the structure of the solid. Each `MATL` can undergo several reactions that may occur at different temperatures. It may not undergo any – it may just heat up. However, if it is to change form via one or more reactions, designate the number of reactions with the integer `N_REACTIONS`. It is very important that you designate `N_REACTIONS` or else FDS will ignore all parameters associated with reactions. Note that experimental evidence of multiple reactions does not imply that a single material is undergoing multiple reactions, but rather that multiple material components are undergoing individual reactions at distinct temperatures. Currently, the maximum number of reactions for each material is 10 and the chain of consecutive reactions may contain up to 20 steps.

For a given material, the j th reaction can produce other solid materials whose names are designated with `MATL_ID(i, j)`, and gas species whose names are designated with `SPEC_ID(i, j)`. Note that the index, i , runs from 1 to the number of material or gaseous species. This index does *not* correspond to the order in which the `MATL` or `SPEC` lines are listed in the input file. For a given reaction, the relative amounts of solid or gaseous products are input to FDS via the *yields*: `NU_MATL(i, j)` and `NU_SPEC(i, j)`, respectively. The yields are all zero by default. If `NU_MATL(i, j)` or `NU_SPEC(i, j)` is non-zero, then you *must* indicate what the solid residue is via `MATL_ID(j)`, the ID of another `MATL` that is also listed in the input file. Ideally, the sum of the yields should add to 1, meaning that the mass of the reactant is conserved. However, there are times when it is convenient to have the yields sum to something less than one. For example, the spalling or ablation of concrete can be described as a “reaction” that consumes energy but does not produce any “product” because the concrete is assumed to have either fallen off the surface in chunks or pulverized powder. The concrete’s mass is not conserved *in the model* because it has essentially disappeared from that particular surface.

For consistency, the `HEAT_OF_COMBUSTION(j)` can also be specified for each reaction, j . These values are used only if the corresponding heats of combustion for the gaseous species are greater than zero.

In the example below, the pyrolysis of wood is included within a simulation that uses a finite-rate reaction instead of the default mixing-controlled model. Notice in this case that all of the gas species

(except for the background nitrogen) are explicitly defined, and as a result, FDS needs to be told explicitly what gaseous species are produced by the solid phase reactions. In this case, 82 % of the mass of wood is converted to gaseous 'PYROLYZATE' and 18 % is converted to solid 'CHAR'.

```
&SPEC ID = 'PYROLYZATE', MW=53.6 /
&SPEC ID = 'OXYGEN', MASS_FRACTION_0 = 0.23 /
&SPEC ID = 'WATER VAPOR' /
&SPEC ID = 'CARBON DIOXIDE' /

&MATL ID              = 'WOOD'
  EMISSIVITY           = 0.9
  CONDUCTIVITY          = 0.2
  SPECIFIC_HEAT         = 1.3
  DENSITY               = 570.
  N_REACTIONS           = 1
  A(1)                  = 1.89E10
  E(1)                  = 1.51E5
  N_S(1)                = 1.0
  MATL_ID(1,1)          = 'CHAR'
  NU_MATL(1,1)          = 0.18
  SPEC_ID(1:4,1)        = 'PYROLYZATE', 'OXYGEN', 'WATER VAPOR', 'CARBON DIOXIDE'
  NU_SPEC(1:4,1)        = 0.82, 0, 0, 0
  HEAT_OF_REACTION(1)   = 430.
  HEAT_OF_COMBUSTION(1) = 14500. /
```

Note that the indices associated with the parameters are not needed *in this case*, but they are shown to emphasize that, in general, there can be multiple reactions with corresponding kinetic parameters and products.

8.5.2 Reaction Rates

For each reaction that each material component undergoes you must specify kinetic parameters of the reaction rate. The general evolution equation for a material undergoing one or more reactions is:

$$\frac{dY_{s,i}}{dt} = - \sum_{j=1}^{N_{r,i}} r_{ij} + \sum_{i'=1}^{N_m} \sum_{j=1}^{N_{r,i'}} \nu_{s,i'j} r_{i'j} \quad (i' \neq i) \quad (8.6)$$

where

$$r_{ij} = A_{ij} Y_{s,i}^{n_{s,ij}} \exp\left(-\frac{E_{ij}}{RT_s}\right) X_{O_2}^{n_{O_2,ij}} \quad ; \quad Y_{s,i} = \left(\frac{\rho_{s,i}}{\rho_s(0)}\right) \quad (8.7)$$

The term, r_{ij} , defines the rate of reaction at the temperature, T_s , of the i th material undergoing its j th reaction. The second term on the right of the equation (8.6) represents the contributions of other materials producing the i th material as a residue with a yield of $\nu_{s,i'j}$. This term is denoted by $NU_MATL(:, j)$ on the i' -th `MATL` line. $\rho_{s,i}$ is the density of the i th material component of the layer, defined as the mass of the i th material component divided by the volume of the layer. $\rho_s(0)$ is the initial density of the layer. Thus, $Y_{s,i} = \rho_{s,i}/\rho_s(0)$ is a quantity that increases if the i th material component is produced as a residue of some other reaction, or decreases if the i th component decomposes. If the layer is composed of only one material, then $\rho_{s,i}/\rho_s(0)$ is initially 1. $n_{s,ij}$ is the reaction order and prescribed under the name $N_S(j)$, and is 1 by default. If the value of n_s is not known, it is a good starting point to assume it is 1.

The pre-exponential factor, A_{ij} , is prescribed under the name $A(j)$ on the `MATL` line of the i th material, with units of s^{-1} . E_{ij} , the activation energy, is prescribed via $E(j)$ in units of kJ/kmol. Remember that 1 kcal is 4.184 kJ, and be careful with factors of 1000. For a given reaction, specify both A and E , or neither.

Do not specify only one of these two parameters. Typically, these parameters only have meaning when both are derived from a common set of experiments, like TGA (thermogravimetric analysis).

The fourth term of the reaction rate equation (8.7) can be used to simulate oxidation reactions. If the heterogeneous reaction order $n_{O_2,ij}$ is greater than zero, the reaction rate is affected by the local oxygen volume fraction, X_{O_2} . It is calculated from the gas phase (first grid cell) oxygen volume fraction $X_{O_2,g}$ by assuming simultaneous diffusion and consumption so that the concentration profile is in equilibrium, and the concentration at depth x is given by

$$X_{O_2}(x) = X_{O_2,g} \exp(-x/L_g) \quad (8.8)$$

where L_g is the gas diffusion length scale. $n_{O_2,ij}$ is prescribed under the name `N_O2 (j)` on the `MATL` line of the i th material. It is zero by default. L_g is prescribed under the name `GAS_DIFFUSION_DEPTH (j)`, and it is 0.001 m by default.

Estimating Kinetic Parameters

It is very important to keep in mind that the kinetic constants, A and E , are not available for most real materials. However, there is a way to model such materials using a simplified reaction scheme. The key assumption is that the material components can undergo only one reaction, at most. If, for example, the material undergoes three distinct reactions, there must be at least three material components, each of which undergoes one reaction. If there is an additional residue left over, then a fourth material component is needed.

In lieu of specifying A and E , there are several parameters that can be used by FDS to derive effective values, the most important of which is the `REFERENCE_TEMPERATURE` (°C). To understand this parameter, consider the plot shown in Fig. 8.1. These curves represent the results of a hypothetical TGA experiment in which a single component material undergoes a single reaction that converts the solid into a gas. The Mass Fraction (blue curve) is the normalized density of the material (Y_s) which decreases as the sample is slowly heated, in this case at a rate of 5 K/min. The Reaction Rate (green curve) is the rate of change of the mass fraction as a function of time ($-dY_s/dt$). Where this curve peaks is referred to in FDS as the `REFERENCE_TEMPERATURE`.² Note that the `REFERENCE_TEMPERATURE` is *not* the same as an ignition temperature, nor is it necessarily the surface temperature of the burning solid. Rather, it is simply the temperature at which the mass fraction of the material decreases at its maximum rate within the context of a TGA or similar experimental apparatus. Although you cannot specify an ignition temperature, you can specify a threshold temperature, see Section 8.5.6 for details.

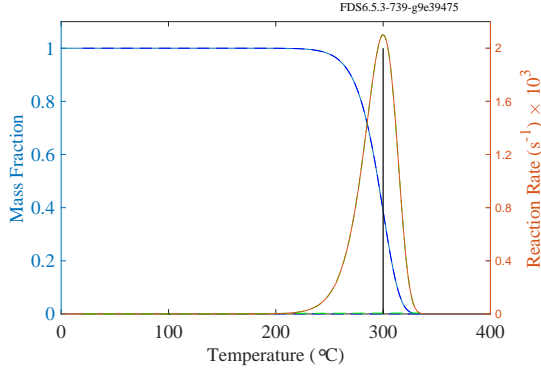
The kinetic constants for component i of a multi-component solid are given by³:

$$E_{i,1} = \frac{e r_{p,i}}{Y_{s,i}(0)} \frac{R T_{p,i}^2}{\dot{T}} \quad ; \quad A_{i,1} = \frac{e r_{p,i}}{Y_{s,i}(0)} e^{E/R T_{p,i}} \quad (8.9)$$

where $T_{p,i}$ and $r_{p,i}/Y_{s,i}(0)$ are the reference temperature and rate, respectively. The `REFERENCE_RATE` is the reaction rate, in units of s^{-1} , at the given `REFERENCE_TEMPERATURE` divided by the mass fraction, $Y_{s,i}(0)$, of material in the original sample undergoing the reaction. For a single component, single reaction material, $Y_{s,1}(0) = 1$. The `HEATING_RATE` (\dot{T}) is the rate at which the temperature of the TGA (or equivalent) test apparatus was increased. It is input into FDS in units of K/min (in the formula, it is expressed in K/s). Its default value is 5 K/min. In Fig. 8.1, the area under the green curve (Reaction Rate) is equal to the heating rate (in units of K/s).

²The term “reference temperature” is used simply to maintain backward compatibility with earlier versions of FDS.

³These formulas have been derived from an analysis that considers a first-order reaction. When using the proposed method, do not specify non-unity value for the reaction order `N_S` on the `MATL` line.



$$\frac{dY_s}{dt} = -A Y_s \exp(-E/RT_s) \quad Y_s(0) = 1$$

$$\begin{aligned} T_p &= 300 \text{ }^\circ\text{C} \\ r_p &= 0.002 \text{ s}^{-1} \\ \dot{T} &= 5 \text{ K/min} \\ v_s &= 0 \end{aligned}$$

Figure 8.1: The blue curve represents the normalized mass, $Y_s = \rho_s/\rho_s(0)$, of a solid material undergoing heating at a rate of 5 K/min. The green curve represents the reaction rate, $-dY_s/dt$. The ordinary differential equation that describes the transformation is shown at right. Note that the parameters T_p , r_p , and v_s represent the “reference” temperature, reaction rate, and residue yield of the single reaction. From these parameters, values of A and E can be estimated using the formulae in (8.9). The full set of parameters for this case are listed in `pyrolysis_1.fds`.

There are many cases where it is only possible to estimate the `REFERENCE_TEMPERATURE` (T_p) of a particular reaction because micro-scale calorimetry data is unavailable. In such cases, an additional parameter can be specified to help fine tune the shape of the reaction rate curve, assuming some sort of measurement or estimate has been made to indicate at what temperature, and over what temperature range, the reaction takes place. The `PYROLYSIS_RANGE` (ΔT) is the approximate width (in degrees Celsius or Kelvin) of the green curve, assuming its shape to be roughly triangular. Its default value is 80 °C. Using these input parameters, an estimate is made of the peak reaction rate, $r_{p,i}$, with which $E_{i,1}$, then $A_{i,1}$, are calculated.

$$\frac{r_{p,i}}{Y_{s,i}(0)} = \frac{2\dot{T}}{\Delta T} (1 - v_{s,i}) \quad (8.10)$$

The parameter, $v_{s,i}$, is the yield of solid residue.

When in doubt about the values of these parameters, just specify the `REFERENCE_TEMPERATURE`. Note that FDS will automatically calculate A and E using the above formulae. Do not specify A and E if you specify the `REFERENCE_TEMPERATURE`, and do not specify `PYROLYSIS_RANGE` if you specify `REFERENCE_RATE`. For the material decomposition shown in Fig. 8.1, the `MATL` would have the form:

```
&MATL ID = 'My Fuel'
...
N_REACTIONS = 1
SPEC_ID(1,1) = '...'
NU_SPEC(1,1) = 1.
REFERENCE_TEMPERATURE(1) = 300.
REFERENCE_RATE(1) = 0.002
HEATING_RATE(1) = 5.
HEAT_OF_COMBUSTION(1) = ...
HEAT_OF_REACTION(1) = ... /
```

Note that the indices have been added to the reaction parameters to emphasize the fact that these parameters are stored in arrays of length equal to `N_REACTIONS`. If there is only one reaction, you need not include the (1), but it is a good habit to get into. Note also that if the default combustion model is used, you can denote that the reaction produces fuel gas using the appropriate `SPEC_ID`. Note also that

the `HEAT_OF_COMBUSTION` is the energy released per unit mass of fuel gas that mixes with oxygen and combusts. This has nothing to do with the pyrolysis process, so why is it specified here? The answer is that there can be only one *gas phase* reaction of fuel and oxygen in FDS, but there can be dozens of different materials and dozens of *solid phase* reactions. To ensure that the fuel vapors from different materials combust to produce the proper amount of energy, it is very important to specify a `HEAT_OF_COMBUSTION` for each material. That way, the mass loss rate of fuel gases is automatically adjusted so that the effective mass loss rate multiplied by the single, global, gas phase heat of combustion produces the expected heat release rate. If, for example, the `HEAT_OF_COMBUSTION` specified on the `REAC` line is twice that specified on the `MATL` line, the mass of contained within wall cell will be decremented by that determined by the pyrolysis model, but the mass added to gas phase would be reduced by 50 %. A different value of heat of combustion can be specified for each reaction, j , via the parameter `HEAT_OF_COMBUSTION(j)`.

Modeling Upholstered Furnishings

The example input files called `Fires/couch.fds` and `Fires/room_fire.fds` demonstrate a simple way to model upholstered furniture. In residential fires, upholstered furniture makes up a significant fraction of the combustible load. A single couch can generate several megawatts of energy and sometimes lead to compartment flashover. Modeling a couch fire requires a simplification of its structure and materials. At the very least, we want the upholstery to be modeled as a layer of fabric covering polyurethane foam. We need the thermal properties of each, along with estimates of the “reference” temperatures as described above. The foam might be described as follows:

```
&MATL ID                = 'FOAM'
  SPECIFIC_HEAT          = 1.0
  CONDUCTIVITY           = 0.05
  DENSITY                = 40.0
  N_REACTIONS            = 1
  SPEC_ID                = 'FUEL'
  NU_SPEC                = 1.
  REFERENCE_TEMPERATURE = 350.
  HEAT_OF_REACTION       = 1500.
  HEAT_OF_COMBUSTION     = 30000. /
```

Note that these properties are completely made up. Both the fabric and the foam decompose into fuel gases via single-step reactions. The fuel gases from each have different composition and heats of combustion. FDS automatically adjusts the mass loss rate of each so that the “effective” fuel gas is that which is specified on the `REAC` line. Figure 8.2 shows the fire after 10 min. Only the reaction zone of the fire is shown; the smoke is hidden so that you can see the fire progressing along the couch.

8.5.3 Shrinking and swelling materials

Many practical materials change in thickness during the thermal reactions. For example:

- Non-charring materials will shrink as material is removed from the condensed phase to the gas phase.
- Porous materials like foams would shrink when the material melts and forms a non-porous layer.
- Some charring materials swell, i.e., get thicker, when a porous char layer is formed.
- Intumescent fire protection materials would swell significantly, creating an insulating layer.

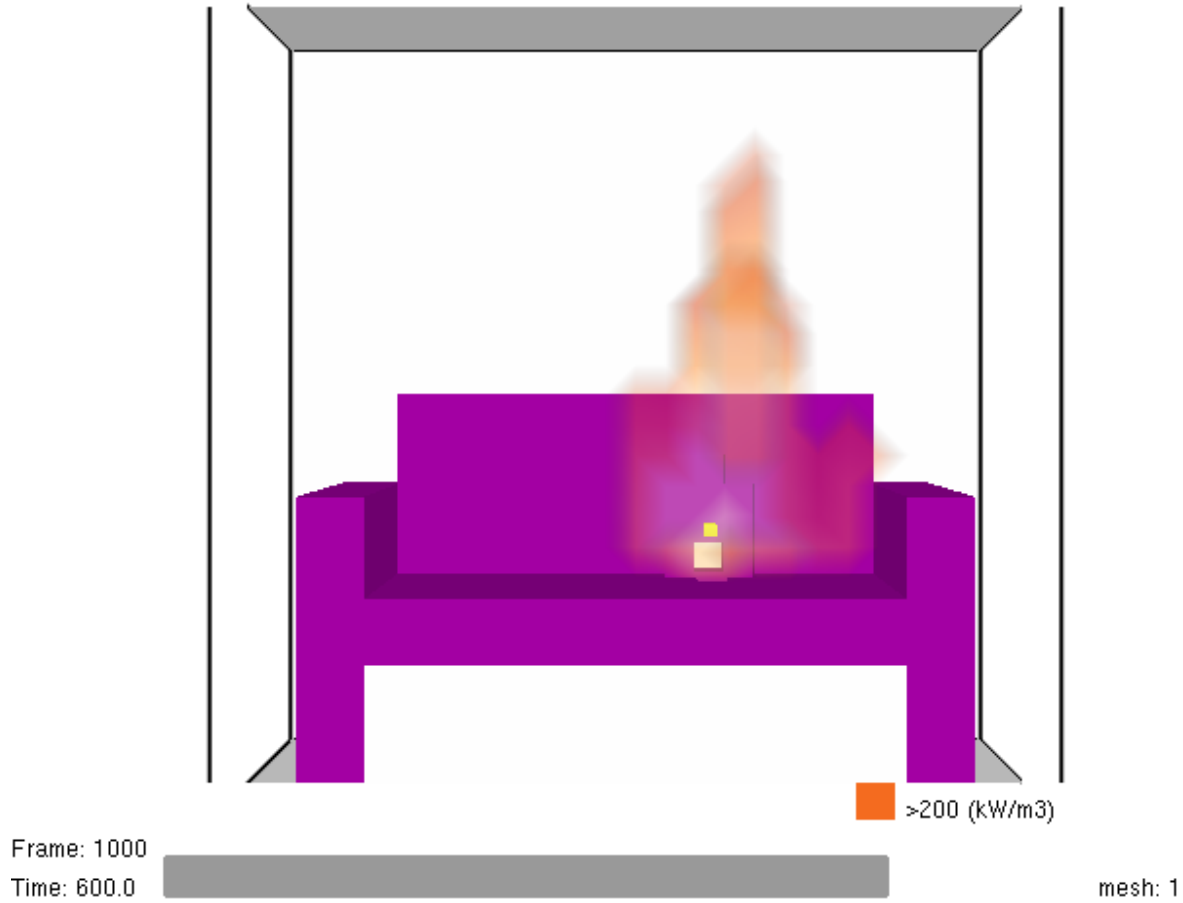


Figure 8.2: Output of `couch` test case showing fire on the couch at 10 min.

In FDS, the layer thickness is updated according to the ratio of the instantaneous material density and the density of the material in its pure form, i.e., the `DENSITY` on the `MATL` line. In cases involving several material components, the amount of swelling and shrinking is determined by the maximum and sum of these ratios, respectively. In mathematical terms, this means that in each time step the size of each condensed phase cell is changed according to the ratio δ

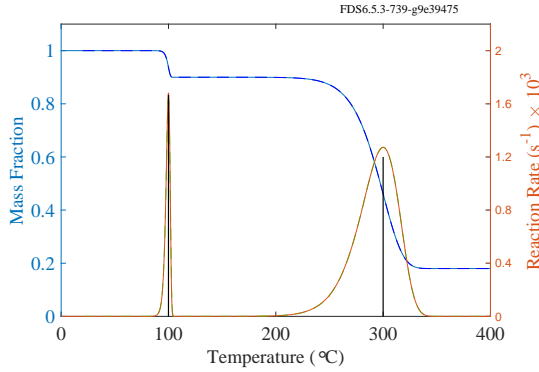
$$\delta = \begin{cases} \max_i \left(\frac{\rho_{s,i}}{\rho_i} \right) & \text{if } \max_i \left(\frac{\rho_{s,i}}{\rho_i} \right) \geq 1 \\ \sum_i \left(\frac{\rho_{s,i}}{\rho_i} \right) & \text{if } \max_i \left(\frac{\rho_{s,i}}{\rho_i} \right) < 1 \end{cases} \quad (8.11)$$

For example, if the original material with a `DENSITY` of 500 kg/m^3 is completely converted into a residue with a `DENSITY` of 1000 kg/m^3 , the thickness of the material layer will be half of the original.

You can prevent shrinking by setting `ALLOW_SHRINKING` to `.FALSE.` on the `MATL` line. You can prevent swelling by specifying `ALLOW_SWELLING` to `.FALSE.` on the `MATL` line. By default, these flags are true. Shrinking/swelling does not take place if any of the materials with non-zero density has the corresponding flag set to false.

8.5.4 Multiple Solid Phase Reactions

The solid phase reaction represented by Fig. 8.1 is fairly simple – a single, homogeneous material is heated and gasified completely. In general, real materials are not so simple. First, they consist of more than one material component, each of which can react over a particular temperature interval, and some of which leave behind a solid residue. Some material components may even undergo multiple reactions that form different residues, like woods that form various amounts of tar, char, and ash, depending on the rate of heating. Figure 8.3 demonstrates a more complicated material than the one previously described. It is a hypothetical material that contains 10 % (by mass) water and 90 % solid material. The water evaporates in the neighborhood of 100 °C and the solid pyrolyzes in the neighborhood of 300 °C, leaving 20 % of its mass behind in the form of a solid residue. The full set of parameters for this case are listed in `pyrolysis_2.fds`.



$$\begin{aligned} \frac{dY_{s,1}}{dt} &= -A_{1,1} Y_{s,1} \exp(-E_{1,1}/RT_s) & Y_{s,1}(0) &= 0.1 \\ \frac{dY_{s,2}}{dt} &= -A_{2,1} Y_{s,2} \exp(-E_{2,1}/RT_s) & Y_{s,2}(0) &= 0.9 \\ \frac{dY_{s,3}}{dt} &= -v_{s,2,1} \frac{dY_{s,2}}{dt} & Y_{s,3}(0) &= 0.0 \\ T_{p,1,1} &= 100 + 273 \text{ K} & T_{p,2,1} &= 300 + 273 \text{ K} \\ r_{p,1,1} &= 0.0016 \text{ s}^{-1} & r_{p,2,1} &= 0.0012 \text{ s}^{-1} \\ v_{s,1,1} &= 0 & v_{s,2,1} &= 0.2 \\ \dot{T} &= 5 \text{ K/min} \end{aligned}$$

Figure 8.3: The blue curve represents the combined mass fraction, $\sum Y_{s,i}$, and the green curve the net reaction rate, $-d/dt(\sum Y_{s,i})$, for a material that contains 10 % water (by mass) that evaporates at a temperature of 100 °C, and 90 % solid material that pyrolyzes at 300 °C, leaving a 20 % (by mass) residue behind. Note that the numbered subscripts refer to the material component and the reaction, respectively. In this case, there are three material components, and the first two each undergo a single reaction. The third material component is formed as a residue of the reaction of the second material. The system of ordinary differential equations that governs the transformation of the materials is shown at right.

8.5.5 The Heat of Reaction

Equation (8.7) describes the rate of the reaction as a function of temperature. Most solid phase reactions require energy; that is, they are *endothermic*. The amount of energy consumed, per unit mass of reactant that is converted into reaction products, is specified by the `HEAT_OF_REACTION(j)`. Technically, this is the enthalpy difference between the products and the reactant. A positive value indicates that the reaction is *endothermic*; that is, the reaction takes energy out of the system. Usually the `HEAT_OF_REACTION` is accurately known only for simple phase change reactions like the vaporization of water. For other reactions, it must be determined empirically (e.g., by differential scanning calorimetry).

8.5.6 Special Topic: The “Threshold” Temperature

In FDS, the reaction rate expression in Eq. (8.7) includes an optional term:

$$r_{ij} = A_{ij} Y_{s,i}^{n_{s,ij}} \exp\left(-\frac{E_{ij}}{RT_s}\right) \max[0, S_{\text{thr},ij}(T_s - T_{\text{thr},ij})]^{n_{t,ij}} \quad (8.12)$$

$T_{thr,ij}$ is an optional “threshold” temperature that allows the definition of non-Arrhenius pyrolysis functions and ignition criteria, and is prescribed by `THRESHOLD_TEMPERATURE(j)`. $S_{thr,ij}$ is the “threshold direction” that allows the triggering of reaction when temperature gets “above” $T_{thr,ij}$ ($S_{thr,ij} = +1$) or “below” $T_{thr,ij}$ ($S_{thr,ij} = -1$). $n_{t,j}$ is prescribed under the name `N_T(j)` and $S_{thr,ij}$ under `THRESHOLD_SIGN`. By default, $T_{thr,ij}$ is -273.15 degrees Celsius, $n_{t,j}$ is zero and $S_{thr,ij} = +1$; thus, the last term of Equation 8.12 does not affect the pyrolysis rate. The term can be used to describe a threshold temperature for the pyrolysis reaction by setting $T_{thr,ij}$ and $n_{t,j} = 0$. Then the term is equal to 0 at temperatures below $T_{thr,ij}$ and 1 at temperatures above.

The threshold temperature can be used to simulate simple phase change reactions, such as melting and freezing. To make the reaction rate controlled by available energy, i.e., *not* kinetics, another optional term should be included in the reaction rate formula

$$r_{ij} = A_{ij} \frac{1}{H_{r,ij} \Delta t} \max [0, S_{thr,ij} (T_s - T_{thr,ij})]^{n_{t,ij}} \quad (8.13)$$

This form of reaction rate can be implemented by setting a logical parameter `PCR(j) = .TRUE.`. The pre-exponential factor A_{ij} should then be given a value that is close or slightly smaller than the specific heat (kJ/(kg · K)) of the material mixture at phase change temperature.

The input file `water_ice_water.fds` demonstrates the use of the “threshold” temperature. A small amount of liquid water at 10 °C is cooled down to -10 °C in 10 min, and then heated up again to 10 °C. The concentration of the liquid water as a function of temperature is plotted in Fig. 8.4. The cooling phase is indicated by the blue line and heating phase by the red line.

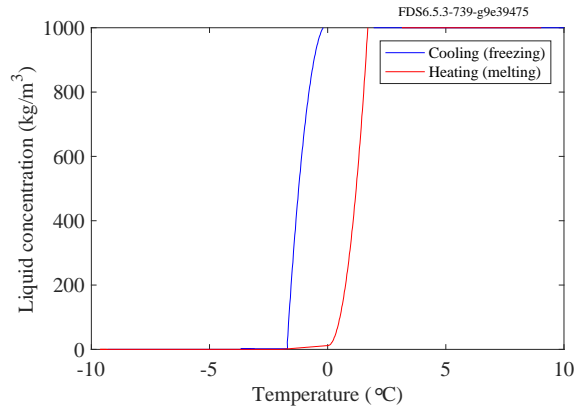


Figure 8.4: Freezing and melting of water.

8.5.7 Liquid Fuels

For a liquid fuel, the thermal properties are similar to those of a solid material, with a few exceptions. The evaporation rate of the fuel is governed by the mass transfer number (see FDS Technical Reference Guide for details). The properties of a liquid fuel are given on the `MATL` line:

```
&MATL ID           = 'ETHANOL LIQUID'
  EMISSIVITY        = 1.
  NU_SPEC           = 0.97
  SPEC_ID           = 'ETHANOL'
```

```

HEAT_OF_REACTION      = 836.98
CONDUCTIVITY           = 0.17
SPECIFIC_HEAT          = 2.4398
DENSITY                = 789.
BOILING_TEMPERATURE    = 78.5
ABSORPTION_COEFFICIENT = 1534.3 /

```

The inclusion of `BOILING_TEMPERATURE` on the `MATL` line tells FDS to use its liquid pyrolysis model. It also automatically sets `N_REACTIONS=1`, that is, the only “reaction” is the phase change from liquid to gaseous fuel. Thus, `HEAT_OF_REACTION` in this case is the latent heat of vaporization. The gaseous fuel yield, `NU_SPEC`, is 0.97 instead of 1 to account for impurities in the liquid that do not take part in the combustion process.

The thermal conductivity, density and specific heat are used to compute the loss of heat into the liquid via conduction using the same one-dimensional heat transfer equation that is used for solids. Obviously, the convection of the liquid is important, but is not considered in the model.

Note also the `ABSORPTION_COEFFICIENT` for the liquid. This denotes the absorption in depth of thermal radiation. Liquids do not just absorb radiation at the surface, but rather over a thin layer near the surface. Its effect on the burning rate is significant.

8.5.8 Fuel Burnout

The thermal properties of a solid or liquid fuel determine the length of time for which it can burn. In general, the burnout time is a function of the mass loss rate, \dot{m}'' , the density, ρ_s , and the layer thickness, δ_s :

$$t_b = \frac{\rho_s \delta_s}{\dot{m}''} \quad (8.14)$$

However, each type of pyrolysis model handles fuel burnout in a slightly different way. These differences will be highlighted in the individual sections below.

Solid Fuel Burnout When the Burning Rate is Specified

If you *specify* the burning rate using `HRRPUA` (Heat Release Rate Per Unit Area) or `MLRPUA` (Mass Loss Rate Per Unit Area), the solid surface will continue to burn at the specified rate indefinitely with no fuel burnout. You may use `RAMP_Q` to stop the burning at a desired time. You can estimate this “burnout time” for a surface using the heat of combustion, ΔH , material density, ρ_s , material thickness, δ_s , and `HRRPUA`, \dot{q}_f'' :

$$t_b = \frac{\rho_s \delta_s \Delta H}{\dot{q}_f''} \quad (8.15)$$

The burnout time is not calculated and applied automatically because there are instances where the underlying solid is not considered fuel; that is, the burning rate and duration are not determined by the composition of the solid surface.

Solid Fuel Burnout When the Burning Rate is Not Specified

If you are using a pyrolysis model in which the `MATL` lines have `N_REACTIONS` greater than or equal to 1, the burnout time of the pyrolyzing solid fuel is calculated automatically by FDS based on the layer `THICKNESS`, component `DENSITY`, and the calculated reactions rates of the materials.

Liquid Fuel Burnout

The burnout time of a liquid fuel is calculated automatically based on the liquid layer `THICKNESS`, liquid `DENSITY`, and the calculated burning rate.

Special topic: Making Fuels Disappear

If a burning object is to disappear from the simulation once it is consumed, set `BURN_AWAY = .TRUE.` on the corresponding `SURF` line. The solid object disappears from the calculation cell by cell, as the mass contained within each solid cell is consumed either by the pyrolysis reactions or by the prescribed `HRR`. The following issues should be kept in mind when using `BURN_AWAY`:

- Use `BURN_AWAY` parameter cautiously. If an object has the potential of burning away, a significant amount of extra memory has to be set aside to store additional surface information as the mesh cells disappear.
- If `BURN_AWAY` is prescribed, the `SURF` should be applied to the entire object, not just a face of the object because it is unclear how to handle edges of solid obstructions that have different `SURF_IDS` on different faces.
- If the volume of the obstruction changes because it has to conform to the uniform mesh, FDS does *not* adjust the burning rate to account for this as it does with various quantities associated with areas, like `HRRPUA`.
- A parameter called `BULK_DENSITY` (kg/m^3) can be applied to the `OBST` rather than the `SURF` line. This parameter is used to determine the *combustible* mass of the solid object. The calculation uses the user-specified object dimensions, not those of the mesh-adjusted object. This parameter overrides all other parameters from which a combustible mass would be calculated. Note that without a `BULK_DENSITY` specified, the total amount of mass burned will depend upon the grid resolution. The use of the `BULK_DENSITY` parameter ensures a specific fuel mass per unit volume that is independent of the grid resolution. Note that in the event that the solid phase reaction involves the production of solid residue (like char or ash), the `BULK_DENSITY` refers to the mass of the solid that is converted to gas upon reaction.
- The mass of the object is based on the densities of all material components (`MATL`), but it is only consumed by mass fluxes of the *known* species. If the sum of the gaseous yields is less than one, it will take longer to consume the mass.

Simple examples demonstrating how solid fuels can be forced to disappear from the domain are labeled `Fires/box_burn_away`. These are examples of a solid block of solid material that is pyrolyzed until it is completely consumed. The heat flux is generated by placing hot surfaces around the box. There is no combustion. In the first example, `box_burn_away1`, the released gas is (`'METHANE'`), and in the second example, `box_burn_away2`, it is an additional species called `'GAS'`. In the third and fourth examples `box_burn_away3` and `box_burn_away4`, the released gas is fuel but the pyrolysis rate is specified. In the fourth case, the heat of combustion for the foam material is set different from that of the gas, with a ratio of 0.75. The properties of the solid material were chosen simply to assure a quick calculation. The objective of the test is to check that the released mass and the integrated burning rate are consistent with the material properties of the block. The block is 0.4 m on a side, with a density of 20 kg/m^3 . The integrated densities of the pyrolysis product gases (written to `box_burn_away#_devc.csv`), as well as the integrated burning rate (written to `box_burn_away#_hrr.csv`) at the end of the 30 s calculation ought to be:

$$(0.4)^3 \text{ m}^3 \times 20 \text{ kg/m}^3 = 1.28 \text{ kg} \quad (8.16)$$

except for the fourth case, where the amount of released gas is affected by the ratio of heats of combustion

$$0.75 \times 1.28 \text{ kg} = 0.96 \text{ kg} \quad (8.17)$$

The same case is tested in two dimensions (`box_burn_away_2D` and `box_burn_away_2D_residue`). In the latter case, only half of the mass is converted to fuel, leaving behind a residue that is 50 % of the original mass. The box is forced to burn away by setting the `BULK_DENSITY` to 10 kg/m³. This is the *combustible* mass. These two cases exhibit a fictitious increase in solid mass when new unburned surfaces are exposed as entire mesh cells disappear. The increased mass is just an artifact of reporting the residual solid mass as the product of surface density and surface area. Both the final solid mass and the gaseous degradation products should match the expected values at the end of the simulation.

8.6 Testing Your Pyrolysis Model

Modeling the burning of real materials can be very complicated. Undoubtedly, the `SURF` and `MATL` lines in the input file will consist of a combination of empirical and fundamental properties, often originating from different sources. How do you know that the various property values and the associated thermo-physical model in FDS constitute an appropriate description of the solid? For a full-scale simulation, it is hard to untangle the uncertainties associated with the gas and solid phase routines. However, it is easy to perform a simple check of any set of solid phase model by essentially turning off the gas phase. In the following sections, guidance is provided on how to perform a quick simulation of the cone calorimeter and bench-scale measurements like thermal gravimetric analysis (TGA), differential scanning calorimetry (DSC), and micro-combustion calorimetry (MCC).

8.6.1 Simulating the Cone Calorimeter

This section describes how to set up a simple model of the cone calorimeter or other similar apparatus. This is not a full 3-D simulation of the apparatus, but rather a 1-D simulation of the solid phase degradation under an imposed external heat flux. You can literally create a model of the cone heater and sample holder in FDS to simulate the coupling of gas and solid phase phenomena, but before even attempting this, it is worthwhile to perform a quick simulation like the one described here to test the solid phase model only.

1. Create a trivially small mesh, just to let FDS run. Since the gas phase calculation is essentially being shut off, you just need 4 cells in each direction (`IJK=4, 4, 4`) for the pressure solver to function properly.
2. On the `TIME` line, set `WALL_INCREMENT=1` to force FDS to update the solid phase every time step (normally it does this every other time step), and set `DT` to whatever value appropriate for the solid phase calculation. Since there is no gas phase calculation that will limit the time step, it is best to control this yourself.
3. Set `HEAT_TRANSFER_COEFFICIENT=0` on the `SURF` line. This turns off the convective heat flux from gas to surface and vice versa. The heat flux to the solid is specified via `EXTERNAL_FLUX`⁴ (kW/m²) on the `SURF` line that is assigned to the solid surface.
4. Turn off all the gas phase computations by setting `SOLID_PHASE_ONLY=.TRUE.` on the `MISC` line. This will also speed up the computations significantly. If a `REAC` line is needed to define a fuel gas, you

⁴You can control `EXTERNAL_FLUX` using either `TAU_EF` or `RAMP_EF`. This is useful if you want to ramp up the heat flux following ignition to account for the additional radiation from the flame. See Section 11.1 for more details about ramps.

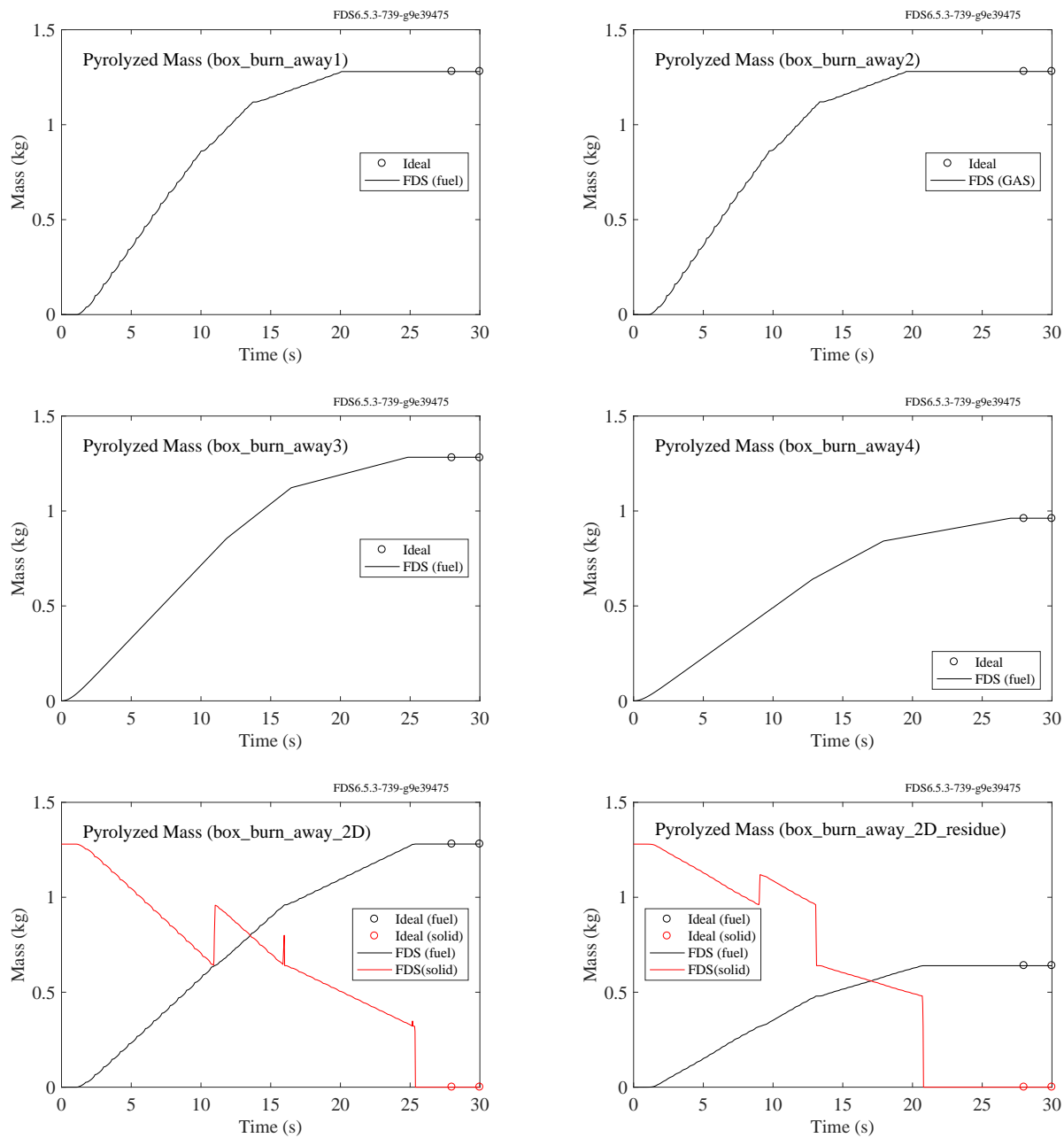


Figure 8.5: Output of box_burn_away test cases.

may turn off combustion by setting `Y_O2_INFTY=0.01` on the `MISC` line. This sets the background oxygen mass fraction to 0.01, too low to support any burning.

5. Generate `MATL` lines, plus a single `SURF` line, as you normally would, except add `EXTERNAL_FLUX` to the `SURF` line. This is simply a “virtual” source that heats the solid. Think of this as a perfect radiant panel or conical heating unit.
6. Assign the `SURF_ID` to a `VENT` that spans the bottom of the computational domain. Create `OPEN` vents on all other faces.
7. Finally, add solid phase output devices to the solid surface, like `'WALL TEMPERATURE'`, `'NET HEAT FLUX'`, `'BURNING RATE'`, `'GAUGE HEAT FLUX'`, and `'WALL THICKNESS'` (assuming the solid is to burn away). Use these to track the condition of the solid as a function of time. In particular, make sure that the `'BURNING RATE'` is appropriate for the particular external heat flux applied. Make sure that the `'WALL TEMPERATURE'` is appropriate. Compare your results to measurements made in a bench-scale device, like the cone calorimeter. Keep in mind, however, that the calculation and the experiment are not necessarily perfectly matched. The calculation is designed to eliminate uncertainties related to convection, combustion, and apparatus-specific phenomena.

Below is an FDS input file that demonstrates how you can test a candidate pyrolysis model by running very short calculations. The simulation only involves the solid phase model. Essentially, the gas phase calculation is shut off except for the imposition of a 52 kW/m^2 “external” heat flux. The solid in this example is a 8.5 mm thick slab of PMMA. For more details, see the FDS Validation Guide under the heading “FAA Polymers.”

```
&HEAD CHID='FAA_Polymers_PMMA', TITLE='Black PMMA at 50 kW/m2' /
&MESH IJK=3,3,4, XB=-0.15,0.15,-0.15,0.15,0.0,0.4 /
&TIME T_END=600., WALL_INCREMENT=1, DT=0.01 /
&MISC Y_O2_INFTY=0.01, SOLID_PHASE_ONLY=.TRUE. /
&REAC FUEL='METHANE' /
&MATL ID='BLACKPMMA'
    ABSORPTION_COEFFICIENT=2700.
    N_REACTIONS=1
    A(1) = 8.5E12
    E(1) = 188000
    EMISSIVITY=0.85
    DENSITY=1100.
    SPEC_ID='METHANE'
    NU_SPEC=1.
    HEAT_OF_REACTION=870.
    HEAT_OF_COMBUSTION=25200.
    CONDUCTIVITY = 0.20
    SPECIFIC_HEAT = 2.2
&SURF ID='PMMA SLAB'
    COLOR='BLACK'
    BACKING='INSULATED'
    MATL_ID='BLACKPMMA'
    THICKNESS=0.0085
    HEAT_TRANSFER_COEFFICIENT=0.
    EXTERNAL_FLUX=52 /
&VENT XB=-0.05,0.05,-0.05,0.05,0.0,0.0, SURF_ID = 'PMMA SLAB' /
&DUMP DT_DEVC=5. /
&DEVC XYZ=0.0,0.0,0.0, IOR=3, QUANTITY='WALL TEMPERATURE', ID='temp' /
&DEVC XYZ=0.0,0.0,0.0, IOR=3, QUANTITY='BURNING RATE', ID='MLR' /
&DEVC XYZ=0.0,0.0,0.0, IOR=3, QUANTITY='WALL THICKNESS', ID='thick' /
&TAIL /
```


8.6.2 Simulating Bench-scale Measurements like the TGA, DSC, and MCC

There are a number of techniques to measure the thermo-physical properties of a solid material. Most of these involve heating a very small sample at a relatively slow, linear rate. In this way, thermal conduction is minimized and the sample can be considered thermally-thin. FDS has a special feature that mimics thermogravimetric analysis (TGA), differential scanning calorimetry (DSC), and micro-combustion calorimetry (MCC) measurements. To use it, set up your input file as you normally would. Then, add the flag `TGA_ANALYSIS=.TRUE.` to the `SURF` line you want to analyze. You can only analyze one `SURF` line at a time. Optionally, you can specify `TGA_HEATING_RATE` (K/min) and `TGA_FINAL_TEMPERATURE` (°C) to indicate the linear heating rate and the final temperature of the sample. The default values are 5 K/min and 800 °C. The initial temperature is `TMPI`. Note that this feature is only appropriate for a `SURF` line that describes a thermally-thick sample consisting of a single layer with multiple reacting components. For example, the following `SURF` line describes a material that consists of three components:

```
&SURF ID = 'Cable Insulation'
      TGA_ANALYSIS = .TRUE.
      TGA_HEATING_RATE = 60.
      THICKNESS = 0.005
      MATL_ID(1,1) = 'Component A',
      MATL_ID(1,2) = 'Component B',
      MATL_ID(1,3) = 'Component C',
      MATL_MASS_FRACTION(1,1:3) = 0.26,0.33,0.41 /
```

The two TGA entries will force FDS to perform a numerical version of the TGA, DSC and MCC measurements. The `THICKNESS` and other boundary conditions on the `SURF` line will be ignored. After running the analysis, which only takes a second or two, FDS will then shut down without running the actual simulation. To run the simulation, either remove the TGA entries or set `TGA_ANALYSIS` to `.FALSE.`

The result of the `TGA_ANALYSIS` is a single comma-delimited file called `CHID_tga.csv`. The first and second columns of the file consist of the time and sample temperature. The third column is the normalized sample mass; that is, the sample mass divided by its initial mass. The fourth column is the mass loss rate, in units of s^{-1} . The fifth column is the heat release rate per unit mass of the sample in units of W/g, typical of an MCC measurement. The sixth column is the heat absorbed by the sample normalized by its mass, also in units of W/g, typical of a DSC measurement. Results for a typical analysis of wood are shown in Fig. 8.6. In this case, a sample of wood containing about 10 % water by mass heats up and undergoes three reactions, including the evaporation of water. Note that the TGA plots include both fuel and water vapor, while the MCC results only show fuel.

Details of the output quantities are discussed in Section 18.10.8. Further details on these measurement techniques and how to interpret them are found in the FDS Verification Guide [?].

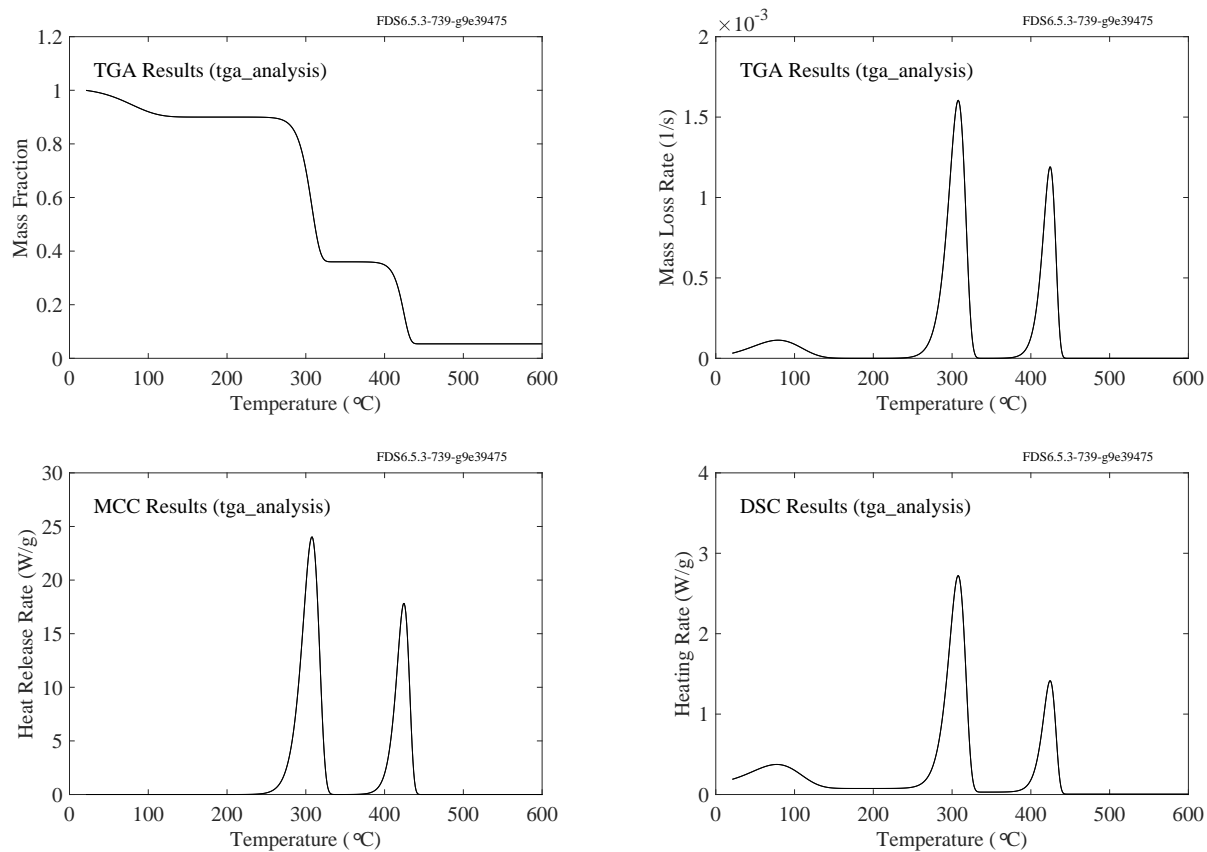


Figure 8.6: Sample results of a tga_analysis.

Chapter 9

Ventilation

This chapter explains how to model a ventilation system. There are two ways to do this. First, if you only want to specify air flow rates into and out of compartments, read Section 9.1 for a description of simple velocity boundary conditions. However, if you want to model the entire HVAC system, read Section 9.2.

9.1 Simple Vents, Fans and Heaters

The ventilation system of individual compartments within a building is described using velocity *boundary conditions*. For example, fresh air can be blown into, and smoke can be drawn from, a compartment by specifying a velocity in the *normal* direction to a solid surface. However, there are various other facets of velocity boundary conditions that are described below.

9.1.1 Simple Supply and Exhaust Vents

The easiest way to describe a supply or exhaust fan is to specify a VENT on a solid surface, and designate a SURF_ID with some form of specified velocity or volume flow rate. The normal component of velocity is usually specified directly via the parameter VEL. If VEL is negative, the flow is directed *into* the computational domain, i.e., a supply vent. If VEL is positive, the flow is drawn *out of* the domain, i.e., an exhaust vent. For example, the lines

```
&SURF ID='SUPPLY', VEL=-1.2, COLOR='BLUE' /  
&VENT XB=5.0,5.0,1.0,1.4,2.0,2.4, SURF_ID='SUPPLY' /
```

create a VENT that *supplies* air at a velocity of 1.2 m/s through an area of nominally 0.16 m², depending on the realignment of the VENT onto the FDS mesh. Regardless of the orientation of the plane $x = 5$, the flow will be directed *into* the room because of the sign of VEL. In this example the VENT may not be exactly 0.16 m² in area because it may not align exactly with the computational mesh. If this is the case then VOLUME_FLOW can be prescribed instead of VEL. The units are m³/s. If the flow is entering the computational domain, VOLUME_FLOW should be a negative number, the same convention as for VEL. Note that a SURF with a VOLUME_FLOW prescribed can be invoked by either a VENT or an OBST, but be aware that in the latter case, the resulting velocity on the face or faces of the obstruction will be given by the specified VOLUME_FLOW divided by the area of that particular face. For example:

```
&SURF ID='SUPPLY', VOLUME_FLOW=-5.0, COLOR='GREEN' /  
&OBST XB=..., SURF_ID6='BRICK','SUPPLY','BRICK','BRICK','BRICK','BRICK' /
```

dictates that the forward x -facing surface of the obstruction is to have a velocity equal to $5 \text{ m}^3/\text{s}$ divided by the area of the face (as approximated within FDS) flowing into the computational domain.

Note that `VEL` and `VOLUME_FLOW` should not be specified on the same `SURF` line. The choice depends on whether an exact velocity is desired at a given vent, or whether the given volume flux is desired.

Note also that if the `VENT` or `OBST` crosses mesh boundaries, the specified `VOLUME_FLOW` will be recomputed in each mesh so that the desired volume flow is achieved. This was not the case in FDS version 6.3 and earlier. The sample cases called `volume_flow_1.fds` and `volume_flow_2.fds` in the `Flowfields` folder demonstrate that a `VENT` or an `OBST` can be divided among several meshes. In both cases, air is drawn from a 1 m by 1 m by 1 m box at a rate of $0.01 \text{ m}^3/\text{s}$. These cases also ensure that the `VENT` or `OBST` need not be aligned with the mesh to yield the desired flow rate. Figure 9.1 displays the volume flow drawn through an `OPEN` boundary on an opposite face of the box with either a `VENT` (left) or `OBST` (right) with a specified volume flow.

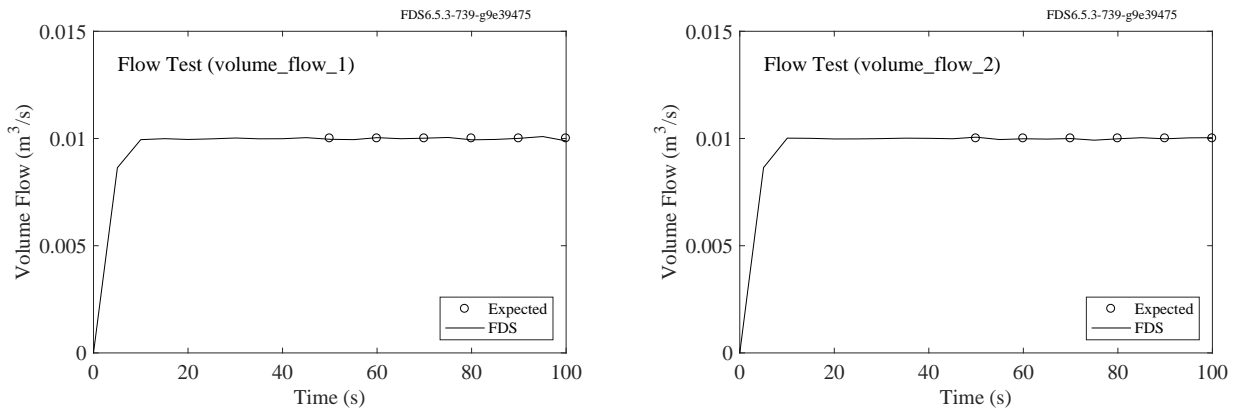


Figure 9.1: Flow rate of air drawn through a unit cube via a `VENT` (left) and `OBST` (right) with specified `VOLUME_FLOW`.

9.1.2 Total Mass Flux

Most often, you specify a simple supply or exhaust vent by setting either a normal velocity or volume flux at a solid surface. However, you may wish to control the total mass flow rate per unit area ($\text{kg}/(\text{m}^2 \cdot \text{s})$) via the parameter `MASS_FLUX_TOTAL`. This parameter uses the same sign convention as `VEL` above. In fact, the value entered for `MASS_FLUX_TOTAL` is converted internally into a velocity boundary condition whose value for an outflow is adjusted based on the local density. Note that `MASS_FLUX_TOTAL` should only be used for an outflow boundary condition; for inflow use `MASS_FLUX` which is discussed in Section 9.1.6.

9.1.3 Heaters

You can create a simple heating vent by changing the temperature of the incoming air

```
&SURF ID='BLOWER', VEL=-1.2, TMP_FRONT=50. /
```

The `VENT` with `SURF_ID='BLOWER'` would blow 50°C air at 1.2 m/s into the flow domain. Making `VEL` positive would suck air out, in which case `TMP_FRONT` would not be necessary.

Note that if `HRRPUA` or solid phase reaction parameters are specified, no velocity should be prescribed. The combustible gases are ejected at a velocity computed by FDS.

9.1.4 Louvered Vents

Most real supply vents are covered with some sort of grill or louvers which act to redirect, or *diffuse*, the incoming air stream. It is possible to mimic this effect, to some extent, by prescribing both a normal and the tangential components of the flow. The normal component is specified with `VEL` as described above. The tangential is prescribed via a pair of real numbers `VEL_T` representing the desired tangential velocity components in the other two coordinate directions (x or y should precede y or z). For example, the line in the example case `Flowfields/tangential_velocity.fds`

```
&SURF ID='LOUVER', VEL=-2.0, VEL_T=3.0,0.0, TAU_V=5., COLOR='GREEN' /
```

is a boundary condition for a louvered vent that pushes air into the space with a normal velocity of 2 m/s and a tangential velocity of 3 m/s in the first of the two tangential directions. Note that the negative sign of the normal component of velocity indicates that the fluid is injected into the computational domain. The tangential velocity of 3 m/s indicates that the flow is in the positive y direction. Both the normal and tangential velocity components are ramped up with either `TAU_V` or `RAMP_V`, as shown in Fig. 9.2.

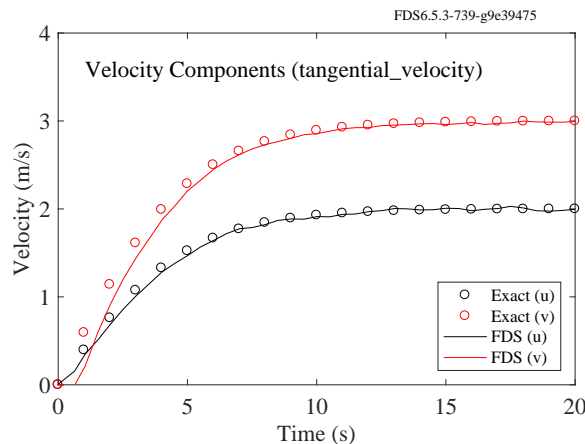


Figure 9.2: Normal and tangential velocity components at a louvered vent, compared to the ideal curve.

In cases of limited mesh resolution, it may not be possible to describe a louvered vent or slot diffuser using `VEL_T` because there may not be enough mesh cells spanning the opening. In these cases, you might consider simply specifying a flat plate obstruction in front of the `VENT` with an offset of one mesh cell. The plate will simply redirect the air flow in all lateral directions.

If the louvered vent is part of an HVAC system, see 9.2.7 for details on how to specify the louver.

9.1.5 Specified Normal Velocity Gradient

It is sometimes desirable to specify a Neumann boundary condition (specified gradient) for the velocity in the direction normal to the boundary. For example, the following allows inflow and outflow along the top of the domain, but $\partial w / \partial z = 0$. Note that `FREE_SLIP=.TRUE.` only sets $\partial u / \partial z = 0$ and $\partial v / \partial z = 0$.

```
&SURF ID = 'sky', COLOR = 'INVISIBLE', VEL_GRAD=0., FREE_SLIP=.TRUE. /
&VENT MB='ZMAX', SURF_ID='sky' /
```

9.1.6 Species and Species Mass Flux Boundary Conditions

There are two types of species boundary conditions (see Chapter 12 for a general discussion of gas species). By default, gas species do not penetrate solid surfaces and you need not specify anything if this is all you need. If the mass fraction of the species is to be some value at a forced flow boundary where `VEL`, `VOLUME_FLOW`, or `MASS_FLUX_TOTAL` is specified, set `MASS_FRACTION(:)` equal to the desired species mass fractions on the appropriate `SURF` line. If the mass flux of the species is desired, set `MASS_FLUX(:)` instead of `MASS_FRACTION(:)`. If `MASS_FLUX(:)` is set, do not specify `VEL`, `VOLUME_FLOW`, or `MASS_FLUX_TOTAL`. These are automatically calculated based on the specified mass flux. The inputs `MASS_FLUX(:)` and typically `MASS_FRACTION(:)` should only be used for inflow boundary conditions. `MASS_FLUX(:)` should be positive with units of $\text{kg}/(\text{m}^2 \cdot \text{s})$. Also note that the background species cannot be specified when using `MASS_FRACTION`. The mass fraction of the background species will be set to account for any mass fraction not specified with other species.

Here is an example of how to specify a surface that generates methane at a rate of $0.025 \text{ kg}/(\text{m}^2 \cdot \text{s})$:

```
&SPEC ID='METHANE' /
&SURF ID='METHANE BURNER', SPEC_ID(1)='METHANE', MASS_FLUX(1)=0.025 /
&VENT XB=..., SURF_ID='METHANE BURNER' /
```

Here is example of how to specify a surface that blows methane with a velocity of 0.1 m/s :

```
&SPEC ID='METHANE' /
&SURF ID='METHANE BLOWER', MASS_FRACTION(1)=1.0, SPEC_ID(1)='METHANE', VEL=-0.1 /
&VENT XB=..., SURF_ID='METHANE BLOWER' /
```

Note that specifying a combination of `VEL` and `MASS_FRACTION` can lead to inaccurate results if the specified velocity is small because diffusion will dominate the mass transport. To obtain an accurate species mass flux at a boundary, use `MASS_FLUX`.

Alternatively, add `CONVERT_VOLUME_TO_MASS=.TRUE.` for velocity boundaries (`VEL`, `VOLUME_FLOW`, or `MASS_FLUX_TOTAL`), which converts volume flow to a mass flux based on the specified boundary composition (`MASS_FRACTION`) and temperature (`TMP_FRONT`):

$$\dot{m}''_{\alpha} = \rho Z_{\alpha} u = \frac{p_{\infty} \bar{W}}{RT} Z_{\alpha} \frac{\dot{Q}}{A} \quad (9.1)$$

where ρ is the density, Z_{α} is the mass fraction of α , u is the velocity normal to the surface, p_{∞} is the ambient pressure, \bar{W} is the mixture molecular weight, R is the ideal gas constant, T is the surface temperature, \dot{Q} is the volume flow rate, and A is the area of the vent.

9.1.7 Tangential Velocity Boundary Conditions at Solid Surfaces

The no-slip condition implies that the continuum tangential gas velocity at a surface is zero. In turbulent flow the velocity increases rapidly through a boundary layer that is only a few millimeters thick to its “free-stream” value. In most practical simulations, it is not possible to resolve this boundary layer directly; thus, an empirical model is used to represent its effect on the overall flow field. For a DNS (Direct Numerical Simulation), the velocity gradient at the wall is computed directly from the resolved velocity near the wall (`NO_SLIP=.TRUE.` by default). For an LES (Large Eddy Simulation), a “log law” wall

model is applied. The surface roughness (in meters) is set by `ROUGHNESS` on `SURF`. See the FDS Technical Reference Guide [?] for wall model details. To force a solid boundary to have a free-slip condition, set `FREE_SLIP=.TRUE.`¹ on the `SURF` line. In LES, to override the wall model and force a no-slip boundary condition, set `NO_SLIP=.TRUE.` on the `SURF` line.

9.1.8 Synthetic Turbulence Inflow Boundary Conditions

Real flows of low-viscosity fluids like air are rarely perfectly stationary in time or uniform in space—they are turbulent (to some degree). Of course, the turbulence characteristics of the flow may have a significant impact on mixing and other behaviors so the specification of nominally constant and uniform boundary conditions may be insufficient. To address this issue, FDS employs a synthetic eddy method (SEM)². Refer to Jarrin [?] for a detailed description. In brief, “eddies” are injected into the flow at random positions on the boundary and advect with the mean flow over a short distance near the boundary equivalent to the maximum eddy length scale. Once the eddy passes through this region it is recycled at the inlet of the boundary with a new random position and length scale. The eddies are idealized as velocity perturbations over a spherical region in space with a diameter (eddy length scale) selected from a uniform random distribution. The selection procedures guarantee that prescribed first and second-order statistics (including Reynolds stresses) are satisfied.

Synthetic turbulence is invoked by setting the number of eddies, `N_EDDY`, the characteristic eddy length scale, `L_EDDY`, and either the root mean square (RMS) velocity fluctuation, `VEL_RMS`, or the Reynolds stress tensor components, `REYNOLDS_STRESS(3,3)` on the `VENT` line. In Fig. 9.3 we show examples using SEM for flat, parabolic, atmospheric, and ramp profiles with 10 % turbulence intensity (see the `sem_*` test series in the Turbulence verification subdirectory). The input lines for the atmospheric case are (see Section 10.1 for further discussion of profile parameters).

```
&SURF ID='inlet', VEL=-1, PROFILE='ATMOSPHERIC', Z0=0.5, PLE=0.3 /
&VENT MB='XMIN', SURF_ID='inlet', N_EDDY=100, L_EDDY=0.2, VEL_RMS=.1 /
```

Note that the Reynolds stress is symmetric and only the lower triangular part needs to be specified. The RMS velocity fluctuation is isotropic (equivalent for each component). Thus, $VEL_RMS \equiv \sqrt{2k/3}$, where $k \equiv \langle \frac{1}{2} u'_i u'_i \rangle$ is the turbulent kinetic energy per unit mass. Below is an example illustrating the equivalence between the RMS velocity fluctuation and the diagonal components of the Reynolds stress. Note that if `VEL_RMS` is specified, this is equivalent to

```
REYNOLDS_STRESS(1,1) = VEL_RMS**2
REYNOLDS_STRESS(2,2) = VEL_RMS**2
REYNOLDS_STRESS(3,3) = VEL_RMS**2
```

and all other components of `REYNOLDS_STRESS` are zero. If the fluctuations are not isotropic, then the Reynolds stresses must be specified componentwise.

In Chapter 7 of Jarrin’s thesis [?], he introduces the Modified Synthetic Eddy Method in which the eddy length scales are anisotropic. This allows more realistic characterization of streamwise vortices in a turbulent boundary layer. To specify the length scales corresponding to the σ_{ij} values in Jarrin’s Eq. (7.1) use `L_EDDY_IJ(3,3)`. Here is an example with random values for the eddy length scales and Reynolds

¹This parameter sets the wall stress to zero (removes viscous friction). It is *not* equivalent to the `SAWTOOTH=.FALSE.` functionality from versions of FDS prior to Version 6.

²SEM only applies to velocity boundary conditions and so may not be used for HVAC vents, which are strict mass flux boundaries.

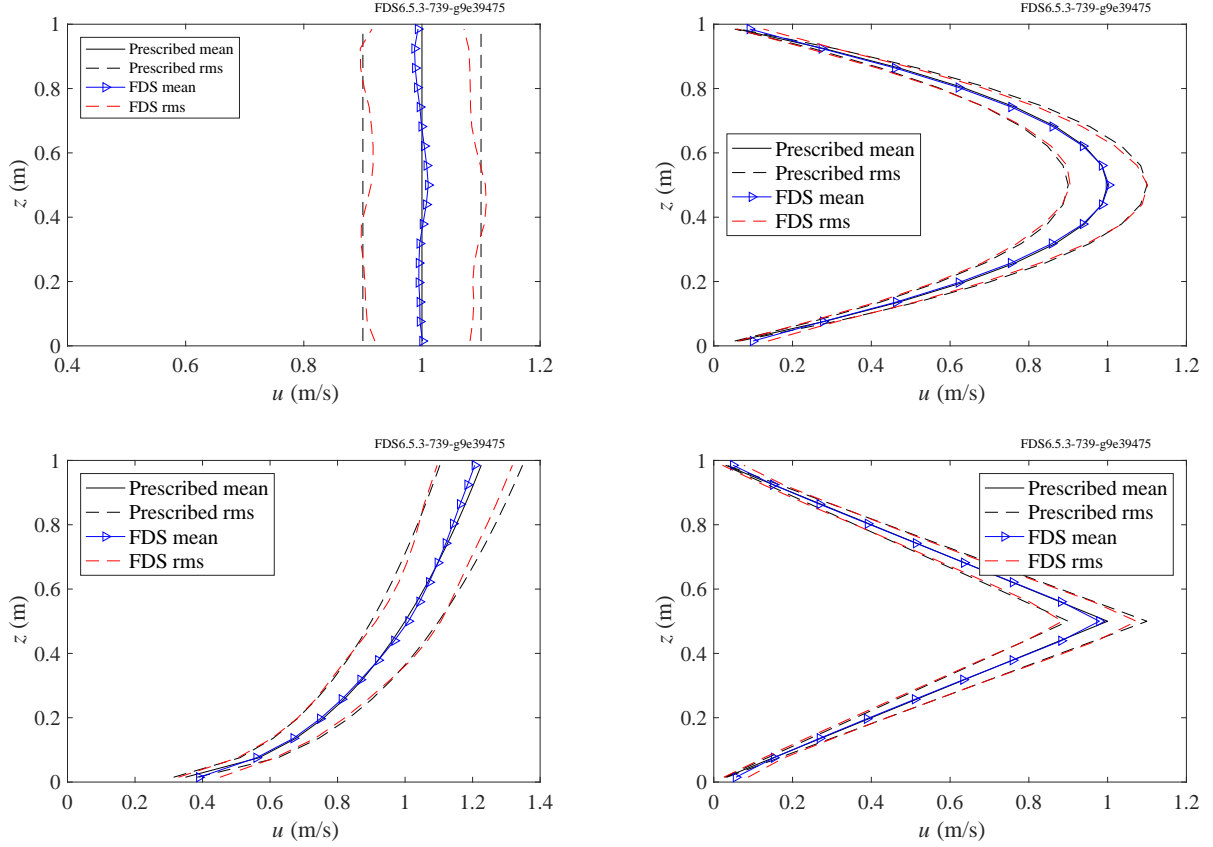


Figure 9.3: Synthetic Eddy Method vent profiles: flat (upper left), parabolic (upper right), atmospheric (lower left), and linear ramp (lower right).

stress components:

```
&VENT XB=... , SURF_ID='WIND', N_EDDY=500,
  L_EDDY_IJ(1,1)=21., L_EDDY_IJ(1,2)=6.22, L_EDDY_IJ(1,3)=4.23
  L_EDDY_IJ(2,1)=2.35, L_EDDY_IJ(2,2)=5.66, L_EDDY_IJ(2,3)=2.50
  L_EDDY_IJ(3,1)=5.42, L_EDDY_IJ(3,2)=0.78, L_EDDY_IJ(3,3)=1.01
  REYNOLDS_STRESS(1,1)=2.16, REYNOLDS_STRESS(1,2)=0., REYNOLDS_STRESS(1,3)=-0.47
  REYNOLDS_STRESS(2,1)=0., REYNOLDS_STRESS(2,2)=1.53, REYNOLDS_STRESS(2,3)=0.
  REYNOLDS_STRESS(3,1)=-0.47, REYNOLDS_STRESS(3,2)=0., REYNOLDS_STRESS(3,3)=4.259 /
```

9.1.9 Random Mass Flux Variation

The current implementation of the Synthetic Eddy Method does not allow variation in mass flux defined on a SURF line. For example, HRRPUA and MASS_FLUX boundary conditions are by default taken as constant for a given SURF. In reality, of course, this is rarely the case—generally there is some level of random variation in the mass flux on the surface. In an attempt to accommodate this effect, we have implemented an experimental feature where the user may specify a mass flux variation as a fractional value with the parameter MASS_FLUX_VAR. For example, if you want a 10 % variation in a heat release rate per unit area of 100 kW/m² then use


```
&SURF ID='burner', HRRPUA=100., MASS_FLUX_VAR=0.1 /
```

It may be helpful to use the boundary file output quantity `MASS FLUX WALL CELL` to visualize the variation. For example,

```
&BNDF QUANTITY='MASS FLUX WALL CELL', CELL_CENTERED=.TRUE. /
```

9.2 HVAC Systems: The HVAC Namelist Group (Table 19.9)

There are occasions where simply defining fixed flow and fixed species boundary conditions is not sufficient to model the behavior of an HVAC (Heating, Ventilation, and Air Conditioning) system. If the ability to transport heat and combustion products through a duct network or the ability to fully account for the pressurization of a compartment due to a fire on the flows in a duct network is important, you can make use of a coupled HVAC network solver. The solver computes the flows through a duct network described as a mapping of duct segments and nodes where a node is either the joining of two or more ducts (a tee for example) or where a duct segment connects to the FDS computational domain.

By default the HVAC solver does not allow for mass storage in the duct network (i.e., what goes in during a time step, goes out during a time step). However, if you set `HVAC_MASS_TRANSPORT=.TRUE.` on the `MISC` line then the HVAC solver will account for mass storage and will compute transient species and energy transport through the ducts.

HVAC components such as fans and binary dampers (fully open or fully closed) can be included in the HVAC network and are coupled to the FDS control function capability. You can select from three fan models.

The HVAC solver is invoked if there is an HVAC namelist group present in the input file. An HVAC network is defined by providing inputs for the ducts; duct nodes; and any fans, dampers, filters, or heating and coiling coils present in the system. Additionally you must define the locations where the HVAC network joins the computational domain. The basic syntax for an HVAC component is:

```
&HVAC TYPE_ID='componenttype', ID='componentname', ... /
```

`TYPE_ID` is a character string that indicates the type of component that the namelist group is defining.

`TYPE_ID` can be `DUCT`, `NODE`, `FAN`, `FILTER`, `AIRCOIL`, or `LEAK` (see Section 9.3.2).

`ID` is a character string giving a name to the component. The name must be unique amongst all other components of that type; however, the same name can be given to components of different types (i.e., a duct and a node can have the same name but two ducts cannot).

A number of examples of simple HVAC systems are given in the HVAC folder of the sample cases and are discussed in the FDS Verification Guide.

As described in the Technical Reference Manual, the HVAC pressure solution is not directly coupled to the FDS pressure solution. Rather there is implicit coupling using the wall boundary condition for HVAC vents. At times this can result in stability problems. If this occurs, setting the keyword `DT_HVAC` on the `MISC` line may help. When set, this will cause FDS to use the value of `DT_HVAC` in the HVAC solver rather than the current FDS time step (filter loading will use the FDS time step). In effect as `DT_HVAC` is increased, the HVAC solution will approach a steady-state solution for the current conditions in the domain.

9.2.1 HVAC Duct Parameters

A typical input line specifying a duct is as follows:

```
&HVAC TYPE_ID='DUCT', ID='ductname', NODE_ID='node 1','node 2', AREA=3.14,  
      LOSS=1.,1., LENGTH=2., ROUGHNESS=0.001, FAN_ID='fan 1', DEVC_ID='device 1' /
```

Or, if you are using `MASS_TRANSPORT=.TRUE.`, as follows:

```
&HVAC TYPE_ID='DUCT', ID='ductname', NODE_ID='node 1','node 2', AREA=3.14,  
      LOSS=1.,1., LENGTH=2., ROUGHNESS=0.001, FAN_ID='fan 1', DEVC_ID='device 1',  
      DUCT_INTERP_TYPE='NODE1', N_CELLS=200 /
```

where:

`AIRCOIL_ID` is the ID of an aircoil located in the duct. The operation of the aircoil can be controlled by either a device or a control function.

