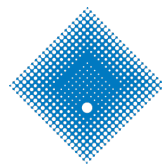


# **Development of an interface using penalisation method for improving computational performance of bushfire simulation tools**

Sesa Singha Roy

Supervisor:  
A/Prof. Khalid Moinuddin  
Dr. Duncan Sutherland

Thesis submitted in fulfilment of the requirements for the degree of  
Master of Engineering by Research



**VICTORIA  
UNIVERSITY**

---

MELBOURNE AUSTRALIA

Institute for Sustainable Industries and Liveable Cities (formerly CESARE)  
Victoria University

May 2019

# ABSTRACT

Wind is the dominant environmental factor affecting wildland fire intensity and spread. Previously, fire analysts and managers have relied on local measurements and site-specific forecasts to determine how winds influence fire. The advancements in computer hardware, increased availability of electronic topographical and experimental data, and advances in numerical methods for computing winds, have led to the development of new tools capable of simulating wind flow. Several numerical models have been developed for fire prediction and forecasting. Modelling wind in physics-based models like Fire Dynamics Simulator (FDS) has been shown to produce promising results, but at an inordinate cost. Because of the high computational expense, physics-based models are not suitable for operational use. Little research has been conducted to improve the computational speed of these models. The current study intends to decrease the computational cost of physics-based fire simulations and improve physics-based models by including more complicated driving winds.

Physics-based wildfire simulations are driven by inlet boundary conditions which model the atmospheric boundary layer. Various inlet conditions, such as the 1/7-powerlaw or the log law models with artificial turbulence (e.g. the synthetic eddy method, [SEM]) can be used as an inlet to generate a statistically steady wind field for a fire simulation. The power-law inlet is the default inlet condition used in FDS where the wind develops turbulence as it sweeps through the domain, and is often used with *wall-of-wind* type methods. The log-law inlet generates a log wind profile similar to Atmospheric Boundary Layer (ABL). Development of techniques for imposing inlet conditions and initial conditions for flow simulations have been topics of interest for the past few decades. Current inlet and initial conditions requires time in a scale order of 100s of CPU hours, for gen-

---

erating an appropriate condition to start a fire simulation, hence resulting in increased computational expense. A novel nesting method has been implemented, which involves two regions : penalisation and blending, named as the *PenaBlending* method. The initial conditions of the fire simulations in FDS are set to the initial condition prescribed by an external model or simulation. This is achieved by a one-way coupling method. External wind data, for which  $u,v,w$  can vary in space and time, can be obtained. The precursor data can be generated either from any reduced wind model such as *Windninja*, which gives terrain modified wind data, or by using analytical methods such as generating logarithmic windfield using Matlab. These external data can be introduced into the FDS domain through a penalization region at the inlet/outlet. A blending region has also been implemented near the specified inlet/outlet which allows a smooth mixing of a precursor wind field to that in the simulation domain. This new inlet condition allows complicated terrain modified temporally and spatially varying wind fields, obtained from precursor simulations or any other models, to be implemented relatively easily in the FDS domain. To test the implementation of this method, a flat terrain is considered in the current study. However, this method could also be used for complicated terrain structures, as a part of future studies. The *PenaBlending* method provides appropriate flow conditions with reduced computational effort (up to  $\sim 80\%$ ), to start a fire simulation, and, hence, reduces the computational expense of physics-based models.

The results obtained using the *PenaBlending* method have been compared with that obtained using the existing inlet conditions of FDS, like the SEM method, wall-of-wind method and mean-forcing methods, using the 1/7 power-law or log-law inlets. To test these three methods, a set of fire simulations have been conducted and tested against the *PenaBlending* method. It was found that the results of the *PenaBlending* methods agree well with that of existing methods, with small variations for both the *wind* and *fire* cases.

FDS 6.6.0 (the version used in this study) requires a very fine grid to obtain grid convergence. This is not feasible in the case of a large-scale simulation because of very high computation cost. FDS 6.2.0, with a reaction-rate-limiter combustion model, needs less fine grids to obtain grid convergence. Therefore, this combustion model is re-introduced

---

into FDS 6.6.0, providing an option of choosing between two different combustion models, as a part of this study. For all the simulations, the reaction-rate-limiter combustion model has been used. The simulations are carried out in a neutral-atmospheric stability condition. However, the PenBlending method can apply any general driving wind, and the effect of atmospheric stability, could be included, as part of future studies. The PenBlending method could be extended in conjunction with Monin-Obukhov Similarity Theory (introduced in FDS 6.6.0) to model fire in various atmospheric stability conditions.



# DECLARATION

I, Sesa Singha Roy, declare that the Master by Research thesis entitled '**Development of an interface using penalisation method for improving computational performance of bushfire simulation tools**' is no more than 60,000 words in length including quotes and exclusive of tables, figures, appendices, bibliography, references and footnotes. This thesis contains no material that has been submitted previously, in whole or in part, for the award of any other academic degree or diploma. Except where otherwise indicated, this thesis is my own work.



Sesa Singha Roy

29 May, 2019

# ACKNOWLEDGEMENT

Prima facie, I am grateful to my parents, Dr. Dilip Kumar Singha Roy and Sima Singha Roy, for their blessings and trust that gave me good health, wellbeing and support necessary to complete my journey of Masters by Research. This journey has become reality with the help of several notable people to whom I would like to express my sincere gratitude.

First and foremost, I would like to express my heartfelt gratitude to my supervisors, A/Prof. Khalid Moinuddin and Dr. Duncan Sutherland for their genuine support, constant guidance, motivation and continuous encouragement throughout my journey. Their vast knowledge about the subject, expertise and constant supervision entitled me to complete my work to the best of my abilities. They went above and beyond their way to help me without any hesitation and open up opportunities for me, which I will be grateful forever.

I place on record, my sincere thanks to you, A/Prof. Paul Joseph, for your positive motivation, untiring guidance and vital advice to help me achieve this milestone.

I thank Victoria University for granting me VU Strategic Research Scholarship 2017 (Bushfires) to fund my research. I am also grateful to Bushfire & Natural Hazards Cooperative Research Centre (BNHCRC) for entitling me as an associate student and providing me various opportunities throughout my tenure. This research was undertaken using the HPC-SPARTAN facility hosted at the University of Melbourne, and I would like to thank the support team for their promptness of resolving issues whenever required.

---

I thank all my fellow labmates and friends for all stimulating discussions, mental support, motivation and for all the fun we have had in the last two years.

Last but not the least, I thank my little sister, Sera, for all her discussions, encouragement, support and unconditional love through the thick n' thin of my journey. Most importantly, above all, I would like to thank my husband and best friend, Debaditya Acharya, for his endless support, inspiration, motivation, endless discussions, patience and love which kept me focussed and made my journey flawless and smooth.

I also place on record, my sense of gratitude to one and all, who directly or indirectly, have lent their hand in this venture.

Sesa Singha Roy

May, 2019

# Contents

<b>ABSTRACT</b>	<b>ix</b>
<b>DECLARATION BY AUTHOR</b>	<b>ix</b>
<b>ACKNOWLEDGEMENT</b>	<b>ix</b>
<b>LIST OF FIGURES</b>	<b>ix</b>
<b>LIST OF TABLES</b>	<b>ix</b>
<b>LIST OF TERMS AND ABBREVIATIONS</b>	<b>ix</b>
<b>1 Introduction</b>	<b>5</b>
1.1 Wildland fires . . . . .	6
1.2 Fire behaviour models . . . . .	7
1.2.1 Wildland fire models . . . . .	8
1.3 Problems and Motivation . . . . .	11
1.4 Addressing an Omission in Previous Research . . . . .	12
<b>2 Physics-based modelling</b>	<b>15</b>
2.1 The model's description . . . . .	15
2.1.1 Mass Continuity . . . . .	16
2.1.2 Conservation of Momentum . . . . .	16
2.1.3 Turbulence Model . . . . .	17
2.1.4 Wall Model . . . . .	18
2.1.5 Pyrolysis . . . . .	19
2.1.6 Combustion . . . . .	20
2.2 Traditional methods of wind field generation . . . . .	21

---

2.2.1	Wall of Wind method . . . . .	21
2.2.2	Synthetic Eddy Method . . . . .	22
2.2.3	Mean-forcing method . . . . .	23
<b>3</b>	<b>Methodology and Code development</b>	<b>25</b>
3.1	Assessment of Inlet and Initial Condition for fire simulations . . . . .	26
3.1.1	Pena-Blending method . . . . .	28
3.2	Re-introduction of FDS 6.2.0 Combustion Model into FDS 6.6.0 . . . . .	44
3.2.1	Implementing the re-introduced combustion model in FDS 6.6.0 . . . . .	45
3.2.2	Verification Cases . . . . .	47
<b>4</b>	<b>Test Results and Discussions</b>	<b>53</b>
4.1	Simulation Domain . . . . .	53
4.2	Boundary Conditions . . . . .	56
4.3	Wind-only cases - results and discussion . . . . .	60
4.4	Fire cases : results and discussions . . . . .	67
4.5	Effect of under-developed wind field on fire simulation . . . . .	78
4.6	Gusting effect of wind on fire . . . . .	80
4.7	Modelling wind using reduced wind model to use with <i>PenaBlending method</i>	83
<b>5</b>	<b>Conclusion and Future Directions</b>	<b>90</b>
5.1	Future work and recommendations . . . . .	93
	<b>Appendices</b>	<b>104</b>
<b>A</b>	<b>Source Code files for edited portions of FDS</b>	<b>105</b>
<b>B</b>	<b>Publication</b>	<b>326</b>
<b>C</b>	<b>Monin-Obukhov Similarity Theory</b>	<b>331</b>

# List of Figures

1.1	Schematic representation of wind-driven fires (Rothermel (1972)) . . . . .	5
1.2	Black Saturday Wildfires, Victoria (Source: <a href="http://australianrotaryhealth.org.au">australianrotaryhealth.org.au</a> )	7
1.3	An overview of wildland fire behaviour and their relation to its fire modelling . . . . .	8
3.1	Schematic representation of research methodology . . . . .	25
3.2	Schematic representation of turbulent eddies in FS domain using the Pen-aBlending method . . . . .	30
3.3	Flowchart representation of the pre-processing algorithm for <i>pSim</i> data . .	38
3.4	Format for writing out <i>pSim</i> data . . . . .	39
3.5	Domain setup for the couch case . . . . .	48
3.6	Heat Release Rates comparison of in-built combustion model for edited and unedited versions of FDS 6.6.0 for couch case . . . . .	49
3.7	Heat Release Rates comparison of FDS 6.2.0 combustion model and identical model introduced in FDS 6.6.0 for couch case . . . . .	49
3.8	Tree burning case: This figure shows the temperature contours of burning the tree represented by the <i>green triangle</i> . The red colour represents the maximum temperature ( $1020^{\circ}\text{C}$ ),the fire plume region and the blue colour represents minimum temperature, the smoke region . . . . .	51
3.10	Heat Release Rates comparison of FDS 6.2.0 combustion model and identical model introduced in FDS 6.6.0 for the tree burning case . . . . .	51

---

3.9	Heat Release Rates comparison of in-built combustion model for edited and unedited versions of FDS 6.6.0 for the tree burning case . . . . .	52
4.1	A generalised schematic representation of <i>Small Domain</i> for simulation, representing the external dimensions, fire-line, fire plot and a slice of establishing ABL . . . . .	55
4.2	A generalised schematic representation of the <i>large domain</i> for a simulation representing the external dimensions, fire-line, fire plot and a slice of establishing ABL . . . . .	56
4.3	A generalised schematic representation of the domain set-up with boundary conditions of <i>small domain</i> cases corresponding to Table-3.3-b for (a) wind1; (b) wind2; (c) wind3; (d) wind4, including the external dimensions, fire-line, fire plot and a slice of establishing ABL. The fire simulations have a similar set-up . . . . .	58
4.4	A generalised schematic representation of the domain set-up with boundary conditions of <i>large domain</i> cases corresponding to Table-3.3-b for (a) wind1; (b) wind2; (c) wind3; (d) wind4, including the external dimensions, fire-line, fire plot and a slice of establishing ABL. The fire simulations have a similar set-up . . . . .	59
4.5	Development of wind-field using the PenaBlending method for large domain	60
4.6	The mean velocity profiles are plotted over the fire-ground for wall-of-wind method (wind1), SEM method (wind2), mean-forcing method (wind3) and the PenaBlending method (wind4) for: (a)Small domain of 130m X 40m X 80m ; and (b) Large Domain of 600m X 300m X 100m. . . . .	65
4.7	The mean velocity profiles are plotted over the fire-ground for wall-of-wind method (wind1), SEM method (wind2), mean-forcing method (wind3) and the PenaBlending method (wind4) in semi-logarithmic scale for: (a)Small domain of 130m X 40m X 80m ; and (b) Large Domain of 600m X 300m X 100m. It is observed that the profiles converges with the theoretical log-law plot and shows a clear logarithmic layer. . . . .	66

4.8	The fire propagation contour for a small domain with a fire-plot of (40m X 40m) where the green area represents the 'burnable grass plot' and the non green area represents the 'non-burnable grassplot'. The propagation of fire is represented at various time-steps for (a) fire1; (b) fire2; (c) fire3; (d) fire4 cases. The first three cases requires almost equal times (~ 24s) to burn through the fire plot, whereas fire4 requires much lesser time (~ 18s).	69
4.9	The fire propagation contour for a large domain with fire plot of (100m X 300m) where the green area represents the 'burnable grass plot' and the non green area represents the 'non-burnable grassplot'. The propagation of fire is represented at various time-steps for (a)fire1; (b)fire2; (c)fire3; (d)fire4 cases. The first two cases requires almost equal times (~ 50s) to burn through the fire plot, whereas fire3 requires ~ 45s and fire4 requires ~ 38s to do so. . . . .	70
4.10	The fire front ( $x_*$ ) location in four fire cases ( <i>fire1; fire2; fire3; fire4</i> ) as a function of time for (a) small domain and (b) large domain, over the fire-plot. .	72
4.11	The boundary temperature contours showing the fire-front propagation over the fire plot for all the fire simulation cases:(a) fire1; (b) fire2; (c) fire3; (d) fire4 for (i) small domain and (ii)large domain. The legend shows the temperature variation in K. The pyrolysis region is obtained when temperature becomes greater than 400K, which is represented by the yellow contours. . . . .	73
4.12	The Heat Release Rates (HRR) as a function of time for the fire simulations using wall-of-wind method (fire1), SEM method (fire2), mean-forcing method (fire3) and the PenaBlending method (fire4) for: (a)Small domain of 130m X 40m X 80m ; and (b) Large domain of 600m X 300m X 100m. . .	75
4.13	The rate-of-spread of fire as a function of time using wall-of-wind method (fire1), SEM method (fire2), mean-forcing method (fire3) and the PenaBlending method (fire4) for: (a)Small domain of 130m X 40m X 80m ; and (b) Large Domain of 600m X 300m X 100m. . . . .	77



---

4.14 The RoS comparison for underdeveloped-wind field for fire simulations (fire5) compared with the wall-of-wind method (fire1) for: (a) small domain and (b) large domain. In both the cases, all the conditions parameters used, boundary conditions and inlet method are same . . . . .	79
4.15 The average boundary temperature contours: (a) <i>velo1</i> with $u_{10} = \sim 4.8$ m/s is read at 2 s after the simulation starts ; (b) <i>velo2</i> with $u_{10} = \sim 7$ m/s is read at 2 s after the start of simulation ; (c) This figure shows the average boundary temperature contour when a gust of wind is applied in the middle of fire simulation. <i>velo2</i> is read at 20 s after the start of ignition, in the middle of burning, with <i>velo1</i> read initially at 2 s after the start of simulation . . . . .	81
4.16 Rate-of-spread comparison of fire simulations with velocity1( <i>velo1</i> ), velocity2( <i>velo2</i> ) and gusting effect of <i>velo1</i> and <i>velo2</i> . The area marked with a red semi-circle shows how an increase in velocity during a fire propagation changes the RoS and hence blows the plume out of the domain faster. . . .	82
4.17 The 3D grid representation of the simulated wind field in Windninja. The grid generated is parallel to the underlying terrain. . . . .	84
4.18 (a) The original and interpolated uniform grids with required resolution similar to FDS domain. The mean velocity profiles for the original wind data and the interpolated data. The y-axis in the velocity profiles represents the grid points along z-axis; 20 grid points along z-axis have been interpolated to 80 grid points as per the requirement, keeping the vertical distance constant; (b) The Y-Z plane showing the original data from Windninja with coarser non-uniform grid resolution and the cut-out uniform grid Windninja data with fine resolution of $1\mathbf{m} \times 1\mathbf{m}$ used as pSim data for FDS. . . . .	86

---

4.19 Development of wind profile over the domain using Windninja data over various time-steps: (a) at time=0s; (b) at time= 2s, when the Windninja data is read at inlet/outlet; (c) at time=10s, depicting the wind developing from the inlet; (d) at time=50s, depicting a fully developed wind profile obtained. . . . .	87
4.20 Mean velocity profile over the fire-plot using Windninja data as pSim data for PenaBlending method. . . . .	88
4.21 The fire propagation over the fire plot using Windninja data at: time=0s; time=10s; time=14s; time=18s. . . . .	89
4.22 (a) The RoS profile for fire over the fire plot using Windninja data; (b) The boundary temperature contour showing the fire front propagation over the fire plot . . . . .	89
C.1 Mean velocity profile over the fire-plot at various atmospheric stabilities: (a) The $u_{10}$ velocities for different stabilities are matched immediately before the ignition line at $\sim 7\text{m/s}$ , the inlet velocity may vary ; (b) The $u_{10}$ velocities matched at the inlet to $6.5\text{m/s}$ for different stabilities. . . . .	333
C.2 The HRR plot for the fire simulation at different atmospheric stabilities: (a) The $u_{10}$ velocity is matched immediately before the ignition line to $\sim 7\text{m/s}$ for various stabilities; (b) The $u_{10}$ velocity is matched at the inlet to $6.5\text{m/s}$ for various stabilities. . . . .	334
C.3 The RoS of the fire over the fire plot at different atmospheric stabilities; (a) The $u_{10}$ velocity matched immediately before the ignition line to $\sim 7\text{m/s}$ for various stabilities; (b) The $u_{10}$ velocity is matched at the inlet to $6.5\text{m/s}$ for various stabilities. . . . .	335

# List of Tables

1.1	Some examples of (a) <i>Empirical Models</i> and (b) <i>Simulators</i> . . . . .	9
3.1	Wind simulation cases - for both <i>large</i> and <i>small</i> domain . . . . .	41
3.2	Fire simulation cases - for both <i>large</i> and <i>small</i> domain . . . . .	41
3.3	Simulation parameter values and characteristics - (a) Numerical parameters used for <i>small domain</i> and <i>large domain</i> for both <i>fire</i> as well as <i>wind-only</i> simulations mentioned in 3.2 and 3.1; (b) List of boundary conditions used for <i>wind-only</i> and <i>fire</i> simulations; (c) Fuel parameters used for running <i>fire simulations</i> . . . . .	42
3.4	FDS input parameters to be used for selecting the combustion model . . . . .	47
4.1	Sampling time for <i>large</i> and <i>small</i> domain . . . . .	61
4.2	Parameters used in the <i>PenaBlending method</i> for both <i>large</i> and <i>small</i> domain . . . . .	63
4.3	Time for the flame to reach the end of fire-plot for <i>small</i> and <i>large</i> domains . . . . .	76
C.1	Different Atmospheric stability parameters . . . . .	332

# LIST OF TERMS AND ABBREVIATIONS

WUI	Wild-land Urban Interface
RoS	Rate-of-Spread
TRANS	Transient Reynolds Average Navier-Stokes
LES	Large Eddy Simulation
WFDS	Wildland-urban Interface Fire Dynamics Simulator
FDS	Fire Dynamics Simulator
ABL	Atmospheric Boundary Layer
NIST	National Institute of Standards and Technology
FORTTRAN	FORmula TRANslator
CFD	Computational Fluid Dynamics
RANS	Reynolds Averaged Navier-Stokes
SGS	Sub-Grid Scale
SEM	Synthetic Eddy Method
<i>pSim</i>	precursor simulation
FS	Fire-spread Simulation
HRR	Heat Release Rate
HRRPUV	Heat Release Rate per Unit Volume

---

$\rho$	fluid density
$U$	instantaneous velocity
$p$	pressure
$g$	gravity
$\tau_{ij}$	viscous stress tensor
$\bar{U}$	filtered velocity
$\nu$	molecular viscosity
$S_{ij}$	strain-rate tensor
$\tau_{turb}$	sub-grid scale Reynolds stress
$\nu_T$	turbulent viscosity
$\delta_{ij}$	Kronecker delta
$C_v$	Deardorff model constant
$k_{sgs}$	sub-grid scale kinetic energy
$\tau_w$	wall shear stress
$u^+$	non-dimensional stream-wise velocity
$y^+$	non-dimensional wall normal distance
$u_\tau$	friction velocity
$\kappa$	<i>von Kármán</i> constant
$s^+$	roughness length in viscous units
$\delta_y$	cell height adjacent to wall
$s$	dimensionless roughness
$T_s$	Vegetation surface temperature
$\dot{m}_{vap}$	evaporation rate
$\dot{Q}_{net}$	total energy on fuel surface
$\Delta h_{vap}$	latent heat of evaporation
$\dot{m}_{pyr}$	rate of pyrolysis
$\Delta h_{pyr}$	heat of pyrolysis/ heat of reaction

---

$a_{CO_2}, a_{H_2O}, a_{CO}, a_{N_2}$	stoichiometric coefficients
$\zeta(t)$	unmixed fraction
$\tau_{mix}$	mixing time scale
$u_0$	reference velocity
$z$	domain height
$z_0$	specified height of the domain
$p = 1/7$	empirically derived value of power in power-law
$N$	number of eddies
$\sigma$	size of eddies
$V_B$	cross-section of inlet
$Z_0$	aerodynamic roughness length
$\bar{u}_i$	resolved part of velocity
$\bar{p}$	resolved pressure
$F$	general forcing term
$F_{blend}$	blending forcing term
$\chi_{blend}$	blending factor
$\eta_{blend}$	blending parameter
$F_{penal}$	penalisation forcing term
$\chi_{penal}$	penalisation factor
$\eta_{penal}$	penalisation parameter
$penXmin, penXmax$	penalisation and blending regions along x
$penYmin, penYmax$	penalisation and blending regions along y
$penZmin, penZmax$	penalisation and blending regions along z
$mX, mY, mZ$	inlet/outlet along x,y,z, directions
$pena\_I, pena\_J, pena\_K$	no. of grid-points along x,y,z
$\dot{q}'''$	Heat release rate per unit volume
$\dot{m}_\alpha'''$	lumped species mass production rate
$\Delta H_{f,\alpha}$	heat of formation

*LIST OF TABLES*

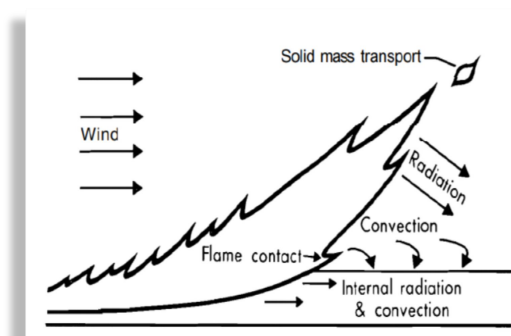
---

-	average
^	weighted average
CO <sub>2</sub>	carbon-di-oxide
H <sub>2</sub> O	water
N <sub>2</sub>	nitrogen
CO	carbon-monoxide

# Chapter 1

## Introduction

**W**IND is an eminent environmental factor that plays a major role in affecting wild-land fire behaviour. Various environmental structures including mountains, terrain features, valleys and grasslands produce complex local wind patterns that provides difficulties when predicting fire behaviour. As discussed in [Rothermel \(1972\)](#), the vertical heat flux is more significant for wind-driven fires as the flame tilts, resulting in direct contact with fuel loads, increasing the radiation and convective heat transfer to the fuel bed, and hence, making the fire spread faster and more severely. The wind tilted flame can be represented by the schematic given in Figure-1.1 following [Rothermel \(1972\)](#) . Therefore, an accurate and good prediction of wind pattern is required for accurate fire behaviour prediction.



Schematic of wind-driven fire

*Figure 1.1: Schematic representation of wind-driven fires ([Rothermel \(1972\)](#))*



## 1.1 Wildland fires

Wildland fires or wild fires are an intrinsic part of many ecosystems around the world. These fires can be classified as different types like forest-fires, bush-fires, grass-fires. Wildland fires have been a major component of the natural environment for at least 350 million years(Kemp (1981); Cope and Chaloner (1985)). Ronchi et al. (2017) gives an account of some well known wildland fires around the world. Severe wildfires that occurred in British Columbia, Canada; California, USA; Portugal and Italy caused more than 100 fatalities in July 2017.

Uncontrolled fire spread occurring on the boundary of residential areas, so called Wildland Urban Interfaces (WUI) Manzello et al. (2018), have the greatest impact on society. An increase in population results in an increase in the number of WUI which further increases the risk of severe impact of fires, in terms of loss of life and property, in these countries. A WUI can be considered as a zone of transition between any unoccupied land like forests, grasslands, and areas of human development including houses, and other structures. In some countries like the United States, WUI has increased from 1990 to 2010 several folds in terms of land area, as well as the number of new houses as given by Radeloff et al. (2018). Some of the most prominent wildland fires, including the Oakland fire, California in 1991 (Pagni (1993)), the Fort McMurray fire in Alberta, Canada in 2016 (Westhaver (2017)) had a damage cost worth more than a billion dollars in terms of structures, fatalities and ecology (Ronchi et al. (2017)).

Wildfires also adversely affect the underlying structures such as vegetation (forests or grasslands), habitat and living creatures, and poses far more complex problems. In Australia, wildfires generally occur from late Spring to early Autumn. Jolly et al. (2015) and Bedia et al. (2015) discussed that climate has a major impact on wildfires and further climatic changes have amplified the frequency of wildland fires, exponentially. Some of the effects include the disturbance in the water supplies as a result of erosion and contamination caused by the fires. Pyne (1991) describes that geological features and weather patterns of the Australian continent have a rich history with wildland fires. McAneney

[et al. \(2009\)](#) discussed that, from 1900 to 2003, around 5000 wildfire occurrences were recorded in Australia. The most significant among them is the infamous Black Saturday fire that occurred at Kilmore East in Victoria ([Whittaker et al. \(2009\)](#)) on February 7, 2009 and lasted till March 14, 2009 (Figure-1.2). The death toll rose to 172 and caused severe damage to Victoria, Australia both ecologically and economically. As stated by [Jolly et al. \(2015\)](#), the effect of climate change will increase the frequency of wildfires in the upcoming decades.

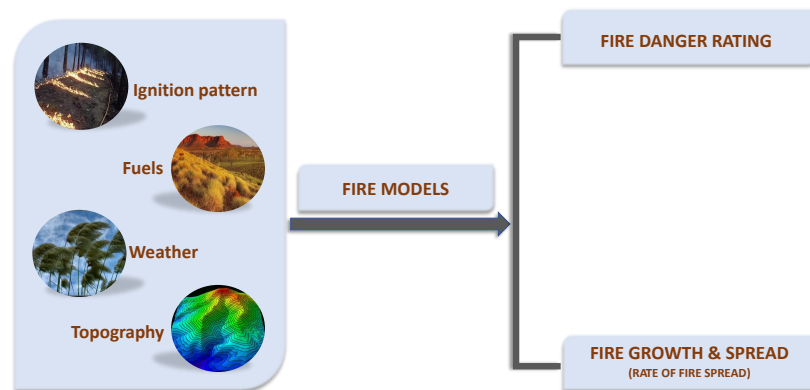


*Figure 1.2: Black Saturday Wildfires, Victoria (Source: [australianrotaryhealth.org.au](http://australianrotaryhealth.org.au))*

## 1.2 Fire behaviour models

[Perry \(1998\)](#) states that the research in the field of wildland fire can be considered as being two-fold: firstly, to quantify the fire danger and thereby develop fire danger rating systems; secondly, the development of a 'new generation' of fire spread models which help in accurate prediction of fire spread in wildlands. He further states that "fire spread through any kind of fuel bed such as wildland, shrubland, grassland incorporates a handful of complex physical and chemical processes." These situations have all been considered in several research studies. In addition, specific environmental variables significantly affects the spread of fire as discussed by [McArthur \(1967\)](#). One of the reasons may be the variation in vegetation type, which affects the fire propagation, such as sur-

face fires on grasslands and shrublands, crown fires in pine forests, and spotting fires in eucalyptus. Figure-1.3 depicts the basic ideas of a fire spread model. Data such as fuel load, topographical slope, wind speed, temperature, and humidity, are input into the model. The model outputs a fire danger index, used by emergency services organisations to implement warnings, and fire bans, to prepare for possible incidents, and to estimate a rate of spread to predict the location of a fire after a certain period of time.



*Figure 1.3: An overview of wildland fire behaviour and their relation to its fire modelling*

Rate of spread (RoS) is one of the most important parameters for fire management. RoS is the frontal speed of the propagating fire. Often practitioners assume a fire propagates at a quasi-steady rate. Yet, RoS depends only on environmental parameters and is independent of the state of the fire. Various fire models are needed to predict such behaviours to manage fire. Weber (1991) states that fire spread is the combination of three physical processes: the source of fire, the heat produced in the event and further fire spread by absorption of energy by unburnt fuels. An overview about various fire behaviour models is given in Section 1.2.1.

### 1.2.1 Wildland fire models

Several wildland fire models have been developed and used through about 60 years of scientific research (Cruz et al. (2015)) to predict the fire spread. Various models oper-

ates in various conditions and proper type of modelling needs to be chosen for accurate prediction of fire spread. These models can be broadly classified into: *empirical models* and *theoretical models*.

*Empirical Fire Models* : Empirical models are based on experiments and observation, not directly on underlying physical principles. In their studies, [Sullivan \(2009b\)](#) and [Perry \(1998\)](#) discuss types of empirical models. Some examples of simulators and empirical models used are in Table-1.1. A simulator is software that takes input weather and fuel data and uses a set of mathematical equations to provide a simulation of a fire across a landscape. Simulators such as BEHAVE, FARSITE, SPARK and PHOENIX for wild-fire prediction have been developed based on the empirical models and are used operationally. In Australia, PHOENIX is used operationally by the Country Fire Authority, Victoria and the New South Wales Rural Fire Service. PHOENIX uses a modified form of the CSIRO's grassland model and the McArthur MK5 forest fire models. SPARK is another operational tool developed by CSIRO. SPARK allows a user-defined RoS model and hence is more flexible than any other hard-coded operational models.

**Table 1.1:** Some examples of (a) Empirical Models and (b) Simulators

Empirical Models	
McArthur MK5 forest fire model	<a href="#">McArthur (1967)</a>
CSIRO grassland model	<a href="#">Cheney et al. (1998)</a> (a)
CSIRO forest model	<a href="#">Gould et al. (2007)</a>
Rothermal Model	<a href="#">Rothermel (1972)</a>
Simulators	
BEHAVE	<a href="#">Andrews (1986)</a>
PHOENIX	<a href="#">Tolhurst et al. (2008)</a>
SPARK	<a href="#">Miller et al. (2015)</a> (b)
FARSITE	<a href="#">Finney (1998)</a>

*Theoretical Fire Models* : (Mell et al. (2007)) includes coupled fire-atmosphere models and fire-fuel models. These models try to solve the equations of fluid dynamics, heat transfer and combustion. Wildland fire is the combination of various physical and chemical processes which include the energy released as heat is due to chemical reactions taking place while burning, and transport of that energy to the surrounding unburnt fuel and then reigniting Mell et al. (2007). Complete physics-based models include those that solve both fire-atmosphere and fire-fuel interactions simultaneously. As discussed by Mell et al. (2007), the theoretical models which were developed upto about 1989 (discussed thoroughly by Weber (1991)) did not include approaches to model fire-atmospheric interactions. In those models, the physical modelling of fire was focused on heat-transfer within the fuel and hence were mostly 'fuel driven' models. In subsequent years, with the advancement of these models, models for fire-atmospheric interaction have also been incorporated at various levels of complexity. Zhou and Pereira (2000), Morvan and Dupuy (2001), Morvan and Dupuy (2004), Dupuy and Morvan (2005) model both fire-atmospheric as well as fire-fuel interactions. These models were later extended by Morvan and Dupuy (2004) to a 2-D model (known as FireStar2D) which uses a renormalised group  $k - \epsilon$  (TRANS) turbulence model. They were able to replicate the Mediterranean shrub experiments performed by Fernandes (2001) using this model. The studies carried out by these models were on two-dimensional grids; hence, fire-atmospheric interactions were not captured completely. Three-dimensional simulations are needed to capture the realistic fire-atmospheric interactions properly. FIRETEC is a full three-dimensional fire model coupled to an atmospheric model HIGRAD (Reisner et al. (2000)). The governing model equations for mass, momentum, energy and chemical species in this model are based on ensemble averaging technique, which is similar to the Reynolds's averaging. One such model, FIRESTAR was initially developed as a one-dimensional model. It was later developed to become a three-dimensional grassfire model (Morvan et al. (2006)) known as FireStar3D. This model can now use both Transient Reynolds Averaged Navier Stokes (TRANS) and Large Eddy Simulation (LES) turbulence models (Morvan et al. (2018), Frangieh et al. (2018)). WFDS (Wildland-Urban Interface Fire Dynamics Simulator)(Mell et al. (2007)) extends the capabilities of FDS to outdoor fire spread and

smoke transport that take into consideration vegetative and structural fuels as well as the potential to deal with complicated terrain (Mell et al. (2009)). WFDS uses many empirical parameterisations similar to FIRESTAR, and is a full three-dimensional physics-based model which uses LES to capture turbulence. Several other physics-based models have been discussed in Sullivan (2009a). The discussions put forward by Mell et al. (2018) state that, though empirical models are used for operational purposes, physics-based models act as a powerful research tool.

### 1.3 Problems and Motivation

The fires spread rate is strongly dependent on wind speed, as discussed in previous sections. The velocity profile of wind varies over different types of environment structures, such as forests, open ground or flat grasslands. Several numerical or theoretical models have been developed for predicting the wind behaviour as discussed in Section-1.2.1. Following this discussion, it is evident that physics-based models act as a strong research tool to study fire behaviour. Looking to the future, most research will be conducted using physics-based models. Currently, these models are slow in performing simulations. The current work will improve the capability and reduce the computational time of physics-based fire models.

The current work uses FDS (Fire Dynamics Simulator), as a physics-based model for simulating the wind behaviour and establishing a required ABL (Atmospheric Boundary Layer) to carry out fire simulations. The computation domain used in FDS is divided into a number of grids or cells. The governing equations of fire dynamics are solved iteratively at each grid point. To obtain numerically converged fire-spread results, often very fine grid sizes (for example, for a simulation of a 100 m wide fire, a resolution of 250 mm is required) are used for fire simulations. This results in a huge computation expense. For example, a small domain of size 130 m X 40 m X 80 m with minimum terrain features having a uniform grid size of 1 m in all directions, requires approximately 600-800 s of simulation time to generate the required ABL for starting fire. This requires ( $\sim 15 - 20$  computational hours or  $\sim 60 - 80$  CPU hours, in a computer with 4 cores. For

fire simulations, a smaller grid size is needed to capture different fire behaviours, especially near the boundaries, which increases the computation time further, particularly with FDS version 6.3.0 onwards. It is evident from this that the computation time increases with an increase in the domain size, and complexity of the domain with respect to terrain structures, where finer grid cells are required to capture the effects. This is a major drawback of the physics-based modelling. For fire simulations in FDS, a mixing-controlled combustion model has been used to model the combustion of the vegetative burnable fuels. Until FDS 6.2.0, an upper-bound of local heat release rate was imposed, based on [Orloff and De Ris \(1982\)](#). Since FDS 6.3.0, this limiter was removed. Finer grid resolution is required to obtain grid converged results. While this may provide better results for small-scale fires, large-scale fires require this bound to avoid simulation errors like numerical instabilities. This also poses a problem in carrying out fire simulations in terms of computational time, as finer grids require more simulation time to model fire. The current research aims to address this drawback and tries to overcome part of this and hence, improve the speed of physics-based simulations.

## 1.4 Addressing an Omission in Previous Research

Several methods of wind-generation are already available in FDS, which have been termed '*traditional methods*' in this document. The major problem with these methods is that most of them require a considerable amount of simulation time to reach a required state so that fire can be ignited to carry out fire simulations, as discussed before. This results in an increase in computation time. This is one of the major challenges being addressed which will bring a significant benefit. The current research tries to reduce simulation time by reducing the time to reach a required wind profile to start fire.

A novel method, called the *PenaBlending* method, has been introduced in FDS to reduce the computational time required to initialise simulations, and to allow complicated time varying driving wind fields to be applied. Currently, FDS does not allow any external wind data to be used for modelling fires. The *Pena-Blending method* will allow the users to use any external data, generated by analytical methods, terrain-perturbed mod-



els or experimental data, to be used as inlet conditions to model fire. However, there is still time needed for generating external wind data for FDS using these methods. If a full Large Eddy Simulation (LES) model is used, it will take just as much time as FDS does to generate wind data. Therefore, the *Pena-Blending Method* substantially reduces time if reduced-models are used for generating terrain-perturbed wind data. These models use coarser resolutions and can generate data in seconds. Terrain-perturbed wind refers to wind data that is modified according to the underlying structures over which it flows. Such data can be obtained from reduced wind models like mass-conserving model of Windninja (Forthofer et al. (2014b)). Wind data can also be generated using analytical methods with MATLAB or any other programming languages. A single precursor simulation can also be used for multiple simulation cases to reduce the time in generating external wind datasets for FDS. The *Pena-Blending Method* allows the use of temporally and spatially varying wind data. This will further allow the study of fire behaviour in gusty wind conditions.

To carry out fire simulations, a combustion model is required which can resolve fire faster with a justified grid resolution and does not require very fine grids. This can be achieved by imposing an upper-bound of local heat release rate as discussed in Section-(1.3). The current version used in this study, FDS 6.6.0, does not have an upper-bound of local heat release rate. Therefore, the combustion model of FDS 6.2.0, with reaction-rate limiters, has been re-implemented in FDS 6.6.0 and has been used to carry out all the fire simulations. This further will contribute to reducing the computational time for fire simulations. The rest of the part of this research report is divided thus: Chapter 2 gives an overview of physics-based modelling and a brief account of FDS, the model used in the current study, followed by a discussion about the prevailing inlet methods in FDS. Chapter 3 gives detailed information about the new sub-routines that have been implemented in FDS 6.6.0, as a part of this research, along with some verification cases. Wind and grassfire modelling results using *traditional* wind generating method are presented in Chapter 4. Lastly, Chapter 5 concludes the current research and states the future directions. The source codes of the FDS files where new codes have been implemented is given in Appendix(A). The full set of edited FDS 6.6.0 source code can be found in (<https://drive.google.com>).



[com/open?id=18uQEmprdpmNDgBGIDVswu9ER\\_mHpN9Ho](https://arxiv.org/abs/1808.09999)). A list of publications and a copy of the publication is in Appendix(B). A preliminary study has been carried out by conducting fire simulations with different atmospheric stabilities using Monin-Obukhov similarity theory introduced in FDS 6.6.0 and can be found in Appendix(C). The results need further investigation. A small demonstration video of the working of the *PenaBlending* method has been made and can be found at (<https://youtu.be/g5s4QZQ1DmU>).

## Chapter 2

# Physics-based modelling

Physics-based wildfire models are developed based on the physics of fire, as discussed in Section-(1.2.1). The current study uses Fire Dynamics Simulator (FDS) [McGrattan et al. \(2017d\)](#). FDS was developed and first released by the National Institute of Standards and Technology (NIST) in collaboration with VTT Technical Research Centre, Finland in the year 2000. Since then several versions have been released with some improvements in each version. Currently, FDS version 6.6.0 (released in 2017) has been used to carry out the studies. FDS is a tool to predict large-scale fire effects including plume characteristics, combustion product dispersion as well as heat effects to adjacent objects, as discussed by [Ryder et al. \(2004\)](#). FDS has the ability to model pyrolysis, most importantly coupled pyrolysis, and turbulent combustion. Moreover, FDS is an open source code written in FORTRAN, which is free and easy to download and the code can be easily modified by any colleague to implement several other features and improve the performance. This is an additional benefit of using FDS as a research tool to carry out the current study.

### 2.1 The model's description

FDS is a computational fluid dynamics (CFD) model for fire driven fluid flows. It solves the Navier-Stokes equation for low speed thermally driven flows, appropriate for low mach number flows of smoke and hot gases which results from fire ([McGrattan et al. \(2017d\)](#)). FDS is a finite difference approximation to the equations of fluid motion. The computational domain that is considered is discretised into small grids (commonly referred to as cells or control volumes). The scalar quantities such as temperature, pressure

and density are calculated at the centre of each cell by solving respective equations. The cells are often not sufficiently small enough to capture the small turbulent eddies. The discretised equations of mass, momentum and energy yield a large system of algebraic equations. These equations are then solved numerically to acquire the estimated values at the centre of each cell. Hence, turbulence models are used to take into consideration these small eddies in the flow field. FDS uses this LES methodology where the small and unresolved eddies are modelled, using a turbulent viscosity. LES turbulence models have advantages over Reynolds's Averaged Navier-Stokes (RANS) to model the effect of turbulence [Rodi \(1997\)](#). While stability issues exist in flows with high temperature and large pressure gradients, FDS with the appropriate resolution has been found to reliably simulate grass fires ([Mell et al. \(2007\)](#), [Moinuddin et al. \(2018\)](#)) and tree fires ([Mell et al. \(2009\)](#)) and is reliable for the present study.

### 2.1.1 Mass Continuity

FDS conserves the mass of the fluid by solving the *continuity equation* given by Equation(2.1).

$$\frac{\partial \rho}{\partial t} + \nabla \cdot \rho U = 0, \quad (2.1)$$

where  $\nabla \cdot \rho U$  defines the mass convection and  $U$  represents the instantaneous velocities in x, y and z directions.

### 2.1.2 Conservation of Momentum

FDS solves Equation(2.2) ([McGrattan et al. \(2017a\)](#)) for conservation of momentum for the fluid flows.

$$\frac{\partial \rho \bar{U}}{\partial t} + \nabla \cdot (\rho \bar{U} \bar{U}) = -\nabla p + \nabla \cdot \tau_{ij} + \rho g + \nabla \cdot \tau_{turb} + \dot{m}''' u_{b,i}, \quad (2.2)$$

where pressure  $p$ , gravity  $g$ , viscous stress tensor  $\tau_{ij}$  is acting on the fluid within a control volume.  $\bar{U}$  represents the filtered velocity and  $\dot{m}''' u_{b,i}$  represents the effect of buoyancy. The

stress tensor can be defined as:

$$\tau_{ij} = \mu(2\overline{S}_{ij} - \frac{2}{3}(\nabla \cdot \bar{U})\delta_{ij}), \quad (2.3)$$

where  $\mu$  represents the molecular viscosity and  $\overline{S}_{ij}$  represents the strain-rate tensor given by :

$$\overline{S}_{ij} = \frac{1}{2}(\frac{\partial \bar{u}_i}{\partial x_j} + \frac{\partial \bar{u}_j}{\partial x_i}) \quad (2.4)$$

The  $\tau_{turb}$  represents the filtered turbulence known as sub-grid scale Reynolds stress.

### 2.1.3 Turbulence Model

The Sub-Grid Scale(SGS) model accounts for the dissipative processes (such as thermal dispersion, viscosity, and material diffusivity) which takes place in smaller scales apart from those resolved in the numerical grids Pope (2001). The SGS model equation can be given by Equation(2.5).

$$\tau_{turb} = \rho\nu_T(\overline{2S}_{ij} - \frac{2}{3}(\nabla \cdot \bar{U})\delta_{ij}) \quad (2.5)$$

where  $\bar{U}$  represents the velocity field and  $\delta_{ij}$  is known as the Kronecker delta and  $\nu_T$  is the eddy viscosity. The default eddy viscosity model in FDS, and the one adopted throughout this thesis is the Deardorff model (Deardorff (1980)). The model can be given by Equation(2.6).

$$\nu_T = \rho C_v \Delta \sqrt{k_{sgs}} \quad (2.6)$$

where  $\rho$  represents the fluid density,  $C_v$  represents the Deardorff model constant is set to the value 0.1 from Pope (2001),  $\Delta$  represents the LES length scale and  $k_{sgs}$  represents the sub-grid scale kinetic energy and can be given by Equation(2.7).

$$k_{sgs} = \frac{1}{2}((\bar{u} - \hat{u})^2 + (\bar{v} - \hat{v})^2 + (\bar{w} - \hat{w})^2) \quad (2.7)$$

where  $\bar{u}$  represents the average value of  $u$  at the centre of the grid cell and  $\hat{u}$  represents the weighted average of  $u$  over the adjacent cells. The definition is similar for  $v$  and  $w$ .

#### 2.1.4 Wall Model

A wall model is used to estimate the instantaneous wall shear stress,  $\tau_w$ , which is applied as the boundary condition for solid surfaces to the LES equations (Pope (2001)). This model is implemented using the wall function. For a realistic atmospheric flow, it is computationally infeasible to resolve the viscous sub-layer near the wall region, where the solution variable changes sharply. The wall model helps to overcome this problem and helps to obtain the appropriate flow variables near the boundary. The dimensionless analysis of a fluid near the wall leads to the logarithmic law of the wall (Von Kármán (1930)). Several other models for the near wall flow also exist. As discussed in studies by Barenblatt (1993), Barenblatt and Prostokishin (1993), Barenblatt and Goldenfeld (1995) and Barenblatt and Chorin (1997), there exist *power law* models as well.

FDS applies different laws-of-the-wall for different type of surfaces. For smooth surfaces, it follows Werner and Wengle (1993) model using Equations(2.8).

$$\begin{aligned} u^+ &= y^+ & \text{for } y^+ < 11.81, & \text{ (Viscous sub - layer region)} \\ u^+ &= \frac{1}{\kappa} \ln y^+ + B & \text{for } y^+ \geq 11.81, & \text{ (Log - law region inner layer)} \end{aligned} \quad (2.8)$$

where  $u^+$  represents the non-dimensional stream-wise velocity, ( $u^+ = u/u_\tau$ ),  $u_\tau$  is the friction velocity,  $y^+$  represents the non-dimensional wall normal distance,  $\kappa = 0.41$  is the *von Kármán* constant, and  $B=5.2$ .

For rough surfaces, FDS uses the log-law as presented by Pope (2001) as given by Equation(2.9).

$$u^+ = \frac{1}{k} \ln\left(\frac{y}{s}\right) + \bar{B}(s^+) \quad (2.9)$$

where  $s^+ = s/\partial_y$  represents roughness length in viscous units,  $\partial_y$  represents the cell height adjacent to the wall,  $s$  represents dimensional roughness,  $\kappa = 0.41$  is the *von*

$Kármán$  constant and  $y$  is the distance to the wall.  $\bar{B}$  is defined as the following piece-wise function:

$$\bar{B} = \begin{cases} B + (\frac{1}{\kappa})\ln(s^+) , & \text{for } s^+ < 5.83 \\ \bar{B}_{max} , & \text{for } 5.83 \leq s^+ < 30.0 \\ B_2 , & \text{for } s^+ \geq 30.0 \end{cases} \quad (2.10)$$

where  $\bar{B}_{max} = 9.5$  and  $B_2 = 8.5$  for fully rough surfaces. From the work of [Werner and Wengle \(1993\)](#), it is shown by matching the log regions and the viscous region, that the log layer starts at  $y^+ = 11.81$ .

### 2.1.5 Pyrolysis

FDS has many pyrolysis models available, among which it incorporates two vegetation sub-models for thermal degradation which are used in FDS : The 'linear' model and the 'Arrhenius' model, which are based on empirical studies as given in [McGrattan et al. \(2017a\)](#). The current research uses the 'linear' model to predict ignition. The linear model involves a two-stage endothermic thermal decomposition which involves evaporation of water and then solid fuel pyrolysis. When the temperature  $T_s = 373K$ , the water evaporates and it follows Equation(2.11).

$$\dot{m}_{vap} = \frac{\dot{Q}_{net}}{\Delta h_{vap}} , \quad (2.11)$$

where  $T_s$  is the vegetation surface temperature,  $\dot{m}_{vap}$  represents the evaporation rate,  $\dot{Q}_{net}$  is the total energy including convection and radiation on the fuel's surface and  $\Delta h_{vap}$  represents the latent heat of evaporation. Following [Morvan and Dupuy \(2004\)](#), FDS uses the temperature-dependent mass loss rate expression given by Equation(2.12) for modelling solid fuel degradation, by considering that pyrolysis begins at 400 K.

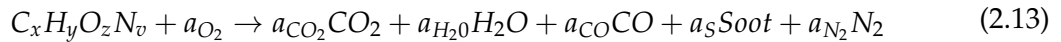
$$\text{If } 400 \text{ K} \leq T_s \leq 500 \text{ K, } \dot{m}_{pyr} = \frac{\dot{Q}_{net}}{\Delta h_{pyr}} \chi \frac{T_s - 400}{500 - 400} , \quad (2.12)$$

where  $\dot{m}_{pyr}$  represents the rate of pyrolysis and  $\Delta h_{pyr}$  represents the heat of pyrolysis or heat of reaction.

In the current study, the fuel is modelled using the boundary-fuel representation. For simplicity, the solid fuel is considered as a series of layers which are consumed starting from the top to the bottom layer. [Moinuddin et al. \(2018\)](#) discusses this stating that, in a linear model, within the range of 400 – 500 K, ignition and sustained burning occurs at low air temperature; hence, coarser gas-phase grid resolutions are sufficient. The user needs to supply a bound on the maximum mass loss rate per unit volume ( $kg/s/m^3$ ) to avoid any kind of numerical instabilities during calculations. Char oxidation is not considered as it occurs at a much higher temperature than that obtained in the simulations performed.

### 2.1.6 Combustion

FDS models the combustion using mixing-controlled chemistry as discussed in [McGrattan et al. \(2017a\)](#). The mixing-controlled model presumes that the reaction between fuel and the oxygen is very quick and the rate of reaction is controlled only by mixing ([McGrattan et al. \(2017d\)](#)). This model usually involves only one gaseous fuel which solves the transport equations for only the lumped species, which means the other products (such as  $O_2$ ,  $CO_2$ ,  $H_2O$ ,  $N_2$ ,  $CO$  and soot) and fuel. The lumped species air acts as a default background for all these reactions. As discussed in [McGrattan et al. \(2017d\)](#), in this method, single fuel species which are mainly composed of C, O, H and N react with oxygen in one mixing-controlled step to produce  $H_2O$ ,  $CO$ ,  $CO_2$  and soot. The simple chemical reaction can take the form of Equation(2.13).



where  $a_{CO_2}$ ,  $a_{H_2O}$ ,  $a_{CO}$ ,  $a_S$ ,  $a_{N_2}$  are the stoichiometric coefficients. FDS regulates the rate at which the fuel and oxygen mixes within a given mesh cell at a particular time-step. Each computational grid cell can be considered as a cluster of reactors where a mixed-composition reaction can only take place.  $\zeta(t)$  represents the *unmixed fraction*, which is

the fraction of mass within the existing cell/grid, is governed by Equation(2.14).

$$\frac{d\zeta}{dt} = -\frac{\zeta}{\tau_{mix}} \quad (2.14)$$

where  $\tau_{mix}$  denotes the mixing time scale.  $\zeta$  holds a value of 1 by default if a cell is initially unmixed and the combustion is called non-premixed. If the cell is initially mixed,  $\zeta$  holds a value of 0. This type of combustion is called premixed. FDS determines the amount of combustion products formed ( $CO_2, CO, N_2, H_2O$  and soot) in the process, and is determined from the chemical formation of the fuel used. A detailed account of the model can be found in [McGrattan et al. \(2017a\)](#) and [McGrattan et al. \(2017d\)](#).

The models that are relevant to the current work have been discussed here. Apart from these, FDS solves other equations including heat transfer equations for conduction, convection and radiation, species equations to simulate smoke transport, ideal gas equations for temperature, and Poisson's equation for pressure. The detailed set of equations and the models used by FDS can be found in [McGrattan et al. \(2017a\)](#).

Outdoor fire simulations require wind fields. As discussed before, there are *traditional* and novel means of generating the wind fields. These methods are discussed in the next section.

## 2.2 Traditional methods of wind field generation

Wind needs to be modelled properly to perform a fire simulation correctly. There are several methods of wind generation and obtaining a stable ABL available in FDS. In the current research, these already existing methods are termed as '*traditional methods*'. The following sections describe these existing methods of wind field generation.

### 2.2.1 Wall of Wind method

[McGrattan et al. \(2017d\)](#) discusses the 'wall of wind' method as specifying any inlet condition. FDS uses a power-law wind profile ([Touma \(1977\)](#)), by default, at the inlet boundary of the computational domain. The simulated atmosphere will transition to turbulence due to random perturbations included in the initial velocity fields. This profile



acts as a wall of wind and sweeps through the domain; hence, generating a wind profile across the domain and eventually establishing a statistically steady ABL. A change in the roughness of the ground acts as a trip to enhance turbulence. The change in roughness leads to a fully turbulent boundary layer, which will eventually (roughly) stabilise over the fire ground. In FDS, the wall of wind can be specified as follows:

$$u = u_0 \left( \frac{z}{z_0} \right)^p, \quad (2.15)$$

where  $u_0$  is the reference velocity, called *VEL* in FDS, given at height  $z_0$  called *Z0*,  $z$  is the domain height,  $p$  is empirically derived and considered to be  $1/7$  or  $0.143$  for neutral atmospheric conditions, called *PLE*. In FDS, these values are specified in the *SURF* line in the following manner:

```
&SURF ID='WIND', PROFILE='ATMOSPHERIC', Z0=10, PLE=0.143, VEL=-4.7/ power law
```

Here *VEL* represents an inlet velocity of  $4.7$  m/s, at a reference height of  $10$  m and a power of  $1/7$ .

## 2.2.2 Synthetic Eddy Method

The development of turbulent structures in a flow is one of the most important aspect of LES of bounded fluid flows. The infusion of random numbers or roughness trip needs a significant distance to be travelled by the wind before becoming a fully turbulent flow. The SEM in FDS uses the *SEM* developed by [Jarrin et al. \(2006\)](#), which reduced the distance to be travelled before becoming fully turbulent. This method produces realistic inflow conditions, based on the view of turbulence as a superposition of coherent structures. In this method, eddies are injected into the inlet at random positions and advect with the inlet velocity inflow which subsequently gets rescaled to match the desired turbulent characteristics ([Singha Roy et al. \(2018\)](#)). A log-law [Pope \(2001\)](#) inlet profile can be used while using SEM in FDS. The user needs to specify the velocity scales (*VEL\_RMS*) of each coherent structure that adds up to the velocity field, length of eddies (*L\_EDDY*), precisely the diameter of the eddies, and the number of eddies (*N\_EDDY*) in the input file in the *VENT* line in the following way:

```

|| &VENT XB=0,0,0,300,0,100, SURF_ID = 'INLET', N_EDDY=200, L_EDDY=10, VEL_RMS
|| =0.185/ SEM

```

Typically, the velocity and the length scales of the eddies should be chosen in a way so that some turbulent statistics, usually Reynolds stresses, are reproduced. Following the work of [Jarrin et al. \(2006\)](#), the total number of eddies can be calculated using equation(2.16).

$$N = \max\left(\frac{V_B}{\sigma^3}\right), \quad (2.16)$$

where  $\sigma$  (represented by *L\_EDDY* in an FDS input file) represents the size of eddies, which is  $\sim 3$  times the size of grids,  $V_B$  represents the box-volume or cross-section of the inlet where the eddies are embedded. The number of eddies ( $N$  - represented by *N\_EDDY* in FDS) as given in the input file, should be large enough to ensure that the eddies cover a cross section of the inlet ([Pavlidis et al. \(2010\)](#)). The wind develops over time and space, and finally reaches a fully developed flow condition albeit over a shorter distance.

### 2.2.3 Mean-forcing method

[McGrattan et al. \(2017d\)](#) discusses that FDS uses a simple data assimilation technique ([Kalnay \(2003\)](#)) known as '*nudging*'. A mean forcing term is added to the momentum equation to push (or '*nudge*') the wind profile so that the mean velocity approaches the specified mean value. The wind speed can be specified using *SPEED* or specifying any of its components *U0*, *V0* or *W0*. The mean wind flow in the domain will be driven towards this specified velocity. This method uses the log law to calculate the wind profile varying with height. In this method, no inlet profile is need to be specified as it is calculated automatically using the underlying log-law equations. The nudging wind speed needs to be defined, which can be specified in the *WIND* line as follows:

```

|| &WIND SPEED=6.0, DIRECTION=225., Z_REF=10, Z_0=0.03/

```

Here, the *Z\_REF* refers to the reference where the inlet velocity of 6 m/s is given at 225° and *Z\_0* is the aerodynamic roughness length which is different from *Z0* as mentioned in Section-2.2.1. The direction of wind is measured similarly to the normal meteorological

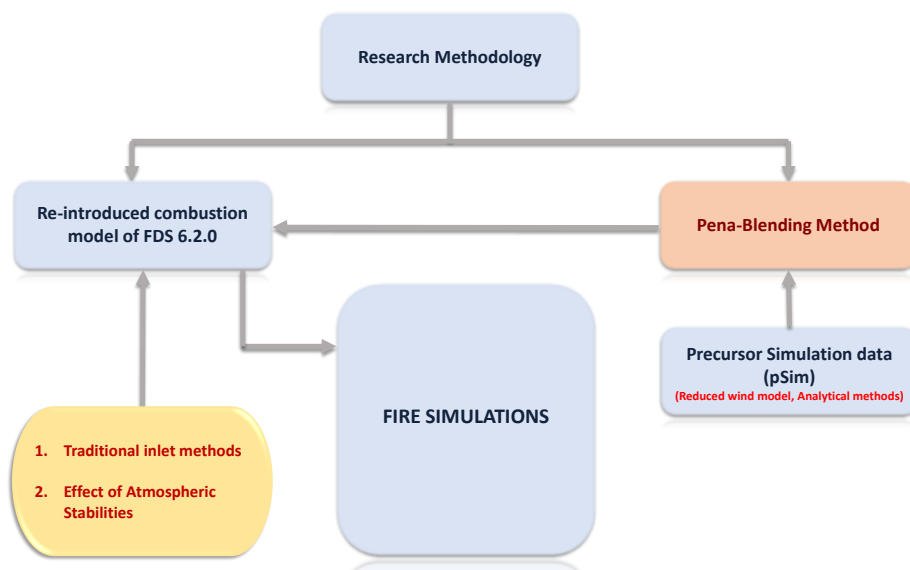
convention where  $0^\circ$  represents a northerly wind blowing from north to south along the direction of negative y-axis.

In order to specify a terrain modified temporally and spatially varying inlet condition, and reduce the simulation time, some modification of the underlying source code of FDS has been made. The *Pena-Blending Method* has been implemented to enhance computation speed and hence reduce the time for large scale simulations. These enhancements are explained in detail in Chapter(3). Local Heat Release Rate (HRR) limiter is also re-introduced into the code to obtain a grid-converged solution with coarser grids which is also discussed in Chapter(3).

# Chapter 3

## Methodology and Code development

The current research predominantly aims at reducing the computational cost of a physics-based model, and extending its simulation capacity to time varying inlet wind fields over complicated terrain. Although only flat terrain conditions are simulated in this study, the groundwork is being laid to account for various terrain conditions.



*Figure 3.1: Schematic representation of research methodology*

The current research deals with the role of wind in fire spread. Hence, this work will comprise studies related to wind development in a simulation domain to obtain a statistically-steady Atmospheric Boundary Layer (ABL). This is extended by studying

the effect of wind in fire-spread. The main focus is to check the time needed to do these simulations using FDS.

The recently developed *PenaBlending* method allows the ABL to develop quickly taking into consideration velocity, temperature and terrain, and; hence, fire simulation can be started quickly, resulting in a decrease in the overall computation time for carrying out fire simulations. This method allows the use of time-varying wind and temperature data obtained from any precursor simulations or reduced models, for carrying out fire simulations in FDS. This method has been discussed in details in Section-3.1.1.

Some modifications have been made in the existing combustion model of FDS 6.6.0, in order to carry out large scale fire simulations with reduced computational costs. The combustion model of FDS 6.2.0 has been re-introduced into FDS 6.6.0 and the current source-code used for running the simulations has an option to choose the combustion model of 6.2.0 or 6.6.0. A detailed analysis of what has been done related to this is given in Section-3.2.

A series of *wind* and *fire* simulations have been carried out with the existing *traditional* inlet and initial wind conditions as well as with the *PenaBlending* method. Large and small domain sizes, namely large and small have been considered in this study. The computation time for all these methods has also been compared. These methods have been discussed in detailed in Section-2.2 and the corresponding results are discussed in Chapter 4.

The complete methodology followed is depicted in Figure-3.1. The inline source-codes included in the following sections are the new additions to the existing code as a result of the current research.

### **3.1 Assessment of Inlet and Initial Condition for fire simulations**

Physics-based wild fire simulations are driven by inlet boundary conditions and initial flow conditions (eg., initial wind profile is a mean log-law profile) which model the atmospheric boundary layer. A faithful representation of the atmospheric boundary layer is required in order to confidently reproduce the rate-of-spread and intensity of the fire, the heat transfer to unburnt vegetation, and the transport of smoke and combustion prod-

ucts away from the fire ground. The inlet and initial conditions prescribed for the simulation preferably lead to a realistic flow over the fire ground, which does not non-physically develop in space or time. For example, Mell et al. (2007) uses a 1/7-power-law model at the inlet of their simulations. While the power-law profile is a model of a turbulent flow, the inlet condition does not include time varying turbulent fluctuations. Due to the lack of initial perturbations in the simulation, a fully turbulent flow profile will develop in time and space when an imposed perturbation or an obstacle is introduced, as the simulation progresses. Spatial and temporal development of the flow is undesirable for two reasons. *Firstly*, simulating the developing flow requires a great deal of computational effort to simulate the flow downstream of the fire ground and, as well as the additional time required for the simulated flow over the fire ground, to reach a statistically steady state. *Secondly*, a flow which develops, albeit slowly, over the fire ground can cause difficulties of interpretation. For example, if the aim of the simulation is to study fires in the wind field developing over a flat terrain with grass canopy, the influence of a developing inlet and initial boundary layer flow is undesirable and needs to be minimised. There are two main threads of physics-based wild fire simulation: simulations that seek to replicate experimental and field observations as validation of the physics-based approach (Mell et al. (2007), Moinuddin et al. (2018)); and simulations performed to gain additional insight into observed phenomena (Morvan et al. (2009), Sutherland et al. (2017)). Currently, discrepancies between the experimental observations are attributed to gusts and subtle changes in the wind direction which are not often considered in physics-based modelling. Large-scale LES simulations of weather systems are possible and may be downscaled by interpolation to give inlet and initial conditions. Similarly, log-law models, with artificial turbulence, or mass-conserving reduced models may be used to generate initial and boundary wind fields.

There are several ways of generating the inlet and initial flow conditions. The gold standard is a precursor simulation, where the flow is simulated over the same or similar domain, which is used as the inlet condition for the fire simulation. Precursors are restricted to fully-developed turbulent flow. For fire simulations, the atmospheric boundary layer (ABL) is typically assumed to be developed, in most of the cases. A relatively mod-

ern technique which minimises the flow development region is the *synthetic eddy method* (SEM) (Jarrin et al. (2006)). However, the SEM method still requires computational time and additional (albeit small) domain length to obtain a fully developed turbulent flow. The initial mean flows for the fire simulations can be generated with relative ease. For example, we can use the log-law model (analytical approach), a mass-conserving perturbation model or a large scale numerical weather prediction model as an initial mean flow condition. Each of these methods has an associated computational cost, with the analytic mean model being the cheapest, the perturbation method has intermediate cost, and a large scale simulation is extremely costly. However, as discussed before, this cost needs to be kept minimised by, for example, using the less costly methods (analytical methods or reduced models) or re-using a precursor simulation multiple times). The challenge is to implement these boundary conditions in a physics-based fire model. The *Pena-Blending Method* helps in implementing such boundary conditions in FDS, following Vonlanthen et al. (2016). This method inserts an artificial forcing term in the Navier-Stokes equation to force the velocities to the desired values.

### 3.1.1 Pena-Blending method

Unlike the *traditional* wind generating models, the *PenaBlending* method sets both inlet and outlet conditions; hence, it is designated as initial conditions. It sets the initial conditions of the fire simulations to the initial conditions prescribed by the external model or simulation. This can be achieved by implementing a one-way coupling algorithm in FDS. Assuming there exists external data for  $u, v, w$  which varies in space and time. This data is referred to as a '*precursor simulation*' and is abbreviated to *pSim*. This *pSim* can be a specified analytic profile (for example, generated from Matlab), generated from a known gust spectrum, come from some other reduced models of wind fields, or some experimental wind data. The *pSim* can be conducted over a larger domain with coarser resolution in time and space than the fire-spread simulation (FS) domain, if required. The sole idea is to develop a one-way coupling technique so that the *pSim* data can be used to drive a (FS) by using the *pSim* data as an inlet boundary condition for the fire spread. The *pSim* is enforced as an initial condition which is known as the '*penalisation region*' and then a specific transition region, known as the '*blending region*' is applied to smoothly transi-

tion the simulated flow in the fire domain to the flow enforced on the boundary. This is based on the blending method as proposed by [Davies \(1976\)](#). In the current study, the *Pena-Blending Method* has been implemented along x-direction. The generalisation of this method along y and z is also possible.

The *pSim* may have a coarser grid than the FS domain. In such a case, these are required to be interpolated in order to achieve the same grid resolution as the FS domain. The finer scale eddies generated within the fire simulation domain may not match with the eddies that are applied as a forcing in the *penalisation* region. This may give rise to inconsistency in the turbulent structures at the inlet and outlet coupling regions. The blending region allows a smooth transition of the eddies at the vicinity of the coupled boundaries. At the inlet of the FS domain, eddies from the *pSim* are enforced by coupling. They travel into the FS domain by advection and give rise to new turbulent structures as the simulation progresses. These new structures are transported to the nested outlet. If there is no blending region used, then there are inconsistent eddies at the coupled boundaries. Following [Vonlanthen et al. \(2016\)](#), this is depicted in Figure-3.2. Since these two regions are required to implement this method, it has been named as the *Pena-Blending method*. In the current research, a preprocessing step is conducted to process the *pSim* data into a number of files equivalent to the number of meshes, each file containing the velocity data for the penalisation and blending regions on the same grid as the FS domain. In case of a time-stepping simulation, the intermediate data at arbitrary times between the subsequent timesteps of *pSim* are obtained. These data may have a coarser grid than that required for the FDS domain. These data are then converted to the required grid resolution, using a linear interpolation method, so that it can be fed to FDS at required time-steps. The penalisation and blending regions are imposed using a straightforward forcing term in the Navier-Stokes equation.



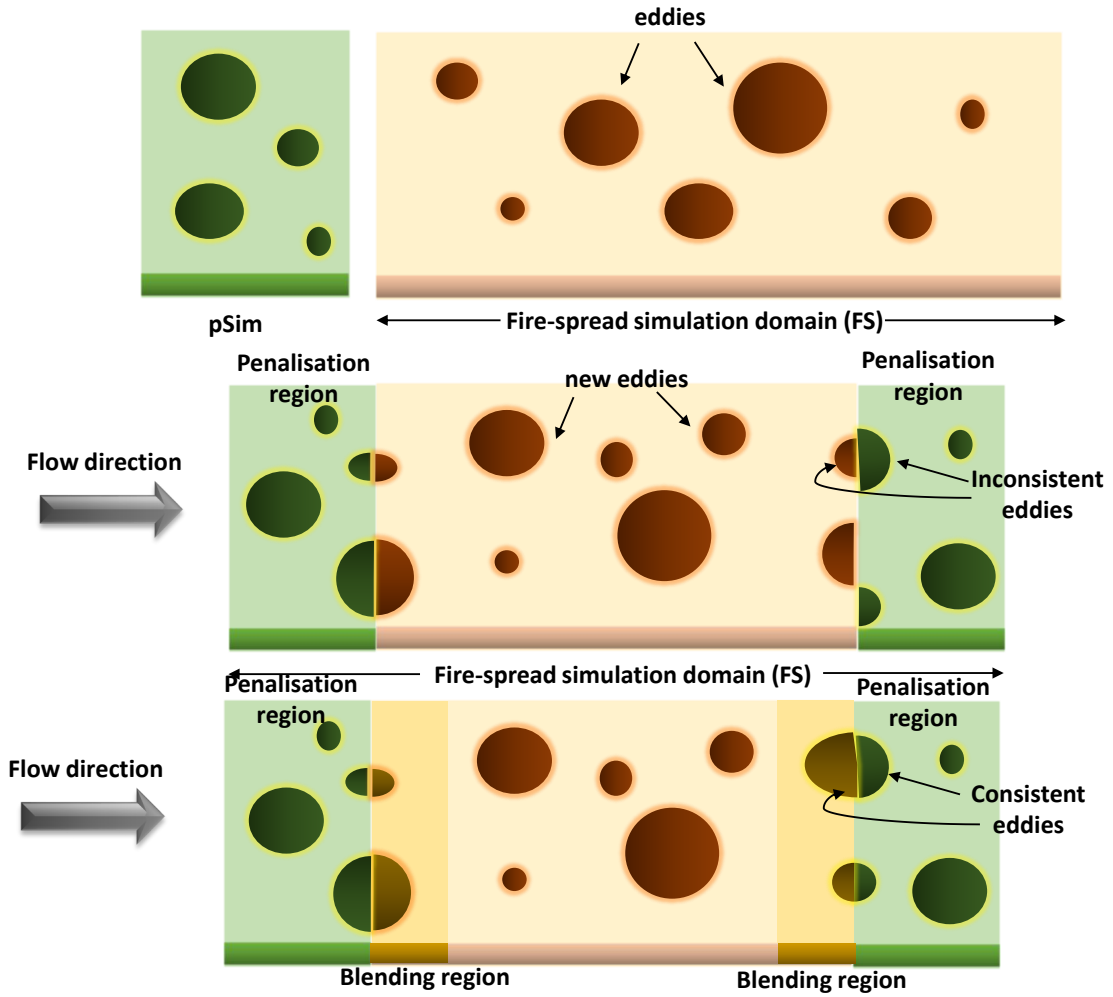


Figure 3.2: Schematic representation of turbulent eddies in FS domain using the PenaBlending method

### 3.1.1.1 Model Equations

The Pena-Blending method follows the one-way coupling method as given by [Vonlanthen et al. \(2016\)](#), to blend the turbulent flow created within the simulation domain to the turbulent conditions enforced at the boundaries. The heart of this scheme comprises a forcing term added to the Navier-Stokes equation. For convenience, we omit the combustion terms because they are not relevant here.

$$\frac{\partial \bar{u}_i}{\partial t} + \bar{u}_j \left( \frac{\partial \bar{u}_i}{\partial y_j} - \frac{\partial \bar{u}_j}{\partial x_i} \right) = \frac{1}{\rho} \frac{\partial \bar{p}}{\partial x_j} + \frac{\partial \tau_{ij}}{\partial x_j} + F, \quad (3.1)$$

$$\frac{\partial u_i}{\partial x_i} = 0, \quad (3.2)$$

where  $\bar{u}_i$  is the resolved part of velocities,  $i, j \in (x, y, z)$  are the three spatial coordinates,  $\rho$  is the fluid density,  $\bar{p}$  is the resolved pressure,  $\tau_{ij}$  is the deviatoric part of the stress tensor, given by :

$$\tau_{ij} = \bar{u}_i \bar{u}_j - \bar{u}_i \bar{u}_j \quad (3.3)$$

$F$  represents the general forcing term used to represent physical boundaries, computation boundary regions and other forces such as drag of canopy and can be represented as:

$$F = F_{blend} + F_{penal} + F_{damp} + F_{drag} + \dots, \quad (3.4)$$

The *penalisation region* represents an immersed boundary method which uses the forcing term of the form of Equation(3.5)

$$F_{penal} = \frac{\chi_{penal}}{\eta_{penal}} (\bar{u}_i^{pSim} - \bar{u}_i)$$

$$where, \quad \chi_{penal}(x, y, z) = \begin{cases} 1 & X_{min} \leq x \leq X_{max}, \\ & Y_{min} \leq y \leq Y_{max}, \\ & Z_{min} \leq z \leq Z_{max}, \\ 0 & otherwise, \end{cases} \quad (3.5)$$

The term  $\chi_{penal}$  represents that the 'penalisation factor' is applied uniformly in space. The *penalisation parameter* ( $\eta_{penal}$ ) is prescribed arbitrarily rather than by a physical scale. This value should be less than 1 ( $\eta_{penal} \ll 1$ ), so that the desired velocities can be obtained. A 'trial-and-error' approach has been followed to determine this value in this current work. It should be noted that the value of  $\eta_{penal}$  must be chosen sufficiently large so that numerical instability is avoided. This *penalisation parameter* can have different values for different velocities. The full implementation along with this region has been included in

*velo.f90*<sup>4</sup>. The part of the source code showing this region can be given in the following lines of code:

```

!penalisation region start
if (pendat(11,13).eq.1) then !checking if the penalisation factor=1

!checking if inside penalisation boundaries
if ((Z(K).le.pendat(11,9)).and.(Z(K).ge.pendat(11,8))) then
if ((Y(J).le.pendat(11,7)).and.(Y(J).ge.pendat(11,6))) then
if ((X(I).le.pendat(11,5)).and.(X(I).ge.pendat(11,4))) then

!assigning the counter to traverse the array(pendat) where the penalisation
    velocities are stored to do further velocity calculations
cntr=cntr+1

!Adding up the any other forces(FVX(I,J,K)) with the penalisation forcing
    term, (F(x,y,z)=F(x,y,z)+F_penal(x,y,z)) corresponds to Equation(3.5)

FVX(I,J,K) = FVX(I,J,K) + ((pendat(11,13))* (UU(I,J,K)- pendat(11,14+cntr)))
    /pendat(11,1)
FVY(I,J,K) = FVY(I,J,K) + ((pendat(11,13))* (VV(I,J,K)- pendat(11,14+size(
    penU0)+cntr))) /pendat(11,1)
FVZ(I,J,K) = FVZ(I,J,K) + ((pendat(11,13))* (WW(I,J,K)- pendat(11,14+size(
    penU0)+size(penV0)+cntr))) /pendat(11,1)

endif
endif
endif

```

As discussed before, the *blending region* is similar to penalisation region but with a smoother transition of the flow into the domain. The forcing term ( $F_{blend}$ ) can be given by:

$$F_{blend} = \frac{\chi_{blend}}{\eta_{blend}} (\overline{u_i}^{pSim} - \overline{u_i}) \quad (3.6)$$

<sup>4</sup>The complete *velo.f90* source code is in Appendix I

The *blending factor*  $\chi_{blend}$  varies depending on whether it is an inlet or an outlet and the generalisation along  $x$  can be given by the Equation(3.7).

$$\chi_{blend}(x, y, z) = \begin{cases} -x + X_{max-inlet} & \text{if } x \in [X_{min-inlet}, X_{max-inlet}] , \\ x - X_{min-outlet} & \text{if } x \in [X_{min-outlet}, X_{max-outlet}] , \\ 0 & \text{otherwise} \end{cases} \quad (3.7)$$

The relaxation scale( $\eta_{blend}$ ) also known as a '*blending parameter*' can be given depending on the type of *pSim* data. In the current study, velocity data at particular times have been used to validate the method, which may have some turbulence information. If the data used does not contain 'turbulence' or eddies, then difficulties of miss-matched eddies do not occur. Therefore, a fixed value of  $\eta_{blend}$  is taken, sufficient to damp the eddies in the blending region. Therefore, this *blending parameter* has been determined by a 'trial-and-error' approach, a priori, for the current study. However, for time-varying *pSim* data,  $\eta_{blend}$  can be determined by the turbulence integral timescale as given by Equation(3.8) and can be computed a priori from the *pSim* data.

$$\eta_{blend} = \int_0^{\infty} \frac{\langle \bar{u}(t) \bar{u}(t + \tau) \rangle}{\langle (\bar{u}(t))^2 \rangle} d\tau \quad (3.8)$$

The complete code including the implementation of a blending region has been included in the *velo.f90* source file<sup>3</sup> This part of the implementation can be depicted by the following lines of code:

```

|!blending region start
|!checking if inside blending boundaries
|if ((Z(K) .le. pendat(11, 9)) .and. (Z(K) .ge. pendat(11, 8))) then
|if ((Y(J) .le. pendat(11, 7)) .and. (Y(J) .ge. pendat(11, 6))) then
|if ((X(I) .le. pendat(11, 5)) .and. (X(I) .ge. pendat(11, 4))) then
|
|!assigning the counter to traverse the array(pendat) where the penalisation

```

<sup>3</sup>The complete *velo.f90* source code is in Appendix I

```

        velocities are stored to do further velocity calculations
cntr=cntr+1

!Adding up the any other forces(FVX(I,J,K)) with the blending forcing term,
        (F(x,y,z)=F(x,y,z)+F_blend(x,y,z)) corresponds to Equation(3.6)
FVX(I,J,K) = FVX(I,J,K) + (pendat(11,13) + pendat(11,10)*X(I)-((pendat(11
        ,10)*(1-pendat(11,10))*pendat(11,5))/2d0) &
-(pendat(11,10)*(1+pendat(11,10))*pendat(11,4)/2d0))*(UU(I,J,K)-pendat(11
        ,14+cntr))/pendat(11,2) &
+ (pendat(11,13) + pendat(11,11)*Y(J)-((pendat(11,11)*(1-pendat(11,11))*
        pendat(11,7))/2d0) &
-(pendat(11,11)*(1+pendat(11,11))*pendat(11,6)/2d0))*(UU(I,J,K)-pendat(11
        ,14+cntr))/pendat(11,2) &
+ (pendat(11,13) + pendat(11,12)*Z(K)-((pendat(11,12)*(1-pendat(11,12))*
        pendat(11,9))/2d0) &
-(pendat(11,12)*(1+pendat(11,12))*pendat(11,8)/2d0))*(UU(I,J,K)-pendat(11
        ,14+cntr))/pendat(11,2)

FVY(I,J,K) = FVY(I,J,K) +(pendat(11,13) + pendat(11,10)*X(I)-((pendat(11
        ,10)*(1-pendat(11,10))*pendat(11,5))/2d0) &
-(pendat(11,10)*(1+pendat(11,10))*pendat(11,4)/2d0))*(VV(I,J,K)-pendat(11
        ,14+size(penU0)+cntr))/pendat(11,2) &
+ (pendat(11,13) + pendat(11,11)*Y(J)-((pendat(11,11)*(1-pendat(11,11))*
        pendat(11,7))/2d0) &
-(pendat(11,11)*(1+pendat(11,11))*pendat(11,6)/2d0))*(VV(I,J,K)-pendat(11
        ,14+size(penU0)+cntr))/pendat(11,2) &
+ (pendat(11,13) + pendat(11,12)*Z(K)-((pendat(11,12)*(1-pendat(11,12))*
        pendat(11,9))/2d0) &
-(pendat(11,12)*(1+pendat(11,12))*pendat(11,8)/2d0))*(VV(I,J,K)-pendat(11
        ,14+size(penU0)+cntr))/pendat(11,2)

FVZ(I,J,K) = FVZ(I,J,K) +(pendat(11,13) + pendat(11,10)*X(I)-((pendat(11
        ,10)*(1-pendat(11,10))*pendat(11,5))/2d0) &
-(pendat(11,10)*(1+pendat(11,10))*pendat(11,4)/2d0))*(WW(I,J,K)-pendat(11
        ,14+size(penU0)+size(penV0)+cntr))/pendat(11,2) &
+ (pendat(11,13) + pendat(11,11)*Y(J)-((pendat(11,11)*(1-pendat(11,11))*
        pendat(11,7))/2d0) &

```

```

- (pendat (11, 11) * (1 + pendat (11, 11)) * pendat (11, 6) / 2d0 ) * (WW(I, J, K) - pendat (11
, 14 + size (penU0) + size (penV0) + cntr) ) / pendat (11, 2) &
+ (pendat (11, 13) + pendat (11, 12) * Z(K) - (pendat (11, 12) * (1 - pendat (11, 12)) *
pendat (11, 9)) / 2d0) &
- (pendat (11, 12) * (1 + pendat (11, 12)) * pendat (11, 8) / 2d0 ) * (WW(I, J, K) - pendat (11
, 14 + size (penU0) + size (penV0) + cntr) ) / pendat (11, 2)

endif
endif
endif

```

For implementing both the *penalisation* and *blending* regions, the same array *pendat* has been used, which is re-allocated everytime with a dynamic size, everytime when the values of these regions are read. The *pendat* array stores all the information related to the *penalisation* and *blending* regions as implemented by following lines of code (included in the *read.f90* source file)<sup>4</sup>:

```

pendat (11, :) = (/ penalizationParameter, blendingParameter, dampingParameter,
&
penXmin, penXmax, penYmin, penYmax, penZmin, penZmax, &
mX, mY, mZ, b, timestep, &
penU0(:, :, :), penV0(:, :, :), penW0(:, :, :)/)

```

Here, *penXmin*, *penXmax* determines the *penalisation* and *blending* regions along x-direction, *penYmin*, *penYmax* determines that in y-direction and *penZmin*, *penZmax* along z-directions. *mX*, *mY*, *mZ* are relevant to the *blending* region as they determine whether it is an inlet or an outlet. In the current study, the inlet and outlets have been considered only along x-direction. Hence, *mY*, *mZ* have been assigned 0 value for all the cases. These values are set to -1 if the surface is an inlet (introducing velocity into the flow domain) in that particular direction, 1 if the surface is an outlet (removing velocity from the flow domain) and 0 if the surface is neither. For example, 1, 0, 0 represents a blending outlet in the x-direction, whereas 0, -1, 0 represents a blending inlet in the y-direction. It is theoretically feasible that the boundaries change in time from having positive net flux (inlet) to negative net flux (outlet) as the wind direction changes. For simplicity, the current study

<sup>4</sup>The complete *read.f90* source code is in Appendix I

is restricted by fixing the sign of this parameter a priori. This means, the regions have been specified globally as inlets and outlets and do not change over time. In case of the *penalisation* region, all the values of  $mX, mY, mZ$  are set to 0.  $b$  represents whether it is the *blending region* or *penalisation region*. When  $b = 1$ , it corresponds to *penalisation region* and when  $b = 0$ , it corresponds to *blending region*.  $timestep$  determines the time step of FS at which the *pSim* data will be read.  $penU0(:, :, :), penV0(:, :, :), penW0(:, :, :)$  represents the velocity components of *pSim* along  $x, y, z$  directions. An arbitrary number of blending regions can be defined along with the penalisation region, as per the requirement.

### 3.1.1.2 Pre-processing of *pSim* data

The Pena-Blending method requires a pre-processing phase for the *pSim* data. As already discussed, this *pSim* data can be obtained from analytical methods (e.g., generated from Matlab), any terrain modified reduced model (eg. Windninja), field methods like a log-law specified everywhere in the domain, or a precursor LES simulation. The FS will typically require a finer grid than the *pSim*. Therefore the *pSim* data needs to be interpolated into the finer grid similar to the FS. There can be two approaches of interpolating this data: firstly, *pSim* data can be interpolated in a pre-processing step and read in a large boundary file; or secondly, read in a small data file and interpolated in each time-step. In the current research, we have considered that the *pSim* data points are uniformly spaced in space and time. If the *pSim* data points are non-uniformly spaced, then a pre-processing interpolation step to convert them to a uniformly spaced grid will be required. FS grid points are typically uniform in space and time. Performing interpolation at each time-step can be computationally expensive and a very complex coding structure may be required. In order to avoid this, the *pSim* data points will be interpolated to the FS grid points a priori, in a straightforward interpolation step that can be performed using Matlab.

The main advantage of the Pena-Blending method is that time-varying velocity data can be read for performing fire simulations in FDS. The velocity data will be in the form of csv files, with information of all the velocities at each grid point and each desired timestep  $(u, v, w, t_i)$  where  $t_i$  is the timestep of the *pSim* data. It is not a practical approach to read *pSim* data for each and every simulation of time-steps. For instance, suppose

a representative simulation has 500 points in both  $x$  and  $y$  directions and 100 points in  $z$  direction. For a single time-step, the size on the disc  $pSim$  will be approximately 230 megabytes. Suppose the time step of the  $pSim$  simulation is 1 second. Typically, a fire simulation is of the order of minutes, which implies that the total file size can increase to as much as 115 gigabytes, which is impractical. In order to cope up with this situation, the coarse spatial  $pSim$  data is interpolated at every  $n$  simulation time-step to FS resolution and loaded into the memory once and for all at the start of the simulation. Loading  $pSim$  data at fewer timesteps can further reduce the memory usage. Within the FDS code, each mesh is treated separately and a simulation domain may be composed of several meshes. Therefore, the  $pSim$  data must be decomposed into pieces of data for each particular mesh in the FS. Provided that the grid sizes are known, the interpolation from coarser  $pSim$  grid to a grid equivalent to FS is straightforward and has a very low computational cost. Therefore, spatial interpolation at each time-step is performed on the coarse  $pSim$  data and the data files are read '*once-and-for-all*' at the start of the simulation. This requirement could be relaxed later on, by reading several different wind field files throughout the simulation. The algorithm for pre-processing of  $pSim$  data is depicted in Figure-3.3.

### 3.1.1.3 Reading the data files and penalisation region

The most challenging part of the process is to read the penalisation regions from the input file and the corresponding csv files to calculate the velocity. A new namelist group *PENA* has been introduced in FDS 6.6.0 to input the penalisation data in the input file. The namelist *PENA* should be written in following way in the FDS input file:

```
&PENA    penalizationParameter=2,
pena_I=6,
pena_J=40,
pena_K=80,
dataFileName='sample_read_pena_in_1.csv'
/
&PENA    blendingParameter=1e-1,
pena_I=6,
pena_J=40,
```



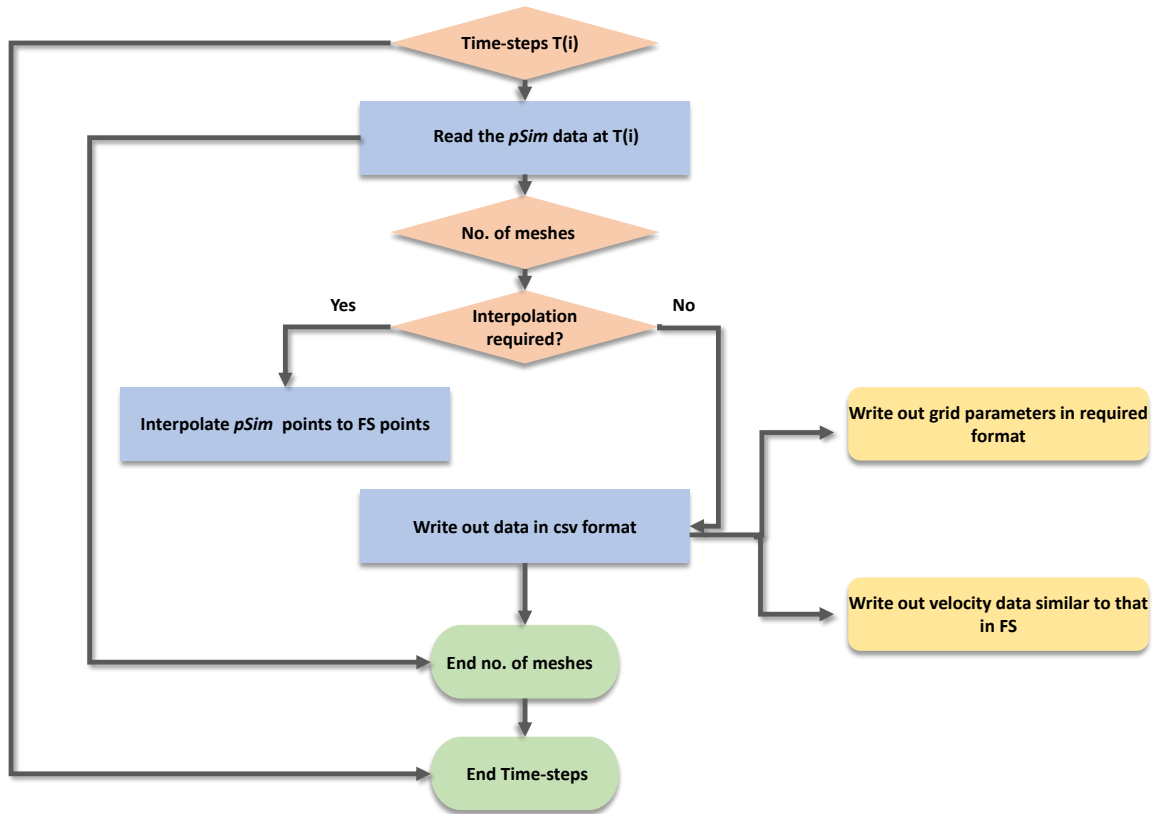


Figure 3.3: Flowchart representation of the pre-processing algorithm for  $pSim$  data

```

pena_K=80,
dataFileName='sample_read_blend_in_1.csv'

```

Here the *penalizationParameter* and *blendingParameter* represents the  $\eta_{penal}$  and  $\eta_{blend}$  respectively, and *pena\_I*, *pena\_J*, *pena\_K* represents the number of grid points in three different directions in the penalisation and blending regions. The *dataFileName* reads in the csv file with the  $pSim$  data. This csv file should comply with the following format as depicted in Figure-3.4. *penXmin*, *penXmax* defines the penalisation/blending region extent along  $x$ -direction, *penYmin*, *penYmax* defines that along  $y$ -direction and *penZmin*, *penZmax* along  $z$ -direction. *mX*, *mY*, *mZ* depicts whether it is inlet or outlet along  $x$ ,  $y$ ,  $z$ -directions, *b* represents whether this is penalisation region or blending region and *pena\_I*, *pena\_J*, *pena\_K* have similar values as that in input file. The time-

step value is depicted in the next line of the csv file, below which there are the velocity component data along  $x$ ,  $y$ ,  $z$ -directions.

penXmin	penXmax	penYmin	penYmax	penZmin	penZmax	mX	mY	mZ	b	pena_I	pena_J	pena_K	
0	6	0	40	0	80	0	0	0	0	1	3	20	80
2													
4.6848	0	0											
4.6848	0	0											
4.6848	0	0											
4.6848	0	0											
4.6848	0	0											
4.6848	0	0											
4.6848	0	0											
4.6848	0	0											
4.6848	0	0											
4.6848	0	0											

Figure 3.4: Format for writing out *pSim* data

There may be as many penalisation/blending regions as required and the number of these regions are counted using the following snippet of the *read.f90*<sup>5</sup>:

```
!counting the number of penalisation regions mentioned in input file
COUNT_PEN_LOOP: DO

CALL CHECKREAD('PENA', LU_INPUT, IOS)
IF (IOS==1) EXIT COUNT_PEN_LOOP
READ(LU_INPUT, NML=PENA, END=66, ERR=17, IOSTAT=IOS)
!write(*, nml=PENA)
!print*, 'pena_I, pena_J, pena_K =', pena_I, pena_J, pena_K
npen=npen+1
pendat_size(npen)=(pena_I +1)*(pena_J +1)*(pena_K +1)
ENDDO COUNT_PEN_LOOP
```

Each csv file read with *pSim* data is read and stored into dynamic arrays, to be later used in calculating the velocities at each time-step. This can be depicted in the following code-

<sup>5</sup>The complete *read.f90* source code is in Appendix I

segment from *read.f90*:<sup>5</sup>

```

READ(fileread,*) !skipping the first line
READ(fileread,*) penXmin,penXmax,penYmin,penYmax,penZmin,penZmax,mX,mY,mZ,b
    ,pena_I, pena_J, pena_K
READ(fileread,*) timesttep

if(ALLOCATED(penU0)) deallocate(penU0)
allocate(penU0((pena_I+1), (pena_J+1), (pena_K+1)))

if(ALLOCATED(penV0)) deallocate(penV0)
allocate(penV0((pena_I+1), (pena_J+1), (pena_K+1)))

if(ALLOCATED(penW0)) deallocate(penW0)
allocate(penW0((pena_I+1), (pena_J+1), (pena_K+1)))

DO penZ=1,pena_K+1
DO penY=1,pena_J+1
DO penX=1,pena_I+1
READ(fileread,*,IOSTAT=IERROR) penU0(penX,penY,penZ),penV0(penX,penY,penZ),
    penW0(penX,penY,penZ)

IF (IERROR/=0) THEN
penU0(penX,penY,penZ)=0._EB
penV0(penX,penY,penZ)=0._EB
penW0(penX,penY,penZ)=0._EB
ENDIF
ENDDO
ENDDO
ENDDO

```

The implementation cases for working of the PenaBlending method are discussed in Chapter 4. The implementation cases comprise of both wind and fire simulations. Tables-3.1 and 3.2 provide the list of simulation cases for the current study. Table-3.3 provides the list of all the parameters used for carrying out the *wind* and *fire* simulation in the current study.

**Table 3.1:** Wind simulation cases - for both large and small domain

Case	Generation method	Mean Profile	Turbulent Profile
wind1	Wall-of-wind (2.2.1)	1/7–power-law	Random number
wind2	SEM (2.2.2)	Log-law	SEM
wind3	Mean-forcing (2.2.3)	Log-law	Random Number
wind4	<i>PenaBlending Method</i> (3.1.1)	Log-law	Random Number
demo	Windninja	Terrain perturbed	SEM

**Table 3.2:** Fire simulation cases - for both large and small domain

Case	Generation method	Mean Profile	Turbulent Profile
fire1	Wall-of-wind (2.2.1)	1/7–power-law	Random Number
fire2	SEM (2.2.2)	Log-law	SEM
fire3	Mean-forcing (2.2.3)	Log-law	Random Number
fire4	<i>PenaBlending Method</i> (3.1.1)	Log-law	Random Number
fire5	Underdeveloped ABL	1/7–power-law	Random Number

**Table 3.3:** Simulation parameter values and characteristics - (a) Numerical parameters used for small domain and large domain for both fire as well as wind-only simulations mentioned in 3.2 and 3.1; (b) List of boundary conditions used for wind-only and fire simulations; (c) Fuel parameters used for running fire simulations.

Numerical parameters ( <i>small domain</i> )	
Domain size	130 × 40 × 80 m
Grid spacing	$\delta x(\text{for } 50 \leq x \leq 90) = \delta y = \delta z(\text{for } z \leq 6\text{m}) = 250 \text{ mm}$ (fire simulations) $\delta x(\text{for } 50 \leq x \leq 90) = \delta y = \delta z(\text{for } z > 6\text{m}) = 1.0 \text{ m}$ (fire simulations) $\delta x(\text{for } x < 50, x > 90) = \delta y = \delta z = 1.0 \text{ m}$ (fire simulations) $\delta x = \delta y = \delta z = 1.0 \text{ m}$ (wind-only simulations)
Burnable grass plot	40 m X 40 m
Turbulence model	Deardorff Model $C_v = 0.1$
Numerical parameters ( <i>large domain</i> )	
Domain size	600 × 300 × 100 m
Grid spacing	$\delta x(\text{for } 300 \leq x \leq 400) = \delta y = \delta z(\text{for } z \leq 6\text{m}) = 250 \text{ mm}$ (fire case) $\delta x(\text{for } 300 \leq x \leq 400) = \delta y = 1.0\text{m}, \delta z(\text{for } z > 6\text{m}) = 0.5 \text{ m}$ (fire case) $\delta x(\text{for } x < 300, x > 400) = \delta y = 2.0\text{m}, \delta z = 1.0 \text{ m}$ (fire case) $\delta x = \delta y = 2 \text{ m}, \delta z = 1.0 \text{ m}$ (wind-only simulations)
Burnable grass plot	100 m X 300 m
Turbulence model	Deardorff Model $C_v = 0.1$

(a)

Boundary conditions	<i>wind simulations</i>
Lateral	Periodic ( <i>wind1, wind2</i> ) Open ( <i>wind3</i> ) Free-slip, no normal velocity ( <i>wind4</i> )
Bottom (ground)	No-slip
Top (sky)	Free-slip, no normal velocity
Inlet	Refer to Table-3.1
Roughness length $z_0$	0.9 m
$L_{eddy}$	10 m (for <i>wind2</i> )
$N_{eddy}$	40 (for <i>wind2</i> )
$\sigma_{eddy}(large\ domain)$	1.0 ms <sup>-1</sup> if $z < 20$ m 0.5 ms <sup>-1</sup> if $20 \leq z < 40$ m 0 ms <sup>-1</sup> if $z \geq 40$ m
$\sigma_{eddy}(small\ domain)$	0.185 ms <sup>-1</sup>
Outlet	Open ( <i>wind1, wind2, wind3, wind4</i> )
Temperature BCs	zero fluxes
Boundary conditions	<i>fire simulations</i>
Lateral	Periodic ( <i>fire1, fire2, fire5</i> ) Open ( <i>fire3</i> ) Free-slip, no normal velocity ( <i>fire4</i> )
Bottom (ground)	No-slip
Top (sky)	Free-slip, no normal velocity
Inlet	Refer to Table-3.2
Roughness length $z_0$	0.9 m
$L_{eddy}$	10 m (for <i>fire2</i> )
$N_{eddy}$	40 (for <i>fire2</i> )
$\sigma_{eddy}(large\ domain)$	1.0 ms <sup>-1</sup> if $z < 20$ m 0.5 ms <sup>-1</sup> if $20 \leq z < 40$ m 0 ms <sup>-1</sup> if $z \geq 40$ m
$\sigma_{eddy}(small\ domain)$	0.185 ms <sup>-1</sup>
Outlet	Open
Temperature BCs	zero fluxes

(b)

Fuel Parameters	Used Values	Source
Drag coefficient	0.125	Following <a href="#">Morvan and Dupuy (2004)</a>
Vegetation load	0.4245 kg m <sup>-2</sup>	Following <a href="#">Moinuddin et al. (2018)</a>
Vegetation height	0.315 m	Following <a href="#">Moinuddin et al. (2018)</a>
Moisture content	0.063 %	Following <a href="#">Mell et al. (2007)</a>
Element surface/ volume ratio	9770 m <sup>-1</sup>	Following <a href="#">Mell et al. (2007)</a>
Element density	440 kg m <sup>-3</sup>	Following <a href="#">Moinuddin et al. (2018)</a>
Char fraction	17%	Following <a href="#">Moinuddin et al. (2018)</a>
Emissivity	99 %	Following <a href="#">Mell et al. (2007)</a>
Maximum mass loss rate	0.15 kg m <sup>2</sup> s <sup>-1</sup>	Following <a href="#">Mell et al. (2007)</a>

(c)

In order to carry out fire simulations, a combustion model is needed. A reaction rate limited combustion model is required to reduce the computational expense, as well as reduce the numerical instabilities. [Moinuddin et al. \(2018\)](#) used grid convergence based on combustion model of FDS 6.2.0 and before. That grid sensitivity study is not valid for the current FDS 6.6.0, and hence the combustion model of FDS 6.2.0 has been re-introduced. The re-introduction is explained in the next section.

### 3.2 Re-introduction of FDS 6.2.0 Combustion Model into FDS 6.6.0

The rate of heat generation by fire, known as the Heat Release Rate (HRR) is a very critical parameter to characterize a fire. There are various methods to estimate it. FDS calculates the Heat Release Rate per unit volume (HRRPUV,  $\dot{q}'''$ ) using Equation(3.9).

$$\dot{q}''' = - \sum_{\alpha} \dot{m}_{\alpha}''' \Delta H_{f,\alpha} \quad (3.9)$$

where  $\dot{m}_{\alpha}'''$  represents the lumped species mass production rates and  $\Delta H_{f,\alpha}$  represent the respective heat of formation. FDS while doing the calculations for computing HRR, certain critical point calculation, like the moment of ignition, can lead to very high local reaction rates. This could be as a result of limitations of the model or lengthy time-steps

or both. Such fictitiously high reaction rates can cause *numerical instabilities*. In order to prevent this, an upper bound needs to be imposed on the local HRRPUV. In the previous versions of FDS (FDS 6.2.0), the combustion model had this limiting upper bound on HRRPUV. Following the scaling analysis of pool fires <sup>6</sup> by Orloff and De Ris (1982), FDS imposes an upper bound using Equation(3.10).

$$q''_{upper} = 200/\delta x + 2500(KW/m^3) \quad (3.10)$$

where  $\delta x$  represents the characteristic cell size (in metres) and the value  $200KW/m^2$  is the upper bound on the heat release rate per unit area of the fire flame section, obtained empirically.

With the release of newer versions of FDS, this reaction rate threshold has been removed. These versions expect the computation grid to be sufficiently resolved to avoid such numerical instabilities. Hence, the current version of FDS (FDS 6.6.0), that has been used in current research, does not have this HRRPUV upper bound. However, for large scale wildfire simulations, it is difficult to maintain such smaller grid resolutions throughout as it will increase the computation cost extensively. In order to avoid this, the upper bound threshold equation has been re-introduced in the combustion model of FDS 6.6.0. This has been done with the following purpose :

- to be consistent with the previous fire simulations that were carried out by Moinuddin et al. (2018).
- to use the grid resolution for fire simulations from the grid sensitivity analysis as done by Moinuddin et al. (2018) using FDS 6.2.0.
- to avoid the restrictive grid resolution requirement and avoid numerical instabilities for large-scale fire simulations

### 3.2.1 Implementing the re-introduced combustion model in FDS 6.6.0

The implementation has been done in a way so that there is an option for the user to choose between the in-built combustion model or the newly introduced combustion

---

<sup>6</sup>Pool fire can be defined as a pool or pile of flammable substance catching fire.



model of FDS 6.6.0. In order to make this selection, the user can put the following command in the *MISC* line :

```
&MISC COMBUSTION_MODEL_SELECT = COMBUSTION_TWO, SIX = .FALSE./
```

Selecting this model will invoke the reaction rate limiters using the Equation(3.11)

$$\dot{q}_{max}'' = HRRPUA\_SHEET/\partial x + HRRPUV\_AVERAGE \quad (3.11)$$

similar to FDS 6.2.0 [McGrattan et al. \(2015\)](#). This has been included in the *fire.f90*<sup>1</sup> source file with the following source-code:

```
! Upper bounds on local HRR per unit volume
Q_UPPER = HRRPUA_SHEET/CELL_SIZE + HRRPUV_AVERAGE
```

The default values for *HRRPUA\_SHEET* is 200 KW/m<sup>2</sup> and that of *HRRPUV\_AVERAGE* is 2500 KW/m<sup>3</sup>. These default values were obtained from [Orloff and De Ris \(1982\)](#). These parameters can be changed with user defined values in the *MISC* line as discussed in [McGrattan et al. \(2015\)](#).

In order to select the in-built combustion model of FDS 6.6.0, the user needs to mention the following in the *MISC* line :

```
&MISC COMBUSTION_MODEL_SELECT = COMBUSTION_SIX, SIX = .TRUE./
```

Upon selecting this, the existing combustion model will be invoked and the heat calculations will be carried out accordingly. However *HRRPUA\_SHEET* and *HRRPUV\_AVERAGE* options will be disabled in the *MISC* line, and providing values to these in the input file will produce error and the simulation will be stopped. A select-case statement has been written in the *read.f90*<sup>2</sup> source file as given below:

```
SELECT CASE (TRIM(COMBUSTION_MODEL_SELECT))
CASE ('COMBUSTION_SIX')
COMB_MODEL=COMBUSTIONSIX
CASE ('COMBUSTION_TWO')
COMB_MODEL=COMBUSTIONTWO
END SELECT
```

These options are given in Table-3.4 for clarity.

<sup>1</sup>The complete *fire.90* source code is in Appendix A (A.1)

<sup>2</sup>The complete *read.90* source code is in Appendix I

	Input Parameters	Value	Reaction Rate Limiter Parameters
Existing Combustion Model	COMBUSTION_MODEL_SELECT	COMBUSTION_SIX	-
	SIX	TRUE	-
New Combustion Model	COMBUSTION_MODEL_SELECT	COMBUSTION_TWO	HRRPUA_SHEET
	SIX	FALSE	HRRPUV_AVERAGE

**Table 3.4:** FDS input parameters to be used for selecting the combustion model

The new combustion model has been checked in FDS 6.6.0 with that in FDS 6.2.0 using two cases as discussed in the below Subsection-3.2.2.

### 3.2.2 Verification Cases

FDS has been subjected to a number of validation and verification cases which can be found in McGrattan et al. (2017c). *couch.fds* is an example case for modelling coupled pyrolysis and combustion as given by McGrattan et al. (2017c). This example case has been run using the newly re-introduced combustion model in FDS 6.6.0 and the results are compared with that of the existing combustion model of FDS 6.6.0 and 6.2.0 to verify that the new combustion model is working. This is a simple example of a burning couch, where the fuel is considered as *propane*, the couch is made of *fabric, foam, gypsum plaster* with all properties mentioned in the input file, and the ignitor has been considered as a point source. The fire is started by a cylindrical ignitor particle with surface temperature of 1000°C. A uniform grid resolution of  $0.1m \times 0.1m \times 0.1m$  has been taken in this case. The detailed information about this case can be found in the Verification suite of FDS (<https://github.com/firemodels/fds/tree/master/Verification/Fires>). The simulation cases have been run using both the combustion models to verify whether the models are giving identical results. Figure-3.5 shows the domain setup for this case.

The HRR values have been plotted, which designates the intensity of the fire. Figure-

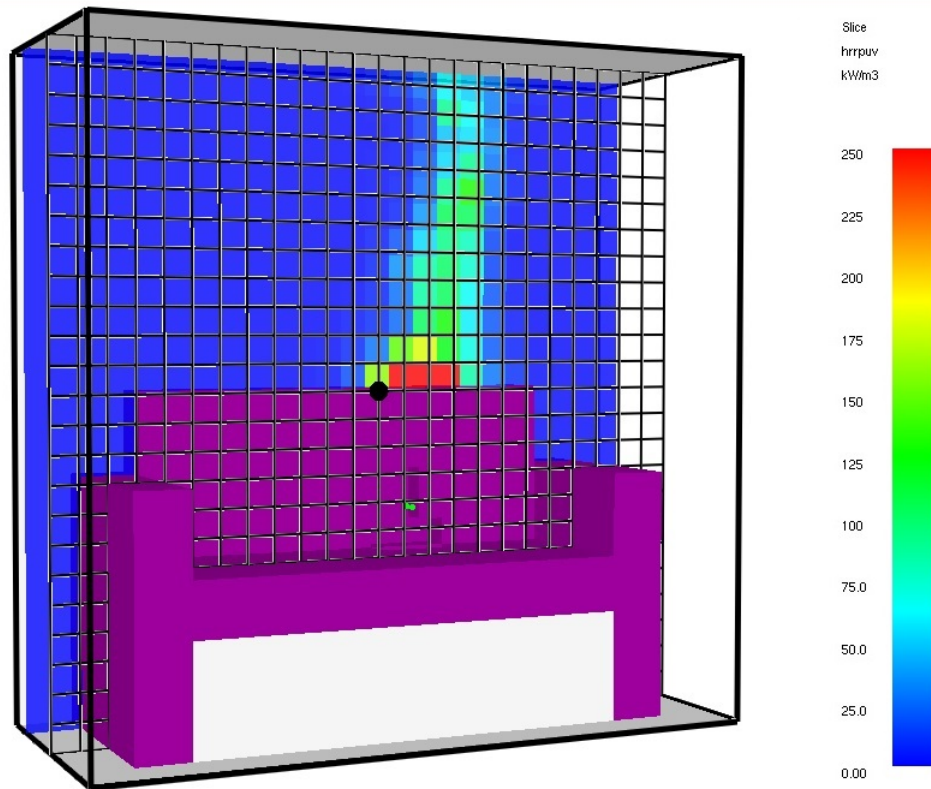
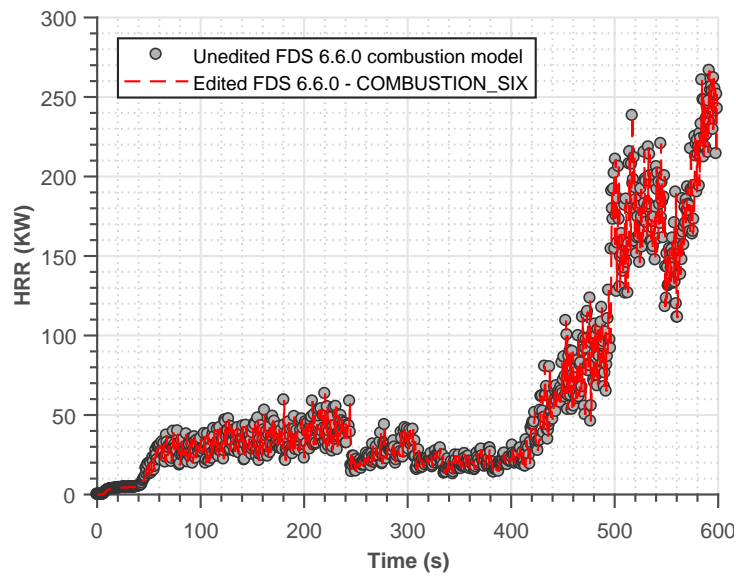


Figure 3.5: Domain setup for the couch case

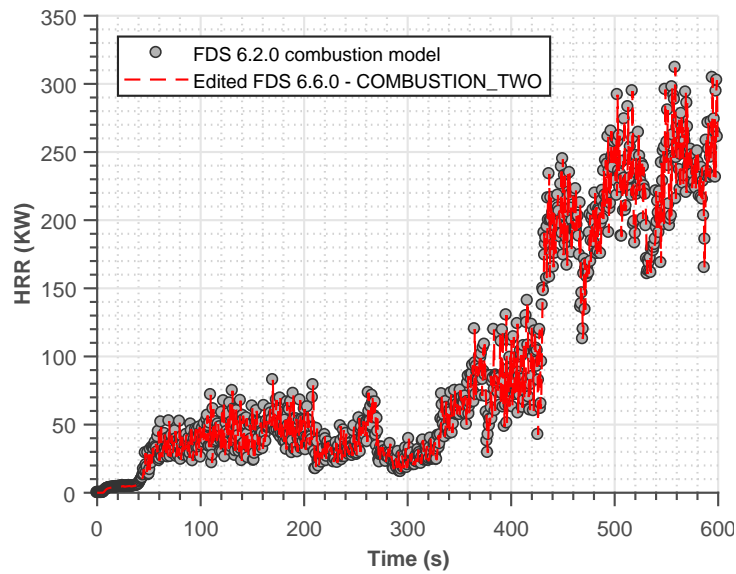
3.6 represents the HRR comparison of the fire generated by the original combustion model of FDS 6.6.0 with the selection of the same model using `COMBUSTION_MODEL_SELECT = 'COMBUSTION_SIX'` in `MISC` line for the edited FDS 6.6.0.

Figure-3.7 represents the HRR comparison of the fire generated by the combustion model of FDS 6.2.0 and that with selection of the option `COMBUSTION_MODEL_SELECT = 'COMBUSTION_TWO'` in the edited version of FDS 6.6.0. For both the cases, it is found that the figures clearly represent that the values are exactly the same and overlap each other, concluding that the newly introduced input parameters are working as expected.

For verifying the working of the new re-introduced combustion model in FDS 6.6.0 for



*Figure 3.6: Heat Release Rates comparison of in-built combustion model for edited and unedited versions of FDS 6.6.0 for couch case*



*Figure 3.7: Heat Release Rates comparison of FDS 6.2.0 combustion model and identical model introduced in FDS 6.6.0 for couch case*

fire over vegetation, simulation was run following [Moinuddin and Sutherland \(2019\)](#). In this simulation, a single tree with a set-up similar to the experiment performed by NIST (National Institute of Standards and Technology) with a Douglas fir tree species ([Mell et al. \(2009\)](#)) was burned. In this experiment trees 2.25 m high were placed on custom

stands to dry. After this process, they were set on fire using circular natural gas burners. The simulation carried out is similar to the experiment performed by Mell et al. (2009). In the simulation, a single tree is modeled with four different sized particles namely: foliage, small roundwood, medium roundwood and large roundwood. All these particles are considered to have a cylindrical shape. The fire was ignited using a burner similar to the experiment. This simulation has been recently carried out by Moinuddin and Sutherland (2019). As a part of the current work, this case has been considered as an example to verify the working of the re-introduced combustion model in FDS 6.6.0 and to compare the results with that of the existing combustion model of FDS 6.6.0 and that of FDS 6.2.0. The graphical representation of this simulation is given in Figure-3.8

Figure-3.9 represents the HRR comparison of the fire generated by the original combustion model of FDS 6.6.0 with the selection of the same model using `COMBUSTION_MODEL_SELECT='COMBUSTION_SIX'` in the `MISC` line for the edited FDS 6.6.0 similar to the previous case. It was observed that the plots overlap each other reasonably; hence, showing the option in the edited version was working properly. The small differences were attributed to the changes in other sub-models (from FDS 6.2.0 to FDS 6.6.0)

Figure-3.10 represents the HRR comparison of the fire generated by the combustion model of FDS 6.2.0 and that with selection of the option `COMBUSTION_MODEL_SELECT='COMBUSTION_TWO'` in the edited version of FDS 6.6.0, similar to previous case. It is observed that the edited combustion model plot follows a similar trend as the native combustion model of FDS 6.2.0 towards the start and end of the plot. There are some variations towards the peak, and both the plots do not overlap completely there. Overall, it can be concluded that the plots agree with each other reasonably well with slight variations. These variations can be attributed because of the changes in sub-models from FDS 6.2.0 to FDS 6.6.0. This shows that the combustion model of FDS 6.2.0 which is re-introduced into FDS 6.6.0 is working correctly and gives reasonable results.

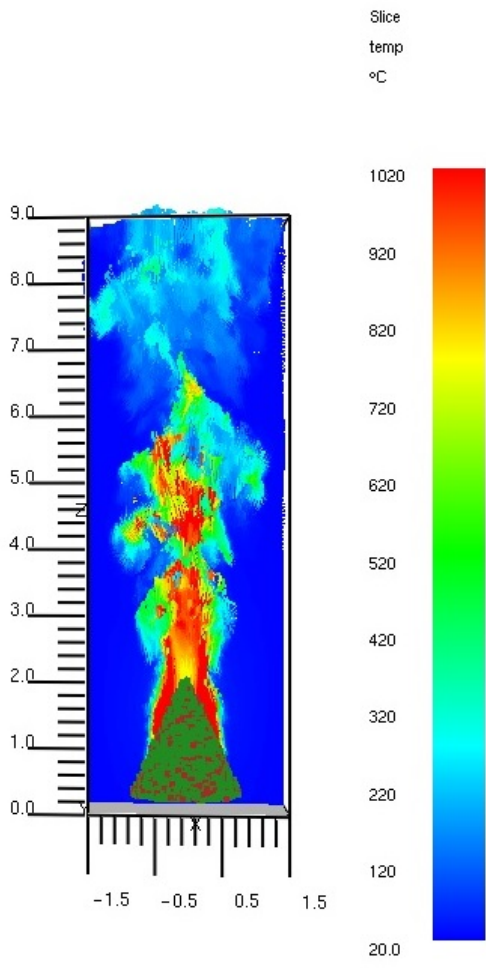


Figure 3.8: Tree burning case: This figure shows the temperature contours of burning the tree represented by the green triangle. The red colour represents the maximum temperature ( 1020°C),the fire plume region and the blue colour represents minimum temperature, the smoke region

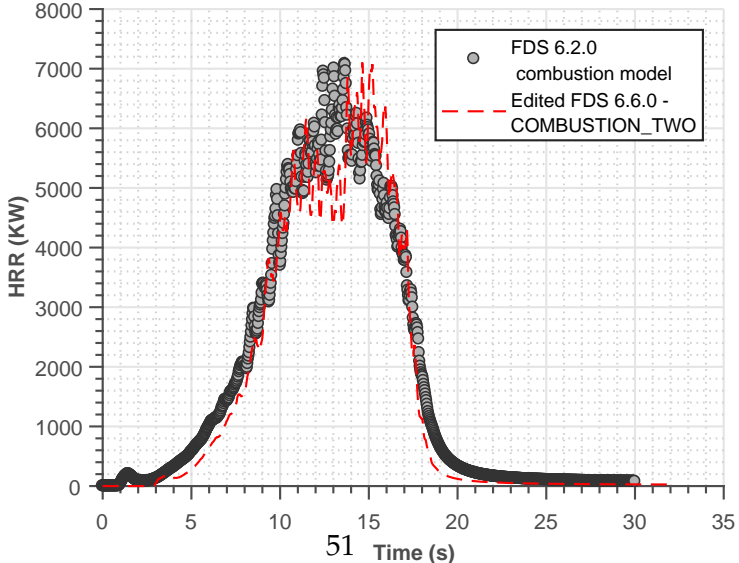
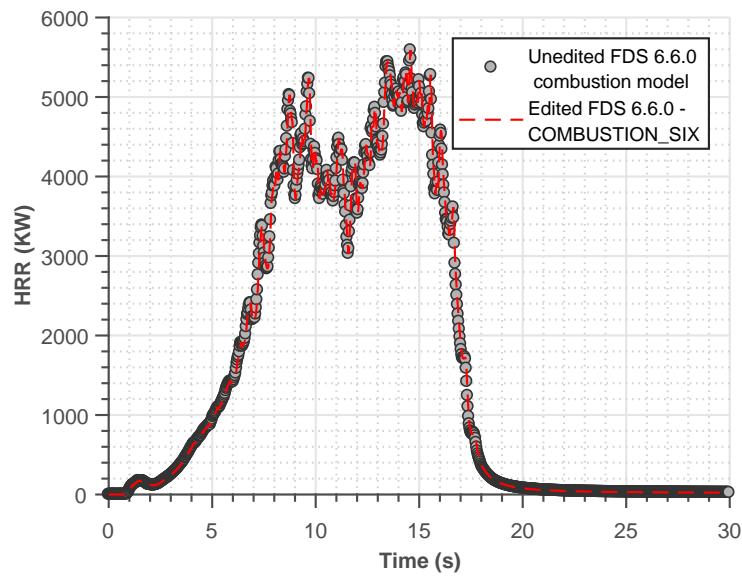


Figure 3.10: Heat Release Rates comparison of FDS 6.2.0 combustion model and identical model introduced in FDS 6.6.0 for the tree burning case



**Figure 3.9:** Heat Release Rates comparison of in-built combustion model for edited and unedited versions of FDS 6.6.0 for the tree burning case

This chapter gives a detailed overview of the code implementations that have been performed in the current study. Chapter 4 includes some case studies that have been carried out using the edited FDS 6.6.0, which prove the validity of the *PenaBlending* method. All the *fire* simulations conducted in this study use the new re-introduced model of FDS 6.2.0 in FDS 6.6.0.

# 4

## Chapter 4

### Test Results and Discussions

The effectiveness of the inlet condition implementation by the *PenaBlending Method* as discussed above has been tested in two sets of idealised simulations in a channel-flow configuration. The first set of simulations, referred to as *wind simulations*, is used to assess the development time and the quality of the developed atmospheric boundary layer. The *PenaBlending Method* has been compared to all the *traditional methods* (2.2) which are available in FDS. The second set, referred to as *fire simulations* is a set of fire simulations using different approaches of initialisation to directly quantify the effect of the initial and inlet conditions on fire simulation. In order to validate the new method, two types of domain sizes have been used which are named *Large domain* and *Small domain*. The detailed specification of these two domain types are given in Section-4.1. The ability of the *PenaBlending method* to model the effect of gusting wind on fire is demonstrated. Using wind fields from terrain modified reduced wind models, like Windninja, into FDS, have also been tested.

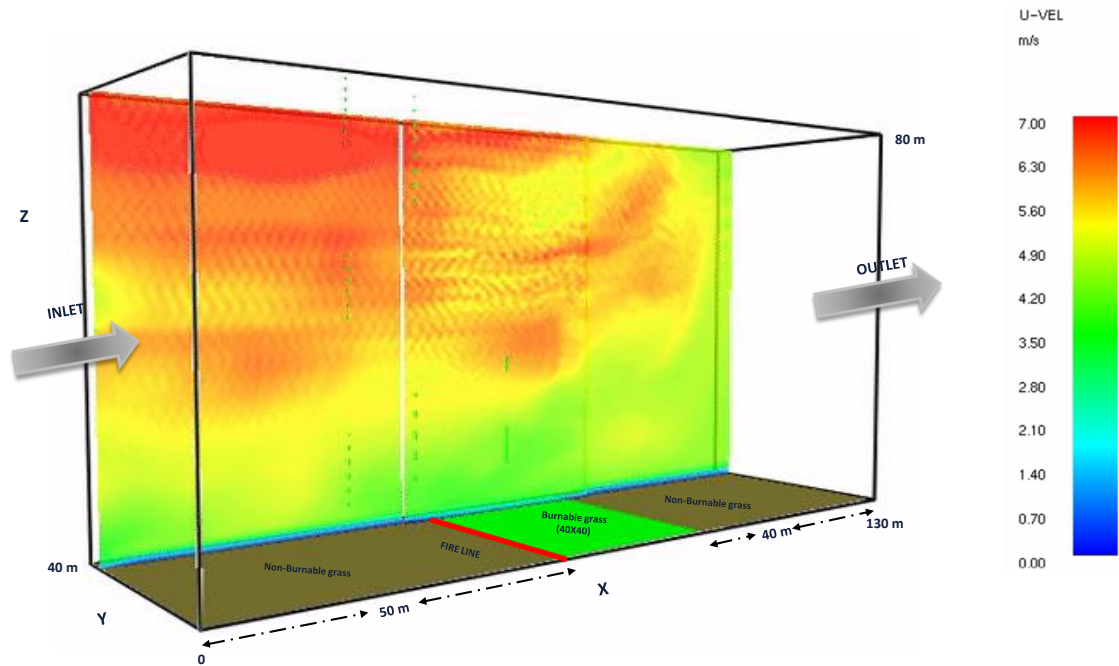
#### 4.1 Simulation Domain

A proper domain set-up is necessary to carry out simulations. The size of the external domain should be chosen so that it is able to capture all the pertinent fluid structures. In the current study, two types of domain size have been considered which are termed '*small domain*' and '*large domain*' in this thesis. The external sizes of the *small domain* have been chosen following [Singha Roy et al. \(2018\)](#) and the *large domain* has been chosen following [Sutherland et al. \(2018\)](#) for both the *wind* and *fire* simulations. The simulation domains



are setup in a channel-flow configuration. The height of the domain is chosen so that it is able to capture the plume (Moinuddin et al. (2018), Mell et al. (2007)). An infinitely long line fire is simulated following Linn et al. (2012) in the cross-stream direction. This means that the line -fire is extended throughout the width of the domain. This results in fire properties like depth of fire front, flame length, flame angle and rate-of-spread (RoS) of fire not varying along the cross-stream direction (in this case the  $y$ -direction). This configuration has been followed in the current study so that the fire-spread results can be easily averaged across the domain.

The *small domain* has a dimension of  $130 \text{ m} \times 40 \text{ m} \times 80 \text{ m}$  and is divided into a range of sizes. To avoid any instabilities or error in simulation, the aspect ratio is maintained less than 2 for each grid cell. For *wind* simulations, an uniform grid-resolution of 1 m is set for all the three directions ( $x,y,z$ ) Table-3.1. For the fire simulations, a uniform grid-resolution of 0.25 m is maintained on the fire-plot upto  $z = 6 \text{ m}$ . The rest of the domain has a uniform grid-resolution of 1 m. A list of numerical parameters used for the *small domain* is in Table-3.3-a. The burnable grass plot of dimension ( $40 \text{ m} \times 40 \text{ m}$ ) is located at  $x=40 \text{ m}$  from the inlet, followed by a non-burnable grass plot of 50 m down-stream before reaching the outlet. Enough distance is maintained in the up-stream of the fire plot to let the wind develop and reach a required state for starting the fire simulation. Similarly, enough distance in down-stream is also maintained so that the plumes can travel and escape the domain and finally, extinguish the fire completely. The fuel parameters are given in Table-3.3-c. Figure-4.1 represents a generalised domain set-up for the *small domain*.



**Figure 4.1:** A generalised schematic representation of Small Domain for simulation, representing the external dimensions, fire-line, fire plot and a slice of establishing ABL

The *Large domain* has a dimension of 600 m × 300 m × 100 m. The domain is divided into grid cells of various resolutions, as per requirements, with an aspect ratio maintained to be > 2 m, similar to the *small domain*. For the *wind* simulations, a 2 m × 2 m × 1 m grid-resolution is maintained uniformly throughout the domain. In case of the *fire* simulations, a grid-resolution of 0.25 m × 0.25 m × 0.25m is maintained over the fire plot upto z = 6 m. Above the fire plot (form z > 6 m), a resolution of 1 m × 1 m × 0.5 m is maintained and the upstream and the downstream of the fire plot has a 2 m × 2 m × 1m grid-resolution. The summary of these numerical parameters is given in Table-3.3-a. The fire plot of dimension 100 m × 300 m in the *large domain* is located at x = 300 m from the inlet, stretching throughout the width of the domain (y = (0,300)), with a down-stream of a further 200 m having non-burnable grass, similar to *small domain*. The fuel parameters are similar to the *small domain* and are given in Table-3.3-c. Figure-4.2 represents a generalised domain set-up for the *large domain* case.

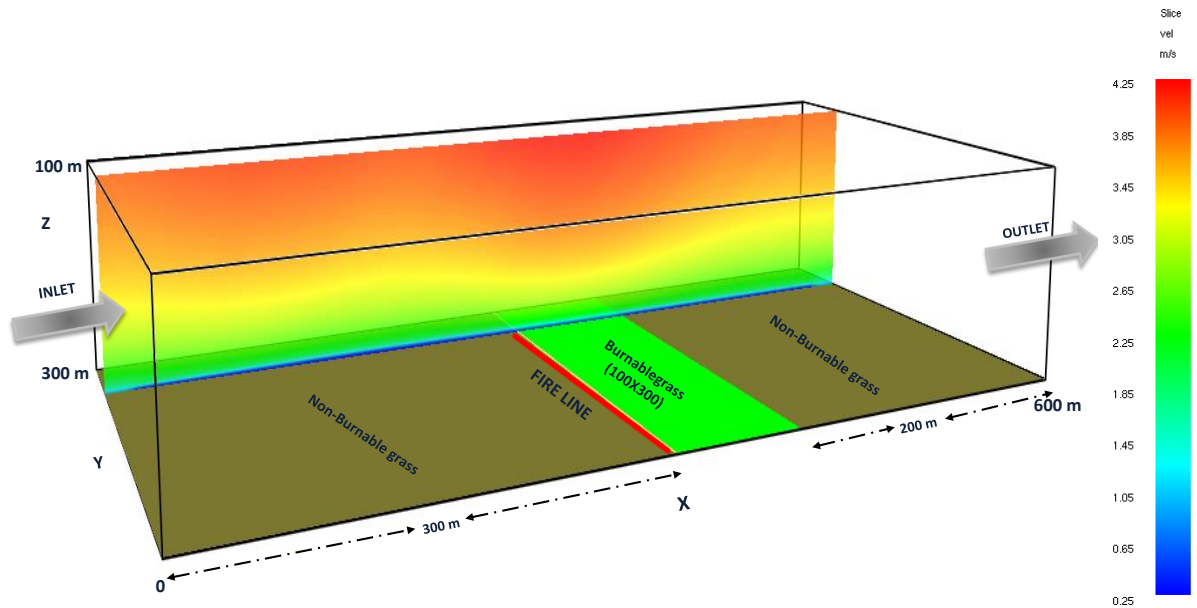


Figure 4.2: A generalised schematic representation of the large domain for a simulation representing the external dimensions, fire-line, fire plot and a slice of establishing ABL

## 4.2 Boundary Conditions

Boundary conditions play an important role in any type of simulation. A proper inlet boundary condition needs to be prescribed to obtain appropriate results from a fire simulation. The simulations carried out to test the *PenaBlending Method* in the current study are done in *neutral* atmospheric stability conditions, which means there is no effect of applied surface heat flux on any of the simulations. The *PenaBlending Method* sets the boundary conditions of the simulation domain similar to the boundary conditions of the external model or simulation (*pSim*) as discussed previously. In this study, the *PenaBlending Method* is applied only along the x-direction. Hence, the boundary conditions along x-direction are set as per *pSim*. So, for the modified FDS 6.6.0, in case of both *wind* and *fire* simulations, a *free-slip* or no normal velocity conditions are used in cross-stream directions, and at the top of the domain. The ground is prescribed as a solid boundary and is set to *no-slip*. The ground has vegetation comprising burnable and non-burnable grass.

A grass height of 0.315m has been taken as an inlet condition for all the simulations. A detailed overview of the fuel properties is given in Table-3.3-c

The simulations using *wall-of-wind* method (*wind1* and *fire1*) have an inlet of a 1/7-power law wind profile and an *open* outlet. The lateral or cross-stream boundaries are set to be *periodic* along with a *no-slip* and *free-slip* boundary conditions at  $z = 0$  and  $z = 80$  (for the *small* domain) and  $z = 100$  (for the *large* domain) respectively. Both the *SEM* (*wind2* and *fire2*) and the *mean-forcing* (*wind3* and *fire3*) methods have a *log-law* mean flow inlet profile as given in Tables-3.2 and 3.1. The details of the boundary conditions for all the simulation are summarized in Table-3.3-b. The schematic representations of the boundary conditions for the *small* and *large* domain is represented by Figures-4.3 and 4.4. The next sections discuss the results of the proposed simulations, both fire and wind, to implement the *PenaBlending Method*.

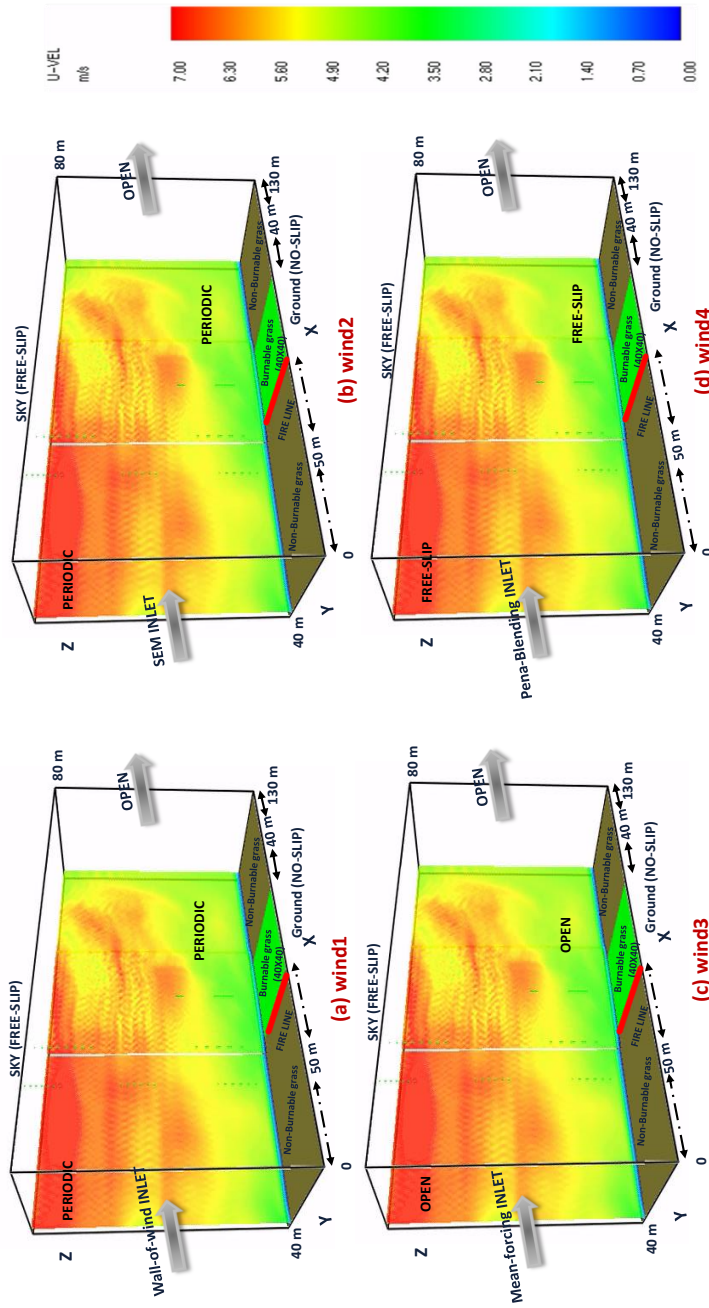


Figure 4.3: A generalised schematic representation of the domain set-up with boundary conditions of small domain cases corresponding to Table-3.3-b for (a) wind1; (b) wind2; (c) wind3; (d) wind4, including the external dimensions, fire-line, fire plot and a slice of establishing ABL. The fire simulations have a similar set-up

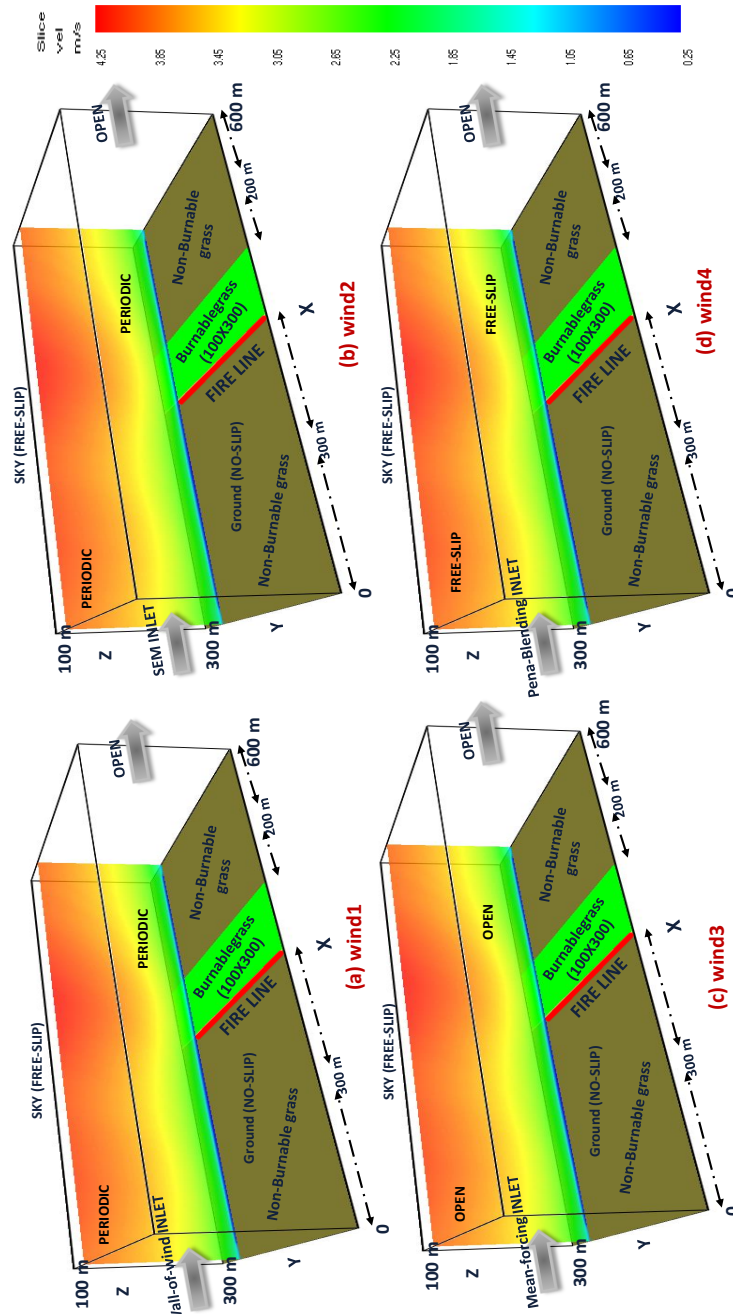


Figure 4.4: A generalised schematic representation of the domain set-up with boundary conditions of large domain cases corresponding to Table 3.3-b for (a) wind1; (b) wind2; (c) wind3; (d) wind4, including the external dimensions, fire-line, fire plot and a slice of establishing ABL. The fire simulations have a similar set-up



### 4.3 Wind-only cases - results and discussion

Development of wind to a statistically steady state is very important for starting the fire simulation. A steady-state wind condition ensures that the wind fields do not spuriously develop across the fire ground, leading to spurious simulation results. This section shows implementation of cases of *PenaBlending Method* against the *traditional methods* in the absence of a fire. The results of *PenaBlending Method* are compared with those obtained by the *traditional methods*. All the wind velocity results are averaged both in time and space denoted by  $\langle u \rangle_{t,s}$ . For convenience, the velocities are denoted by  $u$  and the mean velocity is denoted by  $\bar{u}$ . The main purpose of averaging across the domain is to reduce the noise, thereby making trends in the data more apparent.

Using the *PenaBlending method*, the wind develops with time as depicted in Figure-4.5 as follows.

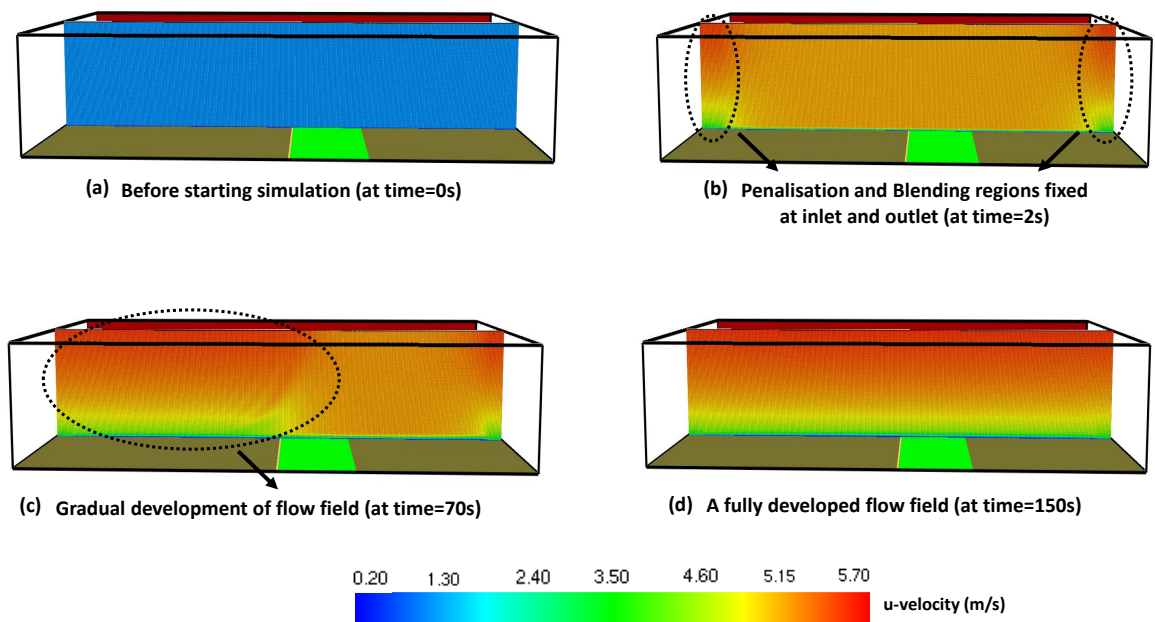


Figure 4.5: Development of wind-field using the *PenaBlending method* for large domain

The figure depicts a case with *large domain*. At 0 s, there is a zero velocity prevailing throughout the domain as depicted in Figure-4.5-a. The *PenaBlending Method* is applied

at the inlet and outlet where  $pSim$  data is read, at 2 s in this case as in Figure-4.5-b. As the simulation progresses, the wind field is developed from inlet towards outlet as depicted at 70 s. Figure-4.5-c depicts this scenario. By 150 s a fully developed wind field is obtained throughout the domain, as seen in Figure-4.5-d and a fire simulation can be commenced.

Table 4.1: Sampling time for large and small domain

Parameters	Small Domain	Large Domain
<b>Spin-up time :</b>	~ 600 – 800 s(wind1)	~ 1000 s(wind1)
	~ 400 – 500 s(wind2)	~ 800 s(wind2)
	> 100 s(wind3)	> 150 s(wind3)
	~ 80 – 100 s(wind4)	~ 200 s(wind4)
<b>Total Simulation time :</b>	3000 s	5000 s
<b>Measurement time :</b>	~ 1 s (varying)	~ 1 s (varying)
<b>Time (improvement):</b>	wind4 is ~ 85% of wind1	wind4 is ~ 80% of wind1
	wind4 is ~ 80% of wind2	wind4 is ~ 75% of wind2
	wind4 is similar to wind3	wind4 is similar to wind3
<b>Accuracy(<math>u_{10}</math>) :</b>	wind4 is ~ 97% of wind1	wind4 is ~ 96% of wind1
	wind4 is ~ 100% of wind2	wind4 is ~ 97% of wind2
	wind4 is ~ 96.6% of wind3	wind4 is ~ 98% of wind3
	Figures(4.6-a, 4.7-a)	Figures(4.6-b, 4.7-b)

As the simulation progresses, the wind field is developed from the inlet towards the outlet as depicted at 70 s. Figure-4.5-c depicts this scenario. By 150 s a fully developed wind field is obtained throughout the domain, as seen in Figure-4.5-d and a fire simulation can be commenced.

For the *traditional method*, the wind simulations for the *small domain* have been run for 3000 s and those for the *large domain* for 5000 s to test the wind development time. It has been observed that, in the case of the *small domain*, the spin-up time <sup>3</sup> for *wind1* is

<sup>3</sup>The time taken by the simulation to reach a statistically steady state wind profile



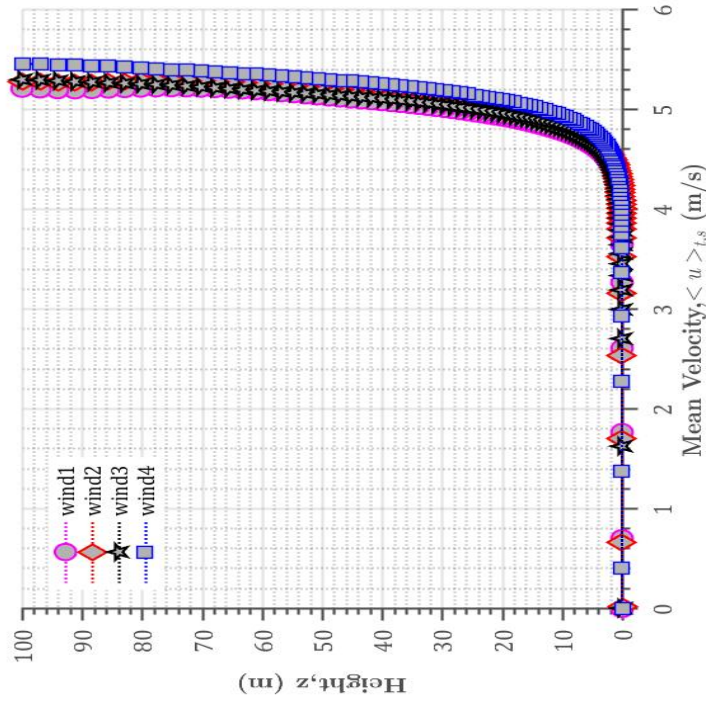
$\sim 600 - 800$  s,  $wind2$  is  $\sim 400 - 500$  s and  $wind3$  is less than 100 s to reach a statistically steady state for starting fire. The spin-up time using the *PenaBlending method*, on the other hand is  $\sim 80 - 100$  s to start a fire simulation. For a *large domain*, the spin-up time for  $wind1$  is  $\sim 1000$  s,  $wind2$  is  $\sim 800$  s and  $wind3$  is less than 150 s. In this case, the spin-up time for the *PenaBlending method* is  $\sim 200$  s. It is observed that the spin-up time for the *PenaBlending method* is comparable to that of the *mean-forcing method* ( $wind3$  case). The sampling time for each simulation is summarised in Table-4.1.

The parameters used for running the simulations have been summarised in Table-3.3-a,b,c. The numerical parameters (Table-3.3-a) and the boundary conditions (Table-3.3-b) have been chosen for the scenarios simulated in the current work to obtain numerical results. These values can be varied as required. The fuel parameters (Table-3.3-c) have been obtained from the literature as cited in the table. The simulations using the *PenaBlending method* uses a *penalisationParameter* of 0.1 and *blendingParameter* of 1.0 for both the *small* and *large* domains. The *pSim* data is synthetic data generated using analytical methods, employing Matlab for testing purposes. All the other required parameters have been summarised in Table-4.2.

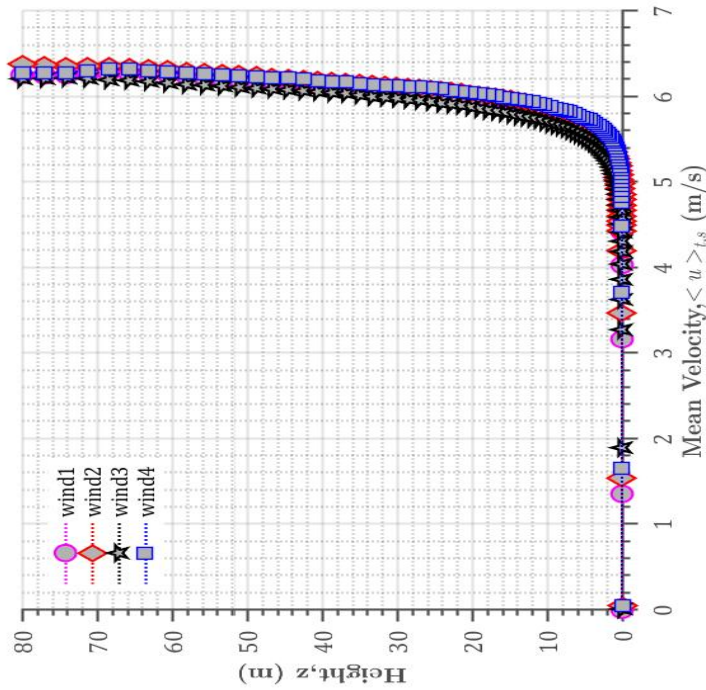
Table 4.2: Parameters used in the PenaBlending method for both large and small domain

Small Domain	Large Domain
<b>Domain Dimension:</b> 130m X 40m X 80m	<b>Domain Dimension:</b> 600m X 300m X 100m
<b>penalisationParameter :</b> 0.1	<b>penalisationParameter :</b> 0.1
<b>blendingParameter :</b> 1	<b>blendingParameter :</b> 1
<b>penXmin :</b> 0 (penalisation-inlet)	<b>penXmin :</b> 0 (penalisation-inlet)
6 (blending inlet)	10 (blending inlet)
113 (blending outlet)	580 (blending outlet)
119 (penalisation outlet)	590 (penalisation outlet)
<b>penXmax :</b> 6 (penalisation-inlet)	<b>penXmax :</b> 10 (penalisation-inlet)
12 (blending inlet)	20 (blending inlet)
119 (blending outlet)	590 (blending outlet)
125 (penalisation outlet)	600 (penalisation outlet)
<b>penYmin :</b> 0	<b>penYmin :</b> 0
<b>penYmax :</b> 40	<b>penYmax :</b> 300
<b>penZmin :</b> 0	<b>penZmin :</b> 0
<b>penZmax :</b> 80	<b>penZmax :</b> 100
<b>mX :</b> 0 (penalisation inlet and outlet)	<b>mX :</b> 0 (penalisation inlet and outlet)
-1 (blending inlet)	-1 (blending inlet)
1 (blending outlet)	1 (blending outlet)
<b>mY :</b> 0	<b>mY :</b> 0
<b>mZ :</b> 0	<b>mZ :</b> 0
<b>b :</b> 1 (penalisation inlet and outlet)	<b>b :</b> 1 (penalisation inlet and outlet)
0 (blending inlet and outlet)	0 (blending inlet and outlet)
<b>pena_I :</b> 6	<b>pena_I :</b> 5
<b>pena_J :</b> 40	<b>pena_J :</b> 150
<b>pens_K :</b> 80	<b>pens_K :</b> 100

Following the discussion in Section-4.2, the boundary layer for all the simulations performed is driven by either a log-law or a  $(1/7)^{th}$  power-law inlet profile. The wind profile in the domain should follow a realistic wind profile for the lower atmosphere (Wyngaard (2010)). In order to verify the profile obtained in the wind simulations, the mean velocity profiles are obtained, as shown in Figure-4.6, where the change in the u-velocity is plotted against the domain height, averaged in space and time. An average u-velocity at height 10 m, referred to as  $u_{10}$  has been maintained at  $\sim 5.5$  m/s for the *small domain* and  $\sim 5$  m/s for the *large domain*. In case of the *small domain* (Figure-4.6-a), these profiles are obtained at 50 m upstream of the inlet and averaged over the fire plot of  $40 \text{ m} \times 40 \text{ m}$ . For the *large domain* (figure(4.6-b)), these profiles are plotted at 300 m upstream of the inlet and averaged over the fire plot of  $100 \text{ m} \times 300 \text{ m}$ . It is observed that the velocity profile of the wind simulation using the *PenaBlending method(wind4)* reasonably collapse on those obtained from the reference cases, using the *traditional method (wind1, wind2, wind3)* with an expected wind profile for both the domains (Moinuddin et al. (2018)), hence verifying a correct implementation of the wind development using the *PenaBlending Method*. Figure-4.7 shows the semi-logarithmic plot of mean-velocity profiles for both the domains, to verify the robustness of the ABL simulation. These profiles are taken over the fire-plot, similar to the mean velocity profiles for both domains. It is observed that all the four profiles reasonably collapse on each-other from  $z = 10$  m onwards with some variations for both the domains and a logarithmic layer. The log-law plotted is observed to be parallel and fitting to the mean-velocity profiles. In the current study, only the  $u_{10}$  at the inlet has been tried to match to see the developed wind-field for comparison. It is observed from these profiles that the  $u_2$  velocity for different cases varies prominently for the *small domain*; whereas the variation is lesser for the *large domain*. The same wind field has been used to carry out the fire simulations which are discussed in Section-4.4.

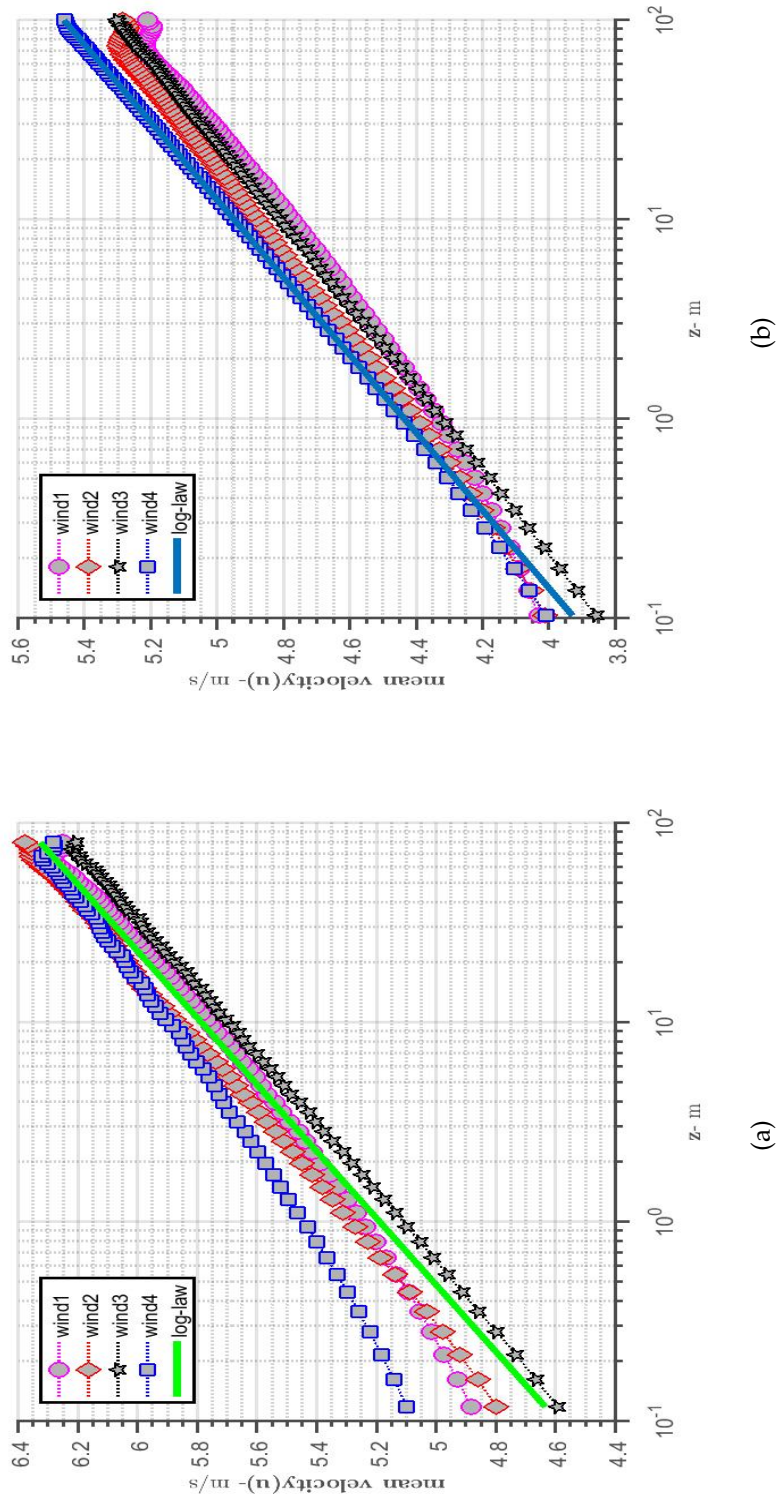


(a)



(b)

**Figure 4.6:** The mean velocity profiles are plotted over the fire-ground for wall-of-wind method (wind1), SEM method (wind2), mean-forcing method (wind3) and the PenaBlending method (wind4) for: (a) Small domain of 130m X 40m X 80m ; and (b) Large Domain of 600m X 300m X 100m.



**Figure 4.7:** The mean velocity profiles are plotted over the fire-ground for wall-of-wind method (wind1), SEM method (wind2), mean-forcing method (wind3) and the PenaBlending method (wind4) in semi-logarithmic scale for: (a) Small domain of 130m X 40m X 80m ; and (b) Large Domain of 600m X 300m X 100m. It is observed that the profiles converges with the theoretical log-law plot and shows a clear logarithmic layer.

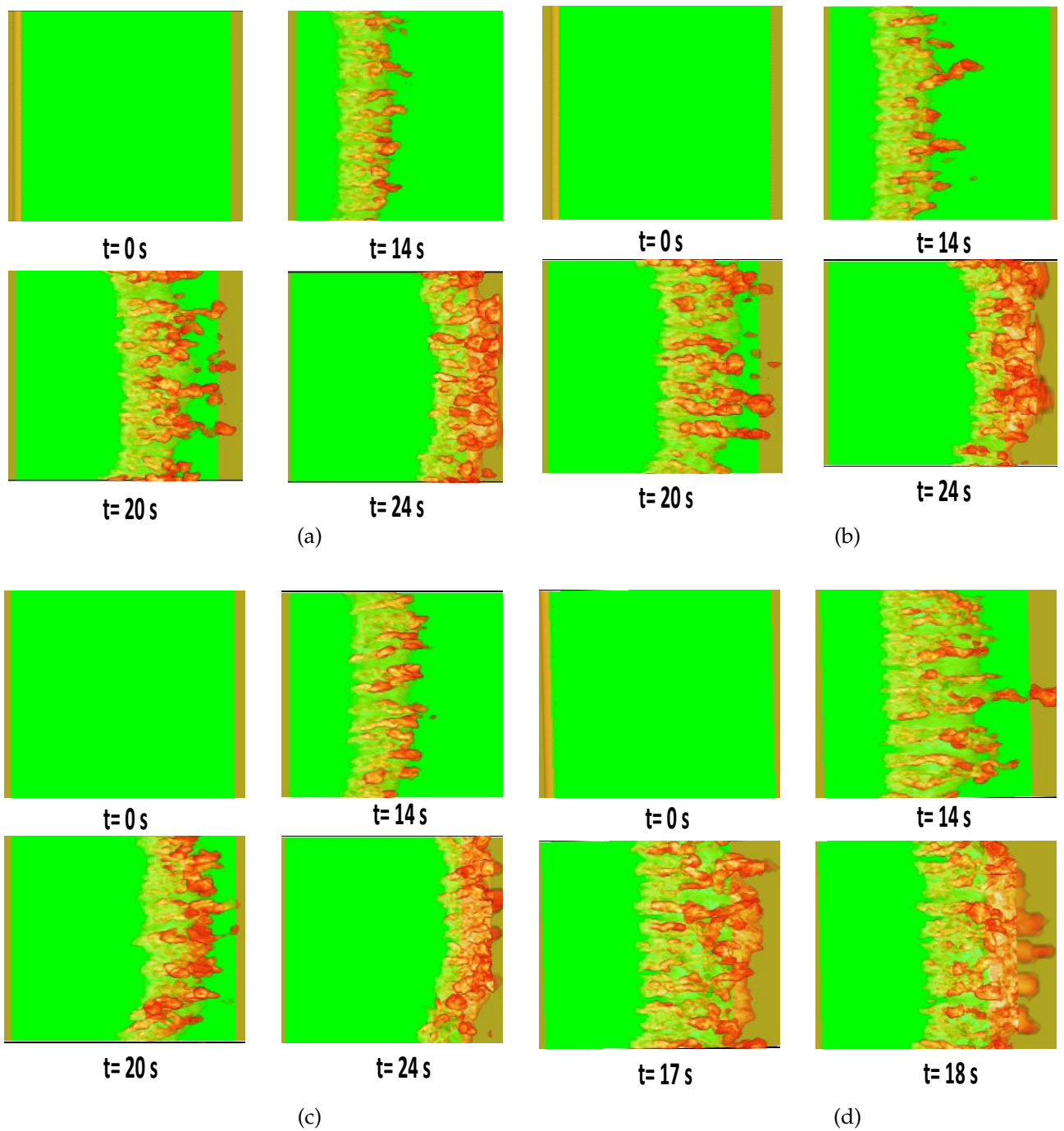
#### 4.4 Fire cases : results and discussions

After the implementation and the testing has been completed with the wind development over the simulation domain, the results can be used to start the fire simulation. An infinitely long fire is simulated across the width of the domain (along  $y$ ; Section-4.1) in all the cases; this minimises any variation along the  $y$ -direction of fire quantities such as RoS and flame front width. The initial and inlet conditions for all the fire cases are taken similarly to the wind cases. In order to obtain satisfactory results of the fire quantities, the fire is ignited after a statistically steady wind field is developed, which is judged from the wind-only simulations. The boundary fuel used is grass and the corresponding fuel properties used have been summarised in Table-3.3-c. The fire-plot consists of burnable-grass, which starts burning after the fire is ignited and burns out all the grass to reach the end of the fire-plot. Finer grid resolution of  $0.25 \text{ m} \times 0.25 \text{ m}$  is selected based on the grid convergence study of Moinuddin et al. (2018), upto a height of  $z = 6 \text{ m}$  for both the domains. The quantities like rate-of-spread are noisy because of turbulence in the fire flame. Therefore, to reduce the noise and to allow easier interpretation, a domain average RoS has been plotted for all the cases.

Figure-4.8 shows how the fire progresses across the fire-plot of  $40\text{m} \times 40\text{m}$  for the *small domain* for all the fire simulation cases. The fire propagation contour is taken at various time-steps to show the fire propagation with time. Four time-steps have been chosen to represent this. For the *small domain*, the fire locations have been taken at  $t = 0 \text{ s}$ ,  $t = 14 \text{ s}$ ,  $t = 20 \text{ s}$  and  $t = \text{time at which fire reaches the end of the plot}$ . For the *large domain*, the fire locations have been taken at  $t = 0 \text{ s}$ ,  $t = 15 \text{ s}$ ,  $t = 30 \text{ s}$  and  $t = \text{time at which fire reaches the end of the plot}$ . In both the domains, it is observed that the flame is wider for *fire4* case using the *PenaBlending method* as compared to the other three cases, and hence, the flame front reaches the end of the fire-plot much faster as compared to the other cases (Figures-4.8-d and 4.9-d). This case requires  $\sim 18 \text{ s}$  and  $\sim 38 \text{ s}$  to reach the end of the fire-plot for the *small* and *large* domains respectively. For the *large domain*, *fire3* also reaches faster (Figure-4.9-c) as compared to *fire1* and *fire2*, and takes  $\sim 45\text{s}$  to reach the end of fire-plot. For the rest of the cases, the fire requires  $\sim 24 \text{ s}$  and  $\sim 50 \text{ s}$  to reach the end of the fire-plot for the *small* and *large* domains respectively. All the fire simulations have been run using

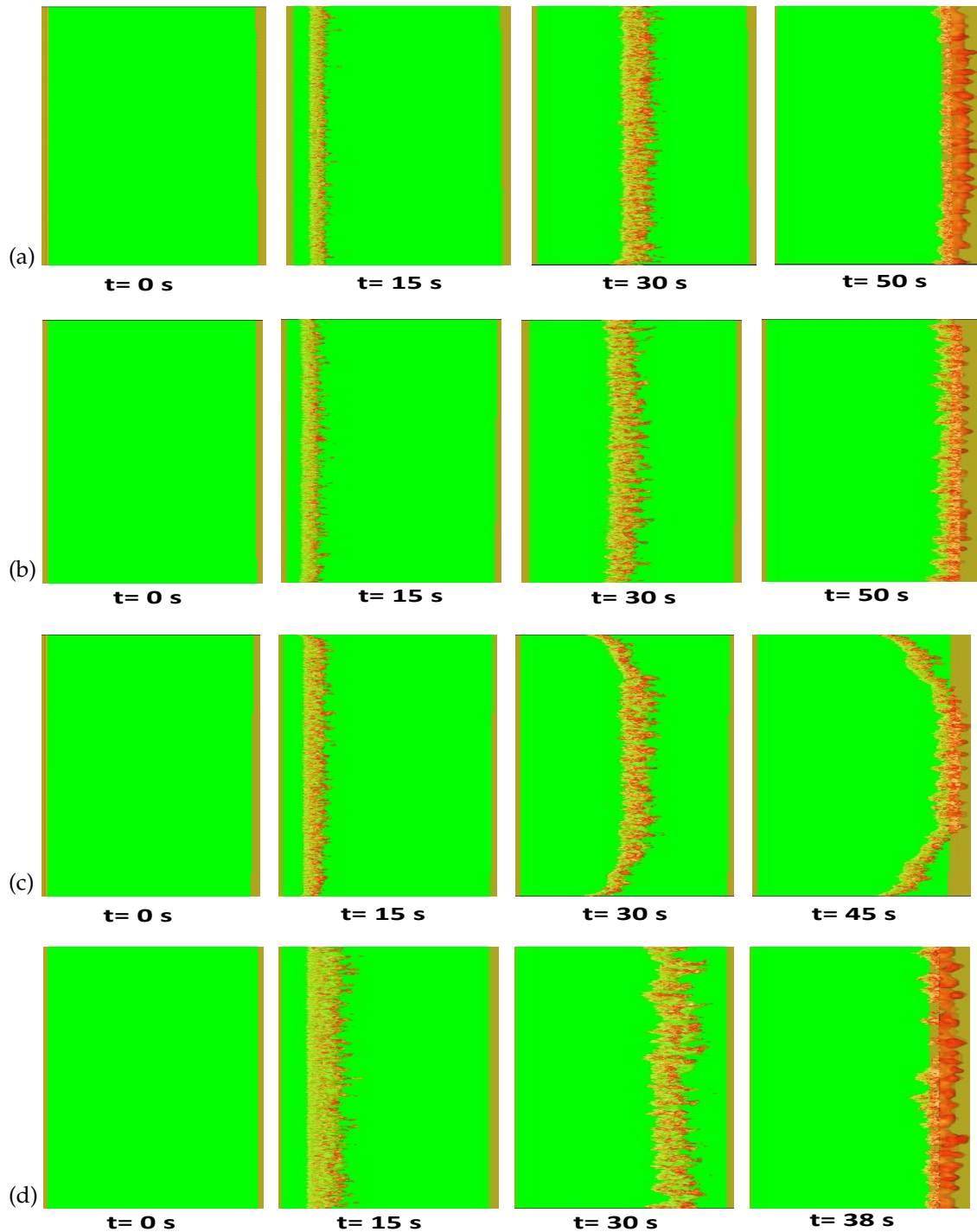
the same wind-field that is used in the wind simulation, where only  $u_{10}$  at the inlet has been matched. Figure-4.7 shows that the  $u_2$  shows considerable variations in the case of the *small* domain, whereas the *large* domain shows lesser variations. Because of this, the flame width varies in the fire cases, resulting in faster propagation for some cases. To test the actual fire propagation variation, ideally at the ignition line,  $u_2$ , (considering the mid-flame height) should be matched. This was done by [Moinuddin et al. \(2018\)](#) and such an approach can be the subject of future studies especially in relation to the *PenaBlending method*. In both cases, the ignition was started using an ignitor as a linefire across the width of the domain. This ignitor was on for 11 s. As the fire transitions from the burnable to non-burnable grass, poorly resolved burning continues downstream of the plot. Hence the non-zero HRR for a short distance downstream of the burnable area is seen in Figures-4.8 and 4.9.





**Figure 4.8:** The fire propagation contour for a small domain with a fire-plot of (40m X 40m) where the green area represents the 'burnable grass plot' and the non green area represents the 'non-burnable grassplot'. The propagation of fire is represented at various time-steps for (a) fire1; (b) fire2; (c) fire3; (d) fire4 cases. The first three cases requires almost equal times ( $\sim 24\text{s}$ ) to burn through the fire plot, whereas fire4 requires much lesser time ( $\sim 18\text{s}$ ).





**Figure 4.9:** The fire propagation contour for a large domain with fire plot of (100m X 300m) where the green area represents the 'burnable grass plot' and the non green area represents the 'non-burnable grassplot'. The propagation of fire is represented at various time-steps for (a)fire1; (b)fire2; (c)fire3; (d)fire4 cases. The first two cases requires almost equal times ( $\sim 50s$ ) to burn through the fire plot, whereas fire3 requires  $\sim 45s$  and fire4 requires  $\sim 38s$  to do so.

To compare the fire simulation using the *PenaBlending method*, various fire parameters can be used such as HRR, rate-of-spread (RoS), flame length, flame angle. As discussed in [Moinuddin et al. \(2018\)](#), Heat Release Rate (HRR) is a primary characteristic of fire, and is related to Bryam’s fire intensity. HRR can be considered fundamentally as the power of the fire. However, rate of spread of the fire (RoS) is often of primary interest to fire behaviour analysts who wish to predict the movement of a fire across the landscape ([Moinuddin et al. \(2018\)](#)). RoS depicts the fire-spread rate as a function of time. Hence, in the current study, these two parameters have been used for comparing the *PenaBlending method* for fire simulations against those using *traditional methods*.