`AREA` is the cross sectional area of the duct in m^2 .

`DAMPER` is a logical parameter indicating the presence of a damper in the duct. The state of the damper is controlled by either a device or a control function (see Section 9.2.2).

`DIAMETER` is the diameter of the duct in m. If only `DIAMETER` is specified, the `AREA` will be computed assuming a round duct. Do not specify both `DIAMETER` and `PERIMETER`.

`DEVC_ID` is the ID of a `DEVC` for a damper, fan, or aircoil in the duct. An alternative is `CTRL_ID`.

`DUCT_INTERP_TYPE` sets whether the duct is initialized with data from the first node (`NODE1`) or the second node (`NODE2`). Continuous runs of ducts cannot have a mix of interpolation methods (refer to Section 9.2.8).

`FAN_ID` is the ID of a fan located in the duct. Instead of specifying a `FAN_ID`, you could specify the `VOLUME_FLOW` rate (m^3/s) through the duct. The operation of the fan can be controlled by either a device or a control function.

`LENGTH` is the `LENGTH` of the duct in m. Note that `LENGTH` is not computed automatically as the difference between the `XYZ` of the duct's endpoints.

`LOSS` is a pair of real numbers giving the forward and reverse dimensionless "minor losses" loss coefficient (K_{minor}) in the duct. Minor losses are pressure losses through components such as tees, valves and bends. However, you can use `LOSS` to represent wall friction losses if you want - in this case ensure you leave `ROUGHNESS` unset so that the HVAC solver does not compute a friction factor. The forward direction is defined as flow from the first node listed in `NODE_ID` to the second node listed in `NODE_ID`.

`MASS_FLOW` is a fixed mass flow rate (kg/s) through the duct. Only specify one of `MASS_FLOW` or `VOLUME_FLOW`. You can change its value in time either using the characteristic time, `TAU_VF`, to define a tanh (`TAU_VF>0`) or t^2 ramp (`TAU_VF<0`); or you can specify a `RAMP_ID`. `MASS_FLOW` should only be specified for conditions where the upstream node density will not change during the solution process.

`N_CELLS` defines the number of cells used in a discretized duct.

`NODE_ID` gives the IDs of the nodes on either end of the duct segment. Positive velocity in a duct is defined as flow from the first node to second node.

`PERIMETER` is used along with `AREA` to specify a duct with non-circular cross-section. The `DIAMETER` will be computed as the hydraulic diameter.

`RAMP_LOSS` If specified this `RAMP` is a multiplier for the `LOSS`.

`REVERSE` is a logical parameter that when `.TRUE.` indicates that the specified `FAN_ID` or `VOLUME_FLOW` blows from the second node to the first.

`ROUGHNESS` is the absolute roughness in m of the duct that is used to compute the friction factor for the duct. If `ROUGHNESS` is not set, the HVAC solver will not compute the friction factor and the wall friction will be zero - if this is the case you may want to account for wall friction losses in `LOSS`. "Perfectly smooth" ducts and pipes still have wall losses and therefore setting `ROUGHNESS` to zero will tell the HVAC solver to compute the minimum friction factor (which is non-zero) - this is not the same as leaving `ROUGHNESS` unset.

`VOLUME_FLOW` is a fixed flow rate (m³/s) through the duct. Only specify one of `MASS_FLOW` or `VOLUME_FLOW`.

If you specify `VOLUME_FLOW`, you can change its value in time either using the characteristic time, `TAU_VF`, to define a tanh (`TAU_VF`>0) or t² ramp (`TAU_VF`<0); or you can specify a `RAMP_ID`. This cannot be controlled by a device or control function; however, a constant volume flow `FAN` can be.

Note that only one of `AIRCOIL_ID`, `DAMPER`, or `FAN_ID` should be specified for a duct. Also note that if one of these is specified, but no device or control function is provided, then the item will be assumed to be on or open as appropriate.

To reduce the computational cost of the HVAC solver, a duct should be considered as any length of duct that connects two items that must be defined as nodes (i.e., a connection to the FDS domain, a filter, or a location where more than two ducts join). That is, a duct should be considered as any portion of the HVAC system where flow can only be in one direction at given point in time (flow can reverse direction over time). For example the top of Figure 9.4 shows a segment of an HVAC system where flow from a tee goes through an expansion fitting, two elbows, an expansion fitting, and a straight length of duct before it terminates as a connection to the FDS domain.

This could be input as each individual fitting or duct with its associated area and loss as shown in the middle of the figure; however, this would result in five duct segments (one for each component) with six node connections resulting in eleven parameters (five velocities and six pressures) which must be solved for. This is not needed since whatever the flow rate is in any one segment of the duct, that same flow rate exists in all other segments; thus, the velocities in any segment can be found by taking the area ratios, $v_1/v_2 = A_2/A_1$. Since flow losses are proportional to the square of the velocity, an equivalent duct can be constructed using the total length of the duct, and a representative area (A_{eff}) or diameter. The pressure losses associated with all the segments of the duct can be collapsed to a single effective loss (K_{eff}) by summing all of the fitting losses (K_{minor}) through the duct as follows:

$$K_{\text{eff}} = \sum_i (K_{\text{minor}})_i \frac{A_{\text{eff}}}{A_i} \quad (9.2)$$

where i is a fitting and A_i is the area associated with the fitting loss.

9.2.2 HVAC Dampers

Dampers can be modeled in one of two ways.

The first method is a simple binary (flow or no flow) damper. This can be placed in a duct by adding the keyword `DAMPER` along with either a `CTRL_ID` or `DEVC_ID`. When the control or device is `.TRUE.` the

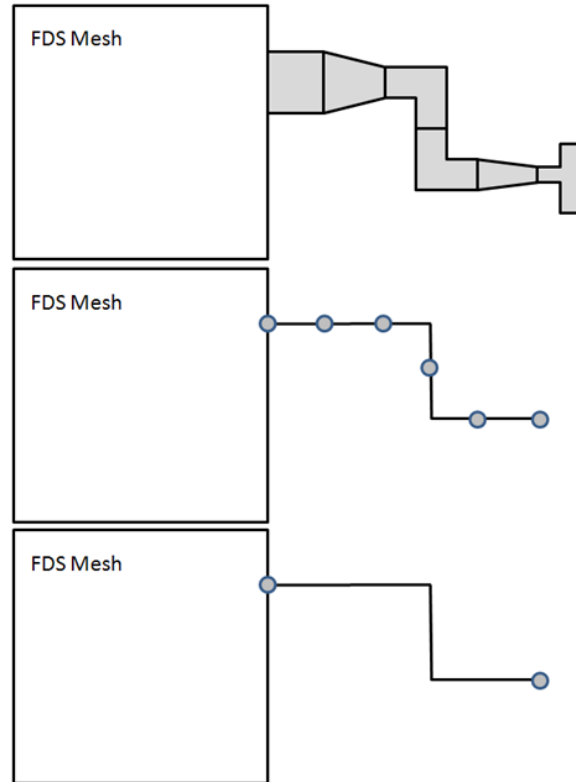


Figure 9.4: An example of simplifying a complex duct.

damper will be open, and when `.FALSE.` the damper will be closed and block 100 % of the duct area. The example below shows a duct with a damper that is linked to a `DEVC` that closes the damper at 10 s. For further details see the `HVAC_damper` example case, which is documented in the Verification Guide.

```
&HVAC TYPE_ID='DUCT',ID='EXHAUST 2',NODE_ID='TEE','EXHAUST 2',AREA=0.01,
      LENGTH=1.0,LOSS=0,0,DAMPER=.TRUE.,DEVC_ID='TIMER'/
&DEVC QUANTITY='TIME',ID='TIMER',SETPOINT=10,INITIAL_STATE=.TRUE.,XYZ=0,0,0/
```

The second method is to specify a `RAMP_LOSS` for the duct. This approach multiplies the `LOSS` array for the duct by the output of the `RAMP`. This allows for dampers that have leakage and/or dampers that have a variable position other than fully open or fully closed. An example is shown below where the `LOSS` in the duct changes from 1,1 at 10 s to 2000,2000 at 11 s.

```
&HVAC TYPE_ID='DUCT',ID='EXHAUST 2',NODE_ID='TEE','EXHAUST 2',AREA=0.01,
      LENGTH=1.0,LOSS=1,1,RAMP_LOSS='LOSS RAMP'/
&RAMP ID='LOSS RAMP',T=10,F=1/
&RAMP ID='LOSS RAMP',T=11,F=2000/
```

9.2.3 HVAC Node Parameters

Below are three example duct node inputs representing a typical tee-type connection (multiple ducts being joined), a connection to the FDS domain, and a connection to the ambient outside the FDS domain.

```

&HVAC TYPE_ID='NODE', ID='tee', DUCT_ID='duct 1','duct 2',...'duct n',
      LOSS=lossarray, XYZ=x,y,z /
&HVAC TYPE_ID='NODE', ID='FDS connection', DUCT_ID='duct 1', VENT_ID='vent',
      LOSS=enter,exit /
&HVAC TYPE_ID='NODE', ID='ambient', DUCT_ID='duct 1', LOSS=enter,exit,
      XYZ=x,y,z, AMBIENT=.TRUE. /

```

where:

AMBIENT is a logical value. If **.TRUE.**, then the node is connected to the ambient (i.e., it is equivalent to the **OPEN** boundary condition on a **SURF** line).

DUCT_ID gives the IDs of the ducts connected to the node. Up to 10 ducts can be connected to a node.

FILTER_ID gives the ID a filter located at the node. A node with a filter can only have two connected ducts.

LOSS is an n by n array of real numbers giving the dimensionless loss coefficients for the node. **LOSS(I,J)** is the loss coefficient for flow from duct **I** to duct **J** expressed in terms of the downstream duct area (see discussion in 9.2.1 on how to adjust losses for area changes). For a terminal node (e.g., a node connected to the ambient or to a **VENT**) the **LOSS** is entered as a pair of numbers representing loss coefficient for flow entering the HVAC system and for flow exiting the HVAC system.

VENT_ID is the name of the **VENT** where the node connects to the FDS computational domain. No two **VENT**s should be defined with the same **VENT_ID**.

XYZ is a triplet of real numbers giving the coordinates of the node. This location is used to compute buoyancy heads. If the node is connected to the FDS domain, then do not specify **XYZ**. FDS will compute it as the centroid of the **VENT**. Note that if you do not specify an **XYZ** for an interior node, then FDS will use the default value of 0,0,0.

A duct node must either have two or more ducts attached to it or it must have either **AMBIENT=.TRUE.** or a specified **VENT_ID**. When defining a **VENT** as a component of an HVAC system you must set **SURF_ID** to 'HVAC' and you must set the **VENT_ID**.

Note, do not split an HVAC **VENT** over multiple-meshes. It is permissible to have individual **VENT** lines for an HVAC system in different meshes, but any single HVAC **VENT** needs to be contained within a single mesh. This restriction does not apply to surfaces with leakage flows.

9.2.4 HVAC Fan Parameters

Below are given sample inputs for the three types of fans supported by FDS.

```

&HVAC TYPE_ID='FAN', ID='constant volume', VOLUME_FLOW=1.0, LOSS=2./
&HVAC TYPE_ID='FAN', ID='quadratic', MAX_FLOW=1., MAX_PRESSURE=1000., LOSS=2. /
&HVAC TYPE_ID='FAN', ID='user fan curve', RAMP_ID='fan curve', LOSS=2. /

```

where:

LOSS is the loss coefficient for flow through the fan when it is not operational.

MAX_FLOW is the maximum volumetric flow of the fan in m^3/s . This input activates a quadratic fan model.

MAX_PRESSURE is the stall pressure of the fan in units of Pa. This input activates a quadratic fan model.

RAMP_ID identifies the RAMP that contains a table of pressure drop across the fan (Pa) versus the volumetric flow rates (m³/s) for a user-defined fan curve.

TAU_FAN defines a tanh (TAU_FAN > 0) or t² ramp (TAU_FAN < 0) for the fan. This is applied to the flow rate computed by any of the three types (constant flow, quadratic, or user-defined ramp) of fans.

VOLUME_FLOW is the fixed volumetric flow of the fan (m³/s). If you wish to have a time dependent flow use the VOLUME_FLOW input for a duct rather than for a fan.

Note that only one set of fan model inputs (VOLUME_FLOW, RAMP_ID, or MAX_FLOW + MAX_PRESSURE) should be specified. Also note that FAN defines a class of fans rather than one specific fan. Therefore, more than one duct can reference a single FAN.

Fan Curves

In Section 9.1 there is a discussion of velocity boundary conditions, in which a fan is modeled simply as a solid boundary that blows or sucks air, regardless of the surrounding pressure field. In the HVAC model, this approach to modeling a fan occurs when the fan is specified with a VOLUME_FLOW. In reality, fans operate based on the pressure drop across the duct or manifold in which they are installed. A very simple “fan curve” is given by:

$$\dot{V}_{\text{fan}} = \dot{V}_{\text{max}} \text{sign}(\Delta p_{\text{max}} - \Delta p) \sqrt{\frac{|\Delta p - \Delta p_{\text{max}}|}{\Delta p_{\text{max}}}} \quad (9.3)$$

This simple “fan curve” is the “quadratic” fan model as the pressure is proportional to the square of the volume flow rate.

The volume flow in the absence of a pressure difference, MAX_FLOW, is given by \dot{V}_{max} . The pressure difference, $\Delta p = p_1 - p_2$, indicates the difference in pressure between the downstream compartment, or “zone,” and the upstream. The subscript 1 indicates downstream and 2 indicates upstream. The term, Δp_{max} , is the maximum pressure difference, MAX_PRESSURE, the fan can operate upon, and it is assumed to be a positive number. The flow through a fan will decrease from \dot{V}_{max} at zero pressure difference to 0 m³/s at Δp_{max} . If the pressure difference increases beyond this, air will be forced backwards through the fan. If the downstream pressure becomes negative, then the volume flow through the fan will increase beyond MAX_FLOW. More complicated fan curves can be specified by defining a RAMP. In the example inputs below, one fan of each type is specified. A constant volume flow fan with a VOLUME_FLOW of 10 m³/s, a quadratic fan with MAX_FLOW=10 and MAX_PRESSURE=500, and a user-defined fan with the RAMP set to the values of the quadratic fan in 200 Pa increments,

```
&HVAC TYPE_ID='FAN', ID='constant volume', VOLUME_FLOW=10.0/
&HVAC TYPE_ID='FAN', ID='quadratic', MAX_FLOW=10., MAX_PRESSURE=500./
&HVAC TYPE_ID='FAN', ID='user fan curve', RAMP_ID='fan curve'/

&RAMP ID='fan curve',T=-10.00,F= 1000/
&RAMP ID='fan curve',T= -7.75,F=  800/
&RAMP ID='fan curve',T= -4.47,F=  600/
&RAMP ID='fan curve',T=  4.47,F=  400/
&RAMP ID='fan curve',T=  7.75,F=  200/
&RAMP ID='fan curve',T= 10.00,F=   0/
&RAMP ID='fan curve',T= 11.83,F= -200/
&RAMP ID='fan curve',T= 13.42,F= -400/
&RAMP ID='fan curve',T= 14.83,F= -600/
&RAMP ID='fan curve',T= 16.12,F= -800/
&RAMP ID='fan curve',T= 17.32,F=-1000/
```

Figure 9.5 displays the fan curves for the inputs shown above. Additional examples can be found in the `ashrae7` and `fan_test` example cases, which are documented in the Verification Guide.

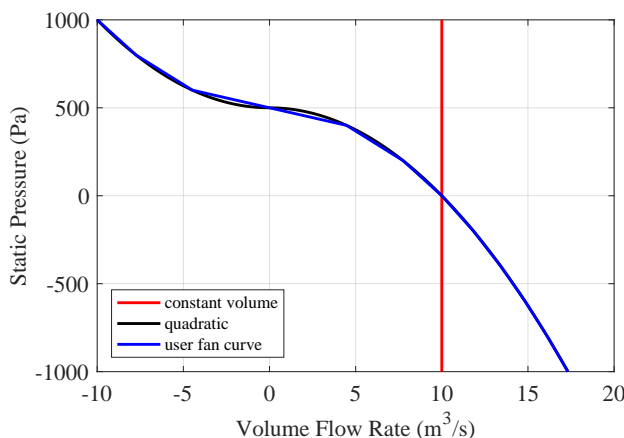


Figure 9.5: Fan curves corresponding to a constant fan with `VOLUME_FLOW=10`, a quadratic fan with `MAX_FLOW=10` and `MAX_PRESSURE=500`, and a user-defined `RAMP` equivalent to the quadratic fan.

Jet Fans

Fans do not have to be mounted on a solid wall, like a supply or an exhaust fan. If you just want to blow gases in a particular direction, create an obstruction `OBST` and apply to it `VENT` lines that are associated with a simple HVAC system. This allows hot, smokey gases to pass through the obstruction, much like a free-standing fan. See the example case `jet_fan.fds` which places a louvered fan (blowing diagonally down) near a fire (see Fig. 9.6).

You may also want to construct a *shroud* around the fan using four flat plates arranged to form a short passageway that draws gases in one side and expels them out the other. The obstruction representing the fan can be positioned about halfway along the passage (if a louvered fan is being used, place the fan at the end of the passage).

9.2.5 HVAC Filter Parameters

A sample input for a filter is given by:

```
&HVAC TYPE_ID='FILTER', ID='filter 1', LOADING=0., SPEC_ID='SOOT',  
      EFFICIENCY=0.99, LOADING_MULTIPLIER=1, CLEAN_LOSS=2., LOSS=100./
```

where:

`CLEAN_LOSS` is the dimensionless loss coefficient for flow through the filter when it is clean (zero loading).

`EFFICIENCY` is an array of the species removal efficiency from 0 to 1 where 0 is no removal of that species and 1 is complete filtration of the species. The species are identified using `SPEC_ID`.

`LOADING` is an array of the initial loading (kg) of the filter for each species being filtered.

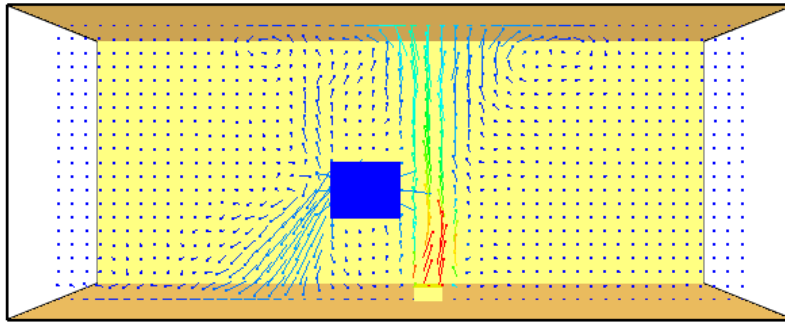


Figure 9.6: Jet fan with a louvered output UVW=-1, 0, -1.

LOADING_MULTIPLIER is an array of the species multiplier, M_i , used in computing the total filter loading when computing the loss coefficient of the filter.

LOSS invokes a linear loss coefficient model where the dimensionless loss coefficient, K , is given as a linear function of the total loading, $K_{\text{FILTER}} = K_{\text{CLEAN_LOSS}} + K_{\text{LOSS}} \sum (L_i M_i)$, where L_i is the species loading and M_i is a multiplier. Only one of LOSS or RAMP_ID should be specified.

RAMP_ID identifies the RAMP that contains a table of pressure drop across the filter as a function of total loading (the summation term given in the definition of LOSS above). Only one of LOSS or RAMP_ID should be specified.

SPEC_ID identifies the tracked species for the inputs of LOADING_MULTIPLIER and LOADING.

A sample set of filter inputs is shown below. These lines define a filter that removes the species PARTICULATE with 100 % efficiency. The filter has an initial loss coefficient of 1 and that loss increases by a factor of 7332 for each kg of PARTICULATE captured by the filter. For further details see the sample case HVAC_filter, which is documented in the Verification Guide.

```
&SPEC ID='PARTICULATE',MW=28.,MASS_FRACTION_0=0.001,SPECIFIC_HEAT=1./
&HVAC TYPE_ID='NODE',ID='FILTER',DUCT_ID='DUCT1','DUCT2',XYZ(3)=0.55,
      FILTER_ID='FILTER'/
&HVAC TYPE_ID='FILTER',ID='FILTER',CLEAN_LOSS=1.,SPEC_ID='PARTICULATE',EFFICIENCY=1.,
      LOSS=7732.446,LOADING_MULTIPLIER=1./
```

Note that a filter input refers to a class of filters and that multiple ducts can reference the same filter definition.

9.2.6 HVAC Aircoil Parameters

An aircoil refers to a device that either adds or removes heat from air flowing through a duct. In a typical HVAC system this is done by blowing the air over a heat exchanger (hence the term aircoil) containing a working fluid such as chilled water or a refrigerant. A sample input line is as follows:

```
&HVAC TYPE_ID='AIRCOIL', ID='aircoil 1', EFFICIENCY=0.5,
      COOLANT_SPECIFIC_HEAT=4.186, COOLANT_TEMPERATURE=10., COOLANT_MASS_FLOW=1./
```


where:

COOLANT_MASS_FLOW is the flow rate of the working fluid (kg/s).

COOLANT_SPECIFIC_HEAT is the specific heat (kJ/(kg · K)) of the working fluid.

COOLANT_TEMPERATURE is the inlet temperature of the working fluid (°C).

EFFICIENCY is the heat exchanger efficiency, η , from 0 to 1. A value of 1 indicates the exit temperatures on both sides of the heat exchanger will be equal.

FIXED_Q is the constant heat exchange rate. A negative value indicates heat removal from the duct. The heat exchange rate can be controlled by either RAMP_ID or by TAU_AC.

TAU_AC defines a tanh (TAU_AC>0) or t^2 ramp (TAU_AC<0) for the aircoil. This is applied to the FIXED_Q of the aircoil. Alternatively, a RAMP_ID can be given.

Note that either FIXED_Q or the set COOLANT_SPECIFIC_HEAT, COOLANT_MASS_FLOW, COOLANT_TEMPERATURE, and EFFICIENCY should be specified. In the latter case, the heat exchange is computed as a two step process. First, the outlet temperature is determined assuming 100 % efficient (i.e., both fluids exit at the same temperature):

$$T_{\text{fluid,out}} = \frac{c_{p,\text{gas}} u_{\text{duct}} A_{\text{duct}} \rho_{\text{duct}} T_{\text{duct,in}} + c_{p,\text{fluid}} \dot{m}_{\text{fluid}} T_{\text{fluid,in}}}{c_{p,\text{gas}} u_{\text{duct}} A_{\text{duct}} \rho_{\text{duct}} + c_{p,\text{fluid}} \dot{m}_{\text{fluid}}} \quad (9.4)$$

Second, the actual heat exchanged is computed using the EFFICIENCY.

$$\dot{q}_{\text{coil}} = \eta c_{p,\text{fluid}} \dot{m}_{\text{fluid}} (T_{\text{fluid,in}} - T_{\text{fluid,out}}) \quad (9.5)$$

Note that an aircoil input refers to a class of aircoils and that multiple ducts can reference the same aircoil definition.

The sample input file HVAC_aircoil.fds demonstrates the use of the aircoil inputs. A constant flow duct removes air (defined as 28 g/mol with a specific heat of 1 kJ/(kg · K)) from the floor and injects in through the ceiling at a volume flow rate of 1 m³/s. An aircoil is defined with a working fluid flowing at 10 kg/s and 100 °C with a specific heat of 4 kJ/(kg · K). The aircoil has an efficiency of 50 %. Using the above equations the aircoil will add 45.2 kW of heat to the gas flowing through the duct resulting in a duct exit temperature of 332 K. These results are shown in Fig. 9.7.

9.2.7 Louvered HVAC Vents

The HVAC system being modeled may either have louvers that redirect the flow leaving a vent or the orientation of the real vent may not lie along one of the axes in FDS. To define the flow direction for an HVAC outlet, you can use the keyword UVW on VENT. UVW is the vector indicating the direction of flow from the VENT. For example:

```
&OBST XB=1.0,2.0,0.0,1.0,0.0,1.0 /
&VENT XB=1.0,1.0,0.0,1.0,0.0,1.0, SURF_ID='HVAC', ID='HVAC OUTLET', UVW=-1,0,1 /
```

The above input defines a vent lying in the y-z plane facing in the $-x$ direction. The flow vector indicates that the flow from this vent is in the $-x$ direction with a 45 degree up angle (the x and z components are equal in size). FDS will set the tangential velocity of the vent to obtain the specified direction indicated by UVW. This will only be done if the vent is inputting gas into the domain.

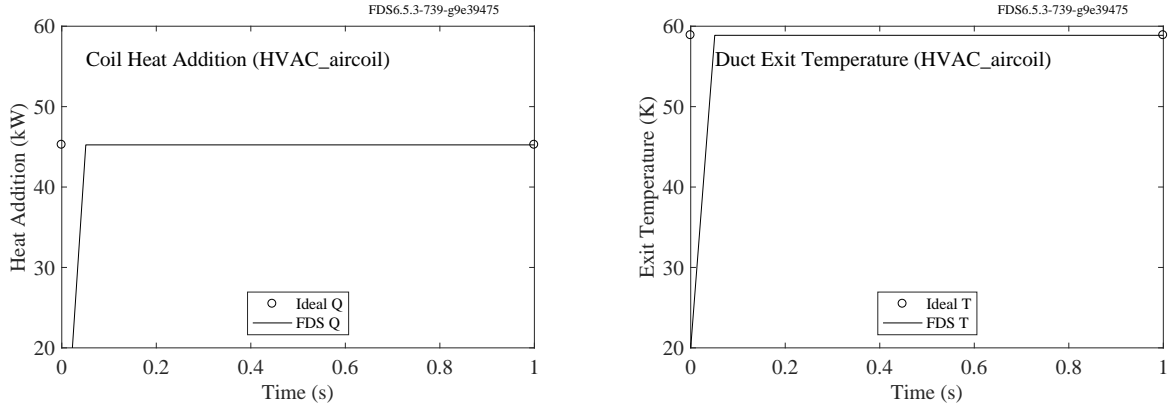


Figure 9.7: (Left) Heat addition and (Right) duct exit temperature for the HVAC_aircoil case.

9.2.8 HVAC Mass Transport

If you set `HVAC_MASS_TRANSPORT=.TRUE.` on the `MISC` line this will call the HVAC mass transport subroutine. This subroutine accounts for mass storage in the duct network and transport time of species and energy. The current HVAC solver does not account for heat loss in the HVAC network.

To avoid the possibility of there being no available initialization data for a duct run (a series of ducts connected to one another via ducts and nodes), `DUCT_INTERP_TYPE` in each comprising duct must be consistent. This ensures that there is a non-internal duct node from which the duct run adopts its initial data. If duct interpolation is set to `NODE1` then data from the first node of the lowest numbered duct in the duct run is initialized in the duct, if it is set to `NODE2` then data from the second node of the highest numbered duct in the duct run is initialized in the duct. `DUCT_INTERP_TYPE` is related to initialization only; if the FDS domain is initialized as ambient/background data then there will be no difference in output when using either duct interpolation method.

You can increase or decrease the number of cells each discretized duct has, to increase solution accuracy or decrease simulation time respectively. The default value is set based upon a cell size of 100 mm. Where the duct is not divisible by 100 mm, the number of cells is rounded up to the nearest integer (e.g. a duct with a length of 1.05 m will have 11 cells). Ducts with a length of less than 100 mm adopt a cell number of 2. If `N_CELLS=1` is assigned to a duct then, even if `HVAC_MASS_TRANSPORT=.TRUE.`, the HVAC mass transport subroutine will not be called for this duct.

If you are using `DEVCS` to output spatially-integrated statistics as per Section 18.10.10 (such as `VOLUME MEAN`, `VOLUME INTEGRAL` or `MASS INTEGRAL`) then be aware that, even if the integration volume bound by `XB` encapsulates the spatial location of a duct, the quantity in the duct will not be recorded by the `DEVCS`. For example, if there is 1 m³ of species 1 initialized in an upstream FDS compartment which is transported to a downstream FDS compartment via an HVAC network with a total volume greater than 1 m³, the value of total mass of species 1 output by a `DEVCS` recording the `VOLUME INTEGRAL` of `DENSITY` will reduce to zero during the time for which it is in the HVAC network domain.

9.3 Pressure-Related Effects: The ZONE Namelist Group (Table 19.30)

FDS assumes pressure to be composed of a “background” component, $\bar{p}(z,t)$, plus a perturbation pressure, $\tilde{p}(\mathbf{x},t)$. Most often, \bar{p} is just the hydrostatic pressure, and \tilde{p} is the flow-induced spatially-resolved perturbation. You can specify any number of sealed compartments within the computational domain that can

have their own “background” pressures, and these compartments, or “pressure zones,” can be connected via leakage and duct paths whose flow rates are tied to the pressure of the adjacent zones.

9.3.1 Specifying Pressure Zones

A pressure zone can be any region within the computational domain that is separated from the rest of the domain, or the exterior, by solid obstructions. There is currently no algorithm within FDS to identify these zones based solely on your specified obstructions. Consequently, it is necessary that you identify these zones explicitly in the input file. The basic syntax for a pressure `ZONE` is:

```
&ZONE XB=0.3,1.2,0.4,2.9,0.3,4.5 /
```

This means that the rectangular region, $0.3 < x < 1.2$, $0.4 < y < 2.9$, $0.3 < z < 4.5$, is assumed to be within a sealed compartment. There can be multiple `ZONES` declared. The indices of the zones, which are required for the specification of leaks and fans, are determined simply by the order in which they are specified in the input file. By default, the exterior of the computational domain is Zone 0. If there are no `OPEN` boundaries, the entire computational domain will be assumed to be Zone 1.

There are several restrictions to assigning pressure zones. First, the declared pressure zones must be completely within a region of the domain that is bordered by solid obstructions. If the sealed region is not rectangular, FDS will extend the specified `ZONE` boundaries to conform to the non-rectangular region. It is possible to “break” pressure zones by removing obstructions between them. An example of how to break pressure zones is given below. Second, pressure zones can span multiple meshes, but it is recommended that you check the pressure in each mesh to ensure consistency. Also, if the `ZONE` does span multiple meshes, make sure that the specified rectangular coordinates do so as well. This allows FDS to determine the actual extent of the `ZONE` independently for each mesh.

Note that if you plan to have one zone open up to another via the removal of an obstruction, make sure that the coordinates of the two zones abut (i.e., touch) even if one of the zones includes the solid obstruction that separates them. FDS recognizes that a zone boundary has been removed when two adjacent cells belonging to two different zones have no solid obstruction between them. It is recommended that you extend at least one of the zone boundaries *into* the solid obstruction separating the two zones. That way, when the obstruction is removed, the newly created gas phase cells will be assigned to one or the other zone and it will become obvious that two adjacent gas phase cells are of two different zones, at which point the zones will merge and no longer have distinct background pressures.

For the special case where a zone has periodic boundaries (`SURF_ID='PERIODIC'` on `VENT`), you must add `PERIODIC=.TRUE.` on the `ZONE` line.

Example Case: Pressure Rise in a Compartment

This example tests several basic features of FDS. A narrow channel, 3 m long, 0.002 m wide, and 1 m tall, has air injected at a rate of $0.1 \text{ kg/m}^2/\text{s}$ over an area of 0.2 m by 0.002 m for 60 s, with a linear ramp-up and ramp-down over 1 s. The total mass of air in the channel at the start is 0.00718 kg. The total mass of air injected is 0.00244 kg. The domain is assumed two-dimensional, the walls are adiabatic, and `STRATIFICATION` is set to `.FALSE.` simply to remove the slight change in pressure and density with height. The domain is divided into three meshes, each 1 m long and each with identical gridding. We expect the pressure, temperature and density to rise during the 60 s injection period. Afterwards, the temperature, density, and pressure should remain constant, according to the equation of state. Figure 9.8 shows the results of this calculation. The density matches exactly showing that FDS is injecting the appropriate amount of

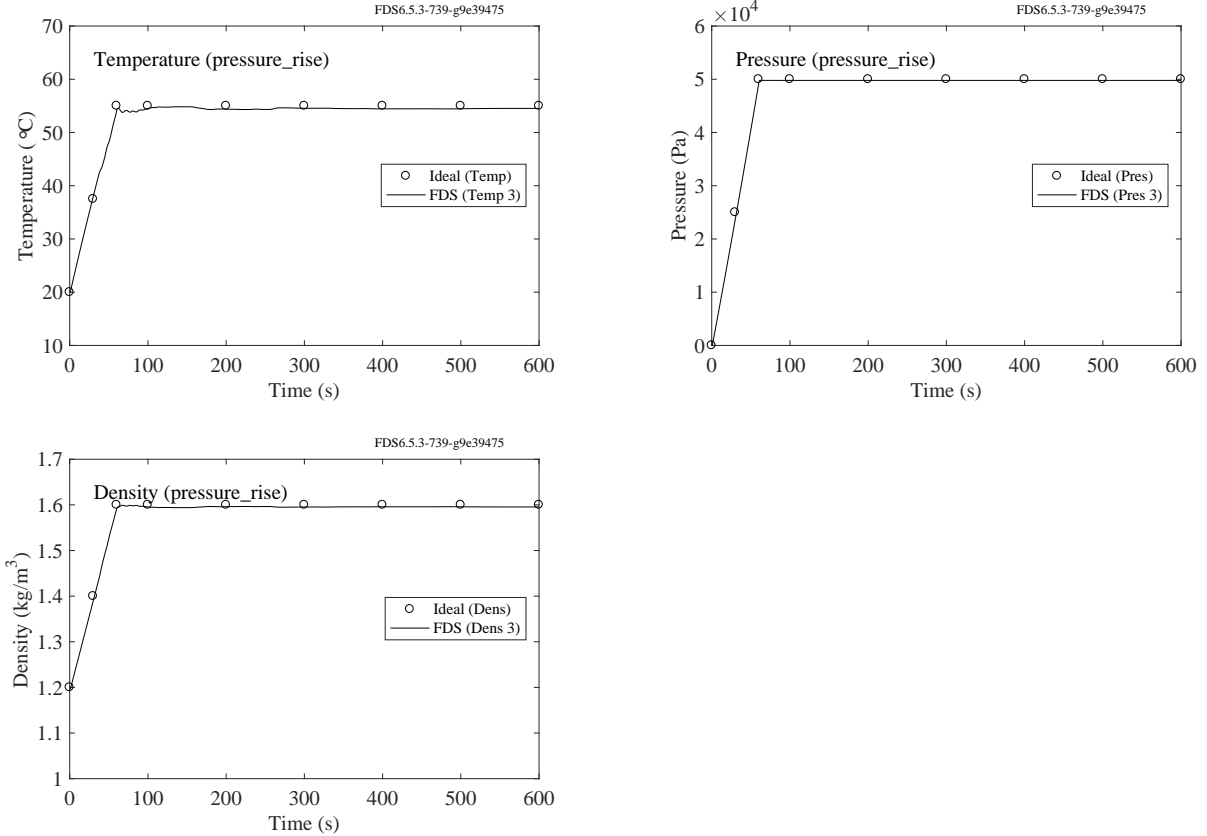


Figure 9.8: Output of `pressure_rise` test case.

mass. The steady state values of the pressure, density and temperature are consistent with the ideal values obtained from the first law of thermodynamics.

Example Case: Breaking Pressure Zones

In this example, three simple compartments are initially isolated from each other and from the ambient environment outside. Each compartment is a separate pressure zone. Air is blown into Zone 1 at a constant rate of 0.1 kg/s, increasing its pressure approximately 2000 Pa by 10 s, at which time Zone 1 is opened to Zone 2, decreasing the overall pressure in the two zones to roughly one-third the original value because the volume of the combined pressure zone has been roughly tripled. At 15 s, the pressure is further decreased by opening a door to Zone 3, and, finally, at 20 s the pressure returns to ambient following the opening of a door to the outside. Figure 9.9 displays the pressure within each compartment. Notice that the pressures do not come to equilibrium instantaneously. Rather, the `PRESSURE_RELAX_TIME` (on the `PRES` line) is applied to bring the zones into equilibrium over a specified period of time. This is done for several reasons. First, in reality doors and windows do not magically disappear as they do in FDS. It takes a finite amount of time to fully open them, and the slowing of the pressure increase/decrease is a simple way to simulate the effect. Second, relatively large pressure differences between zones wreak havoc with flow solvers, especially ones like FDS that use a low Mach number approximation. To maintain numerical stability, FDS gradually brings the pressures into equilibrium. This second point ought to be seen as a warning.

Since pressure zones are defined using `xB`, it might not be possible to define a complexly shaped set of

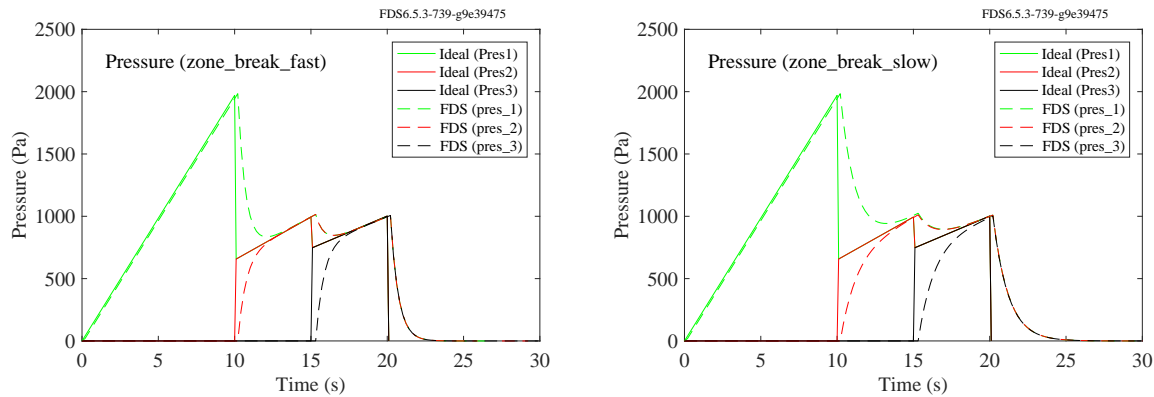


Figure 9.9: Output of `zone_break` test cases. The figure on the left results from using a pressure relaxation time of 0.5 s. The figure on the right uses 1 s, the default.

pressure zones using just the `ZONE` inputs. A work around for this is to define the complex zone as a series of zones in the same manner you would divide a domain into multiple meshes. The check for merged pressure zones only works if two zones were initially isolated at the start of the calculation (there must have been a wall that was removed). Therefore, define an obstruction at the boundary of the zones that is removed after the first time step.

Do not use FDS to study the sudden rupture of pressure vessels! Its low Mach number formulation does not allow for high speed, compressible effects that are very important in such analyses. The zone breaking functionality described in this example is only intended to be used for relatively small pressure differences (<0.1 atm) between compartments. Real buildings cannot withstand substantially larger pressures anyway.

Example Case: Irregularly Shaped Zone

This example is similar to the ones in the previous section, except in this case, the pressure zone is L-shaped and split across two meshes. The objective is simply to ensure that the specification of the pressure zone is properly accounted for in the model. Figure 9.10 compares the predicted pressure in the compartment compared to an exact solution. Air is injected into the compartment for 5 s, after which the compartment is opened to a smaller compartment. At 15 s, the smaller compartment is opened to the outside.

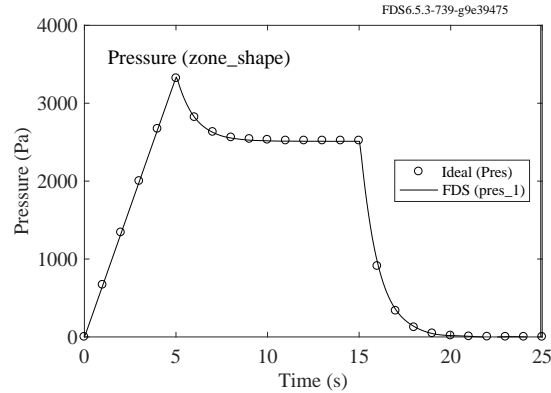


Figure 9.10: Output of `zone_shape` test case. Shown is the pressure in an L-shaped compartment that is opened to another compartment at 5 s, and the outdoors at 15 s.

9.3.2 Leaks

With a few notable exceptions, like containment buildings for nuclear power plants, real world construction is not air tight. Small gaps occur along windows and doors and where walls abut each other and the floors and ceilings. As a compartment is pressurized by a fire, air will escape through these small gaps. This is referred to as leakage.

Leakage is inherently a sub-grid scale phenomenon because the leakage area is usually very small. In other words, it is not possible to define a leak directly on the numerical mesh. It is sometimes possible to “lump” the leaks into a single mesh-resolvable hole, but this is problematic for two reasons. First, the leakage area rarely corresponds neatly to the area of a single mesh cell-sized hole. Second, the flow speeds through the hole can be large and cause numerical instabilities.

A better way to handle leakage is by exploiting the HVAC model. The compartment surface that is leaking can be thought of as a large HVAC vent that connects via a very small duct to the outside. This allows the leakage to be removed over a large area in the domain (just as it would be in reality) while correctly capturing the actual area of the leakage path. There are two approaches to this. The first approach is by exploiting only pressure zones. A pressure zone is a user-specified volume within the computational domain that is entirely surrounded by solid obstructions. For example, the interior of a closed room can be, and should be, declared a pressure zone. In this approach surfaces within a pressure zone are denoted as leaking and those surfaces can be considered an HVAC vent that connects to the outside via a tiny duct whose area is the leakage area. This leakage approach will prevent a compartment from seeing large pressure changes as fires grow and decay, but it cannot account for effects like exterior wind or the stack effect. The second approach is intended for leaks with well defined locations (a cracked open door where the crack size is subgrid) or for leaks where the stack effect is important. It uses the local pressure (which includes the zone pressure), which allows for leakage to vary in magnitude.

Pressure Zone Leakage

The pressure zone leakage approach is intended to capture the bulk leakage that occurs through walls. This approach assumes that the amount of leaking gas is very small and that it will exchange sufficient heat as it moves through a wall to be at the same temperature as the wall surface. With this approach the pressure between the source and destination zones is used to compute a leakage flow via the HVAC model. That flow

is then uniformly imposed over all surfaces designated as part of the leakage path. The first step is to define pressure zones, leakage areas, and a description of the surfaces through which leaking occurs:

```
&ZONE XB=..., LEAK_AREA(0)=0.0001 /
&ZONE XB=..., LEAK_AREA(1)=0.0002, LEAK_AREA(0)=0.0003 /
&SURF ID='LEAKY EXTERIOR WALL',..., LEAK_PATH=1,0 /
&SURF ID='LEAKY INTERIOR WALL',..., LEAK_PATH=1,2 /
```

The first line designates a region of the computational domain to be Pressure Zone 1. Note that the order of the ZONE lines is important; that is, the order implicitly defines Zone 1, Zone 2, etc. Zone 0 is by default the ambient pressure exterior. In this example, a leak exists between Zone 1 and the exterior Zone 0, and the area of the leak is 0.0001 m² (1 cm by 1 cm hole, for example). Zone 2 leaks to Zone 1 (and vice versa) with a leak area of 0.0002 m². Zone 2 also leaks to the outside with an area of 0.0003 m². Note that zones need not be physically connected for a leak to occur, but in each zone, except for the exterior, there must be some surface with a designated LEAK_PATH. Here, in Zones 1 and 2 there are surfaces defined by both 'LEAKY EXTERIOR WALL' and 'LEAKY INTERIOR WALL' so as to provide a surface over which to apply the leakage. Leakage is uniformly distributed over all of the solid surfaces assigned the LEAK_PATH. The order of the two pressure zones designated by LEAK_PATH is unimportant, and the solid obstructions where the leakage is applied need not form a boundary between the two zones.

The volume flow, \dot{V} , through a leak of area A_L is given by

$$\dot{V}_{\text{leak}} = A_L \text{sign}(\Delta p) \sqrt{2 \frac{|\Delta p|}{\rho_{\infty}}} \quad (9.6)$$

where Δp is the pressure difference between the adjacent compartments (in units of Pa) and ρ_{∞} is the ambient density (in units of kg/m³). The discharge coefficient normally seen in this type of formula is assumed to be 1.

In a typical building as the interior pressure rises, the leakage area will grow as small gaps, cracks, and other leakage paths open up. Leakage tests performed according to test standards such as ASTM E 779 provide two additional data points to quantify this behavior. These are the LEAK_PRESSURE_EXPONENT and the LEAK_REFERENCE_PRESSURE. The use of these additional inputs are shown in the equation below as n and Δp_{ref} respectively where $A_{L,\text{ref}}$ is given by LEAK_AREA.

$$A_L = A_{L,\text{ref}} \left(\frac{\Delta p}{\Delta p_{\text{ref}}} \right)^{n-0.5} \quad (9.7)$$

By default, $n = 0.5$ and $\Delta p_{\text{ref}} = 4$ Pa, meaning that the leak area will not change with pressure unless you specify an exponent other than 0.5.

The HVAC output quantities can be used to determine the leakage flows. FDS names the duct connecting Zone A with Zone B 'LEAK A B' and the duct nodes 'LEAK A B' for the Zone A side of the leak and 'LEAK B A' for the Zone B side. Note that for the duct names, FDS will use the lower numbered zone as Zone A.

FDS is limited by default to a maximum of 200 ZONE inputs. This can be increased if needed by the parameter MAX_LEAK_PATHS on the MISC line.

Localized Leakage

The local leakage approach is intended to represent leakage through a specific crack. For example, a cracked open door might have an opening that is too small to resolve with the grid. One would; however, still want to capture the fact that hot gases could escape the top of the crack and cold gases enter the bottom. The

local leakage approach uses the local pressure rather than just the zone pressure. Therefore, one can define multiple leakage paths for different windows, over the height of a door, or over the height of a tall stairwell where stack effect might be important. To use this approach two VENT inputs are linked via an HVAC input with TYPE_ID='LEAK'. In the example below a 0.001 m² leakage path is created between the VENT with ID='VENT 1' and the VENT with ID='VENT 2'. Note that the SURF_ID for a VENT with localized leakage is not 'HVAC'. The input must correspond to a SURF input. Wall heat transfer will be computed based upon the inputs on the referenced SURF input.

```
&VENT XB=...,SURF_ID='SURF 1',ID='VENT 1'/
&VENT XB=...,SURF_ID='SURF 2',ID='VENT 2'/
&HVAC ID='LEAK1',TYPE_ID='LEAK',VENT_ID='VENT 1',VENT2_ID='VENT 2',AREA=0.001/
```

This will create a duct with the name 'LEAK1' whose two nodes will be named VENT_ID and VENT2_ID. If the leakage path is to connect to the ambient outside the domain, then set VENT2_ID='AMBIENT'. In this case the second node will be the first node name with AMB appended (e.g. 'VENT 1 AMB'). Note that each VENT must lie in one pressure zone; however, it may span more than one MESH.

Unlike the zone leakage approach, this approach has the option to preserve the energy of the gas flowing through the leak. For example, for door crack using with pressure zone leakage, the outflowing gas at the top of the door would always be the same temperature as the outside of the door. To maintain hot gas flowing out of the leak, add LEAK_ENTHALPY=.TRUE. to the HVAC input. For each outflowing wall cell, this will compute the enthalpy difference between the temperature of the leak flow and the temperature of the surface and add it as a source of heat to the adjacent gas cell. The default value is LEAK_ENTHALPY=.FALSE.

This approach also has the option of changing the flow loss by specifying LOSS on the HVAC input. The default is LOSS=1; the same as for pressure zone leakage.

Note, for this approach it is not required that one have pressure zones defined; however, it is recommended to do so if different pressure zones do in fact exist in the model.

Example Case: door_crack

This example involves a small compartment that contains a fan in one wall and a closed door with leakage at its bottom in the opposite wall. A small (160 kW) fire is added to the compartment. Initially, the pressure rises due to the heat from the fire and the fan blowing air into the compartment. Eventually the pressure rise inside the compartment exceeds the maximum pressure of the fan, at which point the compartment begins to exhaust from both the fan and the leakage. Pressure will continue to rise due to the fire until the pressure relief due to leakage and back flow through the fan equals the pressure increase from the fire.

9.4 Pressure Boundary Conditions

In some situations, it is more convenient to specify a pressure, rather than a velocity, at a boundary. Suppose, for example, that you are modeling the interior of a tunnel, and a wind is blowing at one of the portals that affects the overall flow within the tunnel. If (and only if) the portal is defined using an OPEN vent, then the *dynamic pressure* at the boundary can be specified like this:

```
&VENT XB=..., SURF_ID='OPEN', DYNAMIC_PRESSURE=2.4, PRESSURE_RAMP='wind' /
&RAMP ID='wind', T= 0.0, F=1.0 /
&RAMP ID='wind', T=30.0, F=0.5 /
.
.
```

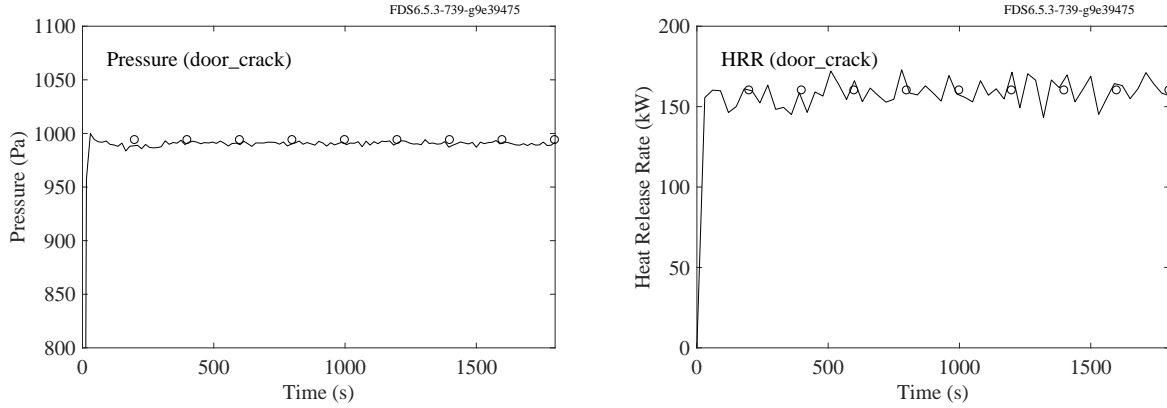



Figure 9.11: Output of `door_crack` test case. Symbols are expected values.

The use of a *dynamic pressure* boundary affects the FDS algorithm as follows. At `OPEN` boundaries, the hydrodynamic pressure (head) H is specified as

$$\begin{aligned} H &= \text{DYNAMIC_PRESSURE} / \rho_{\infty} + |\mathbf{u}|^2 / 2 \quad (\text{outgoing}) \\ H &= \text{DYNAMIC_PRESSURE} / \rho_{\infty} \quad (\text{incoming}) \end{aligned} \quad (9.8)$$

where ρ_{∞} is the ambient density and \mathbf{u} is the most recent value of the velocity on the boundary. The `PRESSURE_RAMP` allows you to alter the pressure as a function of time. Note that you do not need to ramp the pressure up or down starting at zero, like you do for various other ramps. The net effect of a positive dynamic pressure at an otherwise quiescent boundary is to drive a flow into the domain. However, a fire-driven flow of sufficient strength can push back against this incoming flow.

The following lines, taken from the sample case, `pressure_boundary`, demonstrates how to specify a time-dependent pressure boundary at the end of a tunnel. The tunnel is 10 m long, 1 m wide, 1 m tall with a fire in the middle and a pressure boundary imposed on the right side. The left side (`XMIN`) is just an `OPEN` boundary with no pressure specified. It is assumed to be at ambient pressure.

```
&VENT MB = 'XMIN' SURF_ID = 'OPEN' /
&VENT MB = 'XMAX' SURF_ID = 'OPEN', DYNAMIC_PRESSURE=2.4, PRESSURE_RAMP='wind_ramp' /
&RAMP ID='wind_ramp', T= 0., F= 1. /
&RAMP ID='wind_ramp', T=15., F= 1. /
&RAMP ID='wind_ramp', T=16., F=-1. /
```

Figure 9.12 shows two snapshots from Smokeview taken before and after the time when the positive pressure is imposed at the right portal of a tunnel. The fire leans to the left because of the preferential flow in that direction. It leans back to the right when the positive pressure is directed to become negative.

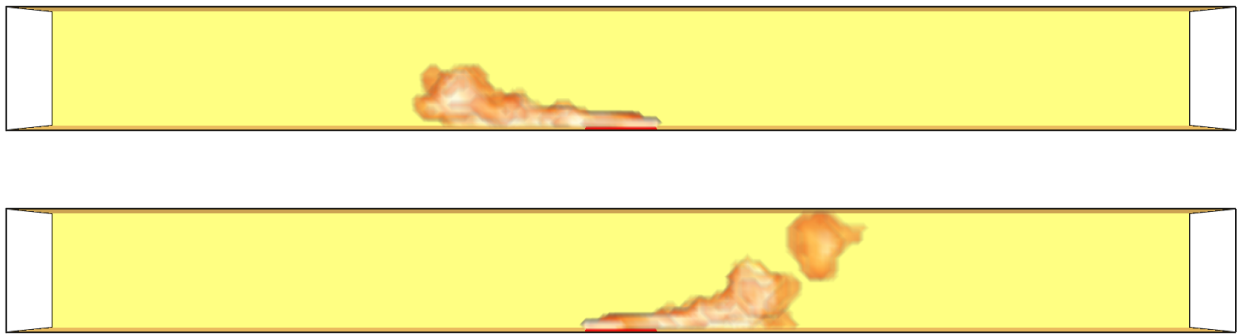


Figure 9.12: Snapshots from the sample case `pressure_boundary` showing a fire in a tunnel leaning left, then right, due to a positive, then negative, pressure imposed at the right portal.

9.5 Special Flow Profiles

By default, the air injected at a vent has a uniform or “top hat” velocity profile, but the parameter `PROFILE` on the `SURF` line can yield other profiles.

Parabolic

`PROFILE='PARABOLIC'` produces a parabolic profile with `VEL` (m/s) being the maximum velocity or `VOLUME_FLOW` (m³/s) being the desired volume flow. As an example, the test case in the `Flowfields` examples folder called `parabolic_profile.fds` demonstrates how you can create a circular or rectangular vent, each with a parabolic inlet profile. The two `VENT` lines below create circular and rectangular inlets, respectively, each of which inject air (or the background gas) at a rate of 0.5 m³/s into the compartment.

```
&SURF ID='BLOW', VOLUME_FLOW=-0.5, PROFILE='PARABOLIC' /
&VENT SURF_ID='BLOW', XB=-3,1,-3,1,0,0, RADIUS=2., XYZ=-1,-1,0 /
&VENT SURF_ID='BLOW', XB= 3,5,-2,1,0,0 /
```

The purpose of the test case is to ensure that the proper amount of gas (in this case nitrogen) is forced into the compartment, as confirmed by the pressure rise. Figure 9.13 displays a comparison of the calculated versus the exact pressure rise in a large compartment with these two parabolic vents. The pressure should rise according to the equation and analytical solution:

$$\frac{dp}{dt} = \frac{\gamma \dot{V}}{V} p \implies p(t) - p_0 = p_0 \left(e^{\frac{\gamma \dot{V}}{V} t} - 1 \right) \quad (9.9)$$

where the ratio of specific heats, $\gamma = 1.4$, volume flow rate, $\dot{V} = 1$ m³/s, volume, $V = 4000$ m³, and ambient pressure, $p_0 = 101325$ Pa. Note that to obtain this simple result, FDS was run with the option `CONSTANT_SPECIFIC_HEAT_RATIO` set to `true`.

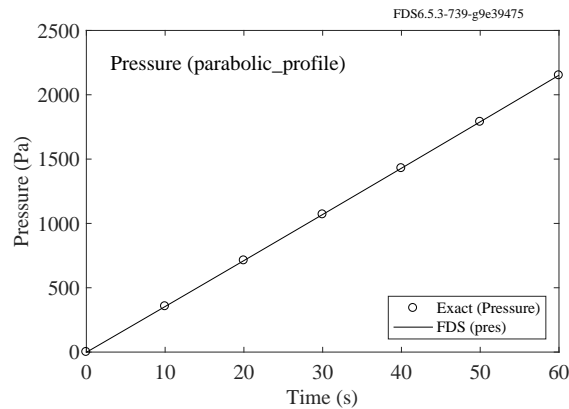


Figure 9.13: Results of the `parabolic_profile` test case

Boundary Layer (Circular Vent)

`PROFILE='BOUNDARY_LAYER'` may be used for circular vents created using `RADIUS`. By adding `VEL_BULK` on the `SURF` line together with `VEL`, FDS will produce a plug flow core profile, with max velocity given by

VEL, and a quadratic profile in the boundary layer. The form of the profile is illustrated in Fig. 9.14 for a circular vent. The functional form of the velocity profile (here taken a vertical profile) is

$$w(r) = \begin{cases} w_{\max} & \text{if } r \leq R - \delta \\ w_{\max} \left(1 - \left(\frac{r - (R - \delta)}{\delta} \right)^2 \right) & \text{if } R - \delta < r \leq R \end{cases} \quad (9.10)$$

The bulk velocity is the volumetric flow rate divided by the circular flow area. VEL_BULK is negative pointing into the domain. The boundary layer thickness is δ . This feature is handy for dealing with the case where the maximum velocity is higher than the bulk velocity. Here is an example input file line:

```
&SURF ID='JET', VEL=-69, VEL_BULK=-53, PROFILE='BOUNDARY LAYER', ... /
```

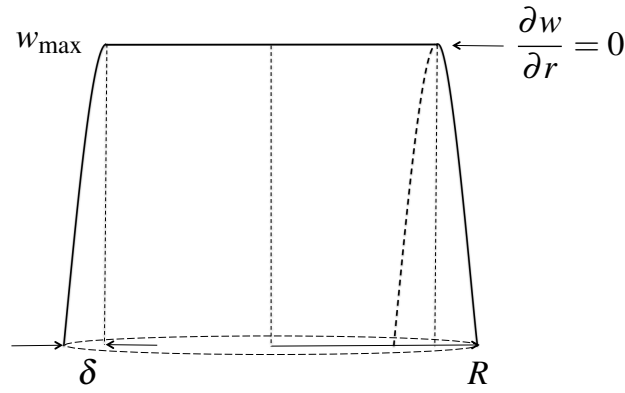


Figure 9.14: Boundary layer profile.

Chapter 10

Atmospheric Effects

The chapter describes techniques for modeling the atmospheric boundary layer.

10.1 Temperature Stratification

Typically, in the first few hundred meters of the atmosphere, the temperature decreases several degrees Celsius per kilometer. This small temperature change is important when considering the rise of smoke since the temperature of the smoke decreases rapidly as it rises. The `LAPSE_RATE` of the atmosphere can be specified on the `MISC` line in units of $^{\circ}\text{C}/\text{m}$. A negative sign indicates that the temperature *decreases* with height. This need only be set for outdoor calculations where the height of the domain is tens or hundreds of meters. The default value of the `LAPSE_RATE` is $0^{\circ}\text{C}/\text{m}$.

Alternatively, you can specify `RAMP_TMP0_Z` on the `MISC` line if you want a non-linear change in temperature with height. For example,

```
&MISC ..., RAMP_TMP0_Z='T profile', TMPA=34.5, ... /

&RAMP ID='T profile', Z= 0.001 , F= 1.0258 /
&RAMP ID='T profile', Z= 0.500 , F= 1.0019 /
&RAMP ID='T profile', Z= 1.000 , F= 1.0000 /
&RAMP ID='T profile', Z= 1.900 , F= 0.9986 /
&RAMP ID='T profile', Z= 3.000 , F= 0.9977 /
&RAMP ID='T profile', Z= 4.000 , F= 0.9972 /
```

The value of `F` is the ratio of the temperature in degrees K to `TMPA`.

By default, FDS assumes that the density and pressure decrease with height, regardless of the application or domain size. For most simulations, this effect is negligible, but it can be turned off completely by setting `STRATIFICATION=.FALSE.` on the `MISC` line.

10.1.1 Stack Effect

Tall buildings often experience buoyancy-induced air movement due to temperature differences between the interior and exterior, known as *stack effect* [?]. These temperature differences create flows within vertical shafts (stairwells, atriums, elevator shafts, etc.) due to leaks or openings at different levels. To simulate this phenomenon in FDS, you must include the entire building, or a substantial fraction of it, both inside and out, in the computational domain. It is important to capture the pressure and density decrease in the atmosphere based on the specified temperature profile that is entered on the `MISC` line.

For the case where the stack flow is through small leakage paths, divide the building into one or more pressure ZONES. The leakage paths can be defined in terms of HVAC components. Note that the leakage model combines all leaky surfaces over the entire height of the building and as a result averages out the pressure gradients. For doing stack effect calculations individual leakage paths should be defined. A simple example is described next.

Example Case: Atmospheric_Effects/stack_effect

The *stack_effect* test case is a two-dimensional simulation of a 100 m tall building whose interior air temperature is slightly warmer than its exterior. The building has leakage paths at the top and ground floors only. Since the inside air temperature is slightly warmer, the inside air pressure is slightly higher as well, and it drives air out of the building and in turn draws air into the building at the ground level. The interior air temperature, T_b , is initially 20 °C (293 K), and the exterior air temperature, T_∞ , is 10 °C (283 K). The LAPSE_RATE is set to 0 °C/m; thus, $T_0(z) = T_\infty$ outside the building and $T_0(z) = T_b$ inside the building. Two small leakage openings are defined 2.5 m above the ground floor and 2.5 m below the roof using the HVAC solver. Each opening is given an area of 0.01 m² and a loss coefficient of 2 (e.g., an entrance loss into the leak path of 1 and an exit loss out of the leak path of 1 both of which represent a sharp edge opening).

The initial density stratification inside and outside the building can be calculated using the relation:

$$\frac{\rho_0(z)}{\rho_\infty} = \exp\left(-\frac{g\bar{W}}{RT_0}z\right) \quad (10.1)$$

where R is the universal gas constant, g is the acceleration of gravity, and \bar{W} is the average molecular weight of the air, z is the height above the GROUND_LEVEL, and T_0 is the ambient temperature. Applying this formula to the external and internal locations at the lower and upper vents results in densities of 1.2412, 1.1989, 1.2272, and 1.1858 kg/m³, respectively.

Since the openings in the building are equally spaced over its height, the neutral plane should be close to its midpoint. This can be computed from:

$$\frac{\Delta H}{H_n} = 1 + \frac{T_b}{T_\infty} \quad (10.2)$$

where H_n is the neutral plane height above the bottom vent and $\Delta H = 95$ m is the distance between the leak points. This gives a neutral plane of 46.68 m above the lower vent or 49.18 m above the bottom of the building. Note that this is close to the midpoint value of 50 m above the bottom of the building. The pressure difference across the building's wall is computed from

$$\Delta p = \frac{\bar{W} p_0(z) g \Delta z}{R} \left(\frac{1}{T_\infty} - \frac{1}{T_b} \right) \quad (10.3)$$

where Δz is the distance from the leak point to the neutral plane. Using the neutral plane location, the Δz values are -46.68 m for the lower vent and +48.32 m for the upper vent which respectively result in lower and upper vent pressure differences of -19.4 Pa and +20.4 Pa. Using the loss of 2 and the pressure difference in the HVAC momentum equation results in a steady-state inflow velocity at the bottom of 3.95 m/s and an outflow velocity at the top of 4.15 m/s. Results for velocity and density are shown in Fig. 10.1.

10.2 Wind

If you want to simulate a wind, set $U0$, $V0$, and $W0$ to the three components of the prevailing wind field on the MISC line. By default, their values are 0 m/s. For example, if the mean wind is 10 m/s in the northeast

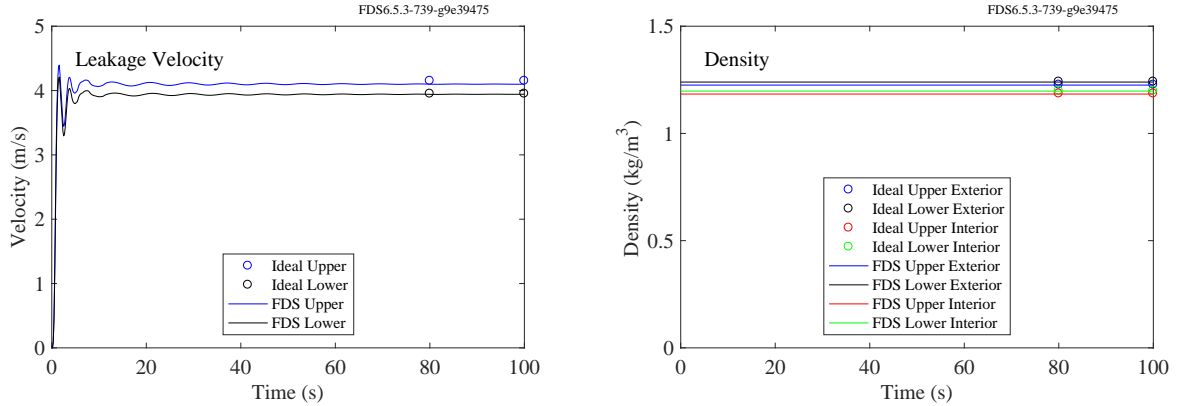


Figure 10.1: (Left) Velocity at the upper and lower vents for the `stack_effect` case. (Right) Upper and lower exterior and interior densities.

direction, set both `U0` and `V0` to 7.07 m/s. Setting these values, however, does not ensure that this flow field will persist. There are several ways to maintain the wind field. One way is to specify the normal component of the wind velocity at one or two exterior boundaries of the computational domain. This is essentially like creating a giant wall of fans on one side of the domain and a passive open boundary on the other. This is not a particularly good strategy to use because the lateral (i.e. tangential to the wind field) boundary conditions are not well-defined.

The better way to specify a wind field is as follows. Suppose there are anemometers located at specific locations within the domain which continuously record wind speed and direction. The process of steering the solution of the mass, momentum, and energy equations to match the statistics of the data gathered at the weather stations is known as *data assimilation*. This branch of modeling is early in its development, but very sophisticated (translation: *complicated*) methods already exist [?] and are employed in operational weather forecasting models. In FDS, you may invoke the most rudimentary of data assimilation techniques, a method called *nudging*, where you add a mean forcing term to the momentum equation to “nudge” the solution toward a desired result. Currently, FDS can only affect the mean flow velocities. To turn on this capability, specify the mean wind using `U0` and `V0` (rarely would you use `W0`), along with the logical parameter `MEAN_FORCING(1:2)=.TRUE.` on `MISC`. This will cause FDS to drive the mean velocity components toward the values of `U0` and `V0`. For example, to point the wind in the northeast direction (assuming that the positive x axis points eastward) at 10 m/s, set the following parameters on the `MISC` line:

```
&MISC ..., MEAN_FORCING(1:2)=.TRUE., U0=7.07, V0=7.07, DT_MEAN_FORCING=0.1,
      RAMP_U0_T='t-ramp', ..., RAMP_V0_Z='z-ramp' /
```

Note that you must also specify a time scale for relaxation, `DT_MEAN_FORCING`. The shorter this time scale, the faster the flow field will “catch up” to the mean wind, but at the expense of possibly washing out important flow structures. Try several values of this time scale for your particular simulation to determine what is appropriate for your case. There is no firm guidance in the atmospheric modeling literature.

The velocity components `U0`, `V0`, and `W0` can be made functions of time and/or height using the `RAMP` functions: `RAMP_U0_T`, `RAMP_V0_T`, and `RAMP_W0_T` to vary the velocity components with time; and `RAMP_U0_Z`, `RAMP_V0_Z`, and `RAMP_W0_Z` to vary the velocity components with height. For example,

```
&RAMP ID='t-ramp', T= 0., F=0. /
&RAMP ID='t-ramp', T=100., F=1. /
```

```
&RAMP ID='z-ramp', Z= 0., F=1.0 /
&RAMP ID='z-ramp', Z=200., F=1.5 /
&RAMP ID='z-ramp', Z=500., F=1.8 /
```

The parameter *F* represents the fraction of the specified velocity component as a function of time *T* (s) or height *Z* (m).

For an outdoor flow, all other boundaries (except the ground) should be set to *OPEN*. You may omit a region of the flow domain from forcing using the *HOLE* feature. If a *HOLE* is used with *MEAN_FORCING* it is advisable to also set *PROJECTION=.TRUE.* on *MISC*; otherwise the combination of *U0* in the bulk region and zero velocity in the *HOLE* region creates an initial divergence error that will take some time to wash out of the domain.

Example Wind Simulation

An input file called *wind_example.fds* in the *Atmospheric_Effects* folder provides an example of a wind simulation. The computational domain is a flat landscape that is 1 km by 1 km by 100 m high. There are various rectangular blocks scattered about to represent buildings. The background wind velocity components vary with height, *z*, and time, *t*:

$$u(z,t) = 10 \cos\left(\frac{2\pi t}{50}\right) \left(\frac{z}{5}\right)^{0.2} \text{ m/s} ; \quad v(z,t) = 10 \sin\left(\frac{2\pi t}{50}\right) \left(\frac{z}{5}\right)^{0.2} \text{ m/s} \quad (10.4)$$

The 360° turn of the wind in 50 s is not realistic, but this case is intended to test the ramp functionality. The plots in Fig. 10.2 display some results of the simulation. On the left is a plot showing the mean velocity components of the simulated wind field compared to the specified (ideal) values. Notice that the simulated wind field lags the specified field by roughly 1 s because the relaxation parameter *DT_MEAN_FORCING* was set to 1 s. This time scale falls into the denominator of the forcing term that “nudges” the simulated wind to follow the specified time history. If it is too small, the “nudging” term becomes too dominant. The right hand plot of Fig. 10.2 displays the specified wind profile compared to the simulation after the wind has turned full circle. The two profiles should be roughly aligned to indicate that the lateral boundary conditions of the mean forcing scheme are working properly.

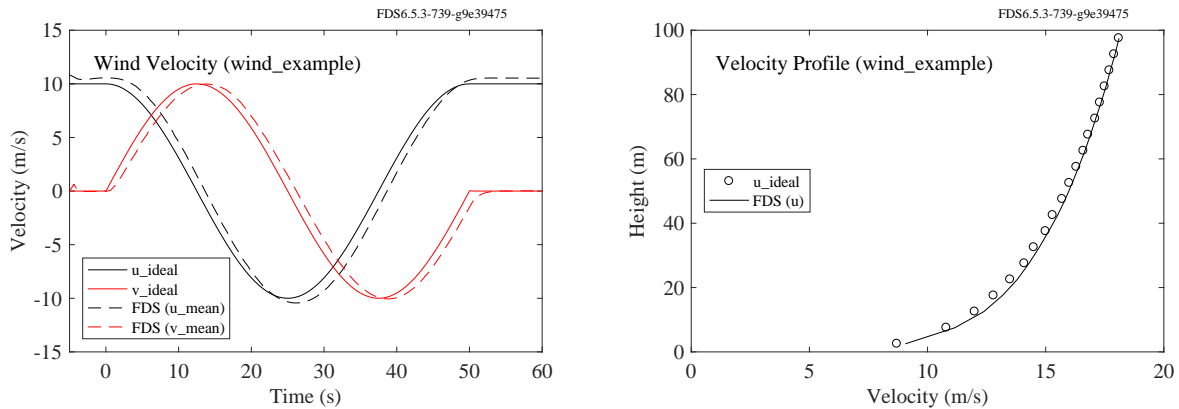


Figure 10.2: (Left) Mean velocity components of the wind vs. the specified values. (Right) Simulated vs. specified wind profile.

10.2.1 Specifying a “Wall of Wind”

An alternative to `MEAN_FORCING` is to specify a power law wind profile at an external boundary of the computational domain. This essentially creates a “wall of wind” and for early versions of FDS it was the recommended method for specifying a wind. However, the `MEAN_FORCING` technique described above is preferable because it handles the lateral boundaries of the computational domain in a more natural way.

To specify a “wall of wind”, set `PROFILE='ATMOSPHERIC'` on the `SURF` line that is assigned to an exterior lateral boundary of the computational domain. This generates a power law atmospheric wind profile of the form $u = u_0(z/z_0)^p$ where z is the height above the ground. If an atmospheric profile is prescribed, also prescribe `Z0` for z_0 and `PLE` for p . `VEL` specifies the reference velocity u_0 . Note that z_0 is not the ground, but rather the height above the ground where the wind speed is measured, like an elevated weather station. It is assumed that the ground is located at 0 m; to change this assumption, set `GROUND_LEVEL` on the `MISC` line to be the appropriate elevation. Be careful not to apply an atmospheric velocity profile (e.g. negative z) below `GROUND_LEVEL` or FDS will stop with an error.

Chapter 11

User-Specified Functions

Many of the parameters specified in the FDS input file are fixed constants. However, there are several parameters that may vary in time, temperature, or space. The namelist groups, `RAMP` and `TABL`, allow you to control the behavior of these parameters. `RAMP` allows you to specify a function with one independent variable (such as time) and one dependent variable (such as velocity). `TABL` allows you to specify a function of multiple independent variables (such as a solid angle) and multiple dependent variables (such as a sprinkler flow rate and droplet speed).

11.1 Time-Dependent Functions

At the start of any calculation, the temperature is ambient everywhere, the flow velocity is zero everywhere, nothing is burning, and the mass fractions of all species are uniform. When the calculation starts temperatures, velocities, burning rates, etc., are ramped-up from their starting values because nothing can happen instantaneously. By default, everything is ramped-up to their prescribed values in approximately 1 s. However, you can control the rate at which things turn on, or turn off, by specifying time histories either with pre-defined functions or with user-defined functions. The parameters `TAU_Q`, `TAU_T`, and `TAU_V` indicate that the heat release rate (`HRRPUA`); surface temperature (`TMP_FRONT`); and/or normal velocity (`VEL`, `VOLUME_FLOW`), or `MASS_FLUX_TOTAL` are to ramp up to their prescribed values in `TAU` seconds and remain there. To prescribe different heat release rate ramps, the `TAU_Q` parameter can be defined as a negative value (t -squared growth rate) or a positive value (\tanh growth rate), which results in a time-dependent heat release rate as

$$\dot{Q}(t) = \begin{cases} \dot{Q}_0 \left(\frac{t}{\tau}\right)^2 & \text{if } \text{TAU_Q is negative} \\ \dot{Q}_0 \cdot \tanh\left(\frac{t}{\tau}\right) & \text{if } \text{TAU_Q is positive} \end{cases} \quad (11.1)$$

where \dot{Q}_0 is the user-specified heat release rate. If the fire ramps up following a t -squared curve, then it remains constant after `TAU_Q` seconds. These rules apply to `TAU_T` and `TAU_V` as well. The default value for all `TAUS` is 1 s. If something other than a \tanh or t -squared ramp up is desired, then a user-defined function must be input. To do this, set `RAMP_Q`, `RAMP_T` or `RAMP_V` equal to a character string designating the ramp function to use for that particular surface type, then somewhere in the input file generate lines of the form:

```
&RAMP ID='rampname1', T= 0.0, F=0.0 /
&RAMP ID='rampname1', T= 5.0, F=0.5 /
&RAMP ID='rampname1', T=10.0, F=0.7 /
```

Here, T is the time, and F indicates the fraction of the heat release rate, wall temperature, velocity, mass fraction, etc., to apply. Linear interpolation¹ is used to fill in intermediate time points. Note that each set of RAMP lines must have a unique ID and that the lines must be listed with monotonically increasing T . Note also that the TAUS and the RAMPs are mutually exclusive. For a given surface quantity, both cannot be prescribed. As an example, a simple blowing vent can be controlled via the lines:

```
&SURF ID='BLOWER', VEL=-1.2, TMP_FRONT=50., RAMP_V='BLOWER RAMP', RAMP_T='HEATER
  RAMP' /
&RAMP ID='BLOWER RAMP', T= 0.0, F=0.0 /
&RAMP ID='BLOWER RAMP', T=10.0, F=1.0 /
&RAMP ID='BLOWER RAMP', T=80.0, F=1.0 /
&RAMP ID='BLOWER RAMP', T=90.0, F=0.0 /
&RAMP ID='HEATER RAMP', T= 0.0, F=0.0 /
&RAMP ID='HEATER RAMP', T=20.0, F=1.0 /
&RAMP ID='HEATER RAMP', T=30.0, F=1.0 /
&RAMP ID='HEATER RAMP', T=40.0, F=0.0 /
```

Use TAU_T or RAMP_T to control the ramp-ups for surface temperature. The surface temperature at time t , $T_w(t)$, is

$$T_w(t) = T_0 + f(t) (TMP_FRONT - T_0) \quad (11.2)$$

where $f(t)$ is the result of evaluating the RAMP_T at time t , T_0 is the ambient temperature, and TMP_FRONT is specified on the same SURF line as RAMP_T. Use TAU_MF(N) or RAMP_MF(N) to control the ramp-ups for either the mass fraction or mass flux of species N. For example:

```
&SURF ID='...', MASS_FLUX(1:2)=0.1,0.3, SPEC_ID(1:2)='ARGON','NITROGEN',
  TAU_MF(1:2)=5.,10. /
```

indicates that argon and nitrogen are to be injected at rates of $0.1 \text{ kg}/(\text{m}^2 \cdot \text{s})$ and $0.3 \text{ kg}/(\text{m}^2 \cdot \text{s})$ over time periods of approximately 5 s and 10 s, respectively.

Table 11.1 lists the various quantities that can be controlled by RAMPs.

11.2 Temperature-Dependent Functions

Thermal properties like conductivity and specific heat can vary significantly with temperature. In such cases, use the RAMP function like this:

```
&MATL ID          = 'STEEL'
  FYI             = 'A242 Steel'
  SPECIFIC_HEAT_RAMP = 'c_steel'
  CONDUCTIVITY_RAMP  = 'k_steel'
  DENSITY          = 7850. /

&RAMP ID='c_steel', T= 20., F=0.45 /
&RAMP ID='c_steel', T=377., F=0.60 /
&RAMP ID='c_steel', T=677., F=0.85 /

&RAMP ID='k_steel', T= 20., F=48. /
&RAMP ID='k_steel', T=677., F=30. /
```

¹By default, FDS uses a linear interpolation routine to find time or temperature-dependent values between user-specified points. The default number of interpolation points is 5000, more than enough for most applications. However, you can change this value by specifying NUMBER_INTERPOLATION_POINTS on any RAMP line.

Table 11.1: Parameters for controlling the time-dependence of given quantities.

Quantity	Group	Input Parameter(s)	TAU	RAMP ID
Heat Release Rate	SURF	HRRPUA	TAU_Q	RAMP_Q
Heat Flux	SURF	NET_HEAT_FLUX, etc.	TAU_Q	RAMP_Q
Temperature	SURF	TMP_FRONT	TAU_T	RAMP_T
Velocity	SURF	VEL	TAU_V	RAMP_V
Volume Flux	SURF	VOLUME_FLOW	TAU_V	RAMP_V
Mass Flux	SURF	MASS_FLUX_TOTAL	TAU_V	RAMP_V
Mass Fraction	SURF	MASS_FRACTION (N)	TAU_MF (N)	RAMP_MF (N)
Mass Flux	SURF	MASS_FLUX (N)	TAU_MF (N)	RAMP_MF (N)
Particle Mass Flux	SURF	PARTICLE_MASS_FLUX	TAU_PART	RAMP_PART
External Heat Flux	SURF	EXTERNAL_FLUX	TAU_EF	RAMP_EF
Pressure	VENT	DYNAMIC_PRESSURE		PRESSURE_RAMP
Flow	PROP	FLOW_RATE	FLOW_TAU	FLOW_RAMP
Gravity	MISC	GVEC (1)		RAMP_GX
Gravity	MISC	GVEC (2)		RAMP_GY
Gravity	MISC	GVEC (3)		RAMP_GZ

Note that for temperature ramps, as opposed to time ramps, the parameter *F* is the actual physical quantity, not just a fraction of some other quantity. Thus, if `CONDUCTIVITY_RAMP` is used, there should be no value of `CONDUCTIVITY` given. Note also that for values of temperature, *T*, below and above the given range, FDS will assume a constant value equal to the first or last *F* specified. Note also that the `DENSITY` of a material cannot be controlled with a `RAMP` function.

11.3 Spatially-Dependent Velocity Profiles

Similar to using `PROFILE='ATMOSPHERIC'` on `SURF`, it is possible to specify `PROFILE='RAMP'` to generate 1D or 2D profiles of the normal component of velocity on a surface. The following code generates a $u(z)$ profile on the 'XMIN' boundary.

```
&SURF ID='inlet', VEL=-7.72, PROFILE='RAMP', RAMP_V_Z='u_prof' /
&VENT MB='XMIN', SURF_ID='inlet' /

&RAMP ID='u_prof', T=0., F=0. /
&RAMP ID='u_prof', T=0.0098, F=0. /
&RAMP ID='u_prof', T=0.01005, F=0.2474 /
&RAMP ID='u_prof', T=0.01029, F=0.4521 /
&RAMP ID='u_prof', T=0.01077, F=0.6256 /
&RAMP ID='u_prof', T=0.01174, F=0.7267 /
&RAMP ID='u_prof', T=0.01368, F=0.8238 /
&RAMP ID='u_prof', T=0.01562, F=0.8795 /
&RAMP ID='u_prof', T=0.01756, F=0.9378 /
&RAMP ID='u_prof', T=0.0195, F=0.9663 /
&RAMP ID='u_prof', T=0.02144, F=0.9922 /
&RAMP ID='u_prof', T=0.02338, F=0.9987 /
&RAMP ID='u_prof', T=0.02532, F=1 /
&RAMP ID='u_prof', T=0.0588, F=1 /
```

Note that v indicates the velocity component normal to the surface. You can also specify `RAMP_V_X` and `RAMP_V_Y` and add these to the `SURF`. Note that only profiles in the planar directions will affect a given surface. That is, if the surface is oriented in the $y-z$ plane, only `RAMP_V_Y` and `RAMP_V_Z` apply. In the `RAMP` definition T is the independent variable and F is the dependent variable. In this example, T is the z coordinate in meters and F is the factor multiplying `VEL` on the `SURF` line. Two ramps may be applied to a surface.

T does not need to be directly related to the FDS mesh. The velocity points will be interpolated linearly by the ramp function. In fact, this functionality is convenient for taking experimental data directly as a boundary condition to FDS. You basically just need to list T and F from the data (you can set `VEL=-1` if you want). The results for this particular case can be seen in the section, “Flow Over a Backward Facing Step”, in the FDS Verification Guide [?]. In this problem, the step height is $h = 0.0098$ m, and the specification of the inlet profile is critical to correctly matching the reattachment point downstream of the step.

Chapter 12

Chemical Species

FDS was designed primarily to study fire phenomena, and much of the basic chemistry of combustion is handled with a minimum of user inputs. However, there are many applications in which you might want to simulate the movement of gases in the absence of fire, or additional chemical species might be added to a simulation that involves fire. Gas species are defined with the input group `SPEC`. This input group is used to define both *primitive* gas species and *lumped* species (mixtures of one or more primitive `SPEC`).

There are different roles that a gas species might play in a simulation. A gas species might be explicitly tracked. In other words, a transport equation is solved for it. A gas species might just serve as the “background” species. Note that no transport equation is needed for the background species as it is whatever mass remains after all other tracked species have been accounted for. Or, a gas species might be one component of a mixture of gases that are transported together. For example, FDS exploits the idea that the products of combustion from a fire mix and travel together; you only need to solve one transport equation for this “lumped species.” The default combustion model in FDS assumes that the reaction is mixing-controlled, and transport equations for only the lumped species—Fuel and Products—are solved (the lumped species Air is the default background). There is no reason to solve individual (and costly) transport equations for the major reactants and products of combustion—Fuel, O₂, CO₂, H₂O, N₂, CO and soot—because they are all pre-tabulated functions of the three lumped species. More detail on combustion is given in Chapter 13. For the moment, just realize that you need not, *and should not*, explicitly list the reactants and products of combustion using `SPEC` lines if all you want is to model a fire involving a hydrocarbon fuel.

12.1 Specifying Primitive Species

The `SPEC` input is used to define a gas species in FDS. Once defined the species can be tracked as a single species (i.e., a primitive species) and/or the species can be used as part of one or more lumped species, also defined with `SPEC`. It is possible for a species to be both part of a lumped species and tracked separately. Often an extra gas introduced into a calculation is the same as a product of combustion, like water vapor from a sprinkler or carbon dioxide from an extinguisher. These gases are tracked separately. Thus, water vapor generated by the combustion is tracked via the Products lumped species variable and water vapor generated by evaporating sprinkler droplets is tracked via its own transport equation.

If a species is only to be used as part of one or more lumped species, `LUMPED_COMPONENT_ONLY=.TRUE.` must be added to the `SPEC` line. This tells FDS not to allocate space for the species in the array of tracked gases. If a species is to be used as the background species, the parameter `BACKGROUND=.TRUE.` should be set on the `SPEC` line. This also tells FDS not to allocate space for the species in the array of tracked gases. A species with `LUMPED_COMPONENT_ONLY=.TRUE.` cannot be used as an individual species, and it cannot be used as the background species. However, a primitive species with `BACKGROUND=.TRUE.` can also be used

as part of a lumped species definition. Note that the default background species is `AIR` which is defined as a lumped species consisting of N_2 , O_2 , CO_2 , and H_2O . If no background species is defined in the input file, then FDS will create the background species of `AIR` by internally creating the input lines shown in Example 2 in Section 12.2.

Note that while you can define as many species using `SPEC` as you wish in an input file, any namelist input tied to a list of species, such as any `SPEC_ID` input or `MASS_FRACTION` input, is limited to using no more than 20 species.

12.1.1 Basics

Each `SPEC` line should include at the very least the name of the species via a character string, `ID`. Once the extra species has been declared, you introduce it at surfaces via the parameters `MASS_FRACTION(:)` or `MASS_FLUX(:)` along with the character array `SPEC_ID(:)`. A very simple example of how a gas—in this case hydrogen—can be introduced into the simulation is given by the simple input file called `gas_filling.fds`. The relevant lines are as follows:

```
&SPEC ID='HYDROGEN' /
&SURF ID='LEAK', SPEC_ID(1)='HYDROGEN', MASS_FLUX(1)=0.01667, RAMP_MF(1)='leak_ramp' /
&RAMP ID='leak_ramp', T= 0., F=0.0 /
&RAMP ID='leak_ramp', T= 1., F=1.0 /
&RAMP ID='leak_ramp', T=180., F=1.0 /
&RAMP ID='leak_ramp', T=181., F=0.0 /
&VENT XB=-0.6,0.4,-0.6,0.4,0.0,0.0, SURF_ID='LEAK', COLOR='RED' /
&DUMP MASS_FILE=.TRUE. /
```

The hydrogen is injected through a 1 m by 1 m vent at a rate of $0.01667 \text{ kg}/(\text{m}^2 \cdot \text{s})$ and shut off after 3 min. The total mass of hydrogen at that point ought to be 3 kg (see Fig. 12.1). Notice that no properties were needed for the `HYDROGEN` because it is a species whose properties are included in Table 12.1. The background species in this case is assumed to be air. The mass flow rate of the hydrogen is controlled via the ramping parameter `RAMP_MF(1)`. The parameter `MASS_FILE=.TRUE.` instructs FDS to produce an output file that contains a time history of the hydrogen mass.

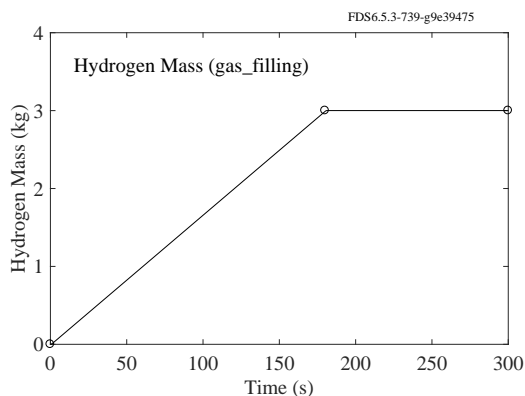


Figure 12.1: Hydrogen mass vs. time for `gas_filling` test case.

Initial Conditions

If the initial mass fraction of the gas is something other than zero, then the parameter `MASS_FRACTION_0` is used to specify it. For example, if you want the initial concentration in the domain to be 90% background (air) diluted with 10% argon, use

```
&SPEC ID='ARGON', MASS_FRACTION_0=0.1 /
```

Specifying Humidity

If you are using the default background lumped species `AIR`, then you can specify `HUMIDITY` on `MISC` to set the ambient mass fraction of water vapor. `HUMIDITY` is the relative humidity of water vapor in units of %. It is 40 % by default.

If you are defining the primitive species of `WATER VAPOR`, then `MASS_FRACTION_0` is independent of `HUMIDITY`. That is setting `MASS_FRACTION_0` for the species `WATER VAPOR` will not change the ambient humidity, it will add additional water vapor.

12.1.2 Specifying Gas and Liquid Species Properties

There are several options for specifying the properties of gas and liquid species.

Option 1: FDS Defined Species

Gases and liquids whose properties are tabulated within FDS are listed in Table 12.1. The physical properties of these species are known and do not need to be specified. When using one of these gases you need only specify the correct `ID` and provide, if needed, the initial mass fraction. FDS will then use precompiled data to compute the various thermophysical properties from 0 K to 5000 K.

Table 12.1: Optional gas and liquid species [?]

Species	Mol. Wt. (g/mol)	Formula	σ (Å)	ε/k (K)	Liquid	RadCal Surrogate
ACETONE	58.07914	C ₃ H ₆ O	4.6	560.2	Y	MMA
ACETYLENE	26.037280	C ₂ H ₂	4.033	231.8		PROPYLENE
ACROLEIN	56.063260	C ₃ H ₄ O	4.549	576.7	Y	MMA
AMMONIA	17.03052	NH ₃	2.9	558.3	Y	
ARGON	39.948000	Ar	3.42	124.0	Y	
BENZENE	78.11184	C ₆ H ₆	5.349	412.3	Y	TOLUENE
BUTANE	58.122200	C ₄ H ₁₀	4.687	531.4	Y	PROPANE
CARBON	12.0107	C	2.94	74.8		
CARBON DIOXIDE	44.009500	CO ₂	3.941	195.2		CARBON DIOXIDE
CARBON MONOXIDE	28.010100	CO	3.690	91.7	Y	CARBON MONOXIDE
CHLORINE	70.906	Cl ₂	4.217	316.0	Y	
DODECANE	170.33484	C ₁₂ H ₂₆	4.701	205.78	Y	N-HEPTANE
ETHANE	30.069040	C ₂ H ₆	4.443	215.7	Y	ETHANE
ETHANOL	46.068440	C ₂ H ₅ OH	4.530	362.6	Y	METHANOL
ETHYLENE	28.053160	C ₂ H ₄	4.163	224.7	Y	ETHYLENE
FORMALDEHYDE	30.025980	CH ₂ O	3.626	481.8	Y	METHANOL
HELIUM	4.002602	He	2.551	10.22	Y	
HYDROGEN	2.015880	H ₂	2.827	59.7	Y	
HYDROGEN ATOM	1.007940	H	2.31	123.6		
HYDROGEN BROMIDE	80.911940	HBr	3.353	449.0	Y	
HYDROGEN CHLORIDE	36.460940	HCl	3.339	344.7	Y	
HYDROGEN CYANIDE	27.025340	HCN	3.63	569.1	Y	
HYDROGEN FLUORIDE	20.006343	HF	3.148	330.0	Y	
HYDROGEN PEROXIDE	34.014680	H ₂ O ₂	3.02	106.5	Y	
HYDROPEROXY RADICAL	33.006740	HO ₂	3.02	106.5		
HYDROXYL RADICAL	17.007340	OH	2.66	92.1		
ISOPROPANOL	60.095020	C ₃ H ₇ OH	4.549	576.7	Y	METHANOL
LJ AIR	28.854760		3.711	78.6		
METHANE	16.042460	CH ₄	3.758	148.6	Y	METHANE
METHANOL	32.041860	CH ₃ OH	3.626	481.8	Y	METHANOL
N-DECANE	142.281680	C ₁₀ H ₂₂	5.233	226.46	Y	N-HEPTANE
N-HEPTANE	100.201940	C ₇ H ₁₆	4.701	205.75	Y	N-HEPTANE
N-HEXANE	86.175360	C ₆ H ₁₂	5.949	399.3	Y	N-HEPTANE
N-OCTANE	114.228520	C ₈ H ₁₈	4.892	231.16	Y	N-HEPTANE
NITRIC OXIDE	30.006100	NO	3.492	116.7	Y	
NITROGEN	28.013400	N ₂	3.798	71.4	Y	
NITROGEN ATOM	14.006700	N	2.66	92.1		
NITROGEN DIOXIDE	46.05500	NO ₂	3.992	204.88	Y	
NITROUS OXIDE	44.012800	N ₂ O	3.828	232.4	Y	
OXYGEN	31.998800	O ₂	3.467	106.7	Y	
OXYGEN ATOM	15.999400	O	2.66	92.1		
PROPANE	44.095620	C ₃ H ₈	5.118	237.1	Y	PROPANE

Table 12.1: Optional gas and liquid species (continued).

Species	Mol. Wt. (g/mol)	Formula	σ (Å)	ϵ/k (K)	Liquid	RadCal Surrogate
PROPYLENE	42.079740	C ₃ H ₆	4.678	298.9	Y	PROPYLENE
SOOT	10.910420	C _{0.9} H _{0.1}	3.798	71.4		SOOT
SULFUR DIOXIDE	64.063800	SO ₂	4.112	335.4	Y	
SULFUR HEXAFLUORIDE	146.055419	SF ₆	5.128	146.0		
TOLUENE	92.138420	C ₆ H ₅ CH ₃	5.698	480.0	Y	TOLUENE
WATER VAPOR	18.015280	H ₂ O	2.641	809.1	Y	WATER VAPOR

Option 2: User-Specified Properties

If the gas species is not included in Table 12.1, then you must specify its thermophysical properties. By using the inputs discussed below, you can also override the default properties for a pre-defined gas species. For a gas species not included in Table 12.1, its molecular weight, MW , should be specified on the SPEC line in units of g/mol, otherwise the molecular weight of nitrogen will be used. If the species is participating in a reaction, then the ENTHALPY_OF_FORMATION in units of kJ/mol must also be specified. Additional discussion on the enthalpy of formation can be found in Chapter 13. The remaining thermophysical properties of conductivity, diffusivity, enthalpy, viscosity, absorptivity (thermal radiation), and liquid properties are discussed below. Most properties can be defined as a constant value or as a temperature dependent look-up table using a RAMP. For the latter, if the temperature in a gas cell is above or below the endpoints of the RAMP, the endpoint value will be used. FDS will not extrapolate beyond the ends of the RAMP.

Conductivity

Conductivity can be specified in one of three ways: it can be defined as a constant using CONDUCTIVITY (W/(m · K)), it can be defined as a temperature vs. specific heat ramp using RAMP_K, or it can be computed by FDS using MW, PR_GAS on SPEC (default value is PR on MISC), and the Lennard-Jones potential parameters σ (SIGMALJ) and ϵ/k (EPSILONKLJ). If no inputs are specified, FDS will compute the conductivity using the MW and the Lennard-Jones parameters for nitrogen.

Diffusivity

Diffusivity is assumed to be the binary diffusion coefficient between the given species and the background species. Diffusivity can be specified in one of three ways: it can be defined as a constant using DIFFUSIVITY (m²/s), it can be defined as a temperature vs. diffusivity ramp using RAMP_D, or it can be computed by FDS using MW and the Lennard-Jones potential parameters σ (SIGMALJ) and ϵ/k (EPSILONKLJ). If no inputs are specified, FDS will compute the diffusivity using the MW and the Lennard-Jones parameters for nitrogen.

Enthalpy

The enthalpy of the gas mixture is given by the following formula:

$$h(T) = h(T_{\text{ref}}) + \int_{T_{\text{ref}}}^T c_p(T') dT' \quad (12.1)$$

where c_p is the SPECIFIC_HEAT (kJ/(kg · K)) with optional temperature dependence using RAMP_CP. The (optional) REFERENCE_TEMPERATURE, T_{ref} (°C), is the temperature that corresponds to the

REFERENCE_ENTHALPY, $h(T_{\text{ref}})$ (kJ/kg). The default value of the REFERENCE_TEMPERATURE is 25 °C. If SPECIFIC_HEAT is specified and the REFERENCE_ENTHALPY is not, the REFERENCE_ENTHALPY will be set to $h(T_{\text{ref}}) = c_p T_{\text{ref}}$.

If no inputs for enthalpy are provided, then the specific heat of the gas will be calculated from its molecular weight using the relation:

$$c_{p,\alpha} = \frac{\gamma}{\gamma - 1} \frac{R}{W_\alpha} \quad (12.2)$$

The ratio of specific heats, GAMMA, is 1.4 by default and can be changed on the MISC line. If you want all the gas specific heats to follow this relation, set CONSTANT_SPECIFIC_HEAT_RATIO=.TRUE. on the MISC line (note: this option also requires STRATIFICATION=.FALSE.). In this case the REFERENCE_ENTHALPY will be assumed to be 0 kJ/kg at a REFERENCE_TEMPERATURE of 0 K.

The reference enthalpy can also be determined by defining the ENTHALPY_OF_FORMATION on the SPEC line with a reference temperature for all species given by H_F_REFERENCE_TEMPERATURE on the MISC line (default is 25 °C). Note that ENTHALPY_OF_FORMATION will override any value given for REFERENCE_ENTHALPY.

Viscosity

The dynamic viscosity of the gas species can be specified in one of three ways: it can be defined as a constant using VISCOSITY, it can be defined as a temperature vs. viscosity ramp using RAMP_MU, or it can be computed using the Lennard-Jones potential parameters σ (SIGMALJ) and ϵ/k (EPSILONKLJ). If no viscosity inputs are provided, FDS will use the Lennard-Jones values for nitrogen.

Radiative Properties

Species that can absorb and emit thermal radiation are defined via the parameter RADCAL_ID on the SPEC line. Some of the predefined species have this parameter already defined as shown in Table 12.1. There are, however, many other species which are absorbing. For absorbing species not listed in Table 12.1, RADCAL_ID can be used to identify a RadCal [?] species to serve as a surrogate. For example:

```
&SPEC ID='ETHANOL', RADCAL_ID='METHANOL' /
```

would use the RadCal absorptivities for METHANOL when computing the absorptivity of ETHANOL. For absorbing species not present in RadCal, it is recommended to choose a RadCal surrogate with similar molecular functional groups and molecular mass. The infrared spectrum is greatly affected by the species molecular functional groups.

Species molecular mass also affects the spectrum: a heavier species of a given molecular functional group tends to absorb and emit more infrared radiation than a lighter species of the same functional group. For simple chemistry, if the fuel is not present in Table 12.1 and no FUEL_RADCAL_ID is provided on the REAC line, then the absorption properties of methane will be used.

Gibbs Energy

If a reverse chemical reaction is specified using REVERSE=.TRUE. on a REAC input, FDS can use the forward reaction kinetics along with the equilibrium values of the reaction to determine the reverse kinetics. The equilibrium is determined by minimizing the Gibbs energy. The temperature vs. Gibbs energy (kJ/mol) for a species can be specified with RAMP_G_F.

Liquids

If the species listed in Table 12.1 includes liquid properties, it can be applied to liquid droplets, in which case the following relationship should hold:

$$h_{\text{gas}}(T_{\text{boil}}) = h_{\text{liquid}}(T_{\text{boil}}) + h_v \quad (12.3)$$

More detail is included in Chapter 15.

12.1.3 Air

There are two predefined species for air in FDS. The first predefined species is the default background species of AIR. This is a lumped species consisting of oxygen, nitrogen, carbon dioxide, and water vapor whose mass fractions are controlled by the Y_CO2_INFTY, Y_O2_INFTY, and HUMIDITY inputs. This lumped species is automatically defined by FDS if no other SPEC input is defined as the BACKGROUND. Note that ID='AIR' cannot be used on a SPEC input unless that input or some other SPEC input is defined as the BACKGROUND. The second predefined species is the primitive species of LJ AIR. This is an effective gas species whose molecular weight and enthalpy are defined based on the Y_CO2_INFTY and Y_O2_INFTY inputs, and whose other thermophysical properties use the Lennard-Jones parameters for air. For simulations without combustion, using LJ AIR as the BACKGROUND species will slightly reduce the computational cost.

Specifying a Chemical Formula

If you want FDS to compute the molecular weight of the gas species, you can input a FORMULA rather than the molecular weight, MW. This will also be used as the label for the gas species by Smokeview. FORMULA is a character string consisting of elements followed by their atom count. Subgroups bracketed by parentheses can also be given. The element name is given by its standard, case-sensitive, IUPAC¹ abbreviation (e.g., C for carbon, He for helium). The following are all equivalent:

```
&SPEC ID='ETHYLENE GLYCOL', FORMULA='C2H6O2' /
&SPEC ID='ETHYLENE GLYCOL', FORMULA='OHC2H4OH' /
&SPEC ID='ETHYLENE GLYCOL', FORMULA='C2H4(OH)2' /
```

Two Gas Species with the Same Properties

In general only one species for a given ID can be defined; however, you may wish to model multiple inlet streams of a species and be able to identify how well the streams are mixing. This can be done by defining a new species with a single component. For example, the lines:

```
&SPEC ID='CARBON DIOXIDE', LUMPED_COMPONENT_ONLY=.TRUE. /
&SPEC ID='CO2 1', SPEC_ID='CARBON DIOXIDE' /
&SPEC ID='CO2 2', SPEC_ID='CARBON DIOXIDE' /
&DEVC XYZ=..., QUANTITY='MASS FRACTION', SPEC_ID='CO2 1', ID='Device 1' /
&DEVC XYZ=..., QUANTITY='MASS FRACTION', SPEC_ID='CO2 2', ID='Device 2' /
```

define two duplicate species, both of which are CARBON DIOXIDE. Both will use the built in property data for CO₂ (note specifying one or more properties for a duplicate species will override the default properties). The ID for each duplicate species can then be used in the remainder of the input file. In this example, the LUMPED_COMPONENT_ONLY was given so that FDS only tracks CO2 1 and CO2 2 and not CARBON

¹International Union of Pure and Applied Chemistry

DIOXIDE. Note that the ID you provide for a duplicate species cannot match the ID of any other primitive or lumped SPEC input. Also note that this feature can only be used to duplicate a predefined species (i.e. a species listed in Table 12.1).

In the above example, the two DEVC lines refer to the IDs of the duplicate species. If instead the primitive species ID was used, then the output would sum the mass fractions over all species containing that primitive species. For example, if the output quantities in the example above are changed as shown below, then Device 1 would just output the mass fraction of CO2 1 and Device 2 would output the sum of both.

```
&DEVC XYZ=..., QUANTITY='MASS FRACTION', SPEC_ID='CO2 1', ID='Device 1'/
&DEVC XYZ=..., QUANTITY='MASS FRACTION', SPEC_ID='CARBON DIOXIDE', ID='Device 2'/
```

Another application of this would be if you wanted to track the water that evaporated from sprinklers, separately from the water that resulted from combustion. The following inputs would allow you to do that:

```
&REAC FUEL='PROPANE', ... /
&SPEC ID='WATER VAPOR SPK', SPEC_ID='WATER VAPOR' /
&PART ID='Sprinkler Droplets', SPEC_ID='WATER VAPOR SPK'/
&DEVC XYZ=..., QUANTITY='MASS FRACTION', SPEC_ID='WATER VAPOR SPK', ID='Spr H2O'/
&DEVC XYZ=..., QUANTITY='MASS FRACTION', SPEC_ID='WATER VAPOR', ID='All H2O'/
```

The gas species called WATER VAPOR SPK has the same properties as WATER VAPOR. The first device records only water that results from droplet evaporation, and the second device records water that originates from both sprinklers and combustion.

By default, duplicate species are considered to be a lumped species and not a primitive species. That is, by default, a duplicate species cannot be used in a lumped species definition. Setting PRIMITIVE=.TRUE. will have FDS treat the duplicate species as a primitive species and allow it to be used in a lumped species definition. Note that when this is done, one can no longer aggregate SPEC_ID based outputs over the original and the duplicate species as the aggregation is done on the basis of the primitive species names. This is demonstrated in the example below. The species O2 and O3 are duplicate species of OXYGEN. O3 is defined as a primitive species and O2 is considered only a lumped species. The initial DEVC outputs in order would be 0.3 (0.1 for OXYGEN plus 0.2 for O2), 0.2 (the O2 initial value), and 0.3 (the O3 initial value). Note the OXYGEN output is not 0.6 since the O3 is defined as a new primitive species.

```
&SPEC ID='NITROGEN', BACKGROUND=.TRUE./
&SPEC ID='OXYGEN', MASS_FRACTION_0=0.1/
&SPEC ID='O2', SPEC_ID='OXYGEN', MASS_FRACTION_0=0.2/
&SPEC ID='O3', SPEC_ID='OXYGEN', MASS_FRACTION_0=0.3, PRIMITIVE=.TRUE./

&DEVC XYZ=0.5,0.5,0.5, QUANTITY='MASS FRACTION', SPEC_ID='OXYGEN' /
&DEVC XYZ=0.5,0.5,0.5, QUANTITY='MASS FRACTION', SPEC_ID='O2' /
&DEVC XYZ=0.5,0.5,0.5, QUANTITY='MASS FRACTION', SPEC_ID='O3' /
```

12.2 Specifying Lumped Species (Mixtures of Primitive Species)

The SPEC namelist group also allows you to define species mixtures. The purpose of a species mixture is to reduce the number of species transport equations that are explicitly solved. For example, consider air. Air is composed of nitrogen, oxygen, water vapor, and carbon dioxide. If we define the four component species of air, we will have four total species. One of these, say nitrogen, can be set to be the background species, leaving three transport equations to solve. Alternatively, we can define a “lumped species” that represents the air mixture and save on CPU time because the three transport equations are no longer needed and the

“lumped” air becomes the background species.

The following inputs define equivalent initial mixtures. But in the first example three species are explicitly tracked and in the second example only a background mixture is specified. Note that this example represents the background species created by FDS when no background is explicitly defined in the input file. It is also noted that any implicitly defined species (such as the nitrogen, oxygen, water vapor, and carbon dioxide for the air background species or the species in the lumped product species for simple chemistry, see 13.1.1), can be referenced by any of the outputs such as `DEVC` or `SLCF`.

Example 1: All primitive species

```
&SPEC ID='NITROGEN', BACKGROUND=.TRUE. / Note: The background must be defined first.
&SPEC ID='OXYGEN',      MASS_FRACTION_0=0.23054 /
&SPEC ID='WATER VAPOR',  MASS_FRACTION_0=0.00626 /
&SPEC ID='CARBON DIOXIDE', MASS_FRACTION_0=0.00046 /
```

Example 2: Defining a background species

```
&SPEC ID='NITROGEN',      LUMPED_COMPONENT_ONLY=.TRUE. /
&SPEC ID='OXYGEN',        LUMPED_COMPONENT_ONLY=.TRUE. /
&SPEC ID='WATER VAPOR',    LUMPED_COMPONENT_ONLY=.TRUE. /
&SPEC ID='CARBON DIOXIDE', LUMPED_COMPONENT_ONLY=.TRUE. /

&SPEC ID='AIR', BACKGROUND=.TRUE.,
  SPEC_ID(1)='NITROGEN',    MASS_FRACTION(1)=0.76274,
  SPEC_ID(2)='OXYGEN',      MASS_FRACTION(2)=0.23054,
  SPEC_ID(3)='WATER VAPOR',  MASS_FRACTION(3)=0.00626,
  SPEC_ID(4)='CARBON DIOXIDE', MASS_FRACTION(4)=0.00046 /
```

The logical parameter `LUMPED_COMPONENT_ONLY` indicates that the species is only present as part of a lumped species. When `.TRUE.`, FDS will not allocate space to track that species individually. The parameters to define a lumped species are:

`BACKGROUND` Denotes that this lumped species is to be used as the background species.

`ID` Character string identifying the name of the species. You must provide this. This cannot be the same as an `ID` of another `SPEC` input.

`SPEC_ID` Character array containing the names of the primitive species that make up the lumped species.

`MASS_FRACTION` The mass fractions of the components of the lumped species in the order listed by `SPEC_ID`. FDS will normalize the values to 1. Alternatively, `VOLUME_FRACTION` can be specified. Do not use both on an `SPEC` line.

`MASS_FRACTION_0` The initial mass fractions of lumped species.

When defining a lumped species, either `MASS_FRACTION` or `VOLUME_FRACTION` must be used to define the component species. The addition of lumped species to FDS has changed the meaning of `SPEC_ID` on some FDS inputs. For `INIT`, `MATL`, `PART`, and `SURF`, `SPEC_ID` refers to either a tracked primitive species or a lumped species. For outputs and devices, `DEVC`, that require a `SPEC_ID`, the `SPEC_ID` input can refer to either a tracked primitive species, a lumped species, or a lumped species component that is not tracked. For `REAC` see the discussion on specifying reactions in Chapter 13.

12.2.1 Combining Lumped and Primitive Species

There are cases where you may wish to have a single primitive species be both part of a lumped species and also a separately tracked species. For example, when using simple chemistry, Section 13.1.1, FDS will include water vapor in the product species and in the air background species. If you also wish to have sprinklers in the simulation, then you will need to track water vapor from the sprinklers separately from that in the air or products. This is simply done by adding the line:

```
&SPEC ID='WATER VAPOR' /
```

This will override the implicitly created water vapor species defined with `LUMPED_COMPONENT_ONLY=.TRUE.` and cause FDS to track water vapor as a separate species. Note that in this case if you requested an output for the mass fraction of `WATER VAPOR` you would get the water vapor in the air and product lumped species as well as that which evaporated from sprinkler droplets. If you wanted in this case (where water vapor is implicitly defined), to be able to track the water vapor from sprinklers separately you could follow the example in Section 12.1.3 and define:

```
&SPEC ID='SPRINKLER WATER VAPOR', SPEC_ID='WATER VAPOR' /
```

Using the species `SPRINKLER WATER VAPOR` for the sprinklers would allow you to track sprinkler generated water vapor separately.

Chapter 13

Combustion

A common source of confusion in FDS is the distinction between gas phase *combustion* and solid phase *pyrolysis*. The former refers to the reaction of fuel vapor and oxygen; the latter the generation of fuel vapor at a solid or liquid surface. Whereas there can be many types of combustibles in an FDS fire simulation, in the simple chemistry, mixing-controlled combustion model there can only be one gaseous fuel. The reason is cost. It is expensive to solve transport equations for multiple gaseous fuels. Consequently, the burning rates of solids and liquids are automatically adjusted by FDS to account for the difference in the heats of combustion of the various combustibles. In effect, you specify a single gas phase reaction as a surrogate for all potential fuels.

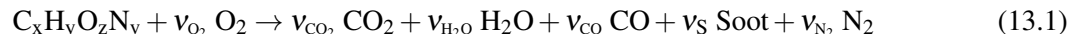
Combustion can be modeled in two ways. By default, the reaction of fuel and oxygen is infinitely fast and controlled only by mixing, hence the label *mixing-controlled*. The alternative is that the reaction is *finite-rate*. The latter approach usually requires very fine grid resolution that is not practical for large-scale fire applications. This chapter describes both methods, with an emphasis on the more commonly used mixing-controlled model. The `REAC` namelist group contains the parameters for both modes of combustion.

13.1 Single-Step, Mixing-Controlled Combustion

This approach to combustion, referred to below as the “simple chemistry” combustion model, considers a single fuel species that is composed primarily of C, H, O, and N that reacts with oxygen in one mixing-controlled step to form H_2O , CO_2 , soot, and CO. Information about the reaction is provided on the `REAC` line. Starting with FDS 6, you *must* specify a `REAC` line to model a fire. You are responsible for defining the basic fuel chemistry and the post-combustion yields of CO and soot. The default values are 0.

13.1.1 Simple Chemistry Parameters

For the simple chemistry model, each reaction is assumed to be of the form:



You need only specify the chemical formula of the fuel along with the yields of CO and soot, and the volume fraction of hydrogen in the soot, X_{H} . FDS will use that information and calculate the stoichiometric coefficients automatically as follows:

$$\begin{aligned} v_{\text{O}_2} &= v_{\text{CO}_2} + \frac{v_{\text{CO}}}{2} + \frac{v_{\text{H}_2\text{O}}}{2} - \frac{z}{2} \\ v_{\text{CO}_2} &= x - v_{\text{CO}} - (1 - X_{\text{H}}) v_{\text{S}} \end{aligned}$$

$$\begin{aligned}
v_{\text{H}_2\text{O}} &= \frac{y}{2} - \frac{X_{\text{H}}}{2} v_{\text{S}} \\
v_{\text{CO}} &= \frac{W_{\text{F}}}{W_{\text{CO}}} y_{\text{CO}} \\
v_{\text{S}} &= \frac{W_{\text{F}}}{W_{\text{S}}} y_{\text{S}} \\
v_{\text{N}_2} &= \frac{v}{2} \\
W_{\text{S}} &= X_{\text{H}} W_{\text{H}} + (1 - X_{\text{H}}) W_{\text{C}}
\end{aligned}$$

The following parameters may be prescribed on the `REAC` line when using the simple chemistry model. Note that the various `YIELDS` are for well-ventilated, post-flame conditions. There are options to predict various species yields in under-ventilated fire scenarios, but these special models still require the post-flame yields for CO, soot and any other species listed below.

FUEL (Required) A character string that identifies fuel species for the reaction. When using simple chemistry, specifying `FUEL` will cause FDS to use the built-in thermophysical properties for that species when computing quantities such as specific heat or viscosity. Table 12.1 provides a listing of the available species. If the `FUEL` is in the table, then FDS will use the built-in formula to obtain the values of C, H, O, and N. If not listed in Table 12.1, FDS uses the gas thermophysical properties of `ETHYLENE` along with the molecular weight given by the `FORMULA` or the values of C, H, O, and N. Either way, FDS will implicitly create a `SPEC` input for `FUEL`. This allows `FUEL` to be used as a `SPEC_ID` input elsewhere (for example as an initial condition or an output quantity). If you define `FUEL` yourself as a `SPEC`, any properties you specify will override the default values.

FORMULA A character string that identifies the chemical formula of the fuel species for the reaction. This input only has meaning when simple chemistry is being used and the formula can only contain C, H, O, or N. Specifying a formula means the individual inputs of C, H, O, and N do not need to be specified. See 12.1.2 for a description on how to input a `FORMULA`.

ID A character string that identifies the reaction. Normally, this label is not used by FDS, but it is useful to label the `REAC` line if more than one reactions are specified.

C, H, O, N The fuel chemical formula. All numbers are positive. One of either C or H must be specified. This input is not needed if `FORMULA` is specified or if the `FUEL` is in Table 12.1.

CO_YIELD The fraction of fuel mass converted into carbon monoxide, y_{CO} . Note that this parameter is only appropriate when the simple chemistry model is applied. (Default 0.)

SOOT_YIELD The fraction of fuel mass converted into smoke particulate, y_{S} . Note that this parameter is only appropriate when the simple chemistry model is applied. (Default 0.)

SOOT_H_FRACTION The fraction of the atoms in the soot that are hydrogen. The default value is 0.1, equivalent to the input `FORMULA='C0.9H0.1'` (Section 12.1.3). Note that this parameter is only appropriate when the simple chemistry model is applied.

FUEL_RADCAL_ID RadCal species to be used for the fuel. The default is the default RadCal species for the fuel species or `'METHANE'` if there is no species default. See Section 14.2.1 for details.

The ambient mass fractions for the constituents of air are specified on `MISC` using the inputs:

Y_O2_INFINITY Ambient mass fraction of oxygen (Default 0.232378)

Y_CO2_INFTY Ambient mass fraction of carbon dioxide (Default 0.000595)

HUMIDITY Relative humidity of the background air species, in units of %. (Default 40 %).

A few sample REAC lines are given here.

```
&REAC FUEL = 'METHANE' /
```

In this case, there is no need for a FORMULA or atom count because the FUEL is listed in Table 12.1. It is assumed that the soot and CO yields are zero. FDS will compute the yields of product species and the heat of combustion based upon predefined values.

```
&REAC FUEL          = 'PROPANE'  
      SOOT_YIELD     = 0.01  
      CO_YIELD       = 0.02  
      HEAT_OF_COMBUSTION = 46460. /
```

In this case, the fuel species is again predefined. However, here the heat of combustion is specified explicitly rather than calculated. Additionally, minor species yields have been specified with the soot yield specified as 0.01 and the CO yield specified as 0.02. See Section 13.1.2 for more details on the heat of combustion.

```
&REAC FUEL          = 'MY FUEL'  
      FORMULA        = 'C3H8O3N4'  
      HEAT_OF_COMBUSTION = 46124. /
```

In this case, the fuel is not predefined. Therefore, either the FORMULA or the atom counts must be defined. This input defined the FORMULA. In this case, the heat of combustion is known and specified; however, if it weren't FDS would compute it using EPUMO2 and the fuel chemistry. Note that simple chemistry can also be used for cases where the fuel is a lumped species so long as the defining primitive species contain only C, H, N, and O atoms. An example can be found in Section 13.2.1.

When simple chemistry is being used, FDS will automatically create three lumped species: AIR, FUEL, and PRODUCTS. The actual name of the fuel species will be the name given on the REAC line (for example MY FUEL in the last sample above). FDS creates these lumped species in the same manner as you would in an input file. FDS first defines the primitive species and then defines the lumped species. In essence FDS internally creates input lines like those shown in Example 2 of Section 12.2. This means when doing simple chemistry, that even though you did not explicitly define oxygen in the input file, you can request an output for oxygen since it was implicitly defined by FDS.

13.1.2 Heat of Combustion

The energy release per unit volume (kJ/m^3) from a gas phase chemical reaction (or system of reactions) is found by taking the sum of the net change in mass for each species in a given time step multiplied by the respective species' enthalpy of formation (kJ/kg). In this formulation, the enthalpy of formation for all participating species needs to be specified. If a reaction (simple chemistry or user-defined) contains only species defined in Table 12.1, then all of the enthalpies of formation are known. These values can be found in the FDS source code in data.f90. For reactions with species that are not included in Table 12.1, there are several options to ensure that all of the enthalpies of formation are specified.

Option 1: Specify Enthalpy of Formation

You can specify unknown enthalpies on the SPEC line in units of kJ/mol :

```
&SPEC ID = 'GLUCOSE', FORMULA = 'C6H12O6', ENTHALPY_OF_FORMATION=-1.297E3 /
```

Option 2: Specify Heat of Combustion

For a given reaction, if the only species missing an enthalpy of formation is the fuel, the missing value can be found if the heat of combustion is specified on REAC. Section 13.2.1 provides an example for which the fuel, polyvinyl chloride, has an unspecified enthalpy of formation but a specified heat of combustion on the REAC line.

Option 3: Use of EPUMO2 (Simple Chemistry)

If the enthalpy of formation of the fuel and heat of combustion are not specified, for simple chemistry cases only, the heat of combustion is assumed to be

$$\Delta h \approx \frac{v_{O_2} W_{O_2}}{v_F W_F} \text{ EPUMO2} \quad \text{kJ/kg} \quad (13.2)$$

The quantity EPUMO2 (kJ/kg) is the amount of energy released per unit mass of oxygen consumed. Its default is 13,100 kJ/kg. Typically, a chemical reaction is balanced by setting the stoichiometric coefficient of the fuel v_F to 1. In FDS, the stoichiometric coefficients of the chemical reaction are normalized by the stoichiometric coefficient of the fuel, effectively setting v_F to 1. Note that if both EPUMO2 and HEAT_OF_COMBUSTION are specified that FDS will ignore the value for EPUMO2. From the HEAT_OF_COMBUSTION, FDS solves for the enthalpy of formation of the fuel.

If heats of reaction have been specified on the MATL lines and the heats of combustion of the materials differ from that specified by the governing gas phase reaction, then add a HEAT_OF_COMBUSTION (kJ/kg) to the MATL line. With the simple chemistry combustion model, it is assumed that there is only one fuel. However, in a realistic fire scenario, there may be many fuel gases generated by the various burning objects in the building. Specify the stoichiometry of the predominant reaction via the REAC namelist group. If the stoichiometry of the burning material differs from the global reaction, the HEAT_OF_COMBUSTION is used to ensure that an equivalent amount of fuel is injected into the flow domain from the burning object.

The heat of combustion can be determined in a couple of ways. One approach is to take the difference in the heats of formation for the products (assuming complete combustion) and the reactants. This is typically how values are tabulated for pure fuels (e.g., one species) in handbooks. This ideal heat of combustion does not account for the SOOT_YIELD or CO_YIELD that occurs in a real fire. Carbon and hydrogen that go to soot and CO rather than CO₂ and H₂O result in a lower effective heat of combustion. Setting IDEAL=.TRUE. will reduce the HEAT_OF_COMBUSTION based upon the inputs for SOOT_YIELD and CO_YIELD. If EPUMO2 is specified instead of HEAT_OF_COMBUSTION, then the EPUMO2 will not be changed.

The second approach to determining the heat of combustion is to burn a known mass of the material in a calorimeter and divide the heat release rate by the mass loss rate (known as the effective heat of combustion). In this approach, represented by IDEAL=.FALSE., the measured value of the heat release rate includes the effects of any CO or soot that is produced and no adjustment is needed. The default value is IDEAL=.FALSE.

Note: If you specify a heat of combustion on the REAC line, FDS will calculate the enthalpy of formation of the fuel such that the user-specified heat of combustion is maintained. This is important to recognize for cases where a heat of combustion is measured experimentally or if you want to model impurities in the fuel that would not be realized using the standard heat of formation of the fuel.

If the reaction is under defined, FDS will return an error at the start of the calculation.

13.1.3 Special Topic: Turbulent Combustion

Unless you are performing a Direct Numerical Simulation (DNS), the reaction rate of fuel and oxygen is not based on the diffusion of fuel and oxygen at a well-resolved flame sheet. Instead, semi-empirical rules are invoked by FDS to determine the rate of mixing of fuel and oxygen within a given mesh cell at a given time step. Each computational cell can be thought of as a batch reactor where only the mixed composition can react. The variable, $\zeta(t)$, denotes the unmixed fraction, ranging from zero to one and governed by the equation:

$$\frac{d\zeta}{dt} = \frac{-\zeta}{\tau_{\text{mix}}} \quad (13.3)$$

Here, τ_{mix} is the mixing time scale. The change in mass of a species is found from the combination of mixing and the production/destruction rate found from the chemical reaction. If a cell is initially unmixed, $\zeta_0 = 1$ by default, then combustion is considered non-premixed within the grid cell. In this case, the fuel and air are considered completely separate at the start of the time step and must mix together before they burn. This means if the mixing is slow enough, that unburned fuel may exist at the end of the time step even if sufficient oxygen is present in the grid cell to burn all the fuel. If the cell is initially fully mixed, $\zeta_0 = 0$, then the combustion is considered premixed (e.g., equivalent to infinitely fast mixing). You can set the amount of mixing in each cell at the beginning of every time step using the parameter `INITIAL_UNMIXED_FRACTION` on the `MISC` line.

Note that the `MIXTURE_FRACTION` output quantity applies the Burke-Schumann solution (see [?]) to map species composition to mixture fraction (i.e., mixture fraction is not a primitive flow variable but rather is calculated from the local fuel and product mass fractions). This requires simple chemistry with a reaction of the type $F + A \rightarrow P$ with infinitely fast mixing and infinitely fast chemistry. Therefore, you must set `INITIAL_UNMIXED_FRACTION=0` to output `MIXTURE_FRACTION`.

The Technical Reference Guide [?] contains more detailed information about the turbulent combustion model.

13.1.4 Special Topic: Flame Extinction

Modeling suppression of a fire due to the introduction of a suppression agent like CO_2 or water mist, or due to the exhaustion of oxygen within a compartment is challenging because the relevant physical mechanisms occur at length scales smaller than a single mesh cell. Flames are extinguished due to lowered temperatures and dilution of the oxygen supply. A simple suppression algorithm has been implemented in FDS that attempts to gauge whether or not combustion is viable based on the local energy release. There must be sufficient energy released to raise the cell temperature above the critical flame temperature for combustion to occur. The Technical Reference Guide [?] contains more details about how the mechanism works.

The only parameter you can control is the `CRITICAL_FLAME_TEMPERATURE` (CFT) (the adiabatic flame temperature at the lower flammability limit) set on the `REAC` line. The default value is 1327 °C. This value represents combustion of typical hydrocarbon fuels. Note that if you are using an effective heat of combustion or other types of fuels, the CFT should be calculated (see below) to reflect your reaction system. To eliminate any gas phase suppression, set `SUPPRESSION=.FALSE.` on the `MISC` line.

If the mixing-controlled combustion model is used and flame extinction occurs, the unburned fuel gas can re-ignite if it mixes with sufficient oxygen somewhere else in the domain. To prevent this from happening, you can set the `AUTO_IGNITION_TEMPERATURE` (AIT) on the `REAC` line, in °C, below which combustion will not occur. Note that if this parameter is used, then some form of heat/ignition source must be present in order for combustion to begin.

You may also specify `AUTO_IGNITION_TEMPERATURE` on an `INIT` region. This feature is intended to be used as a pilot zone. For example, you might use something like this for a piloted methane flame:

```
&REAC FUEL='METHANE', AUTO_IGNITION_TEMPERATURE=627. /
&INIT XB=..., AUTO_IGNITION_TEMPERATURE=-273. /
```

where the XB specifies a region at the base of the burner where AIT is set to achieve “mixed is burnt”.

Finding or Calculating Extinction Parameters For most fuels of interest, the AIT and CFT may be found in [?]. The AITs are tabulated for a range of fuels in Table 2-7.1 of the SFPE Handbook [?], “Summary of Limits of Flammability, Lower Temperature Limits (TL), and Minimum Autoignition Temperatures (AIT) of Individual Gases and Vapors in Air at Atmospheric Pressure.” Similarly, Table 2-7.3, “Thermodynamic Equilibrium Properties at Extinction,” provides CFT (listed as T(LFL), for temperature at lower flammability limit) for several common fuels.

If the lower flammability limit (LFL) of the fuel mixture is known but the CFT is not, then Beyler’s Eq. (4) may be used to estimate the CFT. The formula is repeated here for convenience:

$$T_{\text{CFT}} = T_0 + \left(\frac{\text{LFL}}{100} \right) \frac{\Delta H_c}{nC_p} \quad (13.4)$$

where

ΔH_c = Heat of combustion of the fuel (J/kg)

LFL/100 = Mole fraction of fuel

n = Number of moles of products of combustion per mole of fuel/air mixture

C_p = Heat capacity of products of combustion (J/(kg · K))

T_0 = Initial temperature of the fuel/air mixture (ambient conditions) (K)

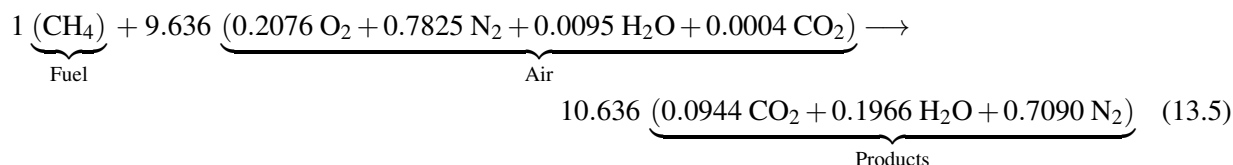
T_{CFT} = Adiabatic flame temperature of a lower flammability limit mixture (K)

13.2 Complex Stoichiometry

The “simple chemistry” parameters described above can only be used when there is a single mixing-controlled reaction and the fuel molecule contains only C, O, H, and N. For any other situation, you must specify the reaction stoichiometry in greater detail. This means that you must explicitly specify the gas species, or species mixtures, along with the stoichiometry of the reaction. The easiest way to explain this is by way of example. Consider a single reaction involving methane. When you specify the REAC line to be:

```
&REAC FUEL='METHANE' /
```

FDS assumes the following reaction:



By default, there are trace amounts of carbon dioxide and water vapor in the air, which, like the nitrogen, is carried along in the reaction. This is important, because the more complicated way to specify a single step reaction of methane is as follows:

```
&SPEC ID='NITROGEN',          LUMPED_COMPONENT_ONLY=.TRUE. /
&SPEC ID='OXYGEN',           LUMPED_COMPONENT_ONLY=.TRUE. /
&SPEC ID='WATER VAPOR',      LUMPED_COMPONENT_ONLY=.TRUE. /
&SPEC ID='CARBON DIOXIDE',   LUMPED_COMPONENT_ONLY=.TRUE. /
&SPEC ID='METHANE' /

&SPEC ID='AIR', BACKGROUND=.TRUE.,
  SPEC_ID(1)='OXYGEN',        VOLUME_FRACTION(1)=0.2076,
  SPEC_ID(2)='NITROGEN',      VOLUME_FRACTION(2)=0.7825,
  SPEC_ID(3)='WATER VAPOR',   VOLUME_FRACTION(3)=0.0095,
  SPEC_ID(4)='CARBON DIOXIDE', VOLUME_FRACTION(4)=0.0004 /

&SPEC ID='PRODUCTS',
  SPEC_ID(1)='CARBON DIOXIDE', VOLUME_FRACTION(1)=0.0944,
  SPEC_ID(2)='WATER VAPOR',   VOLUME_FRACTION(2)=0.1966,
  SPEC_ID(3)='NITROGEN',      VOLUME_FRACTION(3)=0.7090 /

&REAC FUEL='METHANE', SPEC_ID_NU='METHANE','AIR','PRODUCTS',
  NU=-1,-9.636,10.636, HEAT_OF_COMBUSTION=50000. /
```

The reaction stoichiometry is specified using the stoichiometric coefficients, $\text{NU}(N)$, corresponding to the tracked¹ species, $\text{SPEC_ID_NU}(N)$. There are several parameters on the REAC line that control the specification of the stoichiometry:

CHECK_ATOM_BALANCE If chemical formulas are provided for all species that participate in a reaction, then FDS will check the stoichiometry to ensure that atoms are conserved. Setting this flag to `.FALSE.` will bypass this check. (Default `.TRUE.`)

REAC_ATOM_ERROR Error tolerance in units of atoms for the reaction stoichiometry check. (Default 0.00001)

REAC_MASS_ERROR Relative error tolerance computed as (mass of products - mass of reactants)/(mass of products) for the reaction stoichiometry mass balance check. (Default 0.0001)

¹A “tracked” species is one for which `LUMPED_COMPONENT_ONLY` is `.FALSE.`

13.2.1 Complex Fuel Molecules

For complex fuel molecules that contain only C, H, N, and O the simple chemistry reaction parameters can still be applied. Consider natural gas, which is often a mixture of several component gases. To build the fuel mixture, the primitive (component) species need be defined. Each primitive species will get a LUMPED_COMPONENT_ONLY designation, which means that each of these species will not be explicitly tracked as they are components to a mixture. Note that these lumped components can only contain C, H, N, and O atoms. The lumped fuel, 'natural gas', can be created using the defined primitive components and subsequently the reaction can be defined using simple chemistry (Section 13.1.1):

```
&SPEC ID='METHANE',          LUMPED_COMPONENT_ONLY=.TRUE. /
&SPEC ID='ETHYLENE',         LUMPED_COMPONENT_ONLY=.TRUE. /
&SPEC ID='NITROGEN',         LUMPED_COMPONENT_ONLY=.TRUE. /
&SPEC ID='CARBON DIOXIDE', LUMPED_COMPONENT_ONLY=.TRUE. /

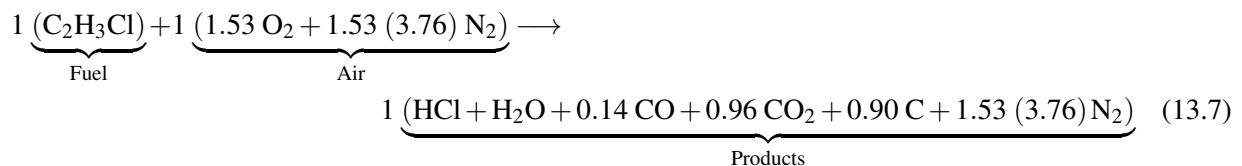
&SPEC ID='natural gas'
  SPEC_ID(1)='METHANE',      VOLUME_FRACTION(1)=92.2
  SPEC_ID(2)='ETHYLENE',     VOLUME_FRACTION(2)= 3.3
  SPEC_ID(3)='NITROGEN',     VOLUME_FRACTION(3)= 3.9
  SPEC_ID(4)='CARBON DIOXIDE',VOLUME_FRACTION(4)= 0.6/

&REAC FUEL='natural gas'
  SOOT_YIELD=0.01 /
```

Fires, however, often involve fuels that do not just consist of C, H, N, and O. For example, chlorine is commonly found in building and household materials, and because of its propensity to form the acid gas HCl, you may want to account for it in the basic reaction scheme. Suppose the predominant fuel in the fire is polyvinyl chloride (PVC). Regardless of its detailed polymeric structure, it can be regarded as C_2H_3Cl for the purpose of modeling. Assuming that all of the Cl in the fuel is converted into HCl, you can derive a single-step reaction mechanism using appropriate soot and CO yields for the specified fuel. In this example, the SFPE Handbook [?] is used to find soot and CO yields for PVC; 0.172 and 0.063, respectively. For a given species, α , its stoichiometric coefficient, ν_α , can be found from its yield, y_α , and its molecular weight, W_α , according to the formula:

$$\nu_\alpha = \frac{W_F}{W_\alpha} y_\alpha \quad (13.6)$$

Since it is assumed that all of the Cl is converted to HCL, the remainder of the stoichiometric coefficients come from an atom balance. An equation can now be written to include the appropriate numerical values for the stoichiometric coefficients.



The choice of fuel in this example, PVC, is not defined in Table 12.1, therefore its properties must be defined on a SPEC line. In this example, we use the species' chemical formula. The example will also use the lumped species formulation to minimize the number of scalar transport equations that need to be solved. Therefore, each species that does not have an explicit transport equation is a LUMPED_COMPONENT_ONLY.

```
&SPEC ID = 'PVC', FORMULA = 'C2H3Cl' /
&SPEC ID = 'OXYGEN',          LUMPED_COMPONENT_ONLY = .TRUE. /
```



```

&SPEC ID = 'NITROGEN',          LUMPED_COMPONENT_ONLY = .TRUE. /
&SPEC ID = 'HYDROGEN CHLORIDE', LUMPED_COMPONENT_ONLY = .TRUE. /
&SPEC ID = 'WATER VAPOR',       LUMPED_COMPONENT_ONLY = .TRUE. /
&SPEC ID = 'CARBON MONOXIDE',   LUMPED_COMPONENT_ONLY = .TRUE. /
&SPEC ID = 'CARBON DIOXIDE',    LUMPED_COMPONENT_ONLY = .TRUE. /
&SPEC ID = 'SOOT', FORMULA='C',  LUMPED_COMPONENT_ONLY = .TRUE. /

```

For the oxidizer and products, which are both composed of multiple primitive species, SPEC lines are needed to define the composition of the lumped species. You can define the SPEC using either the MASS_FRACTIONS of the component gases or the VOLUME_FRACTIONS. If Eq. (13.7) is properly balanced, you can directly use the stoichiometric coefficients of the primitive species to define the lumped species.

```

&SPEC ID='AIR', BACKGROUND=.TRUE.
SPEC_ID(1)='OXYGEN', VOLUME_FRACTION(1)=1.53,
SPEC_ID(2)='NITROGEN', VOLUME_FRACTION(2)=5.76 /

&SPEC ID='PRODUCTS',
SPEC_ID(1)='HYDROGEN CHLORIDE', VOLUME_FRACTION(1)=1.0,
SPEC_ID(2)='WATER VAPOR',       VOLUME_FRACTION(2)=1.0,
SPEC_ID(3)='CARBON MONOXIDE',   VOLUME_FRACTION(3)=0.14,
SPEC_ID(4)='CARBON DIOXIDE',    VOLUME_FRACTION(4)=0.96,
SPEC_ID(5)='SOOT',              VOLUME_FRACTION(5)=0.90,
SPEC_ID(6)='NITROGEN',          VOLUME_FRACTION(6)=5.76 /

```

To set the initial concentration of fuel, an INIT line is used:

```

&INIT MASS_FRACTION(1)=0.229, SPEC_ID(1)='PVC' /

```

Since this is not a simple chemistry problem, either the enthalpy of formation of PVC or the heat of combustion of the reaction should be specified. In this case, the heat of combustion for PVC is taken from the SFPE Handbook [?].

```

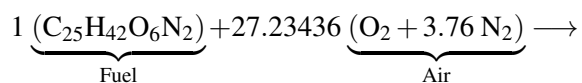
&REAC FUEL='PVC', HEAT_OF_COMBUSTION=16400, SPEC_ID_NU='PVC','AIR','PRODUCTS',
NU=-1,-1,1, FIXED_MIX_TIME=0.1 /

```

Note that the sign of NU corresponds to whether that species is consumed (-) or produced (+). Figure 13.1 displays the mass fractions of the product species for the sample case PVC_Combustion. A fixed turbulent mixing time of 0.1 s is used for this example only because the reactants are initially mixed within a chamber with no imposed flow. Normally, this parameter is not necessary.

13.2.2 Multiple Chemical Reactions

There may be times when your design/model fire is best described by more than one fuel or by more than a single-step chemical reaction. For this example consider two simultaneous, mixing-controlled reactions of polyurethane and wood. Both of these fuels are complex molecules not included in Table 12.1, so they must be defined on the SPEC line. Polyurethane is defined by the chemical formula $C_{25}H_{42}O_6N_2$ and wood is defined by $CH_{1.7}O_{0.74}N_{0.002}$. Consider a combustion reaction for polyurethane with a soot yield of $y_s = 0.131$ and a CO yield of $y_{CO} = 0.01$ [?] is:



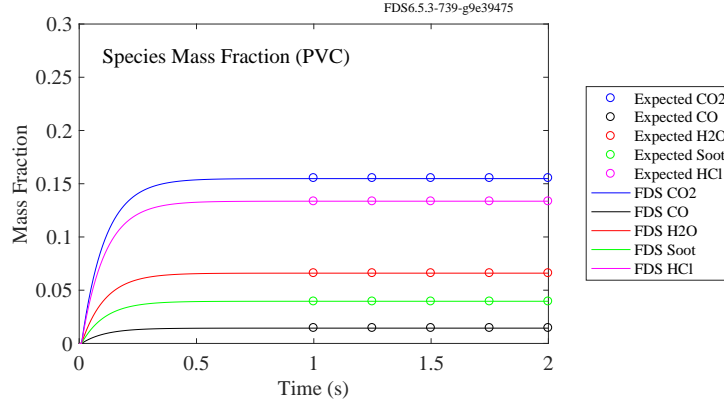
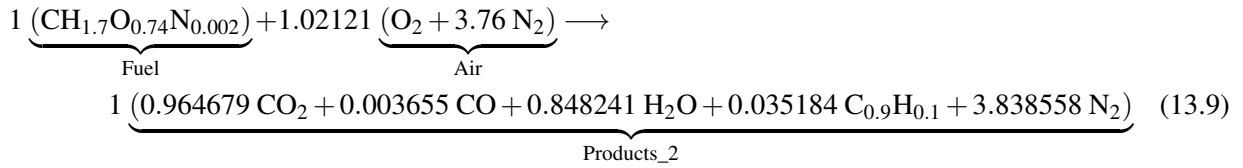


Figure 13.1: Product species mass fractions for model PVC example.

$$1 \underbrace{(19.79113 \text{ CO}_2 + 0.166587 \text{ CO} + 20.71987 \text{ H}_2\text{O} + 5.60253 \text{ C}_{0.9}\text{H}_{0.1} + 103.40121 \text{ N}_2)}_{\text{Products}_1} \quad (13.8)$$

and the reaction for wood with a soot yield of $y_s = 0.015$ and a CO yield of $y_{\text{CO}} = 0.004$ [?] is:



Similar to example in Section 13.2.1, we will use the lumped species approach to minimize the number of species that FDS needs to transport. Therefore, the species are defined in the following manner:

```
&SPEC ID = 'POLYURETHANE', FORMULA = 'C25H42O6N2' /
&SPEC ID = 'WOOD', FORMULA = 'CH1.7O0.74N0.002' /
&SPEC ID = 'OXYGEN', LUMPED_COMPONENT_ONLY = .TRUE. /
&SPEC ID = 'NITROGEN', LUMPED_COMPONENT_ONLY = .TRUE. /
&SPEC ID = 'WATER VAPOR', LUMPED_COMPONENT_ONLY = .TRUE. /
&SPEC ID = 'CARBON MONOXIDE', LUMPED_COMPONENT_ONLY = .TRUE. /
&SPEC ID = 'CARBON DIOXIDE', LUMPED_COMPONENT_ONLY = .TRUE. /
&SPEC ID = 'SOOT', LUMPED_COMPONENT_ONLY = .TRUE. /
```

Examination of Eq. (13.8) and Eq. (13.9) shows that, while Products_1 and Products_2 are composed of the same species, the species do not exist in the same proportion. As a result, we must construct two separate product lumped species.

```
&SPEC ID = 'AIR',
SPEC_ID(1) = 'OXYGEN', VOLUME_FRACTION(1)=1,
SPEC_ID(2) = 'NITROGEN', VOLUME_FRACTION(2)=3.76,
BACKGROUND=.TRUE. /

&SPEC ID = 'PRODUCTS_1',
SPEC_ID(1) = 'CARBON DIOXIDE', VOLUME_FRACTION(1) = 19.79113,
SPEC_ID(2) = 'CARBON MONOXIDE', VOLUME_FRACTION(2) = 0.166587,
SPEC_ID(3) = 'WATER VAPOR', VOLUME_FRACTION(3) = 20.71987,
SPEC_ID(4) = 'SOOT', VOLUME_FRACTION(4) = 5.60253,
SPEC_ID(5) = 'NITROGEN', VOLUME_FRACTION(5) = 103.40121 /
```

```
&SPEC ID = 'PRODUCTS_2',
    SPEC_ID(1) = 'CARBON DIOXIDE', VOLUME_FRACTION(1) = 0.964679,
    SPEC_ID(2) = 'CARBON MONOXIDE', VOLUME_FRACTION(2) = 0.003655,
    SPEC_ID(3) = 'WATER VAPOR', VOLUME_FRACTION(3) = 0.848241,
    SPEC_ID(4) = 'SOOT', VOLUME_FRACTION(4) = 0.035184,
    SPEC_ID(5) = 'NITROGEN', VOLUME_FRACTION(5) = 3.838558 /
```

Once we have constructed the lumped species, we can define the REAC lines.

```
&REAC ID = 'plastic',
    FUEL = 'POLYURETHANE',
    HEAT_OF_COMBUSTION=26200,
    SPEC_ID_NU = 'POLYURETHANE', 'AIR', 'PRODUCTS_1'
    NU=-1,-27.23436,1 /

&REAC ID = 'wood'
    FUEL = 'WOOD',
    HEAT_OF_COMBUSTION=16400,
    SPEC_ID_NU = 'WOOD', 'AIR', 'PRODUCTS_2'
    NU=-1,-1.02063,1 /
```

In both of these reactions, the `ENTHALPY_OF_FORMATION` of the chosen fuels is unknown to FDS, so the `HEAT_OF_COMBUSTION` is specified for each reaction. Typically, the heat release per unit area (`HRRPUA`) parameter is used to specify the fire size on the `SURF` line. The `HRRPUA` parameter cannot currently be used when there is more than one fuel, so the mass flux of fuel must be specified for each fuel. In this example, we want both fuels to flow out of the same burner and ramp up to a 1200 kW fire after 60 s. First, we create a 1 m² burner by defining a `VENT` line:

```
&VENT XB=4.0,5.0,4.0,5.0,0.0,0.0, SURF_ID='FIRE1', COLOR='RED' /
```

The `VENT` points to the `SURF_ID='FIRE1'` which we can define using both fuels. The mass flux values for each fuel are determined by the desired heat release, the proportion of the total heat release rate each fuel contributes, the burner area, and the heat of combustion of each fuel. In this case we want a 1200 kW fire where each fuel contributes 50% to the total heat release rate (600 kW).

$$\dot{m}''_{\text{poly}} = \frac{600 \text{ kW}}{1 \text{ m}^2} \frac{1}{26200 \text{ kJ/kg}} = 0.022901 \text{ kg}/(\text{m}^2 \cdot \text{s}) \quad (13.10)$$

$$\dot{m}''_{\text{wood}} = \frac{600 \text{ kW}}{1 \text{ m}^2} \frac{1}{16400 \text{ kJ/kg}} = 0.036585 \text{ kg}/(\text{m}^2 \cdot \text{s}) \quad (13.11)$$

```
&SURF ID='FIRE1', SPEC_ID(1)='POLYURETHANE', MASS_FLUX(1)=0.022901, RAMP_MF(1)='poly'
    SPEC_ID(2)='WOOD', MASS_FLUX(2)=0.036585, RAMP_MF(2)='wood' /
```

Note that the `SPEC_ID`, `MASS_FLUX`, and `RAMP_MF` correspond to one another for each fuel. It does not matter which fuel is (1), just that the numbering is consistent. We also want each fuel to follow a ramp such that fire starts at 0 kW at the initial time, reaches 1200 kW at 100 s, and remains at 1200 kW for the remainder of a 600 s simulation.

```
&RAMP ID='poly', T = 0, F = 0.0 /
&RAMP ID='poly', T = 50, F = 0.5 /
&RAMP ID='poly', T = 100, F = 1.0 /
&RAMP ID='poly', T = 600, F = 1.0 /
```

```

&RAMP ID='wood', T = 0 , F = 0.0 /
&RAMP ID='wood', T = 50 , F = 0.5 /
&RAMP ID='wood', T = 100, F = 1.0 /
&RAMP ID='wood', T = 600, F = 1.0 /

```

Here, T corresponds to the time and F is the fraction of the mass flux specified on the `SURF` line. Fig. 13.2 compares the resulting FDS heat release rate (HRR) to the expected HRR from the defined ramp. Note that in this simulation, there is 10 s averaging on the FDS output HRR (`&DUMP DT_HRR=10`) and the expected results account for this averaging.

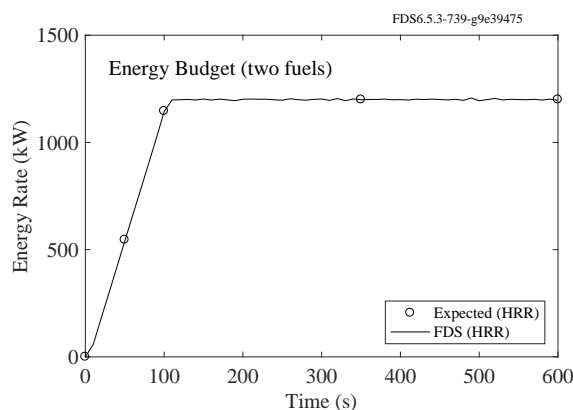


Figure 13.2: HRR for `energy_budget_adiabatic_two_fuels` test case.

Note that when using multiple chemical reactions, you must set `SUPPRESSION=.FALSE.` on the `MISC` line.

13.2.3 Special Topic: Using the `EQUATION` input parameter

When specifying a reaction scheme using all primitive variables (*i.e.*, *no lumped species*), a convenient shortcut for specifying the reaction is via the input parameter `EQUATION`, which allows the specification of the reaction in text form. The rules are:

- The species must be explicitly tracked.
- The species name is its chemical formula as defined on the `SPEC` line or by the species `ID`.
- The stoichiometry is given before each species and is separated by an asterisk. Real numbers are allowed but exponential notation is not (*i.e.*, `201.1` but not `2.011E2`).
- The reactants and products are separated by an equals sign.

For example, if the reaction defines the complete combustion of methane using primitive species, then the following would be equivalent:

```

&REAC FUEL='METHANE', EQUATION = 'METHANE+2*OXYGEN=CARBON DIOXIDE+2*WATER VAPOR' /
&REAC FUEL='METHANE', EQUATION = 'CH4+2*O2=CO2+2*H2O' /
&REAC FUEL='METHANE', EQUATION = 'METHANE+2*O2=CO2+2*H2O' /

```

13.3 Finite Rate Combustion

By default, FDS uses a mixing-controlled combustion model, meaning that the reaction rate is infinite and limited only by species concentrations. However, FDS can also employ finite-rate reactions using an Arrhenius model. It is recommended that finite-rate reactions be invoked only when FDS is running in DNS mode (`DNS=.TRUE.` on the `MISC` line). You can use the finite-rate reaction model in an LES calculation, but because the temperature in a large scale calculation is smeared out over a mesh cell, some of the reaction parameters may need to be modified to account for the lower cell-averaged temperatures.