As discussed in Section-2.1.5, the location of the fire-front ( $x_*$ ) can be determined by the the x-location of the temperature(T) which is greater than 400 K. At the back of the fire, the grass or fuel may not be fully converted to char and the ground temperature should be higher. Therefore, ( $x_*$ ) can be considered as a very good a good indication of the fire-front location. Figure-4.10-a,b represents the fire front location as a function of time over the fire-plot for the *small* and the *large* domains respectively. This parameter can be used to define rate-of-spread (RoS) of the fire in the current study. The RoS is the time-derivative, i.e.:

$$RoS = \frac{dx_*}{dt} \quad (4.1)$$

Figure-4.11 represents the boundary temperature plots which depict the fire width for both the domains. It is observed that the fire width (the pyrolysis region represented by yellow colour) for *fire4* using the *PenaBlending method* is large as compared to the other cases for both the domains. This results in faster burning of the fuel and the fire-front reaches the end of the fire-plot quicker. This can be due to the variation in the  $u_2$  in case of *fire4* as compared to other cases, as previously discussed. It can be observed that the  $u_2$  velocity variations for a *small* domain are greater than those of the *large* domain for *fire4* case. Hence, the fire-width for *fire4* in case of the *small* domain is larger compared to that of the *large* domain. The boundary conditions used can also contribute to these variations. This also results in high HRR and RoS of *fire4* simulation case which is discussed later.

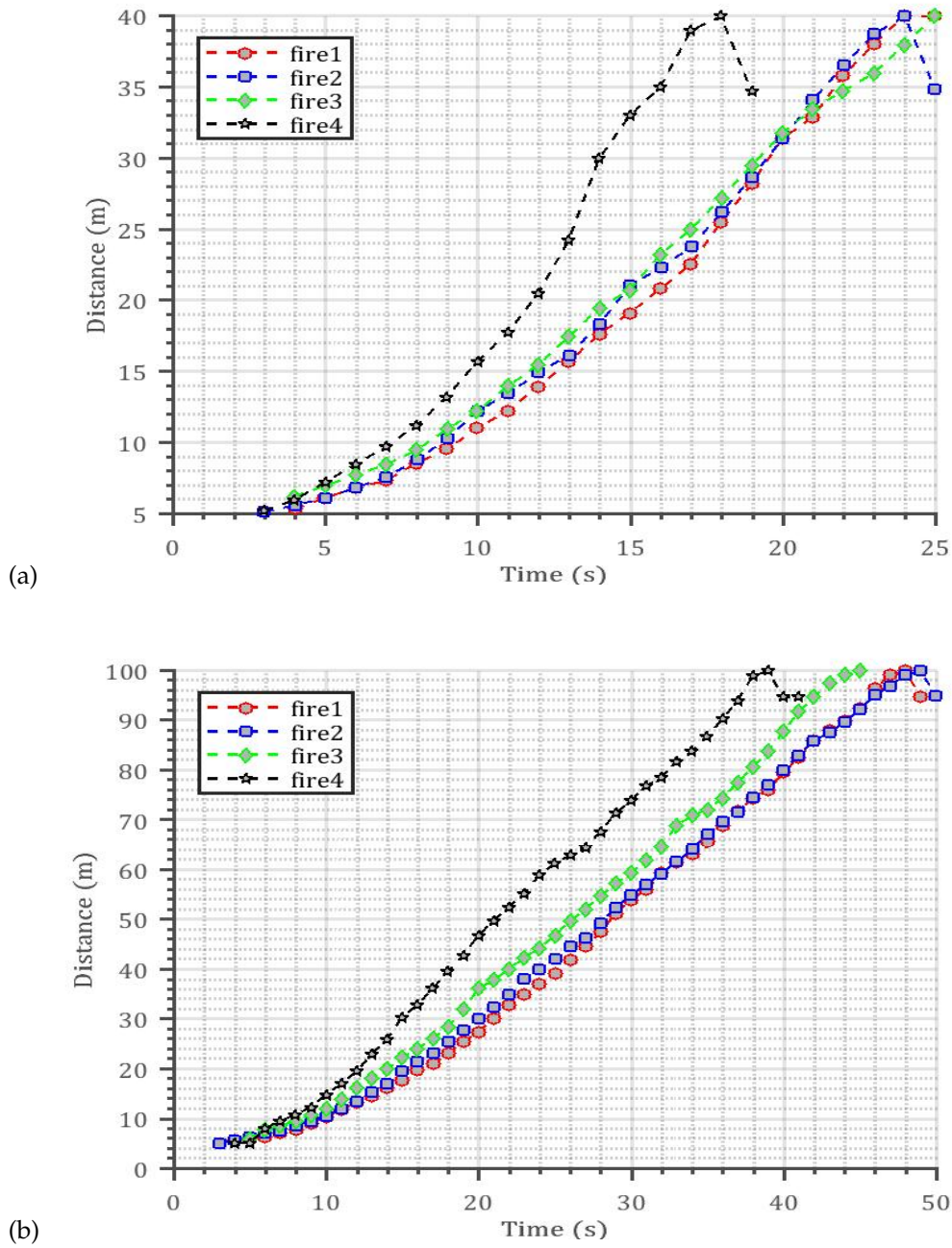
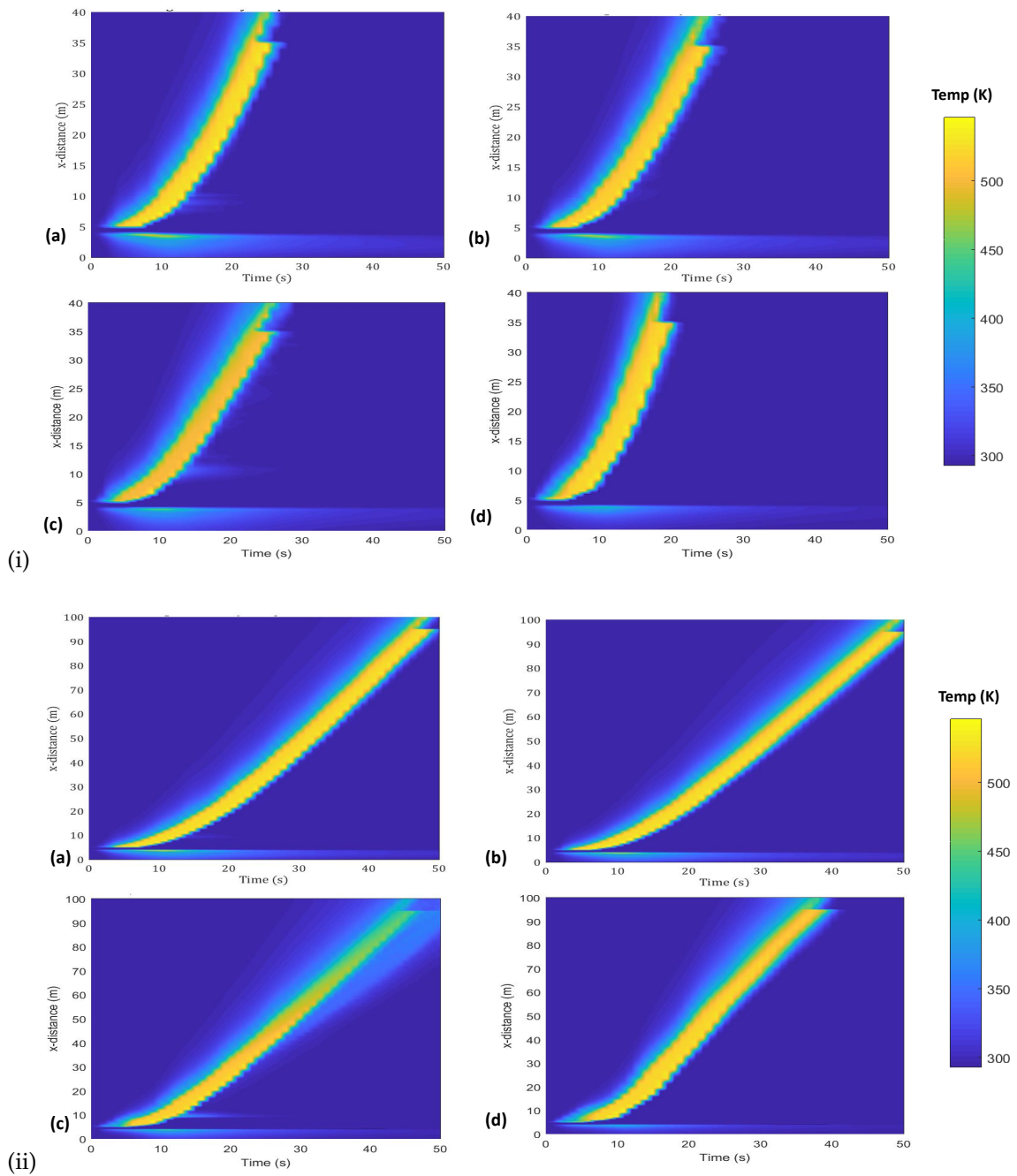
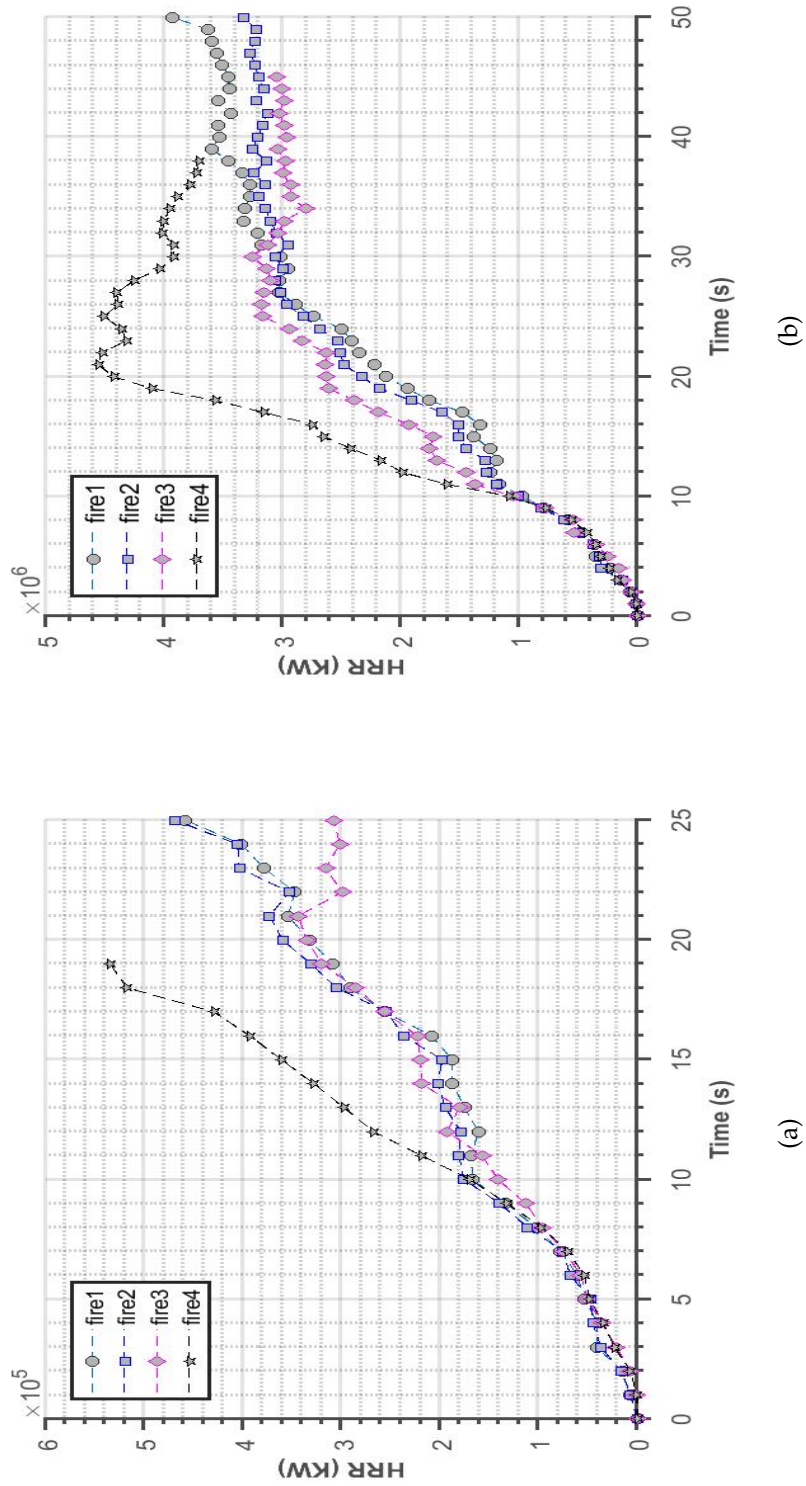


Figure 4.10: The fire front ( $x_*$ ) location in four fire cases (fire1; fire2; fire3; fire4) as a function of time for (a) small domain and (b) large domain, over the fire-plot.



**Figure 4.11:** The boundary temperature contours showing the fire-front propagation over the fire plot for all the fire simulation cases:(a) fire1; (b) fire2; (c) fire3; (d) fire4 for (i) small domain and (ii)large domain. The legend shows the temperature variation in K. The pyrolysis region is obtained when temperature becomes greater than 400K, which is represented by the yellow contours.

Figure-4.12 depicts the HRR obtained for all the fire cases for both the *large* and the *small* domain. Considering the *small domain* (Figure-4.12-a, the HRR for *fire1*, *fire2* and *fire3* have almost similar pattern and takes almost  $\sim 24$  s to travel to the end of the fire plot. The *fire4* case is observed to have a very high HRR and takes only  $\sim 18$  s, to reach the end of the fire plot. Considering the *large domain* cases (Figure-4.12-b, the HRR for *fire1*, *fire2* and *fire3* cases follows a similar pattern with minimum variations. *fire4* is observed to have a very high HRR and reaches the end of fire-plot in  $\sim 38$  s, which is much faster when compared to the other cases.



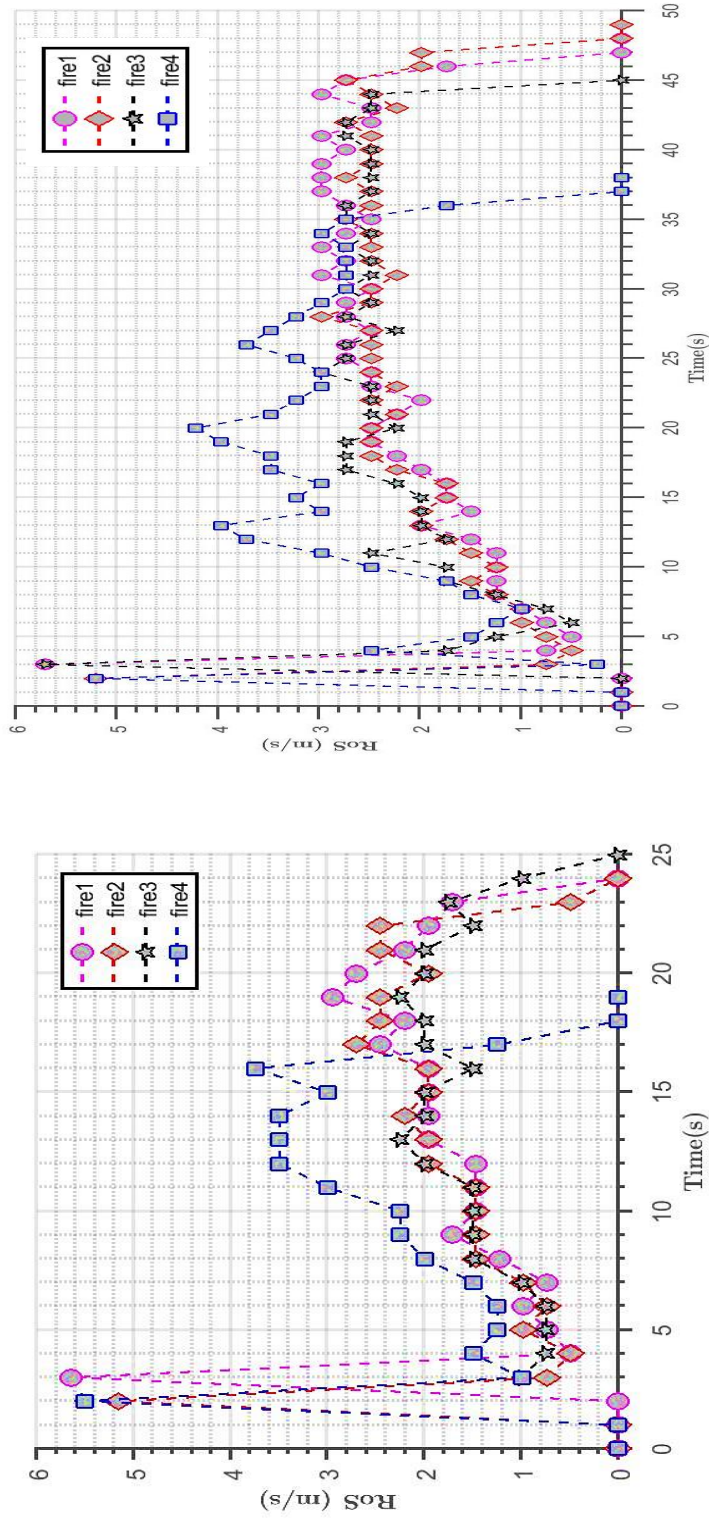
**Figure 4.12:** The Heat Release Rates (HRR) as a function of time for the fire simulations using wall-of-wind method (fire1), SEM method (fire2), mean-forcing method (fire3) and the PenaBlending method (fire4) for: (a) Small domain of 130m X 40m X 80m ; and (b) Large domain of 600m X 300m X 100m.



The Figure-4.13 shows the RoS comparisons for all the fire cases for both *small* and *large* domains. It is observed that for the *small domain*, all the three fire cases (*fire1*, *fire2*, *fire3*) reach the end of fire-plot by  $\sim 24$  seconds. Near the start of the fire, the RoS is maximum for all these cases and then reaches a quasi-steady state of  $\sim 2 - 2.5$  m/s before the fire reaches the end of the fire plot. In the case of *fire4*, the RoS increases initially and then reaches a quasi-steady state of  $\sim 3 - 3.5$  m/s and then reaches the fire-plot end within  $\sim 18$  seconds. Similarly, considering the *large domain*, it is observed that *fire1* and *fire2* require  $\sim 50$  seconds to burn all the fuel, whereas *fire3* requires  $\sim 45$  seconds to complete burning of the fire-plot. All these three cases reach a quasi-steady state of  $\sim 2.5 - 3$  m/s. On the other hand, the *PeaBlending method* (*fire4*) finishes burning in  $\sim 37$  seconds and acquires a quasi-steady state of  $\sim 3 - 3.5$  m/s before completely burning the fire-plot. These times are mentioned with approximate values as there may be some fluctuation of these time values when the same simulation is run on different computers with varying parameters such as number of nodes, CPUs per nodes and speed of each node. Moinuddin et al. (2018) discuss the fact that a minor difference in wind speed and direction can have a considerable effect in the simulation results. Figure-4.7 shows that there are some differences in the mean velocity fields for all the cases. Therefore it can be concluded that the differences introduced in the wind fields by the different inlet conditions leads to the variation in the RoS and HRR for all the fire simulation cases. Table-4.3 gives an overview of time for the fire flame to reach the end of the fire plot for all the fire cases for both the domains.

**Table 4.3:** Time for the flame to reach the end of fire-plot for small and large domains

Small Domain	Large Domain
<b>fire1</b> : $\sim 24$ s	<b>fire1</b> : $\sim 50$ s
<b>fire2</b> : $\sim 24$ s	<b>fire2</b> : $\sim 50$ s
<b>fire3</b> : $\sim 24$ s	<b>fire3</b> : $\sim 45$ s
<b>fire4</b> : $\sim 18$ s	<b>fire4</b> : $\sim 38$ s



(a)

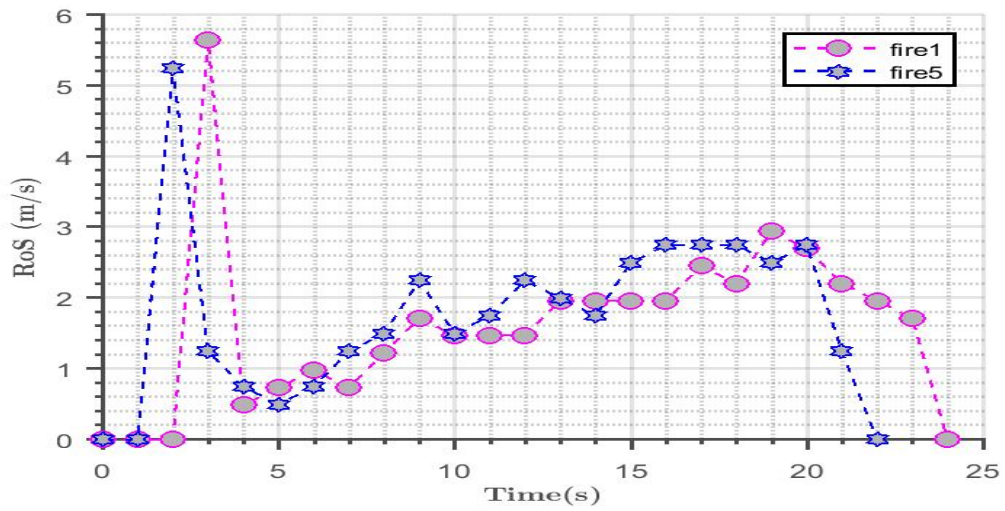
(b)

**Figure 4.13:** The rate-of-spread of fire as a function of time using wall-of-wind method (fire1), SEM method (fire2), mean-forcing method (fire3) and the PenaBlending method (fire4) for: (a) Small domain of 130m X 40m X 80m ; and (b) Large Domain of 600m X 300m X 100m.

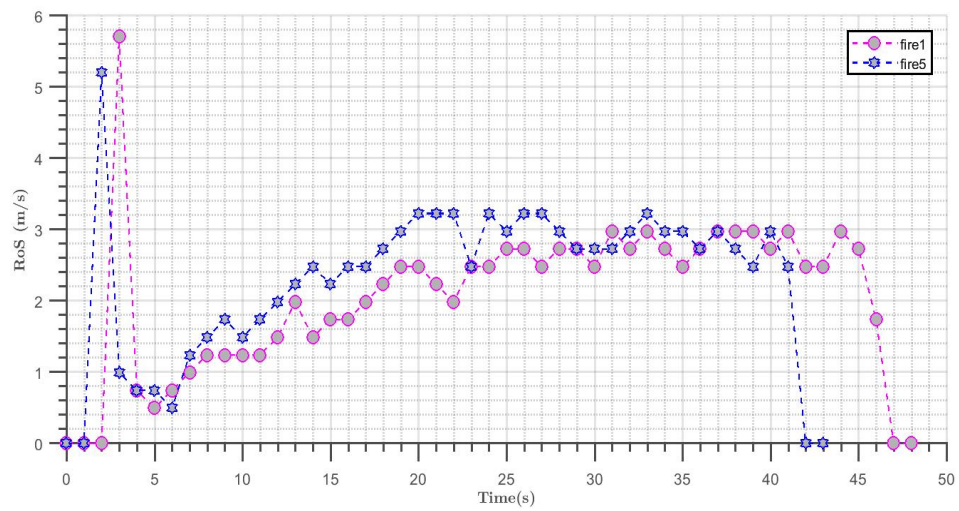


## 4.5 Effect of under-developed wind field on fire simulation

It has been previously discussed that a statistically stable and developed wind-profile is required for starting any fire simulation in order to get non-distorted fire parameters. But, what would be the consequences of an under-developed wind field at the start of the fire? This section will discuss using underdeveloped wind fields and their effect on fire propagation simulations. In this case also, one scenario each for the *small domain* and the *large domain* has been considered. An under-developed wind field is defined to be a wind field that is still developing through the domain and has not reached a statistically steady state for starting a fire. The burnable grass-plot for these cases is set near the inlet, so that a minimum up-stream of the fire plot is allowed, and the wind is not allowed to develop over the space before the fire simulation starts. For both the domain sizes, the fire-plot is set at 25 m from the inlet and all other conditions are set similar to *fire1* case. This case is depicted as *fire5* case in Table-3.2. For the *small domain*, the fire is ignited at 1 s, immediately after the start of the simulation, whereas for the *large domain* the fire is ignited after 100 s of starting the simulation. Figures-4.14-a,b represent the RoS of the underdeveloped-fire (*fire5*) with that of the wall-of-wind method (*fire1*) for the *small* and the *large* domains respectively. In both the cases, it is observed that the *fire5* is stopping before that of the *fire1* case. For the *small domain*, the underdeveloped wind field gives a fire which burns only 1 s shorter than that of *fire1*. It can be argued that the non-burnable grass plot leading to the ignition line is so short and the  $u_{10}$  velocity is so high that by 1 s, a non-zero wind develops over the fire plot. In the *large domain* case, it is observed that the RoS declines rapidly to zero almost 5 s before that of *fire1*, which is a significant amount (10% difference). Interestingly, the most prominent differences are before 25 seconds. However, the fires appear to converge to a similar RoS after that time. It can be argued that the wind-field is not completely developed in such a small upstream of 25m, and hence there is a considerable amount of decrease in RoS for an underdeveloped wind profile.



(a)

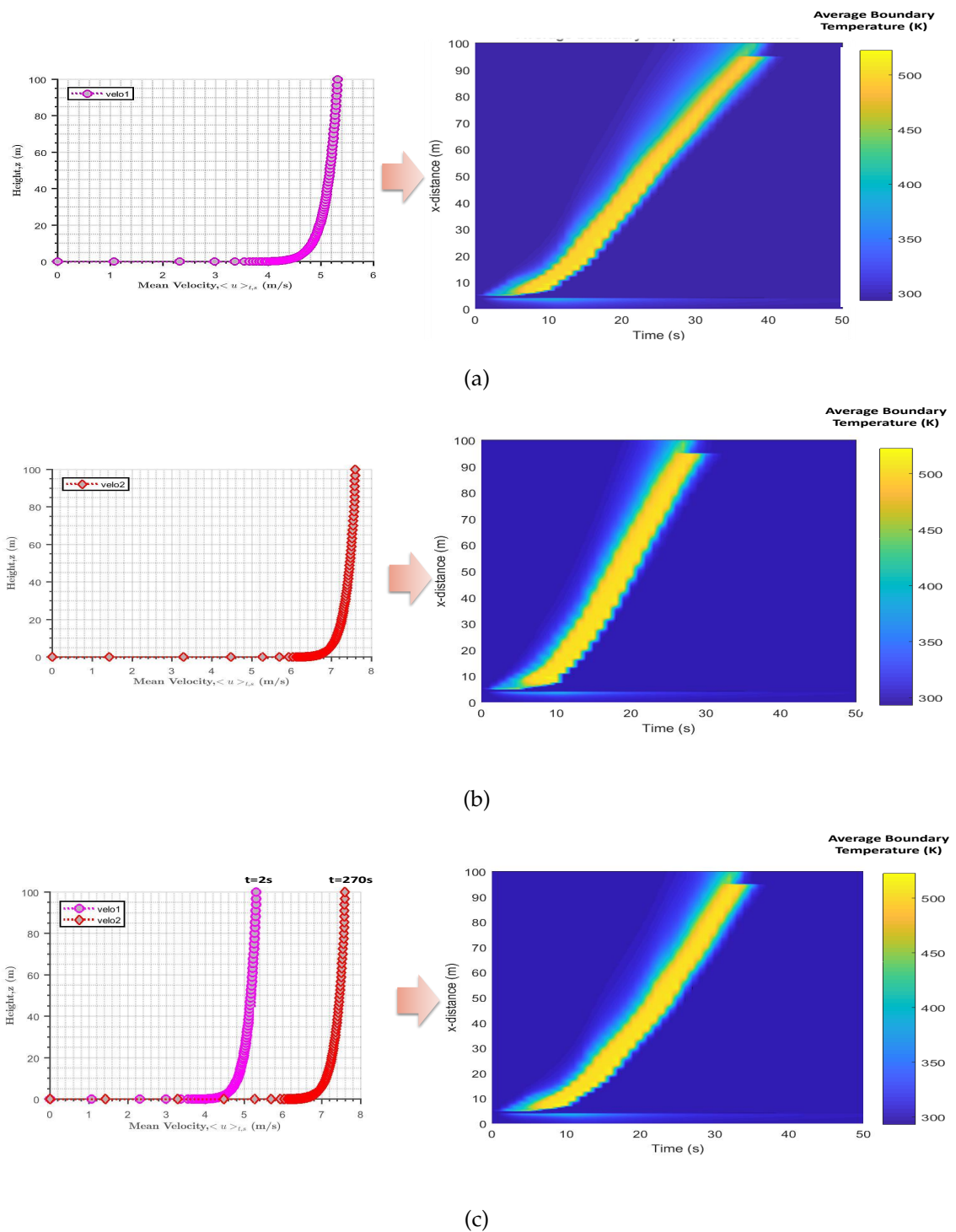


(b)

**Figure 4.14:** The RoS comparison for underdeveloped-wind field for fire simulations (fire5) compared with the wall-of-wind method (fire1) for: (a) small domain and (b) large domain. In both the cases, all the conditions parameters used, boundary conditions and inlet method are same

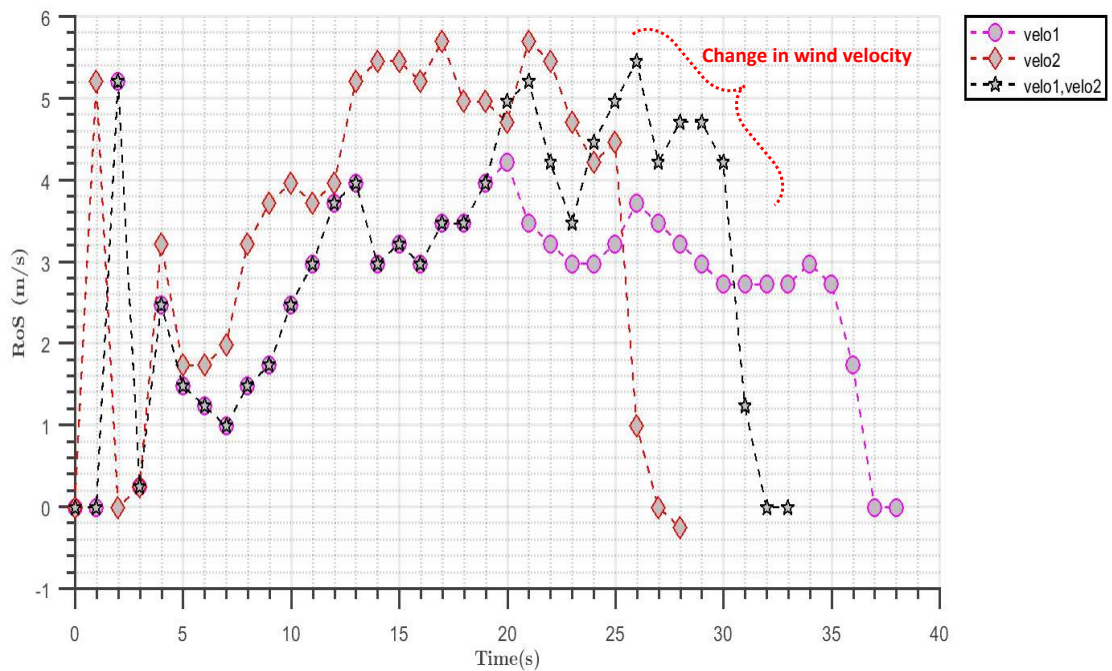
## 4.6 Gusting effect of wind on fire

The *PenaBlending Method* provides the ability for FDS to read different velocity fields at various time-steps from any external model or data (called *pSim* data). This section validates this ability with three sets of simulations. To observe the effect of gusting winds on a fire simulation properly, a *large domain* has been considered with domain configuration, fuel parameters and boundary conditions identical to *fire4* case, discussed previously. Two different velocity fields have been considered in this case, one with  $u_{10} \sim 4.8$  m/s denoted by *velo1* and the other with  $u_{10} \sim 7$  m/s denoted by *velo2*. The fire is ignited at 250 s after the start of the simulation for all three cases. For the first case, *velo1* is read as *pSim* data at 2 s after the start of the simulation. For the second case, *velo2* is read at 2 s after the start of the simulation. For the third case, *velo1* is read at 2 s after the start of the simulation. The simulation progresses and the fire starts at 250 s. As the fire progresses towards the middle of the fire-plot, a gust of *velo2* is introduced at  $\sim 270$  s (20s after the start of the ignition) and is used for completing the rest of the burning process. Figure-4.15 shows the velocity profiles used and their corresponding contours of average boundary temperatures. Figure-4.15-a depicts that it takes  $\sim 38$  s for the fire to propagate over the burnable grass-plot, burning all the fuel. On other hand, Figure-4.15-b depicts that with *velo2*, which is faster than *velo1*, it takes  $\sim 28$  s for the fire to travel over the fire plot. Figure-4.15-c represents the gusting effect of wind on fire propagation and is observed to have taken  $\sim 32$  s to propagate over the fire-plot.



**Figure 4.15:** The average boundary temperature contours: (a) *velo1* with  $u_{10} = \sim 4.8$  m/s is read at 2 s after the simulation starts ; (b) *velo2* with  $u_{10} = \sim 7$  m/s is read at 2 s after the start of simulation ; (c) This figure shows the average boundary temperature contour when a gust of wind is applied in the middle of fire simulation. *velo2* is read at 20 s after the start of ignition, in the middle of burning, with *velo1* read initially at 2 s after the start of simulation

The RoS has also been compared among all the three cases, which provides a clear insight of how the fire is propagating and how the gust of wind is affecting the fire. This is depicted in Figure-4.16. From the figure, it is observed that *velo1,velo2* plots collapse completely over *velo1* plot upto  $\sim 19$  s since the fire ignition (269 s of simulation time), as *velo1* was used as initial velocity for the third case. At the 20<sup>th</sup> second, *velo2* is introduced. Therefore, it is observed that there is a sudden increase in the RoS, which is highlighted with a 'red bracket', and then falling back to 0 as the fire consumes the fuel over the fire plot faster. It is interesting to note that the fire response is almost instantaneous to the change in the wind velocity. It can be concluded that the gusting wind takes a time in between that of *velo1* and *velo2* for propagating the fire over the burnable fire-plot. This proves the ability of the *PenaBlending method* to model the effect of gusting winds on fire, by allowing various velocity profiles as various time-steps, as per requirement.



**Figure 4.16:** Rate-of-spread comparison of fire simulations with velocity1(*velo1*), velocity2(*velo2*) and gusting effect of *velo1* and *velo2*. The area marked with a red semi-circle shows how an increase in velocity during a fire propagation changes the RoS and hence blows the plume out of the domain faster.

## 4.7 Modelling wind using reduced wind model to use with *Pen-aBlending method*

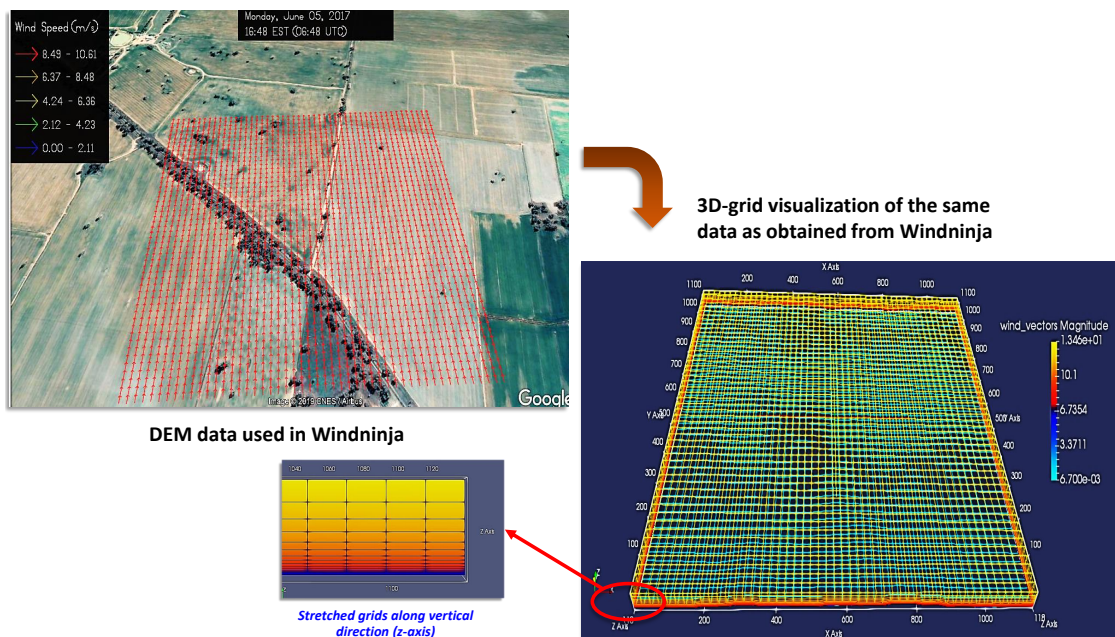
Windninja is a simple diagnostic wind model which has been developed and maintained by the USFS Missoula Fire Sciences Laboratory (Forthofer et al. (2014b)). It applies all required physics, including the conservation of mass and momentum, to account for terrain and temperature effects on an initial flow field obtained from a point measurement or a coarse scale prognostic weather model. The computational requirement for Windninja is much lower than prognostic models. Moreover, this wind model has the ability to simulate terrain and temperature modified wind at less than 50-m scales, which can be beneficial for fire management. This property of Windninja can be utilized to reduce the spin-up time for physics-based modelling. Windninja has two solvers: *conservation of mass solver* and *conservation of mass and momentum solver*. The conservation of mass solver is the fast-running solver; whereas, the conservation of mass and momentum solver is a new solver introduced in Windninja with limited features based on OpenFOAM toolkit (<http://openfoam.org>). All the technical information about these solvers can be found in Forthofer (2007), Forthofer et al. (2014b), Forthofer et al. (2014a). In the current study, the conservation of mass solver has been used which is discussed below in Section-4.7.

### Conservation of mass solver

The mass conserving model of Windninja conserves the mass while mathematically minimizing the change from an initial wind field with an imposed boundary condition. As discussed in Forthofer (2007), the only physics incorporated in this type of model is the conservation of mass (Chan and Sugiyama (1997), Montero et al. (1998), Ross et al. (1988), Sherman (1978)). Other effects including momentum or density driven flow, turbulence are partially accounted for if this information is present in the initial wind field. The model runs very fast as its approximation of governing equations is much simpler. The conservation of mass simulations usually gives less accurate results during stronger winds in the lee sides of the mountains and ridges where recirculation eddies may occur.



These can be accounted for better using the conservation of momentum equation, which is not included in this solver. Although this solver produces large errors on the lee side of the mountains and ridges, it can successfully capture the overall trend in wind speed. This solver captures best results in the upstream and top of the mountains. The conservation of mass solver can quickly compute the wind fields in seconds to a few minutes when run on a typical laptop computer using one CPU. The governing equations and other models included in Windninja are discussed in details in [Forthofer \(2007\)](#).

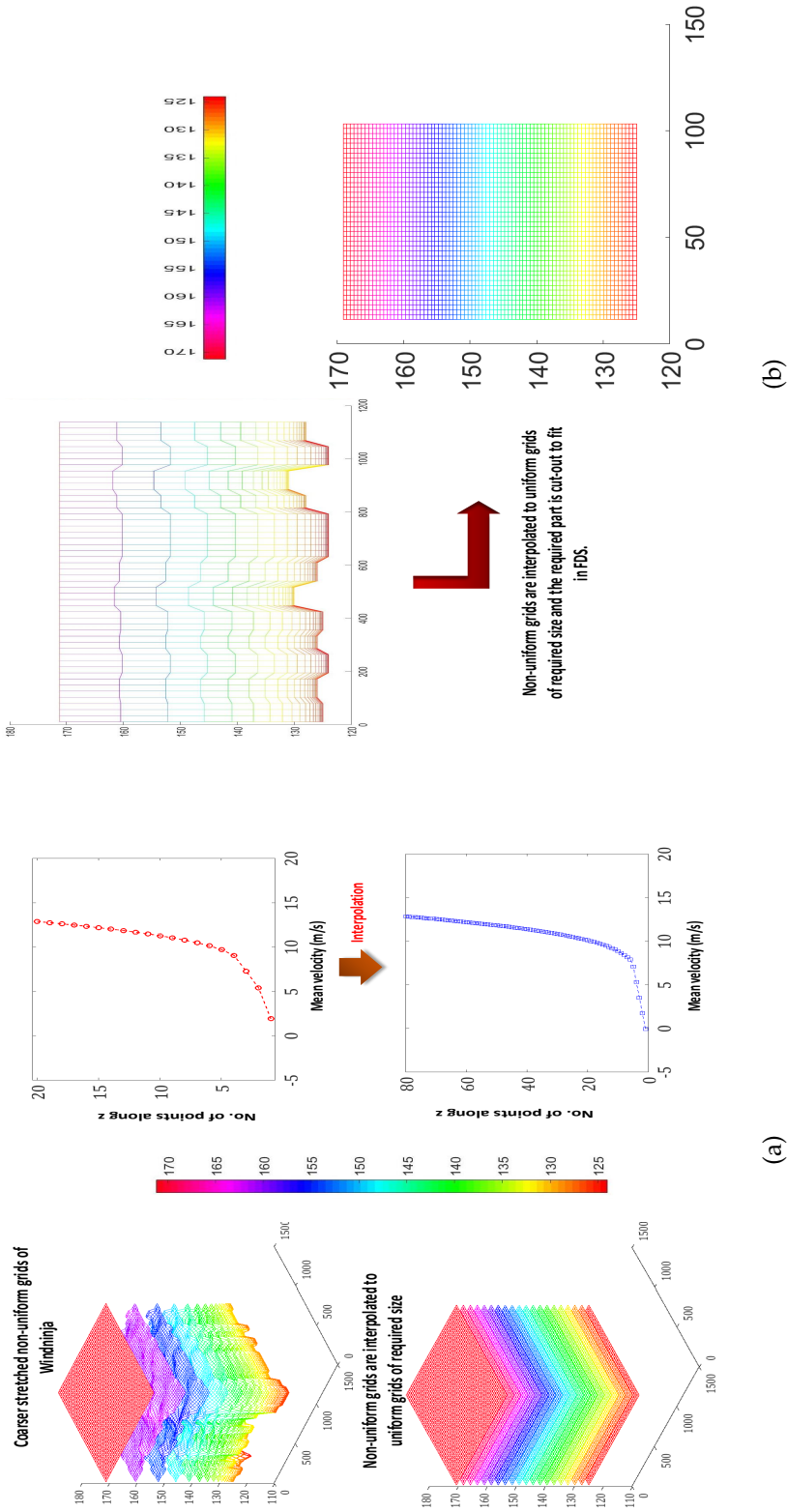


**Figure 4.17:** The 3D grid representation of the simulated wind field in Windninja. The grid generated is parallel to the underlying terrain.

Windninja documentation includes a number of tutorials. Wind ninja only requires a small number of user inputs (including wind height, input wind speed, direction, time, DEM file, mesh resolution and vegetation type). The first tutorial ([Tutorial1](#)) instructs the user through a step-by-step process of using Windninja. It also provides a sample DEM file to test, which complies with the Windninja requirements for practicing. This tool is specifically designed for simulating the terrain and temperature effects on the wind flow. A small number of user inputs are required for this model (as discussed in [Tutorial1](#)). For the current study, this tool has been run for a flat terrain type for obtaining the re-

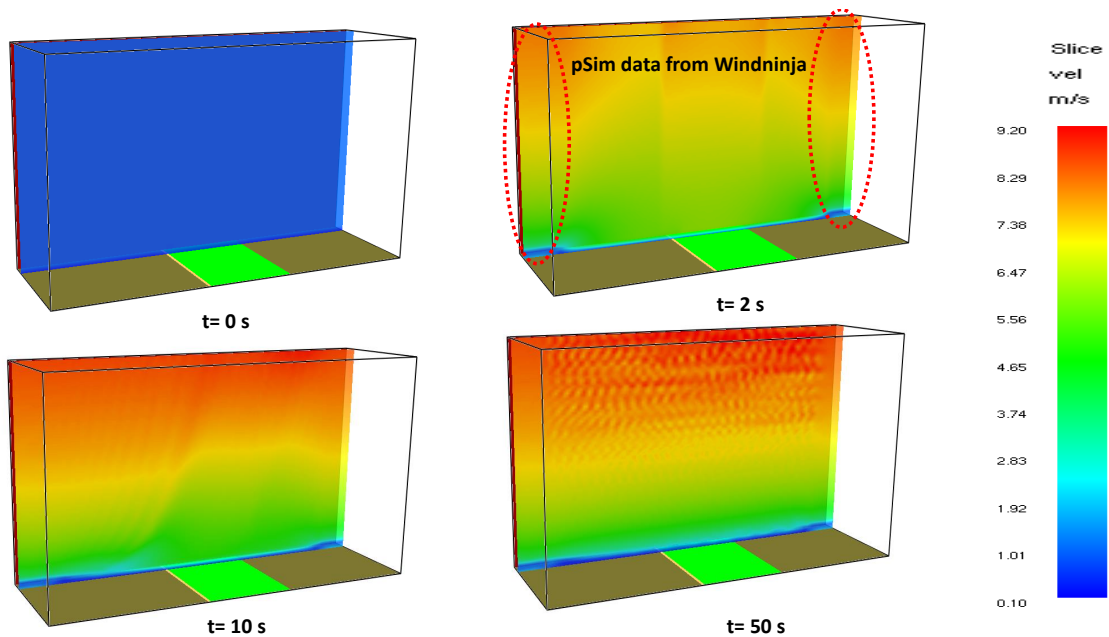
duced wind for *pSim* data. The modelling domain considered is a DEM (Digital Elevation Model) of an area of  $1.17 \text{ km} \times 1.17 \text{ km}$  with latitude and longitude of  $35^\circ 45'$  South and  $146^\circ 6'$  East, near the northern boundary of Melbourne, Australia. An average speed of  $10 \text{ m/s}$  has been considered as a domain average input speed. When running Windninja with required parameters, the simulated is reduced to 3D wind data. The simulation was completed in *4.52 seconds*. The 3D wind data obtained is shown in Figure-4.17. The simulated wind data provides vertically stretched grids. This means, the vertical dimensions of the cells increase with height above the ground. Since the domain considered is a flat land with minimum terrain perturbation, the wind velocity at a certain height remains almost constant. The wind velocity increases with an increase in height until it reaches the maximum domain height where there is free flow of air with a maximum wind speed. The horizontal resolution considered in this scenario is  $23 \text{ m}$ , which is very coarse as compared to physics-based modelling. This data needs to be converted to a required finer grid resolution similar to the FDS domain to be used as *pSim* data. This can be done by the method of interpolation. The stretched non-uniform coarser grids of Windninja data are interpolated into the required uniform fire grids (similar to *wind4* case for *small domain*). The initial and final grids as well as the corresponding wind profiles are depicted in Figure-4.18-a. The area considered for Windninja is very big ( $1.17 \text{ km} \times 1.17 \text{ km}$ ) as compared to that of the required penalisation and blending regions for FDS. Hence, a portion of the Windninja data has been cut out as required to be used as *pSim* data for FDS simulation. This has been depicted in Figure-4.18-b.





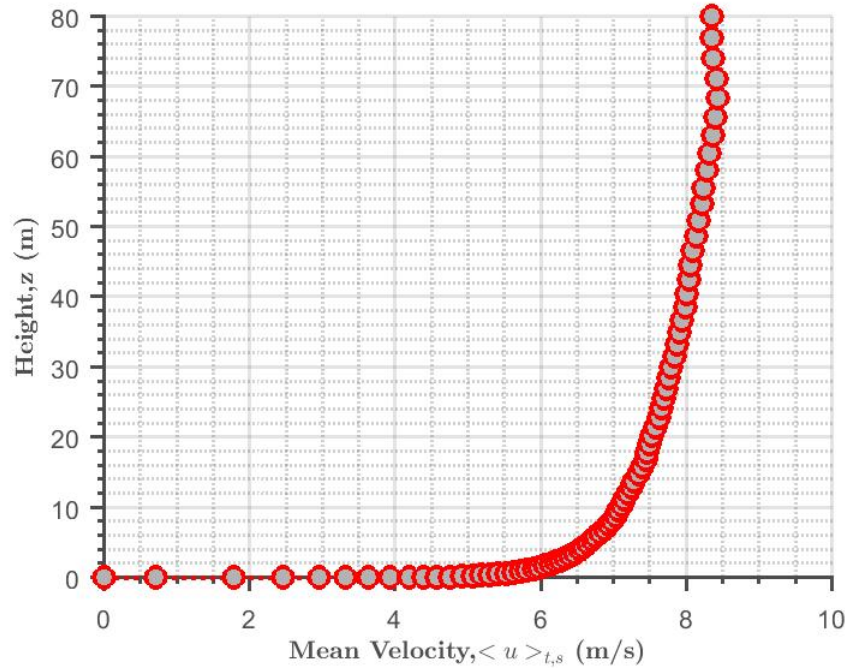
**Figure 4.18:** (a) The original and interpolated uniform grids with required resolution similar to FDS domain. The mean velocity profiles for the original wind data and the interpolated data. The y-axis in the velocity profiles represents the grid points along z-axis; 20 grid points along z-axis have been interpolated to 80 grid points as per the requirement, keeping the vertical distance constant; (b) The Y-Z plane showing the original data from Windinija with coarser non-uniform grid resolution and the cut-out uniform grid Windinija data with fine resolution of  $1m \times 1m$  used as pSim data for FDS.

This interpolated data is used as *pSim* data to develop a statistically steady wind field. In this case, the *small domain* has been considered with similar configuration and properties as *wind4* case. The parameters used for the *PenaBlending method* in this case is the same as that in Table-4.2. The wind is developed over the FDS domain using the Windninja data as shown in Figure-4.19.



**Figure 4.19:** Development of wind profile over the domain using Windninja data over various time-steps: (a) at time=0s; (b) at time= 2s, when the Windninja data is read at inlet/outlet; (c) at time=10s, depicting the wind developing from the inlet; (d) at time=50s, depicting a fully developed wind profile obtained.

It is observed that a fully developed wind profile is obtained as quickly as  $\sim 50$  s from the start of the simulation. The average wind profile over the fire plot is given by Figure-4.20, which resembles an atmospheric boundary layer, as expected.



*Figure 4.20: Mean velocity profile over the fire-plot using Windninja data as pSim data for Pen-aBlending method.*

After a statistically stable wind profile is obtained on running the simulation with Windninja data, a fire was ignited similar to the previous cases at  $\sim 200$  s after the start of the simulation. It is observed from the velocity profile, the  $\langle u \rangle_{10} = \sim 7$  m/s, and the fire head progresses faster over the fire plot as shown in Figure-4.21. It is observed that the fire reaches the end of the fire plot by  $\sim 18$  s. The corresponding RoS and the boundary temperature contour profiles can be given in Figure-4.22. The RoS is seen to achieve a quasi-steady state at  $\sim 3.5$  m/s and reaches the end of the fire-plot by  $\sim 18$  s.

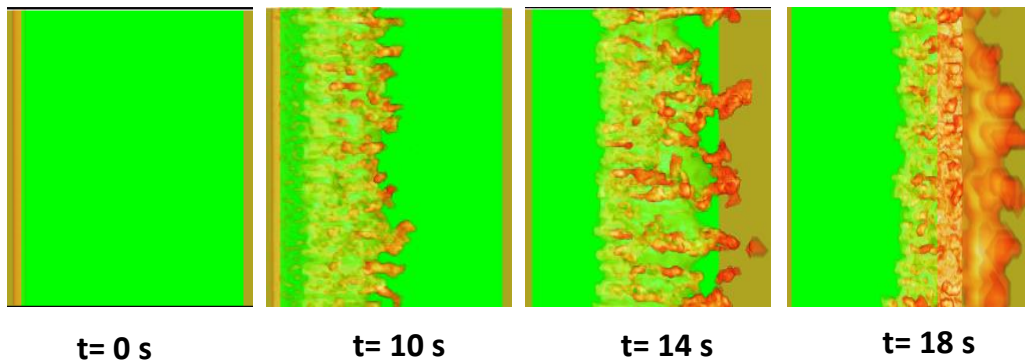


Figure 4.21: The fire propagation over the fire plot using Windninja data at: time=0s; time=10s; time=14s; time=18s.

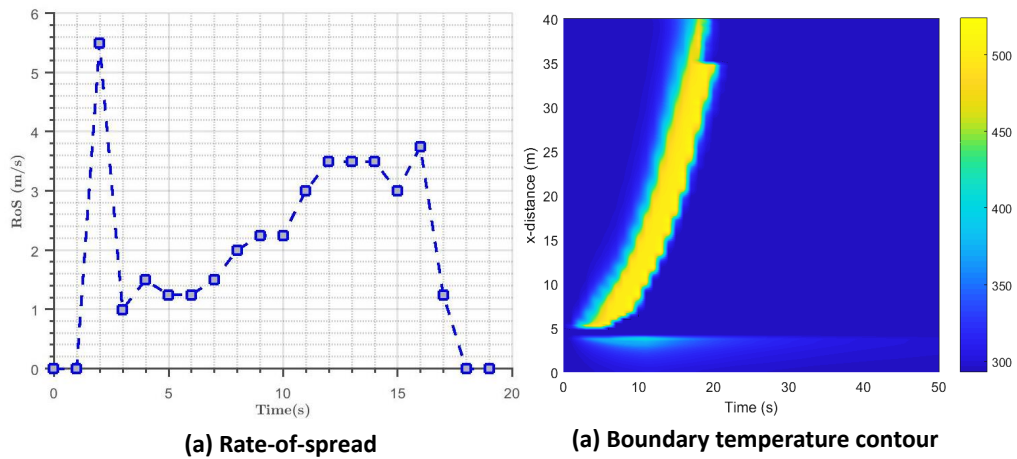


Figure 4.22: (a) The RoS profile for fire over the fire plot using Windninja data; (b) The boundary temperature contour showing the fire front propagation over the fire plot

In summary, this section includes a small case study to show that besides the synthetic data generated using Matlab, terrain modified wind data obtained from reduced wind models like Windninja can also be used as inlet conditions in FDS using the *PenaBlending* method. A corresponding fire simulation using Windninja data has also been done to show the results follows similar trend as the cases discussed before and hence verifying the potentiality of the *PenaBlending* method.

# Chapter 5

## Conclusion and Future Directions

The most important contribution of this study is implementing the *PenaBlending method* to reduce the spin-up time for physics-based fire modelling (FDS). From Table-4.1, it is seen that the *PenaBlending method* is  $\sim 80 - 85\%$  faster than the wind1 and wind2 for the smaller domain and  $\sim 75 - 80\%$  faster than wind1 and wind2 for the larger domain. It does not show any notable improvement in time when compared to wind3 in both domain cases. The accuracy of mean-velocity profiles obtained by using the *PenaBlending method* is good, with slight variations when compared to the traditional wind generation methods (as shown in Table-4.1). The gusting effect of wind on fire is also tested using this method in Section-4.6. The secondary contribution of this study is to bring back the combustion model of FDS 6.2.0 as an alternative option in FDS 6.6.0, to reduce the computational cost and obtain grid converged results for field-scale fire simulations. FDS 6.6.0 was chosen for this study to investigate its Monin-Obukhov (**mean-forcing method**) option.

Wildland fires form an intrinsic part of the Australian, as well as many other countries' emergency events that result in loss of life and property. The damage caused by such fires increases the need to understand fires in order to predict and manage the risk. Several numerical modelling techniques have been developed. Among those, physics-based models are currently promising models to simulate actual wildfires in future. Currently, only idealized wildfires are being modelled, which agree closely with the observed behavior and are predominantly used for research purposes. One such model, FDS has

been used for modelling wind and fires. A flat terrain and uniform fuel of grass has been used in all the simulations in the current study. One of the major shortcomings of FDS is that it is computationally expensive. This study investigates this shortcoming of FDS and tries to improve it by introducing a new initial flow method which is termed as the *PenaBlending method*. This is fast and has some additional capabilities. A detailed code implementation of the *PenaBlending method* in FDS is discussed. The input parameters required to run the simulations using this method have also been discussed thoroughly. Two sets of simulations, namely *wind* and *fire* simulations have been conducted to investigate the results obtained using this new method. Simulations of *wind* and *fire* were carried out using the existing wind methods of FDS, termed as *traditional methods* in this study to verify the working of the *PenaBlending method*. Two sets of domain sizes were used, namely a *small domain* and a *large domain*, to verify the robustness of the new method and is independent of the size of the domain used. For some case studies, such as using data from Windninja, the *small domain* is used to demonstrate the ability. FDS version 6.3.0 onward requires very fine mesh resolution to obtain grid converged results, which in turn contributes to increase in computation time. The reaction rate limiter (similar to versions before FDS 6.3.0) has been re-introduced in FDS 6.6.0 to curb the requirement of very fine mesh over the fire ground and thereby considerably decreasing the computation time for fire simulations. Two reference cases are tested to show that the combustion sub-model is correctly implemented in FDS 6.6.0. Following the grid convergence study by [Moinuddin et al. \(2018\)](#), a mesh resolution of  $0.25\text{m} \times 0.25\text{m} \times 0.25\text{m}$  has been used over the burnable fire plot for all the fire simulations. All the simulations were carried out in neutral, atmospherically stable conditions. Some simulations with different stabilities were conducted and presented in Appendix(C).

Firstly, the *wind* simulations were carried out to test how fast a statistically steady state wind field is achieved using the *PenaBlending method* as compared to the existing *traditional methods*. It is observed that this novel method produces a steady-state wind field faster than that produced by the *wall-of-wind* method without SEM (wind1) and with SEM method (wind2). It is observed that the spin-up time of wind using the *PenaBlending*

*method* is as low as  $\sim 80$  s for the *small* domain and  $\sim 200$  s for the *large* domain. It is also observed that this novel method gives comparable results with that using the *mean-forcing* method (wind3). It can be argued that the *PenaBlending* method can be preferred over the *mean-forcing* method as the *PenaBlending* method has certain additional abilities such as using terrain modified wind field. A detailed overview of the time improvement and accuracy of the *PenaBlending method* is given in Table-4.1. Furthermore, Appendix(C) shows that the *mean-forcing* method (also known as Monin-Obukhov method with neutral stability) gives unreliable results with fire, when different stability values are used. This shows that it is still too pre-mature to conduct fire simulations. The *PenaBlending* method uses external data (referred to as *pSim* data) from other reduced wind models or generated from analytical methods. The external data generated can be in coarser grid resolutions and hence will not be expensive to compute; for example, 4.25 s for generating data over an area of  $1.17\text{km} \times 1.17\text{km}$  using Windninja - Section-4.7 as opposed to a precursor simulation, like using a *wall-of-wind* method, that requires several minutes to hours to generate a wind field to be used in the main simulation. Moreover, wind data can be read-in as *pSim* data at various time-steps to model the gusting effect of wind on fire (refer to Section-4.6). This cannot be done using the *traditional methods*.

A set of *fire* simulations have also been carried out in this study in both the domain to verify the working of the *PenaBlending* method on fire simulations and these have been compared against *traditional methods*. Fire parameters like RoS and HRR have been compared as these are the most important parameters which researchers, as well as the end-users, are interested in. It has been observed that the *PenaBlending* method generates more RoS and HRR as compared to *traditional methods*. This can be explained because in this study, the wind profiles for all the cases have been matched  $u_{10}$  at the inlet. Figure-4.7 shows that  $u_2$  velocity of the *PenaBlending* method is more than that of the *traditional methods*, which plays a major role in RoS and HRR, thus increasing it (Moinuddin et al. (2018)). The overall profiles are matched reasonably well with that of *traditional methods* for both the domains, which verifies the working of the new method. In the current study, *PenaBlending* method has neither been tested for different atmospheric stabilities, nor for



different terrain conditions. A set of simulations have been run with different stabilities as a preliminary study as a part of the Appendix, showing Monin-Obukhov as implemented in FDS is pre-mature to study fire simulations and needs further investigation, which is out of the scope of the current study.

## 5.1 Future work and recommendations

The simulations that are carried out in this study have been conducted on a flat terrain with smaller fires, and structures like trees, buildings, slopes or any other obstructions have not been incorporated. Hence, the *PenaBlending* method has been implemented and checked for flat terrain, but can be extended further on complex terrain as a part of future research. The domain of application is also smaller as compared to the real-time extensive fires which spread for several kilometers. Within the domain considered in the current research, the coupling is effectively both ways. This means that the fire feels the atmosphere and the atmosphere feels the fire. However, the large scale atmospheric processes that are involved in extensive and devastating fires like 'the blow-up fires' (McRae and Sharples (2013)) cannot yet be captured in physics-based simulations, which needs to resolve the small spatial scales. Currently, the *PenaBlending* method is applied only along x-direction. This can be later extended to be applied along y and z-directions and to incorporate more complex wind patterns into FDS, for a future study. Furthermore, gusting effects of wind or wind direction change on fire spread can also be investigated further for complex terrains. The current study has implemented and tested the *PenaBlending* method for neutral atmospheric conditions only. Implementing and testing the *PenaBlending* method with different stability conditions also needs to be explored further, as the current study is restricted to neutral atmospheric stability only.

The physics-based modeling of fires has improved dramatically over the past decade and has become the current state-of-the-art in predicting and simulating wild-land fires. The present work contributes to that improvement by allowing more realistic wind fields to be used for doing fire simulations. The present work has resulted in a new inlet-outlet *PenaBlending method* which allows FDS to take realistic wind-fields from other reduced models like Windninja as input for carrying out fire simulations; hence, reducing the



computation cost of the model. Physics-based modelling will likely supplement empirical research and operational modelling in the short term, and may become the norm in future. Physics-based modelling is rapidly becoming a 'virtual experiment facility'. [Moinuddin and Sutherland \(2019\)](#) demonstrates this fact by assessing the model's capability to simulate transitioning from a forest floor fire to a crown fire, subsequently leading to a quasi-steady state RoS. They have also demonstrated how well the simulation results agree with the experimental results of the tree fire. Moreover, these models can provide insight into the physical mechanisms which results in different fire behaviours including RoS. [Sutherland and Moinuddin \(2019\)](#) have also explored the possibility of performance-based design for wildland building standards. They have also provided insight into how physics-based simulations of realistic fires impacting on the proposed structures will become a routine part of the design process in future. The major drawback of physics-based simulation remains the high computational cost. The current research contributes towards this knowledge and provides a gateway to reduce computational cost. As stated by [Cruz et al. \(2017\)](#), the physics-based models provides an acceptable representation of wildland fire behaviour and are preferred by wildland fire researchers because of the difficulties, costs, danger to life and constraints associated with outdoor experimental fires.

## Bibliography

- Andrews, P. L. Behave: fire behavior prediction and fuel modeling system-burn subsystem, part 1. 1986.
- Barenblatt, G. Scaling laws for fully developed turbulent shear flows. part 1. basic hypotheses and analysis. *Journal of Fluid Mechanics*, 248:513–520, 1993.
- Barenblatt, G. and Chorin, A. J. Scaling laws and vanishing-viscosity limits for wall-bounded shear flows and for local structure in developed turbulence. *Communications on Pure and Applied Mathematics: A Journal Issued by the Courant Institute of Mathematical Sciences*, 50(4):381–398, 1997.
- Barenblatt, G. and Goldenfeld, N. Does fully developed turbulence exist? reynolds number independence versus asymptotic covariance. *Physics of Fluids*, 7(12):3078–3082, 1995.
- Barenblatt, G. and Prostokishin, V. Scaling laws for fully developed turbulent shear flows. part 2. processing of experimental data. *Journal of Fluid Mechanics*, 248:521–529, 1993.
- Bedia, J., Herrera, S., Gutiérrez, J. M., Benali, A., Brands, S., Mota, B., and Moreno, J. M. Global patterns in the sensitivity of burned area to fire-weather: Implications for climate change. *Agricultural and Forest Meteorology*, 214:369–379, 2015.
- Chan, S. T. and Sugiyama, G. Users manual for mc wind: A new mass-consistent wind model for arac-3. *Office of Scientific and Technical Information Document URCLMA-129067, Oak Ridge, TN*, 1997.
- Cheney, N., Gould, J., and Catchpole, W. R. Prediction of fire spread in grasslands. *International Journal of Wildland Fire*, 8(1):1–13, 1998.

- Clark, T. L., Jenkins, M. A., Coen, J., and Packham, D. A coupled atmosphere–fire model: convective feedback on fire-line dynamics. *Journal of Applied Meteorology*, 35(6):875–901, 1996.
- Cope, M. and Chaloner, W. Wildfire: an interaction of biological and physical processes. *Geological factors and the evolution of plants*, pages 257–277, 1985.
- Cruz, M. G., Gould, J. S., Alexander, M. E., Sullivan, A. L., McCaw, W. L., and Matthews, S. *A guide to rate of fire spread models for Australian vegetation*. Australasian Fire and Emergency Service Authorities Council Limited and Commonwealth Scientific and Industrial Research Organisation, 2015.
- Cruz, M. G., Alexander, M. E., and Sullivan, A. L. Mantras of wildland fire behaviour modelling: facts or fallacies? *International journal of wildland fire*, 26(11):973–981, 2017.
- Davies, H. A laterul boundary formulation for multi-level prediction models. *Quarterly Journal of the Royal Meteorological Society*, 102(432):405–418, 1976.
- Deardorff, J. W. Stratocumulus-capped mixed layers derived from a three-dimensional model. *Boundary-Layer Meteorology*, 18(4):495–527, 1980.
- Dupuy, J.-L. and Morvan, D. Numerical study of a crown fire spreading toward a fuel break using a multiphase physical model. *International Journal of Wildland Fire*, 14(2): 141–151, 2005.
- Dyer, A. A review of flux-profile relationships. *Boundary-Layer Meteorology*, 7(3):363–372, 1974.
- Fernandes, P. A. M. Fire spread prediction in shrub fuels in portugal. *Forest ecology and management*, 144(1-3):67–74, 2001.
- Finney, M. A. Farsite: Fire area simulator-model development and evaluation. *Res. Pap. RMRS-RP-4, Revised 2004*. Ogden, UT: US Department of Agriculture, Forest Service, Rocky Mountain Research Station. 47 p., 4, 1998.
- Forthofer, J. M. *Modeling wind in complex terrain for use in fire spread prediction*. PhD thesis, Colorado State University Fort Collins, 2007.

- Forthofer, J. M., Butler, B. W., McHugh, C. W., Finney, M. A., Bradshaw, L. S., Stratton, R. D., Shannon, K. S., and Wagenbrenner, N. S. A comparison of three approaches for simulating fine-scale surface winds in support of wildland fire management. part ii. an exploratory study of the effect of simulated winds on fire growth simulations. *International Journal of Wildland Fire*, 23(7):982–994, 2014a.
- Forthofer, J. M., Butler, B. W., and Wagenbrenner, N. S. A comparison of three approaches for simulating fine-scale surface winds in support of wildland fire management. part i. model formulation and comparison against measurements. *International Journal of Wildland Fire*, 23(7):969–981, 2014b.
- Frangieh, N., Morvan, D., Meradji, S., Accary, G., and Bessonov, O. Numerical simulation of grassland fires behavior using an implicit physical multiphase model. *Fire Safety Journal*, 102:37–47, 2018.
- Gould, J., McCaw, W., Cheney, N., Ellis, P., Knight, I., and Sullivan, A. Project vesta-fire in dry eucalypt forest: fuel structure, fuel dynamics, and fire behaviour. 2007.
- Jarrin, N., Benhamadouche, S., Laurence, D., and Prosser, R. A synthetic-eddy-method for generating inflow conditions for large-eddy simulations. *International Journal of Heat and Fluid Flow*, 27(4):585–593, 2006.
- Jolly, W. M., Cochrane, M. A., Freeborn, P. H., Holden, Z. A., Brown, T. J., Williamson, G. J., and Bowman, D. M. Climate-induced variations in global wildfire danger from 1979 to 2013. *Nature communications*, 6:7537, 2015.
- Kalnay, E. *Atmospheric modeling, data assimilation and predictability*. Cambridge university press, 2003.
- Kemp, E. Pre-queternary fire in australia. *Fire and the Australian biota*, pages 3–21, 1981.
- Linn, R., Canfield, J., Cunningham, P., Edminster, C., Dupuy, J.-L., and Pimont, F. Using periodic line fires to gain a new perspective on multi-dimensional aspects of forward fire spread. *Agricultural and Forest Meteorology*, 157:60–76, 2012.

- Manzello, S. L., Blanchi, R., Gollner, M. J., Gorham, D., McAllister, S., Pastor, E., Planas, E., Reszka, P., and Suzuki, S. Summary of workshop large outdoor fires and the built environment. *Fire safety journal*, 100:76–92, 2018.
- McAneney, J., Chen, K., and Pitman, A. 100-years of Australian bushfire property losses: Is the risk significant and is it increasing? *Journal of Environmental Management*, 90(8): 2819–2822, 2009.
- McArthur, A. G. Fire behaviour in eucalypt forests. 1967.
- McGrattan, K., Klein, B., Hostikka, S., and Floyd, J. Fire dynamics simulator (version 6.2.0), users guide. *National Institute of Standards and Technology special publication*, 1019 (6):1–306, 2015.
- McGrattan, K., Hostikka, S., McDermott, R., Floyd, J., and Vanella, M. *Fire dynamics simulator, technical reference guide Volume 1: Mathematical Model*. US Department of Commerce, Technology Administration, National Institute of Standards and Technology, 2017a. doi: <http://dx.doi.org/10.6028/NIST.SP.1018>.
- McGrattan, K., Hostikka, S., McDermott, R., Floyd, J., and Vanella, M. *Fire dynamics simulator Technical reference guide Volume 3: Validation*. US Department of Commerce, Technology Administration, National Institute of Standards and Technology, 2017b. doi: <http://dx.doi.org/10.6028/NIST.SP.1018>.
- McGrattan, K., Hostikka, S., McDermott, R., Floyd, J., and Vanella, M. *Fire dynamics simulator Technical reference guide Volume 2: Verification*. US Department of Commerce, Technology Administration, National Institute of Standards and Technology, 2017c. doi: <http://dx.doi.org/10.6028/NIST.SP.1018>.
- McGrattan, K., Klein, B., Hostikka, S., and Floyd, J. Fire dynamics simulator (version 6.6.0), users guide. *National Institute of Standards and Technology special publication*, 1019 (6):1–339, 2017d.
- McRae, R. and Sharples, J. A process model for forecasting conditions conducive to blow-up fire events. In *Proceedings of the 2013 MODSIM Conference, Adelaide, Australia*, pages 1–6, 2013.

- Mell, W., Jenkins, M. A., Gould, J., and Cheney, P. A physics-based approach to modelling grassland fires. *International Journal of Wildland Fire*, 16(1):1–22, 2007.
- Mell, W., Maranghides, A., McDermott, R., and Manzello, S. L. Numerical simulation and experiments of burning douglas fir trees. *Combustion and Flame*, 156(10):2023–2041, 2009.
- Mell, W., Simeoni, A., Morvan, D., Hiers, J. K., Skowronski, N., and Hadden, R. M. Clarifying the meaning of mantras in wildland fire behaviour modelling: reply to cruz et al.(2017). *International Journal of Wildland Fire*, 27(11):770–775, 2018.
- Miller, C., Hilton, J., Sullivan, A., and Prakash, M. Spark—a bushfire spread prediction tool. In *International Symposium on Environmental Software Systems*, pages 262–271. Springer, 2015.
- Moinuddin, K. and Sutherland, D. Physics based modelling of tree fires and fires transitioning from the forest floor to the canopy. In *MODSIM2017 22nd International Congress on Modelling and Simulation: Proceedings*, pages 1138–1144. Modelling and Simulation Society of Australia and New Zealand (MSSANZ), 2017.
- Moinuddin, K. and Sutherland, D. Modelling of tree fires and fires transitioning from the forest floor to the canopy with a physics-based model. *Mathematics and Computers in Simulation*, 2019.
- Moinuddin, K., Sutherland, D., and Mell, W. Simulation study of grass fire using a physics-based model: striving towards numerical rigour and the effect of grass height on the rate of spread. *International Journal of Wildland Fire*, 27(12):800–814, 2018.
- Monin, A. and Obukhov, A. Basic laws of turbulent mixing in the surface layer of the atmosphere. *Contrib. Geophys. Inst. Acad. Sci. USSR*, 151:163–187, 1954.
- Montero, G., Montenegro, R., and Escobar, J. A 3-d diagnostic model for wind field adjustment. *Journal of Wind Engineering and Industrial Aerodynamics*, 74:249–261, 1998.
- Morvan, D. and Dupuy, J. Modeling of fire spread through a forest fuel bed using a multiphase formulation. *Combustion and flame*, 127(1-2):1981–1994, 2001.

- Morvan, D. and Dupuy, J. Modeling the propagation of a wildfire through a mediterranean shrub using a multiphase formulation. *Combustion and flame*, 138(3):199–210, 2004.
- Morvan, D., Dupuy, J., Rigolot, E., and Valette, J. Firestar: a physically based model to study wildfire behaviour. *Forest Ecology and Management*, (234):S114, 2006.
- Morvan, D., Meradji, S., and Accary, G. Physical modelling of fire spread in grasslands. *Fire Safety Journal*, 44(1):50–61, 2009.
- Morvan, D., Accary, G., Meradji, S., Frangieh, N., and Bessonov, O. A 3d physical model to study the behavior of vegetation fires at laboratory scale. *Fire Safety Journal*, 101: 39–52, 2018.
- Orloff, L. and De Ris, J. Froude modeling of pool fires. In *Symposium (International) on Combustion*, volume 19, pages 885–895. Elsevier, 1982.
- Pagni, P. J. Causes of the 20 october 1991 oakland hills conflagration. *Fire Safety Journal*, 21(4):331–339, 1993.
- Pavlidis, D., Gorman, G. J., Gomes, J. L., Pain, C. C., and ApSimon, H. Synthetic-eddy method for urban atmospheric flow modelling. *Boundary-layer meteorology*, 136(2):285–299, 2010.
- Perry, G. Current approaches to modelling the spread of wildland fire: a review. *Progress in Physical Geography*, 22(2):222–245, 1998.
- Pope, S. B. Turbulent flows, 2001. URL <http://stacks.iop.org/0957-0233/12/i=11/a=705>.
- Pyne, S. J. *Burning bush: a fire history of Australia*. Macmillan, 1991.
- Radeloff, V. C., Helmers, D. P., Kramer, H. A., Mockrin, M. H., Alexandre, P. M., Bar-Massada, A., Butsic, V., Hawbaker, T. J., Martinuzzi, S., Syphard, A. D., et al. Rapid growth of the us wildland-urban interface raises wildfire risk. *Proceedings of the National Academy of Sciences*, 115(13):3314–3319, 2018.

- Reisner, J., Wynne, S., Margolin, L., and Linn, R. Coupled atmospheric–fire modeling employing the method of averages. *Monthly Weather Review*, 128(10):3683–3691, 2000.
- Rodi, W. Comparison of les and rans calculations of the flow around bluff bodies. *Journal of wind engineering and industrial aerodynamics*, 69:55–75, 1997.
- Ronchi, E., Gwynne, S. M., Rein, G., Wadhvani, R., Intini, P., and Bergstedt, A. e-sanctuary: Open multi-physics framework for modelling wildfire urban evacuation. 2017.
- Ross, D., Smith, I. N., Manins, P., and Fox, D. Diagnostic wind field modeling for complex terrain: model development and testing. *Journal of Applied Meteorology*, 27(7):785–796, 1988.
- Rothermel, R. C. A mathematical model for predicting fire spread in wildland fuels. *Res. Pap. INT-115. Ogden, UT: US Department of Agriculture, Intermountain Forest and Range Experiment Station. 40 p., 115, 1972.*
- Ryder, N. L., Sutula, J. A., Schemel, C. F., Hamer, A. J., and Van Brunt, V. Consequence modeling using the fire dynamics simulator. *Journal of hazardous materials*, 115(1-3): 149–154, 2004.
- Sherman, C. A. A mass-consistent model for wind fields over complex terrain. *Journal of applied meteorology*, 17(3):312–319, 1978.
- Singha Roy, S., Sutherland, D., Khan, N., and Moinuddin, K. A comparative study of wind fields generated by different inlet parameters and their effects on fire spread using fire dynamics simulator. In *Proceedings of the 21st Australasian Fluid Mechanics Conference*. Australasian Fluid Mechanics Society, 2018.
- Sullivan, A. L. Wildland surface fire spread modelling, 1990–2007. 1: Physical and quasi-physical models. *International Journal of Wildland Fire*, 18(4):349–368, 2009a.
- Sullivan, A. L. Wildland surface fire spread modelling, 1990–2007. 2: Empirical and quasi-empirical models. *International Journal of Wildland Fire*, 18(4):369–386, 2009b.



- Sutherland, D. and Moinuddin, K. Simulation of heat fluxes on a structure from a fire in an idealised shrublandr. In *Proceedings of the AFAC19*. AFAC19, 2019.
- Sutherland, D., Moinuddin, K., and Ooi, A. Large-eddy simulation of neutral atmospheric surface layer flow over heterogeneous tree canopies. In *Research Forum 2017: proceedings from the Research Forum at the Bushfire and Natural Hazards CRC and AFAC Conference, Sydney, Australia, 4–6 September 2017*, pages 184–199. Bushfire and Natural Hazards CRC, 2017.
- Sutherland, D., Philip, J., Ooi, A., and Moinuddin, K. Large eddy simulation of flow over streamwise heterogeneous canopies: Quadrant analysis. In *Proceedings of the 21st Australasian Fluid Mechanics Conference*. Australasian Fluid Mechanics Society, 2018.
- Tolhurst, K., Shields, B., Chong, D., et al. Phoenix: development and application of a bushfire risk management tool. *Australian Journal of Emergency Management, The*, 23(4): 47, 2008.
- Touma, J. S. Dependence of the wind profile power law on stability for various locations. *Journal of the Air Pollution Control Association*, 27(9):863–866, 1977.
- Tutorial1. Windninja tutorial 1: The basics.
- Von Kármán, T. Mechanische ahnlichkeit und turbulenz. *Math.-Phys. Klasse*, 1930.
- Vonlanthen, M., Allegrini, J., and Carmeliet, J. Assessment of a one-way nesting procedure for obstacle resolved large eddy simulation of the abl. *Computers & Fluids*, 140: 136–147, 2016.
- Weber, R. Modelling fire spread through fuel beds. *Progress in Energy and Combustion Science*, 17(1):67–82, 1991.
- Werner, H. and Wengle, H. Large-eddy simulation of turbulent flow over and around a cube in a plate channel. In *Turbulent Shear Flows 8*, pages 155–168. Springer, 1993.
- Westhaver, A. *Why some homes survived: Learning from the Fort McMurray wildland/urban interface fire disaster*. Institute for Catastrophic Loss Reduction, 2017.

## BIBLIOGRAPHY

---

Whittaker, J., McLennan, J., Elliott, G., Gilbert, J., Handmer, J., Haynes, K., and Cowlshaw, S. Victorian 2009 bushfire research response: Final report. *Bushfire CRC Post-fire Research Program in Human Behaviour*. (Bushfire CRC: Melbourne) Available at <http://www.bushfirecrc.com/managed/resource>, 2009.

Wyngaard, J. C. *Turbulence in the Atmosphere*. Cambridge University Press, 2010.

Zhou, X. and Pereira, J. A multidimensional model for simulating vegetation fire spread using a porous media sub-model. *Fire and Materials*, 24(1):37–43, 2000.

# Appendices

# Appendix A

## Source Code files for edited portions of FDS

This section provides the source code of some of the files that have major code changes related to this research. The complete source code can be found in ([https://drive.google.com/open?id=18uQEmprrdpmNDgBGIDVswu9ER\\_mHpN9Ho](https://drive.google.com/open?id=18uQEmprrdpmNDgBGIDVswu9ER_mHpN9Ho)).