Consider a single-step reaction mechanism for complete propane combustion:



In this case we explicitly define all primitive species:

```
&SPEC ID='NITROGEN', BACKGROUND=.TRUE./
&SPEC ID='PROPANE' /
&SPEC ID='OXYGEN' /
&SPEC ID='WATER VAPOR' /
&SPEC ID='CARBON DIOXIDE' /
```

The rate expression for the fuel, C_3H_8 , is

$$\frac{dC_{\text{C}_3\text{H}_8}}{dt} = -k \prod C_{\alpha}^{N_{S,\alpha}} \quad (13.13)$$

where the rate constant is defined as

$$k = A T^{N_T} e^{-E_a/RT} \quad (13.14)$$

A is the pre-exponential factor [$((\text{mol}/\text{cm}^3)^{1-n})/\text{s}$], where $n = \sum N_{S,\alpha}$ is the *order* of the reaction,

E_a is the activation energy [J/mol],

$N_{S,\alpha}$ is an array containing the concentration exponents (default 1),

N_T is the temperature exponent (default is 0, meaning no temperature dependence).

If we use Arrhenius parameters found from experiments or the literature (for this case Westbrook and Dryer [?]), the rate expression for the single-step propane reaction defined by Eq. (13.12) becomes:

$$\frac{dC_{\text{C}_3\text{H}_8}}{dt} = -8.6 \times 10^{11} T^0 e^{-125520/RT} C_{\text{C}_3\text{H}_8}^{0.1} C_{\text{O}_2}^{1.65} \quad (13.15)$$

and the resulting `REAC` line is:

```
&REAC ID = 'R1'
  FUEL = 'PROPANE'
  A = 8.6e11
  E = 125520
  SPEC_ID_NU = 'PROPANE', 'OXYGEN', 'CARBON DIOXIDE', 'WATER VAPOR'
  NU = -1, -5, 3, 4
  SPEC_ID_N_S = 'PROPANE', 'OXYGEN'
  N_S = 0.1, 1.65
```

The array `SPEC_ID_NU` contains the list of tracked species participating in the reaction as a product or reactant. Note that a primitive species with `LUMPED_COMPONENT_ONLY=.TRUE.` cannot be used in a reaction

since it is not tracked separately. The array `SPEC_ID_N_S` contains the list of species with the exponents, `N_S`, in Eq. (13.13). This array must contain only primitive species, i.e., no species mixtures. The array of stoichiometric coefficients, `NU`, can be taken directly from Eq. (13.12). If we extend Eq. (13.12) to include reactions from Westbrook and Dryer [?] that account for carbon monoxide production, we obtain:



In this case we will combine fast chemistry with finite-rate chemistry to create a mixed reaction mechanism. As before, we use primitive species rather than mixtures. A `REAC` line is needed for each reaction including the reverse reaction. The propane oxidation reaction is a fast chemistry while the reversible carbon monoxide reaction is finite-rate. The Arrhenius parameters are modified from Andersen et al. [?]:

```
&REAC ID = 'R1'
  FUEL = 'PROPANE'
  SPEC_ID_NU = 'PROPANE', 'OXYGEN', 'CARBON MONOXIDE', 'WATER VAPOR'
  NU = -1, -3.5, 3, 4
  CRITICAL_FLAME_TEMPERATURE = 730./

&REAC ID = 'R2'
  FUEL = 'CARBON MONOXIDE'
  A = 1.5e9
  E = 41840
  SPEC_ID_NU = 'CARBON MONOXIDE', 'OXYGEN', 'CARBON DIOXIDE'
  NU = -1, -0.5, 1
  SPEC_ID_N_S = 'OXYGEN', 'CARBON MONOXIDE', 'WATER VAPOR'
  N_S = 0.25, 1, 0.5/

&REAC ID = 'R3'
  FUEL = 'CARBON DIOXIDE'
  A = 6.16e13
  E = 328026
  SPEC_ID_NU = 'CARBON DIOXIDE', 'OXYGEN', 'CARBON MONOXIDE'
  NU = -1, 0.5, 1
  SPEC_ID_N_S = 'OXYGEN', 'CARBON DIOXIDE', 'WATER VAPOR'
  N_S = -0.25, 1, 0.5
  N_T = -0.97/
```

For `REAC ID='R2'`, the reaction rate depends on the water vapor concentration, as indicated by its assignment of a value of `N_S`. However, it does not explicitly participate in the reaction because it is not assigned a stoichiometric coefficient, `NU`. Reactions of this sort are often written in textbooks as



However, for the purposes of inputting into FDS, since H_2O is neither a product nor a reactant it would not be specified in the chemical reaction using either `EQUATION` or `NU`. It would instead be given a rate exponent using `N_S` to indicate that its presence is required.

Some reactions require the presence of any third body to stabilize the reaction rather than just a specific species. Reactions of this type are often written in textbooks as



This type of reaction can be done by adding `THIRD_BODY=.TRUE.` to the `REAC` line for the reaction. The species `M` should not otherwise be defined.

As discussed previously (Section 13.1.2), energy release is calculated using the net change of species in a time step along with the enthalpy of formation of each species. This approach makes specifying an endothermic reaction, such as the reversible CO₂ reaction (R3), no different than typical FDS exothermic combustion reactions.

13.4 Special Topic: Chemical Time Integration

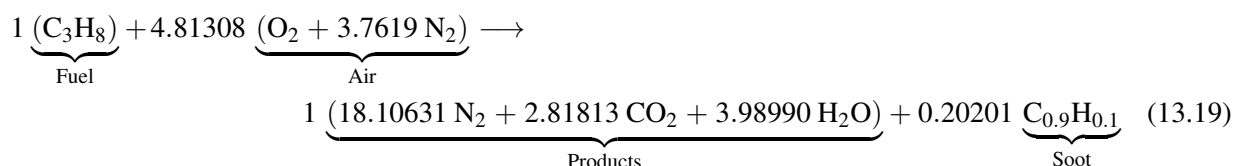
A diagnostic tool for the time integration of chemical reactions is the number of subiterations the ODE solver takes for a given FDS timestep. For infinitely fast chemistry the number of `CHEMICAL SUBITERATIONS` should always be 1. For finite rate chemical reactions the number can vary between 1 and the minimum between `MAX_CHEMISTRY_ITERATIONS` and the number needed to satisfy the error tolerance of the ODE solver. If the maximum value is reached, the user can increase the number of iterations or decrease reduce the error tolerance constraints of the integrator.

13.5 Special Topic: Aerosol Deposition

It is possible within FDS to model the deposition of smoke and aerosols onto solid surfaces. The aerosol deposition model is invoked by defining a species with the parameter `AEROSOL=.TRUE.` on the `SPEC` line along with the parameters `DENSITY_SOLID`, `CONDUCTIVITY_SOLID`, and `MEAN_DIAMETER`. By default, with `AEROSOL=.TRUE.`, FDS will compute all of the aerosol deposition mechanisms discussed in the Technical Reference Guide [?]. For diagnostic purposes, each deposition mechanism can be selectively disabled by using the logical parameters `GRAVITATIONAL_DEPOSITION`, `THERMOPHORETIC_DEPOSITION`, `TURBULENT_DEPOSITION`, and `GRAVITATIONAL_SETTLING` on the `MISC` line. `GRAVITATIONAL_SETTLING` controls the downward movement of aerosols in the gas phase, whereas `GRAVITATIONAL_DEPOSITION` controls the deposition of aerosols onto upward-facing surfaces due to gravity. The deposition velocity at the wall can be output using `QUANTITY='DEPOSITION VELOCITY'` for a wall cell with `DEVC` or `BNDF`.

13.5.1 Example Case: Soot Deposition from a Propane Flame

The `propane_flame_deposition` example shows how to define a reaction that invokes the aerosol deposition model in FDS. The fuel is propane with a specified soot yield of 0.05. Note that this is a fabricated soot yield that is used only for demonstration and verification purposes. The reaction is given by:



Note that the stoichiometric coefficient for soot ensures that the mass of soot produced is 0.05 times the mass of fuel consumed. This example uses the lumped species formulation to minimize the number of scalar transport equations that need to be solved. Note that for soot to deposit it must be explicitly tracked by defining `AEROSOL=.TRUE.` on the `SPEC` line.

```

&SPEC ID = 'PROPANE' /
&SPEC ID = 'OXYGEN',          LUMPED_COMPONENT_ONLY = .TRUE. /
&SPEC ID = 'NITROGEN',        LUMPED_COMPONENT_ONLY = .TRUE. /
&SPEC ID = 'WATER VAPOR',     LUMPED_COMPONENT_ONLY = .TRUE. /
&SPEC ID = 'CARBON DIOXIDE',  LUMPED_COMPONENT_ONLY = .TRUE. /
&SPEC ID = 'SOOT',            AEROSOL = .TRUE. /

```

If Eq. (13.19) is properly balanced, you can directly use the stoichiometric coefficients of the primitive species to define the lumped species:

```
&SPEC ID = 'AIR', SPEC_ID = 'NITROGEN', 'OXYGEN',  
      VOLUME_FRACTION = 3.7619, 1., BACKGROUND = .TRUE. /  
&SPEC ID = 'PRODUCTS', SPEC_ID = 'NITROGEN', 'CARBON DIOXIDE', 'WATER VAPOR',  
      VOLUME_FRACTION = 18.10631, 2.81813, 3.98990 /
```

The heat of combustion for propane is found in the SFPE Handbook [?].

```
&REAC FUEL = 'PROPANE', HEAT_OF_COMBUSTION=44715.,  
      SPEC_ID_NU = 'PROPANE', 'AIR', 'PRODUCTS', 'SOOT',  
      NU=-1., -4.81308, 1, 0.20208 /
```

Note: The sign of NU corresponds to whether that species is consumed (-) or produced (+). Figure 13.3 shows the soot surface deposition on the wall. This boundary quantity is given by the input below.

```
&BNDF QUANTITY='SURFACE DEPOSITION', SPEC_ID='SOOT' /
```



Figure 13.3: Wall soot deposition for the propane_flame_deposition test case.

Chapter 14

Radiation

For most FDS simulations, thermal radiation transport is computed by default and you need not set any parameters to make this happen. However, there are situations where it is important to be aware of issues related to the radiative transport solver.

14.1 Basic Radiation Parameters: The `RADI` Namelist Group

`RADI` is the namelist group that contains all of the parameters related to the radiation solver. There can be only one `RADI` line in the input file. It is possible to turn off the radiation transport solver (saving roughly 20 % in CPU time) by adding the statement `RADIATION=.FALSE.` to the `RADI` line. If burning is taking place and radiation is turned off, then the total heat release rate is reduced by the `RADIATIVE_FRACTION`, which is an input on the `REAC` line. This radiated energy completely disappears from the calculation. For fire scenarios it is not recommended that you turn off the radiation transport. This feature is used mainly for diagnostic purposes or when the changes in temperature are relatively small.

14.1.1 Radiative Fraction

The most important radiation parameter is the fraction of energy released from the fire as thermal radiation, commonly referred to as the *radiative fraction*, symbolically denoted χ_r . It is a function of both the flame temperature and chemical composition, neither of which are reliably calculated in a large scale fire calculation because the flame sheet is not well-resolved on a relatively coarse numerical grid. In calculations in which the mesh cells are on the order of a centimeter or larger, the temperature near the flame surface cannot be relied upon when computing the source term in the radiation transport equation, especially because of the T^4 dependence. As a practical alternative, the parameter `RADIATIVE_FRACTION` on the `REAC` line allows you to specify explicitly the fraction of the total combustion energy that is released in the form of thermal radiation. Some of that energy may be reabsorbed elsewhere, yielding a net radiative loss from the fire or compartment that is less than the `RADIATIVE_FRACTION`, depending mainly on the size of the fire and the soot loading. If it is desired to use the radiation transport equation as is, then `RADIATIVE_FRACTION` ought to be set to zero, and the source term in the radiative transport equation is then based solely on the gas temperature and the chemical composition. By default, the `RADIATIVE_FRACTION` is based on the reaction's `FUEL` for an LES calculation (see 14.1), and zero for DNS.

Multi-fuel radiative fraction If multiple fuels are present, e.g., one fuel for cardboard and one fuel for polystyrene combustion, then the radiant fraction will be computed locally as reaction-weighted value, sim-

Table 14.1: Default Radiative Fraction based on species

Species	χ_r
ACETONE	0.30
ACETYLENE	0.45
BENZENE	0.45
BUTANE	0.35
DODECANE	0.40
ETHANE	0.25
ETHANOL	0.20
ETHYLENE	0.35
HYDROGEN	0.10
ISOPROPANOL	0.30
METHANE	0.20
METHANOL	0.20
N-DECANE	0.40
N-HEPTANE	0.40
N-HEXANE	0.40
N-OCTANE	0.40
PROPANE	0.30
PROPYLENE	0.35
TOLUENE	0.45
All other species	0.35

ilar to the approach described by Gupta et al. [?]. For example, if the two fuels are 20% and 40% radiative fraction with, respectively, 80% and 20% of the heat in a grid cell, χ_r would be $0.8 \times 0.2 + 0.2 \times 0.4 = 0.24$. Thus, χ_r will vary in space and time. The value of the multi-fuel radiative fraction can be output using the output QUANTITY of CHI_R.

Time variation of radiative fraction Even for a single fuel species the global flame radiative fraction may depend on other parameters of the problem like global equivalent ratio. If a time variation of the radiative fraction is necessary, it may be added through a ramp function, RAMP_CHI_R, on the REAC line. The results of running the ramp_chi_r test case containing the RAMP given below is shown in Fig. 14.1.

```
&REAC FUEL='METHANE', RADIATIVE_FRACTION=1.0, RAMP_CHI_R='CHI_R RAMP' /
&RAMP ID='CHI_R RAMP', T= 0.0, F=0.238 /
&RAMP ID='CHI_R RAMP', T= 2.0, F=0.238 /
&RAMP ID='CHI_R RAMP', T= 4.0, F=0.182 /
&RAMP ID='CHI_R RAMP', T= 6.0, F=0.158 /
&RAMP ID='CHI_R RAMP', T= 8.0, F=0.140 /
&RAMP ID='CHI_R RAMP', T= 10.0, F=0.140 /
```

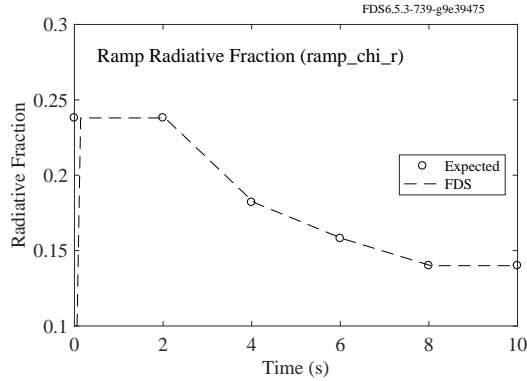


Figure 14.1: Results of the `ramp_chi_r` test case.

14.1.2 Spatial and Temporal Resolution of the Radiation Transport Equation

There are several ways to improve the spatial and temporal accuracy of the Finite Volume Method in solving the radiation transport equation (RTE), but these will increase the computation time. You can increase the number of angles from the default 100 with the integer parameter `NUMBER_RADIATION_ANGLES`. The frequency of calls to the radiation solver can be changed from every 3 time steps with an integer called `TIME_STEP_INCREMENT`. The increment over which the angles are updated can be reduced from 5 with the integer called `ANGLE_INCREMENT`. If `TIME_STEP_INCREMENT` and `ANGLE_INCREMENT` are both set to 1, the radiation field is completely updated in a single time step, but the cost of the calculation increases significantly. By default, the radiation transport equation is fully updated every 15 time steps.

If you are using multiple meshes, the radiation solver cannot transfer energy from mesh to mesh within a single time step. If you notice an obvious delay in the propagation of radiative intensity from one mesh to another, you can increase the number of times the radiative intensity is updated within a single time step using `RADIATION_ITERATIONS`, which is 1 by default.

The radiation solver is called before the start of the calculation to establish the radiation field in the event that you specify something to have a non-ambient temperature initially. By default, the radiation and wall boundary routines are iterated 10 times to establish thermal equilibrium. To change the number of iterations, set `NUMBER_INITIAL_ITERATIONS` on the `RADI` line.

14.2 Radiative Absorption and Scattering

By default FDS employs a gray gas model for the radiation absorption coefficient, a function of gas composition and temperature, which are tabulated in a look-up table using the routines found in RadCal. You can output the absorption coefficient using the output `QUANTITY 'ABSORPTION COEFFICIENT'`.

14.2.1 RadCal Considerations

There are several considerations with regard to RadCal:

Path Length

Because RadCal computes effective absorption coefficients over a range of wavelengths, it requires a user-specified `PATH_LENGTH` (m). Its default value is five times the width of a single grid cell.

Fuel Species

The original version of RadCal included only absorption data for methane, which was used as a surrogate for any fuel. However, more fuel species have been added to RadCal. The current list of fuels includes: METHANE, ETHYLENE, ETHANE, PROPANE, N-HEPTANE, METHANOL, TOLUENE, PROPYLENE, and MMA. These species are in addition to the RadCal species of: CARBON DIOXIDE, CARBON MONOXIDE, WATER VAPOR, and SOOT.

14.2.2 Radiative Absorption and Scattering by Particles

The absorption and scattering of thermal radiation by Lagrangian particles is included in the radiation transport equation. The radiative properties can be given by specifying the components of the material refractive index on the corresponding PART line, using keywords REAL_REFRACTIVE_INDEX and COMPLEX_REFRACTIVE_INDEX. Alternatively, wavelength dependent values of these two quantities can be tabulated in a TABLE and called using the RADIATIVE_PROPERTY_TABLE. More details can be found in Section 15.3.2.

The radiative properties of the water and fuel particles (droplets) are determined automatically. For fuel, the properties of heptane are assumed. The heptane values can be overridden by specifying them on the PART line.

Other parameters affecting the computations of particle-radiation interaction are listed here. RADTMP is the assumed radiative source temperature. It is used in the spectral weighting during the computation of the mean scattering and absorption cross sections. The default is 900 °C. NMIEANG is the number of angles in the numerical integration of the Mie-phase function. Increasing NMIEANG improves the accuracy of the radiative properties of water droplets. The cost of the better accuracy is seen in the initialization phase, not during the actual simulation. The default value for NMIEANG is 15. For each class of particles, the Mie coefficients are calculated for a wide range of droplet diameters to ensure that all possible run-time situations can be covered. To speed up the initialization phase, the range of diameters can be limited by parameters MIE_MINIMUM_DIAMETER and MIE_MAXIMUM_DIAMETER. Also, the size of the Mie coefficient tables can be specified using MIE_NDG parameter.

The radiation properties of most common gases involved in combustion processes (water vapor, carbon dioxide, carbon monoxide, fuel) and soot particles are automatically taken into account if the simulation involves combustion. In simulations with no combustion nor radiating species, it is possible to use a constant absorption coefficient by specifying KAPPA0 on the RADI line.

14.2.3 Wide Band Model

The radiation solver has two modes of operation – a gray gas model (default) and a wide band model [?]. If the optional six band model is desired, set WIDE_BAND_MODEL=.TRUE.. It is recommended that this option only be used when the fuel is relatively non-sooting because it adds significantly to the cost of the calculation. Read the FDS Technical Reference Guide [?] for more details. Note also that when WIDE_BAND_MODEL=.TRUE., the output QUANTITY 'ABSORPTION COEFFICIENT' becomes practically useless, because it then corresponds to one individual band of the spectrum.

It is also possible to set your own band limits for the wide band model by specifying BAND_LIMITS on the RADI line. The limits should be given in ascending order, in units of microns (μm). The maximum number of bands is 9, in which case you would specify 10 real numbers separated by commas.

Chapter 15

Particles and Droplets

Lagrangian particles can be used to represent a wide variety of objects that are too small to resolve on the numerical grid. FDS considers three major classes of Lagrangian particles: massless tracers, liquid droplets, and everything else. The parameters describing particles are found on the `PART` line.

15.1 Basics

Properties of different types of Lagrangian particles are designated via the `PART` namelist group. Once a particular type of particle has been described using a `PART` line, then the name of that particle type is invoked elsewhere in the input file via the parameter `PART_ID`. There are no reserved `PART_ID`s – all must be defined. For example, an input file may have several `PART` lines that include the properties of different types of Lagrangian particles:

```
&PART ID='my smoke',... /  
&PART ID='my water',... /
```

Particles are introduced into the calculation in several different ways: they may be introduced via a sprinkler or nozzle (liquid droplets are usually introduced this way), they may be introduced at a blowing vent or burning surface (mass tracer particles or particles representing embers are usually introduced this way), and they may be introduced randomly or at fixed points within a designated volume (solid particles that represent subgrid-scale objects are usually introduced this way). Details are found below.

The way to describe particles depends on the type. If you simply want massless tracers, specify `MASSLESS=.TRUE.` on the `PART` line. If you specify a `SPEC_ID`, then FDS automatically assumes that you want relatively small, thermally-thin evaporating liquid droplets. For any other type of particle, such as particles that represent subgrid-scale objects, like office clutter or vegetation, you add a `SURF_ID` to the `PART` line. All of these different types of particles are described below.

15.2 Massless Particles

The simplest use of Lagrangian particles is for visualization, in which case the particles are considered massless tracers. In this case, the particles are defined via the line

```
&PART ID='tracers', MASSLESS=.TRUE., ... /
```

Note that if the particles are `MASSLESS`, it is not appropriate to color them according to any particular property. Particles are not colored by gas phase quantities, but rather by properties of the particle itself. For

example, 'PARTICLE TEMPERATURE' for a non-massless particle refers to the temperature of the particle itself rather than the local gas temperature. Also note that if `MASSLESS=.TRUE.`, the `SAMPLING_FACTOR` (Section 18.9) is set to 1 unless you say otherwise, which would be pointless since `MASSLESS` particles are for visualization only.

Turbulent Dispersion

Massless tracer particles may also be useful in modeling dispersion of a tracer gas that does not affect the mean flow field (passive scalar). The number density of the particles then may be translated into a local mass concentration. See how to output number concentration in Sec. 18.10.7. To properly account for subgrid-scale (unresolved) turbulent motions, add the parameter `TURBULENT_DISPERSION=.TRUE.` to the `PART` line. The particles will then undergo a random walk based on the subgrid diffusivity. For further information, see the `random_walk` test cases in the `WUI` directory of the verification suite.

15.3 Liquid Droplets

To define an evaporating liquid droplet, you must explicitly specify the gaseous species via the `SPEC` namelist group (see Section 12), and then designate the appropriate `SPEC_ID` on the `PART` line. If the droplets are defined with `SPEC_ID='WATER VAPOR'`, then the particles will be assigned the thermo-physical properties of water, the radiation absorption properties of water, and will be colored blue in Smokeview.

15.3.1 Thermal Properties

The following parameters should be specified to control the evaporation. The `INITIAL_TEMPERATURE` of liquid droplet; assumed ambient, `TMPA` (°C) is specified on `PART` input. If the fluid given by the `SPEC_ID` is included in Table 12.1, then no further inputs are required. Otherwise, you must provide all of the following properties of the liquid on the `SPEC` input:

`DENSITY_LIQUID` The density of the liquid or solid droplet/particle (kg/m^3).

`SPECIFIC_HEAT_LIQUID` Specific heat of liquid or solid droplet/particle ($\text{kJ}/(\text{kg} \cdot \text{K})$).

`RAMP_CP_L` Ramp of temperature vs. specific heat for the solid droplet/particle.

`VAPORIZATION_TEMPERATURE` Boiling temperature of liquid droplet (°C).

`MELTING_TEMPERATURE` Melting (solidification) temperature of liquid droplet (°C).

`HEAT_OF_VAPORIZATION` Latent heat of vaporization of liquid droplet (kJ/kg).

`ENTHALPY_OF_FORMATION` The heat of formation of the gas (kJ/mol).

`H_V_REFERENCE_TEMPERATURE` The temperature corresponding to the provided `HEAT_OF_VAPORIZATION` (°C).

15.3.2 Radiative Properties

The radiative properties of water and fuel droplets are determined automatically. For fuel, the properties of heptane are assumed. For other types of particles, the radiative properties can be given by specifying the components of the material refractive index on the corresponding `PART` line, using keywords

REAL_REFRACTIVE_INDEX and COMPLEX_REFRACTIVE_INDEX. Alternatively, wavelength dependent values of these two quantities can be specified using a spectral property TABLE and specifying the ID of that table is RADIATIVE_PROPERTY_TABLE property on the PART line. Each row of a spectral property table contains three real numbers: wavelength (μm), real and complex components of the refractive index. The real part of the refractive index should be a positive number. If it is greater than 10.0, the particles are treated as perfectly reflecting spheres. The complex part should be a non-negative number. Values less than 10^{-6} are treated as non-absorbing. Below is an example of the use of spectral property table, listing the properties at wavelengths 1, 5 and 10 μm .

```
&PART ID='particles',..., RADIATIVE_PROPERTY_TABLE='table' /
&TABL ID='table', TABLE_DATA= 1.0,1.33,0.0001 /
&TABL ID='table', TABLE_DATA= 5.0,1.33,0.002 /
&TABL ID='table', TABLE_DATA=10.0,1.33,0.001 /
```

For calculating the absorption of thermal radiation by particles, FDS uses a running average of particle temperature and density. The default averaging factor, RUN_AVG_FAC, is set to 0.5.

15.3.3 Size Distribution

The size distribution of liquid droplets is specified using a cumulative volume fraction (CVF)¹ indicated by the character string DISTRIBUTION on the PART line. The default is 'ROSIN-RAMMLER-LOGNORMAL':

$$F(D) = \begin{cases} \frac{1}{\sqrt{2\pi}} \int_0^D \frac{1}{\sigma D'} \exp\left(-\frac{[\ln(D'/D_{v,0.5})]^2}{2\sigma^2}\right) dD' & (D \leq D_{v,0.5}) \\ 1 - \exp\left(-0.693\left(\frac{D}{D_{v,0.5}}\right)^\gamma\right) & (D > D_{v,0.5}) \end{cases} \quad (15.1)$$

Alternatively, you can specify 'LOGNORMAL' or 'ROSIN-RAMMLER' alone rather than the combination of the two. Figure 15.1 displays the possible size distributions. Notice that the 'LOGNORMAL' and 'ROSIN-RAMMLER' distributions have undesirable attributes at opposite tails, which is why the combination of the two is commonly used. Figure 15.1 also shows a comparison between the prescribed distribution and the actual realized distribution of droplet sizes. The dashed lines show the measured droplet size distributions, while the solid lines show the prescribed sampling distributions. The sampled distributions are measured with the PDPA_HISTOGRAM function. Sample size of 10000 droplets was used.

The median volumetric diameter, $D_{v,0.5}$, is specified via the parameter DIAMETER (μm) on the PART line. You must specify the DIAMETER in cases where the droplets evaporate (in which case you also need to specify a SPEC_ID to indicate the gas species generated by the evaporating droplets). The width of the lognormal distribution, σ , is specified with SIGMA_D on the PART line. The width of the Rosin-Rammler distribution, γ , is specified with GAMMA_D (default 2.4). Note that in the combined distribution, the parameter, σ , is calculated $\sigma = 2/(\sqrt{2\pi}(\ln 2)^\gamma) = 1.15/\gamma$ which ensures that the two functions are smoothly joined at $D = D_{v,0.5}$. You can also add a value for SIGMA_D to the PART line if you want to over-ride this feature. The larger the value of γ , the narrower the droplet size is distributed about the median value.

You can specify your own cumulative number fraction (CNF)² by specifying a CNF_RAMP_ID on the PART line and including a RAMP that gives the CNF:

```
&PART ID='my droplets',..., CNF_RAMP_ID='my CNF' /
&RAMP ID='my CNF', T= 0., F=0.000000 /
```

¹The CVF indicates the fraction of total mass carried by droplets less than the given diameter.

²The CNF indicates the fraction of total droplets whose diameters are less than the given diameter.

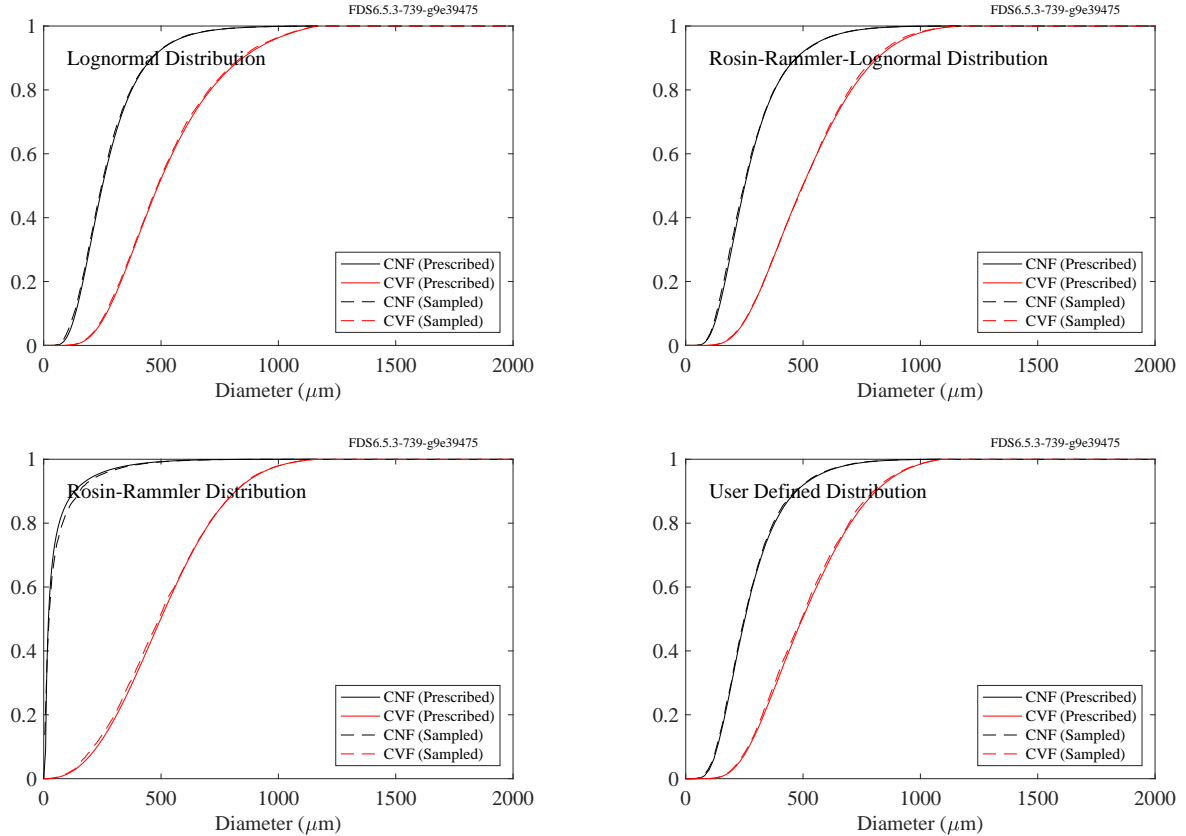


Figure 15.1: Droplet size distributions. The first three plots are based on a specified CDF from which the CNF is derived. The fourth plot (lower right) is an example of a specified CNF from which the CDF is derived. CDF:s are plotted with black lines, while CNF:s are plotted with red lines. The solid lines show the prescribed sampling distribution, while the dashed lines show the actual sampled droplet size distribution, measured with the `PDPA_HISTOGRAM` functionality.

```
&RAMP ID='my CNF', T= 200., F=0.000003 /
...
&RAMP ID='my CNF', T=2000., F=1.000000 /
```

Note that the `RAMP` variable `T` indicates the diameter and is given in micrometers. The fourth plot in Fig. 15.1 is an example of where the CNF is specified and the CDF is calculated from it. It is essentially the reverse of what is shown in the first plot, where the CDF is specified and the CNF is calculated from it.

As droplets are created in the simulation, their diameters are randomly chosen based on the given distribution. You can prevent excessively large droplets from being chosen by specifying a `MAXIMUM_DIAMETER`, which is assigned an infinitely large value by default. Droplets less than a specified `MINIMUM_DIAMETER` are assumed to evaporate in a single time step. The default value is 0.005 times the value of `DIAMETER`. The droplet distribution is divided into a series of bins³. To avoid very small particle weights, the distribution is clipped at the cumulative fractions of `CNF_CUTOFF` and $(1 - \text{CNF_CUTOFF})$. Note that `CNF_CUTOFF` is set on the `MISC` line. The default value of `CNF_CUTOFF` is 0.005.

³By default, the range of particle sizes is divided into six bins, and the sampled particles are divided among these bins. This ensures that a reasonable number of particles are assigned to the entire spectrum of sizes. To change the default number of bins, set `N_STRATA` on the `PART` line.

To prevent FDS from generating a distribution of droplets altogether, set `MONODISPERSE` to `.TRUE.` on the `PART` line, in which case every droplet will be assigned the same `DIAMETER`.

If you set `CHECK_DISTRIBUTION=.TRUE.` on the `PART` line, FDS will write out the cumulative distribution function for that particular particle class in a file called `CHID_PART_ID_cdf.csv`. If you do this, you might want to avoid spaces in the `ID` of the `PART` line.

15.3.4 Secondary Breakup

If `BREAKUP=.TRUE.` is set on the `PART` line, particles may undergo secondary breakup. In this case you should also specify the `SURFACE_TENSION` (N/m) of the liquid and the resulting ratio of the Sauter mean diameters, `BREAKUP_RATIO`. Its default is 3/7. Optionally, specify the distribution parameters `BREAKUP_GAMMA_D` and `BREAKUP_SIGMA_D`.

15.3.5 Fuel Droplets

If the droplets evaporate into the `FUEL` identified on the `REAC` line, they will be colored yellow by default in Smokeview and any resulting fuel vapor will burn according to the combustion model specified on the `REAC` line. The droplets evaporate into an equivalent amount of fuel vapor such that the resulting heat release rate (assuming complete combustion) is equal to the evaporation rate multiplied by the `HEAT_OF_COMBUSTION`, also specified on the `PART` line. Note that the burning rate will be adjusted to account for the difference between the heats of combustion of the droplets and the other fuels in the model.

If a spray nozzle is used to generate the fuel droplets, its characteristics are specified in the same way as those for a sprinkler. If the fuel species is present in the liquid properties table as a fuel, then the droplets will be given fuel radiation absorption properties.

Note that to limit the computational cost of sprinkler simulations, liquid droplets disappear when they hit the “floor” of the computational domain, regardless of whether it is solid or not. However, this may not be desired when using liquid fuels. To stop FDS from removing liquid droplets from the floor of the domain, add the phrase `POROUS_FLOOR=.FALSE.` to the `MISC` line. An alternate solution is make sure the `OBST` the fuel is landing on is at least one cell thick.

Example Case: spray_burner

Controlled fire experiments are often conducted using a spray burner, where a liquid fuel is sprayed out of a nozzle and ignited. In this example (`spray_burner.fds`), heptane from two nozzles is sprayed downwards into a steel pan. The flow rate is increased linearly so that the fire grows to 2 MW in 20 s, burns steadily for another 20 s, and then ramps down linearly in 20 s. The key input parameters are given here:

```
&REAC FUEL='N-HEPTANE',SOOT_YIELD=0.01,HEAT_OF_COMBUSTION=44500./

&DEVC ID='nozzle_1',XYZ=4.0,-.3,0.5,PROP_ID='nozzle',QUANTITY='TIME',SETPOINT=0./
&DEVC ID='nozzle_2',XYZ=4.0,0.3,0.5,PROP_ID='nozzle',QUANTITY='TIME',SETPOINT=0./

&PART ID='heptane droplets',SPEC_ID='N-HEPTANE',
      QUANTITIES(1:2)='PARTICLE DIAMETER','PARTICLE TEMPERATURE',
      DIAMETER=1000.,HEAT_OF_COMBUSTION=44500.,SAMPLING_FACTOR=1/

&PROP ID='nozzle',CLASS='NOZZLE',PART_ID='heptane droplets',
      FLOW_RATE=1.97,FLOW_RAMP='fuel',PARTICLE_VELOCITY=10.,
      SPRAY_ANGLE=0.,30./
&RAMP ID='fuel',T=0.0,F=0.0/
&RAMP ID='fuel',T=20.0,F=1.0/
```

```
&RAMP ID='fuel', T=40.0, F=1.0 /
&RAMP ID='fuel', T=60.0, F=0.0 /
```

Many of these parameters are self-explanatory. Note that a 2 MW fire is achieved via 2 nozzles flowing heptane at 1.96 L/min each:

$$2 \times 1.97 \frac{\text{L}}{\text{min}} \times \frac{1}{60} \frac{\text{min}}{\text{s}} \times 684 \frac{\text{kg}}{\text{m}^3} \times \frac{1}{1000} \frac{\text{m}^3}{\text{L}} \times 44500 \frac{\text{kJ}}{\text{kg}} = 2000 \text{ kW} \quad (15.2)$$

The parameter `HEAT_OF_COMBUSTION` over-rides that for the overall reaction scheme. Thus, if other droplets or solid objects have different heats of combustion, the effective burning rates are adjusted so that the total heat release rate is that which you expect. However, exercises like this ought to be conducted just to ensure that this is the case. The HRR curve for this example is given in Fig. 15.2.

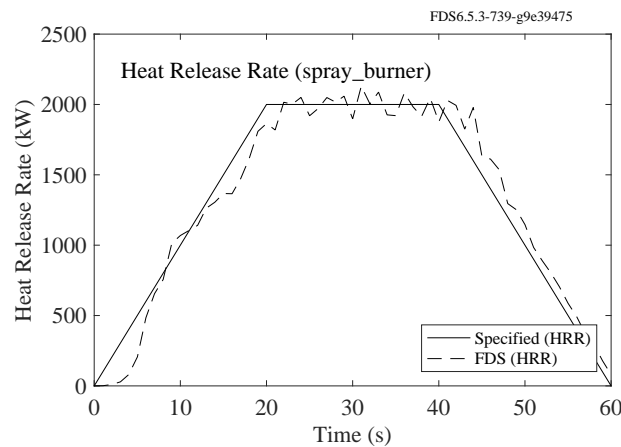


Figure 15.2: Heat Release Rate (HRR) of spray burner test.

Note also that this feature is subject to mesh dependence. If the mesh cells are too coarse, the evaporating fuel can be diluted to such a degree that it may not burn. Proper resolution depends on the type of fuel and the amount of fuel being ejected from the nozzle. Always test your burner at the resolution of your overall simulation.

15.4 Solid Particles

Lagrangian particles can represent a wide variety of subgrid-scale objects, from office clutter to vegetation. To create solid, non-liquid particles, you must add a `SURF_ID` to the `PART` line. The specified `SURF` line contains the parameters that describe the thermophysical properties and geometric parameters of the particle. These properties are the same as those you would apply to an `OBST` or `VENT`. FDS uses the same solid phase conduction and pyrolysis algorithm for particles as it does for solid walls.

If the `SURF` line that is associated with the particle class calls for it, the particles will heat up due to convection from the surrounding gases and radiation from near and distant sources. The convective heat transfer coefficient takes into account the particle geometry, and the radiative heat flux is based on the integrated intensity. That is, the radiation heat flux is the average over all angles. However, you can specify unique directions for the particle if the source of heating does not surround the particles. More about particle splitting is explained in Section 15.4.3.

15.4.1 Basic Geometry and Boundary Conditions

To demonstrate the basic syntax for solid particles, the following input lines create a collection of hot spheres:

```
&PART ID='spheres', SURF_ID='HOT', STATIC=.TRUE., PROP_ID='ball' /
&SURF ID='HOT', TMP_FRONT=500., RADIUS=0.005, GEOMETRY='SPHERICAL' /
&PROP ID='ball', SMOKEVIEW_ID='SPHERE', SMOKEVIEW_PARAMETERS(1)='D=0.01' /
&INIT PART_ID='spheres', XB=0.25,0.75,0.25,0.75,0.25,0.75, N_PARTICLES=10 /
```

The `PART` line establishes the class of particles. In this case, the presence of a `SURF_ID` indicates that the particles are solids with the properties given by the `SURF` line 'HOT'. `STATIC` is a logical parameter whose default is `.FALSE.` that indicates if the particles are stationary. The `PROP_ID` references a `PROP` (property) line that just tells Smokeview that the particles are to be drawn as spheres of diameter 0.01 m. See Section 17.7.3 for details and options. The `INIT` line randomly fills the given volume with 10 of these hot spheres. See Section 15.5.3 for details.

If the `SURF` line includes a `MATL_ID`, the particle mass will be based upon the value(s) of `DENSITY` of the referenced `MATL` line(s). If there is to be no heat conduction calculation in depth, do not specify a `MATL_ID`. Instead, you can specify, for example, the surface temperature, `TMP_FRONT` (°C), heat release rate per unit area, `HRRPUA` (kW/m²), or species `MASS_FLUX` (kg/(m²·s)).

The `GEOMETRY` options for solid particles are 'SPHERICAL', 'CYLINDRICAL', or 'CARTESIAN'. By default, the `GEOMETRY` is 'CARTESIAN', in which case you need to provide the `LENGTH` and `WIDTH` of the rectangular plate. It is assumed that the plate is symmetric front and back (note this means you should set `BACKING='INSULATED'` on the `SURF` line). You need only specify the layers that make up the half-thickness. The array `THICKNESS(N)` indicates the thickness(es) of each layer of the plate, not the total thickness of the plate itself. If the plate is composed of only one material component, the specified `THICKNESS` is taken as the half-thickness of the plate.

For 'CYLINDRICAL' or 'SPHERICAL' particles, specify the `INNER_RADIUS` and `THICKNESS` of the individual layers. Alternatively, you can just specify the `RADIUS` if the cylinder or sphere is solid and has only one material component. The default value of `INNER_RADIUS` is 0 m, which means that the radius of the cylinder or sphere is the sum of the `THICKNESS` values. Remember that the layers are to be listed starting at the surface, not the center. For 'CYLINDRICAL' particles, specify a `LENGTH` as well.

15.4.2 Drag

For solid particles, the default drag law (i.e., the drag coefficient correlation as a function of the Reynolds number based on particle diameter) is that of a sphere. To invoke the cylinder drag law, set `DRAG_LAW` to 'CYLINDER' on the `PART` line, and to invoke the screen drag law (see Section 15.4.8), set it to 'SCREEN'. If neither of these options is applicable, you may specify a constant value of the drag coefficient for a particle class (a specific `PART_ID`) by setting a `DRAG_COEFFICIENT` on the `PART` line. The `DRAG_COEFFICIENT` over-rides the `DRAG_LAW`. If the local particle density is very high, the drag may be reduced by particle wake effects. The threshold volume fraction for considering three-way coupling effects is controlled by the `DENSE_VOLUME_FRACTION` parameter. Setting this parameter to 1 will turn off the three-way coupling model.

15.4.3 Splitting Particles

If the radiation heat flux is not uniformly distributed about the particle, it may be useful to split the particle into pieces, each with its own orientation. This can be done by using the parameter `ORIENTATION` to specify more than one outward facing directions for the particle. For example,

```
&PART ..., ORIENTATION(1:3,1)=0,0,-1, ORIENTATION(1:3,2)=0,0,1 /
```

specifies that half the particle is facing downwards and half is facing upwards. FDS now essentially tracks two particles. The radiative flux to the downward facing particle is the integrated average over the southern hemisphere; the flux to the upward facing particle is the average over the northern hemisphere. The particle mass and the surface area are scaled by the number of orientations. The splitting along the coordinate axis is demonstrated for all geometries in Fig. 15.3. The orientation direction does not have to align with the coordinate axes. In fact, the `ORIENTATIONS` do not have to be symmetric or come in pairs, even though for most applications it makes sense to do it this way. For a Cartesian particle (i.e., a plate), only the orientations that are perpendicular to the plate make physical sense. Keep in mind that the heat conducted within the different facets does not transfer through to the other facets. The heat conduction is still only one dimensional, in the direction normal to the face and towards the center.

Note that if only a single `ORIENTATION` vector is assigned on a `PART` line, the radiative flux to the particle is calculated as if there is a flat plate normal to the direction of the vector, like a conventional heat flux gauge. That is, the heat flux is not an integrated average over the entire particle but rather the directional heat flux with the given orientation. The reason for this exception to the general rule is that often single particles are used as “targets” to record a heat flux at a given point in the domain with a given orientation. These particles can be thought of as tiny heat flux gauges that do not disturb the flow.

15.4.4 Gas Generating Particles

Lagrangian particles can be used to generate gases at a specified rate. The syntax is similar to that used for a solid wall. For example, the following input lines create three particles – one shaped like a rectangular plate, one a cylinder, and one a sphere – that generate argon, sulfur dioxide, and helium, respectively. The particles have no mass; they simply are used to generate the gases at a specified rate.

```
&SPEC ID='ARGON' /  
&SPEC ID='SULFUR DIOXIDE' /  
&SPEC ID='HELIUM' /  
  
&INIT PART_ID='plate', XYZ=-1.,0.,1.5, N_PARTICLES=1 /
```

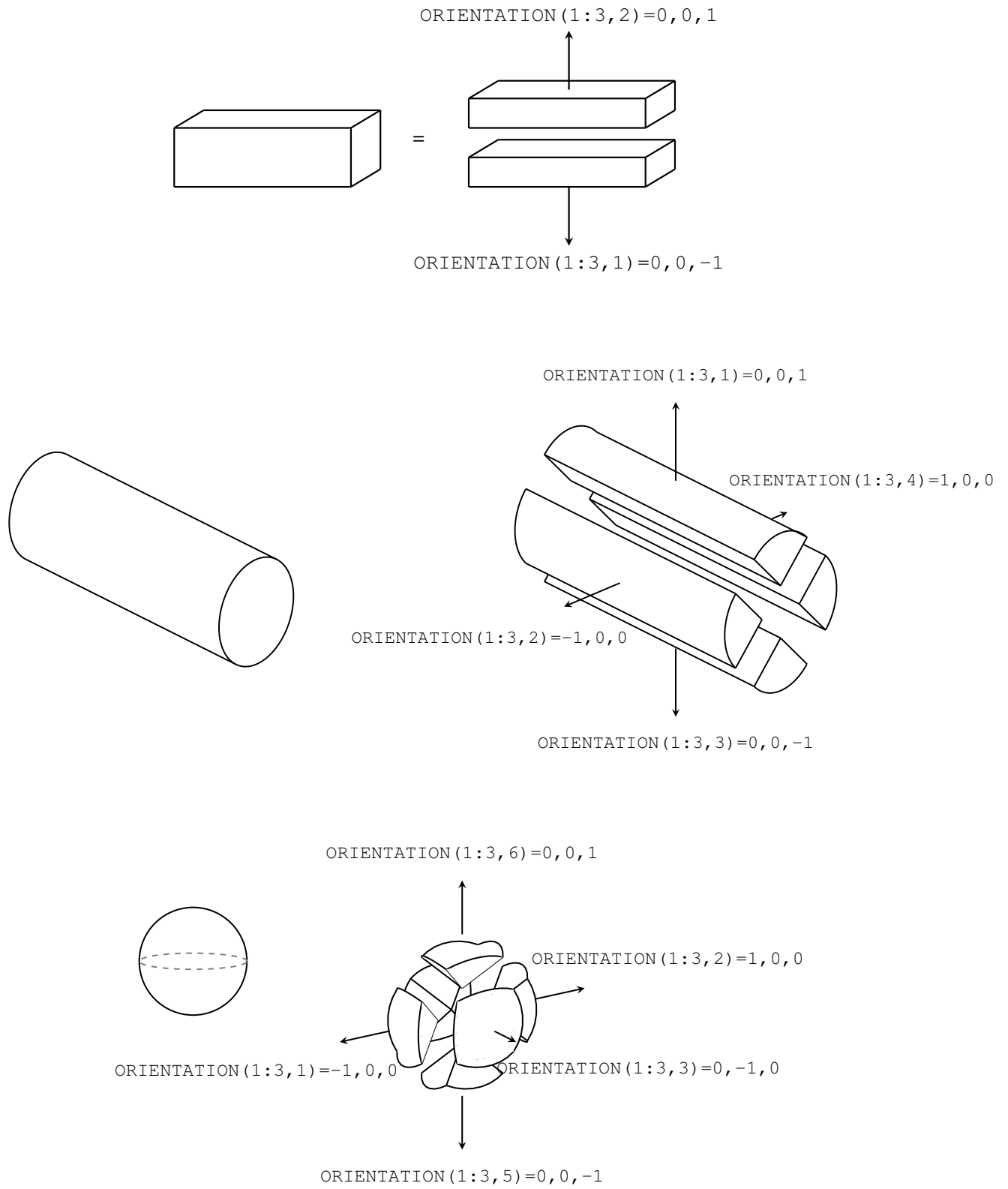


Figure 15.3: Examples of a Cartesian (plate), cylindrical, and spherical particle split into multiple parts.

```

&INIT PART_ID='tube', XYZ= 0.,0.,1.5, N_PARTICLES=1 /
&INIT PART_ID='ball', XYZ= 1.,0.,1.5, N_PARTICLES=1 /

&PART ID='plate', SAMPLING_FACTOR=1, SURF_ID='plate bc', STATIC=.TRUE. /
&PART ID='tube', SAMPLING_FACTOR=1, SURF_ID='tube bc', STATIC=.TRUE. /
&PART ID='ball', SAMPLING_FACTOR=1, SURF_ID='ball bc', STATIC=.TRUE. /

&SURF ID='plate bc', THICKNESS=0.001, LENGTH=0.05, WIDTH=0.05, SPEC_ID(1)='ARGON',
MASS_FLUX(1)=0.1, TAU_MF(1)=0.001 /
&SURF ID='tube bc', GEOMETRY='CYLINDRICAL', LENGTH=0.05, RADIUS=0.01,
SPEC_ID(1)='SULFUR DIOXIDE', MASS_FLUX(1)=0.1, TAU_MF(1)=0.001 /
&SURF ID='ball bc', GEOMETRY='SPHERICAL', RADIUS=0.01, SPEC_ID(1)='HELIUM',
MASS_FLUX(1)=0.1, TAU_MF(1)=0.001 /

```

In this case, there is no calculation of heat conduction in depth. Only the surface area is important. For the plate, the surface area is twice the length times the width. For the cylinder, the area is twice the radius times π times the length. For the sphere, the area is 4π times the radius squared. Figure 15.4 displays the output of the test case called `surf_mass_part_specified.fds`, demonstrating that the production rate of the gases is as expected.

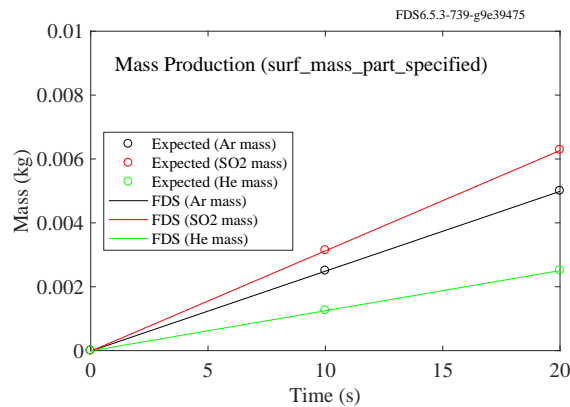


Figure 15.4: Gas production from three Lagrangian particles.

15.4.5 Vegetation

Lagrangian particles can be used to represent different types of vegetation, like leaves, grass, and so on. The best way to explain how to use this feature is by way of example. Suppose we want to describe a collection of wet pine needles that occupy a certain volume. The following lines have been extracted from the sample file `WUI/pine_needles.fds`. Note that all of the values have been chosen simply to demonstrate the technique. These values should not be used for a real calculation.

```

&PART ID='pine needles', SAMPLING_FACTOR=1, SURF_ID='wet vegetation',
PROP_ID='needle image', STATIC=.TRUE. /
&INIT PART_ID='pine needles', XB=0.,1.,0.,1.,0.,1., N_PARTICLES=1000,
MASS_PER_VOLUME=1. /
&PROP ID='needle image', SMOKEVIEW_ID='TUBE',
SMOKEVIEW_PARAMETERS='L=0.1','D=0.0005' /

```

```

&SURF ID = 'wet vegetation'
    MATL_ID(1,1:2) = 'PINE','MOISTURE'
    MATL_MASS_FRACTION(1,1:2) = 0.8,0.2
    THICKNESS = 0.00025
    LENGTH = 0.1
    GEOMETRY = 'CYLINDRICAL' /

&MATL ID = 'PINE'
    DENSITY = 500.
    CONDUCTIVITY = 0.1
    SPECIFIC_HEAT = 1.0
    N_REACTIONS = 1
    REFERENCE_TEMPERATURE = 300.
    NU_MATL = 0.2
    NU_SPEC = 0.8
    SPEC_ID = 'GLUCOSE'
    HEAT_OF_REACTION = 1000
    MATL_ID = 'CHAR' /

&MATL ID = 'MOISTURE'
    DENSITY = 1000.
    CONDUCTIVITY = 0.1
    SPECIFIC_HEAT = 4.184
    N_REACTIONS = 1
    REFERENCE_TEMPERATURE = 100.
    NU_SPEC = 1.0
    SPEC_ID = 'WATER VAPOR'
    HEAT_OF_REACTION = 2500. /

&MATL ID = 'CHAR'
    DENSITY = 200.
    CONDUCTIVITY = 1.0
    SPECIFIC_HEAT = 1.6 /

```

In the example, 1 kg of pine needles occupy 1 m³. The number of particles used to represent the pine needles is somewhat arbitrary. FDS will automatically weight the specified number so that the total mass per volume is 1 kg. The needles are modeled as cylinders that are 0.5 mm in diameter. The `THICKNESS` on the `SURF` line refers to the radius of the cylinder in units of m. The needles are all 0.1 m long. The needles contain 20 % (by mass) moisture, and 80 % cellulose. The moisture is set to evaporate at 100 °C to create water vapor and the cellulose pyrolyzes at 300 °C to form fuel gas and char. In the example case, the original 1 kg of vegetation is heated until all of the water and fuel evaporate. The fuel is not allowed to burn by setting the ambient oxygen concentration to 1 %. Figure 15.5 shows the evolution of the fuel, water and char mass. Agreement with the expected values means that mass is conserved.

15.4.6 Firebrands

Firebrands are small pieces of burning wood and vegetation that can be lofted into the air and blown by the wind ahead of a wildland fire front. Manzello et al. [?] have developed a variety of experimental apparatus designed to generate firebrands in a laboratory setting. The example input file called `dragon_5a` in the `WUI` (Wildland-Urban Interface) folder is a very simple mock-up of one of these experiments. The word “dragon” is based on the nickname of the apparatus; 5a is the figure number in Ref. [?] on which this example case is loosely based. In the experiment, 700 g of small dowels (length 50 mm, diameter 8 mm) made of Ponderosa Pine wood were poured into a small steel chamber equipped with several propane burners. The dowels

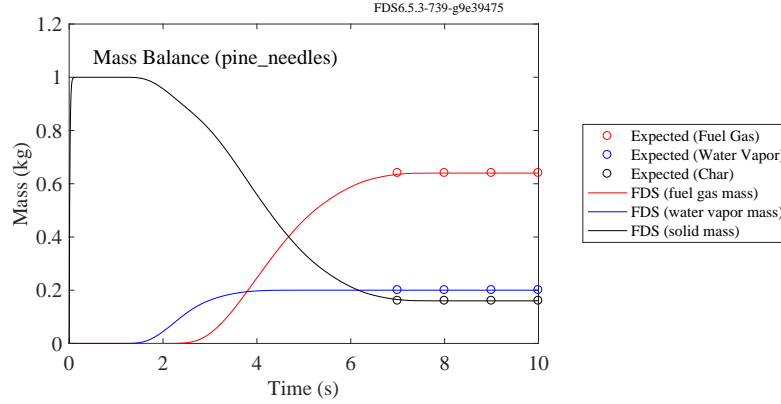


Figure 15.5: Evolution of vegetation mass in the `pine_needles` test case.

were left to burn for roughly a minute subject to a slow induced air flow after which time the air flow was increased and firebrands were propelled horizontally out of a 15 cm duct 2.25 m above the lab floor. It is reported that after several replicate experiments, the average mass of the firebrands collected from pans on the floor was 57 g. The average diameter of the collected dowels was 5.6 mm, and the average length was 13.5 mm.

It is not possible to simulate the experiment in FDS exactly as it was performed. The reason is that in the experiment, all 700 g of the wooden dowels were poured into the heating chamber at once. FDS cannot handle such a dense packing of Lagrangian particles. Instead, the simulated dowels are introduced at a rate of 10 per second. FDS also does not have a mechanism to break-up the dowels, reducing their length from 50 mm to 13.5 mm. Thus, the initial cylindrical particles are 13.5 mm and remain that length throughout the simulation. The diameter of the cylindrical particles is reduced, however, from 8 mm to 5.6 mm, which takes the initial density of 440 kg/m^3 down to 71 kg/m^3 because the mass of the firebrands is assumed to be 8 % of the original. The plot in Fig. 15.6 shows the increasing mass of firebrands thrown to the floor in the simulation after 100 s of particle insertion. The total mass of particles inserted into the apparatus is:

$$\pi (0.004 \text{ m})^2 \times (0.0135 \text{ m}) \times (440 \text{ kg/m}^3) \times (10 \text{ part/s}) \times (100 \text{ s}) \approx 0.3 \text{ kg} \quad (15.3)$$

The amount expected on the floor is approximately 0.024 kg.

15.4.7 Porous Media

A 3-D array of particles can be used to represent the drag exerted by porous media, as in the following example:

```
&SURF ID='LIGAMENT', MATL_ID='ALUMINUM ALLOY', THICKNESS=7.3E-5,
      GEOMETRY='CYLINDRICAL', HEAT_TRANSFER_COEFFICIENT=10. /
&MATL ID='ALUMINUM ALLOY', DENSITY=2690., CONDUCTIVITY=218., SPECIFIC_HEAT=0.9 /
&PART ID='FOAM', DRAG_LAW='POROUS MEDIA', SURF_ID='LIGAMENT',
      POROUS_VOLUME_FRACTION=0.12, STATIC=.TRUE.,
      DRAG_COEFFICIENT=0.1,0.1,0.1, PERMEABILITY=1.0E-7,1.E-7,1.E-7 /
&INIT XB=1.010,1.095,0.0,0.5,0.0,0.5, N_PARTICLES_PER_CELL=1, CELL_CENTERED=.TRUE.,
      PART_ID='FOAM' /
```

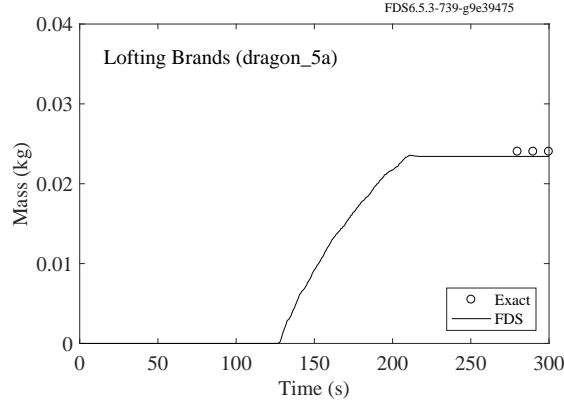



Figure 15.6: Mass generation of firebrands in the `dragon_5a` test case.

These lines are a model of aluminum foam. The basic geometry of the foam ligaments is defined with the `SURF` line. It is assumed that the ligaments are made of an aluminum alloy whose properties are given on the `MATL` line. The radius of the assumed cylindrical ligament is indicated by the `THICKNESS`. Note that the `LENGTH` of the cylinder which is normally required on the `SURF` line is computed automatically so that the volume fraction of the grid cell occupied by the foam, specified by `POROUS_VOLUME_FRACTION` on the `PART` line, is achieved. In a sense, the foam is modeled by a long cylinder chopped up into small pieces and represented by a single particle in each grid cell.

The `PART` line provides information about the particles. The `DRAG_LAW` indicates a special empirical model for the porous media. This model states that the pressure drop through the media is given by

$$\Delta p = \delta \left(\frac{\mu}{K} u + \rho \frac{Y}{\sqrt{K}} u^2 \right) \quad (15.4)$$

where δ is the thickness of the foam block in the flow direction, μ is the viscosity of the gas, u is the velocity component in the flow direction, ρ is the density of the gas, K is the `PERMEABILITY` in units of m^2 , and Y is a dimensionless inertial term that you specify using the parameter `DRAG_COEFFICIENT`. When using the porous media model the `PERMEABILITY` and `DRAG_COEFFICIENT` must be specified for all three directions.

The `INIT` line designates the volume occupied by the porous media using the sextuplet `XB`. A single particle is inserted into the center of each cell occupied by the foam by specifying the parameters `N_PARTICLES_PER_CELL=1` and `CELL_CENTERED=.TRUE.`.

A sample calculation involving porous media is contained in the folder `Sprinklers_and_Sprays`. The input file is called `porous_media.fds`.

15.4.8 Screens

A 2-D array of particles can be used to represent the drag exerted by a window screen, as in the following example:

```
&INIT N_PARTICLES_PER_CELL=1, CELL_CENTERED=.TRUE., PART_ID='SCREEN',
      XB=1.01,1.02,0.0,1.0,0.0,1.0/
&PART ID='SCREEN', DRAG_LAW='SCREEN', FREE_AREA_FRACTION=0.4, STATIC=.TRUE.,
      SURF_ID='SCREEN', ORIENTATION(1:3,1)=1,0,0 /
&SURF ID='SCREEN', THICKNESS=0.00015, GEOMETRY='CYLINDRICAL',
```

```
MATL_ID='ALUMINUM' /
&MATL ID='ALUMINUM', DENSITY=2700., CONDUCTIVITY=200., SPECIFIC_HEAT=0.9 /
```

The `INIT` line designates the plane of the screen using the sextuplet `XB`. A single particle is inserted into each cell by specifying the parameter `N_PARTICLES_PER_CELL=1`. The particles are defined with a `SURF_ID` containing the material properties of the screen. A special drag law for screens is specified via the `DRAG_LAW`. `ORIENTATION` is the direction normal to the screen, and `FREE_AREA_FRACTION` is the fraction of the screen's surface area that is open. In the example, an aluminum screen with a 40 % free area and an 0.0003 m wire diameter is placed normal to the x -axis. Note that the `LENGTH` parameter on the `SURF` line will be computed automatically so the fraction of the grid cell flow area occupied by the screen is equal to $1 - \text{FREE_AREA_FRACTION}$. The pressure drop across the screen is given by

$$\Delta p = l \left(\frac{\mu}{K} u + \rho \frac{Y}{\sqrt{K}} u^2 \right) \quad (15.5)$$

where l is the screen thickness (equal to the wire diameter), μ is the viscosity of the gas, u is the velocity normal to the screen, ρ is the density of the gas, and Y and K are empirical constants given by

$$K = 3.44 \times 10^{-9} \text{ FREE_AREA_FRACTION}^{1.6} \text{ m}^2 \quad (15.6)$$

$$Y = 0.043 \text{ FREE_AREA_FRACTION}^{2.13} \quad (15.7)$$

15.4.9 Electrical Cable Failure

Petra Andersson and Patrick Van Hees of the Swedish National Testing and Research Institute (SP) have proposed that the thermally-induced electrical failure (THIEF) of a cable can be predicted via a simple one-dimensional heat transfer calculation, under the assumption that the cable can be treated as a homogeneous cylinder [?]. Their results for PVC cables were encouraging and suggested that the simplification of the analysis is reasonable and that it should extend to other types of cables. The assumptions underlying the THIEF model are as follows:

1. The heat penetration into a cable of circular cross section is largely in the radial direction. This greatly simplifies the analysis, and it is also conservative because it is assumed that the cable is completely surrounded by the heat source.
2. The cable is homogeneous in composition. In reality, a cable is constructed of several different types of polymeric materials, cellulosic fillers, and a conducting metal, most often copper.
3. The thermal properties – conductivity, specific heat, and density – of the assumed homogeneous cable are independent of temperature. In reality, both the thermal conductivity and specific heat of polymers are temperature-dependent, but this information is very difficult to obtain from manufacturers.
4. It is assumed that no decomposition reactions occur within the cable during its heating, and ignition and burning are not considered in the model. In fact, thermoplastic cables melt, thermosets form a char layer, and both off-gas volatiles up to and beyond the point of electrical failure.
5. Electrical failure occurs when the temperature just inside the cable jacket reaches an experimentally determined value.

Obviously, there are considerable assumptions inherent in the Andersson and Van Hees THIEF model, but their results for various polyvinyl chloride (PVC) cables suggested that it may be sufficient for engineering analyses of a wider variety of cables. The U.S. Nuclear Regulatory Commission sponsored a study of cable failure known as CAROLFIRE [?]. The primary project objective of CAROLFIRE was to characterize

the various modes of electrical failure (e.g., hot shorts, shorts to ground) within bundles of power, control and instrument cables. A secondary objective of the project was to develop a simple model to predict thermally-induced electrical failure when a given interior region of the cable reaches an empirically determined threshold temperature. The measurements used for these purposes are described in Volume II of the CAROLFIRE test report. Volume III describes the modeling.

The THIEF model can only predict the temperature profile within the cable as a function of time, given a time-dependent exposing temperature or heat flux. The model does not predict at what temperature the cable fails electrically. This information is gathered from experiment. The CAROLFIRE experimental program included bench-scale, single cable experiments in which temperature measurements were made on the surface of, and at various points within, cables subjected to a uniform heat flux. These experiments provided the link between internal cable temperature and electrical failure. The model can only predict the interior temperature and infer electrical failure when a given temperature is reached. It is presumed that the temperature of the centermost point in the cable is not necessarily the indicator of electrical failure. This analysis method uses the temperature just inside the cable jacket rather than the centermost temperature, as that is where electrical shorts in a multi-conductor cable are most likely to occur first.

To use the THIEF model in FDS, add lines similar to the following to the input file:

```
&MATL ID='plastic', DENSITY=2535., CONDUCTIVITY=0.2, SPECIFIC_HEAT=1.5 /
&SURF ID='cylinder', THICKNESS=0.00815, LENGTH=0.1, MATL_ID='plastic',
    GEOMETRY='CYLINDRICAL' /
&PART ID='Cable Segment', SURF_ID='cylinder', ORIENTATION(1:3,1)=0.,0.,1.,
    STATIC=.TRUE. /
&INIT ID='Cable', XB=0.01,0.01,0.,0.,0.,0., N_PARTICLES=1, PART_ID='Cable Segment' /
&DEVC ID='Cable Temp', INIT_ID='Cable',
    QUANTITY='INSIDE WALL TEMPERATURE', DEPTH=0.0015 /
```

The THIEF model assumes that the cable plastic material has a thermal conductivity of 0.2 W/(m·K) and a specific heat of 1.5 kJ/(kg·K). If you change these values, you are no longer using the THIEF model. The density is the mass per unit length of the cable divided by its cross sectional area. The THICKNESS is the radius of the cylindrical cable in units of m. The LENGTH, in m, is needed by FDS because it assumes that the cable is a cylindrical segment of a certain length. It has no impact on the simulation, and its value is typically the size of a grid cell. The ORIENTATION tells FDS the direction of the prevailing radiative source. The second argument indicates that there can be more than one ORIENTATION. STATIC=.TRUE. prevents the cable from moving. The INIT line is used to position the cable within the computational domain. The DEVC line records the cable's inner temperature, in this case 1.5 mm below the surface. This is typically the jacket thickness.

15.5 Particle Insertion

There are three ways of introducing droplets or particles into a simulation. The first way is to define a sprinkler or nozzle using a PROP line that includes a PART_ID that specifies the particle or droplet parameters. The second way is to add a PART_ID to a SURF line, in which case particles or droplets will be ejected from that surface. Note that this only works if the surface has a normal velocity pointing into the flow domain. The third way to introduce particles or droplets is via an INIT line that defines a volume within the computational domain in which the particles/droplets are to be introduced initially and/or periodically in time.

It is not unusual to include hundreds of thousands of particles in a simulation. Visualizing all of the particles in Smokeview can sometimes be impractical due to memory limitations. To limit the amount of particles, you can make use of the following parameters on the PART line:

SAMPLING_FACTOR Sampling factor for the output file `CHID.prt5`. This parameter can be used to reduce the size of the particle output file used to animate the simulation. The default value is 1 for **MASSLESS** particles, meaning that every particle or droplet will be shown in Smokeview. The default is 10 for all other types of particles. **MASSLESS** particles are discussed in Section 15.2.

AGE Number of seconds the particle or droplet exists, after which time it is removed from the calculation. This is a useful parameter to use when trying to reduce the number of droplets or particles in a simulation.

15.5.1 Particles Introduced at a Solid Surface

If the particles have mass and are introduced from a solid surface, specify **PARTICLE_MASS_FLUX** on the **SURF** line. The number of particles inserted at each solid cell every **DT_INSERT** seconds is specified by **NPPC** (Number of Particles Per Cell) on the **SURF** line defining the solid surface. The default value of **DT_INSERT** is 0.01 s and **NPPC** is 1. As an example, the following set of input lines:

```
&PART ID='particles', ... /
&SURF ID='SLOT', PART_ID='particles', VEL=-5., PARTICLE_MASS_FLUX=0.1 /
&OBST XB=-0.2,0.2,-0.2,0.2,4.0,4.4, SURF_IDS='INERT','SLOT','INERT' /
```

creates an obstruction that ejects particles out of its sides at a rate of $0.1 \text{ kg}/(\text{m}^2 \cdot \text{s})$ and a velocity of 5 m/s (the minus sign indicates the particles are ejected from the surface). FDS will adjust the mass flux if the obstruction or vent dimensions are changed to conform to the numerical grid. The **IDS** have no meaning other than as identifiers. The surface on which particles are specified must have a non-zero normal velocity directed into the computational domain. This happens automatically if the surface is burning, but must be specified if it is not. There is a simple input file called `particle_flux.fds` that demonstrates how the above input lines can produce a stream of particles from a block. The total mass flux from the block is the product of the **PARTICLE_MASS_FLUX** times the total area of the sides of the block, $0.4 \text{ m} \times 0.4 \text{ m} \times 4$. The expected accumulated mass of particles on the ground after 10 s is expected to be 0.64 kg, as shown in Fig. 15.7.

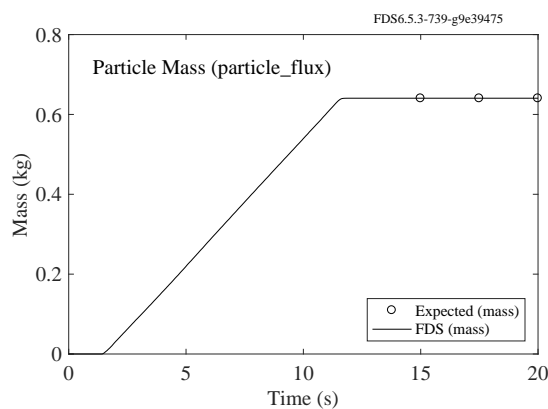


Figure 15.7: Simple test case to demonstrate mass conservation of particles ejected from an obstruction.

Note also that you can independently control particles that emanate from a solid surface. For example, a device might control the activation of a fan, but you can over-ride the device and control the particles

separately. To do this, specify either a device or controller via a `DEVC_ID` or `CTRL_ID` on the `PART` line that defines the particles. For more information on devices and controls, see Sections 17.4 and 17.5.

15.5.2 Particles or Droplets Introduced at a Sprinkler or Nozzle

A sprinkler or nozzle is added to the simulation using a `PROP` line to describe the features of the device and a `DEVC` line to position and orient the device within the computational domain. `PARTICLES_PER_SECOND` is the number of droplets inserted every second per active sprinkler or nozzle (Default 5000). It is listed on the `PROP` line that includes other properties of the sprinkler or nozzle. Note that this parameter only affects sprinklers and nozzles. Changing this parameter does *not* change the flow rate, but rather the number of droplets used to represent the flow.

Note that `PARTICLES_PER_SECOND` can be a very important parameter. In some simulations, it is a good idea to increase this number so that the liquid mass is distributed more uniformly over the droplets. If this parameter is too small, it can lead to a non-physical evaporation pattern, sometimes even to the point of causing a numerical instability. If you encounter a numerical instability shortly after the activation of a sprinkler or nozzle, consider increasing `PARTICLES_PER_SECOND` to produce a smoother evaporation pattern that is more realistic. Keep in mind that for a real sprinkler or nozzle, there are many more droplets created per second than the number that can be simulated.

Note that by default the parameter `PARTICLE_CFL` is set to `.FALSE.` (see Section 6.4.9). Thus, particles inserted with a velocity faster than the local fluid entrainment velocity may traverse more than one cell. If the particles represent a fuel spray or sprinkler droplets that are to be collected in a pan, it may be necessary to set `PARTICLE_CFL=.TRUE.` on `MISC` to precisely account for particle mass. For example, the particles may represent liquid fuel being sprayed into a pan burner made from a zero thickness obstruction. In this case, if the particle position overshoots the cell face representing the boundary of the pan then either spurious burning of the particle will occur underneath the pan or, if the particle does not burn, the heat release rate will be diminished.

15.5.3 Particles or Droplets Introduced within a Volume

Sometimes it is convenient to introduce Lagrangian particles within a particular region of the domain. To do this, use an `INIT` line which contains the `PART_ID` for the type of particle to be inserted. Particles specified via an `INIT` line can represent a number of different kinds of subgrid-scale objects. The particles can be massless tracers or they can be solid or liquid particles with mass. If not massless, specify `MASS_PER_VOLUME` in units of kg/m^3 . Do not confuse this parameter with `DENSITY`, explained in the next section. For example, water has a `DENSITY` of 1000 kg/m^3 , whereas a liter of water broken up into droplets and spread over a cubic meter has a `MASS_PER_VOLUME` of 1 kg/m^3 . The number of Lagrangian particles inserted is controlled by the parameter `N_PARTICLES`.

Randomly Distributed Particles within a Specified Volume

The parameter `N_PARTICLES` on the `INIT` line indicates the number of particles to insert within a specified region of the domain. This region can take on a number of shapes, depending on the parameter `SHAPE`. By default, the region is a rectangular solid designated with the real sextuplet `XB`. The format for `XB` is the same as that used on the `OBST` line. Alternatively, you can specify `SHAPE='CONE'`, in which case the particles will be randomly distributed within a vertical cone. This is primarily used for representing trees. The dimensions of the cone are specified via the parameters `RADIUS`, `HEIGHT`, and base position `XYZ`. The latter is a triplet of real numbers designating the point at the center of the base of the cone. Examples of typical `INIT` lines are:

```
&INIT PART_ID='droplets', XB=..., N_PARTICLES=..., MASS_PER_VOLUME=... /
&INIT PART_ID='leaves', XYZ=..., RADIUS=..., HEIGHT=..., SHAPE='CONE',
    N_PARTICLES=..., MASS_PER_VOLUME=... /
```

Note that the volume of the specified region is calculated according to the `SHAPE` dimensions, regardless of whether there are solid obstructions within this region. Note also that in most applications, the number of particles, `N_PARTICLES`, is somewhat arbitrary but should be chosen to provide at least a few particles per grid cell. FDS will then automatically assign a weighting factor to each particle to ensure that the `MASS_PER_VOLUME` is achieved. In some applications, on the other hand, it may be important to specify the number of particles. For example, if using particles to model the burning of electrical cables, you may want to specify how many cables are actually burning.

If the volume specified by the sextuplet `XB` crosses mesh boundaries, be aware that `N_PARTICLES` refers to the entire volume, not just the volume within a particular mesh. FDS will automatically compute the necessary number of particles to assign to each mesh.

Specifying a Fixed Number of Particles per Grid Cell

There are special applications where you might want to specify `N_PARTICLES_PER_CELL` to indicate the number of particles within each grid cell of a specified region. When using `N_PARTICLES_PER_CELL`, the particles will be randomly placed within each cell. If you set `CELL_CENTERED=.TRUE.`, the particles will be placed at the center of each cell.

Specifying a Weight Factor for Particles

Use `PARTICLE_WEIGHT_FACTOR` to specify how many actual particles each of the computational particles represent. This can be used in conjunction with `N_PARTICLES_PER_CELL` to reduce the computational cost when a large number of identical particles would be placed in the same grid cell.

Single Particle Insertion

If you introduce only a single particle, which is often a handy way of creating a target, you may use the real triplet `XYZ` rather than `XB` to designate the particle's position. You can give this single particle an initial velocity using the real triplet `UVW`. You can also add `DX`, `DY`, and/or `DZ` to create a line of particles that are offset from `XYZ` by these increments in units of meters. For example,

```
&INIT PART_ID='target', XYZ=1.2,3.4,5.6, N_PARTICLES=10, DX=0.1 /
```

creates a line of 10 particles starting at the point (1.2,3.4,5.6) separated by 0.1 m. This is handy for creating arrays of devices, like heat flux gauges. See Section 18.10.5 for more details.

In special cases, you might want a single liquid droplet to be inserted at a particular point with a particular velocity every `DT_INSERT` s following the activation of a particular device, as follows:

```
&INIT N_PARTICLES=1, XYZ=..., UVW=..., DIAMETER=200., DT_INSERT=0.05,
    PART_ID='drops', DEVC_ID='nozzle' /
```

Note that the `DIAMETER` (μm) on the `INIT` line is only valid for liquid droplets. It over-rides the `DIAMETER` on the `PART` line labelled 'drops'. A simple test case that demonstrates this functionality is called `bucket_test_3`, in which water droplets are launched in different directions from a common point. Their

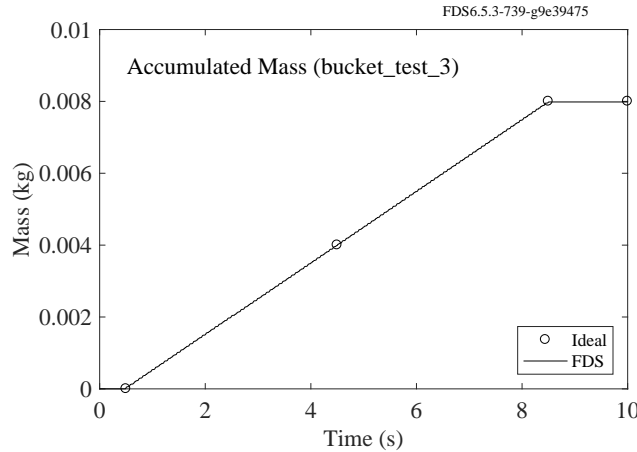


Figure 15.8: Accumulated water collected at the floor in the `bucket_test_3` case.

size, velocity, insertion frequency, and mass flux are varied, and a check is made that water mass is conserved (see Fig. 15.8).

Periodic Insertion of Particles within a Specified Volume

If you want to introduce particles within a given region periodically in time and not just initially, set `DT_INSERT` on the `INIT` line to a positive value indicating the time increment (s) for insertion. The parameter `N_PARTICLES` now indicates the number of droplets/particles inserted every `DT_INSERT` seconds. If the droplets/particles have mass, use `MASS_PER_TIME` (kg/s) instead of `MASS_PER_VOLUME` to indicate how much mass is to be introduced per second.

If you want to delay the insertion of droplets, you can use either a `DEVC_ID` or a `CTRL_ID` on the `INIT` line to name the controlling device. See Section 17.4 for more information on controlling devices.

15.6 Special Topic: Suppression by Water

Modeling fire suppression by water has three principal components: transporting the water droplets through the air, tracking the water along the solid surface, and predicting the reduction of the burning rate. This section addresses the latter two.

15.6.1 Velocity on Solid Surfaces

When a droplet strikes a solid surface⁴, it sticks and is reassigned a new speed and direction. If the surface is horizontal, the direction is randomly chosen. If vertical, the direction is downwards. The rate at which the droplets move over the horizontal and vertical surfaces is difficult to quantify. The parameters `HORIZONTAL_VELOCITY` and `VERTICAL_VELOCITY` on the `PART` line allow you to control the rate at which droplets move horizontally or vertically (downward). The defaults are 0.2 m/s and 0.5 m/s, respectively.

There are some applications, like the suppression of racked storage commodities, where it is useful to allow water droplets to move horizontally along the underside of a solid object. It is difficult to model this

⁴If you do not want droplets to accumulate on solid surfaces, set `ALLOW_SURFACE_PARTICLES=.FALSE.` on the `MISC` line. It is normally `.TRUE.`

phenomenon precisely because it involves surface tension, surface porosity and absorption, and complicated geometry. However, a way to capture some of the effect is to set `ALLOW_UNDERSIDE_PARTICLES=.TRUE.` on the `MISC` line. It is normally false. Also, note that when droplets hit obstructions, the vertical direction is assumed to coincide with the z axis, regardless of any change to the gravity vector, `GVEC`.

A useful sample case to demonstrate various features of droplet motion on solid obstructions is the test case called `cascade.fds`. Figure 15.9 shows a stream of water droplets impinging on the top of a box followed by the cascading of water droplets over the top edge.

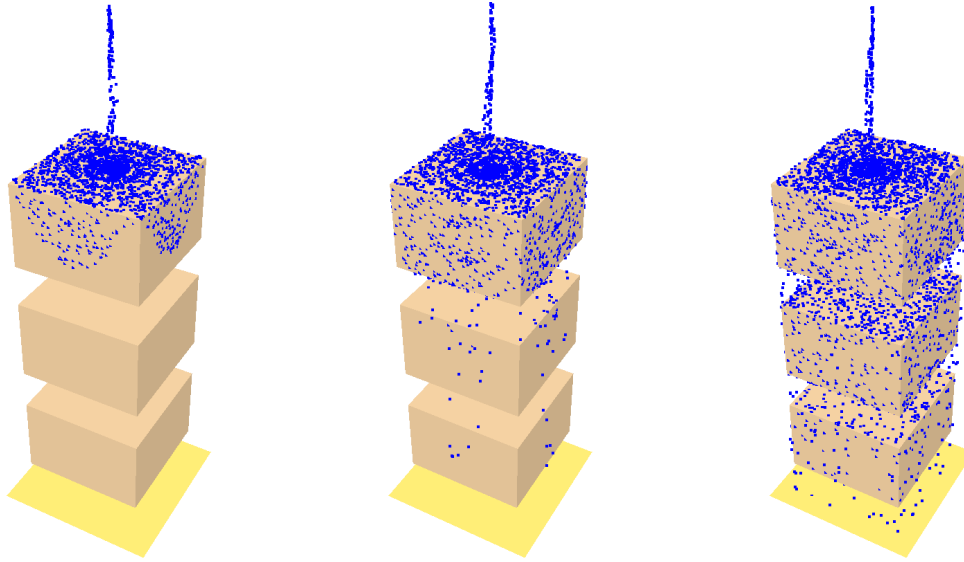


Figure 15.9: Smokeview rendering of the `cascade` test case.

15.6.2 Reduction of the Burning Rate

Water reduces the fuel pyrolysis rate by cooling the fuel surface and also changing the chemical reactions that liberate fuel gases from the solid. If the solid or liquid fuel has been given reaction parameters via the `MATL` line, there is no need to set any additional suppression parameters. It is assumed that water impinging on the fuel surface takes energy away from the pyrolysis process and thereby reduces the burning rate of the fuel. If the surface has been assigned a `HRRPUA` (Heat Release Rate Per Unit Area), a parameter needs to be specified that governs the suppression of the fire by water because this type of simulated fire essentially acts like a gas burner whose flow rate is explicitly specified. An empirical way to account for fire suppression by water is to characterize the reduction of the pyrolysis rate in terms of an exponential function. The local mass loss rate of the fuel is expressed in the form

$$\dot{m}''_f(t) = \dot{m}''_{f,0}(t) e^{-\int k(t) dt} \quad (15.8)$$

Here $\dot{m}''_{f,0}(t)$ is the user-specified burning rate per unit area when no water is applied and k is a function of the local water mass per unit area, m''_w , expressed in units of kg/m^2 .

$$k(t) = E_COEFFICIENT \dot{m}''_w(t) \quad 1/s \quad (15.9)$$

The parameter `E_COEFFICIENT` must be obtained experimentally, and it is expressed in units of $\text{m}^2/(\text{kg} \cdot \text{s})$. Usually, this type of suppression algorithm is invoked when the fuel is complicated, like a cartoned commodity. The example case `e_coefficient` demonstrates the use of this parameter. A sprinkler is placed over a burner defined with an `E_COEFFICIENT` as shown below. The sprinkler is set to operate at 5 s. Figure 15.10 shows the heat release rate and burning rate. Note that expected value is computed using the sprinkler flow rate and the FDS results are delayed slightly by the time it takes for droplets to reach and accumulate on the burning surface.

```
&SURF ID='FUEL', HRRPUA=100., E_COEFFICIENT=4. /
```

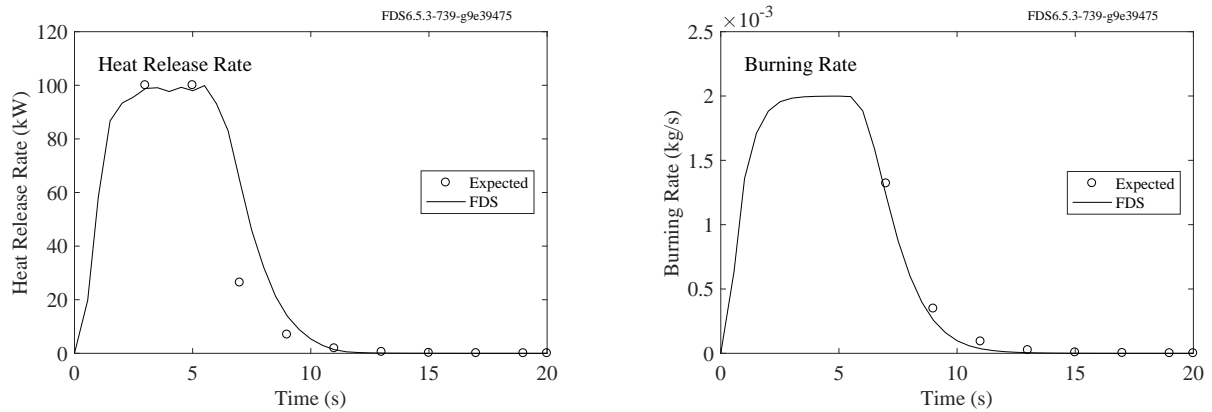


Figure 15.10: Output of the `e_coefficient` test case.

Chapter 16

Vegetation

Vegetation can be modeled using the particle functionality in FDS (for raised vegetation) or as a surface (for vegetation on the ground). In both cases, the vegetation is present as a source of momentum drag and, when undergoing thermal degradation, a source of mass (e.g., water and fuel vapor).

16.1 Using Particles to Represent Vegetation

Four namelist groups are needed to specify the properties and the location of vegetation using particles: SURF, MATL, PART, and INIT. A cone-shape tree crown comprised of needles (i.e., no roundwood) can be defined by the following namelist lines.

The SURF and MATL namelists specify the properties of the needles. In this example, the SURF namelist specifies that the mass of a needle is 70% solid and 30% moisture. The MATL ID='solid fuel' namelist contains the properties of the needle's solid fuel and the MATL ID='moisture' contains the properties of the needle's moisture. As a given mass of solid fuel undergoes pyrolysis, 74% (NU_SPEC = 0.74) is converted to fuel gas (pyrolyzate) and 26% (NU_MATL=0.26) remains as a solid in the form of char. The exothermic process of char oxidation is not modeled. Instead, char remains present as a solid with material properties listed in MATL ID='char'.

```
&SURF ID = 'needles'
  MATL_ID(1,1:2) = 'solid fuel','moisture'
  MATL_MASS_FRACTION(1,1:2) = 0.7,0.3
  THICKNESS = 0.00025
  LENGTH = 0.05
  GEOMETRY = 'CYLINDRICAL' /
```

```
&MATL ID = 'solid fuel'
  DENSITY = 514.
  CONDUCTIVITY = 2.0
  SPECIFIC_HEAT = 1.2
  SPEC_ID = 'pyrolyzate'
  NU_SPEC = 0.74
  NU_MATL = 0.26
  N_REACTIONS = 1
  REFERENCE_TEMPERATURE = 200.
  REFERENCE_RATE = 0.0005
  HEATING_RATE = 1.6
  MATL_ID = 'char'
  HEAT_OF_COMBUSTION = 17700.
```

```

HEAT_OF_REACTION = 418.
ALLOW_SHRINKING=.FALSE.
ALLOW_SWELLING=.FALSE. /

&MATL ID = 'char'
  DENSITY = 300.
  CONDUCTIVITY = 2.0
  SPECIFIC_HEAT= 1.2 /

&MATL ID = 'moisture'
  DENSITY = 1000.
  CONDUCTIVITY = 2.0
  SPECIFIC_HEAT= 4.184
  N_REACTIONS = 1
  REFERENCE_TEMPERATURE = 100.
  REFERENCE_RATE = 0.002
  HEATING_RATE = 1.6
  NU_SPEC = 1.0
  SPEC_ID = 'water vapor'
  HEAT_OF_REACTION= 2500. /

```

The **PART** namelist references **SURF** for material properties and specifies the drag model. The **INIT** namelist defines the geometry of the volume occupied by the vegetation (in this case a cone) and total mass (dry and moist) of the vegetation in that volume (**MASS_PER_VOLUME**).

```

&PART ID='tree crown foliage',
  DRAG_LAW='CYLINDER',
  SURF_ID='needles',
  QUANTITIES='PARTICLE TEMPERATURE','PARTICLE MASS','PARTICLE DIAMETER',
  STATIC=.TRUE.,
  COLOR='FOREST GREEN' /

&INIT PART_ID='tree crown foliage',
  XYZ=0.0,0.0,0.0,
  RADIUS=1,
  HEIGHT=2,
  SHAPE='CONE',
  N_PARTICLES_PER_CELL=1,
  CELL_CENTERED=.TRUE.,
  MASS_PER_VOLUME=2.0 /

```

16.2 Conversion of Vegetation Parameters Commonly Used in Forestry to those Required by FDS

Users of FDS that work in wildland fire or forestry will be unfamiliar with some of the vegetation material properties in FDS input files. Specifically, the thickness and length of needles or roundwood, for a given vegetation type, are not routinely reported (the surface-to-volume ratio is reported instead). Also, while the bulk density is reported (or can be easily found from vegetation loading and height), it is based on the dry mass. Moisture is reported on a dry weight basis. Thus, pending the creation of new namelists or the addition of new namelist parameters, commonly reported vegetation properties cannot be directly used in

FDS input files. Instead, they must be converted to the parameters needed by FDS. These conversions affect parameters in the `SURF` and `INIT` namelists and are given below.

The forestry literature provides vegetation moisture, M_{dw} , on a dry weight basis. The `SURF` namelist parameters for mass fractions of the solid fuel and moisture are then $1/(1 + M_{dw})$ and $M_{dw}/(1 + M_{dw})$, respectively. In addition, the surface-to-volume ratio of the vegetation, σ_v , is reported instead of the physical dimensions. If we assume a cylindrical shape, with the length of the cylinder being much larger than the radius, then $THICKNESS = 2/\sigma_v$ in the `SURF` namelist.

The `INIT` namelist parameter $MASS_PER_VOLUME = \rho_d(1 + M_{dw})$ where ρ_d is the bulk density of the dry vegetation.

Chapter 17

Devices and Control Logic

Sprinklers, smoke detectors, heat flux gauges, and thermocouples may seem to be completely unrelated, but from the point of view of FDS, they are simply devices that operate in specific ways depending on the properties assigned to them. They can be used to record some quantity of the simulated environment, like a thermocouple, or they can represent a mathematical model of a complex sensor, like a smoke detector, and in some cases they can trigger events to happen, like a timer.

All devices, in the broadest sense of the word, are designated via the namelist group `DEVC`. In addition, advanced functionality and properties are accommodated via additional namelist groups called `CTRL` (Control) and `PROP` (Properties).

17.1 Device Location and Orientation: The `DEVC` Namelist Group (Table 19.5)

Regardless of the specific properties, each device needs to be sited either at a point within the computational domain, or over a span of the domain, like a beam smoke detector. For example, a sprinkler is sited within the domain with a line like:

```
&DEVC XYZ=3.0,5.6,2.3, PROP_ID='Acme Sprinkler 123', ID='Spk_39' /
```

The physical coordinates of the device are given by a triplet of real numbers, `XYZ`. FDS uses these coordinates to determine in which gas or wall cell the device is located. The properties of the device are contained on the `PROP` line designated by `PROP_ID`, which will be explained below for each of the special devices included in FDS. The character string `ID` is merely a descriptor to identify the device in the output files, and if any action is tied to its activation.

Not all devices need to be associated with a particular set of properties via the `PROP_ID`. For example, pointwise output quantities are specified with a single `DEVC` line, like

```
&DEVC ID='TC-23', XYZ=3.0,5.6,2.3, QUANTITY='TEMPERATURE' /
```

which tells FDS to record the temperature at the given point as a function of time. The `ID` is a label in the output file whose name is `CHID_devc.csv`. Note that FDS outputs the data stored for that cell without performing any interpolation with surrounding cells.

Some devices have a particular orientation. The parameter `IOR` (Index of Orientation) is required for any device that is placed on the surface of a solid. The values ± 1 or ± 2 or ± 3 indicate the direction that the device “points.” For example, `IOR=-1` means that the device is mounted on a wall that faces in the

negative x direction. `ORIENTATION` is used for devices that are not on a surface and require a directional specification, like a sprinkler. `ORIENTATION` is specified with a triplet of real number values that indicate the components of the direction vector. The default value of `ORIENTATION` is (0,0,-1). For example, a default downward-directed sprinkler spray can be redirected in other direction. If you were to specify

```
&DEVC XYZ=3.0,5.6,2.3, PROP_ID='...', ID='...', ORIENTATION=0.707,0.707,0.0 /
```

the sprinkler would point in the direction halfway between the positive x and y directions. For other devices, the `ORIENTATION` would only change the way the device is drawn by Smokeview.

The delivered density to the floor from a sprinkler depends upon where the sprinkler arms are located. Rather than redefining the spray pattern for every possible direction that the sprinkler can be attached to the pipe, the `DEVC` can be given the parameter `ROTATION`. The default `ROTATION` is 0 degrees, which for a downwards pointing sprinkler is the positive x -axis. Positive `ROTATION` will rotate the 0 degree point towards the positive y -axis.

17.2 Device Output

Each device has a `QUANTITY` associated with it. The time history of each `DEVC` quantity is output to a comma-delimited ASCII file called `CHID_devc.csv` (see Section 22.3 for output file format). This file can be imported into most spreadsheet software packages. Most spreadsheet programs limit the number of columns to some number (for example the 2003 version Microsoft Excel had a 256 column limit). As a default, FDS places no limit on the amount of columns in a comma-separated value (.csv) file. If your spreadsheet application allows fewer columns than the number of `DEVC` or `CTRL` in your input file then set `COLUMN_DUMP_LIMIT` equal to `.TRUE.` on the `DUMP` line. Use `DEVC_COLUMN_LIMIT` and `CTRL_COLUMN_LIMIT` to indicate the limit of columns in the device and control output files. Their default values are 254.

By default, the `DEVC` output is written to a file every `DT_DEVC` seconds. This time increment is specified on the `DUMP` line. Also, by default, a time-averaged value is written out for each quantity of interest. To prevent FDS from time-averaging the `DEVC` output, add `TIME_AVERAGED=.FALSE.` to the `DEVC` line.

A useful option for the `DEVC` line is to add `RELATIVE=.TRUE.`, which will indicate that only the change in the initial value of the `QUANTITY` is to be output. This can be useful for verification and validation studies.

You can change the values of the output `QUANTITY` by multiplying by `CONVERSION_FACTOR` and changing the character string `UNITS`. You can also change the scaling of the coordinates by applying a `COORD_FACTOR`. This is handy for plotting nondimensional distance, for example.

If you do not want the `DEVC QUANTITY` to be included in the output file, set `OUTPUT=.FALSE.` on the `DEVC` line. Sometimes, devices are just used as clocks or control devices. In these cases, you might want to prevent its output from cluttering the output file. If the `DEVC QUANTITY='TIME'`, then `OUTPUT` is set to `.FALSE.` automatically.

All devices must have a specified `QUANTITY`. Some special devices (Section 17.3) have their `QUANTITY` specified on a `PROP` line. A `QUANTITY` specified on a `PROP` line associated with a `DEVC` line will override a `QUANTITY` specified on the `DEVC` line.

17.3 Special Device Properties: The PROP Namelist Group (Table 19.20)

Many devices are fairly easy to describe, like a point measurement, with only a few parameters which can be included on the DEVC line. However, for more complicated devices, it is inconvenient to list all of the properties on each and every DEVC line. For example, a simulation might include hundreds of sprinklers, but it is tedious to list the properties of the sprinkler each time the sprinkler is sited. For these devices, use a separate namelist group called PROP to store the relevant parameters. Each PROP line is identified by a unique ID, and invoked by a DEVC line by the string PROP_ID. The best way to describe the PROP group is to list the various special devices and their properties.

17.3.1 Sprinklers

Here is a very simple example of a sprinkler:

```
&PROP ID='K-11', QUANTITY='SPRINKLER LINK TEMPERATURE', RTI=148., C_FACTOR=0.7,  
      ACTIVATION_TEMPERATURE=74., OFFSET=0.10,PART_ID='water drops', FLOW_RATE=189.3,  
      PARTICLE_VELOCITY=10., SPRAY_ANGLE=30.,80. /  
&DEVC ID='Spr_60', XYZ=22.88,19.76,7.46, PROP_ID='K-11' /
```

A sprinkler, known as 'Spr_60', is located at a point in space given by XYZ. It is a 'K-11' type sprinkler, whose properties are given on the PROP line. Note that the various names (IDs) mean nothing to FDS, except as a means of associating one thing with another, so try to use IDs that are meaningful. The parameter QUANTITY='SPRINKLER LINK TEMPERATURE' *does* have a specific meaning to FDS, directing it to compute the activation of the device using the standard RTI (Response Time Index [?]) algorithm. Properties associated with sprinklers included in the PROP group are:

RTI Response Time Index in units of $(\text{m}\cdot\text{s})^{1/2}$. (Default 100.)

C_FACTOR Conduction Factor in units of $(\text{m/s})^{1/2}$. (Default 0.)

ACTIVATION_TEMPERATURE in units of $^{\circ}\text{C}$. (Default 74°C)

INITIAL_TEMPERATURE of the link in units of $^{\circ}\text{C}$. (Default TMPA)

FLOW_RATE or MASS_FLOW_RATE in units of L/min or kg/s. An alternative is to provide the K_FACTOR in units of $\text{L}/(\text{min}\cdot\text{bar}^{1/2})$ and the OPERATING_PRESSURE, the gauge pressure at the sprinkler, in units of bar. The flow rate is then given by $K\sqrt{p}$. Note that 1 bar is equivalent to 14.5 psi, 1 gpm is equivalent to 3.785 L/min, 1 gpm/psi $^{1/2}$ is equivalent to 14.41 L/min/bar $^{1/2}$. If MASS_FLOW_RATE is given then PARTICLE_VELOCITY must also be defined. Note that FLOW_RATE is only appropriate for liquid droplets; solid particles should use MASS_FLOW_RATE

OFFSET Radius (m) of a sphere surrounding the sprinkler where the water droplets are initially placed in the simulation. It is assumed that beyond the OFFSET the droplets have completely broken up and are transported independently of each other. (Default 0.05 m)

PARTICLE_VELOCITY Initial droplet velocity. (Default 0 m/s)

ORIFICE_DIAMETER Diameter of the nozzle orifice in m (default 0 m). This input provides an alternative way to set droplet velocity by giving values for FLOW_RATE and ORIFICE_DIAMETER, in which case the droplet velocity is computed by dividing the flow rate by the orifice area. Use this method if you do not have any information about droplet velocity. However, quite often you must fine-tune the PARTICLE_VELOCITY in order to reproduce a particular spray profile. The ORIFICE_DIAMETER is not used if either PARTICLE_VELOCITY or SPRAY_PATTERN_TABLE is specified.

SPRAY_ANGLE A pair of angles (in degrees) through which the droplets are sprayed. The angles outline a conical spray pattern relative to the south pole of the sphere centered at the sprinkler with radius **OFFSET**. For example, **SPRAY_ANGLE=30., 80.** directs the water droplets to leave the sprinkler through a band between 60° and 10° south latitude, assuming the orientation of the sprinkler is (0,0,-1), the default. Elliptical spray patterns can be defined by giving a pair of spray angles. For example, **SPRAY_ANGLE(1,1:2)=0., 60.** and **SPRAY_ANGLE(2,1:2)=0., 30.**, defines a spray pattern with 60 degree angle in the direction of *x* axis and a 30 degree angle in the direction of *y* axis. **SPRAY_PATTERN_SHAPE** determines how the droplets are distributed within the specified **SPRAY_ANGLE**. Choices are 'UNIFORM' for uniform distribution and 'GAUSSIAN'. The default distribution is 'GAUSSIAN'. The parameter **SPRAY_PATTERN_MU** controls the latitude of the maximum density of droplets for the 'GAUSSIAN' distribution. The width of the distribution is controlled by the parameter **SPRAY_PATTERN_BETA**.

SPRAY_PATTERN_TABLE Name of a set of **TABL** lines containing the description of the spray pattern.

PART_ID The name of the **PART** line containing properties of the droplets. See Chapter 15 for additional details.

PRESSURE_RAMP The name of the **RAMP** lines specifying the dependence of pipe pressure on the number of active sprinklers and nozzles.

Be aware that sprinklers can produce many droplets. To limit the computational cost, liquid droplets disappear when they hit the “floor” of the computational domain, regardless of whether it is solid or not. This feature mimics the presence of floor drains. To stop FDS from removing liquid droplets from the floor of the domain, add the phrase **POROUS_FLOOR=.FALSE.** to the **MISC** line. Be aware, however, that droplets that land on the floor continue to move horizontally in randomly selected directions; bouncing off obstructions, and consuming CPU time. Note also that solid particles do not disappear from the floor the domain like liquid droplets.

For more information about sprinklers, their activation and spray dynamics, read the FDS Technical Reference Guide [?].

Special Topic: Specifying Complex Spray Patterns

As an example of the more advanced sprinkler options, a sprinkler with an elliptical spray pattern and uniform mass flux distribution within the spray angle is given by:

```
&PROP ID='K-11', QUANTITY='SPRINKLER LINK TEMPERATURE', RTI=148., C_FACTOR=0.7,
      ACTIVATION_TEMPERATURE=74., OFFSET=0.10, PART_ID='water drops', FLOW_RATE=189.3,
      PARTICLE_VELOCITY=10., SPRAY_ANGLE(1:2,1)=0., 60., SPRAY_ANGLE(1:2,2) = 0., 30.,
      SPRAY_PATTERN_SHAPE='UNIFORM' /
&DEVC ID='Spr_60', XYZ=22.88, 19.76, 7.46, PROP_ID='K-11' /
```

For full-cone sprays, the parameter **SPRAY_PATTERN_MU** is set to zero by default. For hollow-cone sprays it is set to the average of **SPRAY_ANGLE(1:2,1)**, the spray angle in *x* direction. The following example uses **SPRAY_PATTERN_MU** to define a spray that is somewhere between full-cone and hollow-cone spray:

```
&PROP ID='K-11', QUANTITY='SPRINKLER LINK TEMPERATURE', RTI=148., C_FACTOR=0.7,
      ACTIVATION_TEMPERATURE=74., OFFSET=0.10, PART_ID='water drops', FLOW_RATE=189.3,
      PARTICLE_VELOCITY=10., SPRAY_ANGLE=0., 30., SPRAY_PATTERN_MU=15./
&DEVC ID='Spr_60', XYZ=22.88, 19.76, 7.46, PROP_ID='K-11' /
```

If a more complex spray pattern is desired than one characterized by a `SPRAY_ANGLE`, then a `SPRAY_PATTERN_TABLE` can be specified using the `TABL` namelist group. Specify the total flow using `FLOW_RATE` on the `PROP` line, the name of the spray pattern using `SPRAY_PATTERN_TABLE` and then one or more `TABL` lines of the form:

```
&TABL ID='table_id', TABLE_DATA=LAT1,LAT2,LON1,LON2,VELO,FRAC /
```

where each `TABL` line for a given 'table_id' provides information about the spherical distribution of the spray pattern for a specified solid angle. `LAT1` and `LAT2` are the bounds of the solid angle measured in degrees from the south pole (0 is the south pole and 90 is the equator, 180 is the north pole). Note that this is not the conventional way of specifying a latitude, but rather a convenient system based on the fact that a typical sprinkler sprays water downwards, which is why 0 degrees is assigned to the “south pole,” or the $-z$ direction. The parameters `LON1` and `LON2` are the bounds of the solid angle (also in degrees), where 0 (or 360) is aligned with the $-x$ axis and 90 is aligned with the $-y$ axis. `VELO` is the velocity (m/s) of the droplets at their point of insertion. `FRAC` the fraction of the total flow rate of liquid that should emerge from that particular solid angle.

In the test case called `bucket_test_2`, the spray pattern is defined as two jets, each with a velocity of 5 m/s and a total flow rate of 60 L/min. The sprinkler is set to operate for only 5 s. The first jet contains 0.2 of the total flow, the second, 0.8 of the total. The jets are centered at points 30° below the “equator,” and are separated by 180° .

```
&PROP ID='K-11', QUANTITY='SPRINKLER LINK TEMPERATURE', OFFSET=0.10,  
PART_ID='water_drops', FLOW_RATE=60., SPRAY_PATTERN_TABLE='TABLE1',  
SMOKEVIEW_ID='sprinkler_upright', PARTICLE_VELOCITY=10. /  
&TABL ID='TABLE1',TABLE_DATA=30,31,0,1,5,0.2/  
&TABL ID='TABLE1',TABLE_DATA=30,31,179,180,5,0.8/
```

Note that each set of `TABL` lines must have a unique `ID`. Also note that the `TABL` lines can be specified in any order. Figure 17.1 verifies that the sprinkler releases 5 kg of water (1 kg/s for 5 s).

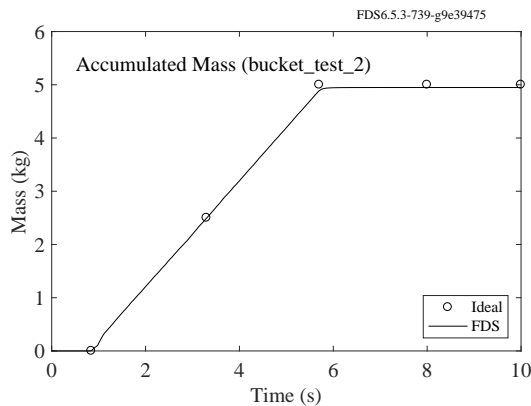


Figure 17.1: Accumulated water collected at the floor in the `bucket_test_2` case.

Special Topic: Varying Pipe Pressure

In real sprinkler systems, the pipe pressure is affected by the number of actuated sprinklers. Typically, the pressure is higher than the design value when the first sprinkler activates, and decreases as more and

more sprinklers are activated. The pipe pressure has an effect on flow rate, droplet velocity and droplet size distribution. In FDS, the varying pipe pressure can be specified on a PROP line using PRESSURE_RAMP. On each RAMP line, the number of open sprinklers or nozzles is associated with certain pipe pressure (bar). For example:

```
&PROP ID='My nozzle'
      OFFSET=0.1
      PART_ID='water drops'
      FLOW_RATE=0.9
      OPERATING_PRESSURE = 10.0
      PARTICLE_VELOCITY=15.0
      SPRAY_ANGLE=0.0,80.0
      PRESSURE_RAMP = 'PR1' /

&RAMP ID = 'PR1' T = 1, F = 16. /
&RAMP ID = 'PR1' T = 2, F = 10. /
&RAMP ID = 'PR1' T = 3, F = 8. /
```

These lines would indicate that the pressure is 16 bar when the first sprinkler activates, 10 bar when two sprinklers are active, and 8 bar when three or more sprinklers are active. When counting the number of active sprinklers, FDS accounts for all active sprinklers or nozzles with a given PART_ID.

When pressure ramps are used, both FLOW_RATE and PARTICLE_VELOCITY are dependent on the OPERATING_PRESSURE. Specify either the FLOW_RATE, or the K_FACTOR and OPERATING_PRESSURE. In the latter case, the flow rate is given by $K\sqrt{p}$ and the droplet velocity by using the liquid density and the ORIFICE_DIAMETER. If spray pattern table is used, the droplet velocity is determined separately for each line of the table by applying $K\sqrt{p}$ and the ORIFICE_DIAMETER. The median diameter of the particle size distribution is scaled as $d_m(p) = d_m(p_o)(p_o/p)^{1/3}$, where p_o is the OPERATING_PRESSURE and $d_m(p_o)$ is specified by parameter DIAMETER on the corresponding PART line.

For some simulations there may be groups of independent sprinklers or nozzles. For example one might have one set of nozzles for a fuel spray and a second set for water spray. In this case the flow of water would not be impacted by how many fuel spray nozzles are open. To have the PRESSURE_RAMP only count a subset of sprinklers or nozzles, the keyword PIPE_INDEX can be used on the DEVC line. For example:

```
&DEVC ID='Spr_1', XYZ=2.00,2.00,8.00, PROP_ID='My nozzle', PIPE_INDEX=1 /
&DEVC ID='Spr_2', XYZ=1.00,1.00,8.00, PROP_ID='My nozzle', PIPE_INDEX=1 /
&DEVC ID='Fuel_1', XYZ=2.00,2.00,1.00, PROP_ID='Fuel Spray', PIPE_INDEX=2 /
&DEVC ID='Fuel_2', XYZ=1.00,1.00,1.00, PROP_ID='Fuel Spray', PIPE_INDEX=2 /
```

These lines indicate that the fuel spray nozzles are a separate pipe network from the water sprinklers. With these inputs, a PRESSURE_RAMP for the water sprinklers would not count any active fuel spray nozzles. See the example case flow_rate_2 in the Verification Guide for further details on the use of PIPE_INDEX.

17.3.2 Nozzles

Nozzles are very much like sprinklers, only they do not activate based on the standard RTI (Response Time Index) model. To simulate a nozzle that activates at a given time, specify a QUANTITY and SETPOINT directly on the DEVC line. The following lines:

```
&DEVC XYZ=23.91,21.28,0.50, PROP_ID='nozzle', ORIENTATION=0,0,1, QUANTITY='TIME',
      SETPOINT=0., ID='noz_1' /
&DEVC XYZ=26.91,21.28,0.50, PROP_ID='nozzle', ORIENTATION=0,0,1, QUANTITY='TIME',
      SETPOINT=5., ID='noz_2' /
```

```
&PROP ID='nozzle', PART_ID='heptane drops', FLOW_RATE=2.132,
      FLOW_TAU=-50., PARTICLE_VELOCITY=5., SPRAY_ANGLE=0.,45. /
```

designate two nozzles of the same type, one which activates at time zero, the other at 5 s. Note that nozzles must have a designated `PROP_ID`, and the `PROP` line must have a designated `PART_ID` to describe the liquid droplets.

Example Case: Setting the Flow Rate of a Nozzle

This example demonstrates the use of pressure ramps and control logic for opening and closing nozzles. It also serves as a verification test for the water flow rate. There are four nozzles that open at designated times: 0 s, 15 s, 30 s and 45 s. At time 60 s, all the nozzles are closed. The number of open nozzles is measured using a device with quantity '`OPEN NOZZLES`'. A comparison of the FDS result and the exact, intended values is shown in Fig. 17.2. Note that '`OPEN NOZZLES`' counts only nozzles belonging to the specified `PIPE_INDEX`. The pressure ramp has been designed to deliver a total flow rate of 10 L/min at all values of open nozzles. Mathematically this means that

$$nK\sqrt{p} = 10 \text{ L/min} \Rightarrow p = \left(\frac{10 \text{ L/min}}{nK} \right)^2 \quad (17.1)$$

where n is the number of open nozzles. The corresponding nozzle and pressure ramp definitions are

```
&PROP ID='Head', OFFSET=0.10, PART_ID='water drops', K_FACTOR=2.5,
      OPERATING_PRESSURE=1.,
      PRESSURE_RAMP='PR', PARTICLE_VELOCITY=2., SPRAY_ANGLE= 0.,60. /

&RAMP ID='PR', T= 1., F=16. /
&RAMP ID='PR', T= 2., F=4. /
&RAMP ID='PR', T= 3., F=1.778 /
&RAMP ID='PR', T= 4., F=1. /
```

The water is tracked using a device measuring the accumulated mass per unit area, integrated over the total floor area. The total mass of water should increase from zero to 10 kg in 60 s. A comparison of the FDS prediction and this analytical result is shown in Fig. 17.2. The slight delay of the FDS result is caused by the time it takes from the droplets to fall down on the floor.

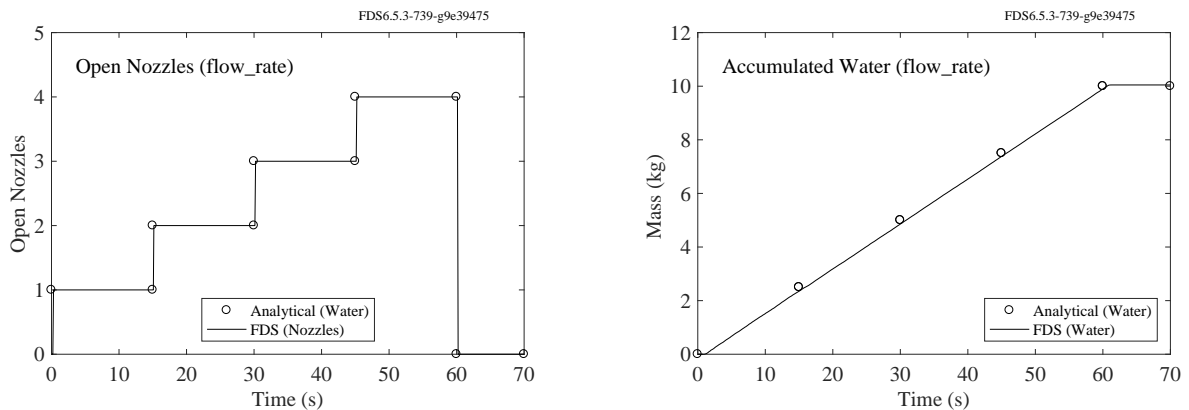


Figure 17.2: Output of the `flow_rate` test case.

17.3.3 Special Topic: Specified Entrainment (Velocity Patch)

The details of the sprinkler head geometry and spray atomization are practically impossible to resolve in a fire calculation. As a result, the local gas phase entrainment by the sprinkler is difficult to predict. As an alternative, it is possible to specify the local gas velocity in the vicinity of the sprinkler nozzle. The `PROP` line may be used to specify a polynomial function for a specific velocity component and this function may be “patched” into the flow field using a device. This device is given the quantity ‘`VELOCITY PATCH`’ and is initially inactive. The velocity patch must be activated with a separate control device, as discussed in Section 17.4. You specify the local region for the velocity patch using `XB` for the device. The polynomial is defined as a second-order Taylor expansion about the point `XYZ` (the default value of `XYZ` is the center of `XB`). FDS then uses an immersed boundary method to force the local velocity component to satisfy the polynomial. The polynomial is specified by the coefficients `P0`, `PX (1:3)`, and `PXX (1:3, 1:3)`, which represent, respectively, the value of the k th velocity component, the first derivatives, and the second derivatives at point `XYZ`. Note that the first derivatives are represented by a three component array and the second derivatives are represented by a symmetric 3×3 array—only the upper triangular part needs to be specified. The polynomial is given by (note that summation of repeated indices is implied):

$$u_k(\mathbf{r}) = \underbrace{(u_k)_0}_{P0} + r_i \underbrace{\left(\frac{\partial u_k}{\partial x_i}\right)_0}_{PX(1:3)} + \frac{r_i r_j}{2} \underbrace{\left(\frac{\partial^2 u_k}{\partial x_i \partial x_j}\right)_0}_{PXX(1:3, 1:3)} \quad (17.2)$$

The vector \mathbf{r} is the position of the velocity storage location relative to the point `XYZ`. The specific velocity component is specified on `PROP` by the integer `VELOCITY_COMPONENT`. Below we provide an example set of `PROP` and `DEVC` lines to specify a parabolic profile for the vertical component of velocity.

```
&PROP ID='p1', VELOCITY_COMPONENT=3, P0=-1,PXX(1,1)=5,PXX(2,2)=5 /
&DEVC XB=-.1,.1,-.1,.1,.9,.95, QUANTITY='VELOCITY PATCH',PROP_ID='p1', DEVC_ID='t1' /
&DEVC ID='t1', XYZ=0,0,.9, QUANTITY='TIME', SETPOINT=10/
```

In this example, a velocity patch is activated at 10 s in the simulation. Any w components of velocity with staggered storage locations within the box `XB=-.1,.1,-.1,.1,.9,.95` will be driven toward the value specified by the polynomial profile ‘`p1`’. You must ensure that the device box encompasses the staggered storage locations (see the theory manual [?] for a discussion on the face-centered velocity storage locations).

17.3.4 Heat Detectors

`QUANTITY='LINK TEMPERATURE'` defines a heat detector, which uses essentially the same activation algorithm as a sprinkler, without the water spray.

```
&DEVC ID='HD_66', PROP_ID='Acme Heat', XYZ=2.3,4.6,3.4 /
&PROP ID='Acme Heat', QUANTITY='LINK TEMPERATURE', RTI=132.,
ACTIVATION_TEMPERATURE=74. /
```

Like a sprinkler, `RTI` is the Response Time Index in units of $\sqrt{\text{m} \cdot \text{s}}$. `ACTIVATION_TEMPERATURE` is the link activation temperature in degrees C (Default 74 °C). `INITIAL_TEMPERATURE` is the initial temperature of the link in units of °C (Default `TMPE`).

17.3.5 Smoke Detectors

A smoke detector is defined in the input file with an entry similar to:

```
&DEVC ID='SD_29', PROP_ID='Acme Smoke Detector', XYZ=2.3,4.6,3.4 /
&PROP ID='Acme Smoke Detector', QUANTITY='CHAMBER OBSCURATION', LENGTH=1.8,
ACTIVATION_OBSCURATION=3.24 /
```

for the single parameter Heskestad model. Note that a PROP line is mandatory for a smoke detector, in which case the DEVC QUANTITY can be specified on the PROP line. For the four parameter Cleary model, use a PROP line like:

```
&PROP ID='Acme Smoke Detector I2', QUANTITY='CHAMBER OBSCURATION',
ALPHA_E=1.8, BETA_E=-1.1, ALPHA_C=1.0, BETA_C=-0.8,
ACTIVATION_OBSCURATION=3.24 /
```

where the two characteristic filling or “lag” times are of the form:

$$\delta t_e = \alpha_e u^{\beta_e} \quad ; \quad \delta t_c = \alpha_c u^{\beta_c} \quad (17.3)$$

The default detector parameters are for the Heskestad model with a characteristic LENGTH of 1.8 m. For the Cleary model, the ALPHAS and BETAS must all be listed explicitly. Suggested constants for unidentified ionization and photoelectric detectors presented in Table 17.1. ACTIVATION_OBSCURATION is the threshold value in units of %/m. The threshold can be set according to the setting commonly provided by the manufacturer. The default setting¹ is 3.24 %/m (1 %/ft).

Table 17.1: Suggested values for smoke detector model [?]. See Ref. [?] for others.

Detector	α_e	β_e	α_c, L	β_c
Cleary Ionization I1	2.5	-0.7	0.8	-0.9
Cleary Ionization I2	1.8	-1.1	1.0	-0.8
Cleary Photoelectric P1	1.8	-1.0	1.0	-0.8
Cleary Photoelectric P2	1.8	-0.8	0.8	-0.8
Heskestad Ionization	—	—	1.8	—

Defining Smoke

By default, FDS assumes that the smoke from a fire is generated in direct proportion to the heat release rate. A value of SOOT_YIELD=0.01 on the REAC line means that the smoke generation rate is 0.01 of the fuel burning rate. The “smoke” in this case is not explicitly tracked by FDS, but rather is assumed to be a function of the combustion products lumped species.

Suppose, however, that you want to define your own “smoke” and that you want to specify its production rate independently of the HRR (or even in lieu of an actual fire, like a smoldering source). You might also want to define a mass extinction coefficient for the smoke and an assumed molecular weight (as it will be tracked like a gas species). Finally, you also want to visualize the smoke using the SMOKE3D feature in Smokeview. Use the following lines:

¹Note that the conversion of obscuration from units of %/ft to %/m is given by:

$$O[\%/m] = \left[1 - \left(1 - \frac{O[\%/ft]}{100} \right)^{3.28} \right] \times 100 \quad (17.4)$$

```

&SPEC ID='MY SMOKE', MW=29., MASS_EXTINCTION_COEFFICIENT=8700. /
&SURF ID='SMOLDER', TMP_FRONT=1000., MASS_FLUX(1)=0.0001, SPEC_ID='MY SMOKE',
    COLOR='RED' /
&VENT XB=0.6,1.0,0.3,0.7,0.0,0.0, SURF_ID='SMOLDER' /

&PROP ID='Acme Smoke', QUANTITY='CHAMBER OBSCURATION', SPEC_ID='MY SMOKE' /
&DEVC XYZ=1.00,0.50,0.95, PROP_ID='Acme Smoke', ID='smoke_1' /

&DUMP SMOKE3D_QUANTITY='MASS FRACTION', SMOKE3D_SPEC_ID='MY SMOKE' /

```

The same smoke detector model is used that was described above. Only now, the mass fraction of your species 'MY SMOKE' is used in the algorithm, rather than that associated with the lumped species. Note that your species will not participate in the radiation calculation. It will merely serve as a surrogate for smoke. Note also that if you specify explicitly a smoke surrogate, you should set `SOOT_YIELD=0` on the `REAC` line to prevent FDS from including smoke as a component of the combustion product lumped species.

17.3.6 Beam Detection Systems

A beam detector can be defined by specifying the endpoints, $(x1, y1, z1)$ and $(x2, y2, z2)$, of the beam and the total percent obscuration at which the detector activates. The two endpoints must lie in the same mesh. FDS determines which mesh cells lie along the linear path defined by the two endpoints. The beam detector response is evaluated as

$$\text{Obscuration} = \left(1 - \exp \left(-K_m \sum_{i=1}^N \rho_{s,i} \Delta x_i \right) \right) \times 100 \% \quad (17.5)$$

where i is a mesh cell along the path of the beam, $\rho_{s,i}$ is the soot density of the mesh cell, Δx_i is the distance within the mesh cell that is traversed by the beam, and K_m is the mass extinction coefficient. The line in the input file has the form:

```

&DEVC XB=x1,x2,y1,y2,z1,z2, QUANTITY='PATH OBSCURATION', ID='beam1', SETPOINT=33.0 /

```

A similar `QUANTITY` is 'TRANSMISSION' which is given by the following expression:

$$\text{Transmission} = \exp \left(-K_m \frac{L_0}{L} \sum_{i=1}^N \rho_{s,i} \Delta x_i \right) \times 100 \% / m \quad (17.6)$$

Note that the transmission is given in units of $\%/m$ rather than $\%$ like obscuration. L is the total path length of the beam, and L_0 is the reference dimension of 1 m.

Since a single linear path cannot span more than one mesh, having a beam detector that crosses multiple meshes will require post processing. Break the beam detector path into multiple `DEVC` lines, one for each mesh that the beam crosses. The total obscuration is given by

$$O = \left[1 - \prod_{i=1}^N (1 - O_i / 100) \right] \times 100 \% \quad (17.7)$$

where O_i is the FDS output for the beam detector of the i th path (note that the bracketed term contains a product rather than a sum).

Example Case: A Beam Detector

A 10 m by 10 m by 4 m compartment is filled with smoke from burning propane, represented as 0.006 kg/kg of the lumped species variable, `PRODUCTS`. The soot yield is specified as 0.01 kg/kg, resulting in a uniform soot density of 71.9 mg/m³. Using the default mass extinction coefficient of 8700 m²/kg, the optical depth is calculated to be 0.626 1/m. The compartment has a series of obstructions located at increasing distance from the front in increments of 1 m. The correlation for the output quantity `VISIBILITY`, Eq. (18.6), produces a visibility distance of 4.8 m. When viewing the smoke levels with Smokeview, you should just barely see the fifth obstacle which is at a distance of 5 m from the front of the compartment. If this is the case, Smokeview is properly displaying the obscuration of the smoke. Three beam detectors are also placed in the compartment. These all have a path length of 10 m, but are at different orientations within the compartment. Using the optical depth of 0.626 1/m and the path length of 10 m, the expected total obscuration is 99.81 %, which is the result computed by FDS for each of the three detectors.

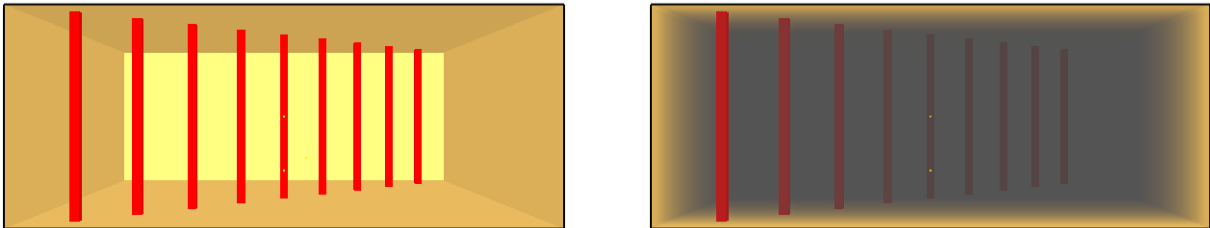


Figure 17.3: Output of the `beam_detector` test case.

Special Topic: Beam Detectors Spanning Multiple Meshes

The data from a device are processed and stored on the mesh in which the device is spatially located; therefore you cannot specify a single beam detector which spans multiple meshes. To model a single beam detector which is required to span multiple meshes, specify one beam on each mesh spanned and use a combination of secondary `DEVCs` and `CTRLs` to sum the obscuration from each of the constituent beams. For example, to model `beam_1`, which is required to span two meshes, specify two constituent beams (`beam_1a` and `beam_1b`) and sum their obscuration to output the total obscuration of `beam_1`:

```
&DEVC ID='beam_1a', XB=..., QUANTITY='PATH OBSCURATION', OUTPUT=.FALSE. /
&DEVC ID='beam_1b', XB=..., QUANTITY='PATH OBSCURATION', OUTPUT=.FALSE. /
&CTRL ID='beam_1a_s', FUNCTION_TYPE='SUBTRACT', INPUT_ID='CONSTANT','beam_1a',
    CONSTANT=100. /
&CTRL ID='beam_1b_s', FUNCTION_TYPE='SUBTRACT', INPUT_ID='CONSTANT','beam_1b',
    CONSTANT=100. /
&CTRL ID='beam_1_m', FUNCTION_TYPE='MULTIPLY',
    INPUT_ID='CONSTANT','beam_1a_s','beam_1b_s', CONSTANT=1E-2 /
&CTRL ID='beam_1_f', FUNCTION_TYPE='SUBTRACT', INPUT_ID='CONSTANT','beam_1_m',
    CONSTANT=100. /
&DEVC ID='beam_1', QUANTITY='CONTROL VALUE', CTRL_ID='beam_1_f', XYZ=... /
```

The outputs of the constituent beams are suppressed and the obscuration of `beam_1` will be in percent. The constant on the line:

```
&CTRL ID='beam_1_m', FUNCTION_TYPE='MULTIPLY',
      INPUT_ID='CONSTANT', 'beam_1a_s', 'beam_1b_s', CONSTANT=1E-2 /
```

needs to be modified to suit the number of constituent beams. To maintain correct decimal placement, the value of the negative exponent should be equal to the number of constituent beams multiplied by 2, minus 2.

17.3.7 Aspiration Detection Systems

An aspiration detection system groups together a series of smoke measurement devices. An aspiration system consists of a sampling pipe network that draws air from a series of locations to a central point where an obscuration measurement is made. To define such a system in FDS, you must provide the sampling locations, sampling flow rates, the transport time from each sampling location, and if an alarm output is desired, the overall obscuration “setpoint.” One or more DEVC inputs are used to specify details of the sampling locations, and one additional input is used to specify the central detector:

```
&DEVC XYZ=..., QUANTITY='DENSITY', SPEC_ID='SOOT', ID='soot1', DEVC_ID='asp1',
      FLOWRATE=0.1, DELAY=20 /
&DEVC XYZ=..., QUANTITY='DENSITY', SPEC_ID='SOOT', ID='soot2', DEVC_ID='asp1',
      FLOWRATE=0.2, DELAY=10 /
...
&DEVC XYZ=..., QUANTITY='DENSITY', SPEC_ID='SOOT', ID='sootN', DEVC_ID='asp1',
      FLOWRATE=0.3, DELAY=30 /
&DEVC XYZ=..., QUANTITY='ASPIRATION', ID='asp1', BYPASS_FLOWRATE=0.4,
      SETPOINT=0.02 /
```

where the DEVC_ID is used at each sampling point to reference the central detector, FLOWRATE is the gas flow rate in kg/s, DELAY is the transport time (in seconds) from the sampling location to the central detector, BYPASS_FLOWRATE is the flow rate in kg/s of any air drawn into the system from outside the computational domain (accounts for portions of the sampling network lying outside the domain defined by the MESH inputs), and SETPOINT is the alarm threshold obscuration in units of %/m. The output of the aspiration system is computed as

$$\text{Obscuration} = \left(1 - \exp \left(-K_m \frac{\sum_{i=1}^N \rho_{s,i}(t - t_{d,i}) \dot{m}_i}{\sum_{i=1}^N \dot{m}_i} \right) \right) \times 100 \text{ \%}/\text{m} \quad (17.8)$$

where \dot{m}_i is the mass FLOWRATE at sampling location i , $\rho_{s,i}(t - t_{d,i})$ is the soot density at sampling location i , $t_{d,i}$ s prior (DELAY) to the current time t , and K_m is the MASS_EXTINCTION_COEFFICIENT associated with visible light.

Example Case: aspiration_detector

A cubical compartment, 2 m on a side has a three sampling location aspiration system. The three locations have equal flow rates of 0.3 kg/s, and transport times of 50, 100, and 150 s, respectively. No bypass flow rate is specified for the aspiration detector. Combustion products are forced into the bottom of the compartment at a rate of 1 kg/s. The SOOT_YIELD=0.001. Mass is removed from the top of the compartment at a rate of 1 kg/s. The aspiration detector shows an increasing obscuration over time. There is a delay of slightly over 50 s in the initial increase which results from the 50 s transport time for the first sampling location plus a short period of time to transport the combustion products to the sampling location. The detector response has three plateaus that result from the delay times of the sampling locations. The sampling points are co-located, so each plateau represents an additional one third of the soot being transported to the detector. The

soot density at the sampling point is $7.1 \times 10^{-5} \text{ kg/m}^3$. Using this value the plateaus are computed as 18 %, 33.2 %, and 45.7 %, as seen in Fig. 17.4.

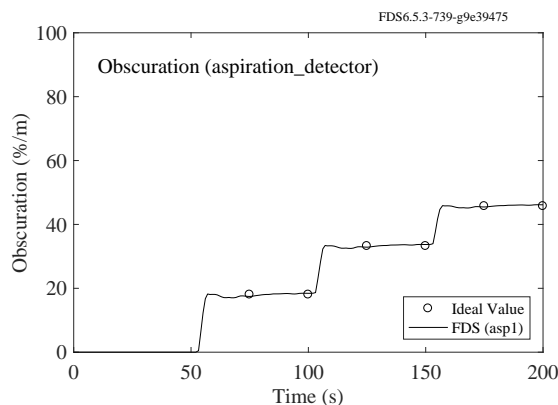


Figure 17.4: Output of `aspiration_detector` test case.

17.4 Basic Control Logic

Devices can be used to control various actions, like creating and removing obstructions, or activating and deactivating fans and vents. Every device has an associated `QUANTITY`, whether it is included directly on the `DEVC` line or indirectly on the optional `PROP` line. Using the `DEVC` parameter `SETPOINT`, you can trigger an action to occur when the `QUANTITY` value passes above, or below, the given `SETPOINT`. The following parameters dictate how a device will control something:

SETPOINT The value of the device at which its state changes. For a detection type of device (e.g., heat or smoke) this value is taken from the device's `PROP` inputs and need not be specified on the `DEVC` line.

TRIP_DIRECTION A positive integer means the device will change from its `INITIAL_STATE` when the value of the device is greater than the `SETPOINT` and be equal to the `INITIAL_STATE` when the value is less than the `SETPOINT`. A negative integer has the opposite behavior. The device will change from its `INITIAL_STATE` when the value of the device is less than the `SETPOINT` and be equal to the `INITIAL_STATE` when the value is greater than the `SETPOINT`. The default value is +1.

LATCH If this logical value is set to `.TRUE.` the device will only change state once. The default value is `.TRUE..`

INITIAL_STATE This logical value is the initial state of the device. The default value is `.FALSE.` For example, if an obstruction associated with the device is to disappear, set `INITIAL_STATE=.TRUE.`

If you desire to control FDS using more complex logic than can be provided by the use of a single device and its setpoint, control functions can be specified using the `CTRL` input. See Section 17.5 for more on `CTRL` functions. The simplest example of a device is just a timer:

```
&DEVC XYZ=1.2,3.4,5.6, ID='my clock', QUANTITY='TIME', SETPOINT=30. /
```

Anything associated with the device via the parameter, `DEVC_ID='my clock'`, will change its state at 30 s. For example, if the text were added to an `OBST` line, that obstruction would change from its `INITIAL_STATE` of `.FALSE.` to `.TRUE.` after 30 s. In other words, it would be created at 30 s instead of at the start of the simulation. This is a simple way to open a door or window.

When using a `DEVC` output to control `FDS`, the instantaneous value of the `DEVC` is used. For some `QUANTITY` types, such as `TEMPERATURE`, this output can be very noisy. To prevent a spurious spike from causing a state change of the `DEVC` you can specify the parameter `SMOOTHING_FACTOR`. This is a parameter that can vary between 0 and 1. It performs an exponential smoothing of the `DEVC` output as follows:

$$\bar{x}^n = \bar{x}^{n-1} \text{ SMOOTHING_FACTOR} + x^n (1 - \text{SMOOTHING_FACTOR}) \quad (17.9)$$

where n is the time step, x is the instantaneous device output and \bar{x} is the smoothed output. The `SMOOTHING_FACTOR` defaults to 0 which means no smoothing is performed. Note that `SMOOTHING_FACTOR` only changes the value passed to control functions; it has no effect on the value of the `DEVC` written to the `CHID_devic.csv` file.

17.4.1 Creating and Removing Obstructions

In many fire scenarios, the opening or closing of a door or window can lead to dramatic changes in the course of the fire. Sometimes these actions are taken intentionally, sometimes as a result of the fire. Within the framework of an `FDS` calculation, these actions are represented by the creation or removal of solid obstacles, or the opening or closing of exterior vents.

Remove or create a solid obstruction by assigning the character string `DEVC_ID` to indicate the name of a `DEVC ID` on the `OBST` line that is to be created or removed. This will direct `FDS` to remove or create the obstruction when the device changes state to `.FALSE.` or `.TRUE.`, respectively. For example, the lines

```
&OBST XB=..., DEVC_ID='det2' /
&DEVC XYZ=..., ID='det2', INITIAL_STATE=.TRUE. /
```

will cause the given obstruction to be removed when the specified `DEVC` changes state.

Creation or removal at a predetermined time can be performed using a `DEVC` that has `TIME` as its measured quantity. For example, the following instructions will cause the specified `HOLES` and `OBST`structions to appear/disappear at the various designated times. These lines are part of the simple test case called `create_remove.fds`.

```
&OBST XB=0.3,0.4,0.1,0.9,0.1,0.9, COLOR='PURPLE' /
&HOLE XB=0.2,0.4,0.2,0.3,0.2,0.3, COLOR='RED', DEVC_ID='timer1' /
&HOLE XB=0.2,0.4,0.7,0.8,0.7,0.8, COLOR='GREEN', DEVC_ID='timer2' /
&OBST XB=0.7,0.8,0.2,0.3,0.2,0.3, COLOR='BLUE', DEVC_ID='timer3' /
&OBST XB=0.7,0.8,0.6,0.7,0.6,0.7, COLOR='PINK', DEVC_ID='timer4' /
&OBST XB=0.5,1.0,0.0,1.0,0.0,0.1, COLOR='YELLOW', DEVC_ID='timer5' /
&HOLE XB=0.7,0.8,0.7,0.8,0.0,0.1, COLOR='BLACK', DEVC_ID='timer6' /
&HOLE XB=0.7,0.8,0.2,0.3,0.0,0.1, COLOR='GRAY 50', DEVC_ID='timer7' /

&DEVC XYZ=..., ID='timer1', SETPOINT=1., QUANTITY='TIME', INITIAL_STATE=.FALSE. /
&DEVC XYZ=..., ID='timer2', SETPOINT=2., QUANTITY='TIME', INITIAL_STATE=.TRUE. /
&DEVC XYZ=..., ID='timer3', SETPOINT=3., QUANTITY='TIME', INITIAL_STATE=.FALSE. /
&DEVC XYZ=..., ID='timer4', SETPOINT=4., QUANTITY='TIME', INITIAL_STATE=.TRUE. /
&DEVC XYZ=..., ID='timer5', SETPOINT=5., QUANTITY='TIME', INITIAL_STATE=.FALSE. /
&DEVC XYZ=..., ID='timer6', SETPOINT=6., QUANTITY='TIME', INITIAL_STATE=.TRUE. /
&DEVC XYZ=..., ID='timer7', SETPOINT=6., QUANTITY='TIME', INITIAL_STATE=.FALSE. /
```

At the start of the simulation, the purple obstruction is present with a red block embedded in it. This red block is actually a `HOLE` whose initial state is `.FALSE.`, i.e., the hole is filled. Also at the start of the simulation, there is a pink obstruction that is visible. At 1 s the red block disappears. At 2 s the empty hole in the purple obstruction is filled with a green block. This hole was initially true, i.e. empty. The blue obstruction appears at 3 s because its initial state is false, meaning that it does not exist initially. The pink obstruction disappears at 4 s because its initial state is true and this state changes at 4 s. At 5 s a yellow obstruction appears with one empty hole and one embedded gray block. At 6 s the gray block disappears because it is a hole that was initially false and therefore was filled with the gray block when its parent obstruction (yellow) was created. Also at 6 s the hole originally present in the yellow obstruction is filled with a black block because it was a hole that was initially empty and then filled when its `DEVC` changed state. *You should always try a simple example first before embarking on a complicated creation/removal scheme for obstructions and holes.*

To learn how to create and remove obstructions multiple times, see Section 17.5.5 for information about the custom control feature.

17.4.2 Activating and Deactivating Vents

When a device or control function is applied to a `VENT`, the purpose is to either activate or deactivate any time ramp associated with the `VENT` via its `DEVC_ID`. For example, to control a fan, do the following:

```
&SURF ID='FAN', VOLUME_FLOW=5. /
&VENT XB=..., SURF_ID='FAN', DEVC_ID='det2' /
&DEVC ID='det2', XYZ=..., QUANTITY='TIME', SETPOINT=30., INITIAL_STATE=.FALSE. /
```

Note that at 30 s, the “state” of the ‘FAN’ changes from `.FALSE.` to `.TRUE.`, or more simply, the ‘FAN’ turns on. Since there is no explicit time function associated with the ‘FAN’, the default 1 s ramp-up will begin at 30 s instead of at 0 s. If `INITIAL_STATE=.TRUE.`, then the fan should turn off at 30 s. Essentially, “activation” of a `VENT` causes all associated time functions to be delayed until the device `SETPOINT` is reached. “Deactivation” of a `VENT` turns off all time functions. Usually this means that the parameters on the `SURF` line are all nullified, so it is a good idea to check the functionality with a simple example.

A ‘MIRROR’ or ‘OPEN’ `VENT` should not be activated or deactivated. You can, however, place an obstruction in front of an ‘OPEN’ `VENT` and then create it or remove it to model the closing or opening of a door or window.

17.5 Advanced Control Functions: The CTRL Namelist Group

There are many systems whose functionality cannot be described by a simple device with a single “setpoint.” Consider for example, a typical HVAC system. It is controlled by a thermostat that is given a temperature setpoint. The system turns on when the temperature goes below the setpoint by some amount and then turns off when the temperature rises above that same setpoint by some amount. This behavior cannot be defined by merely specifying a single setpoint. You must also define the range or “deadband” around the setpoint, and whether an increasing or decreasing temperature activates the system. For the HVAC example, crossing the lower edge of the deadband activates heating; crossing the upper edge activates cooling. These more complicated behaviors can be modeled in FDS using `CTRLs`. The following parameters dictate how a control function will behave:

ID A name for the control function that is unique over all control functions.

FUNCTION_TYPE The type of control function. The possible types are shown in Table 17.2.

INPUT_ID A list of DEVC or CTRL IDs that are the inputs to the control function. Up to forty inputs can be specified. If a DEVC or CTRL is being used as an INPUT_ID for a control function, then it must have a unique ID over both devices and control functions. Additionally, a control function cannot be used as an input for itself.

SETPOINT The value of the control function at which its state changes. This is only appropriate for functions that return numerical values.

TRIP_DIRECTION A positive integer means the control function will change state when its value increases past the setpoint and a negative integer means the control function will change state when its value decreases past the setpoint. The default value is +1.

LATCH If this logical value is set to .TRUE. the control function will only change state once. The default value is .TRUE..

INITIAL_STATE This logical value is the initial state of the control function. The default value is .FALSE. For example, if an obstruction associated with the control function is to disappear, set INITIAL_STATE to .TRUE.

For any object for which a DEVC_ID can be specified (such as OBST or VENT), a CTRL_ID can be specified instead.

Table 17.2: Control function types.

FUNCTION_TYPE	Purpose
ANY	Changes state if <u>any</u> INPUTs are .TRUE.
ALL	Changes state if <u>all</u> INPUTs are .TRUE.
ONLY	Changes state if and <u>only</u> if N INPUTs are .TRUE.
AT_LEAST	Changes state if <u>at least</u> N INPUTs are .TRUE.
TIME_DELAY	Changes state DELAY s after INPUT becomes .TRUE.
CUSTOM	Changes state based on evaluating a RAMP of the function's input
DEADBAND	Behaves like a thermostat
KILL	Terminates code execution if its sole INPUT is .TRUE.
RESTART	Dumps restart files if its sole INPUT is .TRUE.
SUM	Sums the outputs of the INPUTs
SUBTRACT	Subtracts the second INPUT from the first
MULTIPLY	Multiplies the outputs of the INPUTs
DIVIDE	Divides the first INPUT by the second
POWER	The first INPUT to the power of the second
EXP	The exponential of the INPUT
LOG	The natural logarithm of the INPUT
COS	The cosine of the INPUT
SIN	The sine of the INPUT
ACOS	The arccosine of the INPUT
ASIN	The arcsine of the INPUT
PID	A Proportional-Integral-Derivative control function

If you want to design a system of controls and devices that involves multiple changes of state, include the attribute `LATCH=.FALSE.` on the relevant `DEVC` or `CTRL` input lines. By default, devices and controls may only change state once, like a sprinkler activating or smoke detector alarming. `LATCH` is `.TRUE.` by default for both devices and controls.

If you want a `DEVC` to operate based on the logical state of a `CTRL`, set `QUANTITY` equal to `'CONTROL'` and set the `CTRL_ID` on the `DEVC` input to the `ID` of the control function.

The output value of numerical control function is defined by a `DEVC` line with `QUANTITY` set equal to `'CONTROL VALUE'` and `CTRL_ID` set equal to the `ID` of the control function. You can then use `SETPOINT` to have the `DEVC` operate a particular output value of the control function.

17.5.1 Control Functions: ANY, ALL, ONLY, and AT_LEAST

Suppose you want an obstruction to be removed (a door is opened, for example) after any of four smoke detectors in a room has activated. Use input lines of the form:

```
&OBST XB=..., SURF_ID='...', CTRL_ID='SD' /

&DEVC XYZ=1,1,3, PROP_ID='Acme Smoker', ID='SD_1' /
&DEVC XYZ=1,4,3, PROP_ID='Acme Smoker', ID='SD_2' /
&DEVC XYZ=4,1,3, PROP_ID='Acme Smoker', ID='SD_3' /
&DEVC XYZ=4,4,3, PROP_ID='Acme Smoker', ID='SD_4' /
&CTRL ID='SD', FUNCTION_TYPE='ANY', INPUT_ID='SD_1','SD_2','SD_3','SD_4',
    INITIAL_STATE=.TRUE. /
```

The `INITIAL_STATE` of the control function `SD` is `.TRUE.`, meaning that the obstruction exists initially. The “change of state” means that the obstruction is removed when any smoke detector alarms. By default, the `INITIAL_STATE` of the control function `SD` is `.FALSE.`, meaning that the obstruction does not exist initially.

Suppose that now you want the obstruction to be created (a door is closed, for example) after all four smoke detectors in a room have activated. Use a control line of the form:

```
&CTRL ID='SD', FUNCTION_TYPE='ALL', INPUT_ID='SD_1','SD_2','SD_3','SD_4' /
```

The control functions `AT_LEAST` and `ONLY` are generalizations of `ANY` and `ALL`. For example,

```
&CTRL ID='SD', FUNCTION_TYPE='AT_LEAST', N=3, INPUT_ID='SD_1','SD_2','SD_3','SD_4' /
```

changes the state from `.FALSE.` to `.TRUE.` when at least 3 detectors activate. Note that in this example, and the example below, the parameter `N` is used to specify the number of activated devices required for the conditions of the control function to be satisfied. The control function,

```
&CTRL ID='SD', FUNCTION_TYPE='ONLY', N=3, INPUT_ID='SD_1','SD_2','SD_3','SD_4' /
```

changes the state from `.FALSE.` to `.TRUE.` when 3, and only 3, detectors activate.

17.5.2 Control Function: TIME_DELAY

The `TIME_DELAY` control function starts a timer of length `DELAY` when its input changes state. When the timer expires, the `TIME_DELAY` control function will change state. Note, that the timer starts at each change in state of the input; therefore, if the input changes state a second time before the first timer ends, the timer

will get reset. This function enables FDS to model time delays between when a device activates and when some other action occurs, like in a dry pipe sprinkler system.

```
&DEVC XYZ=2,2,3, PROP_ID='Acme Sprinkler_link', QUANTITY='LINK TEMPERATURE',  
      ID='Spk_29_link' /  
&DEVC XYZ=2,2,3, PROP_ID='Acme Sprinkler', QUANTITY='CONTROL', ID='Spk_29',  
      CTRL_ID='dry pipe' /  
&CTRL ID='dry pipe', FUNCTION_TYPE='TIME_DELAY', INPUT_ID='Spk_29_link', DELAY=30. /
```

This relationship between a sprinkler and its pipes means that the sprinkler spray is controlled (in this case delayed) by the 'dry pipe', which adds 30 s to the activation time of Spk_29, measured by Spk_29_link, before water can flow out of the head.

17.5.3 Control Function: DEADBAND

This control function behaves like an HVAC thermostat. It can operate in one of two modes analogous to heating or cooling. The function is provided with an INPUT_ID which is the DEVC whose value is used by the function, an upper and lower SETPOINT, and the mode of operation by ON_BOUND. If ON_BOUND='LOWER', the function changes state from its INITIAL_STATE when the value of the INPUT_ID drops below the lower value in SETPOINT and reverts when it increases past the upper value, i.e., like a heating system. The reverse will occur if ON_BOUND='UPPER', i.e., a cooling system.

For an HVAC system, the following lines of input would set up a simple thermostat:

```
&SURF ID='FAN', TMP_FRONT=40., VOLUME_FLOW=-1. /  
&VENT XB=-0.3,0.3,-0.3,0.3,0.0,0.0, SURF_ID='FAN', CTRL_ID='thermostat' /  
&DEVC ID='TC', XYZ=2.4,5.7,3.6, QUANTITY='TEMPERATURE' /  
&CTRL ID='thermostat', FUNCTION_TYPE='DEADBAND', INPUT_ID='TC',  
      ON_BOUND='LOWER', SETPOINT=23.,27., LATCH=.FALSE./
```

Here, we want to control the VENT that simulates the FAN, which blows hot air into the room. A DEVC called TC is positioned in the room to measure the TEMPERATURE. The thermostat uses a SETPOINT to turn on the FAN when the temperature falls below 23 °C (ON_BOUND='LOWER') and it turns off when the temperature rises above 27 °C.

Note that a deadband controller needs to have LATCH set to .FALSE.