### A.1 *fire.f90*

```
1 | MODULE FIRE
2 |
3 |   ! Compute combustion
4 |
5 |   USE PRECISION_PARAMETERS
6 |   USE GLOBAL_CONSTANTS
7 |   USE MESH_POINTERS
8 |   USE COMP_FUNCTIONS, ONLY: SECOND
9 |
10 |  IMPLICIT NONE
11 |  PRIVATE
12 |
13 |  TYPE(REACTION_TYPE), POINTER :: RN=>NULL()
14 |  REAL(EB) :: Q_UPPER
15 |  LOGICAL :: EXTINGT1 = .FALSE.      !changed name to EXTINGT1 for 6.2.0
16 |
17 |  INTEGER :: NM
18 |  REAL(EB) :: T,DT
19 |
20 |  PUBLIC GET_REV_fire
21 |  PUBLIC COMBUSTION
22 |
23 |  CONTAINS
24 |
25 |  SUBROUTINE COMBUSTION(T,DT,NM)
26 |
27 |    INTEGER, INTENT(IN) :: NM
28 |    REAL(EB), INTENT(IN) :: T,DT
29 |    REAL(EB) :: TNCW
30 |
31 |    IF (EVACUATION_ONLY(NM)) RETURN
32 |
33 |    TNCW=SECOND()
34 |
35 |    IF (INIT_HRRPUV) RETURN
36 |
37 |    CALL POINT_TO_MESH(NM)
38 |
39 |    SELECT_COMB: SELECT CASE (COMB_MODEL)
40 |
41 |    CASE (COMBUSTIONSIX) SELECT_COMB
42 |
43 |    CALL COMBUSTION_GENERAL(T,DT) !combustion model for 6.6.0
44 |
45 |    CASE (COMBUSTIONTWO) SELECT_COMB
46 |
```

Source Code files for edited portions of FDS

```

47 ! Upper bounds on local HRR per unit volume
48 Q_UPPER = HRRPUA.SHEET/CELL.SIZE + HRRPUV.AVERAGE
49
50 CALL COMBUSTION.GENERAL_1(T,DT) !combustion model for 6.2.0
51
52 END SELECT SELECT.COMB
53
54 IF (CC.IBM) CALL CCREGION.COMBUSTION(T,DT,NM)
55
56 T.USED(10)=T.USED(10)+SECOND()-INOW
57
58 END SUBROUTINE COMBUSTION
59
60 SUBROUTINE COMBUSTION.GENERAL_1(T,DT)
61
62 ! Generic combustion routine for multi-step reactions
63
64 USE PHYSICAL_FUNCTIONS, ONLY: GET_SPECIFIC_GAS_CONSTANT,GET_MASS_FRACTION_ALL,GET_SPECIFIC_HEAT,
65     GET_MOLECULAR_WEIGHT, &
66     GET_SENSIBLE_ENTHALPY_Z,IS_REALIZABLE,LES_FILTER_WIDTH_FUNCTION
67 USE COMPLEX_GEOMETRY, ONLY: IBM.CGSC, IBM.GASPHASE
68 INTEGER :: I,J,K,NS,NR,II,JJ,KK,IIG,JJG,KKG,IW,N,CHEM.SUBIT.TMP
69 REAL(EB), INTENT(IN) :: T,DT
70 REAL(EB) :: ZZ.GET(1:N.TRACKED.SPECIES),DZZ(1:N.TRACKED.SPECIES),CP,H.S.N,&
71     REAC.SOURCE.TERM.TMP(N.TRACKED.SPECIES),Q.REAC.TMP(N.REACTIONS)
72 LOGICAL :: Q_EXISTS
73 REAL(EB), POINTER, DIMENSION(:,:,:) :: ZETA.P=>NULL(),AIT.P=>NULL()
74 TYPE (REACTION_TYPE), POINTER :: RN
75 TYPE (SPECIES_MIXTURE_TYPE), POINTER :: SM
76 LOGICAL :: DO.REACTION,REALIZABLE
77
78 Q = 0._EB
79 Q_EXISTS = .FALSE.
80
81 CHL.R = 0._EB
82 IF (REAC.SOURCE.CHECK) Q.REAC=0._EB
83
84 IF (TRANSPORT.UNMIXED.FRACTION .AND. &
85     COMPUTE.ZETA.SOURCE.TERM .AND. &
86     TRANSPORT.ZETA.SCHEME==1 ) CALL ZETA.PRODUCTION(DT) ! scheme 1: zeta production before mixing
87
88 ZETA.P => WORK1
89 ZETA.P = 0._EB
90 IF (TRANSPORT.UNMIXED.FRACTION) ZETA.P = ZZ(:,:,:),ZETA.INDEX)
91
92 AIT.P => WORK2
93 AIT.P = 0._EB
94 IF (REIGNITION.MODEL) AIT.P = AIT
95
96 DO K=1,KBAR
97 DO J=1,JBAR
98 ILOOP: DO I=1,IBAR
99 ! Check to see if a reaction is possible
100 IF (SOLID(CELL.INDEX(I,J,K))) CYCLE ILOOP
101 IF (CC.IBM) THEN
102 IF (CCVAR(I,J,K,IBM.CGSC) /= IBM.GASPHASE) CYCLE ILOOP
103 ENDIF
104 ZZ.GET = ZZ(I,J,K,1:N.TRACKED.SPECIES)
105 IF (CHECK.REALIZABILITY) THEN
106 REALIZABLE=IS.REALIZABLE(ZZ.GET)
107 IF (.NOT.REALIZABLE) THEN
108 WRITE(LU.ERR,*) I,J,K
109 WRITE(LU.ERR,*) ZZ.GET
110 WRITE(LU.ERR,*) SUM(ZZ.GET)
111 STOP,STATUS='ERROR: Unrealizable mass fractions input to COMBUSTION.MODEL'
112 ENDIF
113 ENDIF
114 CALL CHECK_REACTION_1
115 IF (.NOT.DO.REACTION) CYCLE ILOOP ! Check whether any reactions are possible.
116 DZZ = ZZ.GET ! store old ZZ for divergence term
117 !*****
118 ! Call combustion integration routine for Cartesian cell (I,J,K)
119 CALL COMBUSTION.MODEL_1(T,DT,ZZ.GET,Q(I,J,K),MIX.TIME(I,J,K),CHL.R(I,J,K),&
120     CHEM.SUBIT.TMP,REAC.SOURCE.TERM.TMP,Q.REAC.TMP,&
121     TMP(I,J,K),RHO(I,J,K),MU(I,J,K),KRES(I,J,K),&
122     ZETA.P(I,J,K),AIT.P(I,J,K),PBAR(K,PRESSURE.ZONE(I,J,K)),&
123     LES_FILTER_WIDTH_FUNCTION(DX(I),DY(J),DZ(K)),DX(I)*DY(J)*DZ(K) )
124 !*****
125 IF (OUTPUT.CHEM.IT) CHEM.SUBIT(I,J,K) = CHEM.SUBIT.TMP
126 IF (REAC.SOURCE.CHECK) THEN ! Store special diagnostic quantities
127 REAC.SOURCE.TERM(I,J,K,:) = REAC.SOURCE.TERM.TMP
128 Q.REAC(I,J,K,:) = Q.REAC.TMP
129 ENDIF
130 IF (CHECK.REALIZABILITY) THEN
131 REALIZABLE=IS.REALIZABLE(ZZ.GET)
132 IF (.NOT.REALIZABLE) THEN
133 WRITE(LU.ERR,*) ZZ.GET,SUM(ZZ.GET)

```

Source Code files for edited portions of FDS

```

134 WRITE(LU_ERR,*) 'ERROR: Unrealizable mass fractions after COMBUSTION_MODEL'
135 STOP STATUS=REALIZABILITY_STOP
136 ENDIF
137 ENDIF
138 DZZ = ZZ.GET - DZZ
139 ! Update RSUM and ZZ
140 DZZ_IF: IF ( ANY(ABS(DZZ) > TWO_EPSILON_EB ) THEN
141 IF (ABS(Q(I,J,K)) > TWO_EPSILON_EB) Q_EXISTS = .TRUE.
142 ! Divergence term
143 CALL GET_SPECIFIC_HEAT(ZZ.GET, CP, TMP(I,J,K))
144 CALL GET_SPECIFIC_GAS_CONSTANT(ZZ.GET, RSUM(I,J,K))
145 DO N=1, N_TRACKED_SPECIES
146 SM => SPECIES_MIXTURE(N)
147 CALL GET_SENSIBLE_ENTHALPY_Z(N, TMP(I,J,K), H_S_N)
148 D_SOURCE(I,J,K) = D_SOURCE(I,J,K) + ( SM*RCON/RSUM(I,J,K) - H_S_N/(CP*TMP(I,J,K)) ) * DZZ(N)/DT
149 M_DOT_PPP(I,J,K,N) = M_DOT_PPP(I,J,K,N) + RHO(I,J,K) * DZZ(N)/DT
150 ENDDO
151 ENDIF DZZ_IF
152 ENDDO ILOOP
153 ENDDO
154 ENDDO
155
156 IF (TRANSPORT_UNMIXED_FRACTION) ZZ(:, :, ZETA_INDEX) = ZETA_P
157
158 IF (TRANSPORT_UNMIXED_FRACTION .AND. &
159 COMPUTE_ZETA_SOURCE_TERM .AND. &
160 TRANSPORT_ZETA_SCHEME==2 ) CALL ZETA_PRODUCTION(DT) ! scheme 2: zeta production after mixing
161
162 IF (.NOT. Q_EXISTS) RETURN
163
164 ! Set Q in the ghost cell, just for better visualization.
165
166 DO IW=1, N_EXTERNAL_WALL_CELLS
167 IF (WALL(IW)%BOUNDARY_TYPE/=INTERPOLATED_BOUNDARY .AND. WALL(IW)%BOUNDARY_TYPE/=OPEN_BOUNDARY) CYCLE
168 II = WALL(IW)%ONE_D%II
169 JJ = WALL(IW)%ONE_D%JJ
170 KK = WALL(IW)%ONE_D%KK
171 IIG = WALL(IW)%ONE_D%IIG
172 JJG = WALL(IW)%ONE_D%JJG
173 KKG = WALL(IW)%ONE_D%KKG
174 Q(II, JJ, KK) = Q(IIG, JJG, KKG)
175 ENDDO
176
177 CONTAINS
178
179 SUBROUTINE CHECK_REACTION_1
180
181 ! Check whether any reactions are possible.
182
183 LOGICAL :: REACTANTS_PRESENT
184
185 DO REACTION = .FALSE.
186 REACTION_LOOP: DO NR=1, N_REACTIONS
187 RN=>REACTION(NR)
188 REACTANTS_PRESENT = .TRUE.
189 DO NS=1, N_TRACKED_SPECIES
190 IF ( RN%NU(NS) < -TWO_EPSILON_EB .AND. ZZ.GET(NS) < ZZ_MIN_GLOBAL ) THEN
191 REACTANTS_PRESENT = .FALSE.
192 EXIT
193 ENDIF
194 ENDDO
195 DO REACTION = REACTANTS_PRESENT
196 IF (DO_REACTION) EXIT REACTION_LOOP
197 ENDDO REACTION_LOOP
198
199 END SUBROUTINE CHECK_REACTION_1
200
201 END SUBROUTINE COMBUSTION_GENERAL_1
202
203 SUBROUTINE COMBUSTION_GENERAL(T, DT)
204
205 ! Generic combustion routine for multi-step reactions
206
207 USE PHYSICAL_FUNCTIONS, ONLY: GET_SPECIFIC_GAS_CONSTANT, GET_MASS_FRACTION_ALL, GET_SPECIFIC_HEAT,
    GET_MOLECULAR_WEIGHT, &
208 GET_SENSIBLE_ENTHALPY_Z, IS_REALIZABLE, LES_FILTER_WIDTH_FUNCTION
209 USE COMPLEX_GEOMETRY, ONLY: IBM.CGSC, IBM.GASPHASE
210 INTEGER :: I, J, K, NS, NR, II, JJ, KK, IIG, JJG, KKG, IW, N, CHEM.SUBIT, TMP
211 REAL(EB), INTENT(IN) :: T, DT
212 REAL(EB) :: ZZ.GET(1:N_TRACKED_SPECIES), DZZ(1:N_TRACKED_SPECIES), CP, H_S_N, &
213 REAC_SOURCE_TERM_TMP(N_TRACKED_SPECIES), Q_REAC_TMP(N_REACTIONS)
214 LOGICAL :: Q_EXISTS
215 REAL(EB), POINTER, DIMENSION(:, :, :) :: ZETA_P=>NULL(), AIT_P=>NULL()
216 TYPE(REACTION_TYPE), POINTER :: RN
217 TYPE(SPECIES_MIXTURE_TYPE), POINTER :: SM
218 LOGICAL :: DO_REACTION, REALIZABLE
219
220 Q = 0. _EB

```

```

221 Q_EXISTS = .FALSE.
222
223 CHLR = 0._EB
224 IF (REAC.SOURCE.CHECK) Q_REAC=0._EB
225
226 IF (TRANSPORT.UNMIXED.FRACTION .AND. &
227 COMPUTE.ZETA.SOURCE.TERM .AND. &
228 TRANSPORT.ZETA.SCHEME==1 ) CALL ZETA.PRODUCTION(DT) ! scheme 1: zeta production before mixing
229
230 ZETA.P => WORK1
231 ZETA.P = 0._EB
232 IF (TRANSPORT.UNMIXED.FRACTION) ZETA.P = ZZ(:, :, ZETA.INDEX)
233
234 AIT.P => WORK2
235 AIT.P = 0._EB
236 IF (REIGNITION.MODEL) AIT.P = AIT
237
238 DO K=1,KBAR
239 DO J=1,JBAR
240 ILOOP: DO I=1,IBAR
241 ! Check to see if a reaction is possible
242 IF (SOLID(CELL.INDEX(I,J,K))) CYCLE ILOOP
243 IF (CC.IBM) THEN
244 IF (CCVAR(I,J,K,IBM.CGSC) /= IBM.GASPHASE) CYCLE ILOOP
245 ENDIF
246 ZZ.GET = ZZ(I,J,K,1:N.TRACKED.SPECIES)
247 IF (CHECK.REALIZABILITY) THEN
248 REALIZABLE=IS.REALIZABLE(ZZ.GET)
249 IF (.NOT.REALIZABLE) THEN
250 WRITE(LU.ERR,*) I,J,K
251 WRITE(LU.ERR,*) ZZ.GET
252 WRITE(LU.ERR,*) SUM(ZZ.GET)
253 WRITE(LU.ERR,*) 'ERROR: Unrealizable mass fractions input to COMBUSTION.MODEL'
254 STOP.STATUS=REALIZABILITY.STOP
255 ENDIF
256 ENDIF
257 CALL CHECK.REACTION
258 IF (.NOT.DO.REACTION) CYCLE ILOOP ! Check whether any reactions are possible.
259 DZZ = ZZ.GET ! store old ZZ for divergence term
260 !*****
261 ! Call combustion integration routine for Cartesian cell (I,J,K)
262 CALL COMBUSTION.MODEL( T,DT,ZZ.GET,Q(I,J,K),MIX.TIME(I,J,K),CHLR(I,J,K),&
263 CHEM.SUBIT.TMP,REAC.SOURCE.TERM.TMP,Q_REAC.TMP,&
264 TMP(I,J,K),RHO(I,J,K),MU(I,J,K),KRES(I,J,K),&
265 ZETA.P(I,J,K),AIT.P(I,J,K),PBAR(K,PRESSURE.ZONE(I,J,K)),&
266 LES.FILTER.WIDTH.FUNCTION(DX(I),DY(J),DZ(K)),DX(I)*DY(J)*DZ(K) )
267 !*****
268 IF (OUTPUT.CHEM.IT) CHEM.SUBIT(I,J,K) = CHEM.SUBIT.TMP
269 IF (REAC.SOURCE.CHECK) THEN ! Store special diagnostic quantities
270 REAC.SOURCE.TERM(I,J,K,:) = REAC.SOURCE.TERM.TMP
271 Q_REAC(I,J,K,:) = Q_REAC.TMP
272 ENDIF
273 IF (CHECK.REALIZABILITY) THEN
274 REALIZABLE=IS.REALIZABLE(ZZ.GET)
275 IF (.NOT.REALIZABLE) THEN
276 WRITE(LU.ERR,*) ZZ.GET,SUM(ZZ.GET)
277 WRITE(LU.ERR,*) 'ERROR: Unrealizable mass fractions after COMBUSTION.MODEL'
278 STOP.STATUS=REALIZABILITY.STOP
279 ENDIF
280 ENDIF
281 DZZ = ZZ.GET - DZZ
282 ! Update RSUM and ZZ
283 DZZ.IF: IF ( ANY(ABS(DZZ) > TWO.EPSILON.EB) ) THEN
284 IF (ABS(Q(I,J,K)) > TWO.EPSILON.EB) Q_EXISTS = .TRUE.
285 ! Divergence term
286 CALL GET.SPECIFIC.HEAT(ZZ.GET,CP,TMP(I,J,K))
287 CALL GET.SPECIFIC.GAS.CONSTANT(ZZ.GET,RSUM(I,J,K))
288 DO N=1,N.TRACKED.SPECIES
289 SM => SPECIES.MIXTURE(N)
290 CALL GET.SENSIBLE.ENTHALPY.Z(N,TMP(I,J,K),H.S.N)
291 D.SOURCE(I,J,K) = D.SOURCE(I,J,K) + ( SM*RCON/RSUM(I,J,K) - H.S.N/(CP*TMP(I,J,K)) ) *DZZ(N)/DT
292 M.DOT.PPP(I,J,K,N) = M.DOT.PPP(I,J,K,N) + RHO(I,J,K)*DZZ(N)/DT
293 ENDDO
294 ENDIF DZZ.IF
295 ENDDO ILOOP
296 ENDDO
297 ENDDO
298
299 IF (TRANSPORT.UNMIXED.FRACTION) ZZ(:, :, ZETA.INDEX) = ZETA.P
300
301 IF (TRANSPORT.UNMIXED.FRACTION .AND. &
302 COMPUTE.ZETA.SOURCE.TERM .AND. &
303 TRANSPORT.ZETA.SCHEME==2 ) CALL ZETA.PRODUCTION(DT) ! scheme 2: zeta production after mixing
304
305 IF (.NOT.Q_EXISTS) RETURN
306
307 ! Set Q in the ghost cell, just for better visualization.
308

```

Source Code files for edited portions of FDS

```

309 DO IW=1,N,EXTERNAL_WALL_CELLS
310 IF (WALL(IW)%BOUNDARY_TYPE/=INTERPOLATED_BOUNDARY .AND. WALL(IW)%BOUNDARY_TYPE/=OPEN_BOUNDARY) CYCLE
311 II = WALL(IW)%ONE.D%II
312 JJ = WALL(IW)%ONE.D%JJ
313 KK = WALL(IW)%ONE.D%KK
314 IIG = WALL(IW)%ONE.D%IIG
315 JJG = WALL(IW)%ONE.D%JJG
316 KKG = WALL(IW)%ONE.D%KKG
317 Q(II, JJ, KK) = Q(IIG, JJG, KKG)
318 ENDDO
319
320 CONTAINS
321
322 SUBROUTINE CHECK_REACTION
323
324 ! Check whether any reactions are possible.
325
326 LOGICAL :: REACTANTS_PRESENT
327
328 DO_REACTION = .FALSE.
329 REACTION_LOOP: DO NR=1,N_REACTIONS
330 RN=>REACTION(NR)
331 REACTANTS_PRESENT = .TRUE.
332 DO NS=1,N_TRACKED_SPECIES
333 IF ( RN%NU(NS) < -TWO_EPSILON_EB .AND. ZZ_GET(NS) < ZZ_MIN_GLOBAL ) THEN
334 REACTANTS_PRESENT = .FALSE.
335 EXIT
336 ENDDO
337 ENDDO
338 DO_REACTION = REACTANTS_PRESENT
339 IF (DO_REACTION) EXIT REACTION_LOOP
340 ENDDO REACTION_LOOP
341
342 END SUBROUTINE CHECK_REACTION
343
344 END SUBROUTINE COMBUSTION_GENERAL
345
346 SUBROUTINE COMBUSTION_MODEL1(T,DT,ZZ_GET,Q_OUT,MIX_TIME_OUT,CHL_R_OUT,CHEM_SUBIT_OUT,REAC_SOURCE_TERM_OUT,
    Q_REAC_OUT,&
347 TMP_IN,RHO_IN,MU_IN,KRES_IN,ZETA_INOUT,AIT_IN,PBAR_IN,DELTA,CELL_VOLUME)
348 USE COMP_FUNCTIONS, ONLY: SHUTDOWN
349 USE MATH_FUNCTIONS, ONLY: EVALUATE_RAMP
350 USE PHYSICAL_FUNCTIONS, ONLY: GET_AVERAGE_SPECIFIC_HEAT,GET_SPECIFIC_GAS_CONSTANT,GET_ENTHALPY
351 !INTEGER,INTENT(IN) :: I,J,K !added asin 6.2.0
352 REAL(EB), INTENT(IN) :: T,DT,TMP_IN,RHO_IN,MU_IN,KRES_IN,PBAR_IN,AIT_IN,DELTA,CELL_VOLUME
353 REAL(EB), INTENT(OUT) :: Q_OUT,MIX_TIME_OUT,CHL_R_OUT,REAC_SOURCE_TERM_OUT(N_TRACKED_SPECIES),Q_REAC_OUT(
    N_REACTIONS)
354 INTEGER, INTENT(OUT) :: CHEM_SUBIT_OUT
355 REAL(EB), INTENT(INOUT) :: ZZ_GET(1:N_TRACKED_SPECIES),ZETA_INOUT
356 REAL(EB) :: ERR_EST,ERR_TOL,A1(1:N_TRACKED_SPECIES),A2(1:N_TRACKED_SPECIES),A4(1:N_TRACKED_SPECIES),ZETA,ZETA_0,&
357 DT_SUB,DT_SUB_NEW,DT_ITER,ZZ_STORE(1:N_TRACKED_SPECIES,1:4),TV(1:3,1:N_TRACKED_SPECIES),CELL_MASS,&
358 ZZ_DIFF(1:3,1:N_TRACKED_SPECIES),ZZ_MIXED(1:N_TRACKED_SPECIES),ZZ_UNMIXED(1:N_TRACKED_SPECIES),&
359 ZZ_MIXED_NEW(1:N_TRACKED_SPECIES),TAU_D,TAU_G,TAU_U,TAU_MIX,TMP_MIXED,TMP_UNMIXED,DT_SUB_MIN,RHO_HAT,&
360 PBAR_0,VEL_RMS,TAU_RES,ZZ_0(1:N_TRACKED_SPECIES),&
361 Q_REAC_SUB(1:N_REACTIONS),Q_REAC_1(1:N_REACTIONS),Q_REAC_2(1:N_REACTIONS),Q_REAC_4(1:N_REACTIONS),&
362 Q_REAC_SUM(1:N_REACTIONS),CHL_R_SUM,TIME_RAMP_FACTOR,&
363 TOTAL_MIXED_MASS_1,TOTAL_MIXED_MASS_2,TOTAL_MIXED_MASS_4,TOTAL_MIXED_MASS,&
364 ZETA_1,ZETA_2,ZETA_4,AIT_LOC,Q_SUM,Q_CUM,ZZ_TEMP(1:N_TRACKED_SPECIES) !added Q_CUM Q_SUM ZZ_TEMP(1:
    N_TRACKED_SPECIES)
365 INTEGER :: NR,NS,ITER,TVI,RICH_ITER,TIME_ITER,RICH_ITER_MAX,CO_PASS,N_CO_PASS,SR !added SR
366 INTEGER, PARAMETER :: TV_ITER_MIN=5
367 LOGICAL :: TV_FLUCT(1:N_TRACKED_SPECIES),EXTINCT
368 TYPE(REACTION_TYPE), POINTER :: RN=>NULL()
369 REAL(EB), PARAMETER :: C_U = 0.4_EB*0.1_EB*SQRT(1.5_EB) ! C_U*C_DEARDORFF/SQRT(2/3)
370
371 ZZ_TEMP = ZZ_GET !added as in 6.2.0
372 ZZ_0 = ZZ_GET
373 EXTINCT = .FALSE.
374
375 VEL_RMS = 0._EB
376 IF (FIXED_MIX_TIME>0._EB) THEN
377 MIX_TIME_OUT=FIXED_MIX_TIME
378 ELSE
379 TAU_D=0._EB
380 DO NR =1,N_REACTIONS
381 RN => REACTION(NR)
382 TAU_D = MAX(TAU_D,D_Z(MIN(4999,NINT(TMP_IN)),RN%FUEL_SMIX_INDEX))
383 ENDDO
384 TAU_D = DELTA**2/MAX(TAU_D,TWO_EPSILON_EB) ! FDS Tech Guide (5.20)
385 IF (LES) THEN
386 TAU_U = C_U*RHO_IN*DELTA**2/MAX(MU_IN,TWO_EPSILON_EB) ! FDS Tech Guide (5.24)
387 TAU_G = SQRT(2._EB*DELTA/(GRAV+1.E-10_EB)) ! FDS Tech Guide (5.2)
388 MIX_TIME_OUT= MAX(TAU_CHEM,MIN(TAU_D,TAU_U,TAU_G,TAU_FLAME)) ! FDS Tech Guide (5.22)
389 VEL_RMS = SQRT(TWTH)*MU_IN/(RHO_IN*C_DEARDORFF*DELTA)
390 ELSE
391 MIX_TIME_OUT= MAX(TAU_CHEM,TAU_D)
392 ENDDO
393 ENDDO

```



Source Code files for edited portions of FDS

```

394
395 IF (TRANSPORT.UNMIXED.FRACTION) THEN
396 ZETA_0 = ZETA_INOUT
397 ELSE
398 ZETA_0 = INITIAL.UNMIXED.FRACTION
399 ENDF
400 CELL.MASS = RHO.IN*CELL.VOLUME
401 TAU.RES = MU.IN/(RHO.IN*SC)/MAX(2._EB*KRES.IN,TWO.EPSILON.EB)
402
403 IF (REIGNITION.MODEL) THEN
404 AIT.LOC = AIT.IN
405 ELSE
406 AIT.LOC = 1.E20.EB
407 ENDF
408
409 DT.SUB.MIN = DT/REAL(MAX.CHEMISTRY.ITERATIONS,EB)
410
411 N.CO.PASS = 1
412 IF (EXTINCT.MOD==EXTINCTION.3) N.CO.PASS = 2
413
414 CO.EXTINCT.LOOP: DO CO.PASS = 1,N.CO.PASS
415
416 ZZ.STORE(:, :) = 0._EB
417 Q.OUT = 0._EB
418 Q.CUM = 0._EB !added
419 Q.SUM = 0._EB !added
420 ITER= 0
421 DT.ITER = 0._EB
422 CHL.R.OUT = 0._EB
423 CHEM.SUBIT.OUT = 0
424 REAC.SOURCE.TERL.OUT(:) = 0._EB
425 Q.REAC.OUT(:) = 0._EB
426 Q.REAC.SUM(:) = 0._EB
427 IF (N.FIXED.CHEMISTRY.SUBSTEPS>0) THEN
428 DT.SUB = DT/REAL(N.FIXED.CHEMISTRY.SUBSTEPS,EB)
429 DT.SUB.NEW = DT.SUB
430 RICH.ITER.MAX = 1
431 ELSE
432 DT.SUB = DT
433 DT.SUB.NEW = DT
434 RICH.ITER.MAX = 5
435 ENDF
436 ZZ.UNMIXED = ZZ.GET
437 ZZ.MIXED = ZZ.GET
438 A1 = ZZ.GET
439 A2 = ZZ.GET
440 A4 = ZZ.GET
441
442 ZETA = ZETA_0
443 RHO.HAT = RHO.IN
444 TMP.MIXED = TMP.IN
445 TMP.UNMIXED = TMP.IN
446 TAU.MIX = MIX.TIME.OUT
447 PBAR_0 = PBAR.IN
448 added upto as in 6.6.0
449
450 INTEGRATION.LOOP: DO TIME.ITER = 1,MAX.CHEMISTRY.ITERATIONS
451
452 IF (SUPPRESSION) CALL CHECK_AUTO_IGNITION(EXTINCT,TMP.MIXED,AIT.LOC,CO.PASS)
453 IF (EXTINCT) EXIT INTEGRATION.LOOP
454
455 INTEGRATOR.SELECT: SELECT CASE (COMBUSTION.ODE.SOLVER)
456
457 CASE (EXPLICIT.EULER) ! Simple chemistry
458
459 ! May be used with N.FIXED.CHEMISTRY.SUBSTEPS, but default mode is DT.SUB=DT for fast chemistry
460
461 CALL FIRE_FORWARD.EULER(ZZ.MIXED.NEW,ZZ.MIXED,ZZ.UNMIXED,ZETA,ZETA_0,DT.SUB,TMP.MIXED,TMP.UNMIXED,RHO.HAT,&
462 CELL.MASS,TAU.MIX,PBAR_0,DELTA,VEL.RMS,Q.REAC.SUB,TIME.ITER,TOTAL.MIXED.MASS,CO.PASS)
463 ZETA_0 = ZETA
464 ZZ.MIXED = ZZ.MIXED.NEW
465 IF (SIMPLE.CHEMISTRY .AND. N.FIXED.CHEMISTRY.SUBSTEPS<0 .AND. TIME.ITER>1) THEN
466 CALL SHUTDOWN('ERROR: Error in Simple Chemistry')
467 ENDF
468
469 CASE (RK2) ! Runge-Kutta 2 stage (use in combination with N.FIXED.CHEMISTRY.SUBSTEPS)
470
471 CALL FIRE_RK2(ZZ.MIXED.NEW,ZZ.MIXED,ZZ.UNMIXED,ZETA,ZETA_0,DT.SUB,1,TMP.MIXED,TMP.UNMIXED,RHO.HAT,&
472 CELL.MASS,TAU.MIX,PBAR_0,DELTA,VEL.RMS,Q.REAC.SUB,TIME.ITER,TOTAL.MIXED.MASS,CO.PASS)
473 ZETA_0 = ZETA
474 ZZ.MIXED = ZZ.MIXED.NEW
475
476 CASE (RK3) ! Runge-Kutta 3 stage (use in combination with N.FIXED.CHEMISTRY.SUBSTEPS)
477
478 CALL FIRE_RK3(ZZ.MIXED.NEW,ZZ.MIXED,ZZ.UNMIXED,ZETA,ZETA_0,DT.SUB,1,TMP.MIXED,TMP.UNMIXED,RHO.HAT,&
479 CELL.MASS,TAU.MIX,PBAR_0,DELTA,VEL.RMS,Q.REAC.SUB,TIME.ITER,TOTAL.MIXED.MASS,CO.PASS)
480 ZETA_0 = ZETA
481 ZZ.MIXED = ZZ.MIXED.NEW

```

## Source Code files for edited portions of FDS

```

482
483 CASE (RK2.RICHARDSON) ! Finite-rate (or mixed finite-rate/fast) chemistry
484
485 ! May be used with N.FIXED.CHEMISTRY.SUBSTEPS, but default mode is to use error estimator and variable DT.SUB
486
487 ERR_TOL = RICHARDSON.ERROR.TOLERANCE
488 RICH.EX.LOOP: DO RICH.ITER = 1, RICH.ITER.MAX
489
490 DT.SUB = MIN(DT.SUB.NEW, DT-DT.ITER)
491
492 ! FDS Tech Guide (E.3), (E.4), (E.5)
493 CALL FIRE_RK2(A1, ZZ.MIXED, ZZ.UNMIXED, ZETA.1, ZETA.0, DT.SUB, 1, TMP.MIXED, TMP.UNMIXED, RHO.HAT, CELL.MASS, TAU.MIX,
494 PBAR.0, &
495 DELTA, VEL.RMS, Q.REAC.1, TIME.ITER, TOTAL.MIXED.MASS.1, CO.PASS)
496 CALL FIRE_RK2(A2, ZZ.MIXED, ZZ.UNMIXED, ZETA.2, ZETA.0, DT.SUB, 2, TMP.MIXED, TMP.UNMIXED, RHO.HAT, CELL.MASS, TAU.MIX,
497 PBAR.0, &
498 DELTA, VEL.RMS, Q.REAC.2, TIME.ITER, TOTAL.MIXED.MASS.2, CO.PASS)
499 CALL FIRE_RK2(A4, ZZ.MIXED, ZZ.UNMIXED, ZETA.4, ZETA.0, DT.SUB, 4, TMP.MIXED, TMP.UNMIXED, RHO.HAT, CELL.MASS, TAU.MIX,
500 PBAR.0, &
501 DELTA, VEL.RMS, Q.REAC.4, TIME.ITER, TOTAL.MIXED.MASS.4, CO.PASS)
502
503 ! Species Error Analysis
504 ERR_EST = MAXVAL(ABS((4._EB*A4-5._EB*A2+A1)))/45._EB ! FDS Tech Guide (E.7)
505
506 IF (N.FIXED.CHEMISTRY.SUBSTEPS<0) THEN
507 DT.SUB.NEW = MIN(MAX(DT.SUB*(ERR.TOL/(ERR_EST+TWO.EPSILON.EB))**(0.25._EB), DT.SUB.MIN), DT-DT.ITER) ! (E.8)
508 IF (ERR_EST<ERR.TOL) EXIT RICH.EX.LOOP
509 ENDIF
510
511 ENDDO RICH.EX.LOOP
512
513 ZZ.MIXED = (4._EB*A4-A2)*ONIH ! FDS Tech Guide (E.6)
514 Q.REAC.SUB = (4._EB*Q.REAC.4-Q.REAC.2)*ONIH
515 ZETA = (4._EB*ZETA.4-ZETA.2)*ONIH
516 ZETA.0 = ZETA
517
518 END SELECT INTEGRATOR.SELECT
519
520 ZZ.GET = ZETA*ZZ.UNMIXED + (1._EB-ZETA)*ZZ.MIXED ! FDS Tech Guide (5.29)
521 ZETA.INOUT = ZETA
522
523 DT.ITER = DT.ITER + DT.SUB
524 ITER = ITER + 1
525 IF (OUTPUT.CHEM.IT) CHEM.SUBIT.OUT = ITER
526
527 Q.REAC.SUM = Q.REAC.SUM + Q.REAC.SUB
528
529 ! Compute heat release rate
530 print *, 'Calculating heat release rate according to FDS-6.2.0'
531 Q.SUM = 0._EB
532 IF (MAXVAL(ABS(ZZ.GET-ZZ.TEMP)) > TWO.EPSILON.EB) THEN
533 Q.SUM = Q.SUM - RHO.IN*SUM(SPECIES.MIXTURE%H.F*(ZZ.GET-ZZ.TEMP)) ! FDS Tech Guide (5.14) replaced RHO(I,J,K) by
534 RHO.IN as in 6.6.0
535 ! print *, 'first if print Q.SUM, RHO.IN', Q.SUM, RHO.IN
536 ENDIF
537 IF (Q.CUM + Q.SUM > Q.UPPER*DT) THEN
538 Q.OUT = Q.UPPER
539 ZZ.GET = ZZ.TEMP + (Q.UPPER*DT/(Q.CUM + Q.SUM))*(ZZ.GET-ZZ.TEMP)
540 ! print *, 'second print Q.OUT, ZZ.GET', Q.OUT, ZZ.GET
541 EXIT INTEGRATION.LOOP
542 ELSE
543 Q.CUM = Q.CUM+Q.SUM
544 Q.OUT = Q.CUM/DT
545 ! print *, 'last else Q.CUM, Q.OUT', Q.CUM, Q.OUT
546 ENDIF
547
548 ! Total Variation (TV) scheme (accelerates integration for finite-rate equilibrium calculations)
549 ! See FDS Tech Guide Appendix E
550
551 IF (COMBUSTION.ODE.SOLVER==RK2.RICHARDSON .AND. N.REACTIONS>1) THEN
552 DO NS = 1, N.TRACKED.SPECIES
553 DO TV1 = 1, 3
554 ZZ.STORE(NS, TV1) = ZZ.STORE(NS, TV1+1)
555 ENDDO
556 ZZ.STORE(NS, 4) = ZZ.GET(NS)
557 ENDDO
558 TV.FLUCT(:) = .FALSE.
559 IF (ITER >= TV.ITER.MIN) THEN
560 SPECIES.LOOP.TV: DO NS = 1, N.TRACKED.SPECIES
561 DO TV1 = 1, 3
562 TV(TV1, NS) = ABS(ZZ.STORE(NS, TV1+1)-ZZ.STORE(NS, TV1))
563 ZZ.DIFF(TV1, NS) = ZZ.STORE(NS, TV1+1)-ZZ.STORE(NS, TV1)
564 ENDDO
565 IF (SUM(TV(:, NS)) < ERR.TOL .OR. SUM(TV(:, NS))) >= ABS(2.9._EB*SUM(ZZ.DIFF(:, NS)))) THEN ! FDS Tech Guide (E.10)
566 TV.FLUCT(NS) = .TRUE.

```

Source Code files for edited portions of FDS

```

566 ENDF
567 IF (ALL(TV_FLUCT)) EXIT INTEGRATION_LOOP
568 ENDDO SPECIES_LOOP_TV
569 ENDF
570 ENDF
571
572 IF ( DT_ITER > (DT+TWO_EPSILON_EB) ) CALL SHUTDOWN('ERROR: DT_ITER > DT in COMBUSTION_MODEL')
573 IF ( DT_ITER > (DT-TWO_EPSILON_EB) ) EXIT INTEGRATION_LOOP
574
575 ENDDO INTEGRATION_LOOP
576
577 !added as in 6.6.0
578 ! Extinction model
579
580 IF (SUPPRESSION) THEN
581 SELECT CASE (EXTINCT_MOD)
582 CASE (EXTINCTION_1); CALL EXTINCT_1(EXTINCT, ZZ_0, TMP_IN)
583 CASE (EXTINCTION_2); CALL EXTINCT_2(EXTINCT, ZZ_0, ZZ_MIXED, TMP_IN)
584 CASE (EXTINCTION_3); CALL EXTINCT_3(EXTINCT, ZZ_0, ZZ_MIXED, TMP_IN, CO_PASS)
585 END SELECT
586 ENDF
587
588 IF (.NOT.EXTINCT) THEN
589 EXIT CO_EXTINCT_LOOP
590 ELSE
591 ZZ_GET = ZZ_0
592 ZZ_STORE(:, :) = 0._EB
593 Q_OUT = 0._EB
594 CHL_R_OUT = 0._EB
595 CHEM_SUBIT_OUT = 0
596 REAC_SOURCE_TERM_OUT(:, :) = 0._EB
597 Q_REAC_OUT(:, :) = 0._EB
598 Q_REAC_SUM(:, :) = 0._EB
599 ENDF
600
601 ENDDO CO_EXTINCT_LOOP
602
603 !IF (REAC_SOURCE_CHECK) REAC_SOURCE_TERM(I,J,K,:) = (ZZ_UNMIXED-ZZ_GET)*CELL_MASS/DT ! store special output
604 ! quantity
605 ! Reaction rate-weighted radiative fraction
606
607 IF (SUM(Q_REAC_SUM)>TWO_EPSILON_EB) THEN
608 CHL_R_SUM=0._EB
609 DO NR=1,N_REACTIONS
610 RN=>REACTION(NR)
611 TIME_RAMP_FACTOR = EVALUATE_RAMP(T,0._EB,RN%RAMP_CHL_R_INDEX)
612 CHL_R_SUM = CHL_R_SUM + Q_REAC_SUM(NR)*RN%CHL_R*TIME_RAMP_FACTOR
613 ENDDO
614 CHL_R_OUT = CHL_R_SUM/(SUM(Q_REAC_SUM))
615 ENDF
616 CHL_R_OUT = MAX(CHL_R_MIN, MIN(CHL_R_MAX, CHL_R_OUT))
617
618 ! Store special diagnostic quantities
619
620 IF (REAC_SOURCE_CHECK) THEN
621 REAC_SOURCE_TERM_OUT = RHO_IN*(ZZ_GET-ZZ_0)/DT
622 Q_REAC_OUT = Q_REAC_SUM/CELL_VOLUME/DT
623 ENDF
624 add complete as in 6.6.0
625 END SUBROUTINE COMBUSTION_MODEL1 !changing names
626
627 SUBROUTINE COMBUSTION_MODEL(T,DT,ZZ_GET,Q_OUT,MIX.TIME.OUT,CHL_R_OUT,CHEM.SUBIT.OUT,REAC.SOURCE_TERM.OUT,
628 Q_REAC.OUT,&
629 TMP.IN,RHO.IN,MU.IN,KRES.IN,ZETA.INOUT,AIT.IN,PBAR.IN,DELTA,CELL.VOLUME)
630 USE COMP_FUNCTIONS, ONLY: SHUTDOWN
631 USE MATH_FUNCTIONS, ONLY: EVALUATE_RAMP
632 USE PHYSICAL_FUNCTIONS, ONLY: GET_AVERAGE_SPECIFIC_HEAT,GET_SPECIFIC_GAS_CONSTANT,GET_ENTHALPY
633 REAL(EB), INTENT(IN) :: T,DT,TMP.IN,RHO.IN,MU.IN,KRES.IN,PBAR.IN,AIT.IN,DELTA,CELL.VOLUME
634 REAL(EB), INTENT(OUT) :: Q_OUT,MIX.TIME.OUT,CHL_R_OUT,REAC.SOURCE_TERM.OUT(N.TRACKED.SPECIES),Q_REAC.OUT(
635 N.REACTIONS)
636 INTEGER, INTENT(OUT) :: CHEM.SUBIT.OUT
637 REAL(EB), INTENT(INOUT) :: ZZ_GET(1:N.TRACKED.SPECIES),ZETA.INOUT
638 REAL(EB) :: ERR_EST,ERR_TOL,A1(1:N.TRACKED.SPECIES),A2(1:N.TRACKED.SPECIES),A4(1:N.TRACKED.SPECIES),ZETA,ZETA_0,&
639 DT.SUB,DT.SUB.NEW,DT_ITER,ZZ_STORE(1:N.TRACKED.SPECIES,1:4),TV(1:3,1:N.TRACKED.SPECIES),CELL_MASS,&
640 ZZ_DIFF(1:3,1:N.TRACKED.SPECIES),ZZ_MIXED(1:N.TRACKED.SPECIES),ZZ_UNMIXED(1:N.TRACKED.SPECIES),&
641 PBAR_0,VEL_RMS,TAU.RES,ZZ_0(1:N.TRACKED.SPECIES),&
642 Q_REAC.SUB(1:N.REACTIONS),Q_REAC.1(1:N.REACTIONS),Q_REAC.2(1:N.REACTIONS),Q_REAC.4(1:N.REACTIONS),&
643 Q_REAC.SUM(1:N.REACTIONS),CHL_R.SUM,TIME.RAMP.FACTOR,&
644 TOTAL_MIXED_MASS_1,TOTAL_MIXED_MASS_2,TOTAL_MIXED_MASS_4,TOTAL_MIXED_MASS,&
645 ZETA_1,ZETA_2,ZETA_4,AIT.LOC
646 INTEGER :: NR,NS,ITER,TVI,RICH_ITER,TIME_ITER,RICH_ITER_MAX,CO_PASS,N.CO_PASS
647 INTEGER, PARAMETER :: TV_ITER_MIN=5
648 LOGICAL :: TV_FLUCT(1:N.TRACKED.SPECIES),EXTINCT
649 TYPE(REACTION_TYPE), POINTER :: RN=>NULL()
650 REAL(EB), PARAMETER :: C.U = 0.4_EB*0.1_EB*SQRT(1.5_EB) ! C.U*C.DEARDORFF/SQRT(2/3)
651 ZZ_0 = ZZ_GET

```

```

651 EXTINCT = .FALSE.
652
653 VEL_RMS = 0._EB
654 IF (FIXED_MIX.TIME>0._EB) THEN
655   MIX.TIME.OUT=FIXED_MIX.TIME
656 ELSE
657   TAU.D=0._EB
658   DO NR =1,N.REACTIONS
659     RN => REACTION(NR)
660     TAU.D = MAX(TAU.D,D.Z(MIN(4999,NINT(TMP.IN)),RN%FUEL.SMIX.INDEX))
661   ENDDO
662   TAU.D = DELTA**2/MAX(TAU.D,TWO.EPSILON.EB) ! FDS Tech Guide (5.20)
663   IF (LES) THEN
664     TAU.U = C.U*RHO.IN*DELTA**2/MAX(MU.IN,TWO.EPSILON.EB) ! FDS Tech Guide (5.24)
665     TAU.G = SQRT(2._EB*DELTA/(GRAV+1.E-10.EB)) ! FDS Tech Guide (5.2)
666     MIX.TIME.OUT= MAX(TAU.CHEM,MIN(TAU.D,TAU.U,TAU.G,TAU.FLAME)) ! FDS Tech Guide (5.22)
667     VEL_RMS = SQRT(TWIH)*MU.IN/(RHO.IN*C.DEARDORFF*DELTA)
668   ELSE
669     MIX.TIME.OUT= MAX(TAU.CHEM,TAU.D)
670   ENDIF
671 ENDIF
672
673 IF (TRANSPORT.UNMIXED.FRACTION) THEN
674   ZETA.0 = ZETA.INOUT
675 ELSE
676   ZETA.0 = INITIAL.UNMIXED.FRACTION
677 ENDIF
678 CELL.MASS = RHO.IN*CELL.VOLUME
679 TAU.RES = MU.IN/(RHO.IN*SC)/MAX(2._EB*KRES.IN,TWO.EPSILON.EB)
680
681 IF (REIGNITION.MODEL) THEN
682   AIT.LOC = AIT.IN
683 ELSE
684   AIT.LOC = 1.E20.EB
685 ENDIF
686
687 DT.SUB.MIN = DT/REAL(MAX.CHEMISTRY.ITERATIONS,EB)
688
689 N.CO.PASS = 1
690 IF (EXTINCT.MOD==EXTINCTION.3) N.CO.PASS = 2
691
692 CO.EXTINCT.LOOP: DO CO.PASS = 1,N.CO.PASS
693
694 ZZ.STORE(:, :) = 0._EB
695 Q.OUT = 0._EB
696 ITER= 0
697 DT.ITER = 0._EB
698 CHL.R.OUT = 0._EB
699 CHEM.SUBIT.OUT = 0
700 REAC.SOURCE.TERM.OUT(:) = 0._EB
701 Q.REAC.OUT(:) = 0._EB
702 Q.REAC.SUM(:) = 0._EB
703 IF (N.FIXED.CHEMISTRY.SUBSTEPS>0) THEN
704   DT.SUB = DT/REAL(N.FIXED.CHEMISTRY.SUBSTEPS,EB)
705   DT.SUB.NEW = DT.SUB
706   RICH.ITER.MAX = 1
707 ELSE
708   DT.SUB = DT
709   DT.SUB.NEW = DT
710   RICH.ITER.MAX = 5
711 ENDIF
712 ZZ.UNMIXED = ZZ.GET
713 ZZ.MIXED = ZZ.GET
714 A1 = ZZ.GET
715 A2 = ZZ.GET
716 A4 = ZZ.GET
717
718 ZETA = ZETA.0
719 RHO.HAT = RHO.IN
720 TMP.MIXED = TMP.IN
721 TMP.UNMIXED = TMP.IN
722 TAU.MIX = MIX.TIME.OUT
723 PBAR.0 = PBAR.IN
724
725 INTEGRATION.LOOP: DO TIME.ITER = 1,MAX.CHEMISTRY.ITERATIONS
726
727 IF (SUPPRESSION) CALL CHECK_AUTO_IGNITION(EXTINCT,TMP.MIXED,AIT.LOC,CO.PASS)
728 IF (EXTINCT) EXIT INTEGRATION.LOOP
729 INTEGRATOR.SELECT: SELECT CASE (COMBUSTION.ODE.SOLVER)
730
731 CASE (EXPLICIT.EULER) ! Simple chemistry
732
733 ! May be used with N.FIXED.CHEMISTRY.SUBSTEPS, but default mode is DT.SUB=DT for fast chemistry
734
735 CALL FIRE_FORWARD_EULER(ZZ.MIXED.NEW,ZZ.MIXED,ZZ.UNMIXED,ZETA,ZETA.0,DT.SUB,TMP.MIXED,TMP.UNMIXED,RHO.HAT,&
736 CELL.MASS,TAU.MIX,PBAR.0,DELTA,VEL.RMS,Q.REAC.SUB,TIME.ITER,TOTAL.MIXED.MASS,CO.PASS)
737 ZETA.0 = ZETA
738 ZZ.MIXED = ZZ.MIXED.NEW

```

## Source Code files for edited portions of FDS

```

739 IF (SIMPLE_CHEMISTRY .AND. N_FIXED_CHEMISTRY_SUBSTEPS<0 .AND. TIME_ITER>1) THEN
740 CALL SHUTDOWN('ERROR: Error in Simple Chemistry')
741 ENDIF
742
743 CASE (RK2) ! Runge-Kutta 2 stage (use in combination with N_FIXED_CHEMISTRY_SUBSTEPS)
744
745 CALL FIRE_RK2(ZZ_MIXED_NEW,ZZ_MIXED,ZZ_UNMIXED,ZETA,ZETA_0,DT_SUB,1,TMP_MIXED,TMP_UNMIXED,RHO_HAT,&
746 CELL_MASS,TAU_MIX,PBAR_0,DELTA,VEL_RMS,Q_REAC_SUB,TIME_ITER,TOTAL_MIXED_MASS,CO_PASS)
747 ZETA_0 = ZETA
748 ZZ_MIXED = ZZ_MIXED_NEW
749
750 CASE (RK3) ! Runge-Kutta 3 stage (use in combination with N_FIXED_CHEMISTRY_SUBSTEPS)
751
752 CALL FIRE_RK3(ZZ_MIXED_NEW,ZZ_MIXED,ZZ_UNMIXED,ZETA,ZETA_0,DT_SUB,1,TMP_MIXED,TMP_UNMIXED,RHO_HAT,&
753 CELL_MASS,TAU_MIX,PBAR_0,DELTA,VEL_RMS,Q_REAC_SUB,TIME_ITER,TOTAL_MIXED_MASS,CO_PASS)
754 ZETA_0 = ZETA
755 ZZ_MIXED = ZZ_MIXED_NEW
756
757 CASE (RK2_RICHARDSON) ! Finite-rate (or mixed finite-rate/fast) chemistry
758
759 ! May be used with N_FIXED_CHEMISTRY_SUBSTEPS, but default mode is to use error estimator and variable DT_SUB
760
761 ERR_TOL = RICHARDSON_ERROR_TOLERANCE
762 RICH_EX_LOOP: DO RICH_ITER = 1,RICH_ITER_MAX
763
764 DT_SUB = MIN(DT_SUB_NEW,DT-DT_ITER)
765
766 ! FDS Tech Guide (E.3), (E.4), (E.5)
767 CALL FIRE_RK2(A1,ZZ_MIXED,ZZ_UNMIXED,ZETA_1,ZETA_0,DT_SUB,1,TMP_MIXED,TMP_UNMIXED,RHO_HAT,CELL_MASS,TAU_MIX,
768 PBAR_0,&
769 DELTA,VEL_RMS,Q_REAC_1,TIME_ITER,TOTAL_MIXED_MASS_1,CO_PASS)
770 CALL FIRE_RK2(A2,ZZ_MIXED,ZZ_UNMIXED,ZETA_2,ZETA_0,DT_SUB,2,TMP_MIXED,TMP_UNMIXED,RHO_HAT,CELL_MASS,TAU_MIX,
771 PBAR_0,&
772 DELTA,VEL_RMS,Q_REAC_2,TIME_ITER,TOTAL_MIXED_MASS_2,CO_PASS)
773 CALL FIRE_RK2(A4,ZZ_MIXED,ZZ_UNMIXED,ZETA_4,ZETA_0,DT_SUB,4,TMP_MIXED,TMP_UNMIXED,RHO_HAT,CELL_MASS,TAU_MIX,
774 PBAR_0,&
775 DELTA,VEL_RMS,Q_REAC_4,TIME_ITER,TOTAL_MIXED_MASS_4,CO_PASS)
776
777 ! Species Error Analysis
778 ERR_EST = MAXVAL(ABS((4._EB*A4-5._EB*A2+A1)))/45._EB ! FDS Tech Guide (E.7)
779
780 IF (N_FIXED_CHEMISTRY_SUBSTEPS<0) THEN
781 DT_SUB_NEW = MIN(MAX(DT_SUB*(ERR_TOL/(ERR_EST+TWO_EPSILON_EB)))*(0.25._EB),DT_SUB_MIN),DT-DT_ITER) ! (E.8)
782 IF (ERR_EST<ERR_TOL) EXIT RICH_EX_LOOP
783 ENDIF
784
785 ENDDO RICH_EX_LOOP
786
787 ZZ_MIXED = (4._EB*A4-A2)*ONIH ! FDS Tech Guide (E.6)
788 Q_REAC_SUB = (4._EB*Q_REAC_4-Q_REAC_2)*ONIH
789 ZETA = (4._EB*ZETA_4-ZETA_2)*ONIH
790 ZETA_0 = ZETA
791
792 !! debug
793 !ZETA_0 = ZETA
794 !ZZ_MIXED = A4
795 !Q_REAC_SUB = Q_REAC_4
796
797 END SELECT INTEGRATOR_SELECT
798
799 ZZ_GET = ZETA*ZZ_UNMIXED + (1._EB-ZETA)*ZZ_MIXED ! FDS Tech Guide (5.29)
800 ZETA_INOUT = ZETA
801
802 DT_ITER = DT_ITER + DT_SUB
803 ITER = ITER + 1
804 IF (OUTPUT_CHEM_IT) CHEM_SUBIT_OUT = ITER
805
806 Q_REAC_SUM = Q_REAC_SUM + Q_REAC_SUB
807
808 ! Total Variation (TV) scheme (accelerates integration for finite-rate equilibrium calculations)
809 ! See FDS Tech Guide Appendix E
810
811 IF (COMBUSTION_ODE_SOLVER==RK2_RICHARDSON .AND. N_REACTIONS>1) THEN
812 DO NS = 1,N_TRACKED_SPECIES
813 DO TVI = 1,3
814 ZZ_STORE(NS,TVI)=ZZ_STORE(NS,TVI+1)
815 ENDDO
816 ZZ_STORE(NS,4) = ZZ_GET(NS)
817 ENDDO
818 TV_FLUCT(:) = .FALSE.
819 IF (ITER >= TV_ITER_MIN) THEN
820 SPECIES_LOOP_TV: DO NS = 1,N_TRACKED_SPECIES
821 DO TVI = 1,3
822 TV(TVI,NS) = ABS(ZZ_STORE(NS,TVI+1)-ZZ_STORE(NS,TVI))
823 ZZ_DIFF(TVI,NS) = ZZ_STORE(NS,TVI+1)-ZZ_STORE(NS,TVI)
824 ENDDO
825 IF (SUM(TV(:,NS)) < ERR_TOL .OR. SUM(TV(:,NS)) >= ABS(2.9._EB*SUM(ZZ_DIFF(:,NS)))) THEN ! FDS Tech Guide (E.10)
826 TV_FLUCT(NS) = .TRUE.

```

```

824   ENDIF
825   IF ( ALL(TV_FLUCT) ) EXIT INTEGRATION_LOOP
826   ENDDO SPECIES_LOOP_TV
827   ENDIF
828   ENDIF
829
830   IF ( DT_ITER > (DT+TWO_EPSILON_EB) ) CALL SHUTDOWN('ERROR: DT_ITER > DT in COMBUSTION_MODEL')
831   IF ( DT_ITER > (DT-TWO_EPSILON_EB) ) EXIT INTEGRATION_LOOP
832
833   ENDDO INTEGRATION_LOOP
834
835   ! Compute heat release rate
836
837   Q_OUT = -RHO_IN * SUM(SPECIES_MIXTURE%HLF*(ZZ_GET-ZZ_0))/DT ! FDS Tech Guide (5.14)
838   !print *, 'first if print Q_OUT', Q_OUT
839
840   ! Extinction model
841
842   IF (SUPPRESSION) THEN
843     SELECT CASE (EXTINCT_MOD)
844     CASE (EXTINCTION_1); CALL EXTINCT_1(EXTINCT, ZZ_0, TMP_IN)
845     CASE (EXTINCTION_2); CALL EXTINCT_2(EXTINCT, ZZ_0, ZZ_MIXED, TMP_IN)
846     CASE (EXTINCTION_3); CALL EXTINCT_3(EXTINCT, ZZ_0, ZZ_MIXED, TMP_IN, CO_PASS)
847   END SELECT
848   ENDIF
849
850   IF (.NOT.EXTINCT) THEN
851     EXIT CO_EXTINCT_LOOP
852   ELSE
853     ZZ_GET = ZZ_0
854     ZZ_STORE(:, :) = 0. _EB
855     Q_OUT = 0. _EB
856     CHL_R_OUT = 0. _EB
857     CHEM_SUBIT_OUT = 0
858     REAC_SOURCE_TERM_OUT(:) = 0. _EB
859     Q_REAC_OUT(:) = 0. _EB
860     Q_REAC_SUM(:) = 0. _EB
861   ENDIF
862
863   ENDDO CO_EXTINCT_LOOP
864
865   ! Reaction rate-weighted radiative fraction
866
867   IF (SUM(Q_REAC_SUM) > TWO_EPSILON_EB) THEN
868     CHL_R_SUM = 0. _EB
869     DO NR=1, N_REACTIONS
870       RN => REACTION(NR)
871       TIME_RAMP_FACTOR = EVALUATE_RAMP(T, 0. _EB, RN%RAMP_CHL_R_INDEX)
872       CHL_R_SUM = CHL_R_SUM + Q_REAC_SUM(NR)*RN%CHL_R*TIME_RAMP_FACTOR
873     ENDDO
874     CHL_R_OUT = CHL_R_SUM/(SUM(Q_REAC_SUM))
875   ENDIF
876   CHL_R_OUT = MAX(CHL_R_MIN, MIN(CHL_R_MAX, CHL_R_OUT))
877
878   ! Store special diagnostic quantities
879
880   IF (REAC_SOURCE_CHECK) THEN
881     REAC_SOURCE_TERM_OUT = RHO_IN*(ZZ_GET-ZZ_0)/DT
882     Q_REAC_OUT = Q_REAC_SUM/CELL_VOLUME/DT
883   ENDIF
884
885   END SUBROUTINE COMBUSTION_MODEL
886
887
888   SUBROUTINE CHECK_AUTO_IGNITION(EXTINCT, TMP_MIXED, AIT_IN, CO_PASS)
889   LOGICAL, INTENT(INOUT) :: EXTINCT
890   INTEGER, INTENT(IN) :: CO_PASS
891   REAL(EB), INTENT(IN) :: TMP_MIXED, AIT_IN
892   INTEGER :: NR
893   REAL(EB) :: AIT_LOC
894   TYPE(REACTION_TYPE), POINTER :: RN => NULL()
895
896   EXTINCT = .TRUE.
897
898   SELECT CASE (EXTINCT_MOD)
899   CASE DEFAULT
900     ! if ANY reaction exceeds AIT, allow all reactions and proceed to EXTINCTION MODEL
901     ! note: here we include finite-rate reactions, else combustion model will exit
902     ! integration loop (as presently coded)
903     REACTION_LOOP: DO NR=1, N_REACTIONS
904       RN => REACTION(NR)
905       IF (AIT_IN < 1.E10_EB) THEN
906         AIT_LOC = AIT_IN
907       ELSE
908         AIT_LOC = RN%AUTO_IGNITION_TEMPERATURE
909       ENDIF
910     IF ( TMP_MIXED > AIT_LOC ) THEN
911       EXTINCT = .FALSE.

```

```

912 EXIT REACTION_LOOP
913 ENDIF
914 ENDDO REACTION_LOOP
915
916 CASE(EXTINCTION_3)
917 ! special case: CO production with 2 fast reactions
918 ! CO_PASS==1 -> check hydrocarbon AIT
919 ! CO_PASS==2 -> check CO AIT (not subject to pilot zone ignition [AIT.IN < 1.E10.EB])
920 RN => REACTION(CO_PASS)
921 IF (CO_PASS==1 .AND. AIT.IN < 1.E10.EB) THEN
922 AIT.LOC = AIT.IN
923 ELSE
924 AIT.LOC = RN%AUTO_IGNITION_TEMPERATURE
925 ENDIF
926 IF ( TMP_MIXED > AIT.LOC ) EXTINGT = .FALSE.
927
928 END SELECT
929
930 END SUBROUTINE CHECK_AUTO_IGNITION
931
932
933 SUBROUTINE EXTINGT_1(EXTINGT, ZZ.IN, TMP.IN)
934 USE PHYSICAL_FUNCTIONS, ONLY: GET_AVERAGE_SPECIFIC_HEAT
935 REAL(EB), INTENT(IN) :: TMP.IN, ZZ.IN(1:N_TRACKED_SPECIES)
936 LOGICAL, INTENT(INOUT) :: EXTINGT
937 REAL(EB) :: Y_O2, Y_O2_CRIT, CPBAR
938 INTEGER :: NR
939 TYPE(REACTION_TYPE), POINTER :: RN=>NULL()
940
941 EXTINGT = .FALSE.
942 REACTION_LOOP: DO NR=1, N_REACTIONS
943 RN => REACTION(NR)
944 IF (.NOT. RN%FAST_CHEMISTRY) CYCLE REACTION_LOOP
945 CALL GET_AVERAGE_SPECIFIC_HEAT(ZZ.IN, CPBAR, TMP.IN)
946 Y_O2 = ZZ.IN(RN%AIR_SMIX_INDEX)
947 Y_O2_CRIT = CPBAR*(RN%CRIT_FLAME_TMP-TMP.IN)/RN%EPUMO2
948 IF (Y_O2 < Y_O2_CRIT) EXTINGT = .TRUE.
949 ENDDO REACTION_LOOP
950
951 END SUBROUTINE EXTINGT_1
952
953
954 SUBROUTINE EXTINGT_2(EXTINGT, ZZ_0.IN, ZZ.IN, TMP.IN)
955 USE PHYSICAL_FUNCTIONS, ONLY: GET_ENTHALPY
956 REAL(EB), INTENT(IN) :: TMP.IN, ZZ_0.IN(1:N_TRACKED_SPECIES), ZZ.IN(1:N_TRACKED_SPECIES)
957 LOGICAL, INTENT(INOUT) :: EXTINGT
958 REAL(EB) :: ZZ_HAT_0(1:N_TRACKED_SPECIES), ZZ_HAT(1:N_TRACKED_SPECIES), H_0, H_CRIT
959 INTEGER :: NS
960 TYPE(REACTION_TYPE), POINTER :: R1=>NULL()
961
962 IF (N_REACTIONS /= 1 .OR. .NOT. REACTION(1)%FAST_CHEMISTRY) RETURN
963 R1 => REACTION(1)
964
965 DO NS = 1, N_TRACKED_SPECIES
966 IF (NS==R1%FUEL_SMIX_INDEX) THEN
967 ZZ_HAT_0(NS) = ZZ_0.IN(NS) ! FDS Tech Guide (5.15)
968 ZZ_HAT(NS) = ZZ.IN(NS) ! FDS Tech Guide (5.18)
969 ELSEIF (NS==R1%AIR_SMIX_INDEX) THEN
970 ZZ_HAT_0(NS) = ZZ_0.IN(NS) - ZZ.IN(NS) ! FDS Tech Guide (5.16)
971 ZZ_HAT(NS) = 0..EB ! FDS Tech Guide (5.19)
972 ELSE
973 ! FDS Tech Guide (5.17)
974 ZZ_HAT_0(NS) = ( (ZZ_0.IN(R1%AIR_SMIX_INDEX) - ZZ.IN(R1%AIR_SMIX_INDEX)) / ZZ_0.IN(R1%AIR_SMIX_INDEX) ) * ZZ_0.IN(NS)
975 ZZ_HAT(NS) = ZZ_HAT_0(NS) + ZZ.IN(NS) - ZZ_0.IN(NS) ! FDS Tech Guide (5.20)
976 ENDIF
977 ENDDO
978
979 ZZ_HAT_0 = ZZ_HAT_0/SUM(ZZ_HAT_0)
980 ZZ_HAT = ZZ_HAT/SUM(ZZ_HAT)
981
982 ! See if enough energy is released to raise the fuel and required "air" temperatures above the critical flame temp.
983
984 CALL GET_ENTHALPY(ZZ_HAT_0, H_0, TMP.IN) ! H of reactants participating in reaction (includes chemical enthalpy)
985 CALL GET_ENTHALPY(ZZ_HAT, H_CRIT, REACTION(1)%CRIT_FLAME_TMP) ! H of products at the critical flame temperature
986 IF (H_0 < H_CRIT) EXTINGT = .TRUE. ! FDS Tech Guide (5.21)
987
988 END SUBROUTINE EXTINGT_2
989
990
991 SUBROUTINE EXTINGT_3(EXTINGT, ZZ_0.IN, ZZ.IN, TMP.IN, CO_PASS)
992 USE PHYSICAL_FUNCTIONS, ONLY: GET_ENTHALPY
993 REAL(EB), INTENT(IN) :: TMP.IN, ZZ_0.IN(1:N_TRACKED_SPECIES), ZZ.IN(1:N_TRACKED_SPECIES)
994 INTEGER, INTENT(IN) :: CO_PASS
995 LOGICAL, INTENT(INOUT) :: EXTINGT
996 REAL(EB) :: ZZ_HAT_0(1:N_TRACKED_SPECIES), ZZ_HAT(1:N_TRACKED_SPECIES), H_0, H_CRIT
997 INTEGER :: NS

```

Source Code files for edited portions of FDS

```

998 TYPE(REACTION_TYPE), POINTER :: R1=>NULL(), R2=>NULL()
999
1000 ! R1: Fuel + O2 => CO + Products
1001 ! R2: CO + (1/2)O2 => CO2
1002 ! extinction model:
1003 ! 1. (first pass) Evaluate EXTINGT based on R1 + R2
1004 ! 2. (second pass, EXTINGT=T on first pass) Evaluate EXTINGT based on R2
1005
1006 R1 => REACTION(1)
1007 R2 => REACTION(2)
1008
1009 DO NS = 1, N_TRACKED_SPECIES
1010 IF (NS==R1%FUEL.SMIX_INDEX .OR. NS==R2%FUEL.SMIX_INDEX) THEN
1011 ZZ_HAT_0(NS) = ZZ_0.IN(NS)
1012 ZZ_HAT(NS) = ZZ.IN(NS)
1013 ELSEIF (NS==R1%AIR.SMIX_INDEX) THEN
1014 ZZ_HAT_0(NS) = ZZ_0.IN(NS) - ZZ.IN(NS)
1015 ZZ_HAT(NS) = 0..EB
1016 ELSE
1017 ZZ_HAT_0(NS) = ( (ZZ_0.IN(R1%AIR.SMIX_INDEX) - ZZ.IN(R1%AIR.SMIX_INDEX)) / ZZ_0.IN(R1%AIR.SMIX_INDEX) ) * ZZ_0.IN(NS)
1018 ZZ_HAT(NS) = ZZ.IN(NS) - ZZ_0.IN(NS) + ZZ_HAT_0(NS)
1019 ENDIF
1020 ENDDO
1021
1022 ZZ_HAT_0 = ZZ_HAT_0/SUM(ZZ_HAT_0)
1023 ZZ_HAT = ZZ_HAT/SUM(ZZ_HAT)
1024
1025 ! See if enough energy is released to raise the fuel and required "air" temperatures above the critical flame
    temp.
1026
1027 CALL GET_ENTHALPY(ZZ_HAT_0, H_0, TMP.IN) ! H of reactants participating in reaction (includes chemical enthalpy)
1028 CALL GET_ENTHALPY(ZZ_HAT, H_CRIT, REACTION(CO.PASS)%CRIT_FLAME.TMP) ! H of products at the critical flame
    temperature
1029 IF (H_0 < H_CRIT) EXTINGT = .TRUE.
1030
1031 END SUBROUTINE EXTINGT_3
1032
1033 SUBROUTINE FIRE_FORWARD_EULER_1(ZZ_OUT, ZETA_OUT, ZZ_IN, ZETA_IN, DT.LOC, TMP.MIXED, RHO.HAT, ZZ.UNMIXED, CELL.MASS,
    TAU.MIX) !Subroutine FIRE_FORWARD_EULER_1 for 6.2.0
1034 USE COMP_FUNCTIONS, ONLY: SHUTDOWN
1035 USE PHYSICAL_FUNCTIONS, ONLY: GET_REALIZABLE_MF, GET_AVERAGE_SPECIFIC_HEAT
1036 !USE RADCONS, ONLY: RADIATIVE_FRACTION_1 !changed name RADIATIVE_FRACTION_1
1037 REAL(EB) :: RADIATIVE_FRACTION !added this as required
1038 REAL(EB), INTENT(IN) :: ZZ.IN(1:N_TRACKED_SPECIES), ZETA.IN, DT.LOC, RHO.HAT, ZZ.UNMIXED(1:N_TRACKED_SPECIES),
    CELL.MASS,&
1039 TAU.MIX
1040 REAL(EB), INTENT(OUT) :: ZZ.OUT(1:N_TRACKED_SPECIES), ZETA.OUT
1041 REAL(EB), INTENT(INOUT) :: TMP.MIXED
1042 REAL(EB) :: ZZ_0(1:N_TRACKED_SPECIES), ZZ.NEW(1:N_TRACKED_SPECIES), DZZ(1:N_TRACKED_SPECIES), UNMIXED.MASS_0(1:
    N_TRACKED_SPECIES),&
1043 BOUNDEDNESS.CORRECTION, MIXED.MASS(1:N_TRACKED_SPECIES), MIXED.MASS_0(1:N_TRACKED_SPECIES), TOTAL.MIXED.MASS, Q.TEMP,
    CPBAR
1044 INTEGER :: SR
1045 INTEGER, PARAMETER :: INFINITELY_FAST=1, FINITE_RATE=2
1046 LOGICAL :: TEMPERATURE_DEPENDENT_REACTION=.FALSE.
1047
1048 ZETA_OUT = ZETA.IN*EXP(-DT.LOC/TAU.MIX) ! FDS Tech Guide (5.29)
1049 IF (ZETA_OUT < TWO.EPSILON.EB) ZETA_OUT = 0..EB
1050 MIXED.MASS_0 = CELL.MASS*ZZ.IN
1051 UNMIXED.MASS_0 = CELL.MASS*ZZ.UNMIXED
1052 MIXED.MASS = MAX(0..EB, MIXED.MASS_0 - (ZETA_OUT - ZETA.IN)*UNMIXED.MASS_0) ! FDS Tech Guide (5.37)
1053 TOTAL.MIXED.MASS = SUM(MIXED.MASS)
1054 ZZ_0 = MIXED.MASS/MAX(TOTAL.MIXED.MASS, TWO.EPSILON.EB) ! FDS Tech Guide (5.35)
1055
1056 IF (ANY(REACTION(:)%FAST_CHEMISTRY)) THEN
1057 DO SR = 0, N_SERIES_REACTIONS
1058 CALL REACTION_RATE_1(DZZ, ZZ_0, DT.LOC, RHO.HAT, TMP.MIXED, INFINITELY_FAST) !changed name to REACTION_RATE_1 for
    6.2.0
1059 ZZ.NEW = ZZ_0 + DZZ ! test Forward Euler step (5.53)
1060 BOUNDEDNESS.CORRECTION = FUNC.BCOR(ZZ_0, ZZ.NEW) ! Reaction rate boundedness correction
1061 ZZ.NEW = ZZ_0 + DZZ*BOUNDEDNESS.CORRECTION ! corrected FE step for all species (5.54)
1062 ZZ_0 = ZZ.NEW
1063 ENDDO
1064 ENDIF
1065
1066 IF (.NOT.ALL(REACTION(:)%FAST_CHEMISTRY)) THEN
1067 CALL REACTION_RATE_1(DZZ, ZZ_0, DT.LOC, RHO.HAT, TMP.MIXED, FINITE_RATE) !changed name to REACTION_RATE_1 for 6.2.0
1068 ZZ.NEW = ZZ_0 + DZZ
1069 BOUNDEDNESS.CORRECTION = FUNC.BCOR(ZZ_0, ZZ.NEW)
1070 ZZ.NEW = ZZ_0 + DZZ*BOUNDEDNESS.CORRECTION
1071 IF (TEMPERATURE_DEPENDENT_REACTION) THEN
1072 Q.TEMP = SUM(SPECIES_MIXTURE%h.F*DZZ*BOUNDEDNESS.CORRECTION)
1073 CALL GET_AVERAGE_SPECIFIC_HEAT(ZZ.NEW, CPBAR, TMP.MIXED)
1074 TMP.MIXED = TMP.MIXED + DT.LOC*(1..EB-RADIATIVE_FRACTION)*Q.TEMP/CPBAR
1075 ENDIF
1076 ENDIF
1077

```



Source Code files for edited portions of FDS

```

1078 ! Enforce realizability on mass fractions
1079
1080 CALL GET_REALIZABLE_MF(ZZ_NEW)
1081
1082 ZZ_OUT = ZZ_NEW
1083
1084 END SUBROUTINE FIRE_FORWARD_EULER_1 !changing names
1085
1086 SUBROUTINE FIRE_FORWARD_EULER(ZZ_OUT, ZZ_IN, ZZ_UNMIXED, ZETA_OUT, ZETA_IN, DT_LOC, TMP_MIXED, TMP_UNMIXED, RHO_HAT,
1087     CELL_MASS, TAU_MIX, &
1088     PBAR_0, DELTA, VEL_RMS, Q_REAC_LOC, SUB_IT, TOTAL_MIXED_MASS, CO_PASS)
1089 USE COMP_FUNCTIONS, ONLY: SHUTDOWN
1090 USE PHYSICAL_FUNCTIONS, ONLY: GET_REALIZABLE_MF, GET_AVERAGE_SPECIFIC_HEAT
1091 REAL(EB), INTENT(IN) :: ZZ_IN(1:N_TRACKED_SPECIES), ZETA_IN, DT_LOC, RHO_HAT, ZZ_UNMIXED(1:N_TRACKED_SPECIES),
1092     CELL_MASS, TAU_MIX, &
1093     PBAR_0, DELTA, VEL_RMS, TMP_UNMIXED
1094 INTEGER, INTENT(IN) :: SUB_IT, CO_PASS
1095 REAL(EB), INTENT(OUT) :: ZZ_OUT(1:N_TRACKED_SPECIES), ZETA_OUT, Q_REAC_LOC(1:N_REACTIONS), TOTAL_MIXED_MASS
1096 REAL(EB), INTENT(INOUT) :: TMP_MIXED
1097 REAL(EB) :: ZZ_0(1:N_TRACKED_SPECIES), ZZ_NEW(1:N_TRACKED_SPECIES), DZZ(1:N_TRACKED_SPECIES), &
1098     MIXED_MASS(1:N_TRACKED_SPECIES), MIXED_MASS_0(1:N_TRACKED_SPECIES), &
1099     Q_REAC_OUT(1:N_REACTIONS), TOTAL_MIXED_MASS_0
1100 INTEGER, PARAMETER :: INFINITELY_FAST=1, FINITE_RATE=2
1101
1102 ! Determine initial state of mixed reactor zone
1103 TOTAL_MIXED_MASS_0 = (1._EB-ZETA_IN)*CELL_MASS
1104 MIXED_MASS_0 = ZZ_IN*TOTAL_MIXED_MASS_0
1105
1106 ! Mixing step
1107 ZETA_OUT = MAX(0._EB, ZETA_IN*EXP(-DT_LOC/TAU_MIX)) ! FDS Tech Guide (5.28)
1108 TOTAL_MIXED_MASS = (1._EB-ZETA_OUT)*CELL_MASS
1109 MIXED_MASS = MAX(0._EB, MIXED_MASS_0 - (ZETA_OUT - ZETA_IN)*ZZ_UNMIXED*CELL_MASS) ! after mixing step, FDS Tech
1110     Guide (5.36)
1111 ZZ_0 = MIXED_MASS/MAX(TOTAL_MIXED_MASS, TWO_EPSILON_EB) ! FDS Tech Guide (5.37)
1112
1113 ! Enforce realizability on mass fractions
1114
1115 CALL GET_REALIZABLE_MF(ZZ_0)
1116
1117 Q_REAC_LOC(:) = 0._EB
1118
1119 ! Removed TEMPERATURE_DEPENDENT_REACTION until other bugs are sorted out
1120 TMP_MIXED = TMP_UNMIXED
1121 IF (ANY(REACTION(:)%FAST_CHEMISTRY)) THEN
1122 CALL REACTION_RATE(DZZ, ZZ_0, DT_LOC, RHO_HAT, TMP_MIXED, INFINITELY_FAST, PBAR_0, DELTA, VEL_RMS, Q_REAC_OUT, SUB_IT,
1123     CO_PASS)
1124 ZZ_NEW = ZZ_0 + DZZ
1125 ZZ_0 = ZZ_NEW
1126 Q_REAC_LOC = Q_REAC_LOC + Q_REAC_OUT*TOTAL_MIXED_MASS
1127 ENDIF
1128 IF (.NOT.ALL(REACTION(:)%FAST_CHEMISTRY)) THEN
1129 CALL REACTION_RATE(DZZ, ZZ_0, DT_LOC, RHO_HAT, TMP_MIXED, FINITE_RATE, PBAR_0, DELTA, VEL_RMS, Q_REAC_OUT, SUB_IT, CO_PASS)
1130 ZZ_NEW = ZZ_0 + DZZ
1131 Q_REAC_LOC = Q_REAC_LOC + Q_REAC_OUT*TOTAL_MIXED_MASS
1132 ENDIF
1133
1134 ! Enforce realizability on mass fractions
1135
1136 CALL GET_REALIZABLE_MF(ZZ_NEW)
1137
1138 ZZ_OUT = ZZ_NEW
1139
1140 END SUBROUTINE FIRE_FORWARD_EULER
1141
1142 REAL(EB) FUNCTION FUNC_BCOR(ZZ_0, ZZ_NEW)
1143 ! This function finds a correction for reaction rates such that all species remain bounded.
1144
1145 REAL(EB), INTENT(IN) :: ZZ_0(1:N_TRACKED_SPECIES), ZZ_NEW(1:N_TRACKED_SPECIES)
1146 REAL(EB) :: BCOR, DZ_IB, DZ_OB
1147 INTEGER :: NS
1148
1149 !print *, 'BCOR function'
1150
1151 BCOR = 1._EB
1152 DO NS=1,N_TRACKED_SPECIES
1153 IF (ZZ_NEW(NS)<0._EB) THEN ! FDS Tech Guide (5.55)
1154 DZ_IB=ZZ_0(NS) ! DZ "in bounds"
1155 DZ_OB=ABS(ZZ_NEW(NS)) ! DZ "out of bounds"
1156 BCOR = MIN( BCOR, DZ_IB/MAX(DZ_IB+DZ_OB, TWO_EPSILON_EB) )
1157 ENDIF
1158 IF (ZZ_NEW(NS)>1._EB) THEN ! FDS Tech Guide (5.55)
1159 DZ_IB=1._EB-ZZ_0(NS)
1160 DZ_OB=ZZ_NEW(NS)-1._EB
1161 BCOR = MIN( BCOR, DZ_IB/MAX(DZ_IB+DZ_OB, TWO_EPSILON_EB) )

```

Source Code files for edited portions of FDS

```

1162 ENDIF
1163 ENDDO
1164 FUNC.BCOR = BCOR
1165
1166 END FUNCTION FUNC.BCOR
1167
1168 SUBROUTINE FIRE_RK2.1(ZZ_OUT,ZETA_OUT,ZZ_IN,ZETA_IN,DT.SUB,N.INC,TMP.MIXED,RHO.HAT,ZZ.UNMIXED,CELL.MASS,TAU.MIX)
1169     !changed name to FIRE_RK2.1 for 6.2.0
1170     ! This function uses RK2 to integrate ZZ.O from t=0 to t=DT.SUB in increments of DT.LOC=DT.SUB/N.JNC
1171     REAL(EB), INTENT(IN) :: ZZ_IN(1:N.TRACKED.SPECIES),DT.SUB,ZETA_IN,RHO.HAT,ZZ.UNMIXED(1:N.TRACKED.SPECIES),
1172     CELL.MASS,&
1173     TAU.MIX
1174     REAL(EB), INTENT(OUT) :: ZZ_OUT(1:N.TRACKED.SPECIES),ZETA_OUT
1175     REAL(EB), INTENT(INOUT) :: TMP.MIXED
1176     INTEGER, INTENT(IN) :: N.INC
1177     REAL(EB) :: DT.LOC,ZZ.0(1:N.TRACKED.SPECIES),ZZ.1(1:N.TRACKED.SPECIES),ZZ.2(1:N.TRACKED.SPECIES),ZETA.0,ZETA.1,
1178     ZETA.2
1179     INTEGER :: N
1180     DT.LOC = DT.SUB/REAL(N.INC,EB)
1181     ZZ.0 = ZZ_IN
1182     ZETA.0 = ZETA_IN
1183     DO N=1,N.INC
1184         CALL FIRE_FORWARD_EULER.1(ZZ.1,ZETA.1,ZZ.0,ZETA.0,DT.LOC,TMP.MIXED,RHO.HAT,ZZ.UNMIXED,CELL.MASS,TAU.MIX) !changed
1185         name
1186         CALL FIRE_FORWARD_EULER.1(ZZ.2,ZETA.2,ZZ.1,ZETA.1,DT.LOC,TMP.MIXED,RHO.HAT,ZZ.UNMIXED,CELL.MASS,TAU.MIX) !changed
1187         name
1188         ZZ.OUT = 0.5.EB*(ZZ.0 + ZZ.2)
1189         ZZ.0 = ZZ.OUT
1190         ZETA.OUT = ZETA.1
1191         ZETA.0 = ZETA.OUT
1192     ENDDO
1193 END SUBROUTINE FIRE_RK2.1 !changed name
1194
1195 SUBROUTINE FIRE_RK2(ZZ_OUT,ZZ_IN,ZZ.UNMIXED,ZETA_OUT,ZETA_IN,DT.SUB,N.INC,TMP.MIXED,TMP.UNMIXED,RHO.HAT,CELL.MASS
1196     ,TAU.MIX,&
1197     PBAR.0,DELTA,VEL.RMS,Q.REAC.OUT,SUB.IT,TOTAL.MIXED.MASS.OUT,CO.PASS)
1198     ! This function uses RK2 to integrate ZZ.O from t=0 to t=DT.SUB in increments of DT.LOC=DT.SUB/N.JNC
1199     REAL(EB), INTENT(IN) :: ZZ_IN(1:N.TRACKED.SPECIES),DT.SUB,ZETA_IN,RHO.HAT,ZZ.UNMIXED(1:N.TRACKED.SPECIES),
1200     CELL.MASS,&
1201     TAU.MIX,PBAR.0,DELTA,VEL.RMS,TMP.UNMIXED
1202     REAL(EB), INTENT(OUT) :: ZZ_OUT(1:N.TRACKED.SPECIES),ZETA_OUT,Q.REAC.OUT(1:N.REACTIONS),TOTAL.MIXED.MASS.OUT
1203     REAL(EB), INTENT(INOUT) :: TMP.MIXED
1204     INTEGER, INTENT(IN) :: N.INC,SUB.IT,CO.PASS
1205     REAL(EB) :: DT.LOC,ZZ.0(1:N.TRACKED.SPECIES),ZZ.1(1:N.TRACKED.SPECIES),ZZ.2(1:N.TRACKED.SPECIES),ZETA.0,ZETA.1,
1206     ZETA.2,&
1207     Q.REAC.1(1:N.REACTIONS),Q.REAC.2(1:N.REACTIONS),TOTAL.MIXED.MASS.0,TOTAL.MIXED.MASS.1,TOTAL.MIXED.MASS.2
1208     INTEGER :: N
1209     DT.LOC = DT.SUB/REAL(N.INC,EB)
1210     ZZ.0 = ZZ_IN
1211     ZETA.0 = ZETA_IN
1212     Q.REAC.OUT(:) = 0.EB
1213     TOTAL.MIXED.MASS.0 = (1.EB-ZETA.0)*CELL.MASS
1214
1215     DO N=1,N.INC
1216         CALL FIRE_FORWARD_EULER(ZZ.1,ZZ.0,ZZ.UNMIXED,ZETA.1,ZETA.0,DT.LOC,TMP.MIXED,TMP.UNMIXED,RHO.HAT,CELL.MASS,TAU.MIX
1217         ,&
1218         PBAR.0,DELTA,VEL.RMS,Q.REAC.1,SUB.IT,TOTAL.MIXED.MASS.1,CO.PASS)
1219         CALL FIRE_FORWARD_EULER(ZZ.2,ZZ.1,ZZ.UNMIXED,ZETA.2,ZETA.1,DT.LOC,TMP.MIXED,TMP.UNMIXED,RHO.HAT,CELL.MASS,TAU.MIX
1220         ,&
1221         PBAR.0,DELTA,VEL.RMS,Q.REAC.2,SUB.IT,TOTAL.MIXED.MASS.2,CO.PASS)
1222         IF (TOTAL.MIXED.MASS.2>TWO.EPSILON.EB) THEN
1223             ZZ.OUT = 0.5.EB*(ZZ.0*TOTAL.MIXED.MASS.0 + ZZ.2*TOTAL.MIXED.MASS.2)
1224             TOTAL.MIXED.MASS.OUT = SUM(ZZ.OUT)
1225             ZZ.OUT = ZZ.OUT/TOTAL.MIXED.MASS.OUT
1226         ELSE
1227             ZZ.OUT = ZZ.0
1228         ENDIF
1229         ZETA.OUT = MAX(0.EB,1.EB-TOTAL.MIXED.MASS.OUT/CELL.MASS)
1230         Q.REAC.OUT = Q.REAC.OUT + 0.5.EB*(Q.REAC.1+Q.REAC.2)
1231         ZZ.0 = ZZ.OUT
1232         ZETA.0 = ZETA.OUT
1233         TOTAL.MIXED.MASS.0 = TOTAL.MIXED.MASS.OUT
1234     ENDDO
1235 END SUBROUTINE FIRE_RK2