17.5.4 Control Function: RESTART and KILL

There are times when you might only want to run a simulation until some goal is reached, or you might want to create some baseline condition and then run multiple permutations of that baseline. For example, you might want to run a series of simulations where different mitigation strategies are tested once a detector alarms. Using the RESTART control function, you can cause a restart file to be created once a desired condition is met. The simulation can continue and the restart files can be copied to have the job identifying string, CHID, of the various permutations (providing of course that the usual restrictions on the use of restart files are followed). For example, the lines

```
&DEVC ID='temp', QUANTITY='TEMPERATURE', SETPOINT=1000., XYZ=4.5,6.7,3.6 /  
&DEVC ID='velo', QUANTITY='VELOCITY', SETPOINT=10., XYZ=4.5,6.7,3.6 /  
  
&CTRL ID='kill', FUNCTION_TYPE='KILL', INPUT_ID='temp' /  
&CTRL ID='restart', FUNCTION_TYPE='RESTART', INPUT_ID='velo' /
```


will kill the job and output restart files when the temperature at the given point rises above 1000 °C; or just force restart files to be output when the velocity at a given point exceeds 10 m/s.

17.5.5 Control Function: CUSTOM

For most of the control function types, the logical (true/false) output of the devices and control functions and the time they last changed state are taken as inputs. A CUSTOM function uses the numerical output of a DEVC along with a RAMP to determine the output of the function. When the RAMP output for the DEVC value is negative, the CTRL will have the value of its INITIAL_STATE. When the RAMP output for the DEVC value is positive, the CTRL will have the opposite value of its INITIAL_STATE. In the case below, the CUSTOM control function uses the numerical output of a timer device as its input. The function returns true (the default value for INITIAL_STATE is .FALSE.) when the F parameter in the ramp specified with RAMP_ID is a positive value and false when the RAMP F value is negative. In this case, the control would start false and would switch to true when the timer reaches 60 s. It would then stay in a true state until the timer reaches 120 s and would then change back to false.

Note that when using control functions the IDs assigned to both the CTRL and the DEVC inputs must be unique across both sets of inputs, i.e., you cannot use the same ID for both a control function and a device. You can make a fan operate on a fixed cycle by using a CUSTOM control function based on time:

```
&SURF ID='FAN', TMP_FRONT=40., VOLUME_FLOW=-1. /
&VENT XB=-0.3,0.3,-0.3,0.3,0.0,0.0, SURF_ID='FAN', CTRL_ID='cycling timer' /
&DEVC ID='TIMER', XYZ=2.4,5.7,3.6, QUANTITY='TIME' /
&CTRL ID='cycling timer', FUNCTION_TYPE='CUSTOM', INPUT_ID='TIMER', RAMP_ID='cycle' /
&RAMP ID='cycle', T= 59, F=-1 /
&RAMP ID='cycle', T= 61, F= 1 /
&RAMP ID='cycle', T=119, F= 1 /
&RAMP ID='cycle', T=121, F=-1 /
```

In the above example the fan will be off initially, turn on at 60 s and then turn off at 120 s.

You can make an obstruction appear and disappear multiple times by using the following lines

```
&OBST XB=..., SURF_ID='whatever', CTRL_ID='cycling timer' /
&DEVC ID='TIMER', XYZ=..., QUANTITY='TIME' /
&CTRL ID='cycling timer', FUNCTION_TYPE='CUSTOM', INPUT_ID='TIMER', RAMP_ID='cycle' /
&RAMP ID='cycle', T= 0, F=-1 /
&RAMP ID='cycle', T= 59, F=-1 /
&RAMP ID='cycle', T= 61, F= 1 /
&RAMP ID='cycle', T=119, F= 1 /
&RAMP ID='cycle', T=121, F=-1 /
```

The above will have the obstacle initially removed, then added at 60 s, and removed again at 120 s.

Experiment with these combinations using a simple case before trying a case to make sure that FDS indeed is doing what is intended.

17.5.6 Control Function: Math Operations

The control functions that perform simple math operations (SUM, SUBTRACT, MULTIPLY, DIVIDE, and POWER) can have a constant value specified as one of their inputs. This is done by specifying one of the INPUT_IDS as 'CONSTANT' and providing the value using the input CONSTANT. For example, the inputs below represent a control function whose state changes when the square of the velocity exceeds 10 (see Section 17.4 for an explanation of TRIP_DIRECTION).

```
&DEVC ID='SPEED SENSOR', XYZ=..., QUANTITY='VELOCITY' /
&CTRL ID='multiplier', FUNCTION_TYPE='POWER',
      INPUT_ID='SPEED SENSOR','CONSTANT', CONSTANT=2., SETPOINT=10.,
      TRIP_DIRECTION=1 /
```

17.5.7 Control Function: PID Control Function

A PID (Proportional Integral Derivative) control function is a commonly used feedback controller for controlling electrical and mechanical systems. The function computes an error between a process variable and a desired setpoint. The goal of the PID function is to minimize the error. A PID control function is computed as

$$u(t) = K_p e(t) + K_i \int_0^t e(t) dt + K_d \frac{de(t)}{dt} \quad (17.10)$$

where K_p , K_i , and K_d are respectively the `PROPORTIONAL_GAIN`, the `INTEGRAL_GAIN`, and the `DIFFERENTIAL_GAIN`; $e(t)$ is the error given by subtracting the `TARGET_VALUE` from the input; and $u(t)$ is the output.

17.5.8 Combining Control Functions: A Pre-Action Sprinkler System

For a pre-action sprinkler system, the normally dry sprinkler pipes are flooded when a detection event occurs. For this example, the detection event is when two of four smoke detectors alarm. It takes 30 s to flood the piping network. The nozzle is a `DEVC` named 'NOZZLE 1' controlled by the `CTRL` named 'nozzle trigger'. The nozzle activates when both detection and the time delay have occurred. Note that the `DEVC` is specified with `QUANTITY='CONTROL'`.

```
&DEVC XYZ=1,1,3, PROP_ID='Acme Smoker', ID='SD_1' /
&DEVC XYZ=1,4,3, PROP_ID='Acme Smoker', ID='SD_2' /
&DEVC XYZ=4,1,3, PROP_ID='Acme Smoker', ID='SD_3' /
&DEVC XYZ=4,4,3, PROP_ID='Acme Smoker', ID='SD_4' /
&DEVC XYZ=2,2,3, PROP_ID='Acme Nozzle', QUANTITY='CONTROL',
      ID='NOZZLE 1', CTRL_ID='nozzle trigger' /

&CTRL ID='nozzle trigger', FUNCTION_TYPE='ALL', INPUT_ID='smokey','delay' /
&CTRL ID='delay', FUNCTION_TYPE='TIME_DELAY', INPUT_ID='smokey', DELAY=30. /
&CTRL ID='smokey', FUNCTION_TYPE='AT_LEAST', N=2,
      INPUT_ID='SD_1','SD_2','SD_3','SD_4' /
```

Example Case: control_test_2

The `control_test_2` example demonstrates the use of the mathematical and PID control functions. Two compartments are defined with the left hand compartment initialized to 20 °C and the right hand compartment to 10 °C. Control functions are defined to:

- Add the temperatures in the two compartments
- Subtract the right hand compartment temperature from the left hand compartment temperature
- Multiply the left hand temperature by 0.5
- Divide the left hand temperature by the right hand temperature

- Take the square root of the right hand temperature
- Use the time as input to a PID function with a target value of 5 and $K_p=-0.5$, $K_i=0.001$, and $K_d=1$

```
&CTRL ID='Add',FUNCTION_TYPE='SUM',INPUT_ID='LHS Temp','RHS Temp'/
&CTRL ID='Subtract',FUNCTION_TYPE='SUBTRACT',INPUT_ID='RHS Temp','LHS Temp'/
&CTRL ID='Multiply',FUNCTION_TYPE='MULTIPLY',INPUT_ID='LHS
Temp','CONSTANT',CONSTANT=0.5/
&CTRL ID='Divide',FUNCTION_TYPE='DIVIDE',INPUT_ID='LHS Temp','RHS Temp'/
&CTRL ID='Power',FUNCTION_TYPE='POWER',INPUT_ID='RHS Temp','CONSTANT',CONSTANT=0.5/
&CTRL ID='PID',FUNCTION_TYPE='PID',INPUT_ID='Time',TARGET_VALUE=5.,
PROPORTIONAL_GAIN=-0.5,INTEGRAL_GAIN=0.001,DIFFERENTIAL_GAIN=1./
```

Results are shown in Fig. 17.5.

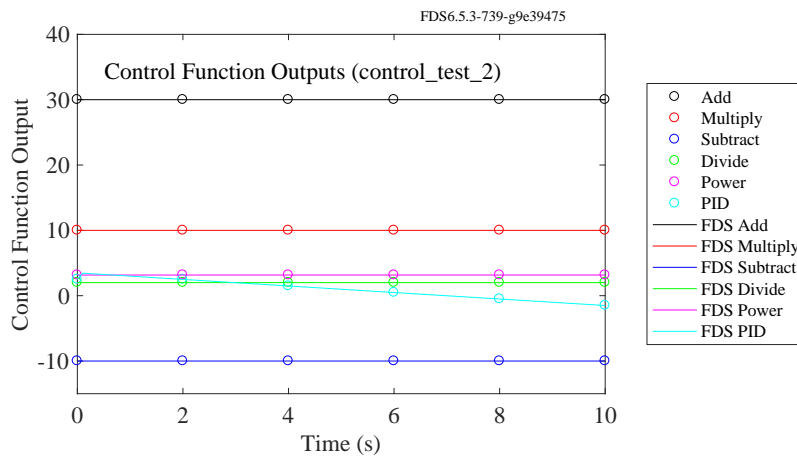


Figure 17.5: Results of the control_test_2 case.

17.5.9 Combining Control Functions: A Dry Pipe Sprinkler System

For a dry-pipe sprinkler system, the normally dry sprinkler pipes are pressurized with gas. When a link activates in a sprinkler head, the pressure drop allows water to flow into the pipe network. For this example it takes 30 s to flood the piping network once a sprinkler link has activated. The sequence of events required for operation is first ANY of the links must activate which starts the 30 s TIME_DELAY. Once the 30 s delay has occurred, each nozzle with an active link, the ALL control functions, will then flow water.

```
&DEVC XYZ=2,2,3, PROP_ID='Acme Sprinkler Link', ID='LINK 1' /
&DEVC XYZ=2,3,3, PROP_ID='Acme Sprinkler Link', ID='LINK 2' /

&PROP ID='Acme Sprinkler Link', QUANTITY='LINK TEMPERATURE',
ACTIVATION_TEMPERATURE=74., RTI=30./

&DEVC XYZ=2,2,3, PROP_ID='Acme Nozzle', QUANTITY='CONTROL',
ID='NOZZLE 1', CTRL_ID='nozzle 1 trigger' /
&DEVC XYZ=2,3,3, PROP_ID='Acme Nozzle', QUANTITY='CONTROL',
ID='NOZZLE 2', CTRL_ID='nozzle 2 trigger' /
```

```
&CTRL ID='check links', FUNCTION_TYPE='ANY', INPUT_ID='LINK 1','LINK 2'/
&CTRL ID='delay', FUNCTION_TYPE='TIME_DELAY', INPUT_ID='check links', DELAY=30. /
&CTRL ID='nozzle 1 trigger', FUNCTION_TYPE='ALL', INPUT_ID='delay','LINK 1'/
&CTRL ID='nozzle 2 trigger', FUNCTION_TYPE='ALL', INPUT_ID='delay','LINK 2'/'
```

17.5.10 Example Case: activate_vents

The simple test case called `activate_vents` demonstrates several of the control functions. Figure 17.6 shows seven differently colored vents that activate at different times, depending on the particular timing or control function.

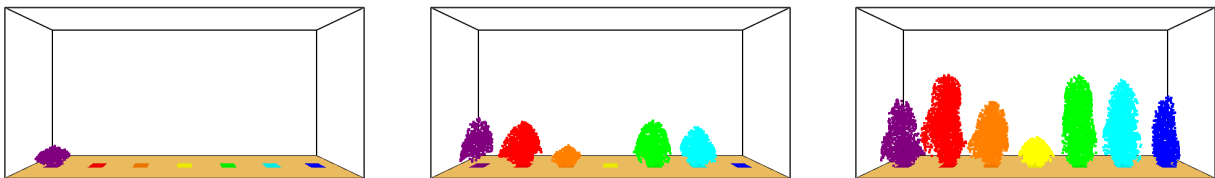


Figure 17.6: Output of the `activate_vents` test case at 5, 10, and 15 s.

17.6 Controlling a RAMP

17.6.1 Changing the Independent variable

For any user-defined `RAMP`, the normal independent variable, for example time for `RAMP_V`, can be replaced by the output of a `DEVC`. This is done by specifying the input `DEVC_ID` on one of the `RAMP` input lines. When this is done, the current output of the `DEVC` is used as the independent variable for the `RAMP`. A `CTRL_ID` can also be specified as long as the control function outputs a numerical value (i.e., is a mathematical function (Section 17.5.6) or a PID function (Section 17.5.7)). In the following example a blower is ramped from 0 % flow at 20 °C, to 50 % flow when the temperature exceeds 100 °C, and to 100 % flow when the temperature exceeds 200 °C. This is similar functionality to the `CUSTOM` control function, but it allows for variable response rather than just on or off.

```
&SURF ID='BLOWER', VEL=-2, RAMP_V='BLOWER RAMP' /
&DEVC XYZ=2,3,3, QUANTITY='TEMPERATURE', ID='TEMP DEVC' /
&RAMP ID='BLOWER RAMP', T= 20,F=0.0, DEVC_ID='TEMP DEVC' /
&RAMP ID='BLOWER RAMP', T=100,F=0.5 /
&RAMP ID='BLOWER RAMP', T=200,F=1.0 /
```

17.6.2 Freezing the Output Value, Example Case: `hrr_freeze`

There are occasions where you may want the value of a `RAMP` to stop updating. For example, if you are simulating a growing fire in a room with sprinklers, you may wish to stop the fire from growing when a sprinkler over the fire activates. This type of action can be accomplished by changing the input of the

RAMP to a DEVC (see the previous section) and then giving that DEVC either a NO_UPDATE_DEVC_ID or a NO_UPDATE_CTRL_ID. When the specified controller changes its state to .TRUE. it will cause the DEVC to stop updating its value. Since the DEVC is being used as the independent variable to a RAMP, the RAMP will have its output remain the same. This is shown in the example below. A fire is given a linear RAMP from 0 to 1000 kW/m² over 50 s. Rather than using the simulation time, the RAMP uses a DEVC for the time. The timer is set to freeze when another DEVC measuring time reaches 200 °C. Figure 17.7 shows the result of these inputs in the test case hrr_freeze where it can be seen that the pyrolysis rate stops increasing once the gas temperature reaches 200 °C.

```
&SURF ID='FIRE', HRRPUA=1000., RAMP_Q='FRAMP', COLOR='ORANGE'/
&RAMP ID='FRAMP', T= 0, F=0, DEVC_ID='FREEZE TIME'/
&RAMP ID='FRAMP', T=50, F=1/
&DEVC XYZ=..., QUANTITY='TEMPERATURE', SETPOINT=200., INITIAL_STATE=.FALSE.,
      ID='TEMP'/
&DEVC XYZ=..., QUANTITY='TIME', NO_UPDATE_DEVC_ID='TEMP', ID='FREEZE TIME'/'
```

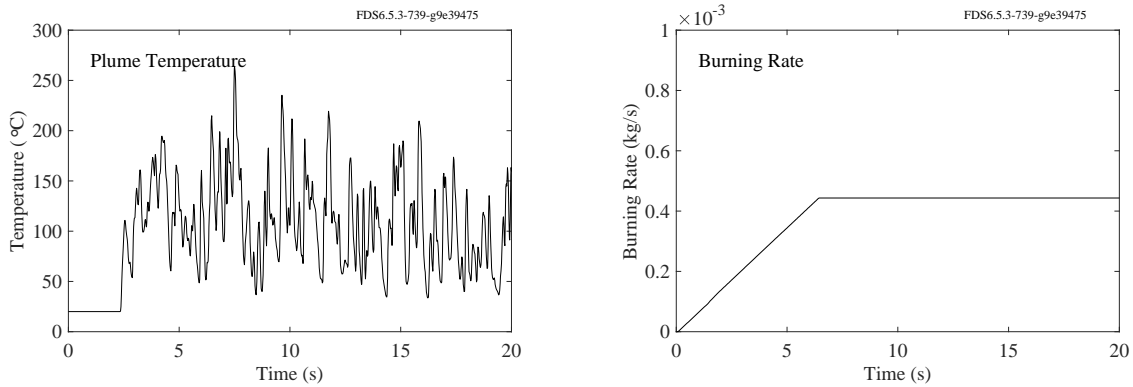


Figure 17.7: Temperature (left) and burning rate (right) outputs of the hrr_freeze test case.

It should be noted that devices are updated sequentially in the order that they are listed in the input file and that devices in different meshes do not share values until the end of a time step. This means that if the device being frozen is on a different mesh or is listed before the device that freezes it, it will not be frozen until the next time step.

17.7 Visualizing FDS Devices in Smokeview

This section provides an overview of various objects that can be drawn by Smokeview and how to customize their appearance. Further technical details may be found in the Smokeview User's Guide [?].

17.7.1 Devices that Indicate Activation

Devices like sprinklers and smoke detectors can be drawn in one of two ways so as to indicate activation. When FDS determines that a device has activated it places a message in the `.smv` file indicating the object number, the activation time and the state (0 for inactive or 1 for active). Smokeview then draws the corresponding object. See Tables 17.3 and 17.4 for images.

The character string, `SMOKEVIEW_ID`, on the `PROP` line associates an FDS device with a Smokeview object. For example, the following lines instruct Smokeview to draw the device in the shape of a 'target':

```
&PROP ID='my target', SMOKEVIEW_ID='target' /  
&DEVC XYZ=0.5,0.8,0.6, QUANTITY='TEMPERATURE', PROP_ID='my target' /
```

Table 17.3: Single frame static objects


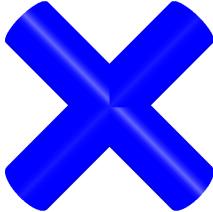
SMOKEVIEW_ID	Image
sensor	
target	

Table 17.4: Dual frame static objects

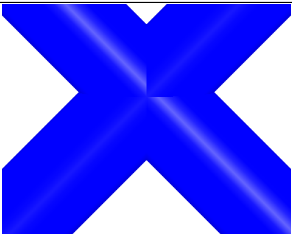
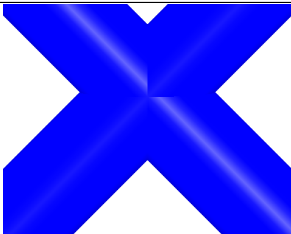

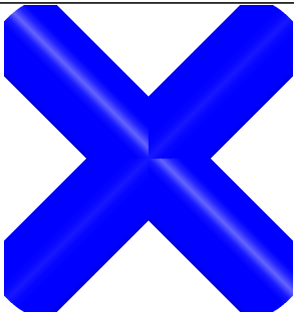
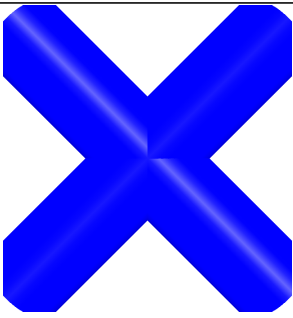
SMOKEVIEW_ID	Image	
	inactive	active
heat_detector		
nozzle		
smoke_detector		
sprinkler_upright		

Table 17.4: Dual frame static objects (continued)

SMOKEVIEW_ID	Image	
	inactive	active
sprinkler_pendent		

17.7.2 Devices with Variable Properties

The appearance of Smokeview objects may be modified using data specified with the array of character strings called SMOKEVIEW_PARAMETERS on the PROP line. For example, the input lines

```
&PROP ID='ballprops', SMOKEVIEW_ID='ball',
      SMOKEVIEW_PARAMETERS (1:6)='R=255','G=0','B=0','DX=0.5','DY=0.25','DZ=0.1' /
&DEVC XYZ=0.5,0.8,1.5, QUANTITY='TEMPERATURE', PROP_ID='ballprops' /
```

create an ellipsoid colored red with x , y , and z axis diameters of 0.5 m and 0.25 m and 0.1 m, respectively. Note that these parameters are enclosed within single quotes because they are character strings passed to Smokeview.

Table 17.5 lists objects with variable properties. Note that the `tsphere` object uses a texture map or image to alter its appearance. The texture map is specified by placing the characters `t%` before the texture file name, for example, `t%texturefile.jpg`.

Table 17.5: Dynamic Smokeview objects



SMOKEVIEW_ID	SMOKEVIEW_PARAMETERS	Image
ball	SMOKEVIEW_PARAMETERS (1:6) = 'R=128','G=192','B=255', 'DX=0.5','DY=.75','DZ=1.0' R, G, B - color components (0 to 255) DX, DY, DZ - amount ball is stretched along x, y, z axis (m)	
cone	SMOKEVIEW_PARAMETERS (1:5) = 'R=128','G=255','B=192', 'D=0.4','H=0.6' R, G, B - color components (0 to 255) D, H - diameter and height (m)	

Table 17.5: Dynamic Smokeview objects (continued)





SMOKEVIEW_ID	SMOKEVIEW_PARAMETERS	Image
fan	<p>SMOKEVIEW_PARAMETERS (1:11) = 'HUB_R=0', 'HUB_G=0', 'HUB_B=0', 'HUB_D=0.1', 'HUB_L=0.12', 'BLADE_R=128', 'BLADE_G=64', 'BLADE_B=32', 'BLADE_ANGLE=60.0', 'BLADE_D=0.5', 'BLADE_H=0.09'</p> <p>HUB_R, HUB_G, HUB_B - color components of fan hub (0 to 255) HUB_D, HUB_L - diameter and length of fan hub (m) BLADE_R, BLADE_G, BLADE_B - color components of fan blades (0 to 255) BLADE_ANGLE, BLADE_D, BLADE_H - angle, diameter and height of a fan blade</p>	
tsphere	<p>SMOKEVIEW_PARAMETERS (1:9) = 'R=255', 'G=255', 'B=255', 'AX0=0.0', 'ELEV0=90.0', 'ROT0=0.0', 'ROTATION_RATE=10.0', 'D=1.0', 'tfile="t%sphere_cover_04.png"'</p> <p>R, G, B - color components (0 to 255) AX0, ELEV0, ROT0 - initial azimuth, elevation and rotation angle (deg) ROTATION_RATE - rotation rate about z axis (deg/s) D - diameter (m) tfile - name of texture map file</p>	

Table 17.5: Dynamic Smokeview objects (continued)

SMOKEVIEW_ID	SMOKEVIEW_PARAMETERS	Image
vent	<p>SMOKEVIEW_PARAMETERS (1:6) = ' R=192' , ' G=192' , ' B=128' , ' W=0.5' , ' H=1.0' , ' ROT=90.0'</p> <p>R, G, B - color components (0 to 255) W, H - width and height (m) ROT - rotation angle (deg)</p>	 inactive vent  active vent

17.7.3 Objects that Represent Lagrangian Particles

Lagrangian particles, like water droplets or small solid particles, are represented in Smokeview as tiny points. However, it is possible to draw Lagrangian particles in other ways, such as those depicted in Table 17.6. For example, the following lines define particles that represent segments of electrical cables that are 10 cm long with a diameter of 1.24 cm:

```
&PART ID='cables', QUANTITIES(1)='PARTICLE TEMPERATURE', ..., PROP_ID='cable image' /
&PROP ID='cable image', SMOKEVIEW_ID='tube', SMOKEVIEW_PARAMETERS='L=0.1','D=0.0124' /
```

By default, the cables are colored black, but you can specify your own default color using the parameters R, G, and B. In addition, you can color the particles according to the listed QUANTITIES on the PART line. Menus in Smokeview allow you to toggle between the various color options.

You can control the orientation of the 'tube' objects using a parameter such as 'RANDXY=1' that causes the cylinders to be drawn randomly in the $x-y$ plane. Objects with the parameters U-VEL, V-VEL, and W-VEL stretch according to the respective velocity components associated with the moving particles.

Table 17.6: Dynamic Smokeview objects for Lagrangian particles




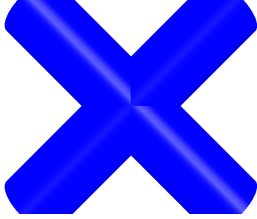
SMOKEVIEW_ID	SMOKEVIEW_PARAMETERS	Image
box	<p>SMOKEVIEW_PARAMETERS (1:6) = ' R=192' , ' G=255' , ' B=128' , ' DX=0.25' , ' DY=.5' , ' DZ=0.125'</p> <p>R, G, B - color components (0 to 255) DX, DY, DZ - amount box is stretched along axes</p>	

Table 17.6: Dynamic Smokeview objects for Lagrangian particles (continued)

SMOKEVIEW_ID	SMOKEVIEW_PARAMETERS	Image
tube	<p>SMOKEVIEW_PARAMETERS (1:6) = 'R=255', 'G=0', 'B=0', 'D=0.2', 'L=0.6', 'RANDXY=1'</p> <p>R, G, B - color components (0 to 255) D, L - diameter and length (m) RANDXY - randomly orient in x-y plane RANDXZ - randomly orient in x-z plane RANDYZ - randomly orient in y-z plane RANDXYZ - random orientation DIRX, DIRY, DIRZ - orient along axis</p>	
velegg	<p>SMOKEVIEW_PARAMETERS (1:9) = 'R=192', 'G=64', 'B=32' 'U-VEL=1.', 'V-VEL=1.', 'W-VEL=1.' 'VELMIN=0.01', 'VELMAX=0.2', 'D=1.0'</p> <p>R, G, B - color components (0 to 255) U-VEL, V-VEL, W-VEL - velocity components (m/s) VELMIN, VELMAX - minimum and maximum velocity D - diameter of egg at maximum velocity (m)</p>	
veltube	<p>SMOKEVIEW_PARAMETERS (1:9) = 'R=0', 'G=0', 'B=0' 'U-VEL=1.', 'V-VEL=1.', 'W-VEL=1.' 'VELMIN=0.01', 'VELMAX=0.2', 'D=0.1'</p> <p>R, G, B - color components (0 to 255) U-VEL, V-VEL, W-VEL - velocity components (m/s) VELMIN, VELMAX - minimum and maximum velocity D - diameter of tube at VELMAX (m)</p>	

Chapter 18

Output

FDS has various types of output files that store computed data. Some of the files are in binary format and intended to be read and rendered by Smokeview. Some of the files are just comma-delimited text files. It is important to remember that you must explicitly declare in the input file most of the FDS output data. A considerable amount of the input file is usually devoted to this.

To visualize the flow patterns better, save planar slices of data, either in the gas or solid phases, by using the `SLCF` (SLiCe File) or `BNDF` (BouNDary File) namelist group. Both of these output formats permit you to animate these quantities in time. For static pictures of the flow field, use the `Plot3D` output, a format that is used by many CFD programs as a simple way to store specified quantities over the entire mesh at one instant in time. Finally, tracer particles can be injected into the flow field from vents or obstacles, and then viewed in Smokeview. Use the `PART` namelist group to control the injection rate, sampling rate and other parameters associated with particles.

18.1 Output Control Parameters: The `DUMP` Namelist Group

The namelist group `DUMP` contains parameters (Table 19.6) that control the rate at which output files are written, and various other global parameters associated with output files. Its parameters include:

`NFRAMES` Number of output dumps per calculation. The default is 1000. Device data, slice data, particle data, isosurface data, 3D smoke data, boundary data, solid phase profile data, and control function data are saved every $(T_END - T_BEGIN) / NFRAMES$ seconds unless otherwise specified using `DT_DEVC`, `DT_SLCF`, `DT_PART`, `DT_ISO`, `DT_BNDF`, `DT_PROF`, or `DT_CTRL`. Note that `DT_SLCF` controls Smoke3D output. `DT_HRR` controls the output of heat release rate and associated quantities.

`MASS_FILE` If `.TRUE.`, produce an output file listing the total masses of all gas species as a function of time. It is `.FALSE.` by default because the calculation of all gas species in all mesh cells is time-consuming. The parameter `DT_MASS` controls the frequency of output.

`MAXIMUM_PARTICLES` Maximum number of Lagrangian particles that can be included on any mesh at any given time. (Default 1000000)

`SMOKE3D` If `.FALSE.`, do not produce an animation of the smoke and fire. It is `.TRUE.` by default.

`DT_PL3D` The time between `Plot3D` file output. Note that versions of FDS before 6 output `Plot3D` files by default. Now, you must specify the interval of output using this parameter. Its default value is 1000000 s, meaning that there is no `Plot3D` output unless specified.

`FLUSH_FILE_BUFFERS` FDS purges the output file buffers periodically and forces the data to be written out into the respective output files. It does this to make it easier to view the case in Smokeview while it is running. It has been noticed on Windows machines that occasionally a runtime error occurs because of file access problems related to the buffer flushing. If this happens, set this parameter to `.FALSE.`, but be aware that it may not be possible to look at output in Smokeview until after the calculation is finished. You may also set `DT_FLUSH` to control the frequency of the file flushing. Its default value is the duration of the simulation divided by `NFRAMES`.

`STATUS_FILES` If `.TRUE.`, produces an output file `CHID.notready` which is deleted, if the simulation is completed successfully. This file can be used as an error indicator. It is `.FALSE.` by default.

18.2 Device Output: The `DEVC` Namelist Group

Every device `DEVC` contains a `QUANTITY` that it monitors. Usually this `QUANTITY` is written out to a comma-delimited spreadsheet file with the suffix `_devc.csv`. The quantities are listed in Table 18.3. There are two types of `DEVC` output. The first is a time history of the given `QUANTITY` over the course of the simulation. The second is a time-averaged profile consisting of a linear array of point devices. Each is explained below.

18.2.1 Single Point Output

If you just want to record the time history of the temperature at a given point, add the line:

```
&DEVC XYZ=6.7,2.9,2.1, QUANTITY='TEMPERATURE', ID='T-1' /
```

and a column will be added to the output file `CHID_devc.csv` under the label `'T-1'`. In this case, the `ID` has no other role than as a column label in the output file. FDS reports the value of the `QUANTITY` in the cell where the point `XYZ` is located.

Devices on Solid Surfaces

When prescribing a solid phase quantity, be sure to position the device at a solid surface. It is not always obvious where the solid surface is since the mesh does not always align with the input obstruction locations. To help locate the appropriate surface, the parameter `IOR` *must* be included when designating a solid phase quantity, except when using the `STATISTICS` feature described in Section 18.10.10 in which case the output quantity is not associated with just a single point on the surface. If the orientation of the solid surface is in the positive x direction, set `IOR=1`. If it is in the negative x direction, set `IOR=-1`, and so for the y and z directions. For example, the line

```
&DEVC XYZ=0.7,0.9,2.1, QUANTITY='WALL TEMPERATURE', IOR=-2, ID='...' /
```

designates the surface temperature of a wall facing the negative y direction. There are still instances where FDS cannot determine which solid surface is being designated, in which case an error message appears in the diagnostic output file. Re-position the device and try again. It is best to position the device, via the real triplet `XYZ`, such that the device location is either at or within a cell width *outside* of the solid surface. The search algorithm in FDS will look for the nearest solid surface in the direction opposite to that indicated by `IOR`.

Integrated Quantities

In addition to point measurements, the DEVC group can be used to report integrated quantities (See Table 18.3). For example, you may want to know the mass flow out of a door or window. To report this, add the line

```
&DEVC XB=0.3,0.5,2.1,2.5,3.0,3.0, QUANTITY='MASS FLOW', ID='whatever' /
```

Note that in this case, a plane is specified rather than a point. The sextuplet XB is used for this purpose. Notice when a flow is desired, two of the six coordinates need to be the same. Another QUANTITY, HRR, can be used to compute the total heat release rate within a subset of the domain. In this case, the sextuplet XB ought to define a volume rather than a plane. Specification of the plane or volume over which the integration is to take place can only be done using XB – avoid planes or volumes that cross multiple mesh boundaries. FDS has to decide which mesh to use in the integration, and it chooses the finest mesh overlapping the centroid of the designated plane or volume.

18.2.2 Linear Array of Point Devices

You can use a single DEVC line to specify a linear array of devices. By adding the parameter POINTS and using the sextuple coordinate array XB, you can direct FDS to create a line of devices from (x_1, y_1, z_1) to (x_2, y_2, z_2) . There are two options.

Steady-State Profile

Sometimes it is convenient to calculate a steady-state profile. For example, the vertical velocity profile along the centerline of a doorway can be recorded with the following line of input:

```
&DEVC XB=X1,X2,Y1,Y2,Z1,Z2, QUANTITY='U-VELOCITY', ID='vel', POINTS=20 /
```

In a file called CHID_line.csv, there will be between 1 and 4 columns of data associated with this single DEVC line. If X1 is different than X2, there will be a column of x coordinates associated with the linear array of points. The same holds for the y and z coordinates. The last column contains the 20 temperature points averaged over the last DT_DEVC_LINE of the simulation—DT_DEVC_LINE is set on the DUMP line. It is half the total simulation time by default. This is a convenient way to output a time-averaged linear profile of a quantity, like an array of thermocouples. Note that the statistics output to the _line.csv file start being averaged at $T=T_END-DT_DEVC_LINE$. Prior to this point in the simulation, the raw values are output. This prevents initial transients from biasing the stat values and forces the “line” file output at the end of the simulation to be equivalent to manually processing a point DEVC time history over the last DT_DEVC_LINE of the simulation.

A single “line” file can hold more than a single line of data. By default, the coordinate columns are labeled using the ID of the DEVC appended with either -x, -y, or -z. To change these labels, use X_ID, Y_ID, and/or Z_ID. To suppress the coordinate columns altogether, add HIDE_COORDINATES=.TRUE. to the DEVC line. This is convenient if you have multiple arrays of data that use the same coordinates. If you want the data plotted as a function of the distance from the origin, $r = \sqrt{x^2 + y^2 + z^2}$, provide the label R_ID.

Time-Varying Profile

If you do not want a steady-state profile, but rather you just want to specify an array of evenly spaced devices, you can use a similar input line, except with the additional attribute TIME_HISTORY.

```
&DEVC XB=X1,X2,Y1,Y2,Z1,Z2, QUANTITY='U-VELOCITY', ID='vel', POINTS=20,
      TIME_HISTORY=.TRUE. /
```

This directs FDS to just add 20 devices to the on-going list, saving you from having to write 20 `DEVC` lines. The `ID` for each device will be 'vel-01', 'vel-02', etc.

Single-Point Statistics

Mean By default, a *line* of devices records a mean or time-averaged profile of a particular quantity,

$$\bar{\phi} = \frac{\sum_{i=1}^n \phi_i}{n} \quad (18.1)$$

where n is the number of uniform output samples.

Min, Max For line devices it is also possible to record the *min* or the *max* values of the output quantity for each point on the line. Set `STATISTICS='TIME MIN'` or `STATISTICS='TIME MAX'`. FDS will output the values over the last `DT_DEVC_LINE` of the simulation.

18.2.3 Quantities at Certain Depth

To record the temperature inside the surface, you can use a device as follows:

```
&DEVC XYZ=..., QUANTITY='INSIDE WALL TEMPERATURE', DEPTH=0.005, ID='Temp_1', IOR=3 /
```

The parameter `DEPTH` (m) indicates the distance inside the solid surface. If `DEPTH` is positive the distance is measured from the front surface. If negative, it is measured from the back surface. Note that if the wall thickness is decreasing over time due to the solid phase reactions, and the distance is measured from the current front surface, the measurement point will be moving towards the back side of the solid. Eventually, the measurement point may emerge from the solid, in which case it starts to show ambient temperature. Measuring the distance from the back surface can then be better suited for the purpose.

Note that `DEPTH` may not perfectly align with the discrete spatial position of the cell center corresponding to the solid cell temperature being output by `INSIDE WALL TEMPERATURE`. Given the stretching and re-meshing done by the solid phase routines, it difficult to compute the local discrete cell position by hand. If the discrete solid cell center position is needed, it may be output using

```
&DEVC XYZ=..., QUANTITY='INSIDE WALL DEPTH', DEPTH=0.005, ID='XC_1', IOR=3 /
```

The output should lie within half a solid grid cell distance from the specified `DEPTH`.

To record the material component's density with time, use the output quantity 'SOLID DENSITY' in the following way:

```
&DEVC ID='...', XYZ=..., IOR=3, QUANTITY='SOLID DENSITY', MATL_ID='wood', DEPTH=0.001
/
```

This produces a time history of the density of the material referred to as 'wood' on a `MATL` line. The density is recorded 1 mm beneath the surface which is oriented in the positive z direction. Note that if 'wood' is part of a mixture, the density represents the mass of 'wood' per unit volume of the mixture.

To record the solid conductivity, use `QUANTITY='SOLID CONDUCTIVITY'`. To record the solid specific heat, use `QUANTITY='SOLID SPECIFIC HEAT'`. These quantities do not need the `MATL_ID` keyword.

Note that these quantities are allowed only as a `DEVC`, not a `BNDF`, output.

18.2.4 Back Surface Temperature

If you just want to know the temperature of the back surface of the “wall,” then use

```
&DEVC XYZ=..., QUANTITY='BACK WALL TEMPERATURE', ID='Temp_b', IOR=3 /
```

Note that this quantity is only meaningful if the front or exposed surface of the “wall” has the attribute `BACKING='EXPOSED'` on the `SURF` line that defines it. The coordinates, `XYZ`, and orientation, `IOR`, refer to the front surface. To check that the heat conduction calculation is being done properly, you can add the additional line

```
&DEVC XYZ=..., QUANTITY='WALL TEMPERATURE', ID='Temp_f', IOR=-3 /
```

where now `XYZ` and `IOR` refer to the coordinates and orientation of the back side of the wall. These two wall temperatures ought to be the same. Remember that the “wall” in this case can only be at most one mesh cell thick, and its `THICKNESS` need not be the same as the mesh cell width. Rather, the `THICKNESS` ought to be the actual thickness of the “wall” through which FDS performs a 1-D heat conduction calculation.

18.3 Profiles of Quantities: The PROF Namelist Group

FDS uses a fine one-dimensional mesh at each boundary cell to compute heat transfer within a solid. Use the `PROF` output to record the properties of the solid over the entire thickness. The parameters (Table 19.19) to specify a given `PROF` are similar to those used to specify a surface quantity in the `DEVC` group. `XYZ` designates the triplet of coordinates, `QUANTITY` is the physical quantity to monitor, `IOR` the orientation, and `ID` an identifying character string. Here is an example of how you would use this feature to get a time history of temperature profiles within a given solid obstruction:

```
&PROF XYZ=..., QUANTITY='TEMPERATURE', ID='T-1', IOR=3 /
```

Other possible quantities are the total density of the wall (`QUANTITY = 'DENSITY'`) or densities of solid material components (`QUANTITY = '[MATL_ID]'`), where `MATL_ID` is the name of the material. Each `PROF` line creates a separate file. The format of the file produced by each `PROF` line includes the node coordinates and specified quantity every `DT_PROF` s. However, if you specify `FORMAT_INDEX=2` on the `PROF` line, the resulting file will contain columns containing only the final set of node coordinates and quantity values. This is handy for displaying a steady-state temperature profile.

18.4 Animated Planar Slices: The SLCF Namelist Group

The `SLCF` (“slice file”) namelist group parameters (Table 19.24) allows you to record various gas phase quantities at more than a single point. A “slice” refers to a subset of the whole domain. It can be a line, plane, or volume, depending on the values of `XB`. The sextuplet `XB` indicates the boundaries of the “slice” plane. `XB` is prescribed as in the `OBST` or `VENT` groups, with the possibility that 0, 2, or 4 out of the 6 values be the same to indicate a volume, plane or line, respectively. A handy trick is to specify, for example, `PBY=5.3` instead of `XB` if it is desired that the entire plane $y = 5.3$ slicing through the domain be saved. `PBX` and `PBZ` control planes perpendicular to the x and z axes, respectively.

By default, 1-D and 2-D slice files are saved `NFRAMES` times per simulation. You can control the frequency of output with `DT_SLCF` on the `DUMP` line. If the “slice” is a 3-D volume, then its output frequency is controlled by the parameter `DT_SL3D`. By default, FDS sets `DT_SL3D = (T_END - T_BEGIN) / 5`. You may

specify a different value of `DT_SL3D` on `DUMP`. Note that 3-D slice files can become extremely large if `DT_SL3D` is small.

Animated vectors can be created in Smokeview if a given `SLCF` line has the attribute `VECTOR=.TRUE.` If two `SLCF` entries are in the same plane, then only one of the lines needs to have `VECTOR=.TRUE.` Otherwise, a redundant set of velocity component slices will be created.

Normally, FDS averages slice file data at cell corners. For example, gas temperatures are computed at cell centers, but they are linearly interpolated to cell corners and output to a file that is read by Smokeview. To prevent this from happening, set `CELL_CENTERED=.TRUE.` This forces FDS to output the actual cell-centered data with no averaging. Note that this feature is mainly useful for diagnostics because it enables you to visualize the values that FDS actually computes. Note also that this feature should only be used for scalar quantities that are computed at cell centers, like temperatures, mass fractions, etc.

Slice file information is recorded in files (See Section 22.7) labeled `CHID_n.sf`, where n is the index of the slice file. A short Fortran program `fds2ascii.f90` produces a text file from a line, plane or volume of data. See Section 18.11 for more details.

By default, Smokeview will blank slice file data inside obstructions. However, this is expensive to load at startup in Smokeview for large cases. If you wish Smokeview not to store this blanking array, set `IBLANK_SMV=.FALSE.` on `MISC`. Another option is to run Smokeview from the command line and to add `-noblack` as an option.

18.5 Animated Boundary Quantities: The `BNDF` Namelist Group

The `BNDF` (“boundary file”) namelist group parameters allows you to record surface quantities at all solid obstructions. As with the `SLCF` group, each quantity is prescribed with a separate `BNDF` line, and the output files are of the form `CHID_n.bf`. No physical coordinates need be specified, however, just `QUANTITY`. See Table 18.3. For certain output quantities, additional parameters need to be specified via the `PROP` namelist group. In such cases, add the character string, `PROP_ID`, to the `BNDF` line to tell FDS where to find the necessary extra information.

Note that `BNDF` files (Section 22.9) can become very large, so be careful in prescribing the time interval, `DT_BNDF` on the `DUMP` line. One way to reduce the size of the output file is to turn off the drawing of boundary information on desired obstructions. On any given `OBST` line, if the string `BNDF_OBST=.FALSE.` is included, the obstruction is not colored. To turn off all boundary drawing, set `BNDF_DEFAULT=.FALSE.` on the `MISC` line. Then individual obstructions can be turned back on with `BNDF_OBST=.TRUE.` on the appropriate `OBST` line. Individual faces of a given obstruction can be controlled via `BNDF_FACE(IOR)`, where `IOR` is the index of orientation (+1 for the positive x direction, -1 for negative, and so on). Normally, FDS averages boundary file data at cell corners. For example, surface temperatures are computed at the center of each surface cell, but they are linearly interpolated to cell corners and output to a file that is read by Smokeview. To prevent this from happening, set `CELL_CENTERED=.TRUE.` on the `BNDF` line. This forces FDS to output the actual cell-centered data with no averaging. Note that this feature is mainly useful for diagnostics.

Sometimes it is useful to render the `QUANTITY` integrated over time. For example, a heat flux in units of kW/m^2 can be integrated in time producing the total energy absorbed by the surface in units of kJ/m^2 . To do this, set `STATISTICS` equal to `'TIME INTEGRAL'` on the `BNDF` line. Note that there are no other options for `STATISTICS` on a `BNDF` line.

18.6 Animated Isosurfaces: The ISO F Namelist Group

The ISO F (“ISOsurface File”) namelist group creates three-dimensional animated contours of gas phase scalar quantities. For example, a 300 °C temperature isosurface is a 3-D surface on which the gas temperature is 300 °C. Three different values of the temperature can be saved via the line:

```
&ISO F QUANTITY='TEMPERATURE', VALUE(1)=50., VALUE(2)=200., VALUE(3)=500. /
```

where the values are in °C. Note that the isosurface output files CHID_*n*.iso can become very large, so experiment with different sampling rates (DT_ISO F on the DUMP line).

Any gas phase quantity can be animated via iso-surfaces, but use caution. To render an iso-surface, the desired quantity must be computed in every mesh cell at every output time step. For quantities like 'TEMPERATURE', this is not a problem, as FDS computes it and saves it anyway. However, species volume fractions demand substantial amounts of time to compute at each mesh cell. Remember to include the SPEC_ID corresponding to the given QUANTITY if necessary.

18.7 Plot3D Static Data Dumps

Data stored in Plot3D [?] files use a format developed by NASA that is used by many CFD programs for representing simulation results. See Section 22.8 for a description of the file structure. Plot3D data is visualized in three ways: as 2-D contours, vector plots and iso-surfaces. Vector plots may be viewed if one or more of the *u*, *v* and *w* velocity components are stored in the Plot3D file. The vector length and direction show the direction and relative speed of the fluid flow. The vector colors show a scalar fluid quantity such as temperature. Five quantities are written out to a file at one instant in time. The default specification is:

```
&DUMP ..., PLOT3D_QUANTITY(1:5)='TEMPERATURE',  
          'U-VELOCITY', 'V-VELOCITY', 'W-VELOCITY', 'HRRPUV' /
```

It's best to leave the velocity components as is, because Smokeview uses them to draw velocity vectors. If any of the specified quantities require the additional specification of a particular species, use PLOT3D_SPEC_ID(*n*) to provide the SPEC_ID for PLOT3D_QUANTITY(*n*).

Plot3D data are stored in files with extension .q. There is an optional file that can be output with coordinate information if another visualization package is being used to render the files. If you write WRITE_XYZ=.TRUE. on the DUMP line, a file with suffix .xyz is written out. Smokeview does not require this file because the coordinate information can be obtained elsewhere.

Past versions of FDS (1-5) output Plot3D files by default. Now, you must specify the time interval between dumps using DT_PL3D on the DUMP line.

18.8 SMOKE3D: Realistic Smoke and Fire

When you do a fire simulation, FDS automatically creates two output files that are rendered by Smokeview as realistic looking smoke and fire. By default, the output quantities are the 'MASS FRACTION' of 'SOOT' and 'HRRPUV' (Heat Release Rate Per Unit Volume). You have the option of rendering any other species mass fraction instead of 'SOOT', so long as the MASS_EXTINCTION_COEFFICIENT on the SPEC line is appropriate in describing the attenuation of visible light by the specified gas species.

An alternative to SOOT mass fraction can be specified via SMOKE3D_QUANTITY on the DUMP line. If the specified quantity requires the additional specification of a particular species, use SMOKE3D_SPEC_ID to provide the SPEC_ID. Here is an example of how to change the smoke species. Normally, you do not need

to do this as the “smoke” is an assumed part of the default combustion model when a non-zero `SOOT_YIELD` is defined on the `REAC` line.

```
&SPEC ID='MY SMOKE', MW=29., MASS_EXTINCTION_COEFFICIENT=8700. /
&DUMP SMOKE3D_QUANTITY='MASS FRACTION', SMOKE3D_SPEC_ID='MY SMOKE' /
```

The `MASS_EXTINCTION_COEFFICIENT` is passed to Smokeview to be used for visualization.

18.9 Particle Output Quantities

This section discusses output options for Lagrangian particles.

18.9.1 Liquid Droplets that are Attached to Solid Surfaces

Liquid droplets (as opposed to solid particles) “stick” to solid surfaces unless directed otherwise. There are various quantities that describe these populations. For example, ‘`MPUA`’ is the Mass Per Unit Area of the droplets defined by `PART_ID`. Likewise, ‘`AMPUA`’ is the Accumulated Mass Per Unit Area. Both of these are given in units of kg/m^2 . Think of these outputs as measures of the instantaneous mass density per unit area, and the accumulated total, respectively. These quantities are not identical measures. The quantity ‘`AMPUA`’ is analogous to a “bucket test,” where the droplets are collected in buckets and the total mass determined at the end of a given time period. In this case each grid cell on the floor is considered its own bucket. Each droplet is counted only once when it reaches the floor¹. `MPUA` counts a droplet whenever it is on any solid surface, including the walls. If the droplet moves from one solid wall cell to another, it will be counted again. The cooling of a solid surface by droplets of a given type is given by ‘`CPUA`’, the Cooling Per Unit Area in units of kW/m^2 . Since a typical sprinkler simulation only tracks a small fraction of the droplets emitted from a sprinkler, both `MPUA` and `CPUA` also perform an exponential smoothing. This avoids having spotted distributions on surfaces due to the infrequent arrival of droplets that likely have a high weighting factor.

Each of the output quantities mentioned above has a variant in which the quantity is summed by species rather than particle type. For example, the quantity ‘`AMPUA_Z`’ along with a specified `SPEC_ID` rather than a `PART_ID` will sum the given output quantity over all particle classes with the given `SPEC_ID`.

As an example of how to use these kinds of output quantities, the test case `bucket_test_1` describes a single sprinkler mounted 10 cm below a 5 m ceiling. Water flows for 30 s at a constant rate of 180 L/min (ramped up and down in 1 s). The simulation continues for another 10 s to allow water drops time to reach the floor. The total mass of water discharged is

$$180 \frac{\text{L}}{\text{min}} \times 1 \frac{\text{kg}}{\text{L}} \times \frac{1}{60} \frac{\text{min}}{\text{s}} \times 30 \text{ s} = 90 \text{ kg} \quad (18.2)$$

In the simulation, the quantity ‘`AMPUA`’ with `STATISTICS='SURFACE INTEGRAL'` is specified on the `DEVC` line. This results in FDS summing ‘`AMPUA`’ over each grid cell in the volume defined by `XB`, in this case the entire floor, analogous to if there were an single bucket present that was the same size as the area specified with `XB`. Summing the values of ‘`AMPUA`’ over the entire floor yields a total of 90 kg (Fig. 18.1). Note that there really is no need to time-average the results. The quantity is inherently accumulating.

¹Be aware of the fact that the default behavior for liquid droplets hitting the “floor,” that is, the plane $z = Z_{\text{MIN}}$, is to disappear (`POROUS_FLOOR=.TRUE.` on the `MISC` line). In this case, ‘`MPUA`’ will be zero, but ‘`AMPUA`’ will not. FDS stores the droplet mass just before removing the droplet from the simulation for the purpose of saving CPU time.

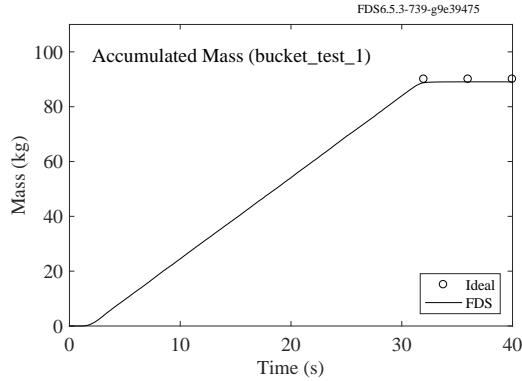


Figure 18.1: Accumulated water collected at the floor in the `bucket_test_1` case.

18.9.2 Solid Particles on Solid Surfaces

If you want to monitor the accumulation of solid particles that have fallen on a solid surface, use a device as follows:

```
&DEVIC ID=..., XB=..., QUANTITY='MPUV', PART_ID='rods', STATISTICS='VOLUME INTEGRAL' /
```

The volume over which to integrate, `XB`, should be at least one grid cell thick above the surface.

18.9.3 Droplet and Particle Densities and Fluxes in the Gas Phase

Away from solid surfaces, `'MPUV'` is the Mass Per Unit Volume of particles or droplets of type (`PART_ID`) in units of kg/m^3 . `'MPUV_Z'` provides the same information integrated over all droplets of a single species, (`SPEC_ID`).

The quantities `'PARTICLE FLUX X'`, `'PARTICLE FLUX Y'`, and `'PARTICLE FLUX Z'` produce slice and Plot3D colored contours of the mass flux of particles in the x , y , and z directions, respectively, in units of $\text{kg/m}^2/\text{s}$. You can also apply these quantities to a device. For example, in the case called `bucket_test_4.fds`, the input line

```
&DEVIC XB=..., ID='flux', QUANTITY='PARTICLE FLUX Z', STATISTICS='AREA INTEGRAL' /
```

records the integrated mass flux of *all* particles passing through the given horizontal plane. Figure 18.2 presents the results of this simple test case in which water spraying at a rate of 0.0005 kg/s for 55 s passes through a measurement plane and onto the floor. The total water accumulated is 0.0275 kg .

18.9.4 Coloring Particles and Droplets in Smokeview

The parameter `QUANTITIES` on the `PART` line is an array of character strings indicating which scalar quantities should be used to color particles and droplets in Smokeview. The choices are

```
'PARTICLE TEMPERATURE' (°C)
'PARTICLE DIAMETER' (μm)
'PARTICLE VELOCITY' (m/s)
'PARTICLE MASS' (kg)
'PARTICLE AGE' (s)
```

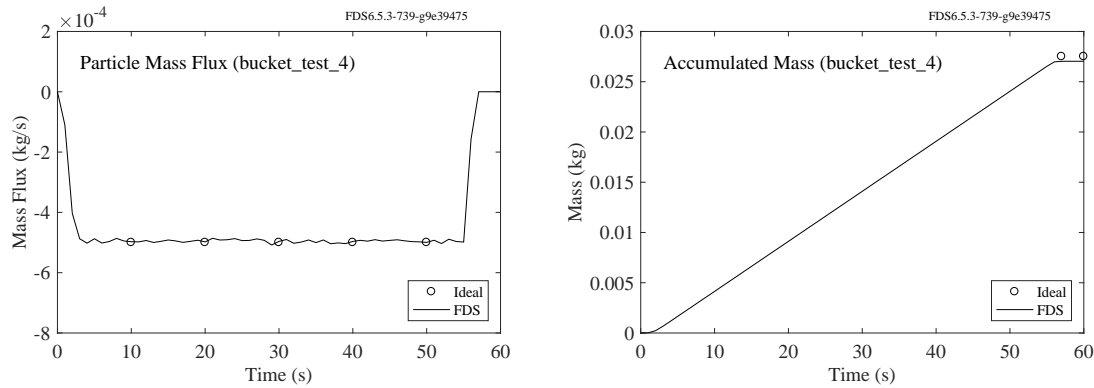


Figure 18.2: Mass flux and accumulated water collected at the floor in the `bucket_test_4` case.

By default, if no `QUANTITIES` are specified and none are selected in Smokeview, then Smokeview will display particles with a single color. To select this color specify either `RGB` or `COLOR`. By default, water droplets are colored blue. All others are colored black.

For solid particles with a specified `SURF_ID`, you may specify any of the solid phase output quantities listed in Table 18.3. If the specified quantity is associated with a species, use the parameter `QUANTITIES_SPEC_ID (N)` to specify the species. Here `N` refers to the order of the specified output quantities on the `PART` line.

18.9.5 Detailed Properties of Solid Particles

You may output properties of a single solid particle using a `DEVC` (device) line. For example, the lines:

```
&INIT ID='my particle', PART_ID='...', XB=..., N_PARTICLES=1 /
&DEVC ID='...', INIT_ID='my particle', QUANTITY='WALL TEMPERATURE' /
```

output the surface temperature of a single particle that has been introduced into the simulation via an `INIT` line.

If the particle is split (by means of multiple `ORIENTATION` vectors on the `PART` line), you can specify which of the particle fragments you want using the parameter `ORIENTATION_NUMBER` on the `DEVC` line.

If the `INIT` line has a `MULT_ID`; that is, an array of particles is introduced via this single `INIT` line, then the `INIT_ID` for each takes the form '`[ID]-00308`', where `ID` is that of the `INIT` line, and 308 refers to the 308-th `INIT` line generated by the multiplicative sequence. The `INIT` lines are generated by looping over the x indices, then y , then z . For example, the following input lines

```
&INIT ID='P', PART_ID='CRIB1', XYZ=0.005,0.005,0.005 ,
      N_PARTICLES=1, CELL_CENTERED=.TRUE., MULT_ID='array' /
&MULT ID='array', DX=0.01, DY=0.01, DZ=0.01, I_LOWER=0, I_UPPER=9,
      J_LOWER=0, J_UPPER=9, K_LOWER=0, K_UPPER=3 /
&DEVC ID='T307', INIT_ID='P-00307', QUANTITY='WALL TEMPERATURE' /
```

specify an array of 10 by 10 by 4 particles. We want to record the surface temperature of the 307-th particle. See Section 7.5 for details of how the `MULT` line is interpreted.

18.10 Special Output Quantities

This section lists a variety of output quantities that are useful for studying thermally-driven flows, combustion, pyrolysis, and so forth. Note that some of the output quantities can be produced in a variety of ways.

18.10.1 Heat Release Rate

Quantities associated with the overall energy budget are reported in the comma delimited file `CHID_hrr.csv`. This file is automatically generated; the only input parameter associated with it is `DT_HRR` on the `DUMP` line. The columns in this file record the time history of the integrals of the terms in the enthalpy transport equation. The columns are defined as follows:

$$\underbrace{\frac{\partial}{\partial t} \int \rho h_s dV}_{Q_ENTH} = \underbrace{\dot{m}_b h_{s,b} - \int \rho \mathbf{u} h_s \cdot d\mathbf{S}}_{Q_CONV} + \underbrace{\dot{q}_{b,w} + \int k \nabla T \cdot d\mathbf{S}}_{Q_COND} + \underbrace{\sum_{\alpha} \int h_{s,\alpha} \rho D_{\alpha} \nabla Y_{\alpha} \cdot d\mathbf{S}}_{Q_DIFF} \\ + \underbrace{\dot{q}_{b,r} - \int \dot{\mathbf{q}}_r'' \cdot d\mathbf{S}}_{Q_RADI} + \underbrace{\int \dot{q}''' dV}_{HRR} + \underbrace{\int \frac{d\bar{p}}{dt} dV}_{Q_PRES} + \underbrace{(-\dot{q}_{b,c} - \dot{q}_{b,r} - \dot{q}_{b,w})}_{Q_PART} \quad (18.3)$$

An additional column, `Q_TOTAL`, includes the sum of the terms on the right hand side of the equation. Ideally, this sum should equal the term on the left, `Q_ENTH`. All terms are reported in units of kW. Note that the terms that make up `Q_PART` are summed over the Lagrangian particles. They represent the heat absorbed by the particles via convection, radiation, and conduction from the wall.

The other columns in the file contain the total burning rate of fuel, in units of kg/s, and the zone pressures. Note that the reported value of the burning rate is not adjusted to account for the possibility that each individual material might have a different heat of combustion. For this reason, it is not always the case that the reported total burning rate multiplied by the gas phase heat of combustion is equal to the reported heat release rate.

Note that the volume integrations in Eq. (18.3) are performed over the entire domain. The differential, dV , is the product of the local grid cell dimensions, $dx dy dz$. For the special case of two-dimensional cylindrical coordinates, $dV = r dr d\theta dz$, where $r = x$, $dr = dx$, and $d\theta = dy$.

As a test of the energy balance, a sample case called `Pressure_Solver/hallways.fds` simulates a fire near the end of five connected hallways. The other end of the hallways is open. As is seen in Fig. 18.3, the quantities `Q_ENTH` and `Q_TOTAL` are very closely matched, indicating that the sources of energy loss and gain are properly added and subtracted from the energy equation. As expected, the net energy gain/loss eventually goes to zero as the compartment reaches a quasi-steady state.

18.10.2 Visibility and Obscuration

If you are performing a fire calculation using the simple chemistry approach, the smoke is tracked along with all other major products of combustion. The most useful quantity for assessing visibility in a space is the *light extinction coefficient*, K [1/m]. The intensity of monochromatic light passing a distance L through smoke is attenuated according to

$$I/I_0 = e^{-KL} \quad (18.4)$$

The light extinction coefficient, K , is a product of the density of smoke particulate, ρY_S , and a mass specific extinction coefficient that is fuel dependent

$$K = K_m \rho Y_S \quad (18.5)$$

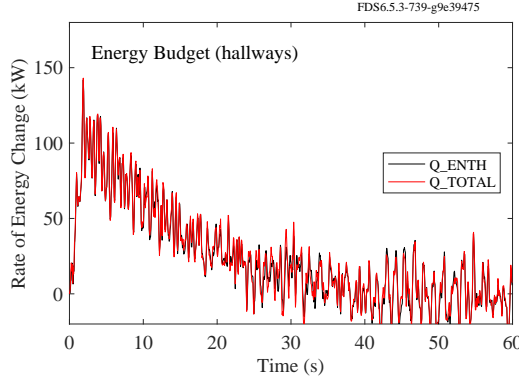


Figure 18.3: The energy budget for the hallways test case.

Devices that output a % obscuration such as a DEVC with a QUANTITY of 'ASPIRATION', 'CHAMBER OBSCURATION' (smoke detector), or 'PATH OBSCURATION' (beam detector) are discussed respectively in Section 17.3.7, Section 17.3.5, and Section 17.3.6.

Estimates of visibility through smoke can be made by using the equation

$$S = C/K \quad (18.6)$$

where C is a non-dimensional constant characteristic of the type of object being viewed through the smoke, i.e., $C = 8$ for a light-emitting sign and $C = 3$ for a light-reflecting sign [?]. Since K varies from point to point in the domain, the visibility S does as well.

Three parameters control smoke production and visibility. The first is the SOOT_YIELD on the REAC line, defined as the fraction of fuel mass that is converted to soot if the simple chemistry approach is being used. The second parameter, MASS_EXTINCTION_COEFFICIENT, is the K_m in Eq. (18.5). It is defined on one or more of the SPEC lines² for the various light absorbing gas species. Its default value is 8700 m²/kg, a value suggested for flaming combustion of wood and plastics³. The third parameter, VISIBILITY_FACTOR on the MISC line, is the constant C in Eq. (18.6). It is 3 by default.

The gas phase output quantity 'EXTINCTION COEFFICIENT' is K . A similar quantity is the 'OPTICAL DENSITY', $D = K/2.3$, the result of using \log_{10} in the definition

$$D \equiv -\frac{1}{L} \log_{10} \left(\frac{I}{I_0} \right) = K \log_{10} e \quad (18.7)$$

The visibility S is output via the QUANTITY called 'VISIBILITY'. Note that, by default, the visibility is associated with the smoke that is implicitly defined by the simple chemistry model. However, this quantity can also be associated with an explicitly defined species via the inclusion of a SPEC_ID. In other words, you can specify the output quantity 'VISIBILITY' along with a SPEC_ID. This does not require that you do a simple chemistry calculation; only that you have specified the given species via a separate SPEC line. You can specify a unique MASS_EXTINCTION_COEFFICIENT on the SPEC line as well.

Note that FDS cannot report a visibility of infinity, but rather reports a MAXIMUM_VISIBILITY that you can control via the MISC line. The default is 30 m.

²When using the simple chemistry combustion model, you can change the default mass extinction coefficient by adding a line to the input file of the form: &SPEC ID='SOOT', MASS_EXTINCTION_COEFFICIENT=..., LUMPED_COMPONENT_ONLY=.TRUE. /

³For most flaming fuels, a suggested value for K_m is 8700 m²/kg \pm 1100 m²/kg at a wavelength of 633 nm [?]

18.10.3 Layer Height and the Average Upper and Lower Layer Temperatures

Fire protection engineers often need to estimate the location of the interface between the hot, smoke-laden upper layer and the cooler lower layer in a burning compartment. Relatively simple fire models, often referred to as *two-zone models*, compute this quantity directly, along with the average temperature of the upper and lower layers. In a computational fluid dynamics (CFD) model like FDS, there are not two distinct zones, but rather a continuous profile of temperature. Nevertheless, there are methods that have been developed to estimate layer height and average temperatures from a continuous vertical profile of temperature. One such method [?] is as follows: Consider a continuous function $T(z)$ defining temperature T as a function of height above the floor z , where $z = 0$ is the floor and $z = H$ is the ceiling. Define T_u as the upper layer temperature, T_l as the lower layer temperature, and z_{int} as the interface height. Compute the quantities:

$$\begin{aligned} (H - z_{\text{int}}) T_u + z_{\text{int}} T_l &= \int_0^H T(z) dz = I_1 \\ (H - z_{\text{int}}) \frac{1}{T_u} + z_{\text{int}} \frac{1}{T_l} &= \int_0^H \frac{1}{T(z)} dz = I_2 \end{aligned}$$

Solve for z_{int} :

$$z_{\text{int}} = \frac{T_l (I_1 I_2 - H^2)}{I_1 + I_2 T_l^2 - 2 T_l H} \quad (18.8)$$

Let T_l be the temperature in the lowest mesh cell and, using Simpson's Rule, perform the numerical integration of I_1 and I_2 . T_u is defined as the average upper layer temperature via

$$(H - z_{\text{int}}) T_u = \int_{z_{\text{int}}}^H T(z) dz \quad (18.9)$$

Further discussion of similar procedures can be found in Ref. [?].

The quantities 'LAYER HEIGHT', 'UPPER TEMPERATURE' and 'LOWER TEMPERATURE' can be designated via DEVC lines in the input file. For example, the line:

```
&DEVC XB=2.0,2.0,3.0,3.0,0.0,3.0, QUANTITY='LAYER HEIGHT', ID='whatever' /
```

produces a time history of the smoke layer height at $x = 2$ and $y = 3$ between $z = 0$ and $z = 3$. If multiple meshes are being used, the vertical path *cannot* cross mesh boundaries.

18.10.4 Thermocouples

The output quantity THERMOCOUPLE is the temperature of a modeled thermocouple. The thermocouple temperature lags the true gas temperature by an amount determined mainly by its bead size. It is found by solving the following equation for the thermocouple temperature, T_{TC} [?]

$$\rho_{\text{TC}} c_{\text{TC}} \frac{dT_{\text{TC}}}{dt} = \epsilon_{\text{TC}} (U/4 - \sigma T_{\text{TC}}^4) + h(T_g - T_{\text{TC}}) = 0 \quad (18.10)$$

where ϵ_{TC} is the emissivity of the thermocouple, U is the integrated radiative intensity, T_g is the true gas temperature, and h is the heat transfer coefficient to a small sphere, $h = k\text{Nu}/D_{\text{TC}}$. The bead DIAMETER, EMISSIVITY, DENSITY, and SPECIFIC_HEAT are given on the associated PROP line. To over-ride the calculated value of the heat transfer coefficient, set HEAT_TRANSFER_COEFFICIENT on the PROP line ($\text{W}/(\text{m} \cdot \text{K})$). The default value for the bead diameter is 0.001 m. The default emissivity is 0.85. The default values for the bead density and specific heat are that of nickel; 8908 kg/m^3 and 0.44 kJ/kg/K , respectively. See the discussion on heat transfer to a water droplet in the Technical Reference Guide for details of the convective heat transfer to a small sphere.

18.10.5 Heat Flux

There are various output quantities related to heat flux. First, consider the *net* heat flux to a solid surface:

$$\dot{q}_{\text{net}}'' = \varepsilon_s (\dot{q}_{\text{inc,rad}}'' - \sigma T_s^4) + h(T_{\text{gas}} - T_s) \quad (18.11)$$

where $\dot{q}_{\text{inc,rad}}''$ is the *incident* radiative heat flux, ε_s is the surface emissivity, h is the convective heat transfer coefficient, T_s is the surface temperature, and T_{gas} is the gas temperature in the vicinity of the surface. If you want to output the *net* heat flux, use the QUANTITY called 'NET HEAT FLUX'. The individual components, the *net* convective and radiative fluxes, are 'CONVECTIVE HEAT FLUX' and 'RADIATIVE HEAT FLUX', respectively. If you just want to output $\dot{q}_{\text{inc,rad}}''$, use 'INCIDENT HEAT FLUX'.

If you want to compare the heat flux predicted by FDS to a measurement made with a heat flux gauge, use 'GAUGE HEAT FLUX':

$$\dot{q}_{\text{gauge}}'' = \varepsilon_{\text{gauge}} (\dot{q}_{\text{inc,rad}}'' - \sigma T_{\text{gauge}}^4) + h(T_{\text{gas}} - T_{\text{gauge}}) \quad (18.12)$$

If the heat flux gauge used in an experiment has a temperature other than ambient or an emissivity other than 1, use GAUGE_TEMPERATURE (T_{gauge}) and GAUGE_EMISSIVITY ($\varepsilon_{\text{gauge}}$) on the PROP line associated with the device:

```
&DEVC ID='hf', XYZ=..., IOR=-2, QUANTITY='GAUGE HEAT FLUX', PROP_ID='hf props' /
&PROP ID='hf props', GAUGE_TEMPERATURE=80., GAUGE_EMISSIVITY=0.9 /
```

The heat transfer coefficient, h , is that of the solid surface to which the device is attached.

Similar to a heat flux gauge, a 'RADIOMETER' measures only the radiative heat flux:

$$\dot{q}_{\text{radiometer}}'' = \varepsilon_{\text{gauge}} (\dot{q}_{\text{inc,rad}}'' - \sigma T_{\text{gauge}}^4) \quad (18.13)$$

The GAUGE_TEMPERATURE (T_{gauge}) and GAUGE_EMISSIVITY ($\varepsilon_{\text{gauge}}$) can be set on the PROP line associated with the device if their values are different than ambient and 1, respectively.

All of the above heat flux output quantities are defined at a solid surface. To record the radiative heat flux away from a solid surface, add a device with the following format:

```
&DEVC ID='flux', QUANTITY='RADIATIVE HEAT FLUX GAS', XYZ=..., ORIENTATION=-1,0,0 /
```

Note that XB and STATISTICS are not appropriate for this quantity. This single line of input is a special shortcut⁴ for the following lines of input that make use of a Lagrangian particle as a surrogate target:

```
&DEVC ID='flux', INIT_ID='f1', QUANTITY='RADIATIVE HEAT FLUX' /
&INIT ID='f1', XYZ=..., N_PARTICLES=1, PART_ID='rad gauge' /
&PART ID='rad gauge', STATIC=.TRUE., ORIENTATION(1:3,1)=-1,0,0, SURF_ID='target' /
&SURF ID='target', RADIUS=0.001, GEOMETRY='SPHERICAL', EMISSIVITY=1. /
```

Note that the DEVC line does not contain device coordinates, but rather a reference to the INIT line that positions the single surrogate particle at the point XYZ. The INIT line references the PART line, which provides information about the particle, in particular the orientation of the heat flux gauge. The reference to the SURF line is mainly for consistency – FDS needs to know something about the particle’s geometry even though it is really just a “target.”

The functionality of surrogate particles can be extended to model an array of devices. Instead of one heat flux gauge, we can create a line of them:

⁴This feature maintains backward compatibility with FDS 5.

```
&DEVC ID='flux', INIT_ID='f1', POINTS=34, QUANTITY='RADIATIVE HEAT FLUX', X_ID='x' /
&INIT ID='f1', XYZ=..., N_PARTICLES=34, DX=0.05, PART_ID='rad gauge' /
```

For more information about specifying arrays of devices via the parameter `POINTS`, see Section 18.2.2. Note also the parameter `DX` on the `INIT` line that creates a line of particles starting at the point `XYZ` and repeating every 0.05 m.

18.10.6 Adiabatic Surface Temperature

FDS includes a calculation of the adiabatic surface temperature (AST), a quantity that is representative of the heat flux to a solid surface. Following the idea proposed by Ulf Wickström [?], T_{AST} is the surface temperature for which the net heat flux (Eq. (18.11)) is zero. The following equation can be solved using an analytical solution given by Malendowski [?]:

$$\varepsilon (\dot{q}_{inc,rad}'' - \sigma T_{AST}^4) + h(T_{gas} - T_{AST}) = 0 \quad (18.14)$$

where, $\dot{q}_{inc,rad}''$ is the *incident* radiative heat flux onto the surface, ε is the surface emissivity, h is the convective heat transfer coefficient, and T_{gas} is the surrounding gas temperature.

The usefulness of the AST is that it represents an effective exposure temperature that can be passed on to a more detailed model of the solid object. It provides the gas phase thermal boundary condition in a single quantity, and it is not affected by the uncertainty associated with the solid phase heat conduction model within FDS. Obviously, the objective in passing information to a more detailed model is to get a better prediction of the solid temperature (and ultimately its mechanical response) than FDS can provide. To reinforce this notion, you can output the adiabatic surface temperature even when there is no actual solid surface using the following lines:

```
&DEVC ID='AST', XYZ=..., QUANTITY='ADIABATIC SURFACE TEMPERATURE GAS',
      ORIENTATION=0.707,0.0,0.707, PROP_ID='props' /
&PROP ID='props', EMISSIVITY=0.9, HEAT_TRANSFER_COEFFICIENT=10. /
```

This output indicates the maximum achievable solid surface temperature at the given location `XYZ` that is *not* actually in the vicinity of a solid surface, facing in any direction as indicated by the `ORIENTATION` vector. Note that you *must* set the `EMISSIVITY` and `HEAT_TRANSFER_COEFFICIENT` (W/(m²·K)) on the `PROP` line because there is no actual solid surface from which to infer these values.

Example: AST vs. Surface Temperature

The test case called `adiabatic_surface_temperature.fds` in the `Radiation` folder simulates a 0.1 mm steel plate being heated by a thermal plume. The plate is perfectly insulated (`BACKING='INSULATED'` on the `SURF` line) and its steady-state temperature should be equivalent to its adiabatic surface temperature or AST. The left plot of Fig. 18.10.6 shows the plate temperature rising towards the AST, much as an actual plate thermometer would. The right plot simply shows the AST calculated using the `QUANTITY 'ADIABATIC SURFACE TEMPERATURE'` applied via a `DEVC` at the plate surface, and the AST calculated using the `QUANTITY 'ADIABATIC SURFACE TEMPERATURE GAS'` applied via a `DEVC` positioned just in front of the plate. The idea of the latter device would be to record the AST even if the plate were not actually represented in the simulation. For these two recordings of the AST to be identical, the plate `SURFACE` conditions and the `AST-Gas` `PROPERTIES` must both include the same explicitly-defined `HEAT_TRANSFER_COEFFICIENT` and `EMISSIVITY`.

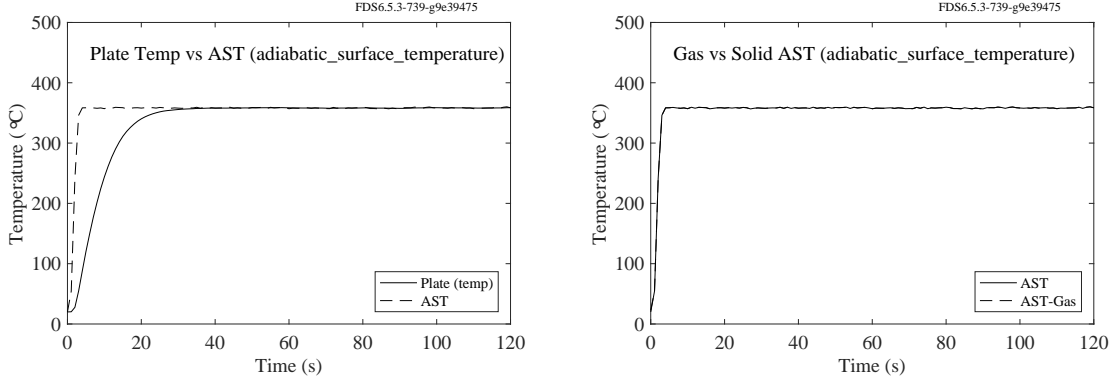


Figure 18.4: (Left) Surface temperature vs. adiabatic surface temperature of an insulated plate. (Right) AST recorded with a device positioned on the plate surface (AST) and one just off the surface (AST-Gas).