```

Source Code files for edited portions of FDS

```

1240
1241 SUBROUTINE FIRE_RK3(ZZ.OUT,ZZ.IN,ZZ.UNMIXED,ZETA.OUT,ZETA.IN,DT.SUB,N.INC,TMP.MIXED,TMP.UNMIXED,RHO.HAT,CELL.MASS
      ,TAU.MIX,&
1242 PBAR.0,DELTA,VEL.RMS,Q.REAC.OUT,SUB.IT,TOTAL.MIXED.MASS.OUT,CO.PASS)
1243
1244 ! This function uses SSP RK3. See Gottlieb, Shu, Tadmor, SIAM Review, 2001.
1245
1246 REAL(EB), INTENT(IN) :: ZZ.IN(1:N.TRACKED.SPECIES),DT.SUB,ZETA.IN,RHO.HAT,ZZ.UNMIXED(1:N.TRACKED.SPECIES),
      CELL.MASS,&
1247 TAU.MIX,PBAR.0,DELTA,VEL.RMS,TMP.UNMIXED
1248 REAL(EB), INTENT(OUT) :: ZZ.OUT(1:N.TRACKED.SPECIES),ZETA.OUT,Q.REAC.OUT(1:N.REACTIONS),TOTAL.MIXED.MASS.OUT
1249 REAL(EB), INTENT(INOUT) :: TMP.MIXED
1250 INTEGER, INTENT(IN) :: N.INC,SUB.IT,CO.PASS
1251 REAL(EB) :: DT.LOC,TOTAL.MIXED.MASS.0,TOTAL.MIXED.MASS.1,TOTAL.MIXED.MASS.2,TOTAL.MIXED.MASS.3,&
1252 ZZ.0(1:N.TRACKED.SPECIES),ZZ.1(1:N.TRACKED.SPECIES),ZZ.2(1:N.TRACKED.SPECIES),ZZ.3(1:N.TRACKED.SPECIES),&
1253 ZETA.0,ZETA.1,ZETA.2,ZETA.3,&
1254 Q.REAC.1(1:N.REACTIONS),Q.REAC.2(1:N.REACTIONS),Q.REAC.3(1:N.REACTIONS)
1255 INTEGER :: N
1256
1257 DT.LOC = DT.SUB/REAL(N.INC,EB) ! in principle, multiple increments could be used for Richardson extrapolation
1258 ZZ.0 = ZZ.IN
1259 ZETA.0 = ZETA.IN
1260 Q.REAC.OUT(:) = 0._EB
1261 TOTAL.MIXED.MASS.0 = (1._EB-ZETA.0)*CELL.MASS
1262
1263 INC.LOOP: DO N=1,N.INC
1264
1265 CALL FIRE_FORWARD_EULER(ZZ.1,ZZ.0,ZZ.UNMIXED,ZETA.1,ZETA.0,DT.LOC,TMP.MIXED,TMP.UNMIXED,RHO.HAT,CELL.MASS,
      TAU.MIX,&
1266 PBAR.0,DELTA,VEL.RMS,Q.REAC.1,SUB.IT,TOTAL.MIXED.MASS.1,CO.PASS)
1267
1268 CALL FIRE_FORWARD_EULER(ZZ.2,ZZ.1,ZZ.UNMIXED,ZETA.2,ZETA.1,DT.LOC,TMP.MIXED,TMP.UNMIXED,RHO.HAT,CELL.MASS,
      TAU.MIX,&
1269 PBAR.0,DELTA,VEL.RMS,Q.REAC.2,SUB.IT,TOTAL.MIXED.MASS.2,CO.PASS)
1270
1271 IF (TOTAL.MIXED.MASS.2>TWO.EPSILON.EB) THEN
1272 ZZ.2 = 0.75._EB*ZZ.0*TOTAL.MIXED.MASS.0 + 0.25._EB*ZZ.2*TOTAL.MIXED.MASS.2
1273 ZZ.2 = ZZ.2/SUM(ZZ.2)
1274 ELSE
1275 ZZ.2 = ZZ.0
1276 ENDIF
1277
1278 Q.REAC.2 = 0.25._EB * (Q.REAC.1 + Q.REAC.2)
1279
1280 ZETA.2 = 0.75._EB*ZETA.0 + 0.25._EB*ZETA.2
1281
1282 CALL FIRE_FORWARD_EULER(ZZ.3,ZZ.2,ZZ.UNMIXED,ZETA.3,ZETA.2,DT.LOC,TMP.MIXED,TMP.UNMIXED,RHO.HAT,CELL.MASS,
      TAU.MIX,&
1283 PBAR.0,DELTA,VEL.RMS,Q.REAC.3,SUB.IT,TOTAL.MIXED.MASS.3,CO.PASS)
1284
1285 IF (TOTAL.MIXED.MASS.3>TWO.EPSILON.EB) THEN
1286 ZZ.OUT = ONI*ZZ.0*TOTAL.MIXED.MASS.0 + TWH*ZZ.3*TOTAL.MIXED.MASS.3
1287 TOTAL.MIXED.MASS.OUT = SUM(ZZ.OUT)
1288 ZZ.OUT = ZZ.OUT/TOTAL.MIXED.MASS.OUT
1289 ELSE
1290 ZZ.OUT = ZZ.0
1291 ENDIF
1292
1293 Q.REAC.OUT = Q.REAC.OUT + TWH * (Q.REAC.2 + Q.REAC.3)
1294
1295 ZETA.OUT = MAX(0._EB,1._EB-TOTAL.MIXED.MASS.OUT/CELL.MASS)
1296
1297 ZZ.0 = ZZ.OUT
1298 ZETA.0 = ZETA.OUT
1299 TOTAL.MIXED.MASS.0 = TOTAL.MIXED.MASS.OUT
1300
1301 ENDDO INC.LOOP
1302
1303 END SUBROUTINE FIRE_RK3
1304
1305 SUBROUTINE REACTION_RATE.1(DZZ,ZZ.0,DT.LOC,RHO.0,TMP.0,KINETICS) !changed name to REACTION_RATE.1 for 6.2.0
1306 USE PHYSICAL_FUNCTIONS, ONLY : GET.MASS.FRACTION.ALL,GET.SPECIFIC.GAS.CONSTANT,GET.GIBBS.FREE.ENERGY,
      GET.MOLECULAR.WEIGHT
1307 REAL(EB), INTENT(OUT) :: DZZ(1:N.TRACKED.SPECIES)
1308 REAL(EB), INTENT(IN) :: ZZ.0(1:N.TRACKED.SPECIES),DT.LOC,RHO.0,TMP.0
1309 INTEGER, INTENT(IN) :: KINETICS
1310 REAL(EB) :: DZ.F(1:N.REACTIONS),YY.PRIMITIVE(1:N.SPECIES),DG.RXN,MW,MOLPCMB
1311 INTEGER :: I,NS
1312 INTEGER, PARAMETER :: INFINITELY_FAST=1,FINITE_RATE=2
1313 TYPE(REACTION_TYPE),POINTER :: RN=>NULL()
1314
1315 DZ.F = 0._EB
1316 DZZ = 0._EB
1317
1318 KINETICS.SELECT: SELECT CASE(KINETICS)
1319 CASE(INFINITELY_FAST)
1320 IF (EXTINCT1) RETURN !changed name for 6.2.0

```

```

1322 REACTION_LOOP.1: DO I=1,N_REACTIONS
1323 RN => REACTION(I)
1324 IF (.NOT.RN%FAST_CHEMISTRY) CYCLE REACTION_LOOP.1
1325 DZ.F(1) = ZZ.0(RN%FUEL_SMIX_INDEX)
1326 DZZ = DZZ + RN%NUMW.OMWF*DZ.F(1)
1327 ENDDO REACTION_LOOP.1
1328
1329 CASE(FINITE_RATE)
1330 REACTION_LOOP.2: DO I=1,N_REACTIONS
1331 RN => REACTION(I)
1332 IF (RN%FAST_CHEMISTRY .OR. ZZ.0(RN%FUEL_SMIX_INDEX) < ZZ.MIN.GLOBAL) CYCLE REACTION_LOOP.2
1333 IF (RN%AIR_SMIX_INDEX > -1) THEN
1334 IF (ZZ.0(RN%AIR_SMIX_INDEX) < ZZ.MIN.GLOBAL) CYCLE REACTION_LOOP.2 ! no expected air
1335 ENDIF
1336 CALL GET_MASS_FRACTION_ALL(ZZ.0, YY.PRIMITIVE)
1337 DO NS=1,N_SPECIES
1338 IF (RN%N.S(NS))>= -998._EB .AND. YY.PRIMITIVE(NS) < ZZ.MIN.GLOBAL) CYCLE REACTION_LOOP.2
1339 ENDDO
1340 DZ.F(1) = RN%A.PRIME*RHO.0**RN%RHO_EXPONENT*TMP.0**RN%N.T*EXP(-RN%E/(R0*TMP.0)) ! FDS Tech Guide, Eq. (5.49)
1341 DO NS=1,N_SPECIES
1342 IF (RN%N.S(NS))>= -998._EB) DZ.F(1) = YY.PRIMITIVE(NS)**RN%N.S(NS)*DZ.F(1)
1343 ENDDO
1344 IF (RN%THIRD_BODY) THEN
1345 CALL GET_MOLECULAR_WEIGHT(ZZ.0,MW)
1346 MOLPCMB = RHO.0/MW*0.001._EB ! mol/cm^3
1347 DZ.F(1) = DZ.F(1) * MOLPCMB
1348 ENDIF
1349 IF (RN%REVERSE) THEN ! compute equilibrium constant
1350 CALL GET_GIBBS_FREE_ENERGY(DG.RXN,RN%NU,TMP.0)
1351 RN%K = EXP(-DG.RXN/(R0*TMP.0))
1352 ENDIF
1353 DZZ = DZZ + RN%NUMW.OMWF*DZ.F(1)*DT.LOC/RN%K
1354 ENDDO REACTION_LOOP.2
1355
1356 END SELECT KINETICS.SELECT
1357
1358 END SUBROUTINE REACTION_RATE.1 !changed name
1359
1360 SUBROUTINE REACTION_RATE(DZZ,ZZ.0,DT.SUB,RHO.0,TMP.0,KINETICS,PBAR.0,DELTA,VEL.RMS,Q_REAC.OUT,SUB.IT,CO.PASS)
1361 USE COMP_FUNCTIONS, ONLY: SHUTDOWN
1362 USE PHYSICAL_FUNCTIONS, ONLY: GET_MASS_FRACTION_ALL,GET_SPECIFIC_GAS_CONSTANT,GET_GIBBS_FREE_ENERGY,
    GET_MOLECULAR_WEIGHT
1363 REAL(EB), INTENT(OUT) :: DZZ(1:N_TRACKED_SPECIES),Q_REAC.OUT(1:N_REACTIONS)
1364 REAL(EB), INTENT(IN) :: ZZ.0(1:N_TRACKED_SPECIES),DT.SUB,RHO.0,TMP.0,PBAR.0,DELTA,VEL.RMS
1365 INTEGER, INTENT(IN) :: KINETICS,SUB.IT,CO.PASS
1366 REAL(EB) :: DZ.F,YY.PRIMITIVE(1:N_SPECIES),DG.RXN,MW,MOLPCMB,DT.TMP(1:N_TRACKED_SPECIES),DT.MIN,DT.LOC,&
    ZZ.TMP(1:N_TRACKED_SPECIES),ZZ.NEW(1:N_TRACKED_SPECIES),Q_REAC.TMP(1:N_REACTIONS),AA,DTHETA
1367 INTEGER :: I,NS,SUB.IT.USE,OUTER.IT
1368 LOGICAL :: REACTANTS.PRESENT
1369 INTEGER, PARAMETER :: INFINITELY_FAST=1,FINITE_RATE=2
1370 TYPE(REACTION_TYPE), POINTER :: RN=>NULL()
1371
1372
1373 ZZ.NEW = ZZ.0
1374 Q_REAC.OUT = 0._EB
1375 Q_REAC.TMP = 0._EB
1376 SUB.IT.USE = SUB.IT ! keep this for debug
1377
1378 KINETICS.SELECT: SELECT CASE(KINETICS)
1379
1380 CASE(INFINITELY_FAST)
1381
1382 FAST_REAC_LOOP: DO OUTER.IT=1,N_REACTIONS
1383 IF (CO.PASS==2 .AND. OUTER.IT/=2) CYCLE FAST_REAC_LOOP
1384 ZZ.TMP = ZZ.NEW
1385 DZZ = 0._EB
1386 REACTANTS.PRESENT = .FALSE.
1387 REACTION_LOOP.1: DO I=1,N_REACTIONS
1388 RN => REACTION(I)
1389 IF (.NOT.RN%FAST_CHEMISTRY) CYCLE REACTION_LOOP.1
1390 IF (RN%AIR_SMIX_INDEX > -1) THEN
1391 DZ.F = ZZ.TMP(RN%FUEL_SMIX_INDEX)*ZZ.TMP(RN%AIR_SMIX_INDEX) ! 2nd-order reaction
1392 ELSE
1393 DZ.F = ZZ.TMP(RN%FUEL_SMIX_INDEX) ! 1st-order
1394 ENDIF
1395 IF (DZ.F > TWO_EPSILON.EB) REACTANTS.PRESENT = .TRUE.
1396 DTHETA = FLAME_SPEED_FACTOR(ZZ.TMP,DT.LOC,RHO.0,TMP.0,PBAR.0,I,DELTA,VEL.RMS)
1397 AA = RN%A.PRIME.FAST * RHO.0**RN%RHO_EXPONENT.FAST * DTHETA
1398 DZZ = DZZ + AA * RN%NUMW.OMWF * DZ.F
1399 Q_REAC.TMP(I) = RN%HEAT_OF_COMBUSTION * AA * DZ.F
1400 ENDDO REACTION_LOOP.1
1401 IF (REACTANTS.PRESENT) THEN
1402 DT.TMP = HUGE.EB
1403 DO NS = 1,N_TRACKED_SPECIES
1404 IF (DZZ(NS) < 0._EB) DT.TMP(NS) = -DZZ(NS)/DZZ(NS)
1405 ENDDO
1406 DT.MIN = MINVAL(DT.TMP)
1407 ZZ.NEW = ZZ.TMP + DZZ*DT.MIN
1408 Q_REAC.OUT = Q_REAC.OUT + Q_REAC.TMP*DT.MIN

```

```

1409 ELSE
1410 EXIT FAST_REAC_LOOP
1411 ENDF
1412 ENDDO FAST_REAC_LOOP
1413 DZZ = ZZ_NEW - ZZ_0
1414
1415 CASE(FINITE_RATE)
1416
1417 DT_LOC = DT_SUB
1418 SLOW_REAC_LOOP: DO OUTER_IT=1,N_REACTIONS
1419 ZZ_TMP = ZZ_NEW
1420 DZZ = 0._EB
1421 REACTANTS_PRESENT = .FALSE.
1422 REACTION_LOOP_2: DO I=1,N_REACTIONS
1423 RN => REACTION(I)
1424 IF (RN%FAST_CHEMISTRY) CYCLE REACTION_LOOP_2
1425 IF (ZZ_TMP(RN%FUEL_SMIX_INDEX) < ZZ_MIN_GLOBAL) CYCLE REACTION_LOOP_2
1426 IF (RN%AIR_SMIX_INDEX > -1) THEN
1427 IF (ZZ_TMP(RN%AIR_SMIX_INDEX) < ZZ_MIN_GLOBAL) CYCLE REACTION_LOOP_2 ! no expected air
1428 ENDF
1429 CALL GET_MASS_FRACTION_ALL(ZZ_TMP,YY_PRIMITIVE)
1430 DO NS=1,N_SPECIES
1431 IF (RN%N_S(NS) > -998._EB .AND. YY_PRIMITIVE(NS) < ZZ_MIN_GLOBAL) CYCLE REACTION_LOOP_2
1432 ENDDO
1433 DZ_F = RN%A_PRIME*RHO_0**RN%RHO_EXPONENT*TMP_0**RN%N_T*EXP(-RN%E/(R0*TMP_0)) ! dZ/dt, FDS Tech Guide, Eq. (5.47)
1434 DO NS=1,N_SPECIES
1435 IF (RN%N_S(NS) > -998._EB) DZ_F = YY_PRIMITIVE(NS)**RN%N_S(NS)*DZ_F
1436 ENDDO
1437 IF (RN%THIRD_BODY) THEN
1438 CALL GET_MOLECULAR_WEIGHT(ZZ_TMP,MW)
1439 MOLPCMB = RHO_0/MW*0.001._EB ! mol/cm^3
1440 DZ_F = DZ_F * MOLPCMB
1441 ENDF
1442 IF (RN%REVERSE) THEN ! compute equilibrium constant
1443 CALL GET_GIBBS_FREE_ENERGY(DG_RXN,RN%NU,TMP_0)
1444 RN%K = EXP(-DG_RXN/(R0*TMP_0))
1445 DZ_F = DZ_F/RN%K
1446 ENDF
1447 IF (DZ_F > TWO_EPSILON_EB) REACTANTS_PRESENT = .TRUE.
1448 Q_REAC_TMP(I) = RN%HEAT_OF_COMBUSTION * DZ_F * DT_LOC ! Note: here DZ_F=dZ/dt, hence need DT_LOC
1449 DZZ = DZZ + RN%NU*MW_0/MW_F*DZ_F*DT_LOC
1450 ENDDO REACTION_LOOP_2
1451 IF (REACTANTS_PRESENT) THEN
1452 DT_TMP = HUGE_EB
1453 DO NS = 1,N_TRACKED_SPECIES
1454 IF (DZZ(NS) < 0._EB) DT_TMP(NS) = -ZZ_TMP(NS)/DZZ(NS)
1455 ENDDO
1456 ! Think of DT_MIN as the fraction of DT_LOC we can take and remain bounded.
1457 DT_MIN = MIN(1._EB,MINVAL(DT_TMP))
1458 DT_LOC = DT_LOC*(1._EB-DT_MIN)
1459 ZZ_NEW = ZZ_TMP + DZZ*DT_MIN
1460 Q_REAC_OUT = Q_REAC_OUT + Q_REAC_TMP*DT_MIN
1461 IF (DT_LOC<TWO_EPSILON_EB) EXIT SLOW_REAC_LOOP
1462 ELSE
1463 EXIT SLOW_REAC_LOOP
1464 ENDF
1465 ENDDO SLOW_REAC_LOOP
1466 DZZ = ZZ_NEW - ZZ_0
1467
1468 END SELECT KINETICS_SELECT
1469
1470 END SUBROUTINE REACTION_RATE
1471
1472
1473 REAL(EB) FUNCTION FLAME_SPEED_FACTOR(ZZ_0,DT_LOC,RHO_0,TMP_0,PBAR_0,NR,DELTA,VEL_RMS)
1474 USE PHYSICAL_FUNCTIONS, ONLY : GET_SENSIBLE_ENTHALPY,GET_SPECIFIC_GAS_CONSTANT,GET_SPECIFIC_HEAT
1475 REAL(EB), INTENT(IN) :: ZZ_0(1:N_TRACKED_SPECIES),RHO_0,TMP_0,PBAR_0,DT_LOC,DELTA,VEL_RMS
1476 INTEGER, INTENT(IN) :: NR
1477 TYPE(REACTION_TYPE),POINTER :: RN=>NULL()
1478 REAL(EB) :: DZ_F,ZZ_B(1:N_TRACKED_SPECIES),TMP_B,H_S_B,RHO_B,H_S_0,RSUM_B,PHI,S_L,S_T,HNEW,TMP_2,CP_B
1479 INTEGER :: IT
1480 ! REAL(EB) :: DPHI ! debug
1481
1482 FLAME_SPEED_FACTOR = 1._EB
1483
1484 RN=>REACTION(NR)
1485 IF (RN%FLAME_SPEED<0._EB) RETURN
1486
1487 ! equivalence ratio of unburnt mixture
1488 PHI = RN%S*ZZ_0(RN%FUEL_SMIX_INDEX)/ZZ_0(RN%AIR_SMIX_INDEX)
1489
1490 ! burnt composition
1491 DZ_F = MIN(ZZ_0(RN%FUEL_SMIX_INDEX),ZZ_0(RN%AIR_SMIX_INDEX)/RN%S)
1492 ZZ_B = ZZ_0 + RN%NU*MW_0/MW_F*DZ_F
1493 ZZ_B = MIN(1._EB,MAX(0._EB,ZZ_B))
1494
1495 ! find burnt zone temperature
1496 CALL GET_SENSIBLE_ENTHALPY(ZZ_0,H_S_0,TMP_0)

```

Source Code files for edited portions of FDS

```

1497 HNEW = H.S_0 + (1._EB-RN%CHL.R)*DZ.F*RN%HEAT.OF.COMBUSTION
1498 TMP.B = TMP_0
1499 TMP_2 = TMP_B
1500
1501 DO IT=1,10
1502 CALL GET_SENSIBLE.ENTHALPY(ZZ.B,H.S.B,TMP.B)
1503 CALL GET_SPECIFIC.HEAT(ZZ.B,CP.B,TMP.B)
1504 TMP.B = TMP.B+(HNEW - H.S.B)/CP.B
1505 ! < 10 K error for determining flame speed is sufficient
1506 IF (ABS(TMP_2-TMP.B)<10._EB) EXIT
1507 TMP_2 = TMP_B
1508 ENDDO
1509
1510 ! compute burnt zone density
1511 CALL GET_SPECIFIC.GAS.CONSTANT(ZZ.B,RSUM.B)
1512 RHO.B = PBAR_0/(RSUM.B*TMP.B)
1513
1514 ! get turbulent flame speed
1515
1516 ! ! (debug) check laminar flame speed ramp
1517 ! PHI = 0._EB
1518 ! DPHI = .1.EB
1519 ! DO IT=1,20
1520 !   PHI = PHI+DPHI
1521 !   S.L = LAMINAR.FLAME.SPEED(TMP_0,PHI,NR)
1522 !   print *,PHI,S.L
1523 ! ENDDO
1524 ! stop
1525
1526 S.L = LAMINAR.FLAME.SPEED(TMP_0,PHI,NR)
1527
1528 IF (S.L<TWO.EPSILON.EB) THEN
1529 FLAME.SPEED.FACTOR = 0._EB
1530 ELSE
1531 S.T = MAX( S.L, S.L*( 1._EB + RN%TURBULENT.FLAME.SPEED.ALPHA*(VEL.RMS/S.L)**RN%TURBULENT.FLAME.SPEED.EXPONENT ) )
1532 FLAME.SPEED.FACTOR = RHO.B/RHO_0 * S.T * DT.LOC/DLTA
1533 ENDF
1534
1535 END FUNCTION FLAME.SPEED.FACTOR
1536
1537
1538 REAL(EB) FUNCTION LAMINAR.FLAME.SPEED(TMP,EQ,NR)
1539 USE MATH.FUNCTIONS, ONLY: EVALUATE.RAMP, INTERPOLATE2D
1540 REAL(EB), INTENT(IN) :: TMP,EQ
1541 INTEGER, INTENT(IN) :: NR
1542 TYPE(REACTION.TYPE), POINTER :: RN=>NULL()
1543
1544 RN=>REACTION(NR)
1545
1546 IF (RN%TABLE.FS.INDEX>0) THEN
1547 CALL INTERPOLATE2D(RN%TABLE.FS.INDEX,EQ,TMP,LAMINAR.FLAME.SPEED)
1548 ELSE
1549 LAMINAR.FLAME.SPEED = RN%FLAME.SPEED*(TMP/RN%FLAME.SPEED.TEMPERATURE)**RN%FLAME.SPEED.EXPONENT &
1550 *EVALUATE.RAMP(EQ,0._EB,RN%RAMP.FS.INDEX)
1551 ENDF
1552
1553 END FUNCTION LAMINAR.FLAME.SPEED
1554
1555
1556 SUBROUTINE ZETA.PRODUCTION(DT)
1557 USE MASS, ONLY: SCALAR.FACE.VALUE
1558
1559 REAL(EB), INTENT(IN) :: DT
1560 INTEGER :: I,J,K,IIG,JJG,KKG,IOR,IW,II,JJ,KK
1561 REAL(EB) :: Z.F,DENOM,ZZZ(1:4),DZDX,DZDY,DZDZ
1562 REAL(EB), POINTER, DIMENSION(:,:,:) :: ZFX=>NULL(),ZFY=>NULL(),ZFZ=>NULL(),ZZP=>NULL(),UU=>NULL(),VV=>NULL(),WW=>
1563 NULL()
1564 TYPE(WALL.TYPE), POINTER :: WC=>NULL()
1565
1566 ZFX =>WORK1
1567 ZFY =>WORK2
1568 ZFZ =>WORK3
1569 ZZP =>WORK4
1570
1571 UU=>U
1572 VV=>V
1573 WW=>W
1574
1575 !$OMP PARALLEL PRIVATE(ZZZ)
1576 !$OMP DO SCHEDULE(STATIC)
1577 DO K=0,KBP1
1578 DO J=0,JBP1
1579 DO I=0,IBP1
1580 ZZP(I,J,K) = ZZ(I,J,K,REACTION(1)%FUEL.SMIX.INDEX)
1581 ENDDO
1582 ENDDO
1583 ENDDO
1584 !$OMP END DO

```

Source Code files for edited portions of FDS

```

1584
1585 ! Compute scalar face values
1586
1587 !$OMP DO SCHEDULE(STATIC)
1588 DO K=1,KBAR
1589 DO J=1,JBAR
1590 DO I=1,IBM1
1591 ZZZ(1:4) = ZZZ(I-1:I+2,J,K)
1592 ZFX(I,J,K) = SCALAR.FACE.VALUE(UU(I,J,K),ZZZ,FLUX.LIMITER)
1593 ENDDO
1594 ENDDO
1595 ENDDO
1596 !$OMP END DO NOWAIT
1597
1598 !$OMP DO SCHEDULE(STATIC)
1599 DO K=1,KBAR
1600 DO J=1,JBAR
1601 DO I=1,IBM1
1602 ZZZ(1:4) = ZZZ(I,J-1:J+2,K)
1603 ZFY(I,J,K) = SCALAR.FACE.VALUE(VV(I,J,K),ZZZ,FLUX.LIMITER)
1604 ENDDO
1605 ENDDO
1606 ENDDO
1607 !$OMP END DO NOWAIT
1608
1609 !$OMP DO SCHEDULE(STATIC)
1610 DO K=1,KBAR
1611 DO J=1,JBAR
1612 DO I=1,IBAR
1613 ZZZ(1:4) = ZZZ(I,J,K-1:K+2)
1614 ZFZ(I,J,K) = SCALAR.FACE.VALUE(WW(I,J,K),ZZZ,FLUX.LIMITER)
1615 ENDDO
1616 ENDDO
1617 ENDDO
1618 !$OMP END DO
1619 !$OMP END PARALLEL
1620
1621 WALL_LOOP_2: DO IW=1,N_EXTERNAL.WALL.CELLS+N_INTERNAL.WALL.CELLS
1622 WC=>WALL(IW)
1623 IF (WC%BOUNDARY.TYPE==NULLBOUNDARY) CYCLE WALL_LOOP_2
1624
1625 II = WC%ONE.D%II
1626 JJ = WC%ONE.D%JJ
1627 KK = WC%ONE.D%KK
1628 IIG = WC%ONE.D%IIG
1629 JJG = WC%ONE.D%JJG
1630 KKG = WC%ONE.D%KKG
1631 IOR = WC%ONE.D%IOR
1632
1633 Z.F = WC%ONE.D%ZZ.F(REACTION(1)%FUEL.SMIX.INDEX)
1634
1635 SELECT CASE(IOR)
1636 CASE( 1); ZFX(IIG-1,JJG,KKG) = Z.F
1637 CASE(-1); ZFX(IIG,JJG,KKG) = Z.F
1638 CASE( 2); ZFY(IIG,JJG-1,KKG) = Z.F
1639 CASE(-2); ZFY(IIG,JJG,KKG) = Z.F
1640 CASE( 3); ZFZ(IIG,JJG,KKG-1) = Z.F
1641 CASE(-3); ZFZ(IIG,JJG,KKG) = Z.F
1642 END SELECT
1643
1644 ! Overwrite first off-wall advective flux if flow is away from the wall and if the face is not also a wall cell
1645
1646 OFF_WALL_IF_2: IF (WC%BOUNDARY.TYPE/=INTERPOLATED.BOUNDARY .AND. WC%BOUNDARY.TYPE/=OPEN.BOUNDARY) THEN
1647
1648 OFF_WALL_SELECT_2: SELECT CASE(IOR)
1649 CASE( 1) OFF_WALL_SELECT_2
1650 !      ghost      FX/UU(II+1)
1651 !  ///  II  ///  II+1  |  II+2  |  ...
1652 !      ^ WALL_INDEX(II+1,+1)
1653 IF ((UU(II+1,JJ,KK)>0..EB) .AND. .NOT.(WALL_INDEX(CELL_INDEX(II+1,JJ,KK),+1)>0)) THEN
1654 ZZZ(1:3) = (/Z.F,ZZP(II+1:II+2,JJ,KK)/)
1655 ZFX(II+1,JJ,KK) = SCALAR.FACE.VALUE(UU(II+1,JJ,KK),ZZZ,FLUX.LIMITER)
1656 ENDF
1657 CASE(-1) OFF_WALL_SELECT_2
1658 !      FX/UU(II-2)      ghost
1659 !  ...  |  II-2  |  II-1  ///  II  ///
1660 !      ^ WALL_INDEX(II-1,-1)
1661 IF ((UU(II-2,JJ,KK)<0..EB) .AND. .NOT.(WALL_INDEX(CELL_INDEX(II-1,JJ,KK),-1)>0)) THEN
1662 ZZZ(2:4) = (/ZZP(II-2:II-1,JJ,KK),Z.F/)
1663 ZFX(II-2,JJ,KK) = SCALAR.FACE.VALUE(UU(II-2,JJ,KK),ZZZ,FLUX.LIMITER)
1664 ENDF
1665 CASE( 2) OFF_WALL_SELECT_2
1666 IF ((VV(II,JJ+1,KK)>0..EB) .AND. .NOT.(WALL_INDEX(CELL_INDEX(II,JJ+1,KK),+2)>0)) THEN
1667 ZZZ(1:3) = (/Z.F,ZZP(II,JJ+1:JJ+2,KK)/)
1668 ZFY(II,JJ+1,KK) = SCALAR.FACE.VALUE(VV(II,JJ+1,KK),ZZZ,FLUX.LIMITER)
1669 ENDF
1670 CASE(-2) OFF_WALL_SELECT_2
1671 IF ((VV(II,JJ-2,KK)<0..EB) .AND. .NOT.(WALL_INDEX(CELL_INDEX(II,JJ-1,KK),-2)>0)) THEN

```

Source Code files for edited portions of FDS

```

1672 ZZZ(2:4) = (/ZZP(II, JJ -2:JJ -1, KK), Z.F/)
1673 ZFY(II, JJ -2, KK) = SCALAR.FACE.VALUE(VV(II, JJ -2, KK), ZZZ, FLUX.LIMITER)
1674 ENDDO
1675 CASE( 3) OFF.WALL.SELECT.2
1676 IF ((WW(II, JJ, KK+1)>0..EB) .AND. .NOT.(WALL.INDEX(CELL.INDEX(II, JJ, KK+1), +3)>0)) THEN
1677 ZZZ(1:3) = (/Z.F, ZZZ(II, JJ, KK+1:KK+2)/)
1678 ZFZ(II, JJ, KK+1) = SCALAR.FACE.VALUE(WW(II, JJ, KK+1), ZZZ, FLUX.LIMITER)
1679 ENDDO
1680 CASE(-3) OFF.WALL.SELECT.2
1681 IF ((WW(II, JJ, KK-2)<0..EB) .AND. .NOT.(WALL.INDEX(CELL.INDEX(II, JJ, KK-1), -3)>0)) THEN
1682 ZZZ(2:4) = (/Z.F, ZZZ(II, JJ, KK-2:KK-1), Z.F/)
1683 ZFZ(II, JJ, KK-2) = SCALAR.FACE.VALUE(WW(II, JJ, KK-2), ZZZ, FLUX.LIMITER)
1684 ENDDO
1685 END SELECT OFF.WALL.SELECT.2
1686
1687 ENDDO OFF.WALL.IF.2
1688
1689 ENDDO WALL.LOOP.2
1690
1691 ! Production term
1692
1693 DO K=1, KBAR
1694 DO J=1, JBAR
1695 DO I=1, IBAR
1696 IF (SOLID(CELL.INDEX(I, J, K))) CYCLE
1697
1698 DZDX = (ZFX(I, J, K)-ZFX(I-1, J, K))*RDX(1)
1699 DZDY = (ZFY(I, J, K)-ZFY(I, J-1, K))*RDY(J)
1700 DZDZ = (ZFZ(I, J, K)-ZFZ(I, J, K-1))*RDZ(K)
1701
1702 DENOM = RHO(I, J, K) * ( ZZZ(I, J, K) - ZZZ(I, J, K)**2 )
1703
1704 IF (DENOM>TWO.EPSILON.EB) THEN
1705 ! scale sgs variance production
1706 ZETA.SOURCE.TERM(I, J, K) = 2..EB*MU(I, J, K)/SC*( DZDX**2 + DZDY**2 + DZDZ**2 ) / DENOM
1707 ELSE
1708 ! cell is pure, unmix
1709 ZETA.SOURCE.TERM(I, J, K) = (1..EB - ZZ(I, J, K, ZETA.INDEX))/DT
1710 ENDDO
1711
1712 ZZ(I, J, K, ZETA.INDEX) = MIN( 1..EB, ZZ(I, J, K, ZETA.INDEX) + DT*ZETA.SOURCE.TERM(I, J, K) )
1713 ENDDO
1714 ENDDO
1715 ENDDO
1716
1717 END SUBROUTINE ZETA.PRODUCTION
1718
1719
1720 ! ----- CCREGION.COMBUSTION -----
1721
1722 SUBROUTINE CCREGION.COMBUSTION(T, DT, NM)
1723
1724 USE PHYSICAL.FUNCTIONS, ONLY: GET.SPECIFIC.GAS.CONSTANT, GET.MASS.FRACTION.ALL, GET.SPECIFIC.HEAT,
1725 GET.MOLECULAR.WEIGHT, &
1726 GET.SENSIBLE.ENTHALPY.Z, IS.REALIZABLE, LES.FILTER.WIDTH.FUNCTION
1727 USE COMPLEX.GEOMETRY, ONLY: IBM.CGSC, IBM.GASPHASE
1728
1729 REAL(EB), INTENT(IN) :: T, DT
1730 INTEGER, INTENT(IN) :: NM
1731
1732 ! Local Variables:
1733 INTEGER :: I, J, K, ICC, JCC, NCELL, NS, NR, N, CHEM.SUBIT.TMP
1734 REAL(EB) :: ZZ.GET(1:N.TRACKED.SPECIES), DZZ(1:N.TRACKED.SPECIES), CP, H.S.N, &
1735 REAC.SOURCE.TERM.TMP(N.TRACKED.SPECIES), Q.REAC.TMP(N.REACTIONS), VCELL
1736 REAL(EB) :: AIT.P, ZETA.P
1737 LOGICAL :: Q.EXISTS.CC
1738 TYPE (REACTION.TYPE), POINTER :: RN
1739 TYPE (SPECIES.MIXTURE.TYPE), POINTER :: SM
1740 LOGICAL :: DO.REACTION, REALIZABLE, DEBUG
1741 LOGICAL :: Q.EXISTS
1742
1743 ! Set to zero Reaction, Radiation sources of heat and thermodynamic div:
1744 DO K=1, KBAR
1745 DO J=1, JBAR
1746 DO I=1, IBAR
1747 IF (CCVAR(I, J, K, IBM.CGSC) == IBM.GASPHASE) CYCLE
1748 Q(I, J, K) = 0..EB
1749 QR(I, J, K) = 0..EB
1750 D.SOURCE(I, J, K) = 0..EB
1751 ENDDO
1752 ENDDO
1753 ENDDO
1754
1755 ! Now do COMBUSTION.GENERAL for cut-cells.
1756 Q.EXISTS.CC = .FALSE.
1757
1758 IF (REAC.SOURCE.CHECK) THEN
1759 DO ICC=1, MESHES(NM)%N.CUTCELL.MESH

```



```

1759 DO JCC=1,CUT_CELL(ICC)%NCELL
1760 CUT_CELL(ICC)%Q_REAC(:,JCC) = 0..EB
1761 ENDDO
1762 ENDDO
1763 ENDDIF
1764
1765 ZETA.P = 0..EB
1766 DEBUG = .FALSE.
1767
1768 ICC_LOOP : DO ICC=1,MESHES(NM)%N.CUTCELL_MESH
1769 I = CUT_CELL(ICC)%IJK(IAXIS)
1770 J = CUT_CELL(ICC)%IJK(JAXIS)
1771 K = CUT_CELL(ICC)%IJK(KAXIS)
1772
1773 VCELL = DX(I)*DY(J)*DZ(K)
1774
1775 IF (SOLID(CELL_INDEX(I,J,K))) CYCLE ICC_LOOP ! Cycle in case Cartesian cell inside OBSTS.
1776
1777 NCELL = CUT_CELL(ICC)%NCELL
1778 JCC_LOOP : DO JCC=1,NCELL
1779
1780 ! Drop if cut-cell is very small compared to Cartesian cells:
1781 IF ( ABS(CUT_CELL(ICC)%VOLUME(JCC)/VCELL) < 1.E-12.EB ) CYCLE JCC_LOOP
1782
1783 CUT_CELL(ICC)%CHLR(JCC) = 0..EB
1784 ZZ_GET = CUT_CELL(ICC)%ZZ(1:N.TRACKED_SPECIES,JCC)
1785
1786 AIT.P = 0..EB
1787 IF (REIGNITION_MODEL) AIT.P = CUT_CELL(ICC)%AIT(JCC)
1788
1789 IF (CHECK_REALIZABILITY) THEN
1790 REALIZABLE=IS_REALIZABLE(ZZ_GET)
1791 IF (.NOT.REALIZABLE) THEN
1792 WRITE(LU_ERR,*) I,J,K
1793 WRITE(LU_ERR,*) ZZ_GET
1794 WRITE(LU_ERR,*) SUM(ZZ_GET)
1795 WRITE(LU_ERR,*) 'ERROR: Unrealizable mass fractions input to COMBUSTION_MODEL'
1796 STOP STATUS=REALIZABILITY_STOP
1797 ENDF
1798 ENDDIF
1799 CALL CCHECK_REACTION
1800 IF (.NOT.DO_REACTION) CYCLE ICC_LOOP ! Check whether any reactions are possible.
1801
1802 DZZ = ZZ_GET ! store old ZZ for divergence term
1803 !*****
1804 ! Call combustion integration routine for CUT_CELL(ICC)%XX(JCC)
1805 CALL COMBUSTION_MODEL( T,DT,ZZ_GET,CUT_CELL(ICC)%Q(JCC),CUT_CELL(ICC)%MIX_TIME(JCC),&
1806 CUT_CELL(ICC)%CHLR(JCC),&
1807 CHEM.SUBIT_TMP,REAC.SOURCE_TERM_TMP,Q_REAC_TMP,&
1808 CUT_CELL(ICC)%GMP(JCC),CUT_CELL(ICC)%RHO(JCC),MU(I,J,K),KRES(I,J,K),&
1809 ZETA.P,AIT.P,PBAR(K,PRESSURE_ZONE(I,J,K)),&
1810 LES_FILTER_WIDTH_FUNCTION(DX(I),DY(J),DZ(K)),&
1811 CUT_CELL(ICC)%VOLUME(JCC)
1812 !*****
1813 IF (REAC.SOURCE_CHECK) THEN ! Store special diagnostic quantities
1814 CUT_CELL(ICC)%REAC.SOURCE_TERM(1:N.TRACKED_SPECIES,JCC)=REAC.SOURCE_TERM_TMP(1:N.TRACKED_SPECIES)
1815 CUT_CELL(ICC)%Q_REAC(1:N.REACTIONS,JCC)=Q_REAC_TMP(1:N.REACTIONS)
1816 ENDF
1817
1818 IF (CHECK_REALIZABILITY) THEN
1819 REALIZABLE=IS_REALIZABLE(ZZ_GET)
1820 IF (.NOT.REALIZABLE) THEN
1821 WRITE(LU_ERR,*) ZZ_GET,SUM(ZZ_GET)
1822 WRITE(LU_ERR,*) 'ERROR: Unrealizable mass fractions after COMBUSTION_MODEL'
1823 STOP STATUS=REALIZABILITY_STOP
1824 ENDF
1825 ENDDIF
1826
1827 DZZ = ZZ_GET - DZZ
1828
1829 ! Update RSLM and ZZ
1830 DZZ_IF: IF ( ANY(ABS(DZZ) > TWO_EPSILON_EB) ) THEN
1831 IF (ABS(CUT_CELL(ICC)%Q(JCC)) > TWO_EPSILON_EB) Q_EXISTS = .TRUE.
1832 ! Divergence term
1833 CALL GET_SPECIFIC_HEAT(ZZ_GET,CP,CUT_CELL(ICC)%TMP(JCC))
1834 CALL GET_SPECIFIC_GAS_CONSTANT(ZZ_GET,CUT_CELL(ICC)%R_SUM(JCC))
1835 DO N=1,N.TRACKED_SPECIES
1836 SM => SPECIES_MIXTURE(N)
1837 CALL GET_SENSIBLE_ENTHALPY_Z(N,CUT_CELL(ICC)%TMP(JCC),H_S,N)
1838 CUT_CELL(ICC)%D_SOURCE(JCC) = CUT_CELL(ICC)%D_SOURCE(JCC) + &
1839 ( SM*RCON/CUT_CELL(ICC)%R_SUM(JCC) - H_S.N/(CP*CUT_CELL(ICC)%TMP(JCC)) ) *DZZ(N)/DT
1840 CUT_CELL(ICC)%M_DOT_PPP(N,JCC) = CUT_CELL(ICC)%M_DOT_PPP(N,JCC) + &
1841 CUT_CELL(ICC)%RHO(JCC)*DZZ(N)/DT
1842 ENDDO
1843 ENDF DZZ_IF
1844 ENDDO JCC_LOOP
1845 ENDDO ICC_LOOP
1846

```

```

1847 ! This is for plotting regular slices:
1848 DO ICC=1,MESHES(NM)%N.CUTCCELL_MESH
1849 I = CUT_CELL(ICC)%IJK (IAXIS)
1850 J = CUT_CELL(ICC)%IJK (JAXIS)
1851 K = CUT_CELL(ICC)%IJK (KAXIS)
1852
1853 VCELL = DX(I)*DY(J)*DZ(K)
1854
1855 IF (SOLID(CELL_INDEX(I,J,K))) CYCLE ! Cycle in case Cartesian cell inside OBSTS.
1856
1857 NCELL = CUT_CELL(ICC)%NCELL
1858 DO JCC=1,NCELL
1859 Q(I,J,K) = Q(I,J,K)+CUT_CELL(ICC)%Q(JCC)*CUT_CELL(ICC)%VOLUME(JCC)
1860 D.SOURCE(I,J,K)=D.SOURCE(I,J,K)+CUT_CELL(ICC)%D.SOURCE(JCC)*CUT_CELL(ICC)%VOLUME(JCC)
1861 M.DOT.PPP(I,J,K,1:N.TOTAL_SCALARS) = M.DOT.PPP(I,J,K,1:N.TOTAL_SCALARS) + &
1862 CUT_CELL(ICC)%M.DOT.PPP(1:N.TOTAL_SCALARS,JCC)*CUT_CELL(ICC)%VOLUME(JCC)
1863 ENDDO
1864 Q(I,J,K) = Q(I,J,K)/VCELL
1865 D.SOURCE(I,J,K)=D.SOURCE(I,J,K)/VCELL
1866 M.DOT.PPP(I,J,K,1:N.TOTAL_SCALARS) = M.DOT.PPP(I,J,K,1:N.TOTAL_SCALARS)/VCELL
1867 ENDDO
1868
1869 RETURN
1870
1871 CONTAINS
1872
1873 SUBROUTINE CCHECK_REACTION
1874
1875 ! Check whether any reactions are possible.
1876
1877 LOGICAL :: REACTANTS_PRESENT
1878
1879 DO REACTION = 1,N.REACTIONS
1880 REACTION_LOOP: DO NR=1,N.REACTIONS
1881 RN=>REACTION(NR)
1882 REACTANTS_PRESENT = .TRUE.
1883 DO NS=1,N.TRACKED_SPECIES
1884 IF ( RN%NU(NS) < -TWO.EPSILON_EB .AND. ZZ_GET(NS) < ZZ_MIN_GLOBAL ) THEN
1885 REACTANTS_PRESENT = .FALSE.
1886 EXIT
1887 ENDF
1888 ENDDO
1889 DO REACTION = REACTANTS_PRESENT
1890 IF (DO_REACTION) EXIT REACTION_LOOP
1891 ENDDO REACTION_LOOP
1892
1893 END SUBROUTINE CCHECK_REACTION
1894
1895
1896 END SUBROUTINE CCREGION_COMBUSTION
1897
1898 !added this function for 6.2.0
1899 LOGICAL FUNCTION FUNC_EXTINCT(ZZ_MIXED_IN,TMP_MIXED)
1900 REAL(EB), INTENT(IN) :: ZZ_MIXED_IN(1:N.TRACKED_SPECIES),TMP_MIXED
1901
1902 FUNC_EXTINCT = .FALSE.
1903 IF (ANY(REACTION(:)%FAST_CHEMISTRY)) THEN
1904 SELECT CASE (EXTINCT_MOD)
1905 CASE(EXTINCTION_1)
1906 FUNC_EXTINCT = EXTINCT_1.1(ZZ_MIXED_IN,TMP_MIXED) !edited name
1907 CASE(EXTINCTION_2)
1908 FUNC_EXTINCT = EXTINCT_2.1(ZZ_MIXED_IN,TMP_MIXED) !edited name
1909 CASE(EXTINCTION_3)
1910 FUNC_EXTINCT = .FALSE.
1911 END SELECT
1912 ENDF
1913
1914 END FUNCTION FUNC_EXTINCT
1915
1916
1917 LOGICAL FUNCTION EXTINCT_1.1(ZZ_IN,TMP_MIXED) !edited name
1918 USE PHYSICAL_FUNCTIONS_ONLY:GET_AVERAGE_SPECIFIC_HEAT
1919 REAL(EB),INTENT(IN) :: ZZ_IN(1:N.TRACKED_SPECIES),TMP_MIXED
1920 REAL(EB) :: Y_O2,Y_O2_CRIT,CPBAR
1921 INTEGER :: NR
1922 TYPE(REACTION_TYPE),POINTER :: RN=>NULL()
1923
1924 EXTINCT_1.1 = .FALSE. !edited name
1925 REACTION_LOOP: DO NR=1,N.REACTIONS
1926 RN => REACTION(NR)
1927 IF (.NOT.RN%FAST_CHEMISTRY) CYCLE REACTION_LOOP
1928 AIT_IF: IF (TMP_MIXED < RN%AUTO_IGNITION_TEMPERATURE) THEN
1929 EXTINCT_1.1 = .TRUE. !edited name
1930 ELSE AIT_IF
1931 CALL GET_AVERAGE_SPECIFIC_HEAT(ZZ_IN,CPBAR,TMP_MIXED)
1932 Y_O2 = ZZ_IN(RN%AIR_SMIX_INDEX)
1933 Y_O2_CRIT = CPBAR*(RN%CRIT_FLAME_TMP-TMP_MIXED)/RN%EPUMO2
1934 IF (Y_O2 < Y_O2_CRIT) EXTINCT_1.1 = .TRUE. !edited name

```

Source Code files for edited portions of FDS

```

1935 ENDIF AIT_IF
1936 ENDDO REACTION_LOOP
1937
1938 END FUNCTION EXTINGT.1.1      !edited name
1939
1940
1941 LOGICAL FUNCTION EXTINGT.2.1 (ZZ_MIXED.IN, TMP_MIXED)      !edited name
1942 USE PHYSICAL_FUNCTIONS, ONLY: GET_SENSIBLE_ENTHALPY
1943 REAL (EB), INTENT (IN) :: ZZ_MIXED.IN (1:N_TRACKED_SPECIES), TMP_MIXED
1944 REAL (EB) :: ZZ_F, ZZ_HAT_F, ZZ_GET_F (1:N_TRACKED_SPECIES), ZZ_A, ZZ_HAT_A, ZZ_GET_A (1:N_TRACKED_SPECIES), ZZ_P, ZZ_HAT_P
1945 &
1946 ZZ_GET_P (1:N_TRACKED_SPECIES), H_F_0, H_A_0, H_P_0, H_F_N, H_A_N, H_P_N
1947 INTEGER :: NR
1948 TYPE (REACTION_TYPE), POINTER :: RN=>NULL()
1949
1950 EXTINGT.2.1 = .FALSE.      !edited name
1951 REACTION_LOOP: DO NR=1,N_REACTIONS
1952 RN => REACTION(NR)
1953 IF (.NOT.RN%FAST_CHEMISTRY) CYCLE REACTION_LOOP
1954 AIT_IF: IF (TMP_MIXED < RN%AUTO_IGNITION_TEMPERATURE) THEN
1955 EXTINGT.2.1 = .TRUE.      !edited name
1956 ELSE AIT_IF
1957 ZZ_F = ZZ_MIXED.IN (RN%FUEL_SMIX_INDEX)
1958 ZZ_A = ZZ_MIXED.IN (RN%AIR_SMIX_INDEX)
1959 ZZ_P = 1._EB - ZZ_F - ZZ_A
1960
1961 ZZ_HAT_F = MIN (ZZ_F, ZZ_MIXED.IN (RN%AIR_SMIX_INDEX) / RN%S) ! burned fuel, FDS Tech Guide (5.16)
1962 ZZ_HAT_A = ZZ_HAT_F * RN%S ! FDS Tech Guide (5.17)
1963 ZZ_HAT_P = (ZZ_HAT_A / (ZZ_A + TWO_EPSILON_EB)) * (ZZ_F - ZZ_HAT_F + ZZ_P) ! reactant diluent concentration, FDS Tech
1964 Guide (5.18)
1965
1966 ! "GET" indicates a composition vector. Below we are building up the masses of the constituents in the various
1967 ! mixtures. At this point these composition vectors are not normalized.
1968
1969 ZZ_GET_F = 0._EB
1970 ZZ_GET_A = 0._EB
1971 ZZ_GET_P = ZZ_MIXED.IN
1972
1973 ZZ_GET_F (RN%FUEL_SMIX_INDEX) = ZZ_HAT_F ! fuel in reactant mixture composition
1974 ZZ_GET_A (RN%AIR_SMIX_INDEX) = ZZ_HAT_A ! air in reactant mixture composition
1975
1976 ZZ_GET_P (RN%FUEL_SMIX_INDEX) = MAX (ZZ_GET_P (RN%FUEL_SMIX_INDEX) - ZZ_HAT_F, 0._EB) ! remove burned fuel from product
1977 composition
1978 ZZ_GET_P (RN%AIR_SMIX_INDEX) = MAX (ZZ_GET_P (RN%AIR_SMIX_INDEX) - ZZ_A, 0._EB) ! remove all air from product
1979 composition
1980
1981 ! Normalize concentrations
1982 ZZ_GET_F = ZZ_GET_F / (SUM (ZZ_GET_F) + TWO_EPSILON_EB)
1983 ZZ_GET_A = ZZ_GET_A / (SUM (ZZ_GET_A) + TWO_EPSILON_EB)
1984 ZZ_GET_P = ZZ_GET_P / (SUM (ZZ_GET_P) + TWO_EPSILON_EB)
1985
1986 ! Get the specific heat for the fuel and diluent at the current and critical flame temperatures
1987 CALL GET_SENSIBLE_ENTHALPY (ZZ_GET_F, H_F_0, TMP_MIXED)
1988 CALL GET_SENSIBLE_ENTHALPY (ZZ_GET_A, H_A_0, TMP_MIXED)
1989 CALL GET_SENSIBLE_ENTHALPY (ZZ_GET_P, H_P_0, TMP_MIXED)
1990 CALL GET_SENSIBLE_ENTHALPY (ZZ_GET_F, H_F_N, RN%CRIT_FLAME_TEMP)
1991 CALL GET_SENSIBLE_ENTHALPY (ZZ_GET_A, H_A_N, RN%CRIT_FLAME_TEMP)
1992 CALL GET_SENSIBLE_ENTHALPY (ZZ_GET_P, H_P_N, RN%CRIT_FLAME_TEMP)
1993
1994 ! See if enough energy is released to raise the fuel and required "air" temperatures above the critical flame
1995 temp.
1996 IF ( ZZ_HAT_F * (H_F_0 + RN%HEAT_OF_COMBUSTION) + ZZ_HAT_A * H_A_0 + ZZ_HAT_P * H_P_0 < &
1997 ZZ_HAT_F * H_F_N + ZZ_HAT_A * H_A_N + ZZ_HAT_P * H_P_N ) EXTINGT.2.1 = .TRUE. ! FDS Tech Guide (5.19)      !edited name
1998 ENDIF AIT_IF
1999 ENDDO REACTION_LOOP
2000
2001 END FUNCTION EXTINGT.2.1      !edited name
2002
2003
2004 LOGICAL FUNCTION EXTINGT.3.1 (ZZ_MIXED.IN, TMP_MIXED)      !edited name
2005 USE PHYSICAL_FUNCTIONS, ONLY: GET_SENSIBLE_ENTHALPY
2006 REAL (EB), INTENT (IN) :: ZZ_MIXED.IN (1:N_TRACKED_SPECIES), TMP_MIXED
2007 REAL (EB) :: H_F_0, H_A_0, H_P_0, H_F_N, Z_F, Z_A, Z_P, Z_A_STOICH, ZZ_HAT_F, ZZ_HAT_A, ZZ_HAT_P, &
2008 ZZ_GET_F (1:N_TRACKED_SPECIES), ZZ_GET_A (1:N_TRACKED_SPECIES), ZZ_GET_P (1:N_TRACKED_SPECIES), ZZ_GET_F_REAC (1:
2009 N_REACTIONS), &
2010 ZZ_GET_P_PP (1:N_TRACKED_SPECIES), DZ_F (1:N_REACTIONS), DZ_FRAC_F (1:N_REACTIONS), DZ_F_SUM, &
2011 HOC_EXTINCT, AIT_EXTINCT, CFT_EXTINCT
2012 INTEGER :: NS, NR
2013 TYPE (REACTION_TYPE), POINTER :: RN=>NULL()
2014
2015 EXTINGT.3.1 = .FALSE.      !edited name
2016 Z_F = 0._EB
2017 Z_A = 0._EB
2018 Z_P = 0._EB
2019 DZ_F = 0._EB
2020 DZ_F_SUM = 0._EB
2021 Z_A_STOICH = 0._EB
2022 ZZ_GET_F = 0._EB

```

```

2017 ZZ_GET_A = 0._EB
2018 ZZ_GET_P = ZZ_MIXED.IN
2019 ZZ_GET_PFP = 0._EB
2020 HOC_EXTINCT = 0._EB
2021 AIT_EXTINCT = 0._EB
2022 CFT_EXTINCT = 0._EB
2023
2024 DO NS=1,N_TRACKED_SPECIES
2025 SUM_FUEL_LOOP: DO NR = 1,N_REACTIONS
2026 RN => REACTION(NR)
2027 IF (RN%FAST_CHEMISTRY .AND. RN%HEAT_OF_COMBUSTION > 0._EB .AND. NS == RN%FUEL_SMIX_INDEX) THEN
2028 Z_F = Z_F + ZZ_MIXED.IN(NS)
2029 EXIT SUM_FUEL_LOOP
2030 ENDDIF
2031 ENDDO SUM_FUEL_LOOP
2032 SUM_AIR_LOOP: DO NR = 1,N_REACTIONS
2033 RN => REACTION(NR)
2034 IF (RN%FAST_CHEMISTRY .AND. RN%HEAT_OF_COMBUSTION > 0._EB .AND. RN%NU(NS) < 0._EB .AND. NS /= RN%FUEL_SMIX_INDEX)
    THEN
2035 Z_A = Z_A + ZZ_MIXED.IN(NS)
2036 ZZ_GET_P(NS) = MAX(ZZ_GET_P(NS) - ZZ_MIXED.IN(NS), 0._EB)
2037 EXIT SUM_AIR_LOOP
2038 ENDDIF
2039 ENDDO SUM_AIR_LOOP
2040 ENDDO
2041 Z_P = 1._EB - Z_F - Z_A
2042 DO NR = 1,N_REACTIONS
2043 RN => REACTION(NR)
2044 IF (RN%FAST_CHEMISTRY .AND. RN%HEAT_OF_COMBUSTION > 0._EB) THEN
2045 DZ_F(NR) = 1.E10._EB
2046 DO NS = 1,N_TRACKED_SPECIES
2047 IF (RN%NU(NS) < 0._EB) THEN
2048 DZ_F(NS) = MIN(DZ_F(NR), -ZZ_MIXED.IN(NS)/RN%NU.MW.OMWF(NS))
2049 ENDDIF
2050 IF (RN%NU(NS) < 0._EB .AND. NS /= RN%FUEL_SMIX_INDEX) THEN
2051 Z_A_STOICH = Z_A_STOICH + ZZ_MIXED.IN(RN%FUEL_SMIX_INDEX)*RN%S
2052 ENDDIF
2053 ENDDO
2054 ENDDIF
2055 ENDDO
2056 IF (Z_A_STOICH > Z_A) DZ_F.SUM = SUM(DZ_F)
2057 DO NR = 1,N_REACTIONS
2058 RN => REACTION(NR)
2059 IF (Z_A_STOICH > Z_A .AND. RN%HEAT_OF_COMBUSTION > 0._EB) THEN
2060 DZ_FRAC_F(NR) = DZ_F(NR)/MAX(DZ_F.SUM, TWO_EPSILON.EB)
2061 ZZ_GET_F(RN%FUEL_SMIX_INDEX) = DZ_F(NR)*DZ_FRAC_F(NR)
2062 ZZ_GET_P(RN%FUEL_SMIX_INDEX) = ZZ_GET_P(RN%FUEL_SMIX_INDEX) - ZZ_GET_F(RN%FUEL_SMIX_INDEX)
2063 ZZ_GET_PFP(RN%FUEL_SMIX_INDEX) = ZZ_GET_P(RN%FUEL_SMIX_INDEX)
2064 DO NS = 1,N_TRACKED_SPECIES
2065 IF (RN%NU(NS) < 0._EB .AND. NS /= RN%FUEL_SMIX_INDEX) THEN
2066 ZZ_GET_A(NS) = RN%S*ZZ_GET_F(RN%FUEL_SMIX_INDEX)
2067 ! ZZ_GET_P(NS) = ZZ_GET_P(NS) - ZZ_GET_A(NS)
2068 ZZ_GET_PFP(NS) = ZZ_GET_P(NS)
2069 ELSEIF (RN%NU(NS) >= 0._EB) THEN
2070 ZZ_GET_PFP(NS) = ZZ_GET_P(NS) + ZZ_GET_F(RN%FUEL_SMIX_INDEX)*RN%NU.MW.OMWF(NS)
2071 ENDDIF
2072 ENDDO
2073 ELSE
2074 ZZ_GET_F(RN%FUEL_SMIX_INDEX) = DZ_F(NR)
2075 ZZ_GET_P(RN%FUEL_SMIX_INDEX) = ZZ_GET_P(RN%FUEL_SMIX_INDEX) - ZZ_GET_F(RN%FUEL_SMIX_INDEX)
2076 ZZ_GET_PFP(RN%FUEL_SMIX_INDEX) = ZZ_GET_P(RN%FUEL_SMIX_INDEX)
2077 DO NS = 1,N_TRACKED_SPECIES
2078 IF (RN%NU(NS) < 0._EB .AND. NS /= RN%FUEL_SMIX_INDEX) THEN
2079 ZZ_GET_A(NS) = RN%S*ZZ_GET_F(RN%FUEL_SMIX_INDEX)
2080 ! ZZ_GET_P(NS) = ZZ_GET_P(NS) - ZZ_GET_A(NS)
2081 ZZ_GET_PFP(NS) = ZZ_GET_P(NS)
2082 ELSEIF (RN%NU(NS) >= 0._EB) THEN
2083 ZZ_GET_PFP(NS) = ZZ_GET_P(NS) + ZZ_GET_F(RN%FUEL_SMIX_INDEX)*RN%NU.MW.OMWF(NS)
2084 ENDDIF
2085 ENDDO
2086 ENDDIF
2087 ZZ_GET_F.REAC(NR) = ZZ_GET_F(RN%FUEL_SMIX_INDEX)
2088 ENDDO
2089
2090 ZZ_HAT_F = SUM(ZZ_GET_F)
2091 ZZ_HAT_A = SUM(ZZ_GET_A)
2092 ZZ_HAT_P = (ZZ_HAT_A/(Z_A+TWO_EPSILON.EB))*(Z_F-ZZ_HAT_F+SUM(ZZ_GET_P))
2093 !M.P.ST = SUM(ZZ_GET_P)
2094
2095 ! Normalize compositions
2096 ZZ_GET_F = ZZ_GET_F/(SUM(ZZ_GET_F)+TWO_EPSILON.EB)
2097 ZZ_GET_F.REAC = ZZ_GET_F.REAC/(SUM(ZZ_GET_F.REAC)+TWO_EPSILON.EB)
2098 ZZ_GET_A = ZZ_GET_A/(SUM(ZZ_GET_A)+TWO_EPSILON.EB)
2099 ZZ_GET_P = ZZ_GET_P/(SUM(ZZ_GET_P)+TWO_EPSILON.EB)
2100 ZZ_GET_PFP = ZZ_GET_PFP/(SUM(ZZ_GET_PFP)+TWO_EPSILON.EB)
2101
2102 DO NR = 1,N_REACTIONS
2103 RN => REACTION(NR)

```

```

2104 AIT_EXTINCT = AIT_EXTINCT+ZZ.GET_F.REAC(NR)*RN%AUTO_IGNITION_TEMPERATURE
2105 CFT_EXTINCT = CFT_EXTINCT+ZZ.GET_F.REAC(NR)*RN%CRIT_FLAME_TMP
2106 HOC_EXTINCT = HOC_EXTINCT+ZZ.GET_F.REAC(NR)*RN%HEAT_OF_COMBUSTION
2107 BNDDO
2108
2109 IF (TMP_MIXED < AIT_EXTINCT) THEN
2110 EXTINCT_3.1 = .TRUE. !edited name
2111 ELSE
2112 ! Get the specific heat for the fuel and diluent at the current and critical flame temperatures
2113 CALL GET_SENSIBLE_ENTHALPY(ZZ.GET_F,H.F.0,TMP_MIXED)
2114 CALL GET_SENSIBLE_ENTHALPY(ZZ.GET_A,H.A.0,TMP_MIXED)
2115 CALL GET_SENSIBLE_ENTHALPY(ZZ.GET_P,H.P.0,TMP_MIXED)
2116 CALL GET_SENSIBLE_ENTHALPY(ZZ.GET_PFP,H.P.N,CFT_EXTINCT)
2117
2118 ! See if enough energy is released to raise the fuel and required "air" temperatures above the critical flame
      temp.
2119 IF (ZZ.HAT_F*(H.F.0+HOC_EXTINCT) + ZZ.HAT_A*H.A.0 + ZZ.HAT_P*H.P.0 < &
2120 (ZZ.HAT_F+ZZ.HAT_A+ZZ.HAT_P)*H.P.N) EXTINCT_3.1 = .TRUE. ! FED Tech Guide (5.19) !edited name
2121 ENDF
2122
2123 END FUNCTION EXTINCT_3.1 !edited name
2124
2125 !added for 6.2.0
2126 SUBROUTINE GET_REV_fire(MODULE_REV,MODULE_DATE)
2127 INTEGER,INTENT(INOUT) :: MODULE_REV
2128 CHARACTER(255),INTENT(INOUT) :: MODULE_DATE
2129
2130 WRITE(MODULE_DATE,'(A)') firerev(INDEX(firerev,':')+2:LEN_TRIM(firerev)-2)
2131 READ(MODULE_DATE,'(15)') MODULE_REV
2132 WRITE(MODULE_DATE,'(A)') firedate
2133
2134 END SUBROUTINE GET_REV_fire
2135
2136 END MODULE FIRE

```

## A.2 read.f90

```

1  MODULE READ_INPUT
2
3  USE PRECISION_PARAMETERS
4  USE MESH_VARIABLES
5  USE GLOBAL_CONSTANTS
6  USE TRAN
7  USE MESH_POINTERS
8  USE OUTPUT_DATA
9  USE COMP_FUNCTIONS, ONLY: SECOND, CHECKREAD, SHUTDOWN, CHECK_XB, SCAN_INPUT_FILE
10 USE MEMORY_FUNCTIONS, ONLY: ChkMemErr, REALLOCATE2D
11 USE COMP_FUNCTIONS, ONLY: GET_INPUT_FILE
12 USE MISC_FUNCTIONS, ONLY: SEARCH_CONTROLLER, WRITE_SUMMARY_INFO
13 USE EVAC, ONLY: READ_EVAC
14 USE HVAC_ROUTINES, ONLY: READ_HVAC, PROC_HVAC
15 USE COMPLEX_GEOMETRY, ONLY: READ_GEOM
16 USE MPI
17
18 !Sesa
19 USE penalization
20 USE SCRC
21
22 IMPLICIT NONE
23 PRIVATE
24
25 PUBLIC READ_DATA, READ_STOP
26
27 CHARACTER(LABEL_LENGTH) :: ID, MB, ODE_SOLVER
28 CHARACTER(MESSAGE_LENGTH) :: MESSAGE, FYI
29 CHARACTER(LABEL_LENGTH) :: SURF_DEFAULT='INERT', EVAC_SURF_DEFAULT='INERT', FUEL_RADCAL_ID='METHANE',
      LES_FILTER_WIDTH='null'
30 LOGICAL :: EX, THICKEN_OBSTRUCTIONS, BAD, IDEAL=.FALSE., SIMPLE_FUEL_DEFINED=.FALSE., TARGET_PARTICLES_INCLUDED=.FALSE
31
32 REAL(EB) :: XB(6), TEXTURE_ORIGIN(3)
33 REAL(EB) :: PBX, PBY, PBZ
34 REAL(EB) :: MW_MIN, MW_MAX
35 REAL(EB) :: REAC_ATOM_ERROR, REAC_MASS_ERROR, HUMIDITY=-1.EB
36 INTEGER :: I, J, K, IZERO, IOS, N_INIT_RESERVED, MAX_LEAK_PATHS, LDUM(10)
37 INTEGER :: FUEL_SMIX_INDEX ! Simple chemistry fuel index
38 TYPE(MESH_TYPE), POINTER :: M=>NULL()
39 TYPE(OBSTRUCTION_TYPE), POINTER :: OB=>NULL()
40 TYPE(VENTS_TYPE), POINTER :: VT=>NULL()
41 TYPE(SURFACE_TYPE), POINTER :: SF=>NULL()
42 TYPE(MATERIAL_TYPE), POINTER :: M1=>NULL()
43 TYPE(REACTION_TYPE), POINTER :: RN=>NULL()
44
45 CONTAINS

```

```

46
47
48 SUBROUTINE READDATA(DT)
49
50 REAL(EB) :: DT, T
51
52 ! Create an array of output QUANTITY names that are included in the various NAMELIST groups
53
54 CALL DEFINE.OUTPUT.QUANTITIES
55
56 ! Get the name of the input file by reading the command line argument
57
58 CALL GET.INPUT.FILE
59
60 ! If no input file is given, just print out the version number and stop
61
62 IF (FN.INPUT(1:1)==' ') THEN
63   IF (MYID==0) THEN
64     CALL WRITE.SUMMARY.INFO(LU.ERR)
65     WRITE(LU.ERR, '(A)') ' Consult FDS Users Guide Chapter, Running FDS, for further instructions.'
66     WRITE(LU.ERR, '(A)') ' Hit Enter to Escape...'
67     READ(5, *, ERR=2, END=2)
68   ENDIF
69   2 STOP
70 ENDIF
71
72 ! Stop FDS if the input file cannot be found in the current directory
73
74 INQUIRE(FILE=FN.INPUT, EXIST=EX)
75 IF (.NOT.EX) THEN
76   IF (MYID==0) WRITE(LU.ERR, '(A,A,A)') "ERROR: The file , ", TRIM(FN.INPUT) , ", does not exist in the current
       directory"
77 STOP
78 ENDIF
79
80 ! Allocate the global orientation vector
81
82 N.ORIENTATION.VECTOR = 0
83 ALLOCATE(ORIENTATION.VECTOR(3,10))
84
85 ! Set humidity data
86 CALL CALC.H2O.HV
87
88 ! Open the input file
89
90 OPEN(LU.INPUT, FILE=FN.INPUT, ACTION='READ')
91
92 ! Read the input file, NAMELIST group by NAMELIST group
93
94 CALL READ.DEAD ; IF (STOP.STATUS==SETUP.STOP) RETURN
95 CALL READ.HEAD ; IF (STOP.STATUS==SETUP.STOP) RETURN
96 CALL READ.MISC ; IF (STOP.STATUS==SETUP.STOP) RETURN
97 CALL READ.MULT ; IF (STOP.STATUS==SETUP.STOP) RETURN
98 CALL READ.MESH(1) ; IF (STOP.STATUS==SETUP.STOP) RETURN
99 CALL READ.EVAC(1) ; IF (STOP.STATUS==SETUP.STOP) RETURN
100 CALL READ.MESH(2) ; IF (STOP.STATUS==SETUP.STOP) RETURN
101 CALL READ.TRAN ; IF (STOP.STATUS==SETUP.STOP) RETURN
102 CALL READ.WIND ; IF (STOP.STATUS==SETUP.STOP) RETURN
103 CALL READ.TIME(DT) ; IF (STOP.STATUS==SETUP.STOP) RETURN
104 CALL READ.PRES ; IF (STOP.STATUS==SETUP.STOP) RETURN
105 CALL READ.REAC ; IF (STOP.STATUS==SETUP.STOP) RETURN
106 CALL READ.SPEC ; IF (STOP.STATUS==SETUP.STOP) RETURN
107 CALL PROC.REAC.1 ; IF (STOP.STATUS==SETUP.STOP) RETURN
108 CALL READ.RADI ; IF (STOP.STATUS==SETUP.STOP) RETURN
109 CALL READ.PROP ; IF (STOP.STATUS==SETUP.STOP) RETURN
110 CALL READ.DEVC ; IF (STOP.STATUS==SETUP.STOP) RETURN
111 CALL READ.PART ; IF (STOP.STATUS==SETUP.STOP) RETURN
112 CALL READ.CTRL ; IF (STOP.STATUS==SETUP.STOP) RETURN
113 CALL READ.MATL ; IF (STOP.STATUS==SETUP.STOP) RETURN
114 CALL READ.SURF ; IF (STOP.STATUS==SETUP.STOP) RETURN
115 CALL READ.CSVF ; IF (STOP.STATUS==SETUP.STOP) RETURN
116 CALL READ.OBST ; IF (STOP.STATUS==SETUP.STOP) RETURN
117 CALL READ.GEOM ; IF (STOP.STATUS==SETUP.STOP) RETURN
118 CALL READ.VENT ; IF (STOP.STATUS==SETUP.STOP) RETURN
119 CALL READ.ZONE ; IF (STOP.STATUS==SETUP.STOP) RETURN
120 CALL READ.EVAC(2) ; IF (STOP.STATUS==SETUP.STOP) RETURN
121 CALL READ.HVAC ; IF (STOP.STATUS==SETUP.STOP) RETURN
122 CALL PROC.SURF.1 ; IF (STOP.STATUS==SETUP.STOP) RETURN
123 CALL READ.RAMP ; IF (STOP.STATUS==SETUP.STOP) RETURN
124 CALL PROC.WIND ; IF (STOP.STATUS==SETUP.STOP) RETURN
125 CALL PROC.SMIX ; IF (STOP.STATUS==SETUP.STOP) RETURN
126 CALL PROC.REAC.2 ; IF (STOP.STATUS==SETUP.STOP) RETURN
127 CALL PROC.HVAC ; IF (STOP.STATUS==SETUP.STOP) RETURN
128 CALL PROC.MATL ; IF (STOP.STATUS==SETUP.STOP) RETURN
129 CALL PROC.SURF.2 ; IF (STOP.STATUS==SETUP.STOP) RETURN
130 CALL READ.DUMP ; IF (STOP.STATUS==SETUP.STOP) RETURN
131 CALL READ.CLIP ; IF (STOP.STATUS==SETUP.STOP) RETURN
132 CALL PROC.WALL ; IF (STOP.STATUS==SETUP.STOP) RETURN

```

Source Code files for edited portions of FDS

```

133 CALL PROC.PART      ; IF (STOP_STATUS==SETUP_STOP) RETURN
134 CALL READ_INIT     ; IF (STOP_STATUS==SETUP_STOP) RETURN
135 CALL READ.TABL     ; IF (STOP_STATUS==SETUP_STOP) RETURN
136 CALL PROC.CTRL     ; IF (STOP_STATUS==SETUP_STOP) RETURN
137 CALL PROC.PROP     ; IF (STOP_STATUS==SETUP_STOP) RETURN
138 CALL PROC.DEVC(DT) ; IF (STOP_STATUS==SETUP_STOP) RETURN
139 CALL PROC.OBST     ; IF (STOP_STATUS==SETUP_STOP) RETURN
140 CALL READ.PROF     ; IF (STOP_STATUS==SETUP_STOP) RETURN
141 CALL READ.SLCF     ; IF (STOP_STATUS==SETUP_STOP) RETURN
142 CALL READ.ISOF     ; IF (STOP_STATUS==SETUP_STOP) RETURN
143 CALL READ.BNDF     ; IF (STOP_STATUS==SETUP_STOP) RETURN
144 CALL READ.BNDE     ; IF (STOP_STATUS==SETUP_STOP) RETURN
145
146 !Sesa
147 CALL read.pen
148 CALL read.trunks
149
150 ! Close the input file , and never open it again
151
152 CLOSE (LU.INPUT)
153
154 ! Set QUANTITY ambient values
155
156 CALL SET_QUANTITIES.AMBIENT
157
158 END SUBROUTINE READ.DATA
159
160
161 SUBROUTINE READ.DEAD
162
163 CHARACTER(80) :: BAD.TEXT
164
165 ! Look for hidden carriage return characters at the beginning of namelist input lines.
166
167 REWIND(LU.INPUT) ; INPUT_FILE.LINE_NUMBER = 0
168 CALL SCAN_INPUT_FILE(LU.INPUT,IOS,BAD.TEXT)
169 IF (IOS==0) THEN
170 WRITE(MESSAGE,'(3A)') 'ERROR: Hidden carriage return character in line starting with: ',BAD.TEXT(2:15),'...'
171 CALL SHUTDOWN(MESSAGE)
172 ENDIF
173
174 ! Look for outdated NAMELIST groups and stop the run if any are found.
175
176 REWIND(LU.INPUT) ; INPUT_FILE.LINE_NUMBER = 0
177 CALL CHECKREAD('GRID',LU.INPUT,IOS)
178 IF (IOS==0) CALL SHUTDOWN('ERROR: GRID is no longer a valid NAMELIST group. Read User Guide discussion on MESH.')
179 REWIND(LU.INPUT) ; INPUT_FILE.LINE_NUMBER = 0
180 CALL CHECKREAD('HEAT',LU.INPUT,IOS)
181 IF (IOS==0) CALL SHUTDOWN('ERROR: HEAT is no longer a valid NAMELIST group. Read User Guide discussion on PROP
and DEVC.')
182 REWIND(LU.INPUT) ; INPUT_FILE.LINE_NUMBER = 0
183 CALL CHECKREAD('PDIM',LU.INPUT,IOS)
184 IF (IOS==0) CALL SHUTDOWN('ERROR: PDIM is no longer a valid NAMELIST group. Read User Guide discussion on MESH.')
185 REWIND(LU.INPUT) ; INPUT_FILE.LINE_NUMBER = 0
186 CALL CHECKREAD('PIPE',LU.INPUT,IOS)
187 IF (IOS==0) CALL SHUTDOWN('ERROR: PIPE is no longer a valid NAMELIST group. Read User Guide discussion on PROP
and DEVC.')
188 REWIND(LU.INPUT) ; INPUT_FILE.LINE_NUMBER = 0
189 CALL CHECKREAD('PL3D',LU.INPUT,IOS)
190 IF (IOS==0) CALL SHUTDOWN('ERROR: PL3D is no longer a valid NAMELIST group. Read User Guide discussion on DUMP.')
191 REWIND(LU.INPUT) ; INPUT_FILE.LINE_NUMBER = 0
192 CALL CHECKREAD('SMOD',LU.INPUT,IOS)
193 IF (IOS==0) CALL SHUTDOWN('ERROR: SMOD is no longer a valid NAMELIST group. Read User Guide discussion on DEVC.')
194 REWIND(LU.INPUT) ; INPUT_FILE.LINE_NUMBER = 0
195 CALL CHECKREAD('SPRK',LU.INPUT,IOS)
196 IF (IOS==0) CALL SHUTDOWN('ERROR: SPRK is no longer a valid NAMELIST group. Read User Guide discussion on PROP
and DEVC.')
197 REWIND(LU.INPUT) ; INPUT_FILE.LINE_NUMBER = 0
198 CALL CHECKREAD('THCP',LU.INPUT,IOS)
199 IF (IOS==0) CALL SHUTDOWN('ERROR: THCP is no longer a valid NAMELIST group. Read User Guide discussion on DEVC.')
200
201 REWIND(LU.INPUT) ; INPUT_FILE.LINE_NUMBER = 0
202
203 END SUBROUTINE READ.DEAD
204
205
206 SUBROUTINE READ.HEAD
207 INTEGER :: NAMELENGTH
208 NAMELIST /HEAD/ CHID,FYI,STOPFDS,TITLE
209
210 CHID = 'null'
211 TITLE = ' '
212 STOPFDS=-1
213
214 REWIND(LU.INPUT) ; INPUT_FILE.LINE_NUMBER = 0
215 HEADLOOP: DO
216 CALL CHECKREAD('HEAD',LU.INPUT,IOS)
217 IF (IOS==1) EXIT HEADLOOP

```

## Source Code files for edited portions of FDS

```

218 READ(LU.INPUT,HEAD,END=13,ERR=14,IOSTAT=IOS)
219 14 IF (IOS>0) THEN ; CALL SHUTDOWN('ERROR: Problem with HEAD line') ; RETURN ; ENDIF
220 ENDDO HEADLOOP
221 13 REWIND(LU.INPUT) ; INPUT_FILE.LINE_NUMBER = 0
222
223 CLOOP: DO I=1,39
224 IF (CHID(1:I)!='.') THEN ; CALL SHUTDOWN('ERROR: No periods allowed in CHID') ; RETURN ; ENDIF
225 IF (CHID(1:I)!=' ') EXIT CLOOP
226 ENDDO CLOOP
227
228 IF (TRIM(CHID)=='null') THEN
229 NAMELENGTH = LEN(TRIM(FN.INPUT))
230 ROOTNAME: DO I=NAMELENGTH,2,-1
231 IF (FN.INPUT(1:I)!='.') THEN
232 WRITE(CHID,'(A)') FN.INPUT(1:I-1)
233 EXIT ROOTNAME
234 ENDIF
235 END DO ROOTNAME
236 ENDF
237
238 ! Define and look for a stop file
239
240 FN_STOP = TRIM(CHID)//'.stop'
241 INQUIRE(FILE=FN_STOP,EXIST=EX)
242 IF (EX) THEN
243 STOP_AT_ITER=READ_STOP() ! READ_STOP() returns 0 if there is nothing in the .stop file
244 IF (STOP_AT_ITER<=0) THEN
245 WRITE(MESSAGE,'(A,A,A)') "ERROR: Remove the file, ",TRIM(FN_STOP)," , from the current directory"
246 CALL SHUTDOWN(MESSAGE) ; RETURN
247 ELSE
248 WRITE(LU.ERR,'(A,A,A)') "NOTE: The file, ",TRIM(FN_STOP)," , was detected."
249 WRITE(LU.ERR,'(A,I0,A)') "This FDS run will stop after ",STOP_AT_ITER," iterations."
250 ENDF
251 ELSE
252 IF (STOPFDS>=0) THEN
253 STOP_AT_ITER = STOPFDS
254 WRITE(LU.ERR,'(A,A,A)') "NOTE: The STOPFDS keyword was detected on the &HEAD line."
255 WRITE(LU.ERR,'(A,I0,A)') "This FDS run will stop after ",STOP_AT_ITER," iterations."
256 ENDF
257 ENDF
258
259 END SUBROUTINE READ.HEAD
260
261
262 INTEGER FUNCTION READ_STOP()
263
264 ! if a stop file exists and it contains a positive integer then
265 ! stop the fds run at when it computes that number of iterations
266
267 INTEGER :: IERROR
268
269 READ_STOP=0
270
271 ! this routine is only called if the stop file exists
272
273 OPEN(UNIT=LU_STOP,FILE=FN_STOP,FORM='FORMATTED',STATUS='OLD',IOSTAT=IERROR)
274 IF (IERROR==0) THEN
275 READ(LU_STOP,'(I5)',END=10,IOSTAT=IERROR) READ_STOP
276 IF (IERROR/=0) READ_STOP=0
277 ENDF
278 10 CLOSE(LU_STOP)
279
280 END FUNCTION READ_STOP
281
282
283 SUBROUTINE READ_MESH(IMODE)
284 USE GLOBAL_CONSTANTS, ONLY : OPENMP_USED_THREADS, OPENMP_USER_SET_THREADS, USE_OPENMP
285 USE EVAC, ONLY: N_DOORS, N_EXITS, N_CO_EXITS, EVAC_EMESH_EXITS_TYPE, EMESH_EXITS, EMESH_ID, EMESH_IJK, EMESH_XB,
286 &
287 EMESH_NM, N_DOOR_MESHES, EMESH_NFIELDS, HUMAN_SMOKE_HEIGHT, EVAC_DELTA_SEE, &
288 EMESH_STAIRS, EVAC_EMESH_STAIRS_TYPE, N_STRS, INPUT_EVAC_GRIDS, NO_EVAC_MESHES
289 INTEGER, INTENT(IN) :: IMODE
290 INTEGER :: IJK(3),NM,NM2,CURRENT_MPL_PROCESS,MPL_PROCESS,RGB(3),LEVEL,N_MESH_NEW,N,II,JJ,KK,NMESHES_READ,NNN,
291 NEVAC_MESHES,IERR,&
292 NMESHES_EVAC, NMESHES_FIRE, NMLEVAC, N_THREADS
293 INTEGER, ALLOCATABLE, DIMENSION(:) :: NEIGHBOR_LIST
294 LOGICAL :: EVACUATION, EVACHUMANS
295 REAL(FB) :: EVAC_Z_OFFSET, XB1, XB2, XB3, XB4, XB5, XB6
296 CHARACTER(25) :: COLOR
297 CHARACTER(LABEL_LENGTH) :: MULT_ID, PERIODIC_MESH_IDS(3)
298 NAMELIST /MESH/ COLOR, CYLINDRICAL, EVACUATION, EVACHUMANS, EVAC_Z_OFFSET, FYI, ID, IJK, LEVEL, MPL_PROCESS, MULT_ID,
299 PERIODIC_MESH_IDS, &
300 RGB, XB, N_THREADS
301 TYPE (MESH_TYPE), POINTER :: M, M2
302 TYPE (MULTIPLIER_TYPE), POINTER :: MR
303
304 NMESHES = 0
305 NMESHES_READ = 0
306

```



```

303 NMESHES.EVAC = 0
304 NMESHES.FIRE = 0
305 NEVAC.MESHES = 0
306 IF (IMODE==1) THEN
307 NO.EVAC.MESHES = .TRUE.
308 INPUT.EVAC.GRIDS = 0
309 IF (NO.EVACUATION) THEN
310 N.EVAC = 0
311 RETURN
312 END IF
313 END IF
314
315 REWIND(LU.INPUT) ; INPUT_FILE.LINE_NUMBER = 0
316 COUNT.MESHLOOP: DO
317 CALL CHECKREAD('MESH',LU.INPUT,IOS)
318 IF (IOS==1) EXIT COUNT.MESHLOOP
319 MULT.ID = 'null'
320 EVACUATION = .FALSE.
321 EVACHUMANS = .FALSE.
322 READ(LU.INPUT,MESH,END=15,ERR=16,IOSTAT=IOS)
323 NMESHES.READ = NMESHES.READ + 1
324 IF (.NOT.EVACUATION .AND. EVACUATION) CYCLE COUNT.MESHLOOP ! skip evacuation meshes
325 IF (EVACUATION.DRILL .AND. .NOT.EVACUATION) CYCLE COUNT.MESHLOOP ! skip fire meshes
326 IF (EVACUATION.MC.MODE .AND. .NOT.EVACUATION) CYCLE COUNT.MESHLOOP ! skip fire meshes
327 IF (EVACUATION) NEVAC.MESHES = NEVAC.MESHES + 1
328 IF (IMODE==1 .AND. EVACHUMANS) NO.EVAC.MESHES = .FALSE.
329 IF (IMODE==1 .AND. EVACHUMANS) INPUT.EVAC.GRIDS = INPUT.EVAC.GRIDS + 1
330 N.MESHNEW = 0
331 IF (MULT.ID=='null') THEN
332 N.MESHNEW = 1
333 ELSE
334 DO N=1,N.MULT
335 MR => MULTIPLIER(N)
336 IF (MULT.ID==MR%ID) N.MESHNEW = MR*N.COPIES
337 ENDDO
338 IF (N.MESHNEW==0) THEN
339 WRITE(MESSAGE,'(A,A,A,I0)') 'ERROR: MULT line ', TRIM(MULT.ID), ' not found on MESH line', &
340 NMESHES.READ
341 CALL SHUTDOWN(MESSAGE) ; RETURN
342 ENDIF
343 ENDIF
344 NMESHES = NMESHES + N.MESHNEW
345 IF (.NOT.EVACUATION) NMESHES.FIRE = NMESHES.FIRE + N.MESHNEW
346 16 IF (IOS>0) THEN ; CALL SHUTDOWN('ERROR: Problem with MESH line.') ; RETURN ; ENDIF
347 ENDDO COUNT.MESHLOOP
348 15 CONTINUE
349
350 EVAC.MODE.IF: IF (IMODE==1) THEN
351 REWIND(LU.INPUT) ; INPUT_FILE.LINE_NUMBER = 0
352 IF (NO.EVAC.MESHES) THEN
353 NO.EVACUATION = .TRUE.
354 EVACUATION.DRILL = .FALSE.
355 EVACUATION.MC.MODE = .FALSE.
356 N.EVAC = 0
357 RETURN
358 END IF
359 ALLOCATE(EMESH.ID( MAX(1,INPUT.EVAC.GRIDS) ), STAT=IZERO)
360 CALL ChkMemErr('READ.EVAC','EMESH.ID',IZERO)
361 ALLOCATE(EMESH.XB(6, MAX(1,INPUT.EVAC.GRIDS) ), STAT=IZERO)
362 CALL ChkMemErr('READ.EVAC','EMESH.XB',IZERO)
363 ALLOCATE(EMESH.IJK(3, MAX(1,INPUT.EVAC.GRIDS) ), STAT=IZERO)
364 CALL ChkMemErr('READ.EVAC','EMESH.IJK',IZERO)
365
366 NM = 0
367 EVAC.MESHLOOP: DO N = 1, NMESHES.READ
368 ! Set evacuation MESH defaults
369 IJK(1)= 10
370 IJK(2)= 10
371 IJK(3)= 1
372 XB(1) = 0. .EB
373 XB(2) = 1. .EB
374 XB(3) = 0. .EB
375 XB(4) = 1. .EB
376 XB(5) = 0. .EB
377 XB(6) = 1. .EB
378 RGB = -1
379 COLOR = 'null'
380 ID = 'null'
381 EVACUATION = .FALSE.
382 EVACHUMANS = .FALSE.
383 ! Read the MESH line
384 CALL CHECKREAD('MESH',LU.INPUT,IOS)
385 IF (IOS==1) EXIT EVAC.MESHLOOP
386 READ(LU.INPUT, MESH)
387 IF (.NOT.EVACUATION) CYCLE EVAC.MESHLOOP ! skip fire meshes
388 IF (.NOT.EVACHUMANS .AND. EVACUATION) CYCLE EVAC.MESHLOOP ! skip additional evac meshes
389 NM = NM + 1
390 ! Reorder XB coordinates if necessary

```

```

391 CALL CHECK_XB(XB)
392 EMESH_ID(NM) = TRIM(ID)
393 EMESH_IJK(1,NM) = IJK(1)
394 EMESH_IJK(2,NM) = IJK(2)
395 EMESH_IJK(3,NM) = IJK(3)
396 EMESH_XB(1,NM) = XB(1)
397 EMESH_XB(2,NM) = XB(2)
398 EMESH_XB(3,NM) = XB(3)
399 EMESH_XB(4,NM) = XB(4)
400 EMESH_XB(5,NM) = XB(5)
401 EMESH_XB(6,NM) = XB(6)
402 END DO EVAC_MESH_LOOP
403 REWIND(LU_INPUT) ; INPUT_FILE_LINE_NUMBER = 0
404 RETURN
405 END IF EVAC_MODE_IF
406
407 IF (.NOT. NO_EVACUATION) NMESHES = NMESHES + N_DOOR_MESHES + NEVAC_MESHES
408 IF (.NOT. NO_EVACUATION) NMESHES = NMESHES + N_STRS
409
410 NMESHES_EVAC = NMESHES - NMESHES_FIRE
411
412 ! Stop the calculation if the number of MPI processes is greater than the number of meshes
413
414 IF (NO_EVACUATION) THEN
415 IF (NMESHES < N_MPI_PROCESSES) THEN
416 CALL MPI_FINALIZE(IERR)
417 WRITE(MESSAGE,'(A,I0,A,I0)') 'ERROR: The number of MPI processes, ',N_MPI_PROCESSES,', exceeds the number of
meshes, ',NMESHES
418 CALL SHUTDOWN(MESSAGE) ; RETURN
419 ENDIF
420 ELSE
421 IF (NMESHES_FIRE+1 < N_MPI_PROCESSES) THEN
422 CALL MPI_FINALIZE(IERR)
423 WRITE(MESSAGE,'(A,I0,A,I0)') 'ERROR: The number of MPI processes, ',N_MPI_PROCESSES,&
', exceeds the number of fire meshes + 1, ',NMESHES_FIRE+1
424 CALL SHUTDOWN(MESSAGE) ; RETURN
425 ENDIF
426 ENDIF
427
428 ! Allocate parameters associated with the mesh.
429
430 ALLOCATE(MESHES(NMESHES),STAT=IZERO)
431 CALL ChkMemErr('READ','MESHES',IZERO)
432 ALLOCATE(PROCESS(NMESHES),STAT=IZERO)
433 CALL ChkMemErr('READ','PROCESS',IZERO)
434 ALLOCATE(MESHNAME(NMESHES),STAT=IZERO)
435 CALL ChkMemErr('READ','MESHNAME',IZERO)
436 ALLOCATE(PERIODIC_MESH_NAMES(NMESHES,3),STAT=IZERO)
437 CALL ChkMemErr('READ','PERIODIC_MESH_NAMES',IZERO)
438 ALLOCATE(CHANGE_TIME_STEP_INDEX(NMESHES),STAT=IZERO)
439 CALL ChkMemErr('READ','CHANGE_TIME_STEP_INDEX',IZERO)
440 CHANGE_TIME_STEP_INDEX = 0
441 ALLOCATE(EVACUATION_ONLY(NMESHES),STAT=IZERO)
442 CALL ChkMemErr('READ','EVACUATION_ONLY',IZERO)
443 EVACUATION_ONLY(1:NMESHES_FIRE) = .FALSE.
444 IF (NMESHES_FIRE < NMESHES) EVACUATION_ONLY(NMESHES_FIRE+1:NMESHES) = .TRUE.
445 ALLOCATE(EVACUATION_SKIP(NMESHES),STAT=IZERO)
446 CALL ChkMemErr('READ','EVACUATION_SKIP',IZERO)
447 EVACUATION_SKIP = .FALSE.
448 ALLOCATE(EVACUATION_Z_OFFSET(NMESHES),STAT=IZERO)
449 CALL ChkMemErr('READ','EVACUATION_Z_OFFSET',IZERO)
450 EVACUATION_Z_OFFSET = 1.0_EB
451
452 ! Read in the Mesh lines from Input file
453
454 REWIND(LU_INPUT) ; INPUT_FILE_LINE_NUMBER = 0
455
456 IF (NMESHES < 1) THEN ; CALL SHUTDOWN('ERROR: No MESH line(s) defined.') ; RETURN ; ENDIF
457
458 NM = 0
459
460 MESH_LOOP: DO N=1,NMESHES_READ
461
462 ! Set MESH defaults
463
464 IJK(1) = 10
465 IJK(2) = 10
466 IJK(3) = 10
467 TWO_D = .FALSE.
468 XB(1) = 0._EB
469 XB(2) = 1._EB
470 XB(3) = 0._EB
471 XB(4) = 1._EB
472 XB(5) = 0._EB
473 XB(6) = 1._EB
474 RGB = -1
475 COLOR = 'null'
476 CYLINDRICAL = .FALSE.
477

```

```

478 ID = 'null'
479 EVACUATION = .FALSE.
480 EVAC_Z_OFFSET = 1.0_EB
481 EVACHUMANS = .FALSE.
482 MPL_PROCESS = -1
483 LEVEL = 0
484 MULT_ID = 'null'
485 PERIODIC_MESH_IDS = 'pnull'
486 N_THREADS = -1
487
488 ! Read the MESH line
489
490 CALL CHECKREAD( 'MESH', LU_INPUT, IOS)
491 IF (IOS==1) EXIT MESHLOOP
492 READ(LU_INPUT, MESH)
493
494 IF (NO.EVACUATION .AND. EVACUATION) CYCLE MESHLOOP ! skip evacuation meshes
495 IF (EVACUATION_DRILL .AND. .NOT.EVACUATION) CYCLE MESHLOOP ! skip fire meshes
496 IF (EVACUATION.MC.MODE .AND. .NOT.EVACUATION) CYCLE MESHLOOP ! skip fire meshes
497
498 ! Reorder XB coordinates if necessary
499
500 CALL CHECK_XB(XB)
501
502 ! Multiply meshes if need be
503
504 MR => MULTIPLIER(0)
505 DO NNN=1, NMULT
506 IF (MULT_ID==MULTIPLIER(NNN)%ID) MR => MULTIPLIER(NNN)
507 ENDDO
508
509 K_MULT_LOOP: DO KK=MR%K_LOWER, MR%K_UPPER
510 J_MULT_LOOP: DO JJ=MR%J_LOWER, MR%J_UPPER
511 I_MULT_LOOP: DO II=MR%I_LOWER, MR%I_UPPER
512
513 IF (.NOT.MR%SEQUENTIAL) THEN
514 XB1 = XB(1) + MR%DX0 + II*MR%DXB(1)
515 XB2 = XB(2) + MR%DX0 + II*MR%DXB(2)
516 XB3 = XB(3) + MR%DY0 + JJ*MR%DXB(3)
517 XB4 = XB(4) + MR%DY0 + JJ*MR%DXB(4)
518 XB5 = XB(5) + MR%DZ0 + KK*MR%DXB(5)
519 XB6 = XB(6) + MR%DZ0 + KK*MR%DXB(6)
520 ELSE
521 XB1 = XB(1) + MR%DX0 + II*MR%DXB(1)
522 XB2 = XB(2) + MR%DX0 + II*MR%DXB(2)
523 XB3 = XB(3) + MR%DY0 + II*MR%DXB(3)
524 XB4 = XB(4) + MR%DY0 + II*MR%DXB(4)
525 XB5 = XB(5) + MR%DZ0 + II*MR%DXB(5)
526 XB6 = XB(6) + MR%DZ0 + II*MR%DXB(6)
527 ENDIF
528
529 ! Increase the MESH counter by 1
530
531 NM = NM + 1
532
533 ! Determine which PROCESS to assign the MESH to
534
535 IF (MPL_PROCESS > -1) THEN
536 CURRENT_MPL_PROCESS = MPL_PROCESS
537 IF (CURRENT_MPL_PROCESS > N_MPL_PROCESSES - 1) THEN
538 IF (N_MPL_PROCESSES > 1) THEN
539 WRITE(MESSAGE, '(A,I0,A)') 'ERROR: MPL_PROCESS for MESH ', NM, ' greater than total number of processes'
540 CALL SHUTDOWN(MESSAGE) ; RETURN
541 ELSE
542 ! Prevents fatal error when testing a run on a single core with MPL_PROCESS set for meshes
543 WRITE(MESSAGE, '(A,I0,A)') 'WARNING: MPL_PROCESS set for MESH ', NM, ' and only one MPI process exists'
544 IF (MYID==0) WRITE(LU_ERR, '(A)') TRIM(MESSAGE)
545 CURRENT_MPL_PROCESS=0
546 ENDIF
547 ENDIF
548 ELSE
549 CURRENT_MPL_PROCESS = MIN(NM-1, N_MPL_PROCESSES-1)
550 ENDIF
551
552 ! Fill in MESH related variables
553
554 M => MESHES(NM)
555 M%MESH_LEVEL = LEVEL
556 M%IBAR = IJK(1)
557 M%JBAR = IJK(2)
558 M%KBAR = IJK(3)
559 IBAR_MAX = MAX(IBAR_MAX, M%IBAR)
560 JBAR_MAX = MAX(JBAR_MAX, M%JBAR)
561 KBAR_MAX = MAX(KBAR_MAX, M%KBAR)
562 M%N_EXTERNAL_WALL_CELLS = 2*M%IBAR*M%JBAR+2*M%JBAR*M%KBAR+2*M%IBAR*M%KBAR
563
564 IF (EVACUATION) EVACUATION_ONLY(NM) = .TRUE.
565 IF (EVACHUMANS) EVACUATION_SKIP(NM) = .TRUE.

```

Source Code files for edited portions of FDS

```

566 IF (EVACUATION) EVACUATION_Z_OFFSET(NM) = EVAC_Z_OFFSET
567 IF (EVACUATION) M%N.EXTERNAL_WALL_CELLS = 2*M%IBAR*M%KBAR+2*M%JBAR*M%KBAR
568 IF (EVACUATION .AND. .NOT.EVAC_HUMANS) THEN
569 WRITE(MESSAGE,'(A)') 'ERROR: NO DOOR FLOW EVACUATION MESHES IN FDS6'
570 CALL SHUTDOWN(MESSAGE) ; RETURN
571 ENDIF
572
573 IF (M%JBAR==1) TWO_D = .TRUE.
574 IF (TWO_D .AND. M%JBAR/=1) THEN
575 WRITE(MESSAGE,'(A)') 'ERROR: IJK(2) must be 1 for all grids in 2D Calculation'
576 CALL SHUTDOWN(MESSAGE) ; RETURN
577 ENDIF
578 IF (EVACUATION .AND. M%KBAR/=1) THEN
579 WRITE(MESSAGE,'(A)') 'ERROR: IJK(3) must be 1 for all evacuation grids'
580 CALL SHUTDOWN(MESSAGE) ; RETURN
581 ENDIF
582
583 ! Associate the MESH with the PROCESS
584
585 IF (MYID==CURRENT_MPL_PROCESS) THEN
586 LOWER_MESH_INDEX = MIN(LOWER_MESH_INDEX,NM)
587 UPPER_MESH_INDEX = MAX(UPPER_MESH_INDEX,NM)
588 ENDIF
589
590 PROCESS(NM) = CURRENT_MPL_PROCESS
591 IF (MYID==0 .AND. VERBOSE) &
592 WRITE(LU_ERR,'(A,I0,A,I0)') ' Mesh ',NM,' is assigned to MPI Process ',PROCESS(NM)
593 IF (EVACUATION_ONLY(NM) .AND. (N_MPL_PROCESSES>1)) EVAC_PROCESS = N_MPL_PROCESSES-1
594
595 ! Check the number of OMP threads for a valid value (positive, larger than 0), -1 indicates default unchanged
596 ! value
597 IF (N_THREADS < 1 .AND. N_THREADS /= -1) THEN
598 WRITE(MESSAGE,'(A)') 'ERROR: N_THREADS must be at least 1'
599 CALL SHUTDOWN(MESSAGE) ; RETURN
600 ENDIF
601
602 ! If OMP number of threads is explicitly set for this mesh and the mesh is assigned to this MPI process,
603 ! then set this value
604 IF (MYID == PROCESS(NM) .AND. N_THREADS > 0) THEN
605 ! Check if OPENMP is active
606 IF (USE_OPENMP .NEQV. .TRUE.) THEN
607 WRITE(MESSAGE,'(A)') 'ERROR: setting N_THREADS, but OPENMP is not active'
608 CALL SHUTDOWN(MESSAGE) ; RETURN
609 END IF
610
611 ! Check if the process' thread number was already set in a previous mesh definition
612 IF (OPENMP_USER_SET_THREADS .EQV. .TRUE.) THEN
613 ! Check if previous definitions are consistent
614 IF (N_THREADS .NE. OPENMP_USED_THREADS) THEN
615 WRITE(MESSAGE,'(A)') 'ERROR: N_THREADS not consistent for MPI process'
616 CALL SHUTDOWN(MESSAGE) ; RETURN
617 END IF
618 END IF
619
620 ! set the value-changed-flag and the new thread number
621 OPENMP_USER_SET_THREADS = .TRUE.
622 OPENMP_USED_THREADS = N_THREADS
623 END IF
624
625 ! Mesh boundary colors
626 IF (ANY(RGB<0) .AND. COLOR=='null') COLOR = 'BLACK'
627 IF (COLOR /= 'null') CALL COLOR2RGB(RGB,COLOR)
628 ALLOCATE(M%RGB(3))
629 M%RGB = RGB
630
631 ! Mesh Geometry and Name
632
633 PERIODIC_MESH_NAMES(NM,:) = PERIODIC_MESH_IDS(:)
634 WRITE(MESH_NAME(NM),'(A,17.7)') 'MESH_',NM
635 IF (ID/'null') MESH_NAME(NM) = ID
636
637 ! Process Physical Coordinates
638
639 IF (XB2-XB1<TWO_EPSILON_EB) THEN
640 WRITE(MESSAGE,'(A,I0)') 'ERROR: XMIN > XMAX on MESH ', NM
641 CALL SHUTDOWN(MESSAGE) ; RETURN
642 ENDIF
643 IF (XB4-XB3<TWO_EPSILON_EB) THEN
644 WRITE(MESSAGE,'(A,I0)') 'ERROR: YMIN > YMAX on MESH ', NM
645 CALL SHUTDOWN(MESSAGE) ; RETURN
646 ENDIF
647 IF (XB6-XB5<TWO_EPSILON_EB) THEN
648 WRITE(MESSAGE,'(A,I0)') 'ERROR: ZMIN > ZMAX on MESH ', NM
649 CALL SHUTDOWN(MESSAGE) ; RETURN
650 ENDIF
651 IF (EVACUATION .AND. ABS(XB5 - XB6) <= SPACING(XB(6))) THEN
652 WRITE(MESSAGE,'(A,I0)') 'ERROR: ZMIN = ZMAX on evacuation MESH ', NM

```

```

653 CALL SHUTDOWN(MESSAGE) ; RETURN
654 ENDIF
655
656 M%XS = XB1
657 M%XF = XB2
658 M%YS = XB3
659 M%YF = XB4
660 M%ZS = XB5
661 M%ZF = XB6
662 IF (.NOT.EVACUATION ) THEN
663 XS_MIN = MIN(XS_MIN,M%XS)
664 XF_MAX = MAX(XF_MAX,M%XF)
665 YS_MIN = MIN(YS_MIN,M%YS)
666 YF_MAX = MAX(YF_MAX,M%YF)
667 ZS_MIN = MIN(ZS_MIN,M%ZS)
668 ZF_MAX = MAX(ZF_MAX,M%ZF)
669 ENDIF
670 M%D XI = (M%XF-M%XS)/REAL(M%JBAR,EB)
671 M%D ETA = (M%YF-M%YS)/REAL(M%JBAR,EB)
672 M%D ZETA = (M%ZF-M%ZS)/REAL(M%KBAR,EB)
673 M%RD XI = 1._EB/M%D XI
674 M%RD ETA = 1._EB/M%D ETA
675 M%RD ZETA = 1._EB/M%D ZETA
676 M%JBM1 = M%JBAR-1
677 M%JBM1 = M%JBAR-1
678 M%KBM1 = M%KBAR-1
679 M%JBP1 = M%JBAR+1
680 M%JBP1 = M%JBAR+1
681 M%KBP1 = M%KBAR+1
682
683 IF (TWO) THEN
684 M%CELL_SIZE = SQRT(M%D XI*M%D ZETA)
685 ELSE
686 M%CELL_SIZE = (M%D XI*M%D ETA*M%D ZETA)**CNH
687 ENDIF
688
689 IF (.NOT.EVACUATION_ONLY(NM)) CHARACTERISTIC_CELL_SIZE = MIN( CHARACTERISTIC_CELL_SIZE , M%CELL_SIZE )
690
691 ENDDO I.MULT_LOOP
692 ENDDO J.MULT_LOOP
693 ENDDO K.MULT_LOOP
694
695 ENDDO MESH_LOOP
696
697 NM_EVAC = NM
698
699 ! Check for bad mesh ordering if MPLPROCESS used
700
701 DO NM=1,NMESHERS
702 IF (NM==1) CYCLE
703 IF (EVACUATION_ONLY(NM)) CYCLE
704 IF (PROCESS(NM) < PROCESS(NM-1)) THEN
705 WRITE(MESSAGE,'(A,I0,A,I0,A)') 'ERROR: MPLPROCESS for MESH ',NM,' < MPLPROCESS for MESH ',NM-1,&
706 '. Reorder MESH lines.'
707 CALL SHUTDOWN(MESSAGE) ; RETURN
708 ENDIF
709 ENDDO
710 DO NM=1,NMESHERS
711 IF (NM==1 .OR. .NOT.EVACUATION_ONLY(NM)) CYCLE
712 IF (.NOT.EVACUATION_SKIP(NM)) CYCLE
713 IF (PROCESS(NM) < PROCESS(NM-1)) THEN
714 WRITE(MESSAGE,'(A,I0,A,I0,A)') 'ERROR: MPLPROCESS for evacuation MESH ',NM,' < MPLPROCESS for MESH ',NM-1,&
715 '. Reorder MESH lines.'
716 CALL SHUTDOWN(MESSAGE) ; RETURN
717 ENDIF
718 ENDDO
719
720 !Sesa-added as in 6.2.0
721 ! Allocation for mean forcing (required here, instead of init, because of hole feature)
722
723 IF (ANY(MEAN_FORCING)) THEN
724 DO NM=1,NMESHERS
725 M=>MESHERS(NM)
726 ALLOCATE(M%MEAN_FORCING_CELL(0:M%JBP1,0:M%JBP1,0:M%KBP1),STAT=IZERO)
727 CALL ChkMemErr('INIT','MEAN_FORCING_CELL',IZERO)
728 M%MEAN_FORCING_CELL=.TRUE.
729 ENDDO
730 ENDIF
731
732 ! Min and Max values of temperature
733
734 TMPMIN = MAX(1._EB , MIN(TMPA,TMPM)-10._EB)
735 IF (LAPSE_RATE < 0._EB) TMPMIN = MIN(TMPMIN,TMPA+LAPSE_RATE*ZF_MAX)
736 TMPMAX = 3000._EB
737
738 REWIND(LU_INPUT) ; INPUT_FILE_LINE_NUMBER = 0
739 !Sesa-addition end
740

```

```

741
742 ! Define the additional evacuation door flow meshes
743
744 !Time: Mesh counter NMEVAC is now fire meshes plus main evac meshes
745 IF (.NOT. NOEVACUATION) CALL DEFINE_EVACUATION_MESHES(NMEVAC)
746
747 ! Determine mesh neighbors
748
749 ALLOCATE(NEIGHBOR_LIST(10000))
750 DO NM=1,NMESHERS
751 M => MESHES(NM)
752 M%N_NEIGHBORING_MESHES = 0
753 NEIGHBOR_LIST = 0
754 DO NM2=1,NMESHERS
755 IF (NM/=NM2 .AND. EVACUATION_ONLY(NM2)) CYCLE
756 M2 => MESHES(NM2)
757 IF ((M2%XS>M%XF+NaNOMETER .OR. M2%XF<M%XS-NANOMETER .OR. &
758 M2%YS>M%YF+NaNOMETER .OR. M2%YF<M%YS-NANOMETER .OR. &
759 M2%ZS>M%ZF+NaNOMETER .OR. M2%ZF<M%ZS-NANOMETER) .AND. &
760 PERIODIC_MESH_NAMES(NM,1)/=MESH_NAME(NM2) .AND. &
761 PERIODIC_MESH_NAMES(NM,2)/=MESH_NAME(NM2) .AND. &
762 PERIODIC_MESH_NAMES(NM,3)/=MESH_NAME(NM2)) CYCLE
763 M%N_NEIGHBORING_MESHES = M%N_NEIGHBORING_MESHES + 1
764 NEIGHBOR_LIST(M%N_NEIGHBORING_MESHES) = NM2
765 ENDDO
766 ALLOCATE(M%N_NEIGHBORING_MESH(M%N_NEIGHBORING_MESHES))
767 DO I=1,M%N_NEIGHBORING_MESHES
768 M%N_NEIGHBORING_MESH(I) = NEIGHBOR_LIST(I)
769 ENDDO
770 ENDDO
771 DEALLOCATE(NEIGHBOR_LIST)
772
773 REWIND(LU_INPUT) ; INPUT_FILE_LINE_NUMBER = 0
774
775 CONTAINS
776
777 SUBROUTINE DEFINE_EVACUATION_MESHES(NM)
778 IMPLICIT NONE
779 ! Passed variables
780 INTEGER, INTENT(INOUT) :: NM
781 ! Local variables
782 INTEGER :: N, N_END, I, J, NN, JMAX, NM_OLD, LMAIN_EVAC_MESH
783 REAL(EB) :: Z_MID
784
785 N = 0
786 DO I = 1, NM
787 IF (EVACUATION_SKIP(I) .AND. EVACUATION_ONLY(I)) THEN
788 N = N + 1 ! Main evacuation mesh index for EMESH.EXIT(N) array
789 EMESH_NM(N) = I
790 END IF
791 END DO
792
793 NM_OLD = NM
794 LOOP_EMESHES: DO N = 1, NEVAC_MESHES
795 ! Additional meshes for the main evacuation meshes. These will be
796 ! at different z level than the corresponding main evacuation mesh.
797
798 LMAIN_EVAC_MESH = NM_OLD - NEVAC_MESHES + N
799
800 ! Set MESH defaults
801 RGB = MESHES(LMAIN_EVAC_MESH)%RGB
802 COLOR = 'null'
803 ID = TRIM(TRIM('Emesh-' // MESH_NAME(LMAIN_EVAC_MESH)))
804 MPL_PROCESS = -1
805 LEVEL = 0
806 EVACUATION = .TRUE.
807 EVAC_HUMANS = .FALSE.
808
809 ! Increase the MESH counter by 1
810
811 NM = NM + 1
812
813 ! Fill in MESH related variables
814
815 M => MESHES(NM)
816 M%MESH_LEVEL = LEVEL
817 M%JBAR = MESHES(LMAIN_EVAC_MESH)%JBAR
818 M%IBAR = MESHES(LMAIN_EVAC_MESH)%IBAR
819 M%KBAR = MESHES(LMAIN_EVAC_MESH)%KBAR
820 M%IBAR_MAX = MAX(IBAR_MAX, M%IBAR)
821 M%JBAR_MAX = MAX(JBAR_MAX, M%JBAR)
822 M%KBAR_MAX = MAX(KBAR_MAX, M%KBAR)
823 EVACUATION_ONLY(NM) = .TRUE.
824 EVACUATION_SKIP(NM) = .FALSE.
825 EVACUATION_Z_OFFSET(NM) = EVAC_Z_OFFSET ! Not used, this line is not needed
826 M%N_EXTERNAL_WALL_CELLS = 2*M%IBAR*M%KBAR+2*M%JBAR*M%KBAR
827 IF (EVACUATION .AND. M%KBAR/=1) THEN

```

Source Code files for edited portions of FDS

```

829 WRITE(MESSAGE,'(A)') 'ERROR: IJK(3) must be 1 for all evacuation grids'
830 CALL SHUTDOWN(MESSAGE) ; RETURN
831 ENDIF
832
833 ! Associate the MESH with the PROCESS
834
835 IF (MYID==CURRENT_MPL_PROCESS) THEN
836 LOWER_MESH_INDEX = MIN(LOWER_MESH_INDEX,NM)
837 UPPER_MESH_INDEX = MAX(UPPER_MESH_INDEX,NM)
838 ENDIF
839
840 PROCESS(NM) = CURRENT_MPL_PROCESS
841 IF (MYID==0 .AND. VERBOSE) WRITE(LU_ERR,'(A,I0,A,I0)') ' Mesh ',NM,' is assigned to MPI Process ',PROCESS(NM)
842 IF (EVACUATION_ONLY(NM) .AND. (N_MPL_PROCESSES>1)) EVAC_PROCESS = N_MPL_PROCESSES-1
843
844 ! Mesh boundary colors
845
846 IF (ANY(RGB<0) .AND. COLOR=='null') COLOR = 'BLACK'
847 IF (COLOR /= 'null') CALL COLOR2RGB(RGB,COLOR)
848 ALLOCATE(M%RGB(3))
849 M%RGB = RGB
850
851 ! Mesh Geometry and Name
852
853 WRITE(MESHNAME(NM),'(A,I7.7)') 'MESH_',NM
854 IF (ID/='null') MESHNAME(NM) = ID
855
856 Z_MID = 0.5_EB*(MESHES(LMAIN_EVAC_MESH)%ZS + MESHES(LMAIN_EVAC_MESH)%ZF)
857 Z_MID = Z_MID - EVACUATION_Z_OFFSET(LMAIN_EVAC_MESH) + HUMAN_SMOKE_HEIGHT
858 M%XS = MESHES(LMAIN_EVAC_MESH)%XS
859 M%XF = MESHES(LMAIN_EVAC_MESH)%XF
860 M%YS = MESHES(LMAIN_EVAC_MESH)%YS
861 M%YF = MESHES(LMAIN_EVAC_MESH)%YF
862 M%ZS = Z_MID - EVAC_DELTA_SEE
863 M%ZF = Z_MID + EVAC_DELTA_SEE
864 M%DXI = MESHES(LMAIN_EVAC_MESH)%DXI
865 M%DETA = MESHES(LMAIN_EVAC_MESH)%DETA
866 M%DZETA = (M%ZF-M%ZS)/REAL(M%KBAR,EB)
867 M%RD XI = MESHES(LMAIN_EVAC_MESH)%RDXI
868 M%RD ETA = MESHES(LMAIN_EVAC_MESH)%RDETA
869 M%RD ZETA = 1._EB/M%DZETA
870 M%bBM1 = M%bBAR-1
871 M%jBM1 = M%jBAR-1
872 M%kBM1 = M%kBAR-1
873 M%jBP1 = M%jBAR+1
874 M%kBP1 = M%kBAR+1
875 M%kBP1 = M%kBAR+1
876 ! WRITE (LU_ERR,FMT='(A,I0,3A)') ' EVAC: Mesh number ', NM, ' name ', TRIM(ID), ' defined for evacuation '
877
878 END DO LOOP_EMESHES
879
880 N_END = N_EXITS - N_CO_EXITS + N_DOORS
881 LOOP_EXITS: DO N = 1, N_END
882 I = EMESH_EXITS(N)%EMESH ! The main evacuation mesh index (for EMESH_EXITS(I) array)
883 IF (.NOT.EMESH_EXITS(N)%DEFINE_MESH) CYCLE LOOP_EXITS
884
885 EMESH_EXITS(N)%MAIN_MESH = EMESH_NM(EMESH_EXITS(N)%EMESH) ! The 1,...,NMESHES index
886 ! Only main evacuation meshes in FDS6
887 EMESH_EXITS(N)%MESH = EMESH_EXITS(N)%MAIN_MESH ! The mesh index (all meshes included)
888
889 ! Set MESH defaults
890
891 IJK(1) = EMESH_IJK(1,1)
892 IJK(2) = EMESH_IJK(2,1)
893 IJK(3) = EMESH_IJK(3,1)
894
895 ALLOCATE(EMESH_EXITS(N)%U_EVAC(0: IJK(1)+1,0: IJK(2)+1),STAT=IZERO)
896 CALL ChkMemErr('READ','EMESH_EXITS(N)%U_EVAC',IZERO)
897 ALLOCATE(EMESH_EXITS(N)%V_EVAC(0: IJK(1)+1,0: IJK(2)+1),STAT=IZERO)
898 CALL ChkMemErr('READ','EMESH_EXITS(N)%V_EVAC',IZERO)
899
900 CYCLE LOOP_EXITS
901 ENDDO LOOP_EXITS
902
903 NN = 0
904 JMAX = 0
905 DO I = 1, NM
906 EV_IF: IF (EVACUATION_SKIP(I) .AND. EVACUATION_ONLY(I)) THEN
907 J = 0 ! Index of the flow field (for a main evacuation mesh)
908 NN = NN + 1 ! Main evacuation mesh index
909 ! NN = EMESH_INDEX(NM)
910 EMESH_NFIELDS(NN) = 0 ! How many fields for this main evacuation mesh
911 LOOP_EXITS_0: DO N = 1, N_END
912 IF (.NOT.EMESH_EXITS(N)%DEFINE_MESH) CYCLE LOOP_EXITS_0
913 IF (.NOT.EMESH_EXITS(N)%EMESH == NN) CYCLE LOOP_EXITS_0
914 J = J + 1
915 EMESH_EXITS(N)%L_DOORS_EMESH = J
916 EMESH_NFIELDS(NN) = J

```

Source Code files for edited portions of FDS

```

917 END DO LOOP_EXITS_0
918 IF (EMESH_NFIELDS(NN)==0) THEN
919 WRITE(MESSAGE,'(A,10,3A)') 'ERROR: EVAC: Emesh ',NN,' ',TRIM(EMESH_ID(NN)), ' needs at least one DOOR/EXIT.'
920 CALL SHUTDOWN(MESSAGE) ; RETURN
921 ELSE
922 WRITE(LU_ERR,FMT='(A,10,3A,10,A)') ' EVAC: Emesh ',NN,' ',TRIM(EMESH_ID(NN)), ' has ',&
923 EMESH_NFIELDS(NN), ' door flow fields'
924 ENDIF
925 ENDIF EV_IF
926 ENDDO
927
928 ! Next line should be executed only once during a FDS+Evac run
929 JMAX = MAXVAL(EMESH_NFIELDS,1)
930 EVAC_TIME_ITERATIONS = EVAC_TIME_ITERATIONS*jMAX
931
932 LOOP_STAIRS: DO N = 1, N_STRS
933
934 ! Evacuation meshes for the stairs.
935
936 ! Set MESH defaults
937
938 RGB = EMESH_STAIRS(N)%RGB
939 COLOR = 'null'
940 ID = TRIM('Emesh_' // TRIM(EMESH_STAIRS(N)%ID))
941 MPL_PROCESS = -1
942 LEVEL = 0
943 EVACUATION = .TRUE.
944 EVAC_HUMANS = .TRUE.
945 EVAC_Z_OFFSET = EMESH_STAIRS(N)%EVAC_Z_OFFSET
946
947 ! Increase the MESH counter by 1
948
949 NM = NM + 1
950 EMESH_STAIRS(N)%MESH = NM
951
952 ! Fill in MESH related variables
953
954 M => MESHES(NM)
955 M%MESH_LEVEL = LEVEL
956 M%IBAR = EMESH_STAIRS(N)%IBAR
957 M%jBAR = EMESH_STAIRS(N)%jBAR
958 M%kBAR = EMESH_STAIRS(N)%kBAR
959 IBAR_MAX = MAX(IBAR_MAX,M%IBAR)
960 jBAR_MAX = MAX(jBAR_MAX,M%jBAR)
961 kBAR_MAX = MAX(kBAR_MAX,M%kBAR)
962 EVACUATION_ONLY(NM) = .TRUE.
963 EVACUATION_SKIP(NM) = .TRUE.
964 EVACUATION_Z_OFFSET(NM) = EVAC_Z_OFFSET
965 M%N_EXTERNAL_WALL_CELLS = 2*M%IBAR*M%kBAR+2*M%jBAR*M%kBAR
966 IF (EVACUATION .AND. M%kBAR/=1) THEN
967 WRITE(MESSAGE,'(A)') 'ERROR: IJK(3) must be 1 for all evacuation grids'
968 CALL SHUTDOWN(MESSAGE) ; RETURN
969 ENDIF
970
971 ! Associate the MESH with the PROCESS
972
973 IF (MYID==CURRENT_MPL_PROCESS) THEN
974 LOWER_MESH_INDEX = MIN(LOWER_MESH_INDEX,NM)
975 UPPER_MESH_INDEX = MAX(UPPER_MESH_INDEX,NM)
976 ENDIF
977
978 PROCESS(NM) = CURRENT_MPL_PROCESS
979 IF (MYID==0 .AND. VERBOSE) WRITE(LU_ERR,'(A,10,A,10)') ' Mesh ',NM,' is assigned to MPI Process ',PROCESS(NM)
980 IF (EVACUATION_ONLY(NM) .AND. (N_MPL_PROCESSES>1)) EVAC_PROCESS = N_MPL_PROCESSES-1
981
982 ! Mesh boundary colors
983
984 ALLOCATE(M%RGB(3))
985 M%RGB = EMESH_STAIRS(N)%RGB
986
987 ! Mesh Geometry and Name
988
989 WRITE(MESH_NAME(NM),'(A,17,7)') 'MESH_',NM
990 IF (ID/'null') MESH_NAME(NM) = ID
991
992 M%X = EMESH_STAIRS(N)%XB(1)
993 M%XF = EMESH_STAIRS(N)%XB(2)
994 M%Y = EMESH_STAIRS(N)%XB(3)
995 M%YF = EMESH_STAIRS(N)%XB(4)
996 M%Z = EMESH_STAIRS(N)%XB(5)
997 M%ZF = EMESH_STAIRS(N)%XB(6)
998 M%DXI = (M%XF-M%X)/REAL(M%IBAR,EB)
999 M%DETA = (M%YF-M%Y)/REAL(M%jBAR,EB)
1000 M%DZETA = (M%ZF-M%Z)/REAL(M%kBAR,EB)
1001 M%RD XI = 1.-EB/M%DXI
1002 M%RDETA = 1.-EB/M%DETA
1003 M%RDZETA = 1.-EB/M%DZETA
1004 M%IBM1 = M%IBAR-1

```



```

1005 M%JBM1 = M%JBAR-1
1006 M%KBM1 = M%KBAR-1
1007 M%JBP1 = M%JBAR+1
1008 M%KBP1 = M%KBAR+1
1009 M%KBP1 = M%KBAR+1
1010 WRITE (LU_ERR,FMT='(A,I0,3A)') ' EVAC: Mesh number ', NM, ' name ', TRIM(ID), ' defined for evacuation'
1011
1012 ENDDO LOOP_STAIRS
1013
1014 IF (ALL(EVACUATION_ONLY)) THEN
1015 DO N = 1, NMESHES
1016 M => MESHES(NM)
1017 XS_MIN = MIN(XS_MIN,M%XS)
1018 XF_MAX = MAX(XF_MAX,M%XF)
1019 YS_MIN = MIN(YS_MIN,M%YS)
1020 YF_MAX = MAX(YF_MAX,M%YF)
1021 ZS_MIN = MIN(ZS_MIN,M%ZS)
1022 ZF_MAX = MAX(ZF_MAX,M%ZF)
1023 ENDDO
1024 ENDIF
1025
1026 RETURN
1027 END SUBROUTINE DEFINE_EVACUATION_MESHES
1028
1029 END SUBROUTINE READ_MESH
1030
1031
1032 SUBROUTINE READ_TRAN
1033 USE MATH_FUNCTIONS, ONLY : GAUSSJ
1034
1035 ! Compute the polynomial transform function for the vertical coordinate
1036
1037 REAL(EB), ALLOCATABLE, DIMENSION(:,:) :: A,XX
1038 INTEGER, ALLOCATABLE, DIMENSION(:,:) :: ND
1039 REAL(EB) :: PC,CC,COEF,XI,ETA,ZETA
1040 INTEGER IEXP,IC,IDERIV,N,K,IERROR,IOS,I,MESHNUMBER,NIPX,NIPY,NIPZ,NIPXS,NIPYS,NIPZS,NIPXF,NIPYF,NIPZF,NM
1041 LOGICAL :: PROCESS_TRANS
1042 TYPE (MESH_TYPE), POINTER :: M=>NULL()
1043 TYPE (TRAN_TYPE), POINTER :: T=>NULL()
1044 NAMELIST /TRNX/ CC,FYI,IDERIV,MESHNUMBER,PC
1045 NAMELIST /TRNY/ CC,FYI,IDERIV,MESHNUMBER,PC
1046 NAMELIST /TRNZ/ CC,FYI,IDERIV,MESHNUMBER,PC
1047
1048 ! Scan the input file, counting the number of NAMELIST entries
1049
1050 ALLOCATE(TRANS(NMESHES))
1051
1052 MESHLOOP: DO NM=1,NMESHES
1053
1054 M => MESHES(NM)
1055
1056 ! Only read and process the TRNX, TRNY and TRNZ lines if the current MPI
1057 ! process (MYID) controls mesh NM or one of its neighbors.
1058
1059 PROCESS_TRANS = .FALSE.
1060 DO N=1,M%N_NEIGHBORING_MESHES
1061 IF (MYID==PROCESS(M%N_NEIGHBORING_MESH(N))) PROCESS_TRANS = .TRUE.
1062 ENDDO
1063
1064 IF (PROCESS_ALL_MESHES) PROCESS_TRANS = .TRUE.
1065
1066 ! A fast fix for fire+evacuation calculation with MPI and neighboring_mesh array problem
1067 ! Evacuation meshes need fire mesh obst information => evacuation process processes all fire meshes
1068 IF (MYID==EVAC_PROCESS .AND. .NOT.EVACUATION_ONLY(NM)) PROCESS_TRANS = .TRUE.
1069
1070 IF (.NOT.PROCESS_TRANS) CYCLE MESHLOOP
1071
1072 T => TRANS(NM)
1073
1074 DO N=1,3
1075 T%NOC(N) = 0
1076 TRNLOOP: DO
1077 IF (EVACUATION_ONLY(NM)) EXIT TRNLOOP
1078 SELECT CASE (N)
1079 CASE(1)
1080 CALL CHECKREAD('TRNX',LU_INPUT,IOS)
1081 IF (IOS==1) EXIT TRNLOOP
1082 MESHNUMBER = 1
1083 READ(LU_INPUT,NML=TRNX,END=17,ERR=18,IOSTAT=IOS)
1084 IF (MESHNUMBER>0 .AND. MESHNUMBER/=NM) CYCLE TRNLOOP
1085 CASE(2)
1086 CALL CHECKREAD('TRNY',LU_INPUT,IOS)
1087 IF (IOS==1) EXIT TRNLOOP
1088 MESHNUMBER = 1
1089 READ(LU_INPUT,NML=TRNY,END=17,ERR=18,IOSTAT=IOS)
1090 IF (MESHNUMBER>0 .AND. MESHNUMBER/=NM) CYCLE TRNLOOP
1091 CASE(3)
1092 CALL CHECKREAD('TRNZ',LU_INPUT,IOS)

```

```

1093 IF (IOS==1) EXIT TRNLOOP
1094 MESHNUMBER = 1
1095 READ(LU.INPUT,NML=TRNZ,END=17,ERR=18,IOSTAT=IOS)
1096 IF (MESHNUMBER>0 .AND. MESHNUMBER/=NM) CYCLE TRNLOOP
1097 END SELECT
1098 T%NOC(N) = T%NOC(N) + 1
1099 18 IF (IOS>0) THEN ; CALL SHUTDOWN('ERROR: Problem with TRN* line') ; RETURN ; ENDIF
1100 ENDDO TRNLOOP
1101 17 REWIND(LU.INPUT) ; INPUT_FILE.LINE_NUMBER = 0
1102 ENDDO
1103
1104 T%NOCMAX = MAX(T%NOC(1),T%NOC(2),T%NOC(3))
1105 ALLOCATE(A(T%NOCMAX+1,T%NOCMAX+1))
1106 ALLOCATE(XX(T%NOCMAX+1,3))
1107 ALLOCATE(ND(T%NOCMAX+1,3))
1108 ALLOCATE(T%CI(0:T%NOCMAX+1,3))
1109 T%CI = 0._EB
1110 T%CI(1,1:3) = 1._EB
1111 ALLOCATE(T%C2(0:T%NOCMAX+1,3))
1112 ALLOCATE(T%C3(0:T%NOCMAX+1,3))
1113 ALLOCATE(T%CCSTORE(T%NOCMAX,3))
1114 ALLOCATE(T%PCSTORE(T%NOCMAX,3))
1115 ALLOCATE(T%IDERIVSTORE(T%NOCMAX,3))
1116
1117 T%dTRAN = 0
1118
1119 DO IC=1,3
1120 NLOOP: DO N=1,T%NOC(IC)
1121 IDERIV = -1
1122 IF (IC==1) THEN
1123 LOOP1: DO
1124 CALL CHECKREAD('TRNX',LU.INPUT,IOS)
1125 IF (IOS==1) EXIT NLOOP
1126 MESHNUMBER = 1
1127 READ(LU.INPUT,TRNX,END=1,ERR=2)
1128 IF (MESHNUMBER==0 .OR. MESHNUMBER/=NM) EXIT LOOP1
1129 ENDDO LOOP1
1130 ENDIF
1131 IF (IC==2) THEN
1132 LOOP2: DO
1133 CALL CHECKREAD('TRNY',LU.INPUT,IOS)
1134 IF (IOS==1) EXIT NLOOP
1135 MESHNUMBER = 1
1136 READ(LU.INPUT,TRNY,END=1,ERR=2)
1137 IF (MESHNUMBER==0 .OR. MESHNUMBER/=NM) EXIT LOOP2
1138 ENDDO LOOP2
1139 ENDIF
1140 IF (IC==3) THEN
1141 LOOP3: DO
1142 CALL CHECKREAD('TRNZ',LU.INPUT,IOS)
1143 IF (IOS==1) EXIT NLOOP
1144 MESHNUMBER = 1
1145 READ(LU.INPUT,TRNZ,END=1,ERR=2)
1146 IF (MESHNUMBER==0 .OR. MESHNUMBER/=NM) EXIT LOOP3
1147 ENDDO LOOP3
1148 ENDIF
1149 T%CCSTORE(N,IC) = CC
1150 T%PCSTORE(N,IC) = PC
1151 T%IDERIVSTORE(N,IC) = IDERIV
1152 IF (IDERIV>=0) T%dTRAN(IC) = 1
1153 IF (IDERIV<0) T%dTRAN(IC) = 2
1154 2 CONTINUE
1155 ENDDO NLOOP
1156 1 REWIND(LU.INPUT) ; INPUT_FILE.LINE_NUMBER = 0
1157 ENDDO
1158
1159 ICLOOP: DO IC=1,3
1160
1161 SELECT CASE (T%dTRAN(IC))
1162
1163 CASE (1) ! polynomial transformation
1164 ND(1,IC) = 0
1165 SELECT CASE(IC)
1166 CASE(1)
1167 XX(1,IC) = M%XF-M%XS
1168 T%CI(1,IC) = M%XF-M%XS
1169 CASE(2)
1170 XX(1,IC) = M%YF-M%YS
1171 T%CI(1,IC) = M%YF-M%YS
1172 CASE(3)
1173 XX(1,IC) = M%ZF-M%ZS
1174 T%CI(1,IC) = M%ZF-M%ZS
1175 END SELECT
1176
1177 NNLOOP: DO N=2,T%NOC(IC)+1
1178 IDERIV = T%IDERIVSTORE(N-1,IC)
1179 IF (IC==1) CC = T%CCSTORE(N-1,IC)-M%XS
1180 IF (IC==2) CC = T%CCSTORE(N-1,IC)-M%YS

```

```

1181 IF (IC==3) CC = T%CCSTORE(N-1,IC)-M%ZS
1182 IF (IC==1 .AND. IDERIV==0) PC = T%PCSTORE(N-1,IC)-M%XS
1183 IF (IC==2 .AND. IDERIV==0) PC = T%PCSTORE(N-1,IC)-M%YS
1184 IF (IC==3 .AND. IDERIV==0) PC = T%PCSTORE(N-1,IC)-M%ZS
1185 IF (IC==1 .AND. IDERIV>0) PC = T%PCSTORE(N-1,IC)
1186 IF (IC==2 .AND. IDERIV>0) PC = T%PCSTORE(N-1,IC)
1187 IF (IC==3 .AND. IDERIV>0) PC = T%PCSTORE(N-1,IC)
1188 ND(N,IC) = IDERIV
1189 XX(N,IC) = CC
1190 T%C1(N,IC) = PC
1191 ENDDO NNLOOP
1192
1193 DO K=1,T%NOC(IC)+1
1194 DO N=1,T%NOC(IC)+1
1195 COEF = IFAC(K,ND(N,IC))
1196 IEXP = K-ND(N,IC)
1197 IF (IEXP<0) A(N,K) = 0._EB
1198 IF (IEXP==0) A(N,K) = COEF
1199 IF (IEXP>0) A(N,K) = COEF**XX(N,IC)**IEXP
1200 ENDDO
1201 ENDDO
1202
1203 IERROR = 0
1204 CALL GAUSSJ(A,T%NOC(IC)+1,T%NOCMAX+1,T%C1(1:T%NOCMAX+1,IC),1,1,IERROR)
1205 IF (IERROR/=0) THEN ; CALL SHUTDOWN('ERROR: Problem with grid transformation') ; RETURN ; ENDIF
1206
1207 CASE (2) ! linear transformation
1208
1209 T%C1(0,IC) = 0._EB
1210 T%C2(0,IC) = 0._EB
1211 DO N=1,T%NOC(IC)
1212 IF (IC==1) CC = T%CCSTORE(N,IC)-M%XS
1213 IF (IC==2) CC = T%CCSTORE(N,IC)-M%YS
1214 IF (IC==3) CC = T%CCSTORE(N,IC)-M%ZS
1215 IF (IC==1) PC = T%PCSTORE(N,IC)-M%XS
1216 IF (IC==2) PC = T%PCSTORE(N,IC)-M%YS
1217 IF (IC==3) PC = T%PCSTORE(N,IC)-M%ZS
1218 T%C1(N,IC) = CC
1219 T%C2(N,IC) = PC
1220 ENDDO
1221
1222 SELECT CASE(IC)
1223 CASE(1)
1224 T%C1(T%NOC(1)+1,1) = M%XF-M%XS
1225 T%C2(T%NOC(1)+1,1) = M%YF-M%YS
1226 CASE(2)
1227 T%C1(T%NOC(2)+1,2) = M%YF-M%YS
1228 T%C2(T%NOC(2)+1,2) = M%ZF-M%ZS
1229 CASE(3)
1230 T%C1(T%NOC(3)+1,3) = M%ZF-M%ZS
1231 T%C2(T%NOC(3)+1,3) = M%XF-M%XS
1232 END SELECT
1233
1234 DO N=1,T%NOC(IC)+1
1235 IF (T%C1(N,IC)-T%C1(N-1,IC)<TWO.EPSILON.EB) THEN
1236 CALL SHUTDOWN('ERROR: Do not specify endpoints in linear grid transformation')
1237 RETURN
1238 ENDIF
1239 T%C3(N,IC) = (T%C2(N,IC)-T%C2(N-1,IC))/(T%C1(N,IC)-T%C1(N-1,IC))
1240 ENDDO
1241 END SELECT
1242 ENDDO ICLOOP
1243
1244 DEALLOCATE(A)
1245 DEALLOCATE(XX)
1246 DEALLOCATE(ND)
1247
1248 ! Set up grid stretching arrays
1249
1250 ALLOCATE(M%R(0:M%IBAR),STAT=IZERO)
1251 CALL ChkMemErr('READ','R',IZERO)
1252 ALLOCATE(M%RC(0:M%IBAR+1),STAT=IZERO)
1253 CALL ChkMemErr('READ','RC',IZERO)
1254 M%RC = 1._EB
1255 ALLOCATE(M%RRN(0:M%IBP1),STAT=IZERO)
1256 CALL ChkMemErr('READ','RRN',IZERO)
1257 M%RRN = 1._EB
1258 ALLOCATE(M%X(0:M%IBAR),STAT=IZERO)
1259 CALL ChkMemErr('READ','X',IZERO)
1260 ALLOCATE(M%XC(0:M%IBP1),STAT=IZERO)
1261 CALL ChkMemErr('READ','XC',IZERO)
1262 ALLOCATE(M%HX(0:M%IBP1),STAT=IZERO)
1263 CALL ChkMemErr('READ','HX',IZERO)
1264 ALLOCATE(M%DX(0:M%IBP1),STAT=IZERO)
1265 CALL ChkMemErr('READ','DX',IZERO)
1266 ALLOCATE(M%RD(0:M%IBP1),STAT=IZERO)
1267 CALL ChkMemErr('READ','RD',IZERO)
1268 ALLOCATE(M%DXN(0:M%IBAR),STAT=IZERO)

```

```

1269 CALL ChkMemErr( 'READ', 'DXN', IZERO)
1270 ALLOCATE(M%RDYN(0:M%IBAR), STAT=IZERO)
1271 CALL ChkMemErr( 'READ', 'RDXN', IZERO)
1272 ALLOCATE(M%Y(0:M%IBAR), STAT=IZERO)
1273 CALL ChkMemErr( 'READ', 'Y', IZERO)
1274 ALLOCATE(M%YC(0:M%IBP1), STAT=IZERO)
1275 CALL ChkMemErr( 'READ', 'YC', IZERO)
1276 ALLOCATE(M%HY(0:M%IBP1), STAT=IZERO)
1277 CALL ChkMemErr( 'READ', 'HY', IZERO)
1278 ALLOCATE(M%DY(0:M%IBP1), STAT=IZERO)
1279 CALL ChkMemErr( 'READ', 'DY', IZERO)
1280 ALLOCATE(M%RDY(0:M%IBP1), STAT=IZERO)
1281 CALL ChkMemErr( 'READ', 'RDY', IZERO)
1282 ALLOCATE(M%DYN(0:M%IBAR), STAT=IZERO)
1283 CALL ChkMemErr( 'READ', 'DYN', IZERO)
1284 ALLOCATE(M%RDYN(0:M%IBAR), STAT=IZERO)
1285 CALL ChkMemErr( 'READ', 'RDYN', IZERO)
1286 ALLOCATE(M%Z(0:M%IBAR), STAT=IZERO)
1287 CALL ChkMemErr( 'READ', 'Z', IZERO)
1288 ALLOCATE(M%ZC(0:M%IBP1), STAT=IZERO)
1289 CALL ChkMemErr( 'READ', 'ZC', IZERO)
1290 ALLOCATE(M%HZ(0:M%IBP1), STAT=IZERO)
1291 CALL ChkMemErr( 'READ', 'HZ', IZERO)
1292 ALLOCATE(M%DZ(0:M%IBP1), STAT=IZERO)
1293 CALL ChkMemErr( 'READ', 'DZ', IZERO)
1294 ALLOCATE(M%RDZ(0:M%IBP1), STAT=IZERO)
1295 CALL ChkMemErr( 'READ', 'RDZ', IZERO)
1296 ALLOCATE(M%DZN(0:M%IBAR), STAT=IZERO)
1297 CALL ChkMemErr( 'READ', 'DZN', IZERO)
1298 ALLOCATE(M%RDZN(0:M%IBAR), STAT=IZERO)
1299 CALL ChkMemErr( 'READ', 'RDZN', IZERO)
1300
1301 ! Define X grid stretching terms
1302
1303 M%DXMIN = 1000._EB
1304 DO I=1,M%IBAR
1305 XI = (REAL(I, EB) - .5)*M%DXI
1306 M%H(XI) = GP(XI, 1, NM)
1307 M%DX(I) = M%H(XI)*M%DXI
1308 M%DXMIN = MIN(M%DXMIN, M%DX(I))
1309 IF (M%DX(I) <= 0._EB) THEN
1310 WRITE(MESSAGE, '(A, I0)') 'ERROR: x transformation not monotonic, mesh ', NM
1311 CALL SHUTDOWN(MESSAGE) ; RETURN
1312 ENDIF
1313 M%RD(XI) = 1._EB/M%DX(I)
1314 ENDDO
1315
1316 M%H(0) = M%H(1)
1317 M%H(M%IBP1) = M%H(M%IBAR)
1318 M%DX(0) = M%DX(1)
1319 M%DX(M%IBP1) = M%DX(M%IBAR)
1320 M%RD(0) = 1._EB/M%DX(1)
1321 M%RD(M%IBP1) = 1._EB/M%DX(M%IBAR)
1322
1323 DO I=0,M%IBAR
1324 XI = I*M%DXI
1325 M%X(I) = M%XS + G(XI, 1, NM)
1326 IF (CYLINDRICAL) THEN
1327 M%R(I) = M%G(XI)
1328 ELSE
1329 M%R(I) = 1._EB
1330 ENDIF
1331 M%DXN(I) = 0.5._EB*(M%DX(I)+M%DX(I+1))
1332 M%RDYN(I) = 1._EB/M%DXN(I)
1333 ENDDO
1334 M%X(0) = M%XS
1335 M%X(M%IBAR) = M%XF
1336
1337 DO I=1,M%IBAR
1338 M%XC(I) = 0.5._EB*(M%X(I)+M%X(I-1))
1339 ENDDO
1340 M%XC(0) = M%XS - 0.5._EB*M%DX(0)
1341 M%XC(M%IBP1) = M%XF + 0.5._EB*M%DX(M%IBP1)
1342
1343 IF (CYLINDRICAL) THEN
1344 DO I=1,M%IBAR
1345 M%RRN(I) = 2._EB/(M%R(I)+M%R(I-1))
1346 M%RC(I) = 0.5._EB*(M%R(I)+M%R(I-1))
1347 ENDDO
1348 M%RRN(0) = M%RRN(1)
1349 M%RRN(M%IBP1) = M%RRN(M%IBAR)
1350 ENDIF
1351
1352 ! Define Y grid stretching terms
1353
1354 M%DYMIN = 1000._EB
1355 DO J=1,M%IBAR
1356 ETA = (REAL(J, EB) - .5)*M%DETA

```

Source Code files for edited portions of FDS

```

1357 M%HY(J) = GP(ETA,2 ,NM)
1358 M%DY(J) = M%HY(J)*M%DETA
1359 M%DYMIN = MIN(M%DYMIN,M%DY(J))
1360 IF (M%HY(J)<=0._EB) THEN
1361 WRITE(MESSAGE, '(A,I0) ' 'ERROR: y transformation not monotonic, mesh ',NM
1362 CALL SHUTDOWN(MESSAGE) ; RETURN
1363 ENDIF
1364 M%RDY(J) = 1._EB/M%DY(J)
1365 ENDDO
1366
1367 M%HY(0) = M%HY(1)
1368 M%HY(M%jBP1) = M%HY(M%jBAR)
1369 M%DY(0) = M%DY(1)
1370 M%DY(M%jBP1) = M%DY(M%jBAR)
1371 M%RDY(0) = 1._EB/M%DY(1)
1372 M%RDY(M%jBP1) = 1._EB/M%DY(M%jBAR)
1373
1374 DO J=0,M%jBAR
1375 ETA = J*M%DETA
1376 M%Y(J) = M%YS + G(ETA,2 ,NM)
1377 M%DYN(J) = 0.5._EB*(M%DY(J)+M%DY(J+1))
1378 M%RDYN(J) = 1._EB/M%DYN(J)
1379 ENDDO
1380
1381 M%Y(0) = M%YS
1382 M%Y(M%jBAR) = M%YF
1383
1384 DO J=1,M%jBAR
1385 M%YC(J) = 0.5._EB*(M%Y(J)+M%Y(J-1))
1386 ENDDO
1387 M%YC(0) = M%YS - 0.5._EB*M%DY(0)
1388 M%YC(M%jBP1) = M%YF + 0.5._EB*M%DY(M%jBP1)
1389
1390 ! Define Z grid stretching terms
1391
1392 M%DZMIN = 1000._EB
1393 DO K=1,M%kBAR
1394 ZETA = (REAL(K,EB) - .5._EB)*M%DZETA
1395 M%HZ(K) = GP(ZETA,3 ,NM)
1396 M%DZ(K) = M%HZ(K)*M%DZETA
1397 M%DZMIN = MIN(M%DZMIN,M%DZ(K))
1398 IF (M%HZ(K)<=0._EB) THEN
1399 WRITE(MESSAGE, '(A,I0) ' 'ERROR: z transformation not monotonic, mesh ',NM
1400 CALL SHUTDOWN(MESSAGE) ; RETURN
1401 ENDIF
1402 M%RDZ(K) = 1._EB/M%DZ(K)
1403 ENDDO
1404
1405 M%HZ(0) = M%HZ(1)
1406 M%HZ(M%kBP1) = M%HZ(M%kBAR)
1407 M%DZ(0) = M%DZ(1)
1408 M%DZ(M%kBP1) = M%DZ(M%kBAR)
1409 M%RDZ(0) = 1._EB/M%DZ(1)
1410 M%RDZ(M%kBP1) = 1._EB/M%DZ(M%kBAR)
1411
1412 DO K=0,M%kBAR
1413 ZETA = K*M%DZETA
1414 M%Z(K) = M%ZS + G(ZETA,3 ,NM)
1415 M%DZN(K) = 0.5._EB*(M%DZ(K)+M%DZ(K+1))
1416 M%RDZN(K) = 1._EB/M%DZN(K)
1417 ENDDO
1418
1419 M%Z(0) = M%ZS
1420 M%Z(M%kBAR) = M%ZF
1421
1422 DO K=1,M%kBAR
1423 M%ZC(K) = 0.5._EB*(M%Z(K)+M%Z(K-1))
1424 ENDDO
1425 M%ZC(0) = M%ZS - 0.5._EB*M%DZ(0)
1426 M%ZC(M%kBP1) = M%ZF + 0.5._EB*M%DZ(M%kBP1)
1427
1428 ! Set up arrays that will return coordinate positions
1429
1430 NIPX = 500*M%jBAR
1431 NIPY = 500*M%jBAR
1432 NIPZ = 500*M%kBAR
1433 NIPXS = NINT(NIPX*M%DX(0)/(M%XF-M%XS))
1434 NIPXF = NINT(NIPX*M%DX(M%jBP1)/(M%XF-M%XS))
1435 NIPYS = NINT(NIPY*M%DY(0)/(M%YF-M%YS))
1436 NIPYF = NINT(NIPY*M%DY(M%jBP1)/(M%YF-M%YS))
1437 NIPZS = NINT(NIPZ*M%DZ(0)/(M%ZF-M%ZS))
1438 NIPZF = NINT(NIPZ*M%DZ(M%kBP1)/(M%ZF-M%ZS))
1439 M%RDXTINT = REAL(NIPX,EB)/(M%XF-M%XS)
1440 M%RDYINT = REAL(NIPY,EB)/(M%YF-M%YS)
1441 M%RDZINT = REAL(NIPZ,EB)/(M%ZF-M%ZS)
1442
1443 ALLOCATE(M%CELLSI(-NIPXS:NIPX+NIPXF),STAT=IZERO)
1444 CALL ChkMemErr('READ', 'CELLSI', IZERO)

```

```

1445 ALLOCATE(M%CELLSJ(-NIPYS:NIPY+NIPYF),STAT=IZERO)
1446 CALL ChkMemErr('READ','CELLSJ',IZERO)
1447 ALLOCATE(M%CELLSK(-NIPZS:NIPZ+NIPZF),STAT=IZERO)
1448 CALL ChkMemErr('READ','CELLSK',IZERO)
1449
1450 DO I=-NIPXS,NIPX+NIPXF
1451 M%CELLSI(I) = GINV(REAL(I,EB)/M%RDXTINT,1,NM)*M%RDXT
1452 M%CELLSI(I) = MAX(M%CELLSI(I),-0.9_EB)
1453 M%CELLSI(I) = MIN(M%CELLSI(I),REAL(M%KBAR)+0.9_EB)
1454 ENDDO
1455 DO J=-NIPYS,NIPY+NIPYF
1456 M%CELLSJ(J) = GINV(REAL(J,EB)/M%RDYINT,2,NM)*M%RDETA
1457 M%CELLSJ(J) = MAX(M%CELLSJ(J),-0.9_EB)
1458 M%CELLSJ(J) = MIN(M%CELLSJ(J),REAL(M%KBAR)+0.9_EB)
1459 ENDDO
1460 DO K=-NIPZS,NIPZ+NIPZF
1461 M%CELLSK(K) = GINV(REAL(K,EB)/M%RDZINT,3,NM)*M%RDZETA
1462 M%CELLSK(K) = MAX(M%CELLSK(K),-0.9_EB)
1463 M%CELLSK(K) = MIN(M%CELLSK(K),REAL(M%KBAR)+0.9_EB)
1464 ENDDO
1465
1466 ENDDO MESHLOOP
1467
1468 CONTAINS
1469
1470 INTEGER FUNCTION IFAC(II,NN)
1471 INTEGER, INTENT(IN) :: II,NN
1472 INTEGER :: III
1473 IFAC = 1
1474 DO III=II-NN+1,II
1475 IFAC = IFAC*III
1476 ENDDO
1477 END FUNCTION IFAC
1478
1479 END SUBROUTINE READ_TRAN
1480
1481
1482 SUBROUTINE READ_TIME(DT)
1483
1484 REAL(EB) :: VEL_CHAR,DT
1485 NAMELIST /TIME/ DT,EVAC_DT.FLOWFIELD,EVAC_DT.STEADY.STATE,FYI,LIMITING_DT.RATIO,LOCK.TIME.STEP,RESTRICT.TIME.STEP
1486 , &
1487 T.BEGIN,T.END,T.END.GEOM,TIME.SHRIK.FACTOR,WALL.INCREMENT,WALL.INCREMENT.HT3D, &
1488 TWIN ! Backward compatibility
1489
1490 DT = -1._EB
1491 EVAC_DT.FLOWFIELD = 0.01_EB
1492 EVAC_DT.STEADY.STATE = 0.05_EB
1493 TIME.SHRIK.FACTOR = 1._EB
1494 T.BEGIN = 0._EB
1495 T.END = 1._EB
1496 T.END.GEOM = T.END
1497 IF (ALL(EVACUATION_ONLY)) DT = EVAC_DT.STEADY.STATE
1498
1499 REWIND(LU.INPUT) ; INPUT_FILE.LINE.NUMBER = 0
1500 READ.TIME.LOOP: DO
1501 CALL CHECKREAD('TIME',LU.INPUT,IOS)
1502 IF (IOS==1) EXIT READ.TIME.LOOP
1503 READ(LU.INPUT,TIME,END=21,ERR=22,IOSTAT=IOS)
1504 22 IF (IOS>0) THEN ; CALL SHUTDOWN('ERROR: Problem with TIME line') ; RETURN ; ENDDIF
1505 ENDDO READ.TIME.LOOP
1506 21 REWIND(LU.INPUT) ; INPUT_FILE.LINE.NUMBER = 0
1507
1508 IF (T.END<=T.BEGIN) SET_UP_ONLY = .TRUE.
1509 T.END = T.BEGIN + (T.END-T.BEGIN)/TIME.SHRIK.FACTOR
1510
1511 ! Compute the starting time step if the user has not specified it.
1512
1513 IF (DT<=0._EB) THEN
1514 VEL_CHAR = 0.2_EB*SQRT(10._EB*(ZF.MAX-ZS.MIN))
1515 IF (ABS(U0)>TWO.EPSILON_EB .OR. ABS(V0)>TWO.EPSILON_EB .OR. ABS(W0)>TWO.EPSILON_EB) &
1516 VEL_CHAR = MAX(VEL_CHAR,SQRT(U0**2+V0**2+W0**2))
1517 DT = CFL.MAX*CHARACTERISTIC.CELL.SIZE/VEL_CHAR
1518 ENDDIF
1519
1520 END SUBROUTINE READ_TIME
1521
1522
1523 SUBROUTINE READ_MULT
1524
1525 REAL(EB) :: DX,DY,DZ,DXB(6),DX0,DY0,DZ0
1526 CHARACTER(LABEL.LENGTH) :: ID
1527 INTEGER :: N,LLOWER,LUPPER,LLOWER,J,LUPPER,K,LOWER,K,LOWER,N,LOWER,N,UPPER
1528 TYPE(MULTIPLIER.TYPE), POINTER :: MR=>NULL()
1529 NAMELIST /MULT/ DX,DXB,DX0,DY,DY0,DZ,DZ0,FYI,ID,LLOWER,LUPPER,J,LOWER,J,LOWER,K,LOWER,K,UPPER,N,LOWER,N,UPPER
1530
1531

```

Source Code files for edited portions of FDS

```

1532 N.MULT = 0
1533 REWIND(LU.INPUT) ; INPUT_FILE.LINE_NUMBER = 0
1534 COUNT.MULT.LOOP: DO
1535 CALL CHECKREAD( 'MULT', LU.INPUT, IOS)
1536 IF (IOS==1) EXIT COUNT.MULT.LOOP
1537 READ(LU.INPUT,NML=MULT,END=9,ERR=10,IOSTAT=IOS)
1538 N.MULT = N.MULT + 1
1539 10 IF (IOS>0) THEN
1540 WRITE(MESSAGE, '(A,I0,A,I0) ' 'ERROR: Problem with MULT number',N.MULT, ', line number',INPUT_FILE.LINE_NUMBER
1541 CALL SHUTDOWN(MESSAGE) ; RETURN
1542 ENDIF
1543 ENDDO COUNT.MULT.LOOP
1544 9 REWIND(LU.INPUT) ; INPUT_FILE.LINE_NUMBER = 0
1545
1546 ALLOCATE(MULTIPLIER(0:N.MULT),STAT=IZERO)
1547 CALL ChkMemErr( 'READ', 'MULTIPLIER', IZERO)
1548
1549 READ.MULT.LOOP: DO N=0,N.MULT
1550
1551 ID = 'null'
1552 IF (N==0) ID = 'MULT DEFAULT'
1553 DX = 0..EB
1554 DY = 0..EB
1555 DZ = 0..EB
1556 DX0 = 0..EB
1557 DY0 = 0..EB
1558 DZ0 = 0..EB
1559 DXB = 0..EB
1560 L.LOWER = 0
1561 I.UPPER = 0
1562 J.LOWER = 0
1563 J.UPPER = 0
1564 K.LOWER = 0
1565 K.UPPER = 0
1566 N.LOWER = 0
1567 N.UPPER = 0
1568
1569 IF (N>0) THEN
1570 CALL CHECKREAD( 'MULT', LU.INPUT, IOS)
1571 IF (IOS==1) EXIT READ.MULT.LOOP
1572 READ(LU.INPUT,MULT)
1573 ENDIF
1574
1575 MR => MULTIPLIER(N)
1576 MR%D = ID
1577 MR%DXB = DXB
1578 MR%DX0 = DX0
1579 MR%DY0 = DY0
1580 MR%DZ0 = DZ0
1581 IF (ABS(DX)>TWO.EPSILON.EB) MR%DXB(1:2) = DX
1582 IF (ABS(DY)>TWO.EPSILON.EB) MR%DXB(3:4) = DY
1583 IF (ABS(DZ)>TWO.EPSILON.EB) MR%DXB(5:6) = DZ
1584
1585 MR%L.LOWER = L.LOWER
1586 MR%I.UPPER = I.UPPER
1587 MR%J.LOWER = J.LOWER
1588 MR%J.UPPER = J.UPPER
1589 MR%K.LOWER = K.LOWER
1590 MR%K.UPPER = K.UPPER
1591 MR%N.COPIES = (I.UPPER-I.LOWER+1)*(J.UPPER-J.LOWER+1)*(K.UPPER-K.LOWER+1)
1592
1593 IF (N.LOWER/=0 .OR. N.UPPER/=0) THEN
1594 MR%SEQUENTIAL = .TRUE.
1595 MR%J.LOWER = N.LOWER
1596 MR%I.UPPER = N.UPPER
1597 MR%J.LOWER = 0
1598 MR%I.UPPER = 0
1599 MR%K.LOWER = 0
1600 MR%K.UPPER = 0
1601 MR%N.COPIES = (N.UPPER-N.LOWER+1)
1602 ENDIF
1603
1604 ENDDO READ.MULT.LOOP
1605 REWIND(LU.INPUT) ; INPUT_FILE.LINE_NUMBER = 0
1606
1607 END SUBROUTINE READ.MULT
1608
1609
1610 SUBROUTINE READ.MISC
1611
1612 USE MATH_FUNCTIONS, ONLY: GET_RAMP_INDEX
1613 REAL(EB) :: MAXIMUM_VISIBILITY
1614 CHARACTER(LABEL_LENGTH) :: ASSUMED_GAS_TEMPERATURE_RAMP, RAMP.GX, RAMP.GY, RAMP.GZ, TURBULENCE_MODEL, &
1615 NEAR_WALL_TURBULENCE_MODEL, COMBUSTION_MODEL_SELECT !Sesa-added this new MISC option COMBUSTION_MODEL_SELECT
1616
1617 NAMELIST /MISC/ AGGLOMERATION, AEROSOL.AL2O3, ALLOW_SURFACE.PARTICLES, ALLOW_UNDERSIDE.PARTICLES,
1618 ASSUMED_GAS_TEMPERATURE, &
1619 ASSUMED_GAS_TEMPERATURE_RAMP, &

```

## Source Code files for edited portions of FDS

```

1619 | BAROCLINIC, BNDF.DEFAULT, CC.IBM, CCVOL.LINK, CC.ZEROIBM.VELO, CHECK.MASS.CONSERVE, &
1620 | CNF.CUTOFF, CFL.MAX, CFL.MIN, CFL.VELOCITY.NORM,&
1621 | CHECK.HT, CHECK.REALIZABILITY, CHECK.VN, CLIP.MASS.FRACTION, COMPUTE.CUTCELLS.ONLY,&
1622 | COMPUTE.VISCOSITY.TWICE, COMPUTE.ZETA.SOURCE.TERM, CONSTANT.H.SOLID, CONSTANT.SPECIFIC.HEAT.RATIO,&
1623 | CORRECT.SUBGRID.TEMPERATURE, COUPLED.1D3D.HEAT.TRANSFER, C.DEARDORFF, C.RNG, C.RNG.CUTOFF, C.SMAGORINSKY, C.VREMAN,&
1624 | C.WALE, DNS, DO.IMPLICIT.CCREGION, DRAG.CFL.MAX, ENTHALPY.TRANSPORT,&
1625 | EVACUATION.DRILL, EVACUATION.MC.MODE, EVAC.PRESSURE.ITERATIONS, EVAC.SURF.DEFAULT, EVAC.TIME.ITERATIONS,&
1626 | EVAPORATION, EXTERNAL.BOUNDARY.CORRECTION, EXTINCTION.MODEL, HVAC.PRES.RELAX, HT3D.TEST,&
1627 | FDS5.OPTIONS, FLUX.LIMITER, FREEZE.VELOCITY, FYI, GAMMA, GRAVITATIONAL.DEPOSITION,&
1628 | GRAVITATIONAL.SETTLING, GVEC, DT.HVAC, H.F.REFERENCE.TEMPERATURE,&
1629 | HRRPUV.MAX.SMV, HUMIDITY, HVAC.MASS.TRANSPORT,&
1630 | IBLANK.SMV, IMMERSED.BOUNDARY.METHOD, INITIAL.UNMIXED.FRACTION,&
1631 | LES.FILTER.WIDTH, MAX.CHEMISTRY.ITERATIONS,&
1632 | MAX.LEAK.PATHS, MAXIMUM.VISIBILITY, MPL.TIMEOUT,&
1633 | N.FIXED.CHEMISTRY.SUBSTEPS, N.INITIAL.PARTICLE.SUBSTEPS, NEAR.WALL.TURBULENCE.MODEL,&
1634 | NOISE, NOISE.VELOCITY, NO.EVACUATION, NO.RAMPS,&
1635 | OVERWRITE, PARTICLE.CFL.MAX, PARTICLE.CFL.MIN, PARTICLE.CFL, PERIODIC.TEST, PROFILING, POROUS.FLOOR,&
1636 | PR.PROJECTION, P.INF, PROCESS.ALL.MESHES, RAMP.GX, RAMP.GY, RAMP.GZ,&
1637 | RADIATION, RESEARCH.MODE, RESTART, RESTART.CHID, RICHARDSON.ERROR.TOLERANCE,&
1638 | SC.SHARED.FILE.SYSTEM, SLIP.CONDITION, SMOKE.ALBEDO, SOLID.PHASE.ONLY, SOOT.OXIDATION,&
1639 | STRATIFICATION, SUPPRESSION, SURF.DEFAULT, TEMPERATURE.DEPENDENT.REACTION,&
1640 | TENSOR.DIFFUSIVITY, TERRAIN.CASE, TERRAIN.IMAGE, TEST.FILTER.QUADRATURE, TEXTURE.ORIGIN,&
1641 | THERMOPHORETIC.DEPOSITION, THICKEN.OBSTRUCTIONS, TRANSPORT.UNMIXED.FRACTION, TRANSPORT.ZETA.SCHEME,&
1642 | TMPA, TURBULENCE.MODEL, TURBULENT.DEPOSITION, TURB.INIT.CLOCK, U.VW.FILE,&
1643 | VEG.LEVEL.SET.COUPLED, VEG.LEVEL.SET.UNCOUPLED, VERBOSE, VISIBILITY.FACTOR, VN.MAX, VN.MIN, Y.CO2.INFNTY, Y.O2.INFNTY,&
1644 | WD.PROPS, WIND.ONLY, COMBUSTION.MODEL.SELECT, HRRPUA.SHEET, HRRPUV.AVERAGE, SIX !Sesa-added to namelist MISC -
      |           COMBUSTION.MODEL.SELECT, HRRPUA.SHEET
1645 | !HRRPUV.AVERAGE
1646 |
1647 | ! Physical constants
1648 |
1649 | TMPA           = 20. _EB                ! Ambient temperature (C)
1650 | GAMMA          = 1.4 _EB                ! Heat capacity ratio for air
1651 | P.INF          = 101325. _EB           ! Ambient pressure (Pa)
1652 | MU.AIR.0      = 1.8E-5. _EB           ! Dynamic Viscosity of Air at 20 C (kg/m/s)
1653 | CP.AIR.0      = 1012. _EB             ! Specific Heat of Air at 20 C (J/kg/K)
1654 | PR.AIR        = 0.7 _EB
1655 | K.AIR.0       = MU.AIR.0*CP.AIR.0/PR.AIR ! Thermal Conductivity of Air at 20 C (W/m/K)
1656 |
1657 | ! Empirical heat transfer constants
1658 |
1659 | PRONTH        = PR.AIR**ONTH
1660 | ASSUMED.GAS.TEMPERATURE = -1000. ! Assumed gas temperature, used for diagnostics
1661 |
1662 | ! Miscellaneous constants
1663 |
1664 | RESTART.CHID  = CHID
1665 | RESTART       = .FALSE.
1666 | NOISE         = .TRUE.
1667 | LES           = .TRUE.
1668 | DNS           = .FALSE.
1669 | SOLID.PHASE.ONLY = .FALSE.
1670 | IBLANK.SMV   = .TRUE.
1671 |
1672 | TEXTURE.ORIGIN(1) = 0. _EB
1673 | TEXTURE.ORIGIN(2) = 0. _EB
1674 | TEXTURE.ORIGIN(3) = 0. _EB
1675 |
1676 | ! EVACuation parameters
1677 |
1678 | EVAC.PRESSURE.ITERATIONS = 50
1679 | EVAC.TIME.ITERATIONS    = 50
1680 | EVACUATION.MC.MODE      = .FALSE.
1681 | NO.EVACUATION           = .FALSE.
1682 | EVACUATION.DRILL       = .FALSE.
1683 |
1684 | ! LES parameters
1685 |
1686 | PR = -1.0_EB ! Turbulent Prandtl number
1687 | SC = -1.0_EB ! Turbulent Schmidt number
1688 |
1689 | ! Misc
1690 |
1691 | ASSUMED.GAS.TEMPERATURE.RAMP = 'null'
1692 | RAMP.GX = 'null'
1693 | RAMP.GY = 'null'
1694 | RAMP.GZ = 'null'
1695 | EXTINCTION.MODEL = 'null'
1696 | GVEC(1) = 0. _EB ! x-component of gravity
1697 | GVEC(2) = 0. _EB ! y-component of gravity
1698 | GVEC(3) = -GRAV ! z-component of gravity
1699 | THICKEN.OBSTRUCTIONS = .FALSE.
1700 | CFL.MAX = 1.0 _EB ! Stability bounds
1701 | CFL.MIN = 0.8 _EB
1702 | VN.MAX = 1.0 _EB
1703 | VN.MIN = 0.8 _EB
1704 | PARTICLE.CFL.MAX = 1. _EB
1705 | PARTICLE.CFL.MIN = 0.8 _EB