18.10.7 Special Topic: Detailed Spray Properties

Detailed experimental measurements of sprays using Phase Doppler Particle Analysis (PDPA) provide information on the droplet size distribution, speed and concentration. A special device type is defined via a `DEVC` line to simulate the PDPA measurement. The actual quantity to measure, and the details of the measurement are defined using an associated `PROP` line.

By default, the PDPA device output at time t is computed as a time integral

$$F(t) = \frac{1}{\min(t, t_e) - t_s} \int_{t_s}^{\min(t, t_e)} f(t) dt \quad (18.15)$$

but instantaneous values can be obtained by setting `PDPA_INTEGRATE` equal to `.FALSE.` on the corresponding `PROP` line, in which case

$$F(t) = f(t) \quad (18.16)$$

The function $f(t)$ has two forms:

$$f_1(t) = \left(\frac{\sum_i n_i D_i^m \phi}{\sum_i n_i D_i^n} \right)^{\frac{1}{m-n}} ; \quad f_2(t) = \frac{\sum_i n_i \phi}{V} \quad (18.17)$$

where n_i is the number of real particles represented by the single simulated particle, D_i is the particle diameter, and ϕ is the quantity to be measured. In each case, the summation goes over all the particles within a sphere with radius `PDPA_RADIUS` and centered at the location given by the device `XYZ`.

The first form $f_1(t)$ is used for the computation of various mean diameters, with associated properties defined using the following keywords on the `PROP` line:

`PDPA_M` m , exponent m of diameter.

`PDPA_N` n , exponent n of diameter. In case $m = n$, the exponent $1/(m - n)$ is removed from the formula.

The second form ($f_2(t)$) is used for the computation of mass and energy related variables that do not include the diameter weighting. The concentrations are based on the sampling volume, V , defined by `PDPA_RADIUS`. The quantity used for x can be chosen with the keyword `QUANTITY`. A summary of the available PDPA quantities is shown in Table 18.1.

Table 18.1: Output quantities available for PDPA.

QUANTITY	ϕ	f	Unit
'DIAMETER' (default)	1	f_1	μm
'ENTHALPY'	$(4/3)\rho_i r_i^3 (c_{p,i}(T_i)T_i - c_{p,i}(T_m)T_m)$	f_2	kJ/m^3
'PARTICLE FLUX X'	$(4/3)\rho_i r_i^3 u_i$	f_2	$\text{kg/m}^2\text{s}$
'PARTICLE FLUX Y'	$(4/3)\rho_i r_i^3 v_i$	f_2	$\text{kg}/(\text{m}^2 \cdot \text{s})$
'PARTICLE FLUX Z'	$(4/3)\rho_i r_i^3 w_i$	f_2	$\text{kg}/(\text{m}^2 \cdot \text{s})$
'U-VELOCITY'	u_i	f_1	m/s
'V-VELOCITY'	v_i	f_1	m/s
'W-VELOCITY'	w_i	f_1	m/s
'VELOCITY'	$(u_i^2 + v_i^2 + w_i^2)^{1/2}$	f_1	m/s
'TEMPERATURE'	T_i	f_1	$^\circ\text{C}$
'MASS CONCENTRATION'	$(4/3)\rho r_i^3$	f_2	kg/m^3
'NUMBER CONCENTRATION'	1	f_2	

* T_m is the melting temperature of the associated species.

It is also possible to output histograms of PDPA output quantities. When `PDPA_HISTOGRAM` is set to `.TRUE.`, normalized histogram bin counts are output to a comma-separated value (.csv) file from all devices associated with this `PROP` line. The number of bins and the limits of the histogram are controlled by parameters on the `PROP` line. The value used in creating the histogram is $D_i^m \phi$. Note that when making a histogram of diameters, the limits must be given in meters, not microns. Values falling outside the histogram limits are included in the counts of the first and last bins. Cumulative distributions can be output by setting `PDPA_HISTOGRAM_CUMULATIVE=.TRUE.` on the `PROP` line. To output unnormalized counts or cumulative distribution, set `PDPA_NORMALIZE` to `.FALSE.`. Note, however, that the counts correspond to the super droplets/particles, not the numerical ones. Due to the stratified sampling technique used (see Section 15.3.3), the counts are not necessarily integers.

The histogram output file contains of two-columns for each device. The first column gives the bin centers, and the second column gives the normalized bin count. The bin counts are normalized so that the total area of the histogram is 1. Here the histogram consists of equal width bars centered at the given bin centers and heights equal to the corresponding bin count. Volume and area based distributions can be output by setting the `PDPA_N` parameter on the `PROP` line to 3 and 2 respectively. Notice that this works differently from the mean diameter computation where the weighting is based on the `PDPA_M` parameter.

The properties of the PDPA device are defined using the following keywords on the `PROP` line:

`PART_ID` Name of the particle group to limit the computation to. Do not specify to account for all particles.

`PDPA_START` t_s , starting time of time integration in seconds. PDPA output is always a running average over time. As the spray simulation may contain some initial transient phase, it may be useful to specify the starting time of data collection.

`PDPA_END` t_e , ending time of time integration in seconds.

`PDPA_INTEGRATE` A logical parameter for choosing between time integrated or instantaneous values. `.TRUE.` by default.

`PDPA_RADIUS` Radius (m) of the sphere, centered at the device location, inside which the particles are monitored.

PDPA_NORMALIZE Can be set `.FALSE.` to force $V = 1$ in the formula for $f_2(t)$, or to unnormalize the histogram output.

QUANTITY Specified on PROP line for choosing the variable ϕ .

PDPA_HISTOGRAM_NBINS Number of bins used for the histogram.

The following example is used to measure the Sauter mean diameter, D_{32} , of the particle type 'water drops', starting from time 5 s.

```
&PROP ID='pdpa_d32'
      PART_ID='water drops'
      PDPA_M=3
      PDPA_N=2
      PDPA_RADIUS=0.01
      PDPA_START=5. /
&DEVC XYZ=0.0,0.0,1.0, QUANTITY='PDPA', PROP_ID='pdpa_d32' /
```

The following example is used to write out a histogram of droplet size using 20 equally sized bins between 0 and 2000 μm .

```
&PROP ID='pdpa_d'
      PART_ID='water drops'
      QUANTITY="DIAMETER"
      PDPA_RADIUS=0.01
      PDPA_START=0.0
      PDPA_M=1
      PDPA_HISTOGRAM=.TRUE.
      PDPA_HISTOGRAM_NBINS=20
      PDPA_HISTOGRAM_LIMITS=0,2000E-6 /
&DEVC XYZ=0.0,0.0,1.0, QUANTITY='PDPA', PROP_ID='pdpa_d' /
```

18.10.8 Useful Solid Phase Outputs

In addition to the profile (PROP) output, there are various additional quantities that are useful for monitoring reacting surfaces. For example, 'WALL THICKNESS' gives the overall thickness of the solid surface element. 'SURFACE DENSITY' gives the overall mass per unit area for the solid surface element, computed as an integral of material density over wall thickness. Both quantities are available both as DEVC and BNDF.

Thermogravimetric Analysis (TGA) Output

Thermogravimetric Analysis or TGA is a bench-scale measurement in which a very small solid material sample is heated up at a constant rate. The results of a TGA measurement are presented in the form of a normalized mass and normalized mass loss rate. Analogous quantities can be output from FDS:

$$\begin{aligned} \text{'NORMALIZED MASS'} &= \frac{\sum_{\alpha} m''_{\alpha}(t)}{\sum_{\alpha} m''_{\alpha}(0)} \quad (\text{dimensionless}) \\ \text{'NORMALIZED MASS LOSS RATE'} &= \frac{\sum_{\alpha} \dot{m}''_{\alpha}(t)}{\sum_{\alpha} m''_{\alpha}(0)} \quad (1/\text{s}) \end{aligned}$$

Micro-Combustion Calorimetry (MCC) Output

Micro-Combustion Calorimetry or MCC is similar to TGA, except the vaporized gas is burned. The result is a normalized heat release rate:

$$' \text{NORMALIZED HEAT RELEASE RATE}' = \dot{m}_f''(t) \Delta H / \sum_{\alpha} \dot{m}_{\alpha}''(0) \quad (\text{W/g})$$

Note that \dot{m}_f'' is the mass flux of fuel and ΔH is the heat of combustion.

Differential Scanning Calorimetry (DSC) Output

Differential Scanning Calorimetry or DSC is a measurement of the rate of heat absorption by a small material sample under constant heating. The result is a normalized heat absorption rate:

$$' \text{NORMALIZED HEATING RATE}' = \dot{q}_c''(t) / \sum_{\alpha} \dot{m}_{\alpha}''(0) \quad (\text{W/g})$$

Note that it is assumed that the sample is heated purely by convection, in which case \dot{q}_c'' is the convective heat flux.

18.10.9 Fractional Effective Dose (FED) and Fractional Irritant Concentration (FIC)

The Fractional Effective Dose index (FED), developed by Purser [?], is a commonly used measure of human incapacitation due to the combustion gases. The FED value is calculated as

$$\text{FED}_{\text{tot}} = (\text{FED}_{\text{CO}} + \text{FED}_{\text{CN}} + \text{FED}_{\text{NO}_x} + \text{FLD}_{\text{irr}}) \times \text{HV}_{\text{CO}_2} + \text{FED}_{\text{O}_2} \quad (18.18)$$

The fraction of an incapacitating dose of CO is calculated as

$$\text{FED}_{\text{CO}} = \int_0^t 2.764 \times 10^{-5} (C_{\text{CO}}(t))^{1.036} dt \quad (18.19)$$

where t is time in minutes and C_{CO} is the CO concentration (ppm). The fraction of an incapacitating dose of CN is calculated as

$$\text{FED}_{\text{CN}} = \int_0^t \left(\frac{1}{220} \exp \left(\frac{C_{\text{CN}}(t)}{43} \right) - 0.0045 \right) dt \quad (18.20)$$

where t is time in minutes and C_{CN} is the concentration (ppm) of HCN corrected for the protective effect of NO_2 . C_{CN} is calculated as

$$C_{\text{CN}} = C_{\text{HCN}} - C_{\text{NO}_2} - C_{\text{NO}} \quad (18.21)$$

The fraction of an incapacitating dose of NO_x is calculated as

$$\text{FED}_{\text{NO}_x} = \int_0^t \frac{C_{\text{NO}_x}(t)}{1500} dt \quad (18.22)$$

where t is time in minutes and C_{NO_x} is the sum of NO and NO_2 concentrations (ppm).

The Fractional Lethal Dose (FLD) of irritants is calculated as

$$\text{FLD}_{\text{irr}} = \int_0^t \left(\frac{C_{\text{HCl}}(t)}{F_{\text{FLD,HCl}}} + \frac{C_{\text{HBr}}(t)}{F_{\text{FLD,HBr}}} + \frac{C_{\text{HF}}(t)}{F_{\text{FLD,HF}}} + \frac{C_{\text{SO}_2}(t)}{F_{\text{FLD,SO}_2}} + \frac{C_{\text{NO}_2}(t)}{F_{\text{FLD,NO}_2}} + \frac{C_{\text{C}_3\text{H}_4\text{O}}(t)}{F_{\text{FLD,C}_3\text{H}_4\text{O}}} + \frac{C_{\text{CH}_2\text{O}}(t)}{F_{\text{FLD,CH}_2\text{O}}} \right) dt \quad (18.23)$$

Table 18.2: Coefficients used for the computation of irritant effects of gases.

	HCl	HBr	HF	SO ₂	NO ₂	C ₃ H ₄ O	CH ₂ O
F _{FLD} (ppm × min)	114000	114000	87000	12000	1900	4500	22500
F _{FIC} (ppm)	900	900	900	120	350	20	30

where t is time in minutes, the nominators are the instantaneous concentrations (ppm) of each irritant and the denominators the exposure doses of respective irritants predicted to be lethal to half the population. The lethal exposure doses [?] are given in Table 18.2. To include the effect of an irritant gas not listed in the table, you should specify F_{FLD} in ppm×min using the FLD_LETHAL_DOSE property of the corresponding SPEC line.

The fraction of an incapacitating dose of low O₂ hypoxia is calculated as

$$\text{FED}_{\text{O}_2} = \int_0^t \frac{dt}{\exp[8.13 - 0.54(20.9 - C_{\text{O}_2}(t))]} \quad (18.24)$$

where t is time in minutes and C_{O_2} is the O₂ concentration (volume percent). The hyperventilation factor induced by carbon dioxide is calculated as

$$\text{HV}_{\text{CO}_2} = \frac{\exp(0.1903 C_{\text{CO}_2}(t) + 2.0004)}{7.1} \quad (18.25)$$

where t is time in minutes and C_{CO_2} is the CO₂ concentration (percent).

The Fractional Irritant Concentration (FIC), also developed by Purser [?], represents the toxic effect which depends upon the immediate concentrations of irritants. The overall irritant concentration FIC is calculated as

$$\text{FIC}_{\text{irr}} = \frac{C_{\text{HCl}}(t)}{F_{\text{FIC,HCl}}} + \frac{C_{\text{HBr}}(t)}{F_{\text{FIC,HBr}}} + \frac{C_{\text{HF}}(t)}{F_{\text{FIC,HF}}} + \frac{C_{\text{SO}_2}(t)}{F_{\text{FIC,SO}_2}} + \frac{C_{\text{NO}_2}(t)}{F_{\text{FIC,NO}_2}} + \frac{C_{\text{C}_3\text{H}_4\text{O}}(t)}{F_{\text{FIC,C}_3\text{H}_4\text{O}}} + \frac{C_{\text{CH}_2\text{O}}(t)}{F_{\text{FIC,CH}_2\text{O}}} \quad (18.26)$$

where the nominators are the instantaneous concentrations of each irritant and the denominators the concentrations of respective irritants expected to cause incapacitation in half the population. The incapacitating concentrations [?] are given in Table 18.2. To include the irritant effect of a gas not listed in the table, you should specify F_{FIC} in ppm using the FIC_CONCENTRATION property on the corresponding SPEC line.

Note that the spatial integration features (Section 18.10.10) cannot be used with FED output because FED makes use of the 'TIME INTEGRAL' statistic (Section 18.10.11). For the same reason, FED output is only available as a point measurement.

18.10.10 Spatially-Integrated Outputs

A useful feature of a device (DEVC) is to specify an output quantity along with a desired statistic. For example,

```
&DEVC XB=..., QUANTITY='TEMPERATURE', ID='maxT', STATISTICS='MAX' /
```

causes FDS to write out the maximum gas phase temperature over the volume bounded by XB. Note that it does not compute the maximum over the entire computational domain, just the specified volume, and this volume must lie within a single mesh. Other STATISTICS are discussed below. Note that some are appropriate for gas phase output quantities, some for solid phase, and some for both.

For solid phase output quantities, like heat fluxes and surface temperatures, the specification of a SURF_ID along with the appropriate statistic limits the calculation to only those surfaces. You can further limit the search by using the sextuplet of coordinates XB to force FDS to only compute statistics for surface cells within the given volume. Be careful to account for the fact that the solid surface might shift to conform to the underlying numerical grid. Also, be careful not to specify a volume that extends beyond a single mesh. Note that you do not (and should not) specify an orientation via the parameter IOR when using a spatial statistic. IOR is only needed to find a specific point on the solid surface.

Use the STATISTICS feature with caution because it demands that FDS evaluate the given QUANTITY in all gas or solid phase cells.

Minimum or Maximum Value

For a given gas phase scalar output quantity defined at the center of each grid cell, ϕ_{ijk} , STATISTICS='MIN' or STATISTICS='MAX' computes the minimum or maximum value, respectively

$$\min_{ijk} \phi_{ijk} \quad ; \quad \max_{ijk} \phi_{ijk} \quad (18.27)$$

over the cells that are included in the specified volume bounded by XB. Note that this statistic is only appropriate for gas phase quantities. Note also that you must specify a volume to sum over via the coordinate parameters, XB, all of which must be contained within the same mesh.

Average Value

For a given gas phase scalar output quantity defined at the center of each grid cell, ϕ_{ijk} , STATISTICS='MEAN' computes the average value,

$$\frac{1}{N} \sum_{ijk} \phi_{ijk} \quad (18.28)$$

over the cells that are included in the specified volume bounded by XB. Note that this statistic is only appropriate for gas phase quantities. Note also that you must specify a volume to sum over via the coordinate parameters, XB, all of which must be contained within the same mesh.

Volume-Weighted Mean

For a given gas phase output quantity, $\phi(x,y,z)$, STATISTICS='VOLUME MEAN' produces the discrete analog of

$$\frac{1}{V} \int \phi(x,y,z) \, dx \, dy \, dz \quad (18.29)$$

which is very similar to 'MEAN', but it weights the values according to the relative size of the mesh cell. Note that this statistic is only appropriate for gas phase quantities. Note also that you must specify a volume to sum over via the coordinate parameters, XB, all of which must be contained within the same mesh.

Mass-Weighted Mean

For a given gas phase output quantity, $\phi(x,y,z)$, STATISTICS='MASS MEAN' produces the discrete analog of

$$\frac{\int \rho(x,y,z) \phi(x,y,z) \, dx \, dy \, dz}{\int \rho \, dx \, dy \, dz} \quad (18.30)$$

which is similar to 'VOLUME MEAN', but it weights the values according to the relative mass of the mesh cell. Note that this statistic is only appropriate for gas phase quantities. Note also that you must specify

a volume to sum over via the coordinate parameters, XB, all of which must be contained within the same mesh.

Volume Integral

For a given gas phase output quantity, $\phi(x,y,z)$, STATISTICS='VOLUME INTEGRAL' produces the discrete analog of

$$\int \phi(x,y,z) \, dx \, dy \, dz \quad (18.31)$$

Note that this statistic is only appropriate for gas phase quantities, in particular those whose units involve m^{-3} . For example, heat release rate per unit volume is an appropriate output quantity. Note also that you must specify a volume to sum over via the coordinate parameters, XB, all of which must be contained within the same mesh.

Mass Integral

For a given gas phase output quantity, $\phi(x,y,z)$, STATISTICS='MASS INTEGRAL' produces the discrete analog of

$$\int \rho(x,y,z) \phi(x,y,z) \, dx \, dy \, dz \quad (18.32)$$

Note that this statistic is only appropriate for gas phase quantities. Note also that you must specify a volume to sum over via the coordinate parameters, XB, all of which must be contained within the same mesh.

Area Integral

For a given gas phase output quantity, $\phi(x,y,z)$, STATISTICS='AREA INTEGRAL' produces the discrete analog of

$$\int \phi(x,y,z) \, dA \quad (18.33)$$

where dA depends on the coordinates you specify for XB. Note that this statistic is only appropriate for gas phase quantities, in particular those whose units involve m^{-2} . For example, the quantity 'MASS FLUX X' along with SPEC_ID='my gas' is an appropriate output quantity if you want to know the mass flux of the gas species that you have named 'my gas' through an area normal to the x direction. Note also that you must specify an area to sum over via the coordinate parameters, XB, all of which must be contained within the same mesh.

Surface Integral

For a given solid phase output quantity, ϕ , STATISTICS='SURFACE INTEGRAL' produces the discrete analog of

$$\int \phi \, dA \quad (18.34)$$

Note that this statistic is only appropriate for solid phase quantities, in particular those whose units involve m^{-2} . For example, the various heat and mass fluxes are appropriate output quantities.

Limiting the Integration

The input parameter `QUANTITY_RANGE` can be used to limit the region of integration for `STATISTICS` types of `'AREA INTEGRAL'`, `'VOLUME INTEGRAL'`, `'MASS INTEGRAL'`, and `'SURFACE INTEGRAL'`. If `QUANTITY_RANGE` is set, the integration will only be performed if the value of the `QUANTITY` lies within the `QUANTITY_RANGE` where `QUANTITY_RANGE(1)` is the lower bound and `QUANTITY_RANGE(2)` is the upper bound. For example:

```
&DEVC XB=..., QUANTITY='MASS FRACTION', SPEC_ID='METHANE', STATISTICS='MASS  
INTEGRAL', QUANTITY_RANGE=0.03,0.15/
```

would output the total mass of methane in the volume XB where the mass fraction was between 0.03 and 0.15.

A set of additional `STATISTICS` are available for use with `QUANTITY_RANGE`: `'AREA'`, `'MASS'`, `'VOLUME'`, and `'SURFACE AREA'`. These `STATISTICS` output the area, mass, volume, or surface area (for a solid phase quantity) where the `QUANTITY` lies within the `QUANTITY_RANGE`. For example:

```
&DEVC XB=..., QUANTITY='NET HEAT FLUX', STATISTICS='SURFACE AREA',  
QUANTITY_RANGE(1)=10./
```

would output the total surface area in the volume XB where the net heat flux exceeds 10 kW/m².

Volume, Mass, and Heat Flow

The net flow of mass and energy into or out of a planar surface can be a useful diagnostic. There are several outputs that address these. All are prescribed via the device (`DEVC`) namelist group only. For example:

```
&DEVC XB=0.3,0.5,2.1,2.5,3.0,3.0, QUANTITY='MASS FLOW', ID='whatever' /
```

outputs the net integrated mass flux through the given planar area, oriented in the positive z direction, in this case. The three flows `'VOLUME FLOW'`, `'MASS FLOW'`, and `'HEAT FLOW'` are defined:

$$\dot{V} = \int \mathbf{u} \cdot d\mathbf{S} \quad (18.35)$$

$$\dot{m} = \int \rho \mathbf{u} \cdot d\mathbf{S} \quad (18.36)$$

$$\dot{q} = \int \rho c_p (T - T_\infty) \mathbf{u} \cdot d\mathbf{S} \quad (18.37)$$

The addition of a + or - to the `QUANTITY` names yields the integral of the flow in the positive or negative direction only. In other words, if you want to know the mass flow out of a compartment, use `'MASS FLOW +'` or `'MASS FLOW -'`, depending on the orientation of the door.

Volume, Mass, and Heat Flow at a Wall

The quantities `'VOLUME FLOW'`, `'MASS FLOW'`, and `'HEAT FLOW'` cannot be applied at a solid boundary. Instead, use the wall equivalents `'VOLUME FLOW WALL'`, `'MASS FLOW WALL'`, and `'HEAT FLOW WALL'`. These quantities require an `IOR` for the surface (positive points into the domain). If you want the mass flow of a particular gas species, include the `SPEC_ID` of that gas species. Note that, in order to be a more useful diagnostic for global energy balances, the heat flow at the wall is defined using the sensible

enthalpy of the mixture (composition taken at the wall) and is not relative to ambient conditions:

$$\dot{q}_w = \int \rho h_s(T) \mathbf{u} \cdot d\mathbf{S} \quad (18.38)$$

18.10.11 Temporally-Integrated Outputs

In addition to the spatial statistics, a time integral of an DEVC output can be computed by specifying `STATISTICS = 'TIME INTEGRAL'` on the DEVC line. This produces a discrete analog of

$$\int_{t_0}^t \phi(\tau) d\tau \quad (18.39)$$

Note that the spatial and time integrals can not be used simultaneously.

18.10.12 Statistical Outputs

Statistical quantities can also be generated for either point or line devices. Note that these are quantities only have significant meaning if the flow is steady-state. These quantities are estimated using a logarithmic averaging process. The weighting coefficient for the averaging is determined by the desired time interval for the device. For a point device this time interval starts at `STATISTICS_START` on the DEVC line and continues until `T_END`. For a line device is is determined by `DT_DEVC_LINE` on the DUMP line.

Root Mean Square

If you set `STATISTICS='RMS'` on the DEVC line, the output will be an unbiased estimate of the *root mean square* value:

$$\phi_{\text{rms}} = \sqrt{\frac{\sum_{i=1}^n (\phi_i - \bar{\phi})^2}{n-1}} \quad (18.40)$$

Covariance

If $u = U - \bar{U}$ and $v = V - \bar{V}$ are the deviations for two random variables, U and V , then an unbiased estimate of the *covariance* is given by

$$\overline{uv} = \frac{\sum_{i=1}^n (U_i - \bar{U})(V_i - \bar{V})}{n-1} \quad (18.41)$$

To output this statistic you must add a `QUANTITY2` to the device line and set `STATISTICS='COV'`. The following lines would create a line of devices and a single point device equivalent to the first point in the line:

```
&DUMP DT_DEVC_LINE=20/
&DEVC XB=X1,X2,Y1,Y2,Z1,Z2, QUANTITY='CELL W', QUANTITY2='TEMPERATURE',
      STATISTICS='COV', ID='wT-cov_line', POINTS=20/
&DEVC XYZ=X1,Y1,Z1, QUANTITY='CELL W', QUANTITY2='TEMPERATURE',
      STATISTICS='COV', ID='wT-cov_point', STATISTICS_DT=20/
```

Correlation Coefficient

Setting `STATISTICS='CORRcoef'` outputs the *correlation coefficient* given by

$$\rho_{uv} = \frac{\overline{uv}}{u_{\text{rms}} v_{\text{rms}}} \quad (18.42)$$

Here again you must add a `QUANTITY2` to the device line.

18.10.13 Wind and the Pressure Coefficient

In the field of wind engineering, a commonly used quantity is known as the `PRESSURE_COEFFICIENT`:

$$C_p = \frac{p - p_\infty}{\frac{1}{2} \rho_\infty U^2} \quad (18.43)$$

p_∞ is the ambient, or “free stream” pressure, and ρ_∞ is the ambient density. The parameter U is the free-stream wind speed, given as `CHARACTERISTIC_VELOCITY` on the `PROP` line

```
&BNDf QUANTITY='PRESSURE COEFFICIENT', PROP_ID='U' /
&DEVC ID='pressure tap', XYZ=..., IOR=2, QUANTITY='PRESSURE COEFFICIENT', PROP_ID='U'
/
&PROP ID='U', CHARACTERISTIC_VELOCITY=3.4 /
```

Thus, you can either paint values of C_p at all surface points, or create a single time history of C_p using a single device at a single point.

18.10.14 Near-wall Grid Resolution

Large-eddy simulations of boundary layer flows fall into two general categories: LES with near-wall resolution and LES with near-wall modeling (wall functions). FDS employs the latter. The wall models used in FDS are the Werner-Wengle wall model [?] for smooth walls and a rough wall log law for rough walls [?]. For the wall models to function properly, the grid resolution near the wall should fall within a certain range of y^+ , the nondimensional distance from the wall expressed in viscous units. To check this, you may add a boundary file output as follows:

```
&BNDf QUANTITY='YPLUS' /
```

The value of y^+ reported is half (since the velocity lives at the cell face center) the wall-normal cell dimension (δn) divided by the local viscous length scale, δ_v [?]:

$$y^+ = \frac{1}{2} \frac{\delta n}{\delta_v}; \quad \delta_v = \frac{\mu/\rho}{u_\tau}; \quad u_\tau = \sqrt{\tau_w/\rho}, \quad (18.44)$$

where $\tau_w = \mu \partial|\mathbf{u}|/\partial n$ is the viscous stress evaluated at the wall (τ_w is computed by the wall function, $|\mathbf{u}|$ is taken as an estimate of the streamwise velocity component near the wall); the quantity u_τ is the *friction velocity*. The friction velocity may also be output in a boundary file or via a device attached to a wall. For example:

```
&DEVC XYZ=1,0,0, QUANTITY='FRICTION VELOCITY', IOR=3, ID='u_tau' /
```

Wall functions for LES are still under development, but as a general guideline it is recommended that the first grid cell fall within the log layer: a value $y^+ = 30$ would be considered highly resolved, the upper limit

of the log region for statistically stationary boundary layers depends on the Reynolds number, and there are no hard rules for transient flows. Beyond $y^+ = 1000$ the first grid cell is likely to fall in the wake region of the boundary layer and may produce unreliable results. A reasonable target for practical engineering LES is $y^+ = \mathcal{O}(100)$.

18.10.15 Dry Volume and Mass Fractions

During actual experiments, species such as CO and CO₂ are typically measured “dry”; that is, the water vapor is removed from the gas sample prior to analysis. For easier comparison of FDS predictions with measured data, you can specify the logical parameter DRY on a DEVC line that reports the ‘MASS FRACTION’ or ‘VOLUME FRACTION’ of a species. For example, the first line reports the actual volume fraction of CO, and the second line reports the output of a gas analyzer in a typical experiment.

```
&DEVC ID='wet CO', XYZ=..., QUANTITY='VOLUME FRACTION', SPEC_ID='CARBON MONOXIDE'/
&DEVC ID='dry CO', XYZ=..., QUANTITY='VOLUME FRACTION', SPEC_ID='CARBON MONOXIDE',
      DRY=.TRUE. /
```

18.10.16 Aerosol and Soot Concentration

Currently there are three different device options for outputting aerosol concentration (e.g., soot concentration) from FDS. It is important to recognize what each device is outputting so that the proper selection can be made.

```
&DEVC ID='MF_SOOT', XYZ=..., QUANTITY='MASS FRACTION', SPEC_ID='SOOT'/
&DEVC ID='VF_SOOT', XYZ=..., QUANTITY='VOLUME FRACTION', SPEC_ID='SOOT'/
&DEVC ID='SOOT_VF', XYZ=..., QUANTITY='AEROSOL VOLUME FRACTION', SPEC_ID='SOOT' /
```

Specifying a DEVC with a ‘MASS FRACTION’ and a SPEC_ID of SOOT will output the mass fraction of soot in the gas phase. The quantity ‘VOLUME FRACTION’ and a SPEC_ID of SOOT will output the volume fraction of soot in the gas phase treating the soot as if it were an ideal gas. The quantity ‘AEROSOL VOLUME FRACTION’ and a SPEC_ID of SOOT will output the volume fraction of soot as if it were a solid particle in the computational cell based on the following equation,

$$f_v = \rho Y_a / \rho_a \quad (18.45)$$

where ρ is the local density, Y_a is the local mass fraction of the aerosol, and ρ_a is density of the aerosol defined using the SPEC input DENSITY_SOLID, which defaults to 1800 (kg/m³) for soot.

18.10.17 Gas Velocity

The gas velocity components, u , v , and w , can be output as slice (SLCF), point device (DEVC), isosurface (ISOF), or Plot3D quantities using the names ‘U-VELOCITY’, ‘V-VELOCITY’, and ‘W-VELOCITY’. The total velocity is specified as just ‘VELOCITY’. Normally, the velocity is always positive, but you can use the parameter VELO_INDEX with a value of 1, 2 or 3 to indicate that the velocity ought to have the same sign as u , v , or w , respectively. This is handy if you are comparing velocity predictions with measurements. For Plot3D files, add PLOT3D_VELO_INDEX (N) = . . . to the DUMP line, where N refers to the Plot3D quantity 1, 2, 3, 4 or 5.

18.10.18 Enthalpy

There are several outputs that report the enthalpy of the gas mixture. First, the 'SPECIFIC ENTHALPY' and the 'SPECIFIC SENSIBLE ENTHALPY' are defined:

$$h(T) = \Delta h_f^\circ + \int_{T_{\text{ref}}}^T c_p(T') dT' \quad ; \quad h_s(T) = \int_{T_{\text{ref}}}^T c_p(T') dT' \quad (18.46)$$

Both have units of kJ/kg. The quantities 'ENTHALPY' and 'SENSIBLE ENTHALPY' are ρh and ρh_s , respectively, in units of kJ/m³.

18.10.19 Computer Performance

There are several useful DEVC QUANTITY's that can help monitor the performance of your computer:

'ACTUATED SPRINKLERS' Number of activated sprinklers.

'CFL MAX' The maximum value of the CFL (Courant-Friedrichs-Lewy) number, the primary constraint on the time step, for the mesh in which the device is located. By default, the time step is chosen so that the CFL number remains within the range of 0.8 to 1.0. If you want to see the CFL number in each grid cell, use a slice (SLCF) file with QUANTITY='CFL' and CELL_CENTERED=.TRUE..

'CPU TIME' Elapsed CPU time since the start of the simulation, in seconds.

'ITERATION' Number of time steps completed at the given time of the simulation.

'NUMBER OF PARTICLES' Number of Lagrangian particles for the MESH in which the DEVC is located.

'TIME STEP' Duration of a simulation time step, δt , in seconds.

'VN MAX' The maximum value of the VN (Von Neumann) number, a secondary constraint on the time step, for the mesh in which the device is located. By default, the time step is chosen so that the VN number remains below 1. If you want to see the VN number in each grid cell, use a slice (SLCF) file with QUANTITY='VN' and CELL_CENTERED=.TRUE..

'WALL CLOCK TIME' Elapsed wall clock time since the start of the simulation, in seconds.

'WALL CLOCK TIME ITERATIONS' Elapsed wall clock time since the start of the time stepping loop, in seconds.

In addition, the following flags can be useful in monitoring the performance of an MPI calculation. They are typically used for debugging.

VELOCITY_ERROR_FILE If set to .TRUE. on the DUMP line, this parameter will cause FDS to create a file with a time history of the maximum error associated with the normal component of velocity at solid or interpolated boundaries.

MPI_TIMEOUT The amount of time, in seconds, to wait for messages sent via MPI (Message Passing Interface) before timing out. This parameter is set on the MISC line. The default value is 10 s. This parameter is only useful in forcing a job with deadlocked messages to finish and print out information about the lost message. It is very unlikely to solve a deadlock problem.

18.10.20 Output File Precision

There are several different output files that have the format of a comma-separated value (.csv) file. These files consist of real numbers in columns separated by commas. By default, the real numbers are formatted

-1.2345678E+123

To change the precision of the numbers, use `SIG_FIGS` on the `DUMP` line to indicate the number of significant figures in the mantissa (default is 8). Use `SIG_FIGS_EXP` to change the number of digits in the exponent (default is 3). Keep in mind that the precision of real numbers used internally in an FDS calculation is approximately 12, equivalent to 8 byte or double precision following conventional Fortran rules.

18.10.21 A Posteriori Mesh Quality Metrics

The quality of a particular simulation is most directly tied to grid resolution. Three output quantities are discussed here for measuring errors in the velocity and scalar fields. It should be noted that the link between these metrics and true simulation quality is still in the research phase. In other words, a good quality score is not sufficient to assure a good simulation (at the present time).

Measure of Turbulence Resolution

A scalar quantity referred to as the *measure of turbulence resolution* [?] is defined locally as

$$M(\mathbf{x}) = \frac{\langle k_{\text{sgs}} \rangle}{\langle \text{TKE} \rangle + \langle k_{\text{sgs}} \rangle} \quad (18.47)$$

where the resolved turbulent kinetic energy per unit mass is

$$\text{TKE} = \frac{1}{2}(\tilde{u}_i - \langle \tilde{u}_i \rangle)(\tilde{u}_i - \langle \tilde{u}_i \rangle) \quad (18.48)$$

TKE must be output via `DEVC` for each velocity component and its mean:

```
&DEVC ..., QUANTITY='U-VELOCITY' /
&DEVC ..., QUANTITY='U-VELOCITY', STATISTICS='MEAN' /
&DEVC ..., QUANTITY='V-VELOCITY' /
&DEVC ..., QUANTITY='V-VELOCITY', STATISTICS='MEAN' /
&DEVC ..., QUANTITY='W-VELOCITY' /
&DEVC ..., QUANTITY='V-VELOCITY', STATISTICS='MEAN' /
```

You must post-process these outputs to obtain TKE.

The subgrid kinetic energy is estimated from Deardorff's eddy viscosity model

$$k_{\text{sgs}} \approx \frac{1}{2}(\tilde{u}_i - \hat{\tilde{u}}_i)(\tilde{u}_i - \hat{\tilde{u}}_i) = (\mu_t / (\rho C_v \Delta))^2 \quad (18.49)$$

Here, \tilde{u}_i is the resolved LES velocity and $\hat{\tilde{u}}_i$ is test filtered at a scale 2Δ where Δ is the LES filter width (in FDS, $\Delta = \delta x$). In the bulk flow region, the model for the SGS fluctuations is taken from scale similarity [?]. Cross-term energy is ignored. Keep in the mind, however, that the test filter operation is ill-defined near walls, and so FDS employs constant Smagorinsky in these cells to compute the eddy viscosity.

To output an estimate of the subgrid kinetic energy per unit mass use

```
&DEVC ..., QUANTITY='SUBGRID KINETIC ENERGY' /
```


The concept behind the measure of turbulence resolution is illustrated in Figure 18.5. Notice that on the left the difference between the grid signal and the test signal is very small. On the right, the grid signal is highly turbulent and the corresponding test signal is much smoother. We infer then that the flow is under-resolved.

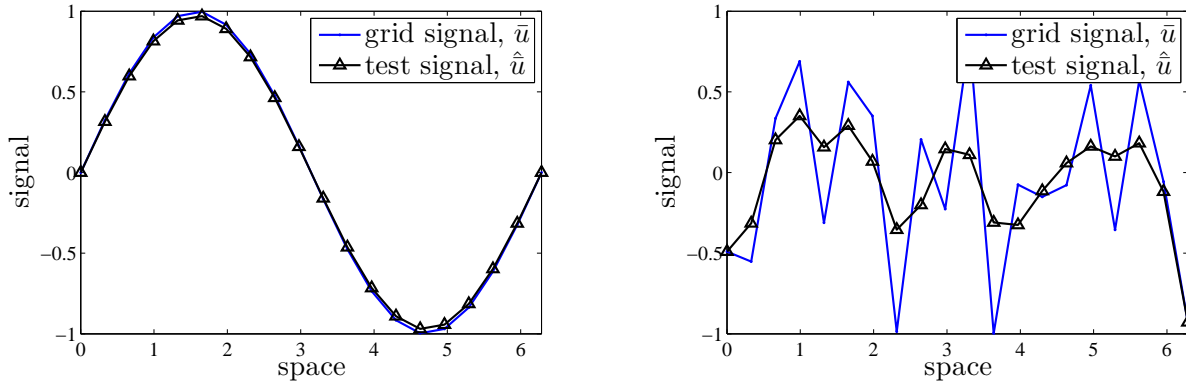


Figure 18.5: (Left) Resolved signal, M is small. (Right) Unresolved signal, M is close to unity.

For the canonical case of isotropic turbulence Pope actually defines LES such that $M < 0.2$. That is, LES requires resolution of 80% of the kinetic energy in the flow field (because this puts the grid Nyquist limit within the inertial subrange). The question remains as to whether this critical value is sufficient or necessary for a given engineering problem. As shown in Ref. [?], maintaining mean values of M near 0.2 indeed provides satisfactory results (simulation results within experimental error bounds) for mean velocities and species concentrations in nonreacting, buoyant plumes.

Wavelet Error Measure

A resolution metric that we call the *wavelet error measure* or WEM may be output using, for example,

```
&SLCF PBY=0, QUANTITY='WAVELET ERROR', QUANTITY2='HRRPUV' /
```

We begin by providing background on the simple Haar wavelet [?]. For a thorough and more sophisticated review of wavelet methods, the reader is referred to Schneider and Vasilyev [?].

Suppose the scalar function $f(r)$ is sampled at discrete points r_j , separated by a distance h , giving values s_j . Defining the *unit step function* over the interval $[r_1, r_2]$ by

$$\phi_{[r_1, r_2]} = \begin{cases} 1 & \text{if } r_1 \leq r < r_2 \\ 0 & \text{otherwise} \end{cases} \quad (18.50)$$

the simplest possible reconstruction of the signal is the step function approximation

$$f(r) \approx \sum_j s_j \phi_{[r_j, r_j+h]}(r) \quad (18.51)$$

By “viewing” the signal at a coarser resolution, say $2h$, an identical reconstruction of the function f over the interval $[r_j, r_j + 2h]$ may be obtained from

$$f_{[r_j, r_j+2h]}(r) = \underbrace{\frac{s_j + s_{j+1}}{2}}_a \phi_{[r_j, r_j+2h]}(r) + \underbrace{\frac{s_j - s_{j+1}}{2}}_c \psi_{[r_j, r_j+2h]}(r) \quad (18.52)$$

where a is as the *average* coefficient and c is as the *wavelet* coefficient. The Haar *mother wavelet* (Figure 18.6) is identified as

$$\psi_{[r_1, r_2]}(r) = \begin{cases} 1 & \text{if } r_1 \leq r < \frac{1}{2}(r_1 + r_2) \\ -1 & \text{if } \frac{1}{2}(r_1 + r_2) \leq r < r_2 \end{cases} \quad (18.53)$$

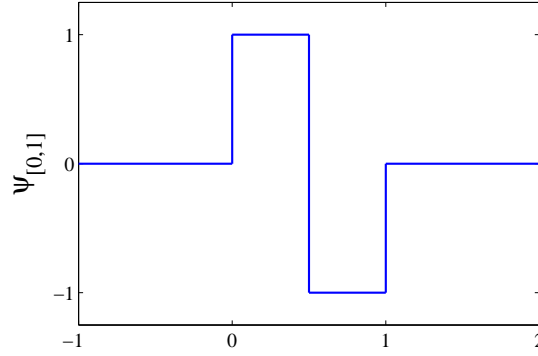


Figure 18.6: Haar mother wavelet on the interval $[0,1]$.

The decomposition of the signal shown in Eq. (18.52) may be repeated at ever coarser resolutions. The result is a *wavelet transform*. The procedure is entirely analogous to the Fourier transform, but with compact support. If we look at a 1D signal with 2^m points, the repeated application of (18.52) results in an $m \times m$ matrix of averages \mathbf{a} with components a_{ij} and an $m \times m$ wavelet coefficient matrix \mathbf{c} with components c_{ij} . Each row i of \mathbf{a} may be reconstructed from the $i+1$ row of \mathbf{a} and \mathbf{c} . Because of this and because small values of the wavelet coefficient matrix may be discarded, dramatic compression of the signal may be obtained.

Here we are interested in using the wavelet analysis to say something about the local level of error due to grid resolution. Very simply, we ask what can be discerned from a sample of four data points along a line. Roughly speaking we might expect to see one of the four scenarios depicted in Figure 18.7. Within each plot window we also show the results of a Haar wavelet transform for that signal. Looking first at the two top plots, on the left we have a step function and on the right we have a straight line. Intuitively, we expect large error for the step function and small error for the line. The following error measure achieves this goal:

$$\text{WEM}(\mathbf{x}, t) = \max_{x,y,z} \left(\frac{|c_{11} + c_{12}| - |c_{21}|}{|a_{21}|} \right) \quad (18.54)$$

Note that we have arbitrarily scaled the measure so that a step function leads to WEM of unity. In practice the transform is performed in all coordinate directions and the max value is reported. The scalar value may be output to Smokeview at the desired time interval.

Looking now at the two plots on the bottom of Figure 18.7, the signal on the left, which may indicate spurious oscillations or unresolved turbulent motion, leads to $\text{WEM} = 2$. Our measure therefore views this situation as the worst case in a sense. The signal to the lower right is indicative of an extremum, which actually is easily resolved by most centered spatial schemes and results again in $\text{WEM} = 0$.

In [?], the time average of WEM was reported for LES of a nonreacting buoyant plume at three grid resolutions. From this study, the best advice currently is to maintain average values of WEM less than 0.5.

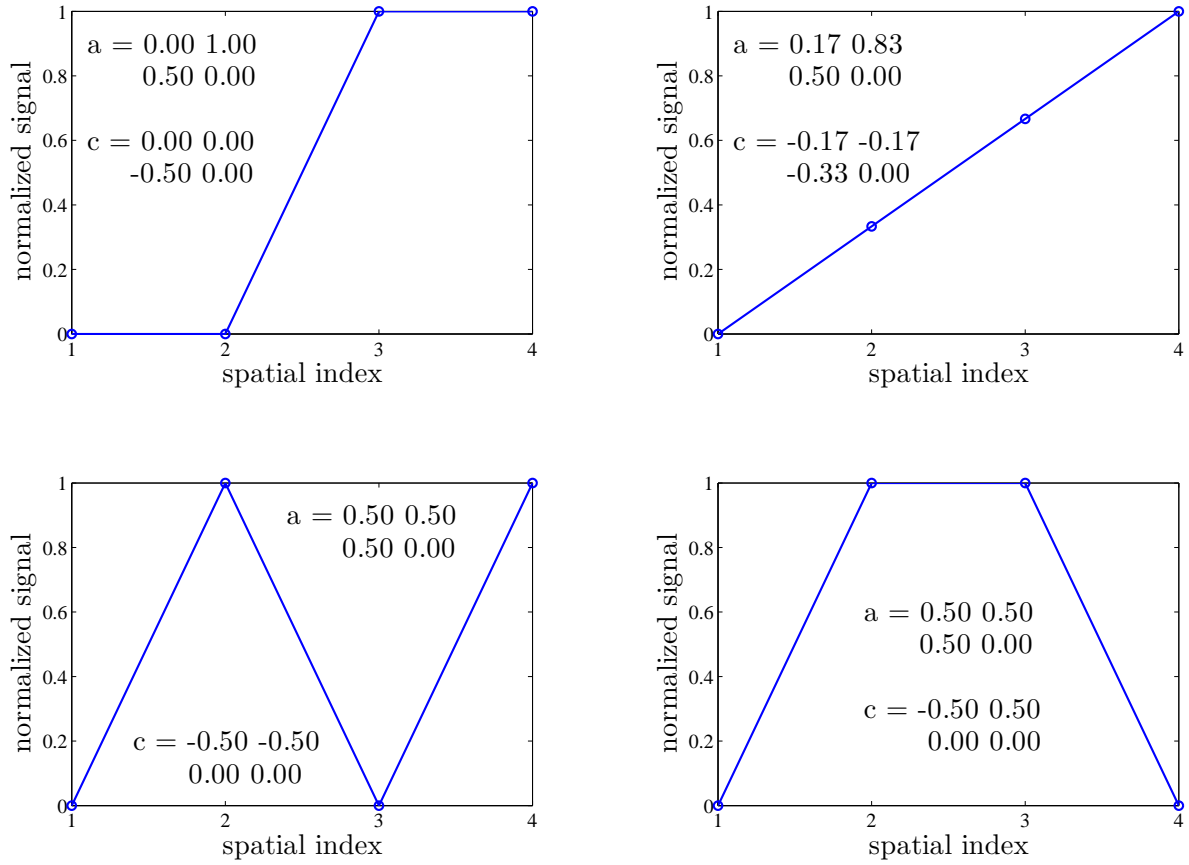


Figure 18.7: Averages and coefficients for local Haar wavelet transforms on four typical signals.

Local Cell Reynolds Number

Additionally, we provide an estimate of the *local cell Reynolds number* given by the ratio of the cell size (LES filter width, Δ) to an estimate of the local Kolmogorov scale, η (see [?]). For a DNS, Δ/η should be less than or equal to one. The Kolmogorov scale is computed from its definition:

$$\eta \equiv \left(\frac{(\mu/\rho)^3}{\varepsilon} \right)^{1/4} \quad (18.55)$$

where μ is the molecular dynamic viscosity, ρ is the density, and ε is the kinetic energy dissipation rate, which requires modeling. In FDS, we assume the dissipation rate is locally equivalent to the production of subgrid-scale kinetic energy. This implies

$$\varepsilon = (\mu_t/\rho)|\tilde{S}|^2 \quad (18.56)$$

where μ_t is the turbulent viscosity and $|\tilde{S}|$ is the filtered strain invariant (see FDS Tech Guide).

```
&SLCF PBX=0, QUANTITY='CELL REYNOLDS NUMBER' /
```

18.10.22 Extinction

In combustion, knowing if, when, or where chemical reactions have been extinguished is important. The output quantity `EXTINCTION` tells the user whether or not combustion has been prevented by the extinction routine. By default, `EXTINCTION = 0`, which means that the FDS extinction routine has not prevented combustion. An `EXTINCTION` value of 1 means that the routine has prevented combustion. The criteria for an `EXTINCTION` value of 1 is the presence of fuel and oxidizer without any energy release. An `EXTINCTION` value of -1 means that there is either no fuel or no oxidizer present.

18.11 Extracting Numbers from the Output Data Files

Often it is desired to present results of calculations in some form other than those offered by Smokeview. In this case, there is a short Fortran 90 program called `fds2ascii.f90`, with a PC compiled version called `fds2ascii.exe`. To run the program, just type:

```
fds2ascii
```

at the command prompt. You will be asked a series of questions about which type of output file to process, what time interval to time average the data, and so forth. A single file is produced with the name `CHID_fds2ascii.csv`. A typical command line session looks like this:

```
>> fds2ascii
  Enter Job ID string (CHID):
bucket_test_1
  What type of file to parse?
  PL3D file? Enter 1
  SLCF file? Enter 2
  BNDF file? Enter 3
3
  Enter Sampling Factor for Data?
  (1 for all data, 2 for every other point, etc.)
1
  Limit the domain size? (y or n)
y
  Enter min/max x, y and z
-5 5 -5 5 0 1
  1 MESH 1, WALL TEMPERATURE
  Enter starting and ending time for averaging (s)
35 36
  Enter orientation: (plus or minus 1, 2 or 3)
3
  Enter number of variables
1
  Enter boundary file index for variable 1
1
  Enter output file name:
bucket_test_1_fds2ascii.csv
  Writing to file...      bucket_test_1_fds2ascii.csv
```

These commands tell `fds2ascii` that you want to convert (binary) boundary file data into a text file. You want to sample every data point within the specified volume, you want only those surfaces that point upwards (+3 orientation), you only want 1 variable (only one is listed anyway and its index is 1 – that is just the number used to list the available files). The data will be time-averaged, and it will be output to a file

listed at the end of the session.

18.12 Summary of Frequently-Used Output Quantities

Table 18.3, spread over the following pages, summarizes the various Output Quantities. The column “File Type” lists the allowed output files for the quantities. “B” is for Boundary (BNDF), “D” is for Device (DEVC), “I” is for Iso-surface (ISOF), “P” is for Plot3D, “PA” for PArticle (PART), “S” is for Slice (SLCF). Be careful when specifying complicated quantities for Iso-surface or Plot3D files, as it requires computation in every gas phase cell.

For those output quantities that require a species name via SPEC_ID, the species implicitly defined when using the simple chemistry combustion model are 'OXYGEN', 'NITROGEN', 'WATER VAPOR', and 'CARBON DIOXIDE'. If CO_YIELD and/or SOOT_YIELD are specified on the REAC line, then 'CARBON MONOXIDE' and 'SOOT' are included as output species. The fuel species can be output via the FUEL specified on the REAC line. As an example of how to use the species names, suppose you want to calculate the integrated mass flux of carbon monoxide through a horizontal plane, like the total amount entrained in a fire plume. Use a “device” as follows

```
&DEVC ID='CO_flow', XB=-5,5,-5,5,2,2, QUANTITY='MASS FLUX Z',  
      SPEC_ID='CARBON MONOXIDE', STATISTICS='AREA INTEGRAL' /
```

Here, the ID is just a label in the output file. When an output quantity is related to a particular gas species or particle type, you must specify the appropriate SPEC_ID or PART_ID on the same input line. Also note that the use of underscores in output quantity names has been eliminated – just remember that all output quantity names ought to be in single quotes.

Table 18.3: Summary of frequently used output quantities.

QUANTITY	Symbol	Units	File Type
ABSORPTION COEFFICIENT	Section 14.2	1/m	D,I,P,S
ACTUATED SPRINKLERS	Section 18.10.19		D
ADIABATIC SURFACE TEMPERATURE	Section 18.10.6	°C	B,D
AEROSOL VOLUME FRACTION*	Section 18.10.16	mol/mol	D,I,P,S
AMPUA**	Section 18.9	kg/m ²	B,D
AMPUA_Z*	Section 18.9	kg/m ²	B,D
ASPIRATION	Section 17.3.7	%/m	D
BACKGROUND PRESSURE	Background pressure, \bar{p}	Pa	D,I,P,S
BACK WALL TEMPERATURE	Section 18.2.4	°C	B,D
BURNING RATE	Mass loss rate of fuel	kg/(m ² · s)	B,D
CHAMBER OBSCURATION	Section 17.3.5	%/m	D
CHI_R	Section 14.1.1		D,I,S
CONDUCTIVITY	Thermal conductivity	W/(m · K)	D,I,P,S
CONTROL	Section 17.5		D
CONTROL VALUE	Section 17.5		D
CONVECTIVE HEAT FLUX	Section 18.10.5	kW/m ²	B,D
CPUA**	Section 18.9	kW/m ²	B,D
CPUA_Z*	Section 18.9	kW/m ²	B,D
CPU TIME	Section 18.10.19	s	D
DENSITY	ρ or ρY_α with SPEC_ID	kg/m ³	D,I,P,S
DEPOSITION VELOCITY	Section 13.5	m/s	B,D
DIVERGENCE	$\nabla \cdot \mathbf{u}$	1/s	D,I,P,S
ENTHALPY	Section 12.1.2	kJ/m ³	D,I,P,S
EXTINCTION COEFFICIENT	Section 18.10.2	1/m	D,I,P,S
FED	Section 18.10.9		D
FIC	Section 18.10.9		D,S
FRICTION VELOCITY	Section 18.10.14	m/s	B,D
GAUGE HEAT FLUX	Section 18.10.5	kW/m ²	B,D
HEAT FLOW	Section 18.10.10	kW	D
HEAT FLOW WALL	Section 18.10.10	kW	D
NET HEAT FLUX	Section 18.10.5	kW/m ²	B,D
HRR	$\int \dot{q}''' dV$	kW	D
HRRPUA	\dot{q}''	kW/m ²	D
HRRPUV	\dot{q}'''	kW/m ³	D,I,P,S
INCIDENT HEAT FLUX	Section 18.10.5	kW/m ²	B,D
INSIDE WALL TEMPERATURE	Section 18.2.3	°C	D
INSIDE WALL DEPTH	Section 18.2.3	m	D
ITERATION	Section 18.10.19		D
LAYER HEIGHT	Section 18.10.3	m	D
LINK TEMPERATURE	Section 17.3.4	°C	D
LOWER TEMPERATURE	Section 18.10.3	°C	D
MASS FLOW	Section 18.10.10	kg/s	D
MASS FLOW WALL	Section 18.10.10	kg/s	D

Table 18.3: Summary of frequently used output quantities (continued).

QUANTITY	Symbol	Units	File Type
MASS FLUX*	Mass flux at solid surface	$\text{kg}/(\text{m}^2 \cdot \text{s})$	B,D
MASS FLUX X*	$\rho u Y_\alpha$	$\text{kg}/(\text{m}^2 \cdot \text{s})$	D,I,P,S
MASS FLUX Y*	$\rho v Y_\alpha$	$\text{kg}/(\text{m}^2 \cdot \text{s})$	D,I,P,S
MASS FLUX Z*	$\rho w Y_\alpha$	$\text{kg}/(\text{m}^2 \cdot \text{s})$	D,I,P,S
MASS FRACTION*	Y_α	kg/kg	D,I,P,S
MIXTURE FRACTION	Z	kg/kg	D,I,P,S
MPUA**	Section 18.9	kg/m^2	B,D
MPUA_Z*	Section 18.9	kg/m^2	B,D
MPUV**	Section 18.9	kg/m^3	D,P,S
MPUV_Z*	Section 18.9	kg/m^3	D,P,S
NORMAL VELOCITY	Wall normal velocity	m/s	D,B
NUMBER OF PARTICLES	Section 18.10.19		D
OPEN NOZZLES	Section 18.10.19		D
OPTICAL DENSITY	Section 18.10.2	1/m	D,I,P,S
PATH OBSCURATION	Section 17.3.6	%	D
PARTICLE AGE	Section 18.9	s	PA
PARTICLE DIAMETER	Section 18.9	μm	PA
PARTICLE FLUX X**	Section 18.9	$\text{kg}/(\text{m}^2 \cdot \text{s})$	P,S
PARTICLE FLUX Y**	Section 18.9	$\text{kg}/(\text{m}^2 \cdot \text{s})$	P,S
PARTICLE FLUX Z**	Section 18.9	$\text{kg}/(\text{m}^2 \cdot \text{s})$	P,S
PARTICLE MASS	Section 18.9	kg	PA
PARTICLE TEMPERATURE	Section 18.9	$^\circ\text{C}$	PA
PARTICLE VELOCITY	Section 18.9	m/s	PA
PRESSURE	Perturbation pressure, $\tilde{p} - p_\infty$	Pa	D,I,P,S
PRESSURE COEFFICIENT	Section 18.10.13		B,D
PRESSURE ZONE	Section 9.3		D,S
RADIATIVE HEAT FLUX	Section 18.10.5	kW/m^2	B,D
RADIATIVE HEAT FLUX GAS	Section 18.10.5	kW/m^2	D
RADIOMETER	Section 18.10.5	kW/m^2	B,D
RELATIVE HUMIDITY	Section 13.1.1	%	D,I,P,S
SENSIBLE ENTHALPY	Section 18.10.18	kJ/m^3	D,I,P,S
SOLID CONDUCTIVITY	Section 18.2.3	$\text{W}/(\text{m} \cdot \text{K})$	D
SOLID DENSITY	Section 18.2.3	kg/m^3	D
SOLID SPECIFIC HEAT	Section 18.2.3	$\text{kJ}/(\text{kg} \cdot \text{K})$	D
SPECIFIC ENTHALPY	Section 18.10.18	kJ/kg	D,I,P,S
SPECIFIC HEAT	c_p	$\text{kJ}/(\text{kg} \cdot \text{K})$	D,I,P,S
SPECIFIC SENSIBLE ENTHALPY	Section 18.10.18	kJ/kg	D,I,P,S
SPRINKLER LINK TEMPERATURE	Section 17.3.1	$^\circ\text{C}$	D
SURFACE DENSITY	Section 18.10.8	kg/m^2	B,D
SURFACE DEPOSITION*	Section 13.5	kg/m^2	B,D
TEMPERATURE	Section 18.10.4	$^\circ\text{C}$	D,I,P,S
THERMOCOUPLE	Section 18.10.4	$^\circ\text{C}$	D
TIME	Section 17.1	s	D

Table 18.3: Summary of frequently used output quantities (continued).

QUANTITY	Symbol	Units	File Type
TIME STEP	Section 18.10.19	s	D
TRANSMISSION	Section 17.3.6	%/m	D
U-VELOCITY	Gas velocity component, u	m/s	D,I,P,S
V-VELOCITY	Gas velocity component, v	m/s	D,I,P,S
W-VELOCITY	Gas velocity component, w	m/s	D,I,P,S
UPPER TEMPERATURE	Section 18.10.3	°C	D
VELOCITY***	Gas velocity	m/s	D,I,P,S
VISCOSITY	Effective viscosity, $\mu + \mu_t$	kg/(m · s)	D,I,P,S
VISIBILITY	Section 18.10.2	m	D,I,P,S
VOLUME FLOW	Section 18.10.10	m ³ /s	D
VOLUME FLOW WALL	Section 18.10.10	m ³ /s	D
VOLUME FRACTION****	X_α	mol/mol	D,I,P,S
WALL CLOCK TIME	Section 18.10.19	s	D
WALL CLOCK TIME ITERATIONS	Section 18.10.19	s	D
WALL TEMPERATURE	Surface temperature	°C	B,D
WALL THICKNESS	Section 18.10.8	m	B,D

* Quantity requires the specification of a gas species using SPEC_ID.

** Quantity requires the specification of a particle name using PART_ID.

*** Add VELO_INDEX=1 to the input line if you want to multiply the velocity by the sign of u .
Use the indices 2 and 3 for v and w , respectively.

**** Quantity requires the specification of a gas species using SPEC_ID.
Do not use for MIXTURE FRACTION.

18.13 Summary of Infrequently-Used Output Quantities

Table 18.4 below lists some less often used output quantities. These are mainly used for diagnostic purposes. Explanations for most can be found in Volume 1 of the FDS Technical Reference Guide [?].

Table 18.4: Summary of *infrequently* used output quantities.

QUANTITY	Symbol	Units	File Type
ADA**	Average Droplet (cross sectional) Area	m ² /m ³	D,I,P,S
ADA_Z*	Average Droplet (cross sectional) Area	m ² /m ³	D,I,P,S
ADD**	Average Droplet Diameter	μm	D,I,P,S
ADD_Z*	Average Droplet Diameter	μm	D,I,P,S
ADT**	Average Droplet Temperature	°C	D,I,P,S
ADT_Z*	Average Droplet Temperature	°C	D,I,P,S
C_SMAG	Smagorinsky coefficient		D,I,P,S
CABLE TEMPERATURE	Inner temperature of cable	°C	D
CELL INDEX I	Mesh cell index in x		D,S
CELL INDEX J	Mesh cell index in y		D,S
CELL INDEX K	Mesh cell index in z		D,S
CELL REYNOLDS NUMBER	Section 18.10.21		D,I,P,S
CELL U	$(u_{i,j,k} + u_{i-1,j,k})/2$	m/s	D,I,S
CELL V	$(v_{i,j,k} + v_{i,j-1,k})/2$	m/s	D,I,S
CELL W	$(w_{i,j,k} + w_{i,j,k-1})/2$	m/s	D,I,S
CFL	Section 18.10.19		D,I,P,S
CFL MAX	Section 18.10.19		D
CHEMICAL SUBITERATIONS	Section 13.4		D,S
DIFFUSIVITY*	Species diffusivity	m ² /s	D,I,S
EMISSIVITY	Surface emissivity (usually constant)		B,D
EXTINCTION	Section 18.10.22		D,S
F_X, F_Y, F_Z	Momentum terms, F_x, F_y, F_z	m/s ²	D,I,P,S
GAS DENSITY	Gas Density near wall	kg/m ³	B,D
GAS TEMPERATURE	Gas Temperature near wall	°C	B,D
H	$H = \mathbf{u} ^2/2 + \tilde{p}/\rho$	(m/s) ²	D,I,P,S
HEAT TRANSFER COEFFICIENT	Convective heat transfer	W/(m ² · K)	B,D
HRRPUL	$\int \dot{q}''' dx dy$	kW/m	D
INTEGRATED INTENSITY	$U = \int I ds$	kW/m ²	D,I,P,S
KINETIC ENERGY	$(u^2 + v^2 + w^2)/2$	(m/s) ²	D,I,P,S
KOLMOGOROV LENGTH SCALE	Section 18.10.21	m	D,I,P,S
MACH NUMBER	$ \mathbf{u} /\sqrt{(R/\bar{W})T\gamma}$		S,D
MASS FLUX WALL CELL	ρu_n at a wall cell face	kg/(m ² · s)	B,D
MAXIMUM VELOCITY ERROR	Section 6.6	m/s	D
MIXING TIME	Combustion mixing time	s	D,I,P,S
MOLECULAR VISCOSITY	Molecular viscosity, $\mu(\mathbf{Z}, T)$	kg/(m · s)	D,I,P,S
NORMALIZED HEATING RATE	Section 18.10.8	W/g	D
NORMALIZED HEAT RELEASE RATE	Section 18.10.8	W/g	D
NORMALIZED MASS	Section 18.10.8		D
NORMALIZED MASS LOSS RATE*	Section 18.10.8	1/s	D

Table 18.4: Summary of *infrequently* used output quantities (continued).

QUANTITY	Symbol	Units	File Type
PARTICLE PHASE	Orientation of droplet		PA
PARTICLE RADIATION LOSS	$\nabla \cdot \mathbf{q}_r''$ due to Lagrangian particles	kW/m ³	D,I,P,S
PDPA	Droplet diagnostics		D
PRESSURE ITERATIONS	No. pressure iterations		D
QABS**	Absorption efficiency of droplets		D,I,P,S
QABS_Z*	Absorption efficiency of droplets		D,I,P,S
QSCA**	Scattering efficiency of droplets		D,I,P,S
QSCA_Z*	Scattering efficiency of droplets		D,I,P,S
RADIATION LOSS	$\nabla \cdot \mathbf{q}_r''$	kW/m ³	D,I,P,S
STRAIN RATE	$2(S_{ij}S_{ij} - 1/3(\nabla \cdot \mathbf{u})^2)$	1/s	D,I,P,S
STRAIN RATE X	$\partial w / \partial y + \partial v / \partial z$	1/s	D,I,P,S
STRAIN RATE Y	$\partial u / \partial z + \partial w / \partial x$	1/s	D,I,P,S
STRAIN RATE Z	$\partial v / \partial x + \partial u / \partial y$	1/s	D,I,P,S
SUBGRID KINETIC ENERGY	Section 18.10.21	m ² /s ²	D,S
SUM LUMPED MASS FRACTIONS	$\sum_i Z_i$ (should be 1)		D,S
SUM PRIMITIVE MASS FRACTIONS	$\sum_\alpha Y_\alpha$ (should be 1)		D,S
VELOCITY ERROR	Section 6.6		B
VN	Section 18.10.19		D,I,P,S
VN MAX	Section 18.10.19		D
VORTICITY X	$\partial w / \partial y - \partial v / \partial z$	1/s	D,I,P,S
VORTICITY Y	$\partial u / \partial z - \partial w / \partial x$	1/s	D,I,P,S
VORTICITY Z	$\partial v / \partial x - \partial u / \partial y$	1/s	D,I,P,S
WALL VISCOSITY	Near-wall viscosity, μ_w	kg/(m · s)	B,D
WAVELET ERROR***	Section 18.10.21		S
YPLUS	Section 18.10.14		B,D

* Quantity requires the specification of a gas species using SPEC_ID.

** Quantity requires the specification of a particle name using PART_ID.

*** Quantity requires specification of an additional scalar using QUANTITY2.

18.14 Summary of HVAC Output Quantities

Table 18.5 summarizes the various Output Quantities for HVAC systems. Quantities for a duct require the specification of a `DUCT_ID`, and quantities for a node require the specification of a `NODE_ID`. Mass and volume fraction outputs also require the specification of a `SPEC_ID`.

Table 18.5: Summary of HVAC output quantities.

QUANTITY	Symbol	Units
AIRCOIL HEAT EXCHANGE	Heat exchange rate for an aircoil	kW
DUCT DENSITY	Density of the flow in a duct	kg/m ³
DUCT MASS FLOW	Mass flow in a duct	kg/s
DUCT MASS FRACTION	Mass fraction of a species in a duct	kg/kg
DUCT TEMPERATURE	Temperature of the flow in a duct	°C
DUCT VELOCITY	Velocity of a duct	m/s
DUCT VOLUME FLOW	Volumetric flow in a duct	m ³ /s
DUCT VOLUME FRACTION	Volume fraction of a species in a duct	mol/mol
FAN PRESSURE	Pressure output of a fan in a duct	Pa
FILTER LOADING	Loading of a species in a filter	kg
FILTER LOSS	Flow loss through a filter	
NODE DENSITY	Density of the flow through a node	kg/m ³
NODE MASS FRACTION	Mass fraction of a species in a node	kg/kg
NODE PRESSURE	Pressure of a node	Pa
NODE PRESSURE DIFFERENCE	Pressure difference between two nodes	Pa
NODE TEMPERATURE	Temperature of the flow though a node	°C
NODE VOLUME FRACTION	Volume fraction of a species in a node	mol/mol

Chapter 19

Alphabetical List of Input Parameters

This appendix lists all of the input parameters for FDS in separate tables grouped by namelist, these tables are in alphabetical order along with the parameters within them. This is intended to be used as a quick reference and does not replace reading the detailed description of the parameters in the main body of this guide. See Table 5.1 for a cross-reference of relevant sections and the tables in this appendix. The reason for this statement is that many of the listed parameters are mutually exclusive – specifying more than one can cause the program to either fail or run in an unpredictable manner. Also, some of the parameters trigger the code to work in a certain mode when specified. For example, specifying the thermal conductivity of a solid surface triggers the code to assume the material to be thermally-thick, mandating that other properties be specified as well. Simply prescribing as many properties as possible from a handbook is bad practice. Only prescribe those parameters which are necessary to describe the desired scenario. Note that you may use the character string `FYI` on any namelist line to make a note or comment.

19.1 BNDF (Boundary File Parameters)

Table 19.1: For more information see Section 18.5.

BNDF (Boundary File Parameters)				
CELL_CENTERED	Logical	Section 18.5		.FALSE.
PART_ID	Character	Section 18.12		
PROP_ID	Character	Section 18.5		
QUANTITY	Character	Section 18.12		
SPEC_ID	Character	Section 18.12		
STATISTICS	Character	Section 18.5		

19.2 CLIP (Clipping Parameters)

Table 19.2: For more information see Section 6.7.

CLIP (Specified Upper and Lower Limits)				
MAXIMUM_DENSITY	Real	Section 6.7	kg/m ³	
MAXIMUM_TEMPERATURE	Real	Section 6.7	°C	
MINIMUM_DENSITY	Real	Section 6.7	kg/m ³	
MINIMUM_TEMPERATURE	Real	Section 6.7	°C	

19.3 CSVF (Comma Separated Velocity Files)

Table 19.3: For more information see Section 6.4.4.

CSVF (Comma Delimited Output Files)				
UVWFILE	Character	Section 6.4.4		

19.4 CTRL (Control Function Parameters)

Table 19.4: For more information see Section 17.5.

CTRL (Control Function Parameters)				
CONSTANT	Real	Section 17.5.6		
DELAY	Real	Section 17.5.9	s	0.
DIFFERENTIAL_GAIN	Real	Section 17.5.7		0.
EVACUATION	Logical	Reference [?]		.FALSE.
FUNCTION_TYPE	Character	Section 17.4		
ID	Character	Section 17.5		

Table 19.4: Continued

CTRL (Control Function Parameters)				
INITIAL_STATE	Logical	Section 17.4		.FALSE.
INPUT_ID	Char. Array	Section 17.5		
INTEGRAL_GAIN	Real	Section 17.5.7		0.
LATCH	Logical	Section 17.4		.TRUE.
N	Integer	Section 17.5		1
ON_BOUND	Character	Section 17.5.3		LOWER
PROPORTIONAL_GAIN	Real	Section 17.5.7		0.
RAMP_ID	Character	Section 17.5.5		
SETPOINT (2)	Real	Section 17.4		
TARGET_VALUE	Real	Section 17.5.7		0.
TRIP_DIRECTION	Integer	Section 17.4		1

19.5 DEVC (Device Parameters)

Table 19.5: For more information see Section 17.1.

DEVC (Device Parameters)				
BYPASS_FLOWRATE	Real	Section 17.3.7	kg/s	0
CONVERSION_FACTOR	Real	Section 17.2		1
COORD_FACTOR	Real	Section 17.2		1
CTRL_ID	Character	Section 17.6.1		
DELAY	Real	Section 17.3.7	s	0
DEPTH	Real	Section 18.10.8	m	0
DEVC_ID	Character	Sections 17.3.7 and 17.6.1		
DRY	Logical	Section 18.10.15		.FALSE.
DUCT_ID	Character	Section 9.2		
EVACUATION	Logical	Reference [?]		.FALSE.
FLOWRATE	Real	Section 17.3.7	kg/s	0
HIDE_COORDINATES	Logical	Section 18.2.2		.FALSE.
ID	Character	Section 17.1		
INITIAL_STATE	Logical	Section 17.4		.FALSE.
INIT_ID	Character	Section 15.4		
IOR	Integer	Section 17.1		
LATCH	Logical	Section 17.4		.TRUE.
MATL_ID	Character	Section 18.10.8		
NODE_ID	Character(2)	Section 9.2		
NO_UPDATE_DEVC_ID	Character	Section 17.6.2		
NO_UPDATE_CTRL_ID	Character	Section 17.6.2		
ORIENTATION	Real Triplet	Section 17.1		0,0,-1
ORIENTATION_NUMBER	Integer	Section 18.9		1
OUTPUT	Logical	Section 17.2		.TRUE.
PART_ID	Character	Section 18.12		

Table 19.5: Continued

DEVC (Device Parameters)				
PIPE_INDEX	Integer	Section 17.3.1		1
POINTS	Integer	Section 18.2.2		1
PROP_ID	Character	Section 17.1		
QUANTITY	Character	Section 17.1		
QUANTITY2	Character	Section 18.2.2		
QUANTITY_RANGE	Real(2)	Section 18.10.10		-1.E50,1.E50
RELATIVE	Logical	Section 17.2		.FALSE.
R_ID	Character	Section 18.2.2		
ROTATION	Real	Section 17.1	deg.	0
SETPOINT	Real	Section 17.4		
STATISTICS	Character	Section 18.10.10		
STATISTICS_START	Real	Section 18.10.12	s	T_BEGIN
SMOOTHING_FACTOR	Real	Section 17.4		0
SPEC_ID	Character	Section 18.12		
SURF_ID	Character	Section 18.10.10		
TIME_AVERAGED	Logical	Section 17.2		.TRUE.
TIME_HISTORY	Logical	Section 18.2.2		
TRIP_DIRECTION	Integer	Section 17.4		1
UNITS	Character	Section 17.2		
VELO_INDEX	Integer	Section 18.10.17		0
XB (6)	Real Sextuplet	Section 18.10.10	m	
XYZ (3)	Real Triplet	Section 17.1	m	
X_ID	Character	Section 18.2.2		ID-x
Y_ID	Character	Section 18.2.2		ID-y
Z_ID	Character	Section 18.2.2		ID-z

19.6 DUMP (Output Parameters)

Table 19.6: For more information see Section 18.1.