```



```

1706 DRAG.CFL_MAX      = 1.0_EB
1707 VEG_LEVEL_SET     = .FALSE.
1708 VEG_LEVEL_SET.COUPLED = .FALSE.
1709 VEG_LEVEL_SET.UNCOUPLED = .FALSE.
1710 TERRAIN_IMAGE     = 'xxxnull'
1711 MAXIMUM_VISIBILITY = 30._EB ! m
1712 VISIBILITY_FACTOR  = 3._EB
1713 TURBULENCE_MODEL   = 'null'
1714 NEAR_WALL_TURBULENCE_MODEL = 'null'
1715 MAX_LEAK_PATHS     = 200
1716 COMBUSTION_MODEL_SELECT = 'null' !Sesa-added for selecting combustion model
1717 HRRPUA_SHEET       = 200._EB !Sesa-added as in 6.2.0 kW/m^2
1718 HRRPUV_AVERAGE    = 2500._EB !Sesa-added as in 6.2.0 kW/m^3
1719 IF (N.MPL.PROCESSES<=50) VERBOSE = .TRUE.
1720
1721 ! Initial read of the MISC line
1722
1723 REWIND(LU.INPUT) ; INPUT_FILE_LINE_NUMBER = 0
1724 MISC_LOOP: DO
1725 CALL CHECKREAD('MISC',LU.INPUT,IOS)
1726 IF (IOS==1) EXIT MISC_LOOP
1727 READ(LU.INPUT,MISC,END=23,ERR=24,Iostat=IOS)
1728 24 IF (IOS>0) THEN ; CALL SHUTDOWN('ERROR: Problem with MISC line') ; RETURN ; ENDIF
1729 ENDDO MISC_LOOP
1730 23 REWIND(LU.INPUT) ; INPUT_FILE_LINE_NUMBER = 0
1731
1732 !Sesa-Combustion model select
1733 !IF (COMBUSTION_MODEL_SELECT/= 'null') THEN
1734 !   WRITE(MESSAGE,'(A,A,A)') 'ERROR: COMBUSTION_MODEL_SELECT, ',TRIM(COMBUSTION_MODEL_SELECT),', is not
!       appropriate.'
1735 !   CALL SHUTDOWN(MESSAGE) ; RETURN
1736 !ELSE
1737 !   COMBUSTION_MODEL_SELECT = 'COMBUSTION_SIX' !Sesa-default model is combustion model of 6.6.0
1738 !ENDIF
1739 !Sesa-Combustion
1740
1741
1742 ! FDS 6 options
1743
1744 IF (DNS) THEN
1745 CFL_VELOCITY_NORM = 1
1746 CFL_MAX = 0.5
1747 CFL_MIN = 0.4
1748 VN_MAX = 0.5
1749 VN_MIN = 0.4
1750 FLUX_LIMITER = CHARM_LIMITER
1751 IF (TURBULENCE_MODEL/= 'null') THEN
1752 WRITE(MESSAGE,'(A,A,A)') 'ERROR: TURBULENCE_MODEL, ',TRIM(TURBULENCE_MODEL),', is not appropriate for DNS.'
1753 CALL SHUTDOWN(MESSAGE) ; RETURN
1754 ENDIF
1755 ELSE
1756 FLUX_LIMITER = SUPERBEE_LIMITER
1757 TURBULENCE_MODEL = 'DEARDORFF'
1758 LES_FILTER_WIDTH = 'MEAN'
1759 ENDIF
1760
1761 ! FDS 5 options (diagnostic timing purposes)
1762
1763 IF (FDS5_OPTIONS) THEN
1764 FLUX_LIMITER = CENTRAL_LIMITER
1765 TURBULENCE_MODEL = 'CONSTANT_SMAGORINSKY'
1766 BAROCLINIC = .FALSE.
1767 CFL_VELOCITY_NORM = 3
1768 CONSTANT_SPECIFIC_HEAT_RATIO = .TRUE.
1769 MAX_PRESSURE_ITERATIONS = 1 ! see PRES
1770 EXTINGUCTION_MODEL = 'EXTINGUCTION_1'
1771 ENDIF
1772
1773 ! Re-read the line to pick up any user-specified options
1774
1775 REWIND(LU.INPUT) ; INPUT_FILE_LINE_NUMBER = 0
1776 CALL CHECKREAD('MISC',LU.INPUT,IOS)
1777 IF (IOS==0) READ(LU.INPUT,MISC)
1778 REWIND(LU.INPUT) ; INPUT_FILE_LINE_NUMBER = 0
1779
1780 ! Temperature conversions
1781
1782 TMPA = TMPA + TMPM
1783 TMPA4 = TMPA**4
1784
1785 ! Miscellaneous
1786
1787 ASSUMED_GAS_TEMPERATURE = ASSUMED_GAS_TEMPERATURE + TMPM
1788 TEX_ORI = TEXTURE_ORIGIN
1789 GRAV = SQRT(DOT_PRODUCT(GVEC,GVEC))
1790
1791 ! Velocity, force, and gravity ramps
1792

```

```

1793 LRAMP_AGT = 0
1794 LRAMP_GX = 0
1795 LRAMP_GY = 0
1796 LRAMP_GZ = 0
1797 NRAMP = 0
1798 ALLOCATE(RAMP_ID(100))
1799 ALLOCATE(RAMP_TYPE(100))
1800 IF (ASSUMED_GAS_TEMPERATURE_RAMP/= 'null') CALL GET_RAMP_INDEX(ASSUMED_GAS_TEMPERATURE_RAMP, 'TIME', LRAMP_AGT)
1801 IF (RAMP_GX/= 'null') CALL GET_RAMP_INDEX(RAMP_GX, 'TIME', LRAMP_GX)
1802 IF (RAMP_GY/= 'null') CALL GET_RAMP_INDEX(RAMP_GY, 'TIME', LRAMP_GY)
1803 IF (RAMP_GZ/= 'null') CALL GET_RAMP_INDEX(RAMP_GZ, 'TIME', LRAMP_GZ)
1804
1805 ! Prandtl and Schmidt numbers
1806
1807 IF (DNS) THEN
1808 LES = .FALSE.
1809 IF (PR<0..EB) PR = 0.7_EB
1810 IF (SC<0..EB) SC = 1.0_EB
1811 ELSE
1812 IF (PR<0..EB) PR = 0.5_EB
1813 IF (SC<0..EB) SC = 0.5_EB
1814 ENDIF
1815
1816 RSC = 1..EB/SC
1817 RPR = 1..EB/PR
1818
1819 ! Check for a restart file
1820
1821 APPEND = .FALSE.
1822 IF (RESTART .AND. RESTART_CHID == CHID) APPEND = .TRUE.
1823 IF (RESTART) NOISE = .FALSE.
1824
1825 ! Min and Max values of flux limiter
1826
1827 IF (FLUX_LIMITER<0 .OR. FLUX_LIMITER>5) THEN
1828 WRITE(MESSAGE, '(A)') 'ERROR on MISC: Permissible values for FLUX_LIMITER=0:5'
1829 CALL SHUTDOWN(MESSAGE) ; RETURN
1830 ENDIF
1831
1832 ! Sesia-Combustion Model
1833 SELECT CASE (TRIM(COMBUSTION_MODEL_SELECT))
1834 CASE ('COMBUSTION_SIX')
1835 COMB_MODEL=COMBUSTIONSIX
1836 CASE ('COMBUSTION_TWO')
1837 COMB_MODEL=COMBUSTIONTWO
1838 END SELECT
1839 ! Sesia
1840
1841 ! Turbulence model
1842
1843 SELECT CASE (TRIM(TURBULENCE_MODEL))
1844 CASE ('CONSTANT_SMAGORINSKY')
1845 TURB_MODEL=CONSMAG
1846 CASE ('DYNAMIC_SMAGORINSKY')
1847 TURB_MODEL=DYNMAG
1848 CASE ('DEARDORFF')
1849 TURB_MODEL=DEARDORFF
1850 CASE ('VREMAN')
1851 TURB_MODEL=VREMAN
1852 CASE ('RNG')
1853 TURB_MODEL=RNG
1854 CASE ('WALE')
1855 TURB_MODEL=WALE
1856 CASE ('null')
1857 TURB_MODEL=NO_TURB_MODEL
1858 CASE DEFAULT
1859 WRITE(MESSAGE, '(A,A,A)') 'ERROR: TURBULENCE_MODEL, ', TRIM(TURBULENCE_MODEL), ', is not recognized.'
1860 CALL SHUTDOWN(MESSAGE) ; RETURN
1861 END SELECT
1862
1863 ! Near wall eddy viscosity model
1864
1865 SELECT CASE (TRIM(NEAR_WALL_TURBULENCE_MODEL))
1866 CASE DEFAULT
1867 NEAR_WALL_TURB_MODEL=CONSMAG
1868 CASE ('WALE')
1869 NEAR_WALL_TURB_MODEL=WALE
1870 END SELECT
1871
1872 ! Extinction Model
1873
1874 IF (TRIM(EXTINCTION_MODEL)/= 'null') THEN
1875 SELECT CASE (TRIM(EXTINCTION_MODEL))
1876 CASE ('EXTINCTION_1')
1877 EXTINCT_MOD = EXTINCTION_1
1878 CASE ('EXTINCTION_2')
1879 ! Single-step product based calculation
1880 EXTINCT_MOD = EXTINCTION_2

```

Source Code files for edited portions of FDS

```

1881 CASE ('EXTINCTION 3')
1882 !Two-step fuel -> CO, CO-> CO2
1883 EXTNCT.MOD = EXTINCTION_3
1884 CASE DEFAULT
1885 WRITE(MESSAGE,'(A,A,A)') 'ERROR: EXTINCTION.MODEL, ',TRIM(EXTINCTION.MODEL),', is not recognized.'
1886 CALL SHUTDOWN(MESSAGE) ; RETURN
1887 END SELECT
1888 ELSE
1889 EXTNCT.MOD = EXTINCTION_2
1890 EXTINCTION.MODEL = 'EXTINCTION 2'
1891 ENDIF
1892
1893 ! Check range of INITIAL_UNMIXED_FRACTION
1894
1895 IF (INITIAL_UNMIXED_FRACTION<0..EB .OR. INITIAL_UNMIXED_FRACTION>1..EB) THEN
1896 WRITE(MESSAGE,'(A)') 'ERROR on MISC: Permissible values for INITIAL_UNMIXED_FRACTION=[0,1]'
1897 CALL SHUTDOWN(MESSAGE) ; RETURN
1898 ENDIF
1899
1900 ! Level set based model of firespread in vegetation
1901
1902 IF (VEG.LEVEL.SET.COUPLED) VEG.LEVEL.SET = .TRUE.
1903 IF (VEG.LEVEL.SET.UNCOUPLED) VEG.LEVEL.SET = .TRUE.
1904 IF (VEG.LEVEL.SET.UNCOUPLED) WIND.ONLY = .TRUE.
1905
1906 IF (HRRPUV.MAX.SMV<0.0) THEN
1907 HRRPUV.MAX.SMV=1200.0
1908 USE_HRRPUV_MAX.SMV=0
1909 ELSE
1910 USE_HRRPUV_MAX.SMV=1
1911 ENDIF
1912
1913 ! Set the lower limit of the extinction coefficient
1914
1915 EC.LL = VISIBILITY.FACTOR/MAXIMUM.VISIBILITY
1916
1917 IF (HUMIDITY<0..EB) HUMIDITY=40..EB
1918
1919 ! Do not allow predefined SURF as DEFAULT
1920
1921 IF (TRIM(SURF.DEFAULT)== 'OPEN' .OR. &
1922 TRIM(SURF.DEFAULT)== 'MIRROR' .OR. &
1923 TRIM(SURF.DEFAULT)== 'INTERPOLATED' .OR. &
1924 TRIM(SURF.DEFAULT)== 'PERIODIC' .OR. &
1925 TRIM(SURF.DEFAULT)== 'HVAC' .OR. &
1926 TRIM(SURF.DEFAULT)== 'MASSLESS TRACER' .OR. &
1927 TRIM(SURF.DEFAULT)== 'DROPLET' .OR. &
1928 TRIM(SURF.DEFAULT)== 'VEGETATION' .OR. &
1929 TRIM(SURF.DEFAULT)== 'EVACUATION.OUTFLOW' .OR. &
1930 TRIM(SURF.DEFAULT)== 'MASSLESS TARGET' ) THEN
1931 WRITE (MESSAGE,'(A,A,A)') 'ERROR: Problem with MISC. Cannot set predefined SURF as SURF.DEFAULT'
1932 CALL SHUTDOWN(MESSAGE) ; RETURN
1933 ENDIF
1934
1935 !Sesa-added as in 6.2.0
1936 !Change units of combustion quantities
1937
1938 HRRPUV.AVERAGE = HRRPUV.AVERAGE*1000..EB !W/m3
1939 HRRPUA.SHEET = HRRPUA.SHEET* 1000..EB !W.m3
1940 !Sesa-added end
1941
1942
1943 FUEL.SMIX.INDEX=2
1944
1945 H.F.REFERENCE.TEMPERATURE = H.F.REFERENCE.TEMPERATURE + TMM
1946
1947 END SUBROUTINE READ_MISC
1948
1949
1950 SUBROUTINE READ_WIND
1951
1952 USE MATH_FUNCTIONS, ONLY: GET_RAMP_INDEX
1953 USE PHYSICAL_FUNCTIONS, ONLY: MONIN_OBUKHOV_SIMILARITY
1954 REAL(EB) :: CORIOLIS_VECTOR(3)=0..EB ,FORCE_VECTOR(3)=0..EB ,OBUKHOV_LENGTH,L,ZZZ,ZETA,Z_0 ,AERODYNAMIC_ROUGHNESS,
SPEED,DIRECTION,&
REFERENCE_HEIGHT,Z_REF,U_STAR,THETA_0,THETA_STAR,TMP,U,REFERENCE_TEMPERATURE,THETA_REF,TMP_REF,P_REF
1955 INTEGER :: NM
1956 LOGICAL :: INITIALIZATION_ONLY
1957 CHARACTER(LABEL_LENGTH) :: RAMP_FVX_T,RAMP_FVY_T,RAMP_FVZ_T,RAMP_U_0_T,RAMP_V_0_T,RAMP_W_0_T,&
RAMP_U_0_Z,RAMP_V_0_Z,RAMP_W_0_Z,RAMP_TMP_0_Z,RAMP_DIRECTION,RAMP_SPEED
1958 TYPE(RESERVED_RAMP_TYPE) , POINTER :: RRP,RRP2
1959 EQUIVALENCE(Z_0 ,AERODYNAMIC_ROUGHNESS)
1960 EQUIVALENCE(Z_REF ,REFERENCE_HEIGHT)
1961 EQUIVALENCE(TMP_REF ,REFERENCE_TEMPERATURE)
1962 EQUIVALENCE(L ,OBUKHOV_LENGTH)
1963 REAL(EB) , PARAMETER :: KAPPA.VK = 0.41..EB
1964
1965
1966
1967 NAMELIST /WIND/ CORIOLIS_VECTOR,DIRECTION,DT_MEAN_FORCING,FORCE_VECTOR,FYI,GROUND_LEVEL,INITIALIZATION_ONLY,L,&

```

Source Code files for edited portions of FDS

```

1968 LAPSE_RATE,MEAN_FORCING,OBUKHOV_LENGTH,&
1969 POTENTIAL_TEMPERATURE_CORRECTION,RAMP_DIRECTION,RAMP_SPEED,RAMP_FVX_T,RAMP_FVY_T,RAMP_FVZ_T,&
1970 RAMP_TMP0_Z,RAMP_U0_T,RAMP_V0_T,RAMP_W0_T,RAMP_U0_Z,RAMP_V0_Z,RAMP_W0_Z,REFERENCE_HEIGHT,REFERENCE_TEMPERATURE,&
1971 SPEED,SPONGE_CELLS,STRATIFICATION,THETA_STAR,TMP_REF,U_STAR,U0,V0,W0,Z_0,Z_REF
1972
1973 ! Default values
1974
1975 DIRECTION = 270._EB ! westerly wind
1976 DT_MEAN_FORCING = 1._EB ! s
1977 INITIALIZATION_ONLY = .FALSE.
1978 LAPSE_RATE = 0._EB ! K/m
1979 MEAN_FORCING = .FALSE.
1980 OBUKHOV_LENGTH = 0._EB ! m
1981 RAMP_DIRECTION = 'null'
1982 RAMP_SPEED = 'null'
1983 RAMP_U0_T = 'null'
1984 RAMP_V0_T = 'null'
1985 RAMP_W0_T = 'null'
1986 RAMP_U0_Z = 'null'
1987 RAMP_V0_Z = 'null'
1988 RAMP_W0_Z = 'null'
1989 RAMP_TMP0_Z = 'null'
1990 RAMP_FVX_T = 'null'
1991 RAMP_FVY_T = 'null'
1992 RAMP_FVZ_T = 'null'
1993 SPEED = -1._EB ! m/s
1994 SPONGE_CELLS = 3
1995 THETA_STAR = 0._EB ! K
1996 TMP_REF = -1._EB ! C
1997 U_STAR = -1._EB ! m/s
1998 U0 = 0._EB ! m/s
1999 V0 = 0._EB ! m/s
2000 W0 = 0._EB ! m/s
2001 Z_0 = 0.03._EB ! m
2002 Z_REF = 2._EB ! m
2003
2004 ! Initial read of the WIND line
2005
2006 REWIND(LU_INPUT) ; INPUT_FILE_LINE_NUMBER = 0
2007 WIND_LOOP: DO
2008 CALL CHECKREAD('WIND',LU_INPUT,IOS)
2009 IF (IOS==1) EXIT WIND_LOOP
2010 READ(LU_INPUT,WIND,END=23,ERR=24,IOSTAT=IOS)
2011 24 IF (IOS>0) THEN ; CALL SHUTDOWN('ERROR: Problem with WIND line') ; RETURN ; ENDF
2012 ENDDO WIND_LOOP
2013 23 REWIND(LU_INPUT) ; INPUT_FILE_LINE_NUMBER = 0
2014
2015 ! Check compatibility of constant specific heat ratio and stratification
2016
2017 IF (CONSTANT_SPECIFIC_HEAT_RATIO .AND. STRATIFICATION) THEN
2018 WRITE(MESSAGE,'(A,A,A)') 'ERROR: CONSTANT_SPECIFIC_HEAT_RATIO option is incompatible with STRATIFICATION.'
2019 CALL SHUTDOWN(MESSAGE) ; RETURN
2020 ENDF
2021
2022 ! Do not impose MEAN_FORCING if the user just wants to initialize the wind speed
2023
2024 IF (INITIALIZATION_ONLY) MEAN_FORCING = .FALSE.
2025
2026 ! Convert wind speed to directions
2027
2028 IF (U_STAR>0._EB) SPEED = U_STAR*LOG(Z_REF/Z_0)/KAPPA_VK
2029 DIRECTION = DIRECTION*PI/180._EB
2030 IF (SPEED>0._EB) THEN
2031 IF (RAMP_DIRECTION/='null') THEN
2032 U0 = SPEED
2033 V0 = SPEED
2034 ELSE
2035 U0 = -SPEED*SIN(DIRECTION)
2036 V0 = -SPEED*COS(DIRECTION)
2037 ENDF
2038 ELSE
2039 SPEED = SQRT(U0**2 + V0**2)
2040 ENDF
2041
2042 ! Apply mean forcing in all directions if any wind component is non-zero
2043
2044 IF ((U0/=0._EB .OR. V0/=0._EB .OR. W0/=0._EB) .AND. .NOT. INITIALIZATION_ONLY) MEAN_FORCING = .TRUE.
2045
2046 ! Miscellaneous
2047
2048 FVEC = FORCE_VECTOR
2049 OVEC = CORIOLIS_VECTOR
2050
2051 ! Velocity, force, and gravity ramps
2052
2053 LRAMP_DIRECTION = 0
2054 LRAMP_SPEED = 0
2055 LRAMP_U0_T = 0

```

Source Code files for edited portions of FDS

```

2056 I.RAMP.V0.T = 0
2057 I.RAMP.W0.T = 0
2058 I.RAMP.U0.Z = 0
2059 I.RAMP.V0.Z = 0
2060 I.RAMP.W0.Z = 0
2061 I.RAMP.TMP0.Z = 0
2062 I.RAMP.FVX.T = 0
2063 I.RAMP.FVY.T = 0
2064 I.RAMP.FVZ.T = 0
2065
2066 IF (RAMP.DIRECTION/= 'null') THEN ! create dummy time RAMPs for U0 and V0 and fill in later
2067 RAMP.U0.T = RAMP.DIRECTION
2068 RAMP.V0.T = RAMP.DIRECTION
2069 ENDIF
2070 IF (RAMP.U0.T/= 'null') CALL GET_RAMP_INDEX(RAMP.U0.T, 'TIME', I.RAMP.U0.T, DUPLICATE_RAMP=.TRUE.)
2071 IF (RAMP.V0.T/= 'null') CALL GET_RAMP_INDEX(RAMP.V0.T, 'TIME', I.RAMP.V0.T, DUPLICATE_RAMP=.TRUE.)
2072 IF (RAMP.W0.T/= 'null') CALL GET_RAMP_INDEX(RAMP.W0.T, 'TIME', I.RAMP.W0.T)
2073 IF (RAMP.U0.Z/= 'null') CALL GET_RAMP_INDEX(RAMP.U0.Z, 'PROFILE', I.RAMP.U0.Z)
2074 IF (RAMP.V0.Z/= 'null') CALL GET_RAMP_INDEX(RAMP.V0.Z, 'PROFILE', I.RAMP.V0.Z)
2075 IF (RAMP.W0.Z/= 'null') CALL GET_RAMP_INDEX(RAMP.W0.Z, 'PROFILE', I.RAMP.W0.Z)
2076 IF (RAMP.FVX.T/= 'null') CALL GET_RAMP_INDEX(RAMP.FVX.T, 'TIME', I.RAMP.FVX.T)
2077 IF (RAMP.FVY.T/= 'null') CALL GET_RAMP_INDEX(RAMP.FVY.T, 'TIME', I.RAMP.FVY.T)
2078 IF (RAMP.FVZ.T/= 'null') CALL GET_RAMP_INDEX(RAMP.FVZ.T, 'TIME', I.RAMP.FVZ.T)
2079 IF (RAMP.SPEED/= 'null') CALL GET_RAMP_INDEX(RAMP.SPEED, 'TIME', I.RAMP.SPEED)
2080 IF (RAMP.DIRECTION/= 'null') CALL GET_RAMP_INDEX(RAMP.DIRECTION, 'TIME', I.RAMP.DIRECTION)
2081
2082 IF (STRATIFICATION) THEN
2083
2084 IF (RAMP.TMP0.Z== 'null' .AND. ABS(OBUKHOV_LENGTH)<1.E-10.EB) THEN
2085 N.RESERVED_RAMPs = N.RESERVED_RAMPs + 1
2086 RRP => RESERVED_RAMPs(N.RESERVED_RAMPs)
2087 ALLOCATE(RRP%INDEPENDENT_DATA(2))
2088 ALLOCATE(RRP%DEPENDENT_DATA(2))
2089 RRP%INDEPENDENT_DATA(1) = ZS_MIN
2090 RRP%INDEPENDENT_DATA(2) = ZF_MAX
2091 RRP%DEPENDENT_DATA(1) = (TMPA+LAPSE_RATE*(ZS_MIN-GROUND_LEVEL))/TMPA
2092 RRP%DEPENDENT_DATA(2) = (TMPA+LAPSE_RATE*(ZF_MAX-GROUND_LEVEL))/TMPA
2093 RRP%NUMBER_DATA_POINTS = 2
2094 RAMP.TMP0.Z = 'RSRVD TEMPERATURE PROFILE'
2095 CALL GET_RAMP_INDEX(RAMP.TMP0.Z, 'PROFILE', I.RAMP.TMP0.Z)
2096 ENDIF
2097
2098 IF (ABS(OBUKHOV_LENGTH)>1.E-10.EB) THEN
2099 N.RESERVED_RAMPs = N.RESERVED_RAMPs + 1
2100 RRP => RESERVED_RAMPs(N.RESERVED_RAMPs)
2101 RRP%NUMBER_DATA_POINTS = 51
2102 N.RESERVED_RAMPs = N.RESERVED_RAMPs + 1
2103 RRP2 => RESERVED_RAMPs(N.RESERVED_RAMPs)
2104 RRP2%NUMBER_DATA_POINTS = 51
2105 ALLOCATE(RRP%INDEPENDENT_DATA(51))
2106 ALLOCATE(RRP%DEPENDENT_DATA(51))
2107 ALLOCATE(RRP2%INDEPENDENT_DATA(51))
2108 ALLOCATE(RRP2%DEPENDENT_DATA(51))
2109 IF (U_STAR<0..EB) U_STAR = KAPPA.VK*SPEED/LOG(Z.REF/Z.0)
2110 IF (TMP.REF<0..EB) THEN
2111 TMP.REF = TMPA
2112 ELSE
2113 TMP.REF = TMP.REF + TMPM ! C to K
2114 ENDIF
2115 P.REF = P.INF - RHOA*GRAV*(Z.REF-Z.0)
2116 THETA.REF = TMP.REF*(P.INF/P.REF)**0.286..EB
2117 IF (ABS(THETA.STAR)<1.E-10.EB) THEN
2118 THETA.0 = THETA.REF/(1..EB+U_STAR**2*LOG(Z.REF/Z.0)/(GRAV*KAPPA.VK**2*OBUKHOV_LENGTH))
2119 THETA.STAR = U_STAR**2*THETA.0/(GRAV*KAPPA.VK*OBUKHOV_LENGTH)
2120 ELSE
2121 THETA.0 = THETA.REF - THETA.STAR*LOG(Z.REF/Z.0)/KAPPA.VK
2122 ENDIF
2123 TMPA = THETA.0 ! Make the ground temperature the new ambient temperature
2124 DO I=1,51
2125 ZETA = (1-1)*(ZF_MAX-ZS_MIN)/50.
2126 ZZZ = Z.0*EXP(LOG(ZF_MAX/Z.0)*(ZETA-ZS_MIN)/(ZF_MAX-ZS_MIN))
2127 CALL MONIN_OBUKHOV_SIMILARITY(ZZZ, Z.0, OBUKHOV_LENGTH, U_STAR, THETA.STAR, THETA.0, U, TMP)
2128 RRP%INDEPENDENT_DATA(1) = ZZZ
2129 RRP%DEPENDENT_DATA(1) = TMP/TMPA
2130 RRP2%INDEPENDENT_DATA(1) = ZZZ
2131 RRP2%DEPENDENT_DATA(1) = U/SPEED
2132 ENDDO
2133 RAMP.TMP0.Z = 'RSRVD TEMPERATURE PROFILE'
2134 CALL GET_RAMP_INDEX(RAMP.TMP0.Z, 'PROFILE', I.RAMP.TMP0.Z)
2135 RAMP.U0.Z = 'RSRVD VELOCITY PROFILE'
2136 CALL GET_RAMP_INDEX(RAMP.U0.Z, 'PROFILE', I.RAMP.U0.Z)
2137 RAMP.V0.Z = 'RSRVD VELOCITY PROFILE'
2138 CALL GET_RAMP_INDEX(RAMP.V0.Z, 'PROFILE', I.RAMP.V0.Z)
2139 ENDIF
2140
2141 ! Add a RAMP for the vertical profile of pressure (the values are computed in INIT)
2142
2143 N.RESERVED_RAMPs = N.RESERVED_RAMPs + 1

```

Source Code files for edited portions of FDS

```

2144 CALL GET_RAMP_INDEX('RSRVD PRESSURE PROFILE','PROFILE',I,RAMP,P0,Z)
2145 RRP => RESERVED_RAMPS(N_RESERVED_RAMPS)
2146 ALLOCATE(RRP%INDEPENDENT_DATA(2))
2147 ALLOCATE(RRP%DEPENDENT_DATA(2))
2148 RRP%INDEPENDENT_DATA(1) = ZS_MIN
2149 RRP%INDEPENDENT_DATA(2) = ZF_MAX
2150 RRP%DEPENDENT_DATA(1) = 0._EB      ! Dummy values to be filled in later
2151 RRP%DEPENDENT_DATA(2) = 1._EB      ! Dummy values to be filled in later
2152 RRP%NUMBER_DATA_POINTS = 2
2153
2154 ENDIF
2155
2156 ! External kinetic energy
2157
2158 H0 = 0.5._EB*(U0**2+V0**2+W0**2)
2159
2160 ! Allocation for mean forcing (required here, instead of init, because of hole feature)
2161
2162 IF (ANY(MEAN_FORCING)) THEN
2163 DO NM=1,NMESHES
2164 IF (MYID/=PROCESS(NM) .OR. EVACUATION_ONLY(NM)) CYCLE
2165 M=>MESHES(NM)
2166 ALLOCATE(M%MEAN_FORCING_CELL(0:M%IBP1,0:M%JBP1,0:M%KBP1),STAT=IZERO)
2167 CALL ChkMemErr('INIT','MEAN_FORCING_CELL',IZERO)
2168 M%MEAN_FORCING_CELL=.TRUE.
2169 ENDDO
2170 ENDIF
2171
2172 ! Min and Max values of temperature
2173
2174 TMPMIN = MAX(1._EB, MIN(TMPA,TMPM) -10._EB)
2175 IF (LAPSE_RATE < 0._EB) TMPMIN = MIN(TMPMIN,TMPA+LAPSE_RATE*(ZF_MAX-GROUND_LEVEL))
2176 TMPMAX = 3000._EB
2177
2178 END SUBROUTINE READ_WIND
2179
2180
2181 SUBROUTINE PROC_WIND
2182
2183 ! This short routine takes a time ramp of wind speed and direction and converts to Cartesian ramps for U0, V0
2184
2185 REAL(EB) :: THETA
2186 INTEGER :: I
2187
2188 IF (LRAMP_DIRECTION/=0) THEN
2189 DO I=0,RAMPS(LRAMP_DIRECTION)%NUMBER_INTERPOLATION_POINTS+1
2190 THETA = RAMPS(LRAMP_DIRECTION)%INTERPOLATED_DATA(1)*PI/180._EB
2191 RAMPS(LRAMP_U0_T)%INTERPOLATED_DATA(1) = -SIN(THETA)
2192 RAMPS(LRAMP_V0_T)%INTERPOLATED_DATA(1) = -COS(THETA)
2193 ENDDO
2194 ENDIF
2195
2196 END SUBROUTINE PROC_WIND
2197
2198
2199 SUBROUTINE READ_DUMP
2200
2201 ! Read parameters associated with output files
2202
2203 REAL(EB) :: DT_DEFAULT
2204 INTEGER :: N,SIG_FIGS,SIG_FIGS_EXP
2205 NAMELIST /DUMP/ CLIP_RESTART_FILES,COLUMN_DUMP_LIMIT,CTRL_COLUMN_LIMIT,&
2206 DEVC_COLUMN_LIMIT,DT_BNDE,DT_BNDF,DT_CPU,DT_CTRL,DT_DEVC,DT_DEVC_LINE,DT_FLUSH,&
2207 DT_GEOM,DT_HRR,DT_ISOJ,DT_MASS,DT_PART,DT_PL3D,DT_PROF,DT_RESTART,DT_SL3D,DT_SLCF,EB_PART_FILE,&
2208 FLUSH_FILE_BUFFERS,GEOM_DIAG,MASS_FILE,MAXIMUM_PARTICLES,MMS_TIMER,NFRAMES,PLOT3D_PART_ID,PLOT3D_QUANTITY,&
2209 PLOT3D_SPEC_ID,PLOT3D_SPEC_ID,PLOT3D_VELO_INDEX,RENDER_FILE,SIG_FIGS,SIG_FIGS_EXP,SMOKE3D,SMOKE3D_QUANTITY,&
2210 SMOKE3D_SPEC_ID,STATUS_FILES,SUPPRESS_DIAGNOSTICS,UUV_TIMER,VELOCITY_ERROR_FILE,WRITE_XYZ
2211
2212 ! Set defaults
2213
2214 GEOM_DIAG = .FALSE.
2215 FLUSH_FILE_BUFFERS = .TRUE.
2216 MAXIMUM_PARTICLES = 1000000
2217 MMS_TIMER = 1.E10._EB
2218 NFRAMES = 1000
2219 PLOT3D_QUANTITY(1) = 'TEMPERATURE'
2220 PLOT3D_QUANTITY(2) = 'U-VELOCITY'
2221 PLOT3D_QUANTITY(3) = 'V-VELOCITY'
2222 PLOT3D_QUANTITY(4) = 'W-VELOCITY'
2223 PLOT3D_QUANTITY(5) = 'HRRPUV'
2224 PLOT3D_PART_ID = 'null'
2225 PLOT3D_SPEC_ID = 'null'
2226 PLOT3D_VELO_INDEX = 0
2227 RENDER_FILE = 'null'
2228 SMOKE3D = .TRUE.
2229 SMOKE3D_QUANTITY = 'null'
2230 SMOKE3D_SPEC_ID = 'null'
2231 SIG_FIGS = 8

```

```

2232 SIG_FIGS_EXP = 3
2233 IF (NMESHES>32) THEN
2234 SUPPRESS_DIAGNOSTICS = .TRUE.
2235 ELSE
2236 SUPPRESS_DIAGNOSTICS = .FALSE.
2237 ENDIF
2238 UVW_TIMER = 1.E10_EB
2239
2240 DT_GEOM = 1._EB
2241 DT_BNDE = -1._EB
2242 DT_BNDF = -1._EB
2243 DT_CPU = 1000000._EB
2244 DT_RESTART = 1000000._EB
2245 DT_FLUSH = -1._EB
2246 DT_DEVC = -1._EB
2247 DT_DEVC_LINE = 0.5_EB*(T.END-T.BEGIN)
2248 DT_HRR = -1._EB
2249 DT_ISOF = -1._EB
2250 DT_MASS = -1._EB
2251 DT_PART = -1._EB
2252 DT_PL3D = 1.E10_EB
2253 DT_PROF = -1._EB
2254 DT_SLCF = -1._EB
2255 DT_SL3D = 0.2_EB*(T.END-T.BEGIN)
2256 DT_CTRL = -1._EB
2257
2258 ! Read the DUMP line
2259
2260 REWIND(LU_INPUT) ; INPUT_FILE_LINE_NUMBER = 0
2261 DUMP_LOOP: DO
2262 CALL CHECKREAD('DUMP',LU_INPUT,IOS)
2263 IF (IOS==1) EXIT DUMP_LOOP
2264 READ(LU_INPUT,DUMP,END=23,ERR=24,IOSTAT=IOS)
2265 24 IF (IOS>0) THEN ; CALL SHUTDOWN('ERROR: Problem with DUMP line') ; RETURN ; ENDIF
2266 ENDDO DUMP_LOOP
2267 23 REWIND(LU_INPUT) ; INPUT_FILE_LINE_NUMBER = 0
2268
2269 ! Set output time intervals
2270
2271 DT_DEFAULT = (T.END - T.BEGIN)/REAL(NFRAMES,EB)
2272
2273 IF (DT_BNDE < 0._EB) THEN ; DT_BNDE = 2._EB*DT_DEFAULT ; ELSE ; DT_BNDE = DT_BNDE /TIME_SHRINK_FACTOR ; ENDIF
2274 IF (DT_BNDF < 0._EB) THEN ; DT_BNDF = 2._EB*DT_DEFAULT ; ELSE ; DT_BNDF = DT_BNDF /TIME_SHRINK_FACTOR ; ENDIF
2275 IF (DT_DEVC < 0._EB) THEN ; DT_DEVC = DT_DEFAULT ; ELSE ; DT_DEVC = DT_DEVC /TIME_SHRINK_FACTOR ; ENDIF
2276 IF (DT_HRR < 0._EB) THEN ; DT_HRR = DT_DEFAULT ; ELSE ; DT_HRR = DT_HRR /TIME_SHRINK_FACTOR ; ENDIF
2277 IF (DT_ISOF < 0._EB) THEN ; DT_ISOF = DT_DEFAULT ; ELSE ; DT_ISOF = DT_ISOF /TIME_SHRINK_FACTOR ; ENDIF
2278 IF (DT_MASS < 0._EB) THEN ; DT_MASS = DT_DEFAULT ; ELSE ; DT_MASS = DT_MASS /TIME_SHRINK_FACTOR ; ENDIF
2279 IF (DT_PART < 0._EB) THEN ; DT_PART = DT_DEFAULT ; ELSE ; DT_PART = DT_PART /TIME_SHRINK_FACTOR ; ENDIF
2280 IF (DT_PROF < 0._EB) THEN ; DT_PROF = DT_DEFAULT ; ELSE ; DT_PROF = DT_PROF /TIME_SHRINK_FACTOR ; ENDIF
2281 IF (DT_SLCF < 0._EB) THEN ; DT_SLCF = DT_DEFAULT ; ELSE ; DT_SLCF = DT_SLCF /TIME_SHRINK_FACTOR ; ENDIF
2282 IF (DT_CTRL < 0._EB) THEN ; DT_CTRL = DT_DEFAULT ; ELSE ; DT_CTRL = DT_CTRL /TIME_SHRINK_FACTOR ; ENDIF
2283 IF (DT_FLUSH < 0._EB) THEN ; DT_FLUSH = DT_DEFAULT ; ELSE ; DT_FLUSH = DT_FLUSH /TIME_SHRINK_FACTOR ; ENDIF
2284
2285 ! Check Plot3D QUANTITIES
2286
2287 PLOOP: DO N=1,5
2288 CALL GET_QUANTITY_INDEX(PLOT3D.SMOKEVIEW_LABEL(N),PLOT3D.SMOKEVIEW_BAR_LABEL(N),PLOT3D.QUANTITY_INDEX(N),LDUM(1)
2289 , &
2290 PLOT3D.Y_INDEX(N),PLOT3D.Z_INDEX(N),PLOT3D.PART_INDEX(N),LDUM(2),LDUM(3),LDUM(4),'PLOT3D', &
2291 PLOT3D.QUANTITY(N),'null',PLOT3D.SPEC_ID(N),PLOT3D.PART_ID(N),'null','null','null')
2292 IF (OUTPUT_QUANTITY(PLOT3D.QUANTITY_INDEX(N))%INTEGRATED_PARTICLES) PL3D.PARTICLE_FLUX = .TRUE.
2293 ENDDO PLOOP
2294
2295 ! Check SMOKE3D viability
2296
2297 IF (TWO.D .OR. SOLID_PHASE_ONLY) SMOKE3D = .FALSE.
2298
2299 IF (SMOKE3D.QUANTITY=='null') THEN
2300 IF (SOOT_INDEX > 0) THEN
2301 SMOKE3D.QUANTITY = 'MASS FRACTION'
2302 SMOKE3D.SPEC_ID = SPECIES(SOOT_INDEX)%ID
2303 ELSE
2304 IF (N_REACTIONS > 0) THEN
2305 SMOKE3D.QUANTITY = 'HRRPUV'
2306 ELSE
2307 SMOKE3D = .FALSE.
2308 ENDIF
2309 ENDIF
2310
2311 IF (SMOKE3D) THEN
2312 CALL GET_QUANTITY_INDEX(SMOKE3D.SMOKEVIEW_LABEL,SMOKE3D.SMOKEVIEW_BAR_LABEL,SMOKE3D.QUANTITY_INDEX,LDUM(1), &
2313 SMOKE3D.Y_INDEX,SMOKE3D.Z_INDEX,LDUM(2),LDUM(3),LDUM(4),LDUM(5),'SMOKE3D', &
2314 SMOKE3D.QUANTITY,'null',SMOKE3D.SPEC_ID,'null','null','null','null')
2315 ENDIF
2316
2317 ! Set format of real number output
2318

```

Source Code files for edited portions of FDS

```

2319 WRITE(FMT,R,'(A,I2.2,A,I2.2,A,I1.1)') 'ES',SIG_FIGS+SIG_FIGS_EXP+4,'.',SIG_FIGS-1,'E',SIG_FIGS_EXP
2320
2321 END SUBROUTINE READDUMP
2322
2323 SUBROUTINE READSPEC
2324
2325 USE MATH_FUNCTIONS, ONLY : GET_RAMP_INDEX
2326 USE PHYSICAL_FUNCTIONS, ONLY : WATER_VAPOR_MASS_FRACTION
2327 USE PROPERTY_DATA, ONLY : GAS_PROPS, FED_PROPS, CHECK_PREDEFINED
2328 USE SOOT_ROUTINES
2329 REAL (EB) :: MW, SIGMALJ, EPSILONKJ, VISCOSITY, CONDUCTIVITY, DIFFUSIVITY, MASS_EXTINCTION_COEFFICIENT, &
2330 SPECIFIC_HEAT, REFERENCE_ENTHALPY, REFERENCE_TEMPERATURE, FIC_CONCENTRATION, FLD_LETHAL_DOSE, &
2331 SPECIFIC_HEAT_LIQUID, DENSITY_LIQUID, VAPORIZATION_TEMPERATURE, HEAT_OF_VAPORIZATION, MELTING_TEMPERATURE, &
2332 H.V.REFERENCE_TEMPERATURE, MEAN_DIAMETER, CONDUCTIVITY_SOLID, DENSITY_SOLID, ENTHALPY_OF_FORMATION, MASS_FRACTION_0, &
2333 CONVERSION, PR_GAS, CONDUCTIVITY_LIQUID, VISCOSITY_LIQUID, BETA_LIQUID, H_F_IN
2334 REAL (EB) :: MASS_FRACTION (MAX_SPECIES), VOLUME_FRACTION (MAX_SPECIES), MIN_DIAMETER, MAX_DIAMETER
2335 INTEGER :: N_SPEC_READ, N, NN, NNN, NS2, NR, N_SPEC_READ_2, N_SUB_SPECIES, NS, N_BINS, N_COPY, N_FOUND
2336 INTEGER, ALLOCATABLE, DIMENSION(:) :: Y_INDEX
2337 LOGICAL :: LUMPED_COMPONENT_ONLY, AEROSOL, BACKGROUND, &
2338 DEFINED_BACKGROUND, REAC_FUEL_READ = .FALSE., PRIMITIVE, COPY_LUMPED
2339 LOGICAL, ALLOCATABLE, DIMENSION(:) :: PREDEFINED, PREDEFINED_SMIX, NEW_PRIMITIVE
2340 CHARACTER (LABEL_LENGTH) :: RAMP_CP, RAMP_CPL, RAMP_K, RAMP_MU, RAMP_D, RADCAL_ID, RAMP_G.F, SPEC_ID (MAX_SPECIES)
2341
2342 CHARACTER (LABEL_LENGTH), ALLOCATABLE, DIMENSION(:) :: PREDEFINED_SPEC_ID, SPEC_ID_READ
2343 CHARACTER (LABEL_LENGTH) :: FORMULA
2344 TYPE (SPECIES_TYPE), POINTER :: SS => NULL()
2345 TYPE (SPECIES_MIXTURE_TYPE), POINTER :: SM => NULL()
2346 NAMELIST /SPEC/ AEROSOL, BACKGROUND, BETA_LIQUID, CONDUCTIVITY, CONDUCTIVITY_LIQUID, CONDUCTIVITY_SOLID, COPY_LUMPED, &
2347 DENSITY_LIQUID, DENSITY_SOLID, DIFFUSIVITY, ENTHALPY_OF_FORMATION, EPSILONKJ, FIC_CONCENTRATION, FLD_LETHAL_DOSE, &
2348 FORMULA, FYI, HEAT_OF_VAPORIZATION, H.V.REFERENCE_TEMPERATURE, ID, LUMPED_COMPONENT_ONLY, &
2349 MASS_EXTINCTION_COEFFICIENT, MASS_FRACTION, MASS_FRACTION_0, MAX_DIAMETER, MEAN_DIAMETER, MELTING_TEMPERATURE, &
2350 MIN_DIAMETER, MW, N_BINS, PR_GAS, PRIMITIVE, RADCAL_ID, &
2351 RAMP_CP, RAMP_CPL, RAMP_D, RAMP_G.F, RAMP_K, RAMP_MU, REFERENCE_ENTHALPY, REFERENCE_TEMPERATURE, SIGMALJ, SPEC_ID, &
2352 SPECIFIC_HEAT, SPECIFIC_HEAT_LIQUID, VAPORIZATION_TEMPERATURE, VISCOSITY, VISCOSITY_LIQUID, VOLUME_FRACTION
2353
2354 IF (SIMPLE_CHEMISTRY) THEN
2355 MINT_SPECIES = 9
2356 ELSE
2357 MINT_SPECIES = 5
2358 ENDIF
2359 REWIND (LU_INPUT) ; INPUT_FILE_LINE_NUMBER = 0
2360 N_SPECIES = 0
2361
2362 COUNT_SPEC_LINES : DO
2363 READ (LU_INPUT, NML=SPEC, END=19, ERR=20, IOSTAT=IOS)
2364 MINT_SPECIES = MINT_SPECIES + 1
2365 IF (N_PARTICLE_BINS > 0) THEN
2366 MINT_SPECIES = MINT_SPECIES + N_PARTICLE_BINS
2367 ENDIF
2368 20 IF (IOS > 0) THEN
2369 WRITE (MESSAGE, '(A,I0,A,I0)') 'ERROR: Problem with SPECies number', N_SPECIES + 1, ' ', line number',
2370 INPUT_FILE_LINE_NUMBER
2371 CALL SHUTDOWN (MESSAGE) ; RETURN
2372 ENDIF
2373 N_SPECIES = N_SPECIES + 1
2374 ENDDO COUNT_SPEC_LINES
2375 19 REWIND (LU_INPUT) ; INPUT_FILE_LINE_NUMBER = 0
2376
2377 ! Allocate species inputs
2378 ALLOCATE (PREDEFINED (MINT_SPECIES))
2379 ALLOCATE (PREDEFINED_SMIX (1 : MINT_SPECIES))
2380 ALLOCATE (NEW_PRIMITIVE (0 : MINT_SPECIES))
2381 NEW_PRIMITIVE = .FALSE.
2382 ALLOCATE (PREDEFINED_SPEC_ID (MINT_SPECIES))
2383 ALLOCATE (SPEC_ID_READ (MINT_SPECIES))
2384
2385 ! Create predefined inputs related to simple chemistry mode
2386
2387 PREDEFINED = .FALSE.
2388 PREDEFINED_SMIX = .FALSE.
2389
2390 N_SPEC_READ = 0
2391 N_TRACKED_SPECIES = 1
2392
2393 IF (SIMPLE_CHEMISTRY) THEN
2394 N_SPECIES = 7
2395 PREDEFINED (1:7) = .TRUE.
2396 PREDEFINED_SPEC_ID (1) = REACTION (1)%FUEL
2397 PREDEFINED_SPEC_ID (2) = 'NITROGEN'
2398 PREDEFINED_SPEC_ID (3) = 'OXYGEN'
2399 PREDEFINED_SPEC_ID (4) = 'CARBON_DIOXIDE'
2400 PREDEFINED_SPEC_ID (5) = 'CARBON_MONOXIDE'
2401 PREDEFINED_SPEC_ID (6) = 'WATER_VAPOR'
2402 PREDEFINED_SPEC_ID (7) = 'SOOT'
2403 PREDEFINED_SMIX (1:3) = .TRUE.
2404 IF (REACTION (1)%C <= 0 .OR. REACTION (1)%H <= 0 .OR. REACTION (1)%F <= 0) PREDEFINED_SMIX (2) = .FALSE.
2405 N_TRACKED_SPECIES = 3

```



## Source Code files for edited portions of FDS

```

2406 ELSE
2407 ! If not simple chemistry look for a background species and if none force creation of AIR lumped species
2408 REWIND(LU_INPUT) ; INPUT_FILE.LINE_NUMBER = 0
2409 CHECK.BACKGROUND_LOOP: DO
2410 BACKGROUND = .FALSE.
2411 READ(LU_INPUT,NML=SPEC,END=21,IOSTAT=IOS)
2412 IF (BACKGROUND) EXIT
2413 ENDDO CHECK.BACKGROUND_LOOP
2414 21 REWIND(LU_INPUT) ; INPUT_FILE.LINE_NUMBER = 0
2415 IF (BACKGROUND) THEN
2416 N_SPECIES = 0
2417 DEFINED.BACKGROUND = .TRUE.
2418 ELSE
2419 N_SPECIES = 4
2420 PREDEFINED(1:4) = .TRUE.
2421 PREDEFINED.SPEC_ID(1) = 'NITROGEN'
2422 PREDEFINED.SPEC_ID(2) = 'OXYGEN'
2423 PREDEFINED.SPEC_ID(3) = 'CARBON DIOXIDE'
2424 PREDEFINED.SPEC_ID(4) = 'WATER VAPOR'
2425 PREDEFINED.SMX(1) = .TRUE.
2426 DEFINED.BACKGROUND = .FALSE.
2427 ENDIF
2428 ENDIF
2429
2430 ! Pass 1: Count SPEC lines determine number of primitive and tracked species and check for errors
2431
2432 REWIND(LU_INPUT) ; INPUT_FILE.LINE_NUMBER = 0
2433 COUNT_SPEC_LOOP: DO
2434 CALL SET_SPEC_DEFAULT
2435 CALL CHECKREAD('SPEC',LU_INPUT,IOS)
2436 IF (IOS==1) EXIT COUNT_SPEC_LOOP
2437 READ(LU_INPUT,NML=SPEC,END=29,IOSTAT=IOS)
2438 N_SPEC_READ = N_SPEC_READ + 1
2439 SPEC_ID_READ(N_SPEC_READ) = ID
2440
2441 IF (ID=='null') THEN
2442 WRITE(MESSAGE,'(A,10,A)') 'ERROR: Species ',N_SPEC_READ, ' needs a name (ID=...)'
2443 CALL SHUTDOWN(MESSAGE) ; RETURN
2444 ENDIF
2445
2446 ! Prevent use of 'AIR' unless a new BACKGROUND has been defined.
2447
2448 IF (ID=='AIR' .AND. .NOT. DEFINED.BACKGROUND) THEN
2449 WRITE(MESSAGE,'(A,10,A)') 'ERROR: SPEC ',N_SPEC_READ, ': Cannot redefine AIR without defining a BACKGROUND species'
2450 CALL SHUTDOWN(MESSAGE) ; RETURN
2451 ENDIF
2452
2453 ! Make sure both ramps and constant values have not been given
2454
2455 IF (SPECIFIC_HEAT > 0..EB .AND. RAMP.CP/='null') THEN
2456 WRITE(MESSAGE,'(A,A,A)') 'ERROR: SPEC ',TRIM(ID),': Cannot specify both SPECIFIC_HEAT and RAMP.CP'
2457 CALL SHUTDOWN(MESSAGE) ; RETURN
2458 ENDIF
2459 IF (SPECIFIC_HEAT.LIQUID > 0..EB .AND. RAMP.CP.L/='null') THEN
2460 WRITE(MESSAGE,'(A,A,A)') 'ERROR: SPEC ',TRIM(ID),': Cannot specify both SPECIFIC_HEAT.LIQUID and RAMP.CP.L'
2461 CALL SHUTDOWN(MESSAGE) ; RETURN
2462 ENDIF
2463 IF (CONDUCTIVITY > 0..EB .AND. RAMP.K/='null') THEN
2464 WRITE(MESSAGE,'(A,A,A)') 'ERROR: SPEC ',TRIM(ID),': Cannot specify both CONDUCTIVITY and RAMP.K'
2465 CALL SHUTDOWN(MESSAGE) ; RETURN
2466 ENDIF
2467 IF (DIFFUSIVITY > 0..EB .AND. RAMP.D/='null') THEN
2468 WRITE(MESSAGE,'(A,A,A)') 'ERROR: SPEC ',TRIM(ID),': Cannot specify both DIFFUSIVITY and RAMP.D'
2469 CALL SHUTDOWN(MESSAGE) ; RETURN
2470 ENDIF
2471 IF (VISCOSITY > 0..EB .AND. RAMP.MU/='null') THEN
2472 WRITE(MESSAGE,'(A,A,A)') 'ERROR: SPEC ',TRIM(ID),': Cannot specify both VISCOSITY and RAMP.MU'
2473 CALL SHUTDOWN(MESSAGE) ; RETURN
2474 ENDIF
2475
2476 ! REFERENCE.ENTHALPY requires additional parameters
2477
2478 IF (REFERENCE.ENTHALPY > -2.E20.EB .AND. (SPECIFIC_HEAT < 0..EB .AND. RAMP.CP=='null')) THEN
2479 WRITE(MESSAGE,'(A,A,A)') 'ERROR: SPEC ',TRIM(ID),': REFERENCE.ENTHALPY requires SPECIFIC_HEAT or RAMP.CP'
2480 CALL SHUTDOWN(MESSAGE) ; RETURN
2481 ENDIF
2482
2483 DO NN = 1,N_SPEC_READ-1
2484 IF (ID==SPEC_ID_READ(NN)) THEN
2485 WRITE(MESSAGE,'(A,10,A,10,A)') 'ERROR: Species ',N_SPEC_READ, ' has the same ID as species ',NN, '.'
2486 CALL SHUTDOWN(MESSAGE) ; RETURN
2487 ENDIF
2488 ENDDO
2489
2490 IF (BACKGROUND) THEN
2491 IF (LUMPED.COMPONENT_ONLY) THEN
2492 WRITE(MESSAGE,'(A)') 'ERROR: Cannot define a LUMPED.COMPONENT_ONLY species as the BACKGROUND species'

```

Source Code files for edited portions of FDS

```

2493 CALL SHUTDOWN(MESSAGE) ; RETURN
2494 ENDIF
2495 IF (SIMPLE.CHEMISTRY) THEN
2496 WRITE(MESSAGE,'(A)') 'ERROR: Cannot define a BACKGROUND species if using the simple chemistry'
2497 CALL SHUTDOWN(MESSAGE) ; RETURN
2498 ENDIF
2499 ENDIF
2500 IF (LUMPED.COMPONENT.ONLY .AND. MASS.FRACTION.0>0..EB) THEN
2501 WRITE(MESSAGE,'(A)') 'ERROR: Cannot define MASS.FRACTION.0 for a LUMPED.COMPONENT.ONLY species'
2502 CALL SHUTDOWN(MESSAGE) ; RETURN
2503 ENDIF
2504
2505 IF (PRIMITIVE) THEN
2506 IF (SPEC_ID(1)/='null' .AND. SPEC_ID(2)=='null') THEN
2507 NEW_PRIMITIVE(N_SPEC_READ)=.TRUE.
2508 IF (.NOT. CHECK.PREDEFINED(SPEC_ID(1))) THEN
2509 WRITE(MESSAGE,'(A,I0,A,I0,A)') 'ERROR: SPEC_ID(1) for species ',N_SPEC_READ,' must be a predefined species'
2510 CALL SHUTDOWN(MESSAGE) ; RETURN
2511 ENDIF
2512 ELSEIF (SPEC_ID(1)/='null' .AND. SPEC_ID(2)/='null') THEN
2513 WRITE(MESSAGE,'(A,I0,A,I0,A)') 'ERROR: Species ',N_SPEC_READ,' is declared PRIMITIVE and has more than one
SPEC_ID given'
2514 CALL SHUTDOWN(MESSAGE) ; RETURN
2515 ENDIF
2516 ENDIF
2517
2518 IF (SPEC_ID(1)=='null' .OR. PRIMITIVE) THEN
2519 N_SPECIES = N_SPECIES+1
2520 IF (SIMPLE.CHEMISTRY) THEN
2521 IF (TRIM(ID)/=TRIM(REACTION(1)%FUEL)) THEN
2522 IF (.NOT. LUMPED.COMPONENT.ONLY .AND. .NOT. BACKGROUND) N_TRACKED_SPECIES = N_TRACKED_SPECIES + 1
2523 ENDIF
2524 ELSE
2525 IF (.NOT. LUMPED.COMPONENT.ONLY .AND. .NOT. BACKGROUND) N_TRACKED_SPECIES = N_TRACKED_SPECIES + 1
2526 ENDIF
2527 ELSE
2528 IF (SIMPLE.CHEMISTRY) THEN
2529 IF (TRIM(ID)/=TRIM(REACTION(1)%FUEL)) N_TRACKED_SPECIES = N_TRACKED_SPECIES + 1
2530 ELSE
2531 IF (.NOT. BACKGROUND) N_TRACKED_SPECIES = N_TRACKED_SPECIES + 1
2532 ENDIF
2533 ENDIF
2534
2535 ! If predefined species, check to see if the species has already been defined.
2536 IF (PREDEFINED.SMIX(1)) THEN
2537 DO NN=1,N_SPECIES-1
2538 IF (TRIM(PREDEFINED.SPEC_ID(NN))==TRIM(ID)) THEN
2539 IF (.NOT. SIMPLE.CHEMISTRY) THEN
2540 IF (SPEC_ID(1)/='null') THEN
2541 WRITE(MESSAGE,'(A,I0,A)') 'ERROR: Species ',N_SPEC_READ+1, &
2542 '. Lumped species has the same ID as a predefined species'
2543 CALL SHUTDOWN(MESSAGE) ; RETURN
2544 ENDIF
2545 ELSE
2546 IF (SPEC_ID(1)/='null') THEN
2547 IF (TRIM(ID)/=TRIM(REACTION(1)%FUEL)) THEN
2548 WRITE(MESSAGE,'(A,I0,A)') 'ERROR: Species ',N_SPEC_READ+1, &
2549 '. Lumped species has the same ID as a predefined species'
2550 CALL SHUTDOWN(MESSAGE) ; RETURN
2551 ELSE
2552 PREDEFINED.SMIX(2) = .FALSE.
2553 REAC_FUEL_READ = .TRUE.
2554 ENDIF
2555 ELSE
2556 IF (TRIM(ID)==TRIM(REACTION(1)%FUEL)) REAC_FUEL_READ = .TRUE.
2557 ENDIF
2558 ENDIF
2559 PREDEFINED(NN) = .FALSE.
2560 N_SPECIES = N_SPECIES - 1
2561 EXIT
2562 ENDIF
2563 ENDDO
2564 ENDIF
2565 ENDDO COUNT SPEC LOOP
2566
2567 29 REWIND(LU.INPUT) ; INPUT_FILE.LINE_NUMBER = 0
2568
2569 IF (SIMPLE.CHEMISTRY .AND. .NOT. (REAC_FUEL_READ .OR. SIMPLE_FUEL_DEFINED)) THEN
2570 WRITE(MESSAGE,'(A,A,A)') 'ERROR: Simple chemistry fuel, ',TRIM(REACTION(1)%FUEL),' , not defined on REAC or SPEC.'
2571 CALL SHUTDOWN(MESSAGE) ; RETURN
2572 ENDIF
2573
2574 ! Allocate the primitive species array.
2575 ALLOCATE(SPECIES(N_SPECIES),STAT=IZERO)
2576 CALL ChkMemErr('READ','SPECIES',IZERO)
2577
2578 ALLOCATE(Y_INDEX(N_SPECIES))
2579

```

```

2580 ! Pass 2: read and process primitive species
2581 N = 0
2582 N_SPEC_READ_2 = N_SPEC_READ
2583 N_SPEC_READ = 0
2584
2585 PRIMITIVE_SPEC_READ_LOOP: DO WHILE (N_SPEC_READ < N_SPEC_READ_2 .OR. N < N_SPECIES)
2586 N = N + 1
2587
2588 CALL SET_SPEC_DEFAULT
2589
2590 IF (PREDEFINED(N)) THEN
2591 ID = PREDEFINED_SPEC_ID(N)
2592 LUMPED_COMPONENT_ONLY = .TRUE.
2593 ELSE
2594 READ(LU_INPUT,NML=SPEC)
2595 N_SPEC_READ = N_SPEC_READ + 1
2596 IF (SIMPLE_CHEMISTRY) THEN
2597 IF (TRIM(ID) == TRIM(REACTION(1)%FUEL)) PREDEFINED_SMIX(2) = .FALSE.
2598 ENDIF
2599
2600 IF (SPEC_ID(1) /= 'null' .AND. .NOT. NEW_PRIMITIVE(N_SPEC_READ)) THEN
2601 N = N - 1
2602 CYCLE PRIMITIVE_SPEC_READ_LOOP
2603 ENDIF
2604
2605 ENDIF
2606
2607 SS => SPECIES(N)
2608
2609 SS%K_USER = CONDUCTIVITY
2610 SS%CONDUCTIVITY_SOLID = CONDUCTIVITY_SOLID
2611 SS%D_USER = DIFFUSIVITY
2612 SS%DENSITY_SOLID = DENSITY_SOLID
2613 SS%EPSK = EPSILON_KLJ
2614 SS%FIC_CONCENTRATION = FIC_CONCENTRATION
2615 SS%FLD_LETHAL_DOSE = FLD_LETHAL_DOSE
2616 SS%FORMULA = FORMULA
2617 SS%H_F = ENTHALPY_OF_FORMATION*1000._EB
2618 SS%ID = ID
2619 SS%RADCAL_ID = RADCAL_ID
2620 SS%MASS_EXTINCTION_COEFFICIENT = MAX(0._EB, MASS_EXTINCTION_COEFFICIENT)
2621 SS%MEAN_DIAMETER = MEAN_DIAMETER
2622 SS%MU_USER = VISCOSITY
2623 SS%MW = MW
2624 SS%PR_USER = PR_GAS
2625 SS%RAMP_CP = RAMP_CP
2626 SS%RAMP_CP_L = RAMP_CP_L
2627 SS%RAMP_D = RAMP_D
2628 SS%RAMP_K = RAMP_K
2629 SS%RAMP_G_F = RAMP_G_F
2630 SS%RAMP_MU = RAMP_MU
2631 IF (REFERENCE_TEMPERATURE < -TMPM) REFERENCE_TEMPERATURE = 25._EB
2632 SS%REFERENCE_TEMPERATURE = REFERENCE_TEMPERATURE + TMPM
2633 SS%SIG = SIGMA_LJ
2634 SS%SPECIFIC_HEAT = SPECIFIC_HEAT*1000._EB
2635 SS%REFERENCE_ENTHALPY = REFERENCE_ENTHALPY*1000._EB
2636 SS%Y0 = MAX(0._EB, MASS_FRACTION_0)
2637
2638 SS%DENSITY_LIQUID = DENSITY_LIQUID
2639 SS%BETA_LIQUID = BETA_LIQUID
2640 SS%K_LIQUID = CONDUCTIVITY_LIQUID
2641 SS%MU_LIQUID = VISCOSITY_LIQUID
2642
2643 IF ((HEAT_OF_VAPORIZATION > 0._EB .AND. SPECIFIC_HEAT_LIQUID <= 0._EB) .OR. &
2644 (HEAT_OF_VAPORIZATION <= 0._EB .AND. SPECIFIC_HEAT_LIQUID > 0._EB)) THEN
2645 WRITE(MESSAGE, '(A,I0,A)') 'ERROR: SPEC ', N_SPEC_READ, &
2646 ': If one of SPECIFIC_HEAT_LIQUID or HEAT_OF_VAPORIZATION defined, both must be'
2647 CALL SHUTDOWN(MESSAGE) ; RETURN
2648 ENDIF
2649 IF (SPECIFIC_HEAT_LIQUID > 0._EB) THEN
2650 IF (MELTING_TEMPERATURE < -TMPM) THEN
2651 WRITE(MESSAGE, '(A,I0,A)') 'ERROR: SPEC ', N_SPEC_READ, ': MELTING_TEMPERATURE not set'
2652 CALL SHUTDOWN(MESSAGE) ; RETURN
2653 ENDIF
2654 SS%SPECIFIC_HEAT_LIQUID = SPECIFIC_HEAT_LIQUID*1000._EB
2655 SS%HEAT_OF_VAPORIZATION = HEAT_OF_VAPORIZATION*1000._EB
2656 SS%TMP_MELT = MELTING_TEMPERATURE + TMPM
2657 IF (H.V.REFERENCE_TEMPERATURE < -TMPM) H.V.REFERENCE_TEMPERATURE = MELTING_TEMPERATURE
2658 SS%H.V.REFERENCE_TEMPERATURE = H.V.REFERENCE_TEMPERATURE + 273.15._EB
2659 IF (VAPORIZATION_TEMPERATURE < -TMPM) THEN
2660 WRITE(MESSAGE, '(A,I0,A)') 'ERROR: SPEC ', N_SPEC_READ, ': VAPORIZATION_TEMPERATURE not set'
2661 CALL SHUTDOWN(MESSAGE) ; RETURN
2662 ENDIF
2663 SS%TMP_V = VAPORIZATION_TEMPERATURE + TMPM
2664 ENDIF
2665
2666 IF (N_BINS > 0) THEN
2667 IF (AGGLOMERATION_INDEX > 0) THEN

```

Source Code files for edited portions of FDS

```

2668 WRITE(MESSAGE,'(A,I0,A)') 'ERROR: SPEC ',N.SPEC.READ,' : Can only specify N.BINS for one &SPEC input'
2669 CALL SHUTDOWN(MESSAGE) ; RETURN
2670 ENDIF
2671 IF (N.BINS < 2) THEN
2672 WRITE(MESSAGE,'(A,I0,A)') 'ERROR: SPEC ',N.SPEC.READ,' : N.BINS must be >=2'
2673 CALL SHUTDOWN(MESSAGE) ; RETURN
2674 ENDIF
2675 IF (.NOT. AEROSOL) THEN
2676 WRITE(MESSAGE,'(A,I0,A)') 'ERROR: SPEC ',N.SPEC.READ,' : AEROSOL must be .TRUE. to use N.BINS'
2677 CALL SHUTDOWN(MESSAGE) ; RETURN
2678 ENDIF
2679 IF (MAX.DIAMETER < 0..EB) THEN
2680 WRITE(MESSAGE,'(A,I0,A)') 'ERROR: SPEC ',N.SPEC.READ,' : MAX.DIAMETER not set'
2681 CALL SHUTDOWN(MESSAGE) ; RETURN
2682 ENDIF
2683 IF (MIN.DIAMETER < 0..EB) THEN
2684 WRITE(MESSAGE,'(A,I0,A)') 'ERROR: SPEC ',N.SPEC.READ,' : MIN.DIAMETER not set'
2685 CALL SHUTDOWN(MESSAGE) ; RETURN
2686 ENDIF
2687 IF (MAX.DIAMETER <= MIN.DIAMETER) THEN
2688 WRITE(MESSAGE,'(A,I0,A)') 'ERROR: SPEC ',N.SPEC.READ,' : MAX.DIAMETER <= MIN.DIAMETER'
2689 CALL SHUTDOWN(MESSAGE) ; RETURN
2690 ENDIF
2691 IF (.NOT. LUMPED.COMPONENT.ONLY) THEN
2692 WRITE(MESSAGE,'(A,I0,A)') 'ERROR: SPEC ',N.SPEC.READ,' : LUMPED.COMPONENT.ONLY must be .TRUE. to use N.BINS'
2693 CALL SHUTDOWN(MESSAGE) ; RETURN
2694 ENDIF
2695 N.PARTICLE.BINS = N.BINS
2696 MAX.PARTICLE.DIAMETER = MAX.DIAMETER
2697 MIN.PARTICLE.DIAMETER = MIN.DIAMETER
2698 N.TRACKED.SPECIES=N.TRACKED.SPECIES+N.PARTICLE.BINS
2699 AGGLOMERATION.INDEX=N
2700 MIN.PARTICLE.DIAMETER = MIN.PARTICLE.DIAMETER * 1.E-6.EB
2701 MAX.PARTICLE.DIAMETER = MAX.PARTICLE.DIAMETER * 1.E-6.EB
2702 SS%AGGLOMERATING = .TRUE.
2703 CALL INITIALIZE.AGGLOMERATION
2704 ENDIF
2705
2706 IF (NEW.PRIMITIVE(N.SPEC.READ)) THEN
2707 SS%PROP.ID = SPEC.ID(1)
2708 ELSE
2709 SS%PROP.ID = ID
2710 ENDIF
2711 H.F.IN = SS%H.F
2712 CALL GAS.PROPS(SS%PROP.ID,SS%SIG,SS%EPSK,SS%PR.GAS,SS%MW,SS%FORMULA,SS%LISTED,SS%ATOMS,SS%H.F,SS%RADCAL.ID)
2713 IF (SIMPLE.CHEMISTRY) THEN
2714 IF (TRIM(SS%ID)==TRIM(REACTION(1)%FUEL) .AND. .NOT. SS%LISTED) SS%H.F = H.F.IN
2715 ENDIF
2716 CALL FED.PROPS(SS%PROP.ID,SS%FLD.LETHAL.DOSE,SS%FC.CONCENTRATION)
2717
2718 IF (SS%H.F > -1.E23.EB) SS%EXPLICIT.H.F=.TRUE.
2719
2720 IF (SS%SPECIFIC.HEAT > 0..EB) THEN
2721 ! H.F overrides REFERENCE.ENTHALPY
2722 IF (ENTHALPY.OF.FORMATION > -1.E23.EB) THEN
2723 SS%REFERENCE.ENTHALPY = SS%H.F/SS%MW*1000..EB - SS%SPECIFIC.HEAT*H.F.REFERENCE.TEMPERATURE
2724 ELSE
2725 IF (SS%REFERENCE.ENTHALPY < -1.E20.EB) SS%REFERENCE.ENTHALPY = SS%SPECIFIC.HEAT * SS%REFERENCE.TEMPERATURE
2726 SS%H.F = (SS%REFERENCE.ENTHALPY + SS%SPECIFIC.HEAT * (H.F.REFERENCE.TEMPERATURE-SS%REFERENCE.TEMPERATURE))*SS%MW
2727 !Adjust SS%REFERENCE.ENTHALPY to 0 K
2728 SS%REFERENCE.ENTHALPY = SS%REFERENCE.ENTHALPY - SS%SPECIFIC.HEAT * SS%REFERENCE.TEMPERATURE
2729 ENDIF
2730 ENDIF
2731
2732 IF (SS%RAMP.CP/= 'null' .AND. SS%REFERENCE.ENTHALPY < -1.E20.EB) SS%REFERENCE.ENTHALPY = 0..EB
2733
2734 IF (TRIM(SS%FORMULA)=='null') WRITE(SS%FORMULA,'(A,I0)') 'SPEC.',N
2735
2736 ! For simple chemistry Determine if the species is the one specified on the REAC line(s)
2737
2738 IF (SIMPLE.CHEMISTRY) THEN
2739 IF (TRIM(ID)==TRIM(REACTION(1)%FUEL)) THEN
2740 FUEL.INDEX = N
2741 WRITE(FORMULA,'(A,I0)') 'SPEC.',N
2742 IF (TRIM(SS%FORMULA)==TRIM(FORMULA)) SS%MW = REACTION(1)%MW.FUEL
2743 ENDIF
2744 IF (TRIM(ID)=='SOOT') SS%MW = REACTION(1)%MW.SOOT
2745 ENDIF
2746
2747 SS%RCON = R0/SS%MW
2748 SS%MODE = GAS.SPECIES
2749
2750 ! Special processing of certain species
2751
2752 SELECT CASE (ID)
2753 CASE ('WATER.VAPOR')
2754 H2O.INDEX = N

```

Source Code files for edited portions of FDS

```

2755 IF (MASS.FRACTION_0 > 0._EB .AND. LUMPED.COMPONENT_ONLY) THEN
2756 WRITE(MESSAGE,'(A)') 'WARNING: MASS.FRACTION_0 specified for WATER VAPOR with LUMPED.COMPONENT_ONLY = .TRUE.'
2757 IF (MYID==0) WRITE(LU_ERR,'(A)') TRIM(MESSAGE)
2758 ENDIF
2759 IF (PREDEFINED_SMIX(1)) Y_H2O_INFNTY = WATER.VAPOR.MASS.FRACTION(HUMIDITY,TMPA,P_INF)
2760 CASE('CARBON DIOXIDE')
2761 CO2_INDEX = N
2762 CASE('CARBON MONOXIDE')
2763 CO_INDEX = N
2764 CASE('OXYGEN')
2765 O2_INDEX = N
2766 CASE('NITROGEN')
2767 N2_INDEX = N
2768 CASE('HYDROGEN')
2769 H2_INDEX = N
2770 CASE('HYDROGEN CYANIDE')
2771 HCN_INDEX = N
2772 CASE('NITRIC OXIDE')
2773 NO_INDEX = N
2774 CASE('NITROGEN DIOXIDE')
2775 NO2_INDEX = N
2776 CASE('SOOT')
2777 SOOT_INDEX = N
2778 IF (MASS.EXTINCTION_COEFFICIENT < 0._EB) SS%MASS.EXTINCTION_COEFFICIENT = 8700._EB
2779 END SELECT
2780
2781 IF (SS%RADCAL_ID=='SOOT' .AND. SOOT_INDEX==0) SOOT_INDEX = N
2782 IF (SS%ID=='SOOT' .AND. AEROSOL.AL2O3) SS%DENSITY_SOLID = 4000.
2783 IF (AEROSOL) SS%MODE = AEROSOL_SPECIES
2784
2785 ! Get ramps
2786 IF (SS%RAMP_CP/='null') THEN
2787 CALL GET_RAMP_INDEX(SS%RAMP_CP,'TEMPERATURE',NR)
2788 SS%RAMP_CP_INDEX = NR
2789 ENDIF
2790 IF (SS%RAMP_CP_L/='null') THEN
2791 CALL GET_RAMP_INDEX(SS%RAMP_CP_L,'TEMPERATURE',NR)
2792 SS%RAMP_CP_L_INDEX = NR
2793 ENDIF
2794 IF (SS%RAMP_D/='null') THEN
2795 CALL GET_RAMP_INDEX(SS%RAMP_D,'TEMPERATURE',NR)
2796 SS%RAMP_D_INDEX = NR
2797 ENDIF
2798 IF (SS%RAMP_G_F/='null') THEN
2799 CALL GET_RAMP_INDEX(SS%RAMP_G_F,'TEMPERATURE',NR)
2800 SS%RAMP_G_F_INDEX = NR
2801 ENDIF
2802 IF (SS%RAMP_K/='null') THEN
2803 CALL GET_RAMP_INDEX(SS%RAMP_K,'TEMPERATURE',NR)
2804 SS%RAMP_K_INDEX = NR
2805 ENDIF
2806 IF (SS%RAMP_MU/='null') THEN
2807 CALL GET_RAMP_INDEX(SS%RAMP_MU,'TEMPERATURE',NR)
2808 SS%RAMP_MU_INDEX = NR
2809 ENDIF
2810 ENDDO PRIMITIVE_SPEC_READ_LOOP
2811
2812 ! Unmixed fraction
2813 IF (TRANSPORT_UNMIXED_FRACTION) THEN
2814 N_PASSIVE_SCALARS=1
2815 ZETA_INDEX=N_TRACKED_SPECIES+1
2816 ENDIF
2817
2818 ! IMPORTANT: define number of total tracked scalars
2819 N_TOTAL_SCALARS=N_TRACKED_SPECIES+N_PASSIVE_SCALARS
2820
2821 ! Pass 3: process tracked species (primitive and lumped)
2822 ALLOCATE(SPECIES_MIXTURE(1:N_TOTAL_SCALARS),STAT=IZERO)
2823 CALL ChkMemErr('READ','SPECIES_MIXTURE',IZERO)
2824
2825 ! Process non-predefined mixtures first
2826 REWIND(LU_INPUT) ; INPUT_FILE_LINE_NUMBER = 0
2827 N = 1
2828 DEFINED_BACKGROUND = .FALSE.
2829 N_SPEC_READ = 0
2830 N_COPY = 0
2831 N_FOUND = 1
2832
2833 TRACKED_SPEC_LOOP_1: DO WHILE (N_FOUND <= N_TRACKED_SPECIES .OR. .NOT. DEFINED_BACKGROUND)
2834 IF (PREDEFINED_SMIX(N)) THEN
2835 CALL SET_SPEC_DEFAULT
2836 IF (N==1) BACKGROUND = .TRUE.
2837 ELSE
2838 FIND_TRACKED: DO
2839 CALL SET_SPEC_DEFAULT
2840 READ(LU_INPUT,NML=SPEC)
2841 N_SPEC_READ = N_SPEC_READ + 1
2842 IF (LUMPED_COMPONENT_ONLY .AND. N_BINS < 0) CYCLE FIND_TRACKED

```

Source Code files for edited portions of FDS

```

2843 IF (COPY_LUMPED) THEN
2844 IF (N_BINS > 0) THEN
2845 WRITE(MESSAGE, '(A,A,A)') 'ERROR: SPEC ', TRIM(SM%ID), ', cannot specify both COPY_LUMPED and N_BINS.'
2846 CALL SHUTDOWN(MESSAGE) ; RETURN
2847 ENDIF
2848 IF (BACKGROUND) THEN
2849 WRITE(MESSAGE, '(A,A,A)') 'ERROR: SPEC ', TRIM(SM%ID), ', cannot specify both COPY_LUMPED and BACKGROUND.'
2850 CALL SHUTDOWN(MESSAGE) ; RETURN
2851 ENDIF
2852 N_COPY = N_COPY+1
2853 EXIT FIND_TRACKED
2854 ENDIF
2855 IF (ANY(MASS_FRACTION > 0.5EB) .AND. ANY(VOLUME_FRACTION > 0.5EB)) THEN
2856 WRITE(MESSAGE, '(A,A,A)') 'ERROR: SPEC ', TRIM(SM%ID), ', cannot specify both MASS_FRACTION and VOLUME_FRACTION.'
2857 CALL SHUTDOWN(MESSAGE) ; RETURN
2858 ENDIF
2859 EXIT
2860 ENDDO FIND_TRACKED
2861 IF (COPY_LUMPED) THEN
2862 N_FOUND = N + N_COPY
2863 CYCLE TRACKED_SPEC_LOOP_1
2864 ENDIF
2865 IF (N_BINS > 0) THEN
2866 DO NNN=1, N_PARTICLE_BINS
2867 MEAN_DIAMETER = 2.5EB*PARTICLE_RADIUS(NNN)
2868 SPEC_ID(1) = SPECIES (AGGLOMERATION_INDEX)%ID
2869 WRITE(ID, '(A,A,I0)') TRIM(SPECIES (AGGLOMERATION_INDEX)%ID), '-', NNN
2870 MASS_FRACTION(1) = 1.5EB
2871 CALL DEFINE_MIXTURE
2872 N = N + 1
2873 ENDDO
2874 AGGLOMERATION_INDEX = N - N_PARTICLE_BINS
2875 N = N - 1
2876 ELSE
2877 CALL DEFINE_MIXTURE
2878 ENDIF
2879
2880 IF (SIMPLE_CHEMISTRY) THEN
2881 SM => SPECIES_MIXTURE(N)
2882 IF (TRIM(SM%ID) == TRIM(REACTION(1)%FUEL)) THEN
2883 FUEL_SMIX_INDEX = N
2884 IF (ABS(SM%ATOMS(1)+SM%ATOMS(6)+SM%ATOMS(7)+SM%ATOMS(8) - SUM(SM%ATOMS)) > SPACING(SUM(SM%ATOMS))) THEN
2885 WRITE(MESSAGE, '(A)') 'ERROR: Fuel FORMULA for SIMPLE_CHEMISTRY can only contain C,H,O, and N'
2886 CALL SHUTDOWN(MESSAGE) ; RETURN
2887 ELSE
2888 REACTION(1)%C = SM%ATOMS(6)
2889 REACTION(1)%H = SM%ATOMS(1)
2890 REACTION(1)%O = SM%ATOMS(8)
2891 REACTION(1)%N = SM%ATOMS(7)
2892 REACTION(1)%MW_FUEL = SM%MW
2893 ENDIF
2894 IF (REACTION(1)%C <= TWO_EPSILON_5EB .AND. REACTION(1)%H <= TWO_EPSILON_5EB) THEN
2895 WRITE(MESSAGE, '(A)') 'ERROR: Must specify fuel chemistry using C and/or H when using simple chemistry'
2896 CALL SHUTDOWN(MESSAGE) ; RETURN
2897 ENDIF
2898 ENDIF
2899 ENDIF
2900 ENDIF
2901 IF (BACKGROUND) THEN
2902 DEFINED_BACKGROUND = .TRUE.
2903 IF (N==1) N = N + 1
2904 ELSE
2905 N = N + 1
2906 ENDIF
2907 N_FOUND = N + N_COPY
2908 ENDDO TRACKED_SPEC_LOOP_1
2909
2910 IF (N_COPY >= 1) NNN = N
2911
2912 ! Process predefined mixtures second
2913
2914 IF (ANY(PREDEFINED_SMIX)) THEN
2915 REWIND(LU_INPUT) ; INPUT_FILE_LINE_NUMBER = 0
2916 N = 1
2917 DEFINED_BACKGROUND = .FALSE.
2918 TRACKED_SPEC_LOOP_2: DO WHILE (N <= N_TOTAL_SCALARS .OR. .NOT. DEFINED_BACKGROUND)
2919 IF (PREDEFINED_SMIX(N)) THEN
2920 CALL SET_SPEC_DEFAULT
2921 CALL SETUP_PREDEFINED_SMIX(N)
2922 IF (N==1) BACKGROUND = .TRUE.
2923 CALL DEFINE_MIXTURE
2924 ELSE
2925 BACKGROUND = .FALSE.
2926 ENDIF
2927 IF (BACKGROUND) THEN
2928 DEFINED_BACKGROUND = .TRUE.
2929 IF (N==1) N = N + 1
2930 ELSE

```

```

2931 N = N + 1
2932 ENDIF
2933 ENDDO TRACKED.SPEC.LOOP.2
2934 ENDIF
2935
2936 REWIND (LU.INPUT)
2937
2938 TRACKED.SPEC.LOOP.3: DO NN = 1, N.COPY
2939 FIND.TRACKED.2: DO
2940 CALL SET.SPEC.DEFAULT
2941 READ(LU.INPUT,NML=SPEC)
2942 IF (.NOT. COPY.LUMPED) CYCLE FIND.TRACKED.2
2943 DO N=1,NNN-1
2944 IF (SPECIES.MIXTURE(N)%ID==SPEC.ID(1)) THEN
2945 SPECIES.MIXTURE(NN) = SPECIES.MIXTURE(N)
2946 SPECIES.MIXTURE(NN)%ID = ID
2947 EXIT FIND.TRACKED.2
2948 ELSE
2949 IF (N==NNN-1) THEN
2950 WRITE(MESSAGE,'(A,A,A,A)') 'ERROR: SPEC ',TRIM(ID),', cannot find tracked species ',TRIM(SPEC.ID(1))
2951 CALL SHUTDOWN(MESSAGE) ; RETURN
2952 ENDIF
2953 ENDIF
2954 ENDDO
2955 ENDDO FIND.TRACKED.2
2956 NNN = NNN + 1
2957 ENDDO TRACKED.SPEC.LOOP.3
2958
2959 REWIND (LU.INPUT) ; INPUT.FILE.LINE.NUMBER = 0
2960
2961 ! Normalize the initial mass fractions of the lumped species if necessary
2962 IF (SUM(SPECIES.MIXTURE(2:N.TRACKED.SPECIES)%ZZO) > 1._EB) &
2963 SPECIES.MIXTURE(2:N.TRACKED.SPECIES)%ZZO = SPECIES.MIXTURE(2:N.TRACKED.SPECIES)%ZZO/ &
2964 SUM(SPECIES.MIXTURE(2:N.TRACKED.SPECIES)%ZZO)
2965
2966 SPECIES.MIXTURE(1)%ZZO = 1._EB - SUM(SPECIES.MIXTURE(2:N.TRACKED.SPECIES)%ZZO)
2967
2968 DEPOSITION = ANY(SPECIES.MIXTURE%DEPOSITING) .AND. DEPOSITION
2969
2970 ! Deallocate species inputs
2971 DEALLOCATE(PREDEFINED)
2972 DEALLOCATE(PREDEFINED.SMIX)
2973 DEALLOCATE(NEW.PRIMITIVE)
2974 DEALLOCATE(PREDEFINED.SPEC.ID)
2975 DEALLOCATE(SPEC.ID.READ)
2976 DEALLOCATE(Y.INDEX)
2977
2978 CONTAINS
2979
2980 SUBROUTINE DEFINE.MIXTURE
2981 USE PROPERTY.DATA, ONLY:GET.FORMULA.WEIGHT
2982 ! Create a species mixture
2983
2984 IF (BACKGROUND) THEN
2985 NN = 1
2986 ELSE
2987 IF (N==1) N = N + 1
2988 NN = N
2989 ENDIF
2990
2991 CONVERSION = 0._EB
2992
2993 SM => SPECIES.MIXTURE(NN)
2994
2995 IF (SPEC.ID(1)=='null') THEN
2996 SPEC.ID(1) = ID
2997 VOLUME.FRACTION(1) = 1.0._EB
2998 ELSE
2999 SM%K.USER = CONDUCTIVITY
3000 SM%D.USER = DIFFUSIVITY
3001 SM%EPSK = EPSILONKIJ
3002 SM%FIC.CONCENTRATION = FIC.CONCENTRATION
3003 SM%FLD.LETHAL.DOSE = FLD.LETHAL.DOSE
3004 SM%MU.USER = VISCOSITY
3005 SM%PR.USER = PR.GAS
3006 SM%RAMP.CP = RAMP.CP
3007 SM%RAMP.D = RAMP.D
3008 SM%RAMP.G.F = RAMP.G.F
3009 SM%RAMP.K = RAMP.K
3010 SM%RAMP.MU = RAMP.MU
3011 IF (REFERENCE.TEMPERATURE < -TMM) REFERENCE.TEMPERATURE = 25._EB
3012 SM%REFERENCE.TEMPERATURE = REFERENCE.TEMPERATURE + TMM
3013 SM%SIG = SIGMALJ
3014 SM%SPECIFIC.HEAT = SPECIFIC.HEAT*1000._EB
3015 SM%REFERENCE.ENTHALPY = REFERENCE.ENTHALPY*1000._EB
3016 SM%H.F = ENTHALPY.OF.FORMATION*1000._EB
3017
3018 IF (SM%RAMP.CP=='null' .AND. SM%REFERENCE.ENTHALPY < -1.E20._EB) SM%REFERENCE.ENTHALPY = 0._EB