DUMP (Output Parameters)				
CLIP_RESTART_FILES	Logical	Section 6.4.3		.TRUE.
COLUMN_DUMP_LIMIT	Logical	Section 17.2		.FALSE.
CTRL_COLUMN_LIMIT	Integer	Section 17.2		254
DEVC_COLUMN_LIMIT	Integer	Section 17.2		254
DT_BNDF	Real	Section 18.1	s	$2 \Delta t / \text{NFRAMES}$
DT_CPU	Real	Section 22.5	s	1000000
DT_CTRL	Real	Section 18.1	s	$\Delta t / \text{NFRAMES}$
DT_DEVC	Real	Section 18.1	s	$\Delta t / \text{NFRAMES}$
DT_DEVC_LINE	Real	Section 18.2.2	s	$\Delta t / 2$
DT_FLUSH	Real	Section 18.1	s	$\Delta t / \text{NFRAMES}$
DT_HRR	Real	Section 18.1	s	$\Delta t / \text{NFRAMES}$

Table 19.6: Continued

DUMP (Output Parameters)				
DT_ISO	Real	Section 18.1	s	Δt / NFRAMES
DT_MASS	Real	Section 18.1	s	Δt / NFRAMES
DT_PART	Real	Section 18.1	s	Δt / NFRAMES
DT_PL3D	Real	Section 18.1	s	1.E10
DT_PROF	Real	Section 18.1	s	Δt / NFRAMES
DT_RESTART	Real	Section 18.1	s	1000000.
DT_SL3D	Real	Section 18.1	s	Δt / 5
DT_SLCF	Real	Section 18.1	s	Δt / NFRAMES
EB_PART_FILE	Logical	Section 18.1		.FALSE.
FLUSH_FILE_BUFFERS	Logical	Section 18.1		.TRUE.
MASS_FILE	Logical	Section 18.1		.FALSE.
MAXIMUM_PARTICLES	Integer	Section 18.1		1000000
NFRAMES	Integer	Section 18.1		1000
PLOT3D_PART_ID (5)	Char. Quint	Section 18.7		
PLOT3D_QUANTITY (5)	Char. Quint	Section 18.7		
PLOT3D_SPEC_ID (5)	Char. Quint	Section 18.7		
PLOT3D_VELO_INDEX	Int. Quint	Section 18.10.17		0
RENDER_FILE	Character	Reference [?]		
SIG_FIGS	Integer	Section 18.10.20		8
SIG_FIGS_EXP	Integer	Section 18.10.20		3
SMOKE3D	Logical	Section 18.8		.TRUE.
SMOKE3D_QUANTITY	Character	Section 18.8		
SMOKE3D_SPEC_ID	Character	Section 18.8		
STATUS_FILES	Logical	Section 18.1		.FALSE.
SUPPRESS_DIAGNOSTICS	Logical	Section 3.3		.FALSE.
UVW_TIMER	Real Vector (10)	Section 6.4.4	s	
VELOCITY_ERROR_FILE	Logical	Section 18.10.19		.FALSE.
WRITE_XYZ	Logical	Section 18.7		.FALSE.

$$\Delta t = T_{\text{END}} - T_{\text{BEGIN}}$$

19.7 HEAD (Header Parameters)

Table 19.7: For more information see Section 6.1.

HEAD (Header Parameters)				
CHID	Character	Section 6.1		'output'
TITLE	Character	Section 18.7		

19.8 HOLE (Obstruction Cutout Parameters)

Table 19.8: For more information see Section 7.2.6.

HOLE (Obstruction Cutout Parameters)				
COLOR	Character	Section 7.4		
CTRL_ID	Character	Section 7.2.6		
DEVC_ID	Character	Section 7.2.6		
EVACUATION	Logical	Reference [?]		
ID	Character	Identifier for input line		
MESH_ID	Character	Reference [?]		
MULT_ID	Character	Section 7.5		
RGB (3)	Integer Triplet	Section 7.4		
TRANSPARENCY	Real	Section 7.2.6		
XB (6)	Real Sextuplet	Section 7.5	m	

19.9 HVAC (HVAC System Definition)

Table 19.9: For more information see Section 9.2.

HVAC (HVAC System Definition)				
AIRCOIL_ID	Character	Section 9.2.1		
AMBIENT	Logical	Section 9.2.3		.FALSE.
AREA	Real	Section 9.2.1	m ²	
CLEAN_LOSS	Real	Section 9.2.5		
COOLANT_MASS_FLOW	Real	Section 9.2.6	kg/s	
COOLANT_SPECIFIC_HEAT	Real	Section 9.2.6	kJ/(kg · K)	
COOLANT_TEMPERATURE	Real	Section 9.2.6	°C	
CTRL_ID	Character	Sections 9.2.1, 9.2.4, 9.2.5		
DAMPER	Logical	Sections 9.2.1, 9.2.2		.FALSE.
DEVC_ID	Character	Sections 9.2.1, 9.2.4, 9.2.5		
DIAMETER	Real	Section 9.2.1	m	
DUCT_ID	Character Array	Section 9.2.3		
DUCT_INTERP_TYPE	Character	Section 9.2.8		'NODE1'
EFFICIENCY	Real Array	Sections 9.2.5, 9.2.6		1.0
FAN_ID	Character	Section 9.2.1		
FILTER_ID	Character	Section 9.2.3		
FIXED_Q	Real	Section 9.2.6	kW	
ID	Character	Section 9.2		
LEAK_ENTHALPY	Logical	Section 9.3.2		.FALSE.
LENGTH	Real	Section 9.2.1	m	
LOADING	Real Array	Section 9.2.5	kg	0.0
LOADING_MULTIPLIER	Real Array	Section 9.2.5	l/kg	1.0
LOSS	Real Array	Sections 9.2.1 – 9.2.5		0.0

Table 19.9: Continued

HVAC (HVAC System Definition)				
MASS_FLOW	Real	Section 9.2.1	kg/s	
MAX_FLOW	Real	Section 9.2.4	m ³ /s	
MAX_PRESSURE	Real	Section 9.2.4	Pa	
N_CELLS	Integer	Section 9.2.8		LENGTH/0.1
NODE_ID	Character Doublet	Section 9.2.1		
PERIMETER	Real	Section 9.2.1	m	
RAMP_ID	Character	Sections 9.2.1, 9.2.5, 9.2.4		
RAMP_LOSS	Character	Sections 9.2.1, 9.2.2		
REVERSE	Logical	Section 9.2.1		.FALSE.
ROUGHNESS	Real	Section 9.2.1	m	0.0
SPEC_ID	Character Array	Section 9.2.5		
TAU_AC	Real	Section 9.2.6	s	1.0
TAU_FAN	Real	Section 9.2.4	s	1.0
TAU_VF	Real	Section 9.2.1	s	1.0
TYPE_ID	Character	Section 9.2		
VENT_ID	Character	Section 9.2.3		
VENT2_ID	Character	Section 9.3.2		
VOLUME_FLOW	Real	Section 9.2.1, 9.2.4	m ³ /s	
XYZ	Real Triplet	Section 9.2.3	m	0.0

19.10 INIT (Initial Conditions)

Table 19.10: For more information see Section 6.5.

INIT (Initial Conditions)				
AUTO_IGNITION_TEMPERATURE	Real	Section 13.1.4	°C	-273.15
CELL_CENTERED	Logical	Section 15.5.3		.FALSE.
CTRL_ID	Character	Section 15.5.3		
DENSITY	Real	Section 6.5	kg/m ³	Ambient
DEVC_ID	Character	Section 15.5.3		
DIAMETER	Real	Section 15.5.3	μm	
DT_INSERT	Real	Section 15.5.3	s	
DX	Real	Section 15.5.3	m	0.
DY	Real	Section 15.5.3	m	0.
DZ	Real	Section 15.5.3	m	0.
HEIGHT	Real	Section 15.5.3	m	
HRRPUV	Real	Section 6.5	kW/m ³	
ID	Character	Section 15.4		
MASS_FRACTION(N)	Real Array	Section 6.5	kg/kg	Ambient
MASS_PER_TIME	Real	Section 15.5.3	kg/s	
MASS_PER_VOLUME	Real	Section 15.5.3	kg/m ³	1
MULT_ID	Character	Section 7.5		

Table 19.10: Continued

INIT (Initial Conditions)				
N_PARTICLES	Integer	Section 15.5.3		0
N_PARTICLES_PER_CELL	Integer	Section 15.5.3		0
PART_ID	Character	Section 15.5.3		
PARTICLE_WEIGHT_FACTOR	Real	Section 15.5.3		1.
RADIUS	Real	Section 15.5.3	m	
SHAPE	Character	Section 15.5.3		'BLOCK'
SPEC_ID (N)	Character Array	Section 6.5		
TEMPERATURE	Real	Section 6.5	°C	TMPA
UVW (3)	Real Triplet	Section 15.5.3	m/s	0.
XB (6)	Real Sextuplet	Section 6.5	m	
XYZ (3)	Real Triplet	Section 15.5.3	m	

19.11 ISOF (Isosurface Parameters)

Table 19.11: For more information see Section 18.6.

ISOF (Isosurface Parameters)				
QUANTITY	Character	Section 18.6		
REDUCE_TRIANGLES	Integer	Reference [?]		1
SPEC_ID	Character	Section 18.6		
VALUE (I)	Real Array	Section 18.6		
VELO_INDEX	Integer	Section 18.10.17		0

19.12 MATL (Material Properties)

Table 19.12: For more information see Section 8.3.

MATL (Material Properties)				
A (:)	Real array	Section 8.5	1/s	
ABSORPTION_COEFFICIENT	Real	Section 8.3.2	1/m	50000.
ALLOW_SHRINKING	Logical	Section 8.5.3		.TRUE.
ALLOW_SWELLING	Logical	Section 8.5.3		.TRUE.
BOILING_TEMPERATURE	Real	Section 8.5.7	°C	5000.
CONDUCTIVITY	Real	Section 8.3.2	W/(m · K)	0.
CONDUCTIVITY_RAMP	Character	Section 8.3.2		
DENSITY	Real	Section 8.3.2	kg/m ³	0.
E (:)	Real array	Section 8.5	kJ/kmol	
EMISSION	Real	Section 8.3.2		0.9
GAS_DIFFUSION_DEPTH (:)	Real array	Section 8.5	m	0.001
HEATING_RATE (:)	Real array	Section 8.5	°C/min	5.

Table 19.12: Continued

MATL (Material Properties)				
HEAT_OF_COMBUSTION (:)	Real array	Section 8.5	kJ/kg	
HEAT_OF_REACTION (:)	Real array	Section 8.5	kJ/kg	0.
ID	Character	Section 8.1		
MATL_ID (: , :)	Character	Section 8.5		
NU_MATL (: , :)	Real array	Section 8.5	kg/kg	0.
NU_SPEC (: , :)	Real array	Section 8.5	kg/kg	0.
N_REACTIONS	Integer	Section 8.5		0
N_O2 (:)	Real array	Section 8.5		0.
N_S (:)	Real array	Section 8.5		1.
N_T (:)	Real array	Section 8.5		0.
PCR (:)	Logical array	Section 8.5		.FALSE.
PYROLYSIS_RANGE (:)	Real array	Section 8.5	°C	80.
REFERENCE_RATE (:)	Real array	Section 8.5	1/s	
REFERENCE_TEMPERATURE (:)	Real array	Section 8.5	°C	
SPECIFIC_HEAT	Real	Section 8.3.2	kJ/(kg · K)	0.
SPECIFIC_HEAT_RAMP	Character	Section 8.3.2		
SPEC_ID (: , :)	Character	Section 8.5		
THRESHOLD_SIGN (:)	Real array	Section 8.5		1.0
THRESHOLD_TEMPERATURE (:)	Real array	Section 8.5	°C	-273.15

19.13 MESH (Mesh Parameters)

Table 19.13: For more information see Section 6.3.

MESH (Mesh Parameters)				
COLOR	Character	Section 6.3.3		'BLACK'
CYLINDRICAL	Logical	Section 6.3.2		.FALSE.
EVACUATION	Logical	Reference [?]		.FALSE.
EVAC_HUMANS	Logical	Reference [?]		.FALSE.
EVAC_Z_OFFSET	Real	Reference [?]	m	1
ID	Character	Reference [?]		
IJK	Integer Triplet	Section 6.3.1		10,10,10
LEVEL	Integer	For future use		0
MPI_PROCESS	Integer	Section 6.3.3		
N_THREADS	Integer	Section 6.3.3		
MULT_ID	Character	Section 7.5		
PERIODIC_MESH_IDS	Character Array	Section 7.3.2		
RGB	Integer Triplet	Section 6.3.3		0,0,0
XB (6)	Real Sextuplet	Section 6.3.1	m	0,1,0,1,0,1

19.14 MISC (Miscellaneous Parameters)

Table 19.14: For more information see Section 6.4.

MISC (Miscellaneous Parameters)				
ALLOW_SURFACE_PARTICLES	Logical	Section 15.6.1		.TRUE.
ALLOW_UNDERSIDE_PARTICLES	Logical	Section 15.6.1		.FALSE.
ASSUMED_GAS_TEMPERATURE	Real	Section 8.6	°C	
ASSUMED_GAS_TEMPERATURE_RAMP	Character	Section 8.6		
BAROCLINIC	Logical	Section 6.4.7		.TRUE.
BNDF_DEFAULT	Logical	Section 18.5		.TRUE.
C_DEARDORFF	Real	Section 6.4.8		0.1
C_SMAGORINSKY	Real	Section 6.4.8		0.20
C_VREMAN	Real	Section 6.4.8		0.07
CFL_MAX	Real	Section 6.4.9		1.0
CFL_MIN	Real	Section 6.4.9		0.8
CFL_VELOCITY_NORM	Integer	Section 6.4.9		0 (LES), 1 (DNS)
CHECK_HT	Logical	Section 6.4.9		.FALSE.
CHECK_VN	Logical	Section 6.4.9		.FALSE.
CLIP_MASS_FRACTION	Logical	Section 6.7		.FALSE.
CNF_CUTOFF	Real	Section 15.3.3		0.005
CONSTANT_SPECIFIC_HEAT_RATIO	Logical	Section 12.1.2		.FALSE.
DNS	Logical	Section 6.4.1		.FALSE.
DRIFT_FLUX	Logical	Section 13.5		.TRUE.
DT_HVAC	Real	Section 9.2	s	
DT_MEAN_FORCING	Real	Section 10.2	s	1.E10
EVACUATION_DRILL	Logical	Reference [?]		.FALSE.
EVACUATION_MC_MODE	Logical	Reference [?]		.FALSE.
EVAC_PRESSURE_ITERATIONS	Integer	Reference [?]		50
EVAC_TIME_ITERATIONS	Integer	Reference [?]		50
FLUX_LIMITER	Integer	Section 6.4.10		2
FORCE_VECTOR(3)	Real	Section 6.4.2		0.
FREEZE_VELOCITY	Logical	Section 6.4.5		.FALSE.
GAMMA	Real	Section 12.1.2		1.4
GRAVITATIONAL_DEPOSITION	Logical	Section 13.5		.TRUE.
GRAVITATIONAL_SETTLING	Logical	Section 13.5		.TRUE.
GROUND_LEVEL	Real	Section 10.1	m	0.
GVEC	Real triplet	Section 6.4.6	m/s ²	0,0,-9.81
H_F_REFERENCE_TEMPERATURE	Real	Section 18.10.18	°C	25.
HVAC_MASS_TRANSPORT	Logical	Section 9.2.8		.FALSE.
HUMIDITY	Real	Section 12.1.1	%	40.
IBLANK_SMV	Logical	Section 18.4		.TRUE.
INITIAL_UNMIXED_FRACTION	Real	Section 13.1.3		1.0
LAPSE_RATE	Real	Section 10.1	°C/m	0
MAX_CHEMISTRY_ITERATIONS	Integer	Section 13.4		1000
MAX_LEAK_PATHS	Integer	Section 9.3.2		200

Table 19.14: Continued

MISC (Miscellaneous Parameters)				
MAXIMUM_VISIBILITY	Real	Section 18.10.2	m	30
MEAN_FORCING(3)	Logical	Section 10.2		.FALSE.
MPI_TIMEOUT	Real	Section 18.10.19	s	10.
NOISE	Logical	Section 6.4.1		.TRUE.
NOISE_VELOCITY	Real	Section 6.4.1	m/s	0.005
NO_EVACUATION	Logical	Reference [?]		.FALSE.
OVERWRITE	Logical	Section 6.4.1		.TRUE.
PARTICLE_CFL	Logical	Section 6.4.9		.FALSE.
PARTICLE_CFL_MAX	Real	Section 6.4.9		1.0
POROUS_FLOOR	Logical	Section 17.3.1		.TRUE.
PR	Real	Section 6.4.8		0.5
PROCESS_ALL_MESHES	Logical	Section 7.3.2		.FALSE.
PROJECTION	Logical	Formal projection		.FALSE.
P_INF	Real	Section 6.4.1	Pa	101325
RAMP_FVX_T	Character	Section 6.4.2		
RAMP_FVY_T	Character	Section 6.4.2		
RAMP_FVZ_T	Character	Section 6.4.2		
RAMP_GX	Character	Section 6.4.6		
RAMP_GY	Character	Section 6.4.6		
RAMP_GZ	Character	Section 6.4.6		
RAMP_TMP0_Z	Character	Section 10.1		
RAMP_U0_T	Character	Section 10.2		
RAMP_V0_T	Character	Section 10.2		
RAMP_W0_T	Character	Section 10.2		
RAMP_U0_Z	Character	Section 10.2		
RAMP_V0_Z	Character	Section 10.2		
RAMP_W0_Z	Character	Section 10.2		
RESTART	Logical	Section 6.4.3		.FALSE.
RESTART_CHID	Character	Section 6.4.3		CHID
RICHARDSON_ERROR_TOLERANCE	Real	Section 13		1.0 E-3
RUN_AVG_FAC	Real	Section 15.3.2		0.5
SC	Real	Section 6.4.8		0.5
SHARED_FILE_SYSTEM	Logical	Section 6.3.3		.TRUE.
SMOKE_ALBEDO	Real	Reference [?]		0.3
SOLID_PHASE_ONLY	Logical	Section 8.6		.FALSE.
STRATIFICATION	Logical	Section 10.1		.TRUE.
SUPPRESSION	Logical	Section 13.1.4		.TRUE.
TEXTURE_ORIGIN(3)	Real Triplet	Section 7.4.2	m	(0.,0.,0.)
THERMOPHORETIC_DEPOSITION	Logical	Section 13.5		.TRUE.
THICKEN_OBSTRUCTIONS	Logical	Section 7.2.1		.FALSE.
TPMA	Real	Section 6.4.1	°C	20.
TURBULENCE_MODEL	Character	Section 6.4.8		'DEARDORFF'
TURBULENT_DEPOSITION	Logical	Section 13.5		.TRUE.

Table 19.14: Continued

MISC (Miscellaneous Parameters)				
U0, V0, W0	Reals	Section 10.2	m/s	0.
VERBOSE	Logical	Section 6.3.3		
VISIBILITY_FACTOR	Real	Section 18.10.2		3
VN_MAX	Real	Section 6.4.9		0.5
VN_MIN	Real	Section 6.4.9		0.4
Y_CO2_INFTY	Real	Section 13.1.1	kg/kg	0.000595
Y_O2_INFTY	Real	Section 13.1.1	kg/kg	0.232378

19.15 MULT (Multiplier Function Parameters)

Table 19.15: For more information see Section 7.5.

MULT (Multiplier Function Parameters)				
DX	Real	Spacing in the x direction	m	0.
DXB	Real Sextuplet	Spacing for all 6 coordinates	m	0.
DX0	Real	Translation in the x direction	m	0.
DY	Real	Spacing in the y direction	m	0.
DY0	Real	Translation in the y direction	m	0.
DZ	Real	Spacing in the z direction	m	0.
DZ0	Real	Translation in the z direction	m	0.
ID	Character	Identification tag		
I_LOWER	Integer	Lower array bound, x direction		0
I_UPPER	Integer	Upper array bound, x direction		0
J_LOWER	Integer	Lower array bound, y direction		0
J_UPPER	Integer	Upper array bound, y direction		0
K_LOWER	Integer	Lower array bound, z direction		0
K_UPPER	Integer	Upper array bound, z direction		0
N_LOWER	Integer	Lower sequence bound		0
N_UPPER	Integer	Upper sequence bound		0

19.16 OBST (Obstruction Parameters)

Table 19.16: For more information see Section 7.2.

OBST (Obstruction Parameters)				
ALLOW_VENT	Logical	Section 7.2.1		.TRUE.
BNDF_FACE (-3:3)	Logical Array	Section 18.5		.TRUE.
BNDF_OBST	Logical	Section 18.5		.TRUE.
BULK_DENSITY	Real	Section 8.5.8	kg/m ³	
COLOR	Character	Section 7.2.1		

Table 19.16: Continued

OBST (Obstruction Parameters)				
CTRL_ID	Character	Section 17.4.2		
DEVC_ID	Character	Section 17.4.2		
EVACUATION	Logical	Reference [?]		.FALSE.
HT3D	Logical	Section 8.3.9		.FALSE.
ID	Character	Section 7.2.1		
MESH_ID	Character	Reference [?]		
MULT_ID	Character	Section 7.5		
OUTLINE	Logical	Section 7.2.1		.FALSE.
OVERLAY	Logical	Section 7.2.1		.TRUE.
PERMIT_HOLE	Logical	Section 7.2.6		.TRUE.
PROP_ID	Character	Reference [?]		
REMOVABLE	Logical	Section 7.2.6		.TRUE.
RGB (3)	Integer Triplet	Section 7.2.1		
SURF_ID	Character	Section 7.2.1		
SURF_ID6 (6)	Character Sextuplet	Section 7.2.1		
SURF_IDS (3)	Character Triplet	Section 7.2.1		
TEXTURE_ORIGIN (3)	Real Triplet	Section 7.4.2	m	(0.,0.,0.)
THICKEN	Logical	Section 7.2.1		.FALSE.
TRANSPARENCY	Real	Section 7.2.1		1
XB (6)	Real Sextuplet	Section 7.2.1	m	

19.17 PART (Lagrangian Particles/Droplets)

Table 19.17: For more information see Chapter 15.

PART (Lagrangian Particles/Droplets)				
AGE	Real	Section 18.9	s	1×10^5
BREAKUP	Logical	Section 15.3.4		.FALSE.
BREAKUP_CNF_RAMP_ID	Character	Section 15.3.4		
BREAKUP_DISTRIBUTION	Character	Section 15.3.4		'ROSIN...'
BREAKUP_GAMMA_D	Real	Section 15.3.4		2.4
BREAKUP_RATIO	Real	Section 15.3.4		3/7
BREAKUP_SIGMA_D	Real	Section 15.3.4		
CHECK_DISTRIBUTION	Logical	Section 15.3.3		.FALSE.
CNF_RAMP_ID	Character	Section 15.3.3		
COLOR	Character	Section 18.9		'BLACK'
COMPLEX_REFRACTIVE_INDEX	Real	Section 15.3.2		0.01
CTRL_ID	Character	Section 15.5.1		
DENSE_VOLUME_FRACTION	Real	Section 15.4.2		1×10^{-5}
DEVC_ID	Character	Section 15.5.1		
DIAMETER	Real	Section 15.3.3	μm	
DISTRIBUTION	Character	Section 15.3.3		'ROSIN...'

Table 19.17: Continued

PART (Lagrangian Particles/Droplets)				
DRAG_COEFFICIENT (3)	Real Array	Section 15.4.2		
DRAG_LAW	Character	Section 15.4.2		' SPHERE'
FREE_AREA_FRACTION	Real	Section 15.4.8		
GAMMA_D	Real	Section 15.3.3		2.4
HEAT_OF_COMBUSTION	Real	Section 15.3.5	kJ/kg	
HORIZONTAL_VELOCITY	Real	Section 15.6.1	m/s	0.2
ID	Character	Section 15.1		
INITIAL_TEMPERATURE	Real	Section 15.3.1	°C	TMPA
MASSLESS	Logical	Section 15.2		.FALSE.
MAXIMUM_DIAMETER	Real	Section 15.3.3	μm	Infinite
MINIMUM_DIAMETER	Real	Section 15.3.3	μm	20.
MONODISPERSE	Logical	Section 15.3.3		.FALSE.
N_STRATA	Integer	Section 15.3.3		7
ORIENTATION (1:3, :)	Real Array	Section 15.4		
PERMEABILITY (3)	Real Array	Section 15.4.7		
PROP_ID	Character	Section 15.1		
QUANTITIES (10)	Character	Section 18.9		
QUANTITIES_SPEC_ID (10)	Character	Section 18.9		
RADIATIVE_PROPERTY_TABLE	Real	Section 15.3.2		
REAL_REFRACTIVE_INDEX	Real	Section 15.3.2		1.33
RGB (3)	Integers	Section 18.9		
SAMPLING_FACTOR	Integer	Section 18.9		1
SECOND_ORDER_PARTICLE_TRANSPORT	Logical	Section 6.4.9		.FALSE.
SIGMA_D	Real	Section 15.3.3		
SPEC_ID	Character	Section 15.3.1		
STATIC	Logical	Section 15.4		.FALSE.
SURFACE_TENSION	Real	Section 15.3.4	N/m	7.28×10^4
SURF_ID	Character	Section 15.4		
TURBULENT_DISPERSION	Logical	Section 15.2		.FALSE.
VERTICAL_VELOCITY	Real	Section 15.6.1	m/s	0.5

19.18 PRES (Pressure Solver Parameters)

Table 19.18: For more information see Section 6.6.

PRES (Pressure Solver Parameters)				
CHECK_POISSON	Logical	Section 6.6		.FALSE.
FISHPAK_BC (3)	Integer	Section 6.6		
ITERATION_SUSPEND_FACTOR	Real	Section 6.6	s	0.95
MAX_PRESSURE_ITERATIONS	Integer	Section 6.6		10
PRESSURE_RELAX_TIME	Real	Section 6.6	s	1.
PRESSURE_TOLERANCE	Real	Section 6.6	m/s ²	1.

Table 19.18: Continued

PRES (Pressure Solver Parameters)				
RELAXATION_FACTOR	Real	Section 6.6		1.
SUSPEND_PRESSURE_ITERATIONS	Logical	Section 6.6		.TRUE.
VELOCITY_TOLERANCE	Real	Section 6.6	m/s	

19.19 PROF (Wall Profile Parameters)

Table 19.19: For more information see Section 18.3.

PROF (Wall Profile Parameters)				
ID	Character	Section 18.3		
FORMAT_INDEX	Integer	Section 18.3		1
IOR	Real	Section 18.3		
QUANTITY	Character	Section 18.3		
XYZ	Real Triplet	Section 18.3	m	

19.20 PROP (Device Properties)

Table 19.20: For more information see Section 17.3.

PROP (Device Properties)				
ACTIVATION_OBSCURATION	Real	Section 17.3.5	%/m	3.24
ACTIVATION_TEMPERATURE	Real	Section 17.3.1	°C	74.
ALPHA_C	Real	Section 17.3.5		1.8
ALPHA_E	Real	Section 17.3.5		0.
BETA_C	Real	Section 17.3.5		1.
BETA_E	Real	Section 17.3.5		1.
CHARACTERISTIC_VELOCITY	Real	Section 18.10.13	m/s	1.
C_FACTOR	Real	Section 17.3.1	(m/s) ^{1/2}	0.
DENSITY	Real	Section 18.10.4	kg/m ³	8908.
DIAMETER	Real	Section 18.10.4	m	0.001
EMISSION	Real	Section 18.10.4		0.85
FLOW_RAMP	Character	Section 17.3.1		
FLOW_RATE	Real	Section 17.3.1	L/min	
FLOW_TAU	Real	Section 17.3.1		0.
GAUGE_EMISSION	Real	Section 18.10.5		0.9
GAUGE_TEMPERATURE	Real	Section 18.10.5	°C	TMPA
HEAT_TRANSFER_COEFFICIENT	Real	Section 18.10.4	W/(m ² · K)	
ID	Character	Section 17.3		
INITIAL_TEMPERATURE	Real	Section 17.3.1	°C	TMPA
K_FACTOR	Real	Section 17.3.1	L/(min · bar ^{1/2})	1.

Table 19.20: Continued

PROP (Device Properties)				
LENGTH	Real	Section 17.3.5	m	1.8
MASS_FLOW_RATE	Real	Section 17.3.1	kg/s	
OFFSET	Real	Section 17.3.1	m	0.05
OPERATING_PRESSURE	Real	Section 17.3.1	bar	1.
ORIFICE_DIAMETER	Real	Section 17.3.1	m	0.
P0, PX(3), PXX(3,3)	Real	Section 17.3.3	m/s	0.
PARTICLES_PER_SECOND	Integer	Section 17.3.1		5000
PARTICLE_VELOCITY	Real	Section 17.3.1	m/s	0.
PART_ID	Character	Section 17.3.1		
PDPA_END	Real	Section 18.10.7	s	T_END
PDPA_HISTOGRAM	Logical	Section 18.10.7		.FALSE.
PDPA_HISTOGRAM_CUMULATIVE	Logical	Section 18.10.7		.FALSE.
PDPA_HISTOGRAM_LIMITS	Real Array	Section 18.10.7		
PDPA_HISTOGRAM_NBINS	Integer	Section 18.10.7		10
PDPA_INTEGRATE	Logical	Section 18.10.7		.TRUE.
PDPA_M	Integer	Section 18.10.7		0
PDPA_N	Integer	Section 18.10.7		0
PDPA_NORMALIZE	Logical	Section 18.10.7		.TRUE.
PDPA_RADIUS	Real	Section 18.10.7	m	0.
PDPA_START	Real	Section 18.10.7	s	0.
PRESSURE_RAMP	Character	Section 17.3.1		
QUANTITY	Character	Section 17.3.1		
RTI	Real	Section 17.3.1	$\sqrt{\text{m} \cdot \text{s}}$	100.
SMOKEVIEW_ID	Char. Array	Section 17.7.1		
SMOKEVIEW_PARAMETERS	Char. Array	Section 17.7.2		
SPEC_ID	Character	Section 17.3.5		
SPECIFIC_HEAT	Real	Section 18.10.4	kJ/(kg · K)	0.44
SPRAY_ANGLE(2,2)	Real	Section 17.3.1	degrees	60.,75.
SPRAY_PATTERN_BETA	Real	Section 17.3.1	degrees	5.
SPRAY_PATTERN_MU	Real	Section 17.3.1	degrees	0.
SPRAY_PATTERN_SHAPE	Character	Section 17.3.1		'GAUSSIAN'
SPRAY_PATTERN_TABLE	Character	Section 17.3.1		
VELOCITY_COMPONENT	Integer	Section 17.3.3		

19.21 RADI (Radiation Parameters)

Table 19.21: For more information see Section 14.1.

RADI (Radiation Parameters)				
ANGLE_INCREMENT	Integer	Section 14.1.2		5
BAND_LIMITS	Real Array	Section 14.2.3	μm	
INITIAL_RADIATION_ITERATIONS	Integer	Section 14.1.2		3

Table 19.21: Continued

RADI (Radiation Parameters)				
KAPPA0	Real	Section 14.2.2	1/m	0
MIE_MINIMUM_DIAMETER	Real	Section 14.2.2	μm	0.5
MIE_MAXIMUM_DIAMETER	Real	Section 14.2.2	μm	$1.5 \times D$
MIE_NDG	Integer	Section 14.2.2		50
NMIEANG	Integer	Section 14.2.2		15
NUMBER_RADIATION_ANGLES	Integer	Section 14.1.2		100
PATH_LENGTH	Real	Section 14.2.3	m	
RADIATION	Logical	Section 14.1		.TRUE.
RADIATION_ITERATIONS	Integer	Section 14.1.2		1
RADTMP	Real	Section 14.2.2	$^{\circ}\text{C}$	900
TIME_STEP_INCREMENT	Integer	Section 14.1.2		3
WIDE_BAND_MODEL	Logical	Section 14.2.3		.FALSE.

19.22 RAMP (Ramp Function Parameters)

Table 19.22: For more information see Chapter 11.

RAMP (Ramp Function Parameters)				
CTRL_ID	Character	Section 17.6.1		
DEVC_ID	Character	Section 17.6.1		
F	Real	Chapter 11		
ID	Character	Chapter 11		
NUMBER_INTERPOLATION_POINTS	Integer	Chapter 11		5000
T	Real	Chapter 11	s (or $^{\circ}\text{C}$)	
X	Real	Section 6.4.6	m	

19.23 REAC (Reaction Parameters)

Table 19.23: For more information see Chapter 13.

REAC (Reaction Parameters)				
A	Real	Section 13.3		
AUTO_IGNITION_TEMPERATURE	Real	Section 13.1.4	$^{\circ}\text{C}$	
C	Real	Section 13.1.1		0
CHECK_ATOM_BALANCE	Logical	Section 13.2		.TRUE.
CO_YIELD	Real	Section 13.1.1	kg/kg	0
CRITICAL_FLAME_TEMPERATURE	Real	Section 13.1.4	$^{\circ}\text{C}$	1327
E	Real	Section 13.3	kJ/kmol	
EPUMO2	Real	Section 13.1.2	kJ/kg	13100
EQUATION	Character	Section 13.2.3		

Table 19.23: Continued

REAC (Reaction Parameters)				
FORMULA	Character	Section 13.1.1		
FUEL	Character	Section 13.1.1		
FUEL_RADCAL_ID	Character	Section 13.1.1		
H	Real	Section 13.1.1		0
HEAT_OF_COMBUSTION	Real	Section 13.1.2	kJ/kg	
ID	Character	Section 13.1.1		
IDEAL	Logical	Section 13.1.1		.FALSE.
N	Real	Section 13.1.1		0
NU (:)	Real Array	Section 13.3		
N_S (:)	Real Array	Section 13.3		
N_T	Real	Section 13.3		
O	Real	Section 13.1.1		0
RADIATIVE_FRACTION	Real	Section 14.1.1		
RAMP_CHI_R	Character	Section 14.1.1		
REAC_ATOM_ERROR	Real	Section 13.2	atoms	1.E-5
REAC_MASS_ERROR	Real	Section 13.2	kg/kg	1.E-4
SOOT_H_FRACTION	Real	Section 13.1.1		0.1
SOOT_YIELD	Real	Section 13.1.1	kg/kg	0.0
SPEC_ID_N_S (:)	Char. Array	Section 13.3		
SPEC_ID_NU (:)	Char. Array	Section 13.3		
THIRD_BODY	Logical	Section 13.3		.FALSE.

19.24 SLCF (Slice File Parameters)

Table 19.24: For more information see Section 18.4.

SLCF (Slice File Parameters)				
CELL_CENTERED	Logical	Section 18.4		.FALSE.
EVACUATION	Logical	Reference [?]		.FALSE.
MAXIMUM_VALUE	Real	Reference [?]		
MESH_NUMBER	Integer	Section 18.4		
MINIMUM_VALUE	Real	Reference [?]		
PART_ID	Character	Section 18.12		
PBX, PBZ, PBZ	Real	Section 18.4	m	
QUANTITY	Character	Section 18.12		
QUANTITY2	Character	Section 18.12		
SPEC_ID	Character	Section 18.12		
VECTOR	Logical	Section 18.4		.FALSE.
VELO_INDEX	Integer	Section 18.10.17		0
XB (6)	Real Sextuplet	Section 18.4	m	

19.25 SPEC (Species Parameters)

Table 19.25: For more information see Section 12.

SPEC (Species Parameters)				
AEROSOL	Logical	Section 13.5		.FALSE.
ALIAS	Character	Section 12.1.3		
BACKGROUND	Logical	Section 12		.FALSE.
CONDUCTIVITY	Real	Section 12.1.2	W/(m · K)	
CONDUCTIVITY_SOLID	Real	Section 13.5	W/(m · K)	0.26
DENSITY_LIQUID	Real	Section 15.3.1	kg/m ³	
DENSITY_SOLID	Real	Section 13.5	kg/m ³	1800.
DIFFUSIVITY	Real	Section 12.1.2	m ² /s	
ENTHALPY_OF_FORMATION	Real	Section 15.3.1	kJ/mol	
EPSILONKLJ	Real	Section 12.1.2		0
FIC_CONCENTRATION	Real	Section 18.10.9	ppm	0.
FLD_LETHAL_DOSE	Real	Section 18.10.9	ppm × min	0.
FORMULA	Character	Section 12.1.2		
HEAT_OF_VAPORIZATION	Real	Section 15.3.1	kJ/kg	
H_V_REFERENCE_TEMPERATURE	Real	Section 15.3.1	°C	
ID	Character	Section 12.1.1		
LUMPED_COMPONENT_ONLY	Logical	Section 12.2		.FALSE.
MASS_EXTINCTION_COEFFICIENT	Real	Section 17.3.5		0
MASS_FRACTION(:)	Real Array	Section 12.2		0
MASS_FRACTION_0	Real	Section 12.1.1		0
MEAN_DIAMETER	Real	Section 13.5	m	1.E-6
MELTING_TEMPERATURE	Real	Section 15.3.1	°C	
MW	Real	Section 12.1.2	g/mol	29.
PR_GAS	Real	Section 12.1.2		PR
PRIMITIVE	Logical	Section 12.1.2		
RADCAL_ID	Character	Section 12.1.3		
RAMP_CP	Character	Section 12.1.2		
RAMP_CP_L	Character	Section 15.3.1		
RAMP_D	Character	Section 12.1.2		
RAMP_G_F	Character	Section 12.1.2		
RAMP_K	Character	Section 12.1.2		
RAMP_MU	Character	Section 12.1.2		
REFERENCE_ENTHALPY	Real	Section 12.1.2	kJ/kg	
REFERENCE_TEMPERATURE	Real	Section 12.1.2	°C	25.
SIGMALJ	Real	Section 12.1.2		0
SPEC_ID(:)	Character Array	Section 12.2		
SPECIFIC_HEAT	Real	Section 12.1.2	kJ/(kg · K)	
SPECIFIC_HEAT_LIQUID	Real	Section 15.3.1	kJ/(kg · K)	
VAPORIZATION_TEMPERATURE	Real	Section 15.3.1	°C	
VISCOSITY	Real	Section 12.1.2	kg/(m · s)	
VOLUME_FRACTION(:)	Real Array	Section 12.2		

19.26 SURF (Surface Properties)

Table 19.26: For more information see Section 7.1.

SURF (Surface Properties)				
ADIABATIC	Logical	Section 8.2.3		.FALSE.
BACKING	Character	Section 8.3.3		'EXPOSED'
BURN_AWAY	Logical	Section 8.5.8		.FALSE.
CELL_SIZE_FACTOR	Real	Section 8.3.8		1.0
C_FORCED_CONSTANT	Real	Section 8.2.2		0.0
C_FORCED_PR_EXP	Real	Section 8.2.2		0.0
C_FORCED_RE	Real	Section 8.2.2		0.0
C_FORCED_RE_EXP	Real	Section 8.2.2		0.0
C_HORIZONTAL	Real	Section 8.2.2		1.52
C_VERTICAL	Real	Section 8.2.2		1.31
COLOR	Character	Section 7.4		
CONVECTION_LENGTH_SCALE	Real	Section 8.2.2	m	1.
CONVECTIVE_HEAT_FLUX	Real	Section 8.2.2	kW/m ²	
CONVERT_VOLUME_TO_MASS	Logical	Section 9.1.6		.FALSE.
DEFAULT	Logical	Section 7.1		.FALSE.
DT_INSERT	Real	Section 15.5.1	s	0.01
EMISSIVITY	Real	Section 8.2.2		0.9
EMISSIVITY_BACK	Real	Section 8.3.3		
EVAC_DEFAULT	Logical	Reference [?]		.FALSE.
EXTERNAL_FLUX	Real	Section 8.6	kW/m ²	
E_COEFFICIENT	Real	Section 15.6	m ² /(kg · s)	
FREE_SLIP	Logical	Section 9.1.7		.FALSE.
GEOMETRY	Character	Section 8.3.7		'CARTESIAN'
HEAT_OF_VAPORIZATION	Real	Section 8.4.3	kJ/kg	
HEAT_TRANSFER_COEFFICIENT	Real	Section 8.2.2	W/(m ² · K)	
HEAT_TRANSFER_MODEL	Character	Section 8.2.2		
HRRPUA	Real	Section 8.4.1	kW/m ²	
HT3D	Logical	Section 8.3.9		.FALSE.
ID	Character	Section 7.1		
IGNITION_TEMPERATURE	Real	Section 8.4.3	°C	5000.
INNER_RADIUS	Real	Section 15.4.1	m	
INTERNAL_HEAT_SOURCE	Real Array	Section 8.3.6	kW/m ³	
LAYER_DIVIDE	Real	Section 8.3.5		N_LAYERS/2
LEAK_PATH	Int. Pair	Section 9.3.2		
LENGTH	Real	Section 15.4.1	m	
MASS_FLUX(:)	Real Array	Section 9.1.6	kg/(m ² · s)	
MASS_FLUX_TOTAL	Real	Section 9.1.2	kg/(m ² · s)	
MASS_FLUX_VAR	Real	Section 9.1.9		
MASS_FRACTION(:)	Real Array	Section 9.1.6		
MASS_TRANSFER_COEFFICIENT	Real	Section 8.5.7	m/s	
MATL_ID (NL, NC)	Char. Array	Section 8.5		

Table 19.26: Continued

SURF (Surface Properties)				
MATL_MASS_FRACTION (NL, NC)	Real Array	Section 8.5		
MINIMUM_LAYER_THICKNESS	Real	Section 8.3.8	m	1.E-6
MLRPUA	Real	Section 8.4.1	kg/(m ² · s)	
N_LAYER_CELLS_MAX	Integer Array	Section 8.3.8		1000
NET_HEAT_FLUX	Real	Section 8.2.2	kW/m ²	
NO_SLIP	Logical	Section 9.1.7		.FALSE.
NPPC	Integer	Section 15.5.1		1
PARTICLE_MASS_FLUX	Real	Section 15.5.1	kg/(m ² · s)	
PART_ID	Character	Section 15.5.1		
PLE	Real	Section 10.1		0.3
PROFILE	Character	Section 9.5		
RADIUS	Real	Section 15.4.1	m	
RAMP_EF	Character	Section 11.1		
RAMP_MF (:)	Character	Section 11.1		
RAMP_PART	Character	Section 11.1		
RAMP_Q	Character	Section 11.1		
RAMP_T	Character	Section 11.1		
RAMP_T_I	Character	Section 8.3.4		
RAMP_V	Character	Section 11.1		
RAMP_V_X	Character	Section 11.3		
RAMP_V_Y	Character	Section 11.3		
RAMP_V_Z	Character	Section 11.3		
RGB (3)	Int. Triplet	Section 7.4		255,204,102
ROUGHNESS	Real	Section 9.1.7	m	0.
SPEC_ID	Character	Section 9.1.6		
SPREAD_RATE	Real	Section 8.4.2	m/s	
STRETCH_FACTOR (:)	Real	Section 8.3.8		2.
TAU_EF	Real	Section 11.1	s	1.
TAU_MF (:)	Real	Section 11.1	s	1.
TAU_PART	Real	Section 11.1	s	1.
TAU_Q	Real	Section 11.1	s	1.
TAU_T	Real	Section 11.1	s	1.
TAU_V	Real	Section 11.1	s	1.
TEXTURE_HEIGHT	Real	Section 7.4.2	m	1.
TEXTURE_MAP	Character	Section 7.4.2		
TEXTURE_WIDTH	Real	Section 7.4.2	m	1.
TGA_ANALYSIS	Logical	Section 8.6.2		.FALSE.
TGA_FINAL_TEMPERATURE	Real	Section 8.6.2	°C	800.
TGA_HEATING_RATE	Real	Section 8.6.2	°C/min	5.
THICKNESS (NL)	Real Array	Section 8.1	m	
TMP_BACK	Real	Section 8.3.4	°C	20.
TMP_FRONT	Real	Section 8.2.1	°C	20.
TMP_INNER (:)	Real Array	Section 8.3.4	°C	20.

Table 19.26: Continued

SURF (Surface Properties)				
TRANSPARENCY	Real	Section 7.4		1.
VEL	Real	Section 9.1	m/s	
VEL_BULK	Real	Section 9.5	m/s	
VEL_GRAD	Real	Section 9.1.5	1/s	
VEL_T	Real Pair	Section 9.1.4	m/s	
VOLUME_FLOW	Real	Section 9.1	m ³ /s	
WIDTH	Real	Section 15.4.1	m	
XYZ (3)	Real Triplet	Section 8.4.2	m	
Z0	Real	Section 10.1	m	10.

19.27 TABL (Table Parameters)

Table 19.27: For more information see Section 17.3.1.

TABL (Table Parameters)				
ID	Character	Section 17.3.1		
TABLE_DATA (9)	Real Array	Section 17.3.1		

19.28 TIME (Time Parameters)

Table 19.28: For more information see Section 6.2.

TIME (Time Parameters)				
DT	Real	Section 6.2.2	s	
EVAC_DT_FLOWFIELD	Real	Reference [?]	s	0.01
EVAC_DT_STEADY_STATE	Real	Reference [?]	s	0.05
LIMITING_DT_RATIO	Real	Section 4.2		0.0001
LOCK_TIME_STEP	Logical	Section 6.2.2		.FALSE.
RESTRICT_TIME_STEP	Logical	Section 6.2.2		.TRUE.
T_BEGIN	Real	Section 6.2.1	s	0.
T_END	Real	Section 6.2.1	s	1.
TIME_SHRINK_FACTOR	Real	Section 6.2.3		1.
WALL_INCREMENT	Integer	Section 8.3.8		2

19.29 TRNX, TRNY, TRNZ (MESH Transformations)

Table 19.29: For more information see Section 6.3.5.

TRNX, TRNY, TRNZ (MESH Transformations)				
CC	Real	Section 6.3.5	m	
IDERIV	Integer	Section 6.3.5		
MESH_NUMBER	Integer	Section 6.3.5		
PC	Real	Section 6.3.5		

19.30 VENT (Vent Parameters)

Table 19.30: For more information see Section 7.3.

VENT (Vent Parameters)				
COLOR	Character	Section 7.4		
CTRL_ID	Character	Section 17.4.2		
DEVC_ID	Character	Section 17.4.2		
DYNAMIC_PRESSURE	Real	Section 9.4	Pa	0.
EVACUATION	Logical	Reference [?]		.FALSE.
ID	Character	Section 7.3.1		
IOR	Integer	Section 7.3.4		
L_EDDY	Real	Section 9.1.8	m	0.
L_EDDY_IJ (3, 3)	Real Array	Section 9.1.8	m	0.
MB	Character	Section 7.3.1		
MESH_ID	Character	Reference [?]		
MULT_ID	Character	Section 7.5		
N_EDDY	Integer	Section 9.1.8		0
OUTLINE	Logical	Section 7.3.1		.FALSE.
PBX, PBX, PBZ	Real	Section 7.3.1		
PRESSURE_RAMP	Character	Section 9.4		
REYNOLDS_STRESS (3, 3)	Real Array	Section 9.1.8	m ² /s ²	0.
RGB (3)	Integer Triplet	Section 7.4		
SPREAD_RATE	Real	Section 8.4.2	m/s	0.05
SURF_ID	Character	Section 7.3.1		' INERT '
TEXTURE_ORIGIN (3)	Real Triplet	Section 7.4.2	m	(0.,0.,0.)
TMP_EXTERIOR	Real	Section 7.3.2	°C	
TMP_EXTERIOR_RAMP	Character	Section 7.3.2		
TRANSPARENCY	Real	Section 7.4		1.0
UVW (3)	Real Triplet	Section 9.2.7	m/s	
VEL_RMS	Real	Section 9.1.8	m/s	0.
XB (6)	Real Sextuplet	Section 7.3.1	m	
XYZ (3)	Real Triplet	Section 8.4.2	m	

19.31 ZONE (Pressure Zone Parameters)

Table 19.31: For more information see Section 9.3.

ZONE (Pressure Zone Parameters)				
ID	Character	Section 9.3.1		
LEAK_AREA (N)	Real	Section 9.3.2	m ²	0
LEAK_PRESSURE_EXPONENT (N)	Real	Section 9.3.2		0.5
LEAK_REFERENCE_PRESSURE (N)	Real	Section 9.3.2	Pa	4
PERIODIC	Logical	Section 9.3.1		.FALSE.
XB (6)	Real Sextuplet	Section 9.3.1	m	

Part III

FDS and Smokeview Development Tools

Chapter 20

The FDS/Smokeview Repository

For those interested in obtaining the FDS and Smokeview source codes, either for development work or simply to compile on a particular platform, it is strongly suggested that you download onto your computer the entire FDS/Smokeview “Repository.” All project documents are maintained using the online utility [GitHub](#), a free service that supports software development for open source applications. GitHub uses Git version control software. Under this system, a centralized repository containing all project files resides on a GitHub server. Anyone can obtain a copy of the repository or retrieve a specific revision of the repository. However, only the FDS and Smokeview developers can commit changes directly to the repository. Others must submit a “pull request.” Detailed instructions for checking out the FDS repository can be found at <https://github.com/firemodels/fds>.

Both FDS and Smokeview live within a GitHub *organization* called “Fire Models”. The current location of the organization is <https://github.com/firemodels>. The repositories that are used by the FDS and Smokeview projects are listed below along with a brief description:

fds	FDS source code, verification and validation tests, wikis, and documentation
smv	Smokeview source code, integration tests, and documentation
exp	Experimental data repository for FDS validation
out	FDS output results for validation
bot	Firebot (continuous integration system) source
fds-smv	Web page html source

The Wiki Pages are particularly useful in describing the details of how you go about working with the repository assets.

Chapter 21

Compiling FDS

If a compiled version of FDS exists for the machine on which the calculation is to be run and no changes have been made to the original source code, there is no need to re-compile the code. For example, the file `fds.exe` is the compiled program for a Windows-based PC; thus PC users do not need a Fortran compiler and do not need to compile the source code. For machines for which an executable has not been compiled, you must compile the code. A Fortran 2008 compliant compiler is required.

21.1 FDS Source Code

Table 21.1 lists the files that make up the FDS source code. Files with the “.f90” suffix contain free-form Fortran 90 instructions conforming to the ANSI and ISO standards (2008 edition). A `makefile` is available in the FDS-SMV Repository that contains platform specific options for compilation. Note the following:

- The source code consists entirely of Fortran statements organized into about 35 files. Some compilers have a standard optimization level, plus various degrees of “aggressive” optimization. Be cautious in using the highest levels of optimization.
- For the non-MPI version of FDS, compile with `mpis.f90`.
- The MPI version of FDS uses `mpip.f90` instead of `mpis.f90`, plus additional MPI libraries need to be installed. More details on MPI can be found at the FDS-SMV website, along with links to the necessary organizations who have developed free MPI libraries.

Table 21.1: FDS source code files

File Name	Description
cons.f90	Global arrays and constants
ctrl.f90	Definitions and routines for control functions
data.f90	Data for output quantities and thermophysical properties
devc.f90	Derived type definitions and constants for devices
divg.f90	Compute the flow divergence
dump.f90	Output data dumps into files
evac.f90	Egress computations (future capability)
fire.f90	Combustion routines
func.f90	Global functions and subroutines
geom.f90	Routines supporting complex, unstructured geometry (under development)
gsmv.f90	Routines supporting complex, unstructured geometry (under development)
hvac.f90	Heating, Ventilation, and Air Conditioning
ieva.f90	Support routines for evac.f90
init.f90	Initialize variables and Poisson solver
irad.f90	Functions needed for radiation solver, including RadCal
main.f90	Main program
mass.f90	Mass equation(s) and thermal boundary conditions
mesh.f90	Arrays and constants associated with each mesh
mpip.f90	MPI “include” statement for MPI compilation
mpis.f90	“Dummy” Fortran/MPI bindings for non-MPI compilation
part.f90	Lagrangian particle transport and sprinkler activation
pois.f90	Poisson (pressure) solver
prec.f90	Specification of numerical precision
pres.f90	Spatial discretization of pressure (Poisson) equation
radi.f90	Radiation solver
read.f90	Read input parameters
samr.f90	Simplified Adaptive Mesh Refinement (under development)
scrc.f90	Alternative pressure solver (under development)
smvv.f90	Routines for computing and outputting 3D smoke and isosurfaces
soot.f90	Soot agglomeration and aerosol deposition
turb.f90	Turbulence models and manufactured solutions
type.f90	Derived type definitions
vege.f90	Experimental vegetation model
velo.f90	Momentum equations
wall.f90	Wall boundary conditions

Chapter 22

Output File Formats

The output from the code consists of the file `CHID.out`, plus various data files that are described below. Most of these output files are written out by the subroutines within `dump.f90`, and can easily be modified to accommodate various plotting packages.

22.1 Diagnostic Output

The file `CHID.out` contains diagnostic output, including an accounting of various important quantities, including CPU usage. Typically, diagnostic information is printed out every 100 time steps as follows:

```
Time Step 137431   December 27, 2015  00:29:49
Step Size:   0.563E-01 s, Total Time:   1800.04 s
Pressure Iterations:      1
Maximum Velocity Error:  0.30E-01 on Mesh  1 at (  0 44  3)
-----
Max CFL number:  0.94E+00 at (  1, 46,  1)
Max divergence:  0.13E+00 at ( 66, 12,  1)
Min divergence: -0.20E+00 at ( 66, 13,  1)
Max VN number:   0.51E+00 at (  1, 25, 18)
No. of Lagrangian Particles:      27
Radiation Loss to Boundaries:      13.830 kW
```

The `Time Step` indicates the total number of iterations. The date and time indicate the current wall clock time. The `STEP SIZE` indicates the size of the numerical time step. The `Total Time` indicates the total simulation time calculated up to that point. The `Pressure Iterations` are the number of iterations of the pressure solver for the corrector (second) half of the time step. The pressure solver iterations are designed to minimize the error in the normal component of velocity at solid walls or the interface of two meshes. The `Maximum Velocity Error` indicates this error and in which grid cell it occurs. `Max/Min divergence` is the max/min value of the function $\nabla \cdot \mathbf{u}$ and is used as a diagnostic when the flow is incompressible (i.e., no heating); `Max CFL number` is the maximum value of the CFL number, the primary time step constraint; `Max VN number` is the maximum value of the Von Neumann number, the secondary time step constraint. The `No. of Lagrangian Particles` refers to the number of particles in the current mesh. The `Radiation Loss to Boundaries` is the amount of energy that is being radiated to the boundaries. As compartments heat up, the energy lost to the boundaries can grow to be an appreciable fraction of the Total Heat Release Rate.

Following the completion of a successful run, a summary of the CPU usage per subroutine is listed in the file called `CHID_cpu.csv` (Section 22.5). This is useful in determining where most of the computational

effort is being placed.

22.2 Heat Release Rate and Related Quantities

The heat release rate of the fire, plus other global energy-related quantities, are automatically written into a text file called `CHID_hrr.csv`. The format of the file is as follows

```
s      , kW      , kW      , ... , kg/s      , Pa      , Pa      , ...
Time  , HRR      , Q_RADI  , ... , BURN_RATE , ZONE_01 , ZONE_02 , ...
T(1)  , VAL(1,1) , VAL(2,1) , ... , VAL(8,1)  , VAL(9,1) , VAL(10,1) , ...
T(2)  , VAL(1,2) , VAL(2,2) , ... , VAL(8,2)  , VAL(9,2) , VAL(10,2) , ...
      .
      .
```

Details of the integrated energy quantities can be found in Section 18.10.1. `BURN_RATE` is the total mass loss rate of fuel, and `ZONE_01`, etc., are the background pressures of the various pressure ZONES. Note that the reported `BURN_RATE` is not adjusted to account for the possibility that each individual material might have a different heat of combustion. It is the actual burning rate of the fuel as predicted by FDS or specified by you. The background pressure is discussed in Section 9.3.

22.3 Device Output Data

Data associated with particular devices (link temperatures, smoke obscuration, thermocouples, etc.) specified in the input file under the namelist group `DEVC` is output in comma delimited format in a file called `CHID_devc.csv`. The format of the file is as follows:

```
s      , UNITS(1) , UNITS(2) , ... , UNITS(N_DEVC)
Time  , ID(1)      , ID(2)      , ... , ID(N_DEVC)
T(1)  , VAL(1,1) , VAL(2,1) , ... , VAL(N_DEVC,1)
T(2)  , VAL(1,2) , VAL(2,2) , ... , VAL(N_DEVC,2)
      .
      .
```

where `N_DEVC` is the number of devices, `ID(I)` is the user-defined ID of the *I*th device, `UNITS(I)` the units, `T(J)` the time of the *J*th dump, and `VAL(I,J)` the value at the *I*th device at the *J*th time. The files can be imported into Microsoft Excel or almost any other spreadsheet program. If the number of columns exceeds 256, the file will automatically be split into smaller files.

22.4 Control Output Data

Data associated with particular control functions specified in the input file under the namelist group `CTRL` is output in comma delimited format in a file called `CHID_ctrl.csv`. The format of the file is as follows:

```
s      , status , status , ... , status
Time  , ID(1)    , ID(2)    , ... , ID(N_CTRL)
T(1)  , -001     , 001      , ... , -001
      .
      .
```

where `N_CTRL` is the number of controllers, `ID(I)` is the user-defined ID of the `I`th control function, and plus or minus 1's represent the state `-1 = .FALSE.` and `+1 = .TRUE.` of the `I`th control function at the particular time. The files can be imported into Microsoft Excel or almost any other spreadsheet program. If the number of columns exceeds 256, the file will automatically be split into smaller files.

22.5 CPU Usage Data

The file called `CHID_cpu.csv` records the amount of CPU time for each of the MPI processes.

```
Rank,MAIN,DIVG, ... , Total T_USED (s)
0, 2.052E+00, 1.058E+01, ... , 5.143+01
1, 2.432E+00, 1.062E+01, ... , 5.123+01
.
.
```

where `Rank` is the number of the MPI process (starting at 0), `MAIN`, `DIVG`, and so on, are major routines, and 'Total T_USED (s)' is the total CPU time consumed by that particular MPI process. Typically, the total time is similar. The time spent in `MAIN` is essentially overhead – time spent *not* working on the calculation. If you want to know if your work load is balanced, take a look at the time spent in `MAIN`. It should be similar for all MPI processes. If one of the MPI processes has a noticeably smaller value for `MAIN`, then that process is working on the core routines while the other processes sit idle in `MAIN`.

The `CHID_cpu.csv` file is printed out at the end of the simulation. To force it to be printed out periodically during the simulation, set `DT_CPU` on the `DUMP` line. Its default value is very large, meaning that by default, the CPU information is only printed at the end of the simulation.

22.6 Gas Mass Data

The total mass of the various gas species at any instant in time is reported in the comma delimited file `CHID_mass.csv`. The file consists of several columns, the first column containing the time in seconds, the second contains the total mass of all the gas species in the computational domain in units of kg, the next lines contain the total mass of the individual species.

You must specifically ask that this file be generated, as it can potentially cost a fair amount of CPU time to generate. Set `MASS_FILE=.TRUE.` on the `DUMP` line to create this output file.

22.7 Slice Files

The slice files defined under the namelist group `SLCF` are named `CHID_n.sf` ($n=01,02,\dots$), and are written out unformatted, unless otherwise directed. These files are written out from `dump.f90` with the following lines:

```
WRITE(LUSF) QUANTITY
WRITE(LUSF) SHORT_NAME
WRITE(LUSF) UNITS
WRITE(LUSF) I1,I2,J1,J2,K1,K2
WRITE(LUSF) TIME
WRITE(LUSF) ((QQ(I,J,K),I=I1,I2),J=J1,J2),K=K1,K2)
.
.
WRITE(LUSF) TIME
```

```
WRITE (LUSF) ( ( (QQ (I, J, K), I=I1, I2), J=J1, J2), K=K1, K2)
```

QUANTITY, SHORT_NAME and UNITS are character strings of length 30. The sextuplet (I1, I2, J1, J2, K1, K2) denotes the bounding mesh cell nodes. The sextuplet indices correspond to mesh cell nodes, or corners, thus the entire mesh would be represented by the sextuplet (0, IBAR, 0, JBAR, 0, KBAR).

There is a short Fortran 90 program provided, called `fds2ascii.f90`, that can convert slice files into text files that can be read into a variety of graphics packages. The program combines multiple slice files corresponding to the same “slice” of the computational domain, time-averages the data, and writes the values into one file, consisting of a line of numbers for each node. Each line contains the physical coordinates of the node, and the time-averaged quantities corresponding to that node. In particular, the graphics package Tecplot reads this file and produces contour, streamline and/or vector plots. See Section 18.11 for more details about the program `fds2ascii`.

22.8 Plot3D Data

Quantities over the entire mesh can be output in a format used by the graphics package Plot3D. The Plot3D data sets are single precision (32 bit reals), whole and unformatted. Note that there is blanking, that is, blocked out data points are not plotted. If the statement `WRITE_XYZ=.TRUE.` is included on the DUMP line, then the mesh data is written out to a file called `CHID.xyz`

```
WRITE (LU13) IBAR+1, JBAR+1, KBAR+1
WRITE (LU13) ( ( (X (I), I=0, IBAR), J=0, JBAR), K=0, KBAR), &
              ( ( (Y (J), I=0, IBAR), J=0, JBAR), K=0, KBAR), &
              ( ( (Z (K), I=0, IBAR), J=0, JBAR), K=0, KBAR), &
              ( ( (IBLK (I, J, K), I=0, IBAR), J=0, JBAR), K=0, KBAR)
```

where X, Y and Z are the coordinates of the cell corners, and IBLK is an indicator of whether or not the cell is blocked. If the point (X, Y, Z) is completely embedded within a solid region, then IBLK is 0. Otherwise, IBLK is 1. Normally, the mesh file is not dumped.

The flow variables are written to a file called `CHID_****_*.q`, where the stars indicate a time at which the data is output. The file is written with the lines

```
WRITE (LU14) IBAR+1, JBAR+1, KBAR+1
WRITE (LU14) ZERO, ZERO, ZERO, ZERO
WRITE (LU14) ( ( (QQ (I, J, K, N), I=0, IBAR), J=0, JBAR), K=0, KBAR), N=1, 5)
```

The five channels N=1, 5 are by default the temperature (°C), the *u*, *v* and *w* components of the velocity (m/s), and the heat release rate per unit volume (kW/m³). Alternate variables can be specified with the input parameter `PLOT3D_QUANTITY(1:5)` on the DUMP line. Note that the data is interpolated at cell corners, thus the dimensions of the Plot3D data sets are one larger than the dimensions of the computational mesh.

Smokeview can display the Plot3D data. In addition, the Plot3D data sets can be read into some other graphics programs that accept the data format. This particular format is very convenient, and recognized by a number of graphics packages.

22.9 Boundary Files

The boundary files defined under the namelist group BNDF are named `CHID_n.bf` (*n*=0001,0002...), and are written out unformatted. These files are written out from `dump.f90` with the following lines:

```

WRITE(LUBF) QUANTITY
WRITE(LUBF) SHORT_NAME
WRITE(LUBF) UNITS
WRITE(LUBF) NPATCH
WRITE(LUBF) I1, I2, J1, J2, K1, K2, IOR, NB, NM
WRITE(LUBF) I1, I2, J1, J2, K1, K2, IOR, NB, NM

      .
      . WRITE(LUBF) TIME
WRITE(LUBF) (( (QQ(I, J, K), I=11, I2), J=J1, J2), K=K1, K2)
WRITE(LUBF) (( (QQ(I, J, K), I=11, I2), J=J1, J2), K=K1, K2)
      .
      . WRITE(LUBF) TIME
WRITE(LUBF) (( (QQ(I, J, K), I=11, I2), J=J1, J2), K=K1, K2)
WRITE(LUBF) (( (QQ(I, J, K), I=11, I2), J=J1, J2), K=K1, K2) .
      .

```

QUANTITY, SHORT_NAME and UNITS are character strings of length 30. NPATCH is the number of planes (or “patches”) that make up the solid boundaries plus the external walls. The sextuplet (I1, I2, J1, J2, K1, K2) defines the cell nodes of each patch. IOR is an integer indicating the orientation of the patch ($\pm 1, \pm 2, \pm 3$). You do not prescribe these. NB is the number of the boundary (zero for external walls) and NM is the number of the mesh. Note that the data is planar, thus one pair of cell nodes is the same. Presently, Smokeview is the only program available to view the boundary files.

22.10 Particle Data

Coordinates and specified quantities related to tracer particles, sprinkler droplets, and other Lagrangian particles are written to a FORTRAN unformatted (binary) file called CHID.prt5. Note that the format of this file has changed from previous versions (4 and below). The file consists of some header material, followed by particle data output every DT_PART seconds. The time increment DT_PART is specified on the DUMP line. It is T_END/NFRAMES by default. The header materials is written by the following FORTRAN code in the file called dump.f90.

```

WRITE(LUPF) ONE_INTEGER           ! Integer 1 to check Endian-ness
WRITE(LUPF) NINT(VERSION*100.)    ! FDS version number
WRITE(LUPF) N_PART                ! Number of PARTICle classes
DO N=1, N_PART
  PC => PARTICLE_CLASS(N)
  WRITE(LUPF) PC%N_QUANTITIES, ZERO_INTEGER ! ZERO_INTEGER is a place holder
  DO NN=1, PC%N_QUANTITIES
    WRITE(LUPF) CDATA(PC%QUANTITIES_INDEX(NN)) ! 30 character output quantity
    WRITE(LUPF) UDATA(PC%QUANTITIES_INDEX(NN)) ! 30 character output units
  ENDDO
ENDDO

```

Note that the initial printout of the number 1 is used by Smokeview to determine the Endian-ness of the file. The Endian-ness has to do with the particular way real numbers are written into a binary file. The version number is used to distinguish new versus old file formats. The parameter N_PART is not the number of particles, but rather the number of particle classes corresponding to the PART namelist groups in the input file. Every DT_PART seconds the coordinates of the particles and droplets are output as 4 byte reals:

```

WRITE(LUPF) REAL(T, FB) ! Write out the time T as a 4 byte real
DO N=1, N_PART
  WRITE(LUPF) NPLIM      ! Number of particles in the PART class

```

```

WRITE(LUPF) (XP(I),I=1,NPLIM), (YP(I),I=1,NPLIM), (ZP(I),I=1,NPLIM)
WRITE(LUPF) (TA(I),I=1,NPLIM) ! Integer "tag" for each particle
IF (PC%N_QUANTITIES > 0) WRITE(LUPF) ((QP(I,NN),I=1,NPLIM),NN=1,PC%N_QUANTITIES)
ENDDO

```

The particle “tag” is used by Smokeview to keep track of individual particles and droplets for the purpose of drawing streamlines. It is also useful when parsing the file. The quantity data, $QP(I, NN)$, is used by Smokeview to color the particles and droplets. Note that it is now possible with the new format to color the particles and droplets with several different quantities.

22.11 Profile Files

The profile files defined under the namelist group `PROF` are named `CHID_prof_nn.csv` ($nn=01,02\dots$), and are written out formatted. These files are written out from `dump.f90` with the following line:

```

WRITE(LU_PROF) T, NWP+1, (X_S(I), I=0, NWP), (Q(I), I=0, NWP)

```

After the time T , the number of node points is given and then the node coordinates. These are written out at every time step because the wall thickness and the local solid phase mesh may change over time due to the solid phase reactions. Array Q contains the values of the output quantity, which may be wall temperature, density or component density.

22.12 3-D Smoke Files

3-D smoke files contain alpha values used by Smokeview to draw semi-transparent planes representing smoke and fire. FDS outputs 3-D smoke data at fixed time intervals. A *pseudo-code* representation of the 3-D smoke file is given by:

```

WRITE(LU_SMOKE3D) ONE, VERSION, 0, NX-1, 0, NY-1, 0, NZ-1
.
.
WRITE(LU_SMOKE3D_SIZE, *) TIME, NCHARS_IN, NCHARS_OUT
WRITE(LU_SMOKE3D) TIME
WRITE(LU_SMOKE3D) NCHARS_IN, NCHARS_OUT
IF (NCHARS_OUT > 0) WRITE(LU_SMOKE3D) (BUFFER_OUT(I), I=1, NCHARS_OUT)

```

The first `ONE` is an endian flag. Smokeview uses this number to determine whether the computer creating the 3-D smoke file and the computer viewing the 3-D smoke file use the same or different byte swap (endian) conventions for storing floating point numbers. The opacity data is converted from 4 byte floating point numbers to one byte integers then compressed using run-length encoding (RLE). The compressed data is contained in `BUFFER_OUT`. Run-length encoding is a compression scheme where repeated “runs” of data are replaced with a number (number of repeats), and the value repeated. Four or more consecutive identical characters are represented by $\#nc$ where $\#$ is a special character denoting the beginning of a repeated sequence, n is the number of repeats and c is the character repeated. n can be up to 254 (255 is used to represent the *special* character). Characters not repeated four or more times are listed as is. For example, the character string `aaaaaabbbbbbcc` is encoded as `#6a#5bcc`.

22.13 Geometry, Isosurface Files

Both immersed geometric surfaces (generalized obstructions) and FDS generated isosurfaces are stored using a file format described in this section. Iso-surface files are used to store one or more surfaces where the specified QUANTITY is a specified value. FDS outputs iso-surface data at fixed time intervals. These surfaces are defined in terms of vertices and triangles. A vertex consists of an (x,y,z) coordinate. A triangle consists of 3 connected vertices. The file format allows one to specify objects that change with time. Static geometry is defined once and displayed by Smokeview unchanged at each time step. Dynamic geometry is defined at each time step either in terms of nodes and faces or in terms of a translation and two rotations (azimuthal and elevation) of dynamic geometry defined in the first time step. These files are written out from `dump.f90` using lines equivalent to the following:

```
! header

WRITE(LU_GEOM) ONE
WRITE(LU_GEOM) VERSION
WRITE(LU_GEOM) N_FLOATS
IF (N_FLOATS>0) WRITE(LU_GEOM) (FLOAT_HEADER(I),I=1,N_FLOATS)
WRITE(LU_GEOM) N_INTS
IF (N_INTS>0) WRITE(LU_GEOM) (INT_HEADER(I),I=1,N_INTS)

! static geometry - geometry specified once and appearing at all time steps

WRITE(LU_GEOM) N_VERT_S, N_FACE_S
IF (N_VERT_S>0) WRITE(LU_GEOM) (Xvert_S(I),Yvert_S(I),Zvert_S(I),I=1,N_VERT_S)
IF (N_FACE_S>0) WRITE(LU_GEOM) (FACE1_S(I),FACE2_S(I),FACE3_S(I),I=1,N_FACE_S)
IF (N_FACE_S>0) WRITE(LU_GEOM) (SURF_S(I),I=1,N_FACE_S)

! dynamic geometry - geometry specified and appearing for each time step

WRITE(LU_GEOM) STIME, GEOM_TYPE
IF (GEOM_TYPE.EQ.0) THEN
  WRITE(LU_GEOM) N_VERT_D, N_FACE_D
  IF (N_VERT_D>0) WRITE(LU_GEOM) (Xvert_D(I),Yvert_D(I),Zvert_D(I),I=1,N_VERT_D)
  IF (N_FACE_D>0) WRITE(LU_GEOM) (FACE1_D(I),FACE2_D(I),FACE3_D(I),I=1,N_FACE_D)
  IF (N_FACE_D>0) WRITE(LU_GEOM) (SURF_D(I),I=1,N_FACE_D)
ELSE IF (GEOM_TYPE.EQ.1) THEN
  ! rotation and translation parameters used to transform geometry from first
  ! dynamic time step
  WRITE(LU_GEOM) Xtran, Ytran, Ztran, Xrot0, Yrot0, Zrot0, rot_az, rot_elev
ENDIF
.
```

- ONE has the value 1. Smokeview uses this number to determine whether the computer creating the geometry file and the computer viewing the geometry file use the same or different byte swap (endian) conventions for storing floating point numbers.
- VERSION currently has value 0 and indicates the version number of this file format.
- N_FLOATS, N_INTS The number of floating point and integer data items stored at the beginning of the file.
- FLOAT_HEADER, INT_HEADER Floating point and integer data stored at the beginning of the file.

- `STIME` is the FDS simulation time.
- `N_VERT_S`, `N_FACE_S`, `N_VERT_D`, `N_FACE_D` are the number of static and dynamic vertices and faces.
- `Xvert_S`, `Yvert_S`, `Zvert_S`, `Xvert_D`, `Yvert_D`, `Zvert_D` are the static and dynamic vertex coordinates.
- `FACE1_S`, `FACE2_S`, `FACE3_S`, `FACE1_D`, `FACE2_D`, `FACE3_D` are the static and dynamic vertex indices for each face (triangle). The indices are numbered relative to how vertices were written out earlier.
- `SURF_S`, `SURF_D` are the static and dynamic SURF indices for each face (triangle).
- `GEOM_TYPE` is flag indicating how dynamic geometry is represented. If `GEOM_TYPE` is 0 then time dependent geometry is written out in terms of nodes and faces using the same format as the static geometry. If `GEOM_TYPE` is 1 then time dependent geometry is written out in terms of a translation and two rotations. These transformations are applied to the dynamic geometry defined at the first time step.
- `Xtran`, `Ytran`, `Ztran` is the translation applied to the initial dynamic geometry (If `GEOM_TYPE` is 1)
- `Xrot0`, `Yrot0`, `Zrot0` is the origin about which rotations occur.
- `rot_az`, `rot_elev` are the azimuthal and elevation rotation angles (in degrees) applied to the initial dynamic geometry.

22.14 Geometry Data Files

The geometry data file contains a description of data values computed by FDS on an immersed geometrical objects. This file is analogous to the boundary file. The data written out to a geometry data file **MUST** correspond to the geometry written out in the corresponding geometry file. Geometry data files are written out from `dump.f90` with the lines equivalent to the following:

```
WRITE(LU_GEOM_DATA) ONE
WRITE(LU_GEOM_DATA) VERSION
WRITE(LU_GEOM_DATA) STIME
WRITE(LU_GEOM_DATA) N_VERT_S_VALS,N_VERT_D_VALS,N_FACE_S_VALS,N_FACE_D_VALS
IF (N_VERT_S_VALS>0) WRITE(LU_GEOM_DATA) (ValVertStatic(I), I=1,N_VERT_S_VALS)
IF (N_VERT_D_VALS>0) WRITE(LU_GEOM_DATA) (ValVertDynamic(I), I=1,N_VERT_D_VALS)
IF (N_FACE_S_VALS>0) WRITE(LU_GEOM_DATA) (ValFaceStatic(I), I=1,N_FACE_S_VALS)
IF (N_FACE_D_VALS>0) WRITE(LU_GEOM_DATA) (ValFaceDynamic(I), I=1,N_FACE_D_VALS)
.
```

The data values written out in this file correspond to the geometry written out in the geometry file.

- `ONE` has the value 1. Smokeview uses this number to determine whether the computer creating the geometry file and the computer viewing the geometry file use the same or different byte swap (endian) conventions for storing floating point numbers.
- `VERSION` currently has value 0 and indicates the version number of this file format.
- `STIME` is the FDS simulation time.

- `N_VERT_S_VALS`, `N_FACE_S_VALS` is the number of data values written out for static vertices and faces. One can write out data values located at nodes, located at the center of faces or both.
- `N_VERT_D_VALS`, `N_FACE_D_VALS` is the number of dynamic values written out for dynamic vertices and faces. One can write out data values located at nodes, located at the center of faces, both or neither (if there is no dynamic geometry).
- `ValVertStatic`, `ValFaceStatic` static vertex and face data.
- `ValVertDynamic`, `ValFaceDynamic` dynamic vertex and face data.