```

```

3019
3020 ! Get ramps
3021 IF (SM%RAMP.CP/= 'null') THEN
3022 CALL GET_RAMP_INDEX(SM%RAMP.CP, 'TEMPERATURE', NR)
3023 SM%RAMP.CP.INDEX = NR
3024 ENDIF
3025 IF (SM%RAMP.D/= 'null') THEN
3026 CALL GET_RAMP_INDEX(SM%RAMP.D, 'TEMPERATURE', NR)
3027 SM%RAMP.D.INDEX = NR
3028 ENDIF
3029 IF (SM%RAMP.G.F/= 'null') THEN
3030 CALL GET_RAMP_INDEX(SM%RAMP.D, 'TEMPERATURE', NR)
3031 SM%RAMP.G.F.INDEX = NR
3032 ENDIF
3033 IF (SM%RAMP.K/= 'null') THEN
3034 CALL GET_RAMP_INDEX(SM%RAMP.K, 'TEMPERATURE', NR)
3035 SM%RAMP.K.INDEX = NR
3036 ENDIF
3037 IF (SM%RAMP.MU/= 'null') THEN
3038 CALL GET_RAMP_INDEX(SM%RAMP.MU, 'TEMPERATURE', NR)
3039 SM%RAMP.MU.INDEX = NR
3040 ENDIF
3041 ENDIF
3042
3043 SM%ID = ID
3044 SM%ZZO = MAX(0. .EB, MASS.FRACTION.0)
3045
3046 ! Count the number of species included in the mixture
3047
3048 N.SUB.SPECIES = 0
3049 COUNT.SPEC: DO NS=1, N.SPECIES
3050 IF (TRIM(SPEC.ID(NS)) /= 'null') THEN
3051 N.SUB.SPECIES = N.SUB.SPECIES + 1
3052 ELSE
3053 EXIT
3054 ENDIF
3055 ENDDO COUNT.SPEC
3056
3057 IF (N.SUB.SPECIES == 1) THEN
3058 MASS.FRACTION=0. .EB
3059 MASS.FRACTION(1)=1. .EB
3060 VOLUME.FRACTION=0. .EB
3061 ENDIF
3062
3063 ! Allocate arrays to store the species id, mass, volume fractions
3064
3065 ALLOCATE (SM%SPEC.ID(N.SPECIES), STAT=IZERO)
3066 ALLOCATE (SM%VOLUME.FRACTION(N.SPECIES), STAT=IZERO)
3067 ALLOCATE (SM%MASS.FRACTION(N.SPECIES), STAT=IZERO)
3068
3069 SM%SPEC.ID = 'null'
3070 SM%VOLUME.FRACTION = 0. .EB
3071 SM%MASS.FRACTION = 0. .EB
3072 Y.INDEX = -1
3073 DO NS = 1, N.SUB.SPECIES
3074 FIND_SPEC.ID: DO NS2 = 1, N.SPECIES
3075 IF ((.NOT. NEW_PRIMITIVE(N.SPEC.READ) .AND. TRIM(SPECIES(NS2)%ID) == TRIM(SPEC.ID(NS))) .OR. &
3076 ( NEW_PRIMITIVE(N.SPEC.READ) .AND. TRIM(SPECIES(NS2)%ID) == TRIM(ID))) THEN
3077 SM%SPEC.ID(NS2) = SPECIES(NS2)%ID
3078 Y.INDEX(NS) = NS2
3079 IF (N.SUB.SPECIES==1) THEN
3080 SM%FORMULA = SPECIES(NS2)%FORMULA
3081 SM%SINGLE.SPEC.INDEX=NS2
3082 ENDIF
3083 IF (SPECIES(NS2)%MODE == AEROSOL.SPECIES) THEN
3084 IF (N.SUB.SPECIES == 1) THEN
3085 SM%DEPOSITING = .TRUE.
3086 IF (ABS(MEAN.DIAMETER-1.E-6.EB)<=TWO.EPSILON.EB) THEN
3087 SM%MEAN.DIAMETER = SPECIES(NS2)%MEAN.DIAMETER
3088 ELSE
3089 SM%MEAN.DIAMETER = MEAN.DIAMETER
3090 ENDIF
3091 IF (ABS(DENSITY.SOLID-1800. .EB) <=TWO.EPSILON.EB .AND. &
3092 ABS(DENSITY.SOLID-SPECIES(NS2)%DENSITY.SOLID) <=TWO.EPSILON.EB) THEN
3093 SM%DENSITY.SOLID = DENSITY.SOLID
3094 ELSE
3095 SM%DENSITY.SOLID = SPECIES(NS2)%DENSITY.SOLID
3096 ENDIF
3097 SM%CONDUCTIVITY.SOLID=SPECIES(NS2)%CONDUCTIVITY.SOLID
3098 ELSE
3099 WRITE(MESSAGE, '(A,A,A)') 'WARNING: Cannot do deposition with a lumped species. Species ', TRIM(SM%ID), &
3100 ' will not have deposition'
3101 IF (MYID=0) WRITE(LU.ERR, '(A)') TRIM(MESSAGE)
3102 ENDIF
3103 ENDIF
3104 EXIT FIND_SPEC.ID
3105 ENDIF
3106 ENDDO FIND_SPEC.ID

```



Source Code files for edited portions of FDS

```

3107 IF (Y_INDEX(NS)<0) THEN
3108 WRITE(MESSAGE,'(A,A,A,I0,A)') 'ERROR: SPEC ',TRIM(SM%D),', sub species ',NS,' not found.'
3109 CALL SHUTDOWN(MESSAGE) ; RETURN
3110 ENDF
3111 IF (MASS.FRACTION(NS)>0._EB) CONVERSION = CONVERSION + MASS.FRACTION(NS) / SPECIES(Y_INDEX(NS))%MW
3112 IF (VOLUME.FRACTION(NS)>0._EB) CONVERSION = CONVERSION + VOLUME.FRACTION(NS) * SPECIES(Y_INDEX(NS))%MW
3113 IF (.NOT. PREDEFINED.SMIX(NN) .AND. MASS.FRACTION(NS)<=0._EB .AND. VOLUME.FRACTION(NS)<=0._EB) THEN
3114 WRITE(MESSAGE,'(A,A,A,I0,A)') 'ERROR: SPEC ',TRIM(SM%D),', mass or volume fraction for sub species ',NS,' not
found.'
3115 CALL SHUTDOWN(MESSAGE) ; RETURN
3116 ENDF
3117
3118 ENDDO
3119
3120 IF (ANY(MASS.FRACTION>0._EB)) THEN
3121 DO NS = 1,N.SUB.SPECIES
3122 SM%VOLUME.FRACTION(Y_INDEX(NS)) = MASS.FRACTION(NS) / SPECIES(Y_INDEX(NS))%MW / CONVERSION
3123 SM%MASS.FRACTION(Y_INDEX(NS)) = MASS.FRACTION(NS)
3124 ENDDO
3125 ENDF
3126
3127 IF (ANY(VOLUME.FRACTION>0._EB)) THEN
3128 DO NS = 1,N.SUB.SPECIES
3129 SM%MASS.FRACTION(Y_INDEX(NS)) = VOLUME.FRACTION(NS) * SPECIES(Y_INDEX(NS))%MW / CONVERSION
3130 SM%VOLUME.FRACTION(Y_INDEX(NS)) = VOLUME.FRACTION(NS)
3131 ENDDO
3132 ENDF
3133
3134 ! Normalize mass and volume fractions , plus stoichiometric coefficient
3135
3136 SM%MASS.FRACTION = SM%MASS.FRACTION / SUM(SM%MASS.FRACTION)
3137 IF (.NOT. SIMPLE.CHEMISTRY) SM%ADJUST_NU = SUM(SM%VOLUME.FRACTION)
3138 SM%VOLUME.FRACTION = SM%VOLUME.FRACTION / SUM(SM%VOLUME.FRACTION)
3139
3140 ! Calculate the molecular weight and extinction coefficient
3141
3142 SM%MW = 0._EB
3143 SM%MASS.EXTINGUCTION_COEFFICIENT = 0._EB
3144 DO NS = 1,N.SPECIES
3145 IF (SM%MASS.FRACTION(NS) <TWO.EPSILON.EB)
3146 IF (MASS.EXTINGUCTION_COEFFICIENT > 0._EB) THEN
3147 SM%MASS.EXTINGUCTION_COEFFICIENT = MASS.EXTINGUCTION_COEFFICIENT
3148 ELSE
3149 SM%MASS.EXTINGUCTION_COEFFICIENT = SM%MASS.EXTINGUCTION_COEFFICIENT+SM%MASS.FRACTION(NS)*SPECIES(NS)%
MASS.EXTINGUCTION_COEFFICIENT
3150 ENDF
3151 IF (MW > 0._EB) THEN
3152 SM%MW = MW
3153 ELSE
3154 SM%MW = SM%MW + SM%VOLUME.FRACTION(NS) * SPECIES(NS)%MW !! *SM%ADJUST_NU ::term for potential non-normalized
inputs
3155 ENDF
3156 IF (SPECIES(NS)%FORMULA(1:5)='SPEC.') SM%VALID_ATOMS = .FALSE.
3157 IF (FORMULA /= 'null' .AND. .NOT. PREDEFINED.SMIX(NN)) THEN
3158 CALL GET.FORMULA.WEIGHT(FORMULA,SM%MW,SM%ATOMS)
3159 ELSE
3160 SM%ATOMS = SM%ATOMS + SM%VOLUME.FRACTION(NS)*SPECIES(NS)%ATOMS !! *SM%ADJUST_NU ::term for potential non-
normalized inputs
3161 ENDF
3162 ENDDO
3163
3164 SM%RCON = R0/SM%MW
3165
3166 IF (SM%H.F > -1.E23.EB) SM%H.F = SM%H.F/SM%MW*1000._EB !!J/mol -> J/kg
3167 IF (SM%SPECIFIC.HEAT > 0._EB) THEN
3168 IF (SM%H.F > -1.E23.EB) THEN
3169 SM%REFERENCE.ENTHALPY = SM%H.F - SM%SPECIFIC.HEAT*H.F.REFERENCE.TEMPERATURE
3170 ELSE
3171 IF (SM%REFERENCE.ENTHALPY < -1.E20.EB) SM%REFERENCE.ENTHALPY = SM%SPECIFIC.HEAT * SM%REFERENCE.TEMPERATURE
3172 SM%H.F = SM%REFERENCE.ENTHALPY + SM%SPECIFIC.HEAT * (H.F.REFERENCE.TEMPERATURE - SM%REFERENCE.TEMPERATURE)
3173 ENDF
3174 ENDF
3175
3176 END SUBROUTINE DEFINE.MIXTURE
3177
3178 SUBROUTINE SET.SPEC.DEFAULT
3179
3180 AEROSOL = .FALSE.
3181 BACKGROUND = .FALSE.
3182 BETA.LIQUID = -1._EB
3183 CONDUCTIVITY = -1._EB
3184 CONDUCTIVITY.LIQUID = -1._EB
3185 CONDUCTIVITY.SOLID = 0.26._EB !W/m/K Ben-Dor, et al. 2002. (~10 x air)
3186 COPY.LUMPED = .FALSE.
3187 DENSITY.SOLID = 1800._EB !kg/m^3 Slowik, et al. 2004
3188 DIFFUSIVITY = -1._EB
3189 EPSILON.KJ = 0._EB
3190 HC.CONCENTRATION = 0._EB

```

```

3191 FLD.LETHAL.DOSE      = 0._EB
3192 FORMULA              = 'null'
3193 FYI                  = 'null'
3194 ENTHALPY.OF.FORMATION = -1.E30._EB ! J/mol
3195 ID                   = 'null'
3196 LUMPED.COMPONENT.ONLY = .FALSE.
3197 RADCAL.ID            = 'null'
3198 MASS.EXTINCTION.COEFFICIENT = -1._EB ! m2/kg
3199 MASS.FRACTION        = 0._EB
3200 MASS.FRACTION.0      = -1._EB
3201 MEAN.DIAMETER        = 1.E-6._EB
3202 MW                   = 0._EB
3203 PR.GAS               = -1._EB
3204 PRIMITIVE            = .FALSE.
3205 REFERENCE.TEMPERATURE = -300._EB ! C
3206 SIGMALJ              = 0._EB
3207 SPEC.ID              = 'null'
3208 SPECIFIC.HEAT        = -1._EB
3209 REFERENCE.ENTHALPY   = -2.E20._EB
3210 VISCOSITY            = -1._EB
3211 VISCOSITY.LIQUID     = -1._EB
3212 VOLUME.FRACTION     = 0._EB
3213
3214 DENSITY.LIQUID       = -1._EB
3215 HEAT.OF.VAPORIZATION = -1._EB ! kJ/kg
3216 H.V.REFERENCE.TEMPERATURE = -300._EB
3217 MELTING.TEMPERATURE = -300._EB ! C
3218 SPECIFIC.HEAT.LIQUID = -1._EB ! kJ/kg-K
3219 VAPORIZATION.TEMPERATURE = -300._EB ! C
3220
3221 RAMP.CP               = 'null'
3222 RAMP.CP.L            = 'null'
3223 RAMP.D               = 'null'
3224 RAMP.G.F             = 'null'
3225 RAMP.K               = 'null'
3226 RAMP.MU              = 'null'
3227
3228 N.BINS               = -1
3229 MIN.DIAMETER         = -1._EB !um
3230 MAX.DIAMETER         = -1._EB !um
3231
3232 END SUBROUTINE SET.SPEC.DEFAULT
3233
3234 SUBROUTINE SETUP.PREDEFINED.SMIX(N)
3235
3236 ! Set up the SMIX line either for the SIMPLE.CHEMISTRY mode or for a primitive species
3237
3238 INTEGER, INTENT(IN) :: N
3239 TYPE(REACTION_TYPE), POINTER :: RN
3240
3241 MASS.FRACTION = 0._EB
3242
3243 SELECT CASE(N)
3244 CASE(1)
3245 ID = 'AIR'
3246 FORMULA = 'Z0'
3247 SPEC.ID(1) = 'WATER VAPOR'
3248 SPEC.ID(2) = 'OXYGEN'
3249 SPEC.ID(3) = 'CARBON DIOXIDE'
3250 SPEC.ID(4) = 'NITROGEN'
3251 MASS.FRACTION(1) = Y.H2O.INFTY
3252 MASS.FRACTION(2) = Y.O2.INFTY*(1._EB-Y.H2O.INFTY)
3253 MASS.FRACTION(3) = Y.CO2.INFTY*(1._EB-Y.H2O.INFTY)
3254 MASS.FRACTION(4) = 1._EB-SUM(MASS.FRACTION)
3255 CASE(2)
3256 RN => REACTION(1)
3257 ID = RN%FUEL
3258 FORMULA = 'Z1'
3259 SPEC.ID(1) = RN%FUEL
3260 MASS.FRACTION(1) = 1._EB
3261 CASE(3)
3262 RN => REACTION(1)
3263 ID = 'PRODUCTS'
3264 FORMULA = 'Z2'
3265 SPEC.ID(1) = 'CARBON MONOXIDE'
3266 SPEC.ID(2) = 'SOOT'
3267 SPEC.ID(3) = 'WATER VAPOR'
3268 SPEC.ID(4) = 'CARBON DIOXIDE'
3269 SPEC.ID(5) = 'NITROGEN'
3270 RN%NU.CO = (SPECIES.MIXTURE(FUEL.SMIX.INDEX)%MW/MW.CO) *RN%CO.YIELD
3271 RN%NU.SOOT = (SPECIES.MIXTURE(FUEL.SMIX.INDEX)%MW/RN%MW.SOOT)*RN%SOOT.YIELD
3272 RN%NU.H2O = 0.5._EB*RN%H - 0.5._EB*RN%NU.SOOT*RN%SOOT.H.FRACTION
3273 RN%NU.CO2 = RN%C - RN%NU.CO - RN%NU.SOOT*(1._EB-RN%SOOT.H.FRACTION)
3274 IF (RN%NU.CO2 < 0._EB) THEN
3275 WRITE(MESSAGE,'(A)') 'ERROR: REAC, Not enough carbon in the fuel for the specified CO.YIELD and/or SOOT.YIELD'
3276 CALL SHUTDOWN(MESSAGE) ; RETURN
3277 ENDF
3278 RN%NU.O2 = RN%NU.CO2 + 0.5._EB*(RN%NU.CO+RN%NU.H2O-RN%O)

```

```

3279 RN%NU.N2 = RN%N*0.5_EB
3280 VOLUME.FRACTION(1) = RN%NU.CO
3281 VOLUME.FRACTION(2) = RN%NU.SOOT
3282 VOLUME.FRACTION(3) = RN%NU.H2O + SPECIES_MIXTURE(1)%VOLUME.FRACTION(H2O.INDEX)*RN%NU.O2 / &
3283 SPECIES_MIXTURE(1)%VOLUME.FRACTION(O2.INDEX)
3284 VOLUME.FRACTION(4) = RN%NU.CO2 + SPECIES_MIXTURE(1)%VOLUME.FRACTION(CO2.INDEX)*RN%NU.O2 / &
3285 SPECIES_MIXTURE(1)%VOLUME.FRACTION(O2.INDEX)
3286 VOLUME.FRACTION(5) = RN%NU.N2 + SPECIES_MIXTURE(1)%VOLUME.FRACTION(N2.INDEX)*RN%NU.O2 / &
3287 SPECIES_MIXTURE(1)%VOLUME.FRACTION(O2.INDEX)
3288 VOLUME.FRACTION = VOLUME.FRACTION/SUM(VOLUME.FRACTION)
3289 SPECIES(SOOT.INDEX)%ATOMS=0._EB
3290 SPECIES(SOOT.INDEX)%ATOMS(1)=RN%SOOT.H.FRACTION
3291 SPECIES(SOOT.INDEX)%ATOMS(6)=1._EB-RN%SOOT.H.FRACTION
3292 END SELECT
3293
3294 END SUBROUTINE SETUP_PREDEFINED_SMIX
3295
3296 END SUBROUTINE READ_SPEC
3297
3298
3299 SUBROUTINE PROC_SMIX
3300
3301 ! Create the Z to Y transformation matrix and fill up the gas property tables
3302
3303 USE PHYSICAL_FUNCTIONS, ONLY: GET_SPECIFIC_GAS_CONSTANT
3304 USE MATH_FUNCTIONS, ONLY: EVALUATE_RAMP
3305 USE PROPERTY_DATA, ONLY: JANAF_TABLE, CALC_GAS_PROPS, GAS_PROPS, CALC_MIX_PROPS
3306 REAL(EB), ALLOCATABLE, DIMENSION(:) :: MU_TMP, CP_TMP, K_TMP, H_TMP, D_TMP, G.F_TMP, ZZ_GET, &
3307 MU_TMP_Z, CP_TMP_Z, K_TMP_Z, H_TMP_Z, D_TMP_Z, G.F_TMP_Z, RSQ_MW_Y
3308 REAL(EB) :: CP1, CP2, H1, H2, H_REF_SENSIBLE(1:N_TRACKED_SPECIES), REF_TEMP
3309 INTEGER :: NN, N
3310 TYPE(SPECIES_TYPE), POINTER :: SS=>NULL()
3311 TYPE(SPECIES_MIXTURE_TYPE), POINTER :: SM=>NULL()
3312
3313 ! Setup the array to convert the tracked species array to array of all primitive species
3314
3315 ALLOCATE(ZZY(N_SPECIES, N_TRACKED_SPECIES), STAT=IZERO)
3316 CALL ChkMemErr('READ', 'ZZY', IZERO)
3317 ZZY = 0._EB
3318
3319 DO N=1, N_TRACKED_SPECIES
3320 SM => SPECIES_MIXTURE(N)
3321 DO NN=1, N_SPECIES
3322 ZZY(NN, N) = SM%MASS_FRACTION(NN)
3323 ENDDO
3324 ENDDO
3325
3326 ALLOCATE(RSQ_MW_Y(N_SPECIES), STAT=IZERO)
3327 CALL ChkMemErr('READ', 'RSQ_MW_Y', IZERO)
3328
3329 RSQ_MW_Y = 1._EB/SQRT(SPECIES%MW)
3330
3331 ! Set up the arrays of molecular weights
3332
3333 ALLOCATE(MWR_Z(N_TRACKED_SPECIES), STAT=IZERO)
3334 CALL ChkMemErr('READ', 'MWR_Z', IZERO)
3335
3336 ALLOCATE(RSQ_MW_Z(N_TRACKED_SPECIES), STAT=IZERO)
3337 CALL ChkMemErr('READ', 'RSQ_MW_Z', IZERO)
3338
3339 MWR_Z = 1._EB/SPECIES_MIXTURE%MW
3340 RSQ_MW_Z = 1._EB/SQRT(SPECIES_MIXTURE%MW)
3341
3342 ALLOCATE(ZZ_GET(N_TRACKED_SPECIES))
3343 ZZ_GET = SPECIES_MIXTURE%ZZO
3344 CALL GET_SPECIFIC_GAS_CONSTANT(ZZ_GET, RSUM0)
3345 DEALLOCATE(ZZ_GET)
3346
3347 MW_MIN = MINVAL(SPECIES_MIXTURE(1:N_TRACKED_SPECIES)%MW)
3348 MW_MAX = MAXVAL(SPECIES_MIXTURE(1:N_TRACKED_SPECIES)%MW)
3349
3350 ! Compute background density from other background quantities
3351
3352 RHOA = P_INF / (TMPA * RSUM0)
3353
3354 ! Compute constant-temperature specific heats
3355
3356 GMICG = (GAMMA - 1._EB) / GAMMA
3357 CP_GAMMA = SPECIES_MIXTURE(1)%RCON / GMICG
3358 CPOPR = CP_GAMMA / PR
3359
3360 ! Compute gas properties for primitive species 1 to N_SPECIES.
3361
3362 ALLOCATE(D_TMP(N_SPECIES))
3363 D_TMP = 0._EB
3364 ALLOCATE(MU_TMP(N_SPECIES))
3365 MU_TMP = 0._EB
3366 ALLOCATE(CP_TMP(N_SPECIES))

```

```

3367 CP.TMP = 0._EB
3368 ALLOCATE(H.TMP(N_SPECIES))
3369 H.TMP = 0._EB
3370 ALLOCATE(K.TMP(N_SPECIES))
3371 K.TMP = 0._EB
3372 ALLOCATE(G.F.TMP(N_SPECIES))
3373 G.F.TMP = 0._EB
3374
3375 ALLOCATE(D.TMP.Z(N_TOTAL_SCALARS))
3376 D.TMP.Z = -1.E30_EB
3377 ALLOCATE(MU.TMP.Z(N_TOTAL_SCALARS))
3378 MU.TMP.Z = -1.E30_EB
3379 ALLOCATE(CP.TMP.Z(N_TOTAL_SCALARS))
3380 CP.TMP.Z = -1.E30_EB
3381 ALLOCATE(H.TMP.Z(N_TOTAL_SCALARS))
3382 H.TMP.Z = -1.E30_EB
3383 ALLOCATE(K.TMP.Z(N_TOTAL_SCALARS))
3384 K.TMP.Z = -1.E30_EB
3385 ALLOCATE(G.F.TMP.Z(N_TOTAL_SCALARS))
3386 G.F.TMP.Z = -1.E30_EB
3387
3388 ALLOCATE(CPBAR.Z(0:5000,N_TOTAL_SCALARS))
3389 CALL ChkMemErr('READ','CPBAR.Z',IZERO)
3390 CPBAR.Z = 0._EB
3391
3392 !ALLOCATE(CP.AVG.Z(0:5000,N_TOTAL_SCALARS))
3393 !CALL ChkMemErr('READ','CP.AVG.Z',IZERO)
3394 !CP.AVG.Z = 0._EB
3395
3396 ALLOCATE(H.SENS.Z(0:5000,N_TOTAL_SCALARS))
3397 CALL ChkMemErr('READ','H.SENS.Z',IZERO)
3398 H.SENS.Z = 0._EB
3399
3400 ALLOCATE(K.RSQMW.Z(0:5000,N_TOTAL_SCALARS))
3401 CALL ChkMemErr('READ','K.RSQMW.Z',IZERO)
3402 K.RSQMW.Z = 0._EB
3403
3404 ALLOCATE(MU.RSQMW.Z(0:5000,N_TOTAL_SCALARS))
3405 CALL ChkMemErr('READ','MU.RSQMW.Z',IZERO)
3406 MU.RSQMW.Z = 0._EB
3407
3408 ALLOCATE(CP.Z(0:5000,N_TOTAL_SCALARS))
3409 CALL ChkMemErr('READ','CP.Z',IZERO)
3410 CP.Z = 0._EB
3411
3412 ALLOCATE(D.Z(0:5000,N_TOTAL_SCALARS))
3413 CALL ChkMemErr('READ','D.Z',IZERO)
3414 D.Z = 0._EB
3415
3416 ALLOCATE(G.F.Z(0:5000,N_TOTAL_SCALARS))
3417 CALL ChkMemErr('READ','G.F.Z',IZERO)
3418 G.F.Z = 0._EB
3419
3420 ! Adjust reference enthalpy to 0 K if a RAMP_CP is given
3421 DO N=1,N_SPECIES
3422 SS => SPECIES(N)
3423 IF (SS%RAMP_CP_INDEX > 0) THEN
3424 IF (SS%HF > -1.E20_EB) THEN
3425 CP2 = EVALUATE_RAMP(1._EB,1._EB,SS%RAMP_CP_INDEX)*1000._EB
3426 H2 = 0._EB
3427 DO J=1,INT(HF.REFERENCE_TEMPERATURE)+1
3428 H1 = H2
3429 CP1 = CP2
3430 CP2 = EVALUATE_RAMP(REAL(J,EB),1._EB,SS%RAMP_CP_INDEX)*1000._EB
3431 H2 = H2 + 0.5_EB*(CP1+CP2)
3432 ENDDO
3433 SS%REFERENCE_ENTHALPY = SS%HF/SS%M*1000._EB - &
3434 (H1 + (H2-H1)*(HF.REFERENCE_TEMPERATURE-INT(HF.REFERENCE_TEMPERATURE)))
3435 ELSE
3436 IF (SS%REFERENCE_TEMPERATURE<=TWO_EPSILON_EB) CYCLE
3437 CP2 = EVALUATE_RAMP(1._EB,1._EB,SS%RAMP_CP_INDEX)*1000._EB
3438 H2 = 0._EB
3439 DO J=1,INT(SS%REFERENCE_TEMPERATURE)+1
3440 H1 = H2
3441 CP1 = CP2
3442 CP2 = EVALUATE_RAMP(REAL(J,EB),1._EB,SS%RAMP_CP_INDEX)*1000._EB
3443 H2 = H2 + 0.5_EB*(CP1+CP2)
3444 ENDDO
3445 SS%REFERENCE_ENTHALPY = SS%REFERENCE_ENTHALPY - &
3446 (H1 + (H2-H1)*(SS%REFERENCE_TEMPERATURE-INT(SS%REFERENCE_TEMPERATURE)))
3447 IF (SS%HF <= -1.E20) THEN
3448 CP2 = EVALUATE_RAMP(1._EB,1._EB,SS%RAMP_CP_INDEX)*1000._EB
3449 H2 = SS%REFERENCE_ENTHALPY
3450 DO J=1,INT(HF.REFERENCE_TEMPERATURE)+1
3451 H1 = H2
3452 CP1 = CP2
3453 CP2 = EVALUATE_RAMP(REAL(J,EB),1._EB,SS%RAMP_CP_INDEX)*1000._EB
3454 H2 = H2 + 0.5_EB*(CP1+CP2)

```

```

3455 ENDDO
3456 SS%H.F = (H1 + (H2-H1)*(H.F.REFERENCE.TEMPERATURE-INT(H.F.REFERENCE.TEMPERATURE)))*SS%MW*0.001_EB
3457 ENDIF
3458 ENDIF
3459 ENDIF
3460 END DO
3461
3462 DO N=1,N_TRACKED.SPECIES
3463 SM => SPECIES.MIXTURE(N)
3464 IF (SM%RAMP.CP.INDEX > 0) THEN
3465 IF (SM%H.F > -1.E20_EB) THEN
3466 CP2 = EVALUATE_RAMP(1._EB,1._EB,SM%RAMP.CP.INDEX)*1000._EB
3467 H2 = 0._EB
3468 DO J=1,INT(H.F.REFERENCE.TEMPERATURE)+1
3469 H1 = H2
3470 CP1 = CP2
3471 CP2 = EVALUATE_RAMP(REAL(J,EB),1._EB,SM%RAMP.CP.INDEX)*1000._EB
3472 H2 = H2 + 0.5_EB*(CP1+CP2)
3473 ENDDO
3474 SM%REFERENCE.ENTHALPY = SM%H.F - &
3475 (H1 + (H2-H1)*(H.F.REFERENCE.TEMPERATURE-INT(H.F.REFERENCE.TEMPERATURE)))
3476 ELSE
3477 IF (SM%REFERENCE.TEMPERATURE<=TWO.EPSILON_EB) CYCLE
3478 CP2 = EVALUATE_RAMP(1._EB,1._EB,SM%RAMP.CP.INDEX)*1000._EB
3479 H2 = 0._EB
3480 DO J=1,INT(SM%REFERENCE.TEMPERATURE)+1
3481 H1 = H2
3482 CP1 = CP2
3483 CP2 = EVALUATE_RAMP(REAL(J,EB),1._EB,SM%RAMP.CP.INDEX)*1000._EB
3484 H2 = H2 + 0.5_EB*(CP1+CP2)
3485 ENDDO
3486 SM%REFERENCE.ENTHALPY = SM%REFERENCE.ENTHALPY - &
3487 (H1 + (H2-H1)*(SM%REFERENCE.TEMPERATURE-INT(SM%REFERENCE.TEMPERATURE)))
3488 IF (SM%H.F <= -1.E20) THEN
3489 CP2 = EVALUATE_RAMP(1._EB,1._EB,SM%RAMP.CP.INDEX)*1000._EB
3490 H2 = SM%REFERENCE.ENTHALPY
3491 DO J=1,INT(H.F.REFERENCE.TEMPERATURE)+1
3492 H1 = H2
3493 CP1 = CP2
3494 CP2 = EVALUATE_RAMP(REAL(J,EB),1._EB,SM%RAMP.CP.INDEX)*1000._EB
3495 H2 = H2 + 0.5_EB*(CP1+CP2)
3496 ENDDO
3497 SM%H.F = H1 + (H2-H1)*(H.F.REFERENCE.TEMPERATURE-INT(H.F.REFERENCE.TEMPERATURE))
3498 ENDIF
3499 ENDIF
3500 ENDIF
3501 IF (SM%G.H.F <= -1.E20_EB) THEN
3502 SM%H.F = 0._EB
3503 DO J=1,N_SPECIES
3504 SM%H.F = SM%H.F + SM%VOLUME.FRACTION(J) * SPECIES(J)%H.F ! Calculate H.F of mixtures
3505 ENDDO
3506 SM%H.F = SM%H.F/SM%MW*1000._EB
3507 ENDIF
3508 END DO
3509
3510 ! Loop through temperatures from 1 K to 5000 K to get temperature-specific gas properties. Data from JANAF 4
3511
3512 TABLE_LOOP: DO J=1,5000
3513
3514 ! For each primitive species, get its property values at temperature J
3515
3516 DO N=1,N_SPECIES
3517 SS => SPECIES(N)
3518 CALL CALC_GAS_PROPS(J,N,D.TMP(N),MU.TMP(N),K.TMP(N),CP.TMP(N),H.TMP(N),SS%SFUEL,G.F.TMP(N))
3519 IF (SS%RAMP.CP.INDEX>0) THEN
3520 CP.TMP(N) = EVALUATE_RAMP(REAL(J,EB),0._EB,SS%RAMP.CP.INDEX)*1000._EB
3521 H.TMP(N) = SS%REFERENCE.ENTHALPY
3522 ENDF
3523 IF (SS%RAMP.D.INDEX>0) D.TMP(N) = EVALUATE_RAMP(REAL(J,EB),1._EB,SS%RAMP.D.INDEX)
3524 IF (SS%RAMP.G.F.INDEX>0) G.F.TMP(N) = EVALUATE_RAMP(REAL(J,EB),1._EB,SS%RAMP.G.F.INDEX)
3525 IF (SS%RAMP.K.INDEX>0) K.TMP(N) = EVALUATE_RAMP(REAL(J,EB),1._EB,SS%RAMP.K.INDEX)/SQRT(SS%MW)
3526 IF (SS%RAMP.MU.INDEX>0) MU.TMP(N) = EVALUATE_RAMP(REAL(J,EB),1._EB,SS%RAMP.MU.INDEX)/SQRT(SS%MW)
3527 ENDDO
3528
3529 DO N=1,N_TRACKED.SPECIES
3530 SM => SPECIES.MIXTURE(N)
3531 CALL CALC_MIX_PROPS(J,D.TMP.Z(N),MU.TMP.Z(N),K.TMP.Z(N),CP.TMP.Z(N),H.TMP.Z(N),SM%EPSK,SM%SIG,SM%D.USER,&
3532 SM%MU.USER,SM%K.USER,SM%MW,SM%SPECIFIC.HEAT,SM%REFERENCE.ENTHALPY,SM%REFERENCE.TEMPERATURE,SM%PR.USER)
3533 IF (SM%RAMP.CP.INDEX>0) THEN
3534 CP.TMP.Z(N) = EVALUATE_RAMP(REAL(J,EB),0._EB,SM%RAMP.CP.INDEX)*1000._EB
3535 H.TMP.Z(N) = SM%REFERENCE.ENTHALPY
3536 ENDF
3537 IF (SM%RAMP.D.INDEX>0) D.TMP.Z(N) = EVALUATE_RAMP(REAL(J,EB),1._EB,SM%RAMP.D.INDEX)
3538 IF (SM%RAMP.G.F.INDEX>0) G.F.TMP.Z(N) = EVALUATE_RAMP(REAL(J,EB),1._EB,SM%RAMP.G.F.INDEX)
3539 IF (SM%RAMP.K.INDEX>0) K.TMP.Z(N) = EVALUATE_RAMP(REAL(J,EB),1._EB,SM%RAMP.K.INDEX)*RSQ.MW.Z(N)
3540 IF (SM%RAMP.MU.INDEX>0) MU.TMP.Z(N) = EVALUATE_RAMP(REAL(J,EB),1._EB,SM%RAMP.MU.INDEX)*RSQ.MW.Z(N)
3541 ENDDO
3542

```

Source Code files for edited portions of FDS

```

3543 ! For each tracked species , store the mass-weighted property values
3544
3545 DO N=1,N_TRACKED_SPECIES
3546 IF (SPECIES_MIXTURE(N)%REFERENCE_ENTHALPY < -1.E20.EB) SPECIES_MIXTURE(N)%REFERENCE_ENTHALPY = SUM(ZZ(:,N) *
      H_TMP(:))
3547 IF (D_TMP_Z(N) > 0.EB) THEN
3548 D_Z(J,N) = D_TMP_Z(N)
3549 ELSE
3550 D_Z(J,N) = SPECIES_MIXTURE(N)%MW*SUM(ZZ(:,N)*D_TMP(:))/SPECIES(:)%MW
3551 ENDIF
3552 IF (CP_TMP_Z(N) > 0.EB) THEN
3553 CP_Z(J,N) = CP_TMP_Z(N)
3554 IF (J==1) CP_Z(0,N) = CP_Z(1,N)
3555 H_SENS_Z(J,N) = H_SENS_Z(J-1,N) + 0.5.EB*(CP_Z(J,N)+CP_Z(J-1,N))
3556 IF (J > 1) THEN
3557 CPBAR_Z(J,N) = (CPBAR_Z(J-1,N)*REAL(J-1,EB)+0.5.EB*(CP_Z(J,N)+CP_Z(J-1,N)))/REAL(J,EB)
3558 ELSE
3559 CPBAR_Z(0,N) = H_TMP_Z(N)
3560 CPBAR_Z(J,N) = CPBAR_Z(0,N) + CP_Z(J,N)
3561 ENDIF
3562 ELSE
3563 CP_Z(J,N) = SUM(ZZ(:,N) * CP_TMP(:))
3564 IF (J==1) CP_Z(0,N) = CP_Z(1,N)
3565 H_SENS_Z(J,N) = H_SENS_Z(J-1,N) + 0.5.EB*(CP_Z(J,N)+CP_Z(J-1,N))
3566 IF (J > 1) THEN
3567 CPBAR_Z(J,N) = (CPBAR_Z(J-1,N)*REAL(J-1,EB)+0.5.EB*(CP_Z(J,N)+CP_Z(J-1,N)))/REAL(J,EB)
3568 ELSE
3569 CPBAR_Z(0,N) = SUM(ZZ(:,N) * H_TMP(:))
3570 CPBAR_Z(J,N) = CPBAR_Z(0,N) + CP_Z(J,N)
3571 ENDIF
3572 ENDIF
3573 IF (MU_TMP_Z(N) > 0.EB) THEN
3574 MU_RSQM_WZ(J,N) = MU_TMP_Z(N)
3575 ELSE
3576 MU_RSQM_WZ(J,N) = SUM(ZZ(:,N) * MU_TMP(:)) / SUM(ZZ(:,N) * RSQ_MW_Y(:)) * RSQ_MW_Z(N)
3577 ENDIF
3578 IF (K_TMP_Z(N) > 0.EB) THEN
3579 K_RSQM_WZ(J,N) = K_TMP_Z(N)
3580 ELSE
3581 K_RSQM_WZ(J,N) = SUM(ZZ(:,N) * K_TMP(:)) / SUM(ZZ(:,N) * RSQ_MW_Y(:)) * RSQ_MW_Z(N)
3582 ENDIF
3583 IF (G_F_TMP_Z(N) > 0.EB) THEN
3584 G_F_Z(J,N) = G_F_TMP_Z(N)
3585 ELSE
3586 G_F_Z(J,N) = SUM(ZZ(:,N) * G_F_TMP(:))
3587 ENDIF
3588 ENDDO
3589 ENDDO TABLE_LOOP
3590
3591 ! Adjust H_SENS_Z to 0 at the H.F.REFERENCE_TEMPERATURE
3592 IF (CONSTANT_SPECIFIC_HEAT_RATIO) THEN
3593 REF_TEMP = 0.EB
3594 ELSE
3595 REF_TEMP = H.F.REFERENCE_TEMPERATURE
3596 ENDIF
3597 J = INT(REF_TEMP)
3598 H_REF_SENSIBLE(:) = H_SENS_Z(J,:)+(REF_TEMP-REAL(J,EB))*(H_SENS_Z(J+1,:)-H_SENS_Z(J,:))
3599 H_SENS_Z(0,:) = -H_REF_SENSIBLE(:)
3600 DO J = 1, 5000
3601 H_SENS_Z(J,:) = H_SENS_Z(J,:)-H_REF_SENSIBLE(:)
3602 ! CP_AVG_Z(J,:) = H_SENS_Z(J,:)/REAL(J,EB)
3603 ENDDO
3604
3605 DO N=1,N_TRACKED_SPECIES
3606 SM=>SPECIES_MIXTURE(N)
3607 IF (SM%HLF <=-1.E20.EB) THEN
3608 H1=CPBAR_Z(INT(H.F.REFERENCE_TEMPERATURE),N)*REAL(INT(H.F.REFERENCE_TEMPERATURE),EB)
3609 H2=CPBAR_Z(INT(H.F.REFERENCE_TEMPERATURE)+1,N)*REAL(INT(H.F.REFERENCE_TEMPERATURE)+1,EB)
3610 SM%HLF = H1+(H2-H1)*(H.F.REFERENCE_TEMPERATURE-REAL(INT(H.F.REFERENCE_TEMPERATURE),EB))
3611 ENDIF
3612 END DO
3613
3614 DEALLOCATE(RSQ_MW_Y)
3615
3616 DEALLOCATE(D_TMP)
3617 DEALLOCATE(MU_TMP)
3618 DEALLOCATE(CP_TMP)
3619 DEALLOCATE(H_TMP)
3620 DEALLOCATE(K_TMP)
3621 DEALLOCATE(G_F_TMP)
3622
3623 DEALLOCATE(D_TMP_Z)
3624 DEALLOCATE(MU_TMP_Z)
3625 DEALLOCATE(CP_TMP_Z)
3626 DEALLOCATE(H_TMP_Z)
3627 DEALLOCATE(K_TMP_Z)
3628
3629 END SUBROUTINE PROC_S MIX

```

```

3630
3631
3632 SUBROUTINE READREAC
3633
3634 USE PROPERTY_DATA, ONLY : ELEMENT,GET_FORMULA_WEIGHT,MAKE_PERIODIC_TABLE,SIMPLE_SPECIES_MW,GAS_PROPS,LOOKUP_CHLR
3635 USE MATH_FUNCTIONS, ONLY : GET_RAMP_INDEX,GET_TABLE_INDEX
3636 CHARACTER(LABEL_LENGTH) :: FUEL,RADCAL_ID='null',SPEC_ID_NU(MAX_SPECIES),SPEC_ID_N_S(MAX_SPECIES),RAMP_FS,&
    TABLE_FS,&
3637 RAMP_CHLR
3638 CHARACTER(LABEL_LENGTH) :: FORMULA
3639 CHARACTER(255) :: EQUATION
3640 CHARACTER(100) :: FWD_ID
3641 INTEGER :: NR,NS,NS2,NFR
3642 REAL(EB) :: SOOT_YIELD,CO_YIELD,EPUMO2,A,&
3643 CRITICAL_FLAME_TEMPERATURE,HEAT_OF_COMBUSTION,E,C,H,N,O,&
3644 AUTO_IGNITION_TEMPERATURE,SOOT_H_FRACTION,N_T,K,NU(MAX_SPECIES),N_S(MAX_SPECIES),&
3645 FLAME_SPEED,FLAME_SPEED_EXPONENT,FLAME_SPEED_TEMPERATURE,&
3646 TURBULENT_FLAME_SPEED_ALPHA,TURBULENT_FLAME_SPEED_EXPONENT,RADIATIVE_FRACTION !RADIATIVE_FRACTION_1 !Sesa-added
    RADIATIVE_FRACTION_1
3647 REAL(EB) :: E_TMP=0._EB,S_TMP=0._EB,ATOM_COUNTS(118),MW_FUEL=0._EB,H_F=0._EB,PR_TMP
3648 LOGICAL :: L_TMP,CHECK_ATOM_BALANCE,FAST_CHEMISTRY,REVERSE,THIRD_BODY,SERIES_REACTION !Sesa-added
    SERIES_REACTION as in 6.2.0
3649 NAMELIST /REAC/ A,AUTO_IGNITION_TEMPERATURE,C,CHECK_ATOM_BALANCE,CO_YIELD,CRITICAL_FLAME_TEMPERATURE,&
3650 E,EPUMO2,K,EQUATION,FIXED_MIX_TIME,FLAME_SPEED,FLAME_SPEED_EXPONENT,FLAME_SPEED_TEMPERATURE,FORMULA,FUEL,&
3651 FUEL_RADCAL_ID,FWD_ID,FYI,H,HEAT_OF_COMBUSTION,&
3652 ID,IDEAL,N,NU,N_S,N_T,O,ODE_SOLVER,RADIATIVE_FRACTION,RAMP_CHLR,RAMP_FS,REAC_ATOM_ERROR,&
3653 REAC_MASS_ERROR,REVERSE,SOOT_H_FRACTION,SOOT_YIELD,&
3654 SPEC_ID_N_S,SPEC_ID_NU,TABLE_FS,TAU_CHEM,TAU_FLAME,&
3655 THIRD_BODY,TURBULENT_FLAME_SPEED_ALPHA,TURBULENT_FLAME_SPEED_EXPONENT,Y_P_MIN_EDC,&
3656 HRRPUA_SHEET,HRRPUV_AVERAGE !RADIATIVE_FRACTION_1 !Sesa-added HRRPUA_SHEET and HRRPUV_AVERAGE
    RADIATIVE_FRACTION_1 as in 6.2.0
3657
3658 CALL MAKE_PERIODIC_TABLE
3659 CALL SIMPLE_SPECIES_MW
3660 ATOM_COUNTS = 0._EB
3661 N_REACTIONS = 0
3662 REWIND(LU_INPUT) ; INPUT_FILE_LINE_NUMBER = 0
3663
3664 COUNT_REAC_LOOP: DO
3665 CALL CHECKREAD('REAC',LU_INPUT,IOS)
3666 IF (IOS==1) EXIT COUNT_REAC_LOOP
3667 CALL SET_REAC_DEFAULTS
3668 READ(LU_INPUT,REAC,END=435,ERR=434,IOSTAT=IOS)
3669 N_REACTIONS = N_REACTIONS + 1
3670 IF (A < 0._EB .AND. E < 0._EB .AND. TRIM(SPEC_ID_NU(1))=='null' .AND. TRIM(EQUATION)=='null') SIMPLE_CHEMISTRY =
    .TRUE.
3671 IF ((A > 0._EB .OR. E > 0._EB) .AND. (SPEC_ID_N_S(1)=='null' .OR. N_S(1) < -998._EB)) THEN
3672 WRITE(MESSAGE,'(A,I0,A)') 'ERROR: Problem with REAC ',N_REACTIONS,'. SPEC_ID_N_S and N_S arrays must be defined'
3673 CALL SHUTDOWN(MESSAGE) ; RETURN
3674 ENDIF
3675 IF (.NOT.SIMPLE_CHEMISTRY .AND. TRIM(SPEC_ID_NU(1))=='null' .AND. TRIM(EQUATION)=='null') THEN
3676 WRITE(MESSAGE,'(A,I0,A)') 'ERROR: Problem with REAC ',N_REACTIONS,'. SPEC_ID_NU and NU arrays or EQUATION must be
    defined'
3677 CALL SHUTDOWN(MESSAGE) ; RETURN
3678 ENDIF
3679 434 IF (IOS>0) THEN
3680 WRITE(MESSAGE,'(A,I0,A)') 'ERROR: Problem with REAC ',N_REACTIONS+1
3681 CALL SHUTDOWN(MESSAGE) ; RETURN
3682 ENDIF
3683 ENDDO COUNT_REAC_LOOP
3684
3685 435 REWIND(LU_INPUT) ; INPUT_FILE_LINE_NUMBER = 0
3686
3687 ALLOCATE(REACTION(N_REACTIONS),STAT=IZERO)
3688
3689 ! Read and store the reaction parameters
3690
3691 NFR = 0 ! Number of fast reactions
3692
3693 REAC_READ_LOOP: DO NR=1,N_REACTIONS
3694
3695 ! Read the REAC line
3696
3697 CALL CHECKREAD('REAC',LU_INPUT,IOS)
3698 IF (IOS==1) EXIT REAC_READ_LOOP
3699 CALL SET_REAC_DEFAULTS
3700 READ(LU_INPUT,REAC)
3701
3702 ! Ensure that there is a specified fuel
3703
3704 IF (FUEL=='null' .AND. ID/'null') FUEL = ID ! Backward compatibility
3705
3706 IF (FUEL=='null' .AND. ID=='null') THEN
3707 WRITE(MESSAGE,'(A,I0,A)') 'ERROR: REAC ',NR,' requires a FUEL'
3708 CALL SHUTDOWN(MESSAGE) ; RETURN
3709 ENDIF
3710
3711 ! Set up the SIMPLE_CHEMISTRY model

```

```

3712 RN => REACTION(NR)
3713 IF (SIMPLE_CHEMISTRY) THEN
3714 IF (C<=TWO_EPSILON_EB .AND. H<=TWO_EPSILON_EB) THEN
3715 IF (TRIM(FORMULA)='null') THEN
3716 CALL GAS_PROPS(FUEL,S_TMP,E_TMP,PR_TMP,MW_FUEL,FORMULA,L_TMP,ATOM_COUNTS,H_F,RADCAL_ID)
3717 ELSE
3718 CALL GET_FORMULA_WEIGHT(FORMULA,MW_FUEL,ATOM_COUNTS)
3719 L_TMP = .TRUE.
3720 ENDIF
3721 IF (L_TMP) THEN
3722 SIMPLE_FUEL_DEFINED = .TRUE.
3723 IF (ATOM_COUNTS(1)+ATOM_COUNTS(6)+ATOM_COUNTS(7)+ATOM_COUNTS(8) - SUM(ATOM_COUNTS) < 0. _EB) THEN
3724 WRITE(MESSAGE,'(A)') 'ERROR: Fuel FORMULA for SIMPLE_CHEMISTRY can only contain C,H,O, and N'
3725 CALL SHUTDOWN(MESSAGE) ; RETURN
3726 ELSE
3727 C = ATOM_COUNTS(6)
3728 H = ATOM_COUNTS(1)
3729 O = ATOM_COUNTS(8)
3730 N = ATOM_COUNTS(7)
3731 ENDIF
3732 IF (C<=TWO_EPSILON_EB .AND. H<=TWO_EPSILON_EB) THEN
3733 WRITE(MESSAGE,'(A)') 'ERROR: Must specify fuel chemistry using C and/or H when using simple chemistry'
3734 CALL SHUTDOWN(MESSAGE) ; RETURN
3735 ENDIF
3736 ELSE
3737 SIMPLE_FUEL_DEFINED = .TRUE.
3738 MW_FUEL = ELEMENT(6)%MASS*C+ELEMENT(1)%MASS*H+ELEMENT(8)%MASS*O+ELEMENT(7)%MASS*N
3739 ENDIF
3740 ENDIF
3741 RN%A_IN = A
3742 RN%A_PRIME = A
3743 RN%AUTO_IGNITION_TEMPERATURE = AUTO_IGNITION_TEMPERATURE + TMFM
3744 RN%C = C
3745 RN%CHECK_ATOM_BALANCE = CHECK_ATOM_BALANCE
3746 RN%CO_YIELD = CO_YIELD
3747 RN%CRIT_FLAME_TMP = CRITICAL_FLAME_TEMPERATURE + TMFM
3748 RN%E = E*1000. _EB
3749 RN%E_IN = E
3750 RN%K = K
3751 RN%EQUATION = EQUATION
3752 RN%EPUMO2 = EPUMO2*1000. _EB
3753 RN%FAST_CHEMISTRY = FAST_CHEMISTRY
3754 RN%FLAME_SPEED = FLAME_SPEED
3755 RN%FLAME_SPEED_EXPONENT = FLAME_SPEED_EXPONENT
3756 RN%FLAME_SPEED_TEMPERATURE = FLAME_SPEED_TEMPERATURE + TMFM
3757 IF (RN%FLAME_SPEED_TEMPERATURE <=0. _EB) RN%FLAME_SPEED_TEMPERATURE = TMPA
3758 RN%FUEL = FUEL
3759 RN%FWD_ID = FWD_ID
3760 RN%FYI = FYI
3761 RN%H = H
3762 RN%HEAT_OF_COMBUSTION = HEAT_OF_COMBUSTION*1000. _EB
3763 RN%ID = ID
3764 RN%MW_FUEL = MW_FUEL
3765 RN%MW_SOOT = ELEMENT(6)%MASS * (1. _EB - SOOT_H_FRACTION) + ELEMENT(1)%MASS*SOOT_H_FRACTION
3766 RN%N = N
3767 RN%N_T = N.T
3768 RN%O = O
3769 RN%RAMP_CHLR = RAMP_CHLR
3770 RN%RAMP_FS = RAMP_FS
3771 RN%TABLE_FS = TABLE_FS
3772 RN%REVERSE = REVERSE
3773 RN%SOOT_H_FRACTION = SOOT_H_FRACTION
3774 RN%SOOT_YIELD = SOOT_YIELD
3775 RN%THIRD_BODY = THIRD_BODY
3776 RN%TURBULENT_FLAME_SPEED_ALPHA = TURBULENT_FLAME_SPEED_ALPHA
3777 RN%TURBULENT_FLAME_SPEED_EXPONENT = TURBULENT_FLAME_SPEED_EXPONENT
3778 RN%SERIES_REACTION = SERIES_REACTION !Sesa-added as in 6.2.0
3779
3780 IF (RN%RAMP_FS='null' .AND. RN%TABLE_FS='null') THEN
3781 WRITE(MESSAGE,'(A)') 'ERROR: Can only specify one of RAMP_FS or TABLE_FS'
3782 CALL SHUTDOWN(MESSAGE) ; RETURN
3783 ENDIF
3784 IF (RN%RAMP_FS='null') CALL GET_RAMP_INDEX(RN%RAMP_FS,'EQUIVALENCE_RATIO',RN%RAMP_FS_INDEX)
3785 IF (RN%RAMP_CHLR='null') CALL GET_RAMP_INDEX(RN%RAMP_CHLR,'TIME',RN%RAMP_CHLR_INDEX)
3786 IF (RN%TABLE_FS='null') CALL GET_TABLE_INDEX(RN%TABLE_FS,FLAME_SPEED_TABLE,RN%TABLE_FS_INDEX)
3787
3788 IF (RN%A_PRIME==-1. _EB .AND. RN%E==-1000. _EB .AND. .NOT.RN%REVERSE) THEN
3789 RN%FAST_CHEMISTRY=.TRUE.
3790 NFR = NFR + 1
3791 ENDIF
3792
3793 ! Check appropriate extinction model
3794
3795 IF (NFR > 1 .AND. (EXTINCT_MOD == 2 .OR. EXTINCT_MOD == 6) .AND. SUPPRESSION) THEN
3796 WRITE(MESSAGE,'(A)') 'ERROR: The default EXTINCTION MODEL is designed for 1 reaction. See Tech Guide'
3797 CALL SHUTDOWN(MESSAGE) ; RETURN
3798

```



```

3800   ENDIF
3801
3802   ! Determine the number of stoichiometric coefficients for this reaction
3803
3804   IF (.NOT.SIMPLE.CHEMISTRY) THEN
3805     NS2 = 0
3806     DO NS=1,MAX_SPECIES
3807       IF (TRIM(SPEC_ID_NU(NS))/= 'null') THEN
3808         NS2=NS2+1
3809       ELSE
3810         EXIT
3811       ENDIF
3812     ENDDO
3813     RN%N_SMIX = NS2
3814     NS2 = 0
3815     IF (TRIM(RN%EQUATION)/= 'null') RN%N_SMIX = MAX_SPECIES
3816     DO NS=1,MAX_SPECIES
3817       IF (TRIM(SPEC_ID_N_S(NS))/= 'null') THEN
3818         NS2=NS2+1
3819       ELSE
3820         EXIT
3821       ENDIF
3822     ENDDO
3823     RN%N_SPEC = NS2
3824     ELSE
3825     RN%N_SMIX = 3
3826     RN%N_SPEC = 0
3827   ENDIF
3828
3829   ! Store the "read in" values of N_S, NU, and SPEC_ID_NU for use in PROC.REAC.
3830
3831   IF (RN%N_SPEC > 0) THEN
3832     ALLOCATE(RN%N_S_READ(RN%N_SPEC))
3833     RN%N_S_READ(1:RN%N_SPEC) = N_S(1:RN%N_SPEC)
3834     ALLOCATE(RN%SPEC_ID_N_S_READ(RN%N_SPEC))
3835     RN%SPEC_ID_N_S_READ = 'null'
3836     RN%SPEC_ID_N_S_READ(1:RN%N_SPEC)=SPEC_ID_N_S(1:RN%N_SPEC)
3837   ENDIF
3838   ALLOCATE(RN%NU_READ(RN%N_SMIX))
3839   RN%NU_READ(1:RN%N_SMIX) = NU(1:RN%N_SMIX)
3840
3841   ALLOCATE(RN%SPEC_ID_NU_READ(RN%N_SMIX))
3842   RN%SPEC_ID_NU_READ = 'null'
3843   RN%SPEC_ID_NU_READ(1:RN%N_SMIX)=SPEC_ID_NU(1:RN%N_SMIX)
3844
3845   IF (RADIATIVE_FRACTION < 0._EB) THEN
3846     IF (DNS) RN%CHLR = 0._EB
3847     IF (LES) CALL LOOKUP_CHLR(FUEL,RN%CHLR)
3848   ELSE
3849     RN%CHLR = RADIATIVE_FRACTION
3850   ENDIF
3851
3852   !Sesa-added for RADIATIVE_FRACTION_I
3853   ! IF (LES) RADIATIVE_FRACTION_I = 0.35_EB
3854   ! IF (DNS) RADIATIVE_FRACTION_I = 0.00_EB
3855   !Sesa-adding end
3856
3857   ENDDO REAC.READ.LOOP
3858
3859   REWIND(LU.INPUT) ; INPUT_FILE_LINE_NUMBER = 0
3860
3861   CONTAINS
3862
3863   SUBROUTINE SET_REAC_DEFAULTS
3864
3865   AUTO_IGNITION_TEMPERATURE = -1MFM
3866   A = -1._EB
3867   C = 0._EB
3868   CHECK_ATOM_BALANCE = .TRUE.
3869   CO_YIELD = 0._EB
3870   CRITICAL_FLAME_TEMPERATURE = 1427._EB ! C (See C. Beyler, Ch 7, Eq 6, SFPE Handbook, 4th Ed.)
3871   E = -1._EB ! kj/kmol
3872   EPUMO2 = 13100._EB ! kj/kg
3873   K = 1._EB
3874   EQUATION = 'null'
3875   FAST_CHEMISTRY = .FALSE.
3876   FLAME_SPEED = -1._EB
3877   FLAME_SPEED_EXPONENT = 0._EB
3878   FLAME_SPEED_TEMPERATURE = -273.15_EB
3879   FORMULA = 'null'
3880   FUEL = 'null'
3881   FWD_ID = 'null'
3882   FYI = 'null'
3883   H = 0._EB
3884   HEAT_OF_COMBUSTION = -2.E20_EB
3885   ID = 'null'
3886   N = 0._EB
3887   NU = 0._EB

```

Source Code files for edited portions of FDS

```

3888 N_S = -999._EB
3889 N_T = 0._EB
3890 O = 0._EB
3891 ODE.SOLVER = 'null'
3892 RADIATIVE.FRACTION = -1._EB
3893 RAMP.CHLR = 'null'
3894 RAMP.FS = 'null'
3895 REAC.ATOM.ERROR = 1.E-4.EB
3896 REAC.MASS.ERROR = 1.E-4.EB
3897 REVERSE = .FALSE.
3898 SOOT.H.FRACTION = 0.1.EB
3899 SOOT.YIELD = 0.0.EB
3900 SPEC.ID.NU = 'null'
3901 SPEC.ID.N.S = 'null'
3902 TABLE.FS = 'null'
3903 THIRD.BODY = .FALSE.
3904 TURBULENT.FLAME.SPEED.ALPHA = 1.EB ! see O. Gulder. 23rd Int. Symp. on Comb. 1990.
3905 TURBULENT.FLAME.SPEED.EXPONENT = 2.EB
3906 SERIES.REACTION = .FALSE. ! Sesa-added as in 6.2.0
3907
3908 END SUBROUTINE SET_REAC_DEFAULTS
3909
3910 END SUBROUTINE READ_REAC
3911
3912
3913 SUBROUTINE PROC_REAC_1
3914 USE PROPERTY_DATA, ONLY : PARSE_EQUATION, SHUTDOWN_ATOM
3915 REAL (EB) :: MASS_PRODUCT, MASS_REACTANT, REACTION_BALANCE(118)
3916 INTEGER :: NS, NS2, NR, NSPEC
3917 LOGICAL :: NAME_FOUND, SKIP_ATOM_BALANCE
3918 TYPE (SPECIES_MIXTURE_TYPE), POINTER :: SM
3919
3920 IF (N_REACTIONS <= 0) RETURN
3921
3922 ! Basic input error checking
3923
3924 IF (SIMPLE_CHEMISTRY .AND. N_REACTIONS > 1) THEN
3925 WRITE(MESSAGE, '(A)') 'ERROR: can not have more than one reaction when using simple chemistry'
3926 CALL SHUTDOWN(MESSAGE) ; RETURN
3927 ENDIF
3928
3929 ! The following information is what the user would have entered into the input file in the more general case
3930
3931 IF (SIMPLE_CHEMISTRY) THEN
3932 RN => REACTION(1)
3933 IF (RN%NU_O2 <= 0._EB) THEN
3934 WRITE(MESSAGE, '(A)') 'ERROR: Fuel specified for simple chemistry has NU_O2 <= 0 and it must require air for
3935 combustion.'
3936 CALL SHUTDOWN(MESSAGE) ; RETURN
3937 ENDIF
3938 RN%SPEC_ID_NU_READ(1) = RN%FUEL
3939 RN%SPEC_ID_NU_READ(2) = 'AIR'
3940 RN%SPEC_ID_NU_READ(3) = 'PRODUCTS'
3941 RN%NU_READ(1) = -1._EB
3942 RN%NU_READ(2) = -RN%NU_O2/SPECIES_MIXTURE(1)%VOLUME_FRACTION(O2_INDEX)
3943 RN%NU_READ(3) = -(RN%NU_READ(1)*SPECIES_MIXTURE(FUEL_SMIX_INDEX)%MW+RN%NU_READ(2)*SPECIES_MIXTURE(1)%MW)/ &
3944 SPECIES_MIXTURE(3)%MW
3945 RN%N_SMIX = 3
3946 ENDIF
3947
3948 REAC_LOOP: DO NR=1, N_REACTIONS
3949 RN => REACTION(NR)
3950
3951 IF ((RN%A_PRIME > 0._EB .OR. RN%E > 0._EB) .AND. (RN%C < TWO_EPSILON_EB .OR. RN%H > TWO_EPSILON_EB)) THEN
3952 WRITE(MESSAGE, '(A)') 'ERROR: cannot use both finite rate REAC and simple chemistry'
3953 CALL SHUTDOWN(MESSAGE) ; RETURN
3954 ENDIF
3955 IF (TRIM(RN%EQUATION) /= 'null') THEN
3956 IF (ANY(ABS(RN%NU_READ) > TWO_EPSILON_EB)) THEN
3957 WRITE(MESSAGE, '(A,I0,A)') 'ERROR: Problem with REAC ', NR, '. Cannot set NUS if an EQUATION is specified.'
3958 CALL SHUTDOWN(MESSAGE) ; RETURN
3959 ENDIF
3960 CALL PARSE_EQUATION(NR)
3961 RN%N_SMIX = 0
3962 DO NS=1, N_TRACKED_SPECIES+1
3963 IF (ABS(RN%NU_READ(NS)) > TWO_EPSILON_EB) THEN
3964 RN%N_SMIX = RN%N_SMIX+1
3965 ENDIF
3966 ENDDO
3967 ENDIF
3968
3969 IF (TRIM(RN%FUEL) == 'null') THEN
3970 WRITE(MESSAGE, '(A,I0,A)') 'ERROR: Problem with REAC ', NR, '. FUEL must be defined'
3971 CALL SHUTDOWN(MESSAGE) ; RETURN
3972 ENDIF
3973
3974 ! Allocate the arrays that are going to carry the mixture stoichiometry to the rest of the code

```

Source Code files for edited portions of FDS

```

3975
3976 ALLOCATE(RN%SPEC_ID_NU(1:N_TRACKED_SPECIES))
3977 ALLOCATE(RN%NU(1:N_TRACKED_SPECIES))
3978 ALLOCATE(RN%NUMW.O.MW.F(1:N_TRACKED_SPECIES))
3979 ALLOCATE(RN%SPEC_ID_N.S(1:N_SPECIES))
3980 ALLOCATE(RN%N.S(1:N_SPECIES))
3981 RN%SPEC_ID_NU = 'null'
3982 RN%SPEC_ID_N.S = 'null'
3983 RN%NU = 0._EB
3984 RN%N.S = -999._EB
3985
3986 ! Transfer SPEC_ID_NU, SPEC_ID_N, NU, and N.S that were indexed by the order they were read in
3987 ! to now be indexed by the SMIX or SPEC index
3988
3989 DO NS=1,RN%N.SMIX
3990 IF (TRIM(RN%SPEC_ID_NU_READ(NS))=='null') CYCLE
3991 NAMEFOUND = .FALSE.
3992 DO NS2=1,N_TRACKED_SPECIES
3993 IF (TRIM(RN%SPEC_ID_NU_READ(NS))=TRIM(SPECIES_MIXTURE(NS2)%ID)) THEN
3994 RN%SPEC_ID_NU(NS2) = RN%SPEC_ID_NU_READ(NS)
3995 RN%NU(NS2) = RN%NU_READ(NS)
3996 NAMEFOUND = .TRUE.
3997 EXIT
3998 ENDIF
3999 IF (TRIM(RN%EQUATION)/='null') THEN
4000 IF (TRIM(RN%SPEC_ID_NU_READ(NS))=TRIM(SPECIES_MIXTURE(NS2)%FORMULA)) THEN
4001 RN%SPEC_ID_NU(NS2) = SPECIES_MIXTURE(NS2)%ID
4002 RN%NU(NS2) = RN%NU_READ(NS)
4003 NAMEFOUND = .TRUE.
4004 EXIT
4005 ENDIF
4006 ENDIF
4007 ENDDO
4008 IF (.NOT. NAMEFOUND) THEN
4009 WRITE(MESSAGE,'(A,I0,A,A,A)') 'ERROR: Problem with REAC ',NR,'. Tracked species ',TRIM(RN%SPEC_ID_NU_READ(NS)),&
4010 ' not found.'
4011 CALL SHUTDOWN(MESSAGE) ; RETURN
4012 ENDIF
4013 ENDDO
4014
4015 ! Look for indices of fuels, oxidizers, and products. Normalize the stoichiometric coefficients by that of the
4016 ! fuel.
4017 DO NS2=1,N_TRACKED_SPECIES
4018 IF (ABS(RN%NU(NS2))<TWO.EPSILON.EB) CYCLE
4019 DO NSPEC=1,N_SPECIES
4020 IF (SPECIES_MIXTURE(NS2)%SPEC_ID(NSPEC)=RN%FUEL .OR. SPECIES_MIXTURE(NS2)%ID=RN%FUEL) THEN
4021 RN%FUEL_SMIX_INDEX = NS2
4022 RN%NU = -RN%NU/RN%NU(NS2)
4023 EXIT
4024 ENDIF
4025 ENDDO
4026 ENDDO
4027
4028 ! Find AIR index
4029
4030 GET_AIR_INDEX_LOOP: DO NS = 1,N_TRACKED_SPECIES
4031 IF (RN%NU(NS) < 0._EB .AND. NS /= RN%FUEL_SMIX_INDEX) THEN
4032 RN%AIR_SMIX_INDEX = NS
4033 EXIT GET_AIR_INDEX_LOOP
4034 ENDIF
4035 ENDDO GET_AIR_INDEX_LOOP
4036
4037 ! Adjust mol/cm^3/s based rate to kg/m^3/s rate
4038
4039 RN%RHO_EXPONENT = 0._EB
4040 DO NS=1,RN%N.SPEC
4041 IF (TRIM(RN%SPEC_ID_N.S_READ(NS))=='null') CYCLE
4042 IF (RN%A.PRIME < 0.0.EB) CYCLE
4043 NAMEFOUND = .FALSE.
4044 DO NS2=1,N_SPECIES
4045 IF (TRIM(RN%SPEC_ID_N.S_READ(NS))=TRIM(SPECIES(NS2)%ID)) THEN
4046 RN%SPEC_ID_N.S(NS2) = RN%SPEC_ID_N.S_READ(NS)
4047 RN%N.S(NS2) = RN%N.S_READ(NS)
4048 RN%A.PRIME = RN%A.PRIME * (1000._EB*SPECIES(NS2)%MW)**(-RN%N.S(NS2)) ! FDS Tech Guide, Eq. (5.46), product
4049 ! term
4049 RN%RHO_EXPONENT = RN%RHO_EXPONENT + RN%N.S(NS2)
4050 NAMEFOUND = .TRUE.
4051 EXIT
4052 ENDIF
4053 ENDDO
4054 IF (.NOT. NAMEFOUND) THEN
4055 WRITE(MESSAGE,'(A,I0,A,A,A)') &
4056 'ERROR: Problem with REAC ',NR,'. Primitive species ',TRIM(RN%SPEC_ID_N.S_READ(NS)), ' not found.'
4057 CALL SHUTDOWN(MESSAGE) ; RETURN
4058 ENDIF
4059 ENDDO
4060 RN%RHO_EXPONENT = RN%RHO_EXPONENT - 1._EB ! subtracting 1 accounts for division by rho in Eq. (5.49)

```

## Source Code files for edited portions of FDS

```

4061 RN%A_PRIME = RN%A_PRIME * 1000._EB*SPECIES_MIXTURE(RN%FUEL_SMIX_INDEX)%MW ! conversion terms in Eq. (5.46)
4062
4063 ! Adjust mol/cm^3/s based rate to kg/m^3/s rate for FAST_CHEMISTRY (this will get removed when we overhaul
      combustion)
4064 ! Fictitious Arrhenius rate is dC.F/dt = -1E10*C.F*C_A
4065
4066 IF (RN%FAST_CHEMISTRY) THEN
4067 IF (RN%AIR_SMIX_INDEX > -1) THEN
4068 RN%RHO_EXPONENT_FAST = 1._EB
4069 RN%A_PRIME_FAST = 1.E10.EB*(1000._EB*SPECIES_MIXTURE(RN%AIR_SMIX_INDEX)%MW)**(-1._EB)
4070 ELSE
4071 RN%RHO_EXPONENT_FAST = 0._EB
4072 RN%A_PRIME_FAST = 1.E10.EB
4073 ENDIF
4074 ENDIF
4075
4076 ! Compute the primitive species reaction coefficients
4077
4078 ALLOCATE(RN%NU_SPECIES(N_SPECIES))
4079 RN%NU_SPECIES = 0._EB
4080 DO NS=1,N_TRACKED_SPECIES
4081 SM => SPECIES_MIXTURE(NS)
4082 RN%NU(NS) = RN%NU(NS)*SM%ADJUST_NU
4083 DO NS2 = 1,N_SPECIES
4084 RN%NU_SPECIES(NS2) = RN%NU_SPECIES(NS2) + RN%NU(NS)*SM%VOLUME_FRACTION(NS2)
4085 ENDDO
4086 IF (SM%ID=='WATER VAPOR') LWATER = NS
4087 IF (SM%ID=='CARBON DIOXIDE') LCO2 = NS
4088 ENDDO
4089
4090 ! Check atom balance of the reaction
4091
4092 IF (.NOT. SIMPLE_CHEMISTRY .AND. RN%CHECK_ATOM_BALANCE) THEN
4093 SKIP_ATOM_BALANCE = .FALSE.
4094 REACTION_BALANCE = 0._EB
4095 DO NS=1,N_TRACKED_SPECIES
4096 IF (ABS(RN%NU(NS))>TWO_EPSILON_EB .AND. .NOT. SPECIES_MIXTURE(NS)%VALID_ATOMS) SKIP_ATOM_BALANCE = .TRUE.
4097 REACTION_BALANCE = REACTION_BALANCE + RN%NU(NS)*SPECIES_MIXTURE(NS)%ATOMS
4098 ENDDO
4099 IF (ANY(ABS(REACTION_BALANCE)>REAC_ATOM_ERROR) .AND. .NOT. SKIP_ATOM_BALANCE) THEN
4100 CALL SHUTDOWN_ATOM(REACTION_BALANCE,NR,REAC_ATOM_ERROR) ; RETURN
4101 ENDIF
4102 ENDIF
4103
4104 ! Check the mass balance of the reaction
4105
4106 MASS_REACTANT = 0._EB
4107 MASS_PRODUCT = 0._EB
4108
4109 DO NS=1,N_TRACKED_SPECIES
4110 IF (RN%NU(NS) < -TWO_EPSILON_EB) MASS_REACTANT = MASS_REACTANT + RN%NU(NS)*SPECIES_MIXTURE(NS)%MW
4111 IF (RN%NU(NS) > TWO_EPSILON_EB) MASS_PRODUCT = MASS_PRODUCT + RN%NU(NS)*SPECIES_MIXTURE(NS)%MW
4112 ENDDO
4113 IF (ABS(MASS_PRODUCT) < TWO_EPSILON_EB .OR. ABS(MASS_REACTANT) < TWO_EPSILON_EB) THEN
4114 IF (ABS(MASS_PRODUCT) < TWO_EPSILON_EB) WRITE(MESSAGE,'(A,10,A)') 'ERROR: Problem with REAC ',NR,'. Products not
      specified.'
4115 IF (ABS(MASS_REACTANT) < TWO_EPSILON_EB) WRITE(MESSAGE,'(A,10,A)') 'ERROR: Problem with REAC ',NR,'. Reactants not
      specified.'
4116 CALL SHUTDOWN(MESSAGE) ; RETURN
4117 ENDIF
4118 IF (ABS(MASS_PRODUCT+MASS_REACTANT)/ABS(MASS_PRODUCT) > REAC_MASS_ERROR) THEN
4119 WRITE(MESSAGE,'(A,10,A,F8.3,A,F8.3)') 'ERROR: Problem with REAC ',NR,'. Mass of products, ',MASS_PRODUCT, &
4120 ', does not equal mass of reactants, ',-MASS_REACTANT
4121 CALL SHUTDOWN(MESSAGE) ; RETURN
4122 ENDIF
4123
4124 ! Mass stoichiometric coefficient of oxidizer
4125
4126 DO NS=1,N_TRACKED_SPECIES
4127 RN%NU_MW_O_MWF(NS) = RN%NU(NS)*SPECIES_MIXTURE(NS)%MW/SPECIES_MIXTURE(RN%FUEL_SMIX_INDEX)%MW
4128 IF (RN%NU(NS) < 0._EB .AND. NS /= RN%FUEL_SMIX_INDEX) THEN
4129 RN%S = -RN%NU_MW_O_MWF(NS)
4130 ENDIF
4131 ENDDO
4132
4133 ENDDO REAC_LOOP
4134
4135 ! Select integrator
4136
4137 IF (TRIM(ODE_SOLVER)/='null') THEN
4138 SELECT CASE (TRIM(ODE_SOLVER))
4139 CASE ('EXPLICIT_EULER')
4140 COMBUSTION_ODE_SOLVER = EXPLICIT_EULER
4141 CASE ('RK2')
4142 COMBUSTION_ODE_SOLVER = RK2
4143 CASE ('RK3')
4144 COMBUSTION_ODE_SOLVER = RK3
4145 CASE ('RK2_RICHARDSON')

```

```

4146 COMBUSTION.ODE.SOLVER = RK2.RICHARDSON
4147 CASE DEFAULT
4148 WRITE(MESSAGE,'(A)') 'ERROR: Problem with REAC. Name of ODE.SOLVER is not recognized.'
4149 CALL SHUTDOWN(MESSAGE) ; RETURN
4150 END SELECT
4151 ELSE
4152 FAST_CHEM_LOOP: DO NR = 1, N_REACTIONS
4153 RN => REACTION(NR)
4154 IF (.NOT. RN%FAST_CHEMISTRY) THEN
4155 COMBUSTION.ODE.SOLVER = RK2.RICHARDSON
4156 EXIT FAST_CHEM_LOOP
4157 ELSE
4158 COMBUSTION.ODE.SOLVER = EXPLICIT_EULER
4159 ENDIF
4160 ENDDO FAST_CHEM_LOOP
4161 ENDIF
4162
4163 END SUBROUTINE PROC_REAC_1
4164
4165
4166 SUBROUTINE PROC_REAC_2
4167 INTEGER :: NS, NR, HF_COUNT, L_HFRT, NRR ! Sesa-added NRR as in 6.2.0
4168 REAL(EB) :: H_F_OLD(1:N_TRACKED_SPECIES), D_HFRT, D_H
4169 LOGICAL :: REDEFINE_H_F(1:N_TRACKED_SPECIES), LISTED_FUEL
4170 TYPE (SPECIES_MIXTURE_TYPE), POINTER :: SM, SMF
4171 TYPE(REACTION_TYPE), POINTER :: RN=>NULL()
4172
4173 IF (N_REACTIONS <=0) RETURN
4174
4175 REDEFINE_H_F = .FALSE.
4176 LISTED_FUEL = .FALSE.
4177 H_F_OLD = SPECIES_MIXTURE%H_F
4178
4179 REAC_LOOP: DO NR=1, N_REACTIONS
4180
4181 RN => REACTION(NR)
4182 SMF => SPECIES_MIXTURE(RN%FUEL_SMIX_INDEX)
4183 IF (SIMPLE_CHEMISTRY) THEN
4184 LISTED_FUEL = .TRUE.
4185 DO NS=1, N_SPECIES
4186 IF (SMP%VOLUME_FRACTION(NS) > 0._EB .AND. .NOT. SPECIES(NS)%EXPLICIT_H_F) LISTED_FUEL = .FALSE.
4187 IF (.NOT. SPECIES(NS)%LISTED .AND. SPECIES(NS)%RADCAL_ID = 'null') SPECIES(NS)%RADCAL_ID = FUEL_RADCAL_ID
4188 ENDDO
4189 ENDIF
4190
4191 ! Heat of Combustion calculation
4192 HOC_IF: IF (RN%HEAT_OF_COMBUSTION > -1.E21) THEN ! User specified heat of combustion
4193 IF (SIMPLE_CHEMISTRY) THEN
4194 IF (IDEAL) THEN
4195 RN%HEAT_OF_COMBUSTION = RN%HEAT_OF_COMBUSTION * SMP%MW * 0.001 ! J/kg -> J/mol
4196 RN%HEAT_OF_COMBUSTION = RN%HEAT_OF_COMBUSTION + ( &
4197 RN%NU_CO * (SPECIES(CO2_INDEX)%H_F - SPECIES(CO_INDEX)%H_F) &
4198 + RN%NU_SOOT * SPECIES(CO2_INDEX)%H_F * (1._EB - RN%SOOT_H_FRACTION) &
4199 + RN%NU_SOOT * SPECIES(H2O_INDEX)%H_F * RN%SOOT_H_FRACTION * 0.5._EB)
4200 RN%HEAT_OF_COMBUSTION = RN%HEAT_OF_COMBUSTION / SMP%MW * 1000._EB ! J/mol->J/kg
4201 ENDIF
4202 RN%EPUMO2 = RN%HEAT_OF_COMBUSTION*SMP%MW / (RN%NU_O2 * SPECIES(O2_INDEX)%MW)
4203 ENDIF
4204 HF_COUNT = 0
4205 DO NS = 1, N_TRACKED_SPECIES
4206 IF (RN%NU(NS) /= 0._EB) THEN
4207 IF (SPECIES_MIXTURE(NS)%H_F <= -1.E21) HF_COUNT = HF_COUNT + 1
4208 IF (HF_COUNT > 1) THEN
4209 WRITE(MESSAGE,'(A,I0,A)') 'ERROR: Problem with REAC ', NR, '. Missing more than 1 species heat of formation.'
4210 CALL SHUTDOWN(MESSAGE) ; RETURN
4211 ENDIF
4212 ENDIF
4213 ENDDO
4214 ! Find heat of formation of lumped fuel to satisfy specified heat of combustion
4215 IF (REDEFINE_H_F(RN%FUEL_SMIX_INDEX)) THEN
4216 WRITE(MESSAGE,'(A,I0,A)') 'WARNING: H_F for FUEL for REACTION ', NR, ' was redefined multiple times.'
4217 IF (MYID=0) WRITE(LU_ERR,'(A)') TRIM(MESSAGE)
4218 ENDIF
4219 REDEFINE_H_F(RN%FUEL_SMIX_INDEX) = .TRUE.
4220 SMP%H_F = RN%HEAT_OF_COMBUSTION * ABS(RN%NU(RN%FUEL_SMIX_INDEX)) * SMP%MW * 0.001._EB
4221 DO NS = 1, N_TRACKED_SPECIES
4222 IF (NS == RN%FUEL_SMIX_INDEX) CYCLE
4223 SM=>SPECIES_MIXTURE(NS)
4224 SMP%H_F = SMP%H_F + RN%NU(NS) * SM%H_F * SMP%MW * 0.001._EB
4225 ENDDO
4226 IF (SMP%SINGLE_SPEC_INDEX > 0) SPECIES(SMP%SINGLE_SPEC_INDEX)%H_F = -SMP%H_F / RN%NU(RN%FUEL_SMIX_INDEX)
4227 SMP%H_F = -SMP%H_F / (RN%NU(RN%FUEL_SMIX_INDEX) * SMP%MW * 0.001._EB)
4228
4229
4230 ELSE HOC_IF ! Heat of combustion not specified
4231 IF (SIMPLE_CHEMISTRY) THEN ! Calculate heat of combustion based oxygen consumption
4232 IF (RN%EPUMO2 > 0._EB .AND. .NOT. LISTED_FUEL) THEN
4233 RN%HEAT_OF_COMBUSTION = -RN%EPUMO2 * RN%NU_SPECIES(O2_INDEX) * SPECIES(O2_INDEX)%MW / SMP%MW

```

Source Code files for edited portions of FDS

```

4234 REDEFINE_H.F(RN%FUEL_SMIX_INDEX) = .TRUE.
4235 SMP%H.F = RN%HEAT_OF_COMBUSTION * ABS(RN%NU(RN%FUEL_SMIX_INDEX)) * SMP%MW * 0.001_EB
4236 DO NS = 1,N_TRACKED_SPECIES
4237 IF (NS == RN%FUEL_SMIX_INDEX) CYCLE
4238 SM=>SPECIES_MIXTURE(NS)
4239 SMP%H.F = SMP%H.F + RN%NU(NS) * SM%H.F * SMP%MW * 0.001_EB
4240 ENDDO
4241 IF (SMP%SINGLE_SPEC_INDEX>0) SPECIES(SMP%SINGLE_SPEC_INDEX)%H.F = -SMP%H.F / RN%NU(RN%FUEL_SMIX_INDEX)
4242 SMP%H.F = -SMP%H.F / (RN%NU(RN%FUEL_SMIX_INDEX) * SMP%MW * 0.001_EB)
4243 ELSE
4244 RN%HEAT_OF_COMBUSTION = 0._EB
4245 DO NS = 1,N_TRACKED_SPECIES
4246 SM=>SPECIES_MIXTURE(NS)
4247 RN%HEAT_OF_COMBUSTION = RN%HEAT_OF_COMBUSTION - RN%NU(NS) * SM%H.F * SMP%MW
4248 ENDDO
4249 RN%HEAT_OF_COMBUSTION = RN%HEAT_OF_COMBUSTION / SMP%MW
4250 RN%EPUMO2 = RN%HEAT_OF_COMBUSTION * SMP%MW / (RN%NU_O2 * SPECIES(O2_INDEX)%MW)
4251 ENDDO
4252 ELSE
4253 RN%HEAT_OF_COMBUSTION = 0._EB
4254 DO NS = 1,N_TRACKED_SPECIES
4255 SM=>SPECIES_MIXTURE(NS)
4256 RN%HEAT_OF_COMBUSTION = RN%HEAT_OF_COMBUSTION - RN%NU(NS) * SM%H.F * SMP%MW
4257 ENDDO
4258 RN%HEAT_OF_COMBUSTION = RN%HEAT_OF_COMBUSTION / SPECIES_MIXTURE(RN%FUEL_SMIX_INDEX)%MW
4259 ENDDO
4260 ENDDO HOC_IF
4261
4262 IF (NR==1) REACTION%HOCCOMPLETE = RN%HEAT_OF_COMBUSTION
4263
4264 ENDDO REAC_LOOP
4265
4266 !Sesa--Added this as in 6.2.0
4267 ! Determine number of fast/potentially fast series reactions
4268 N_SERIES_REACTIONS = 0
4269 SERIES_REACTION_LOOP: DO NR = 1,N_REACTIONS
4270 RN => REACTION(NR)
4271 RN%SERIES_REACTION = .FALSE.
4272 ! Special treatment of series reactions is only needed for fast chemistry
4273 FAST_CHEM_IF: IF (RN%FAST_CHEMISTRY) THEN
4274
4275 ! Determine whether a product of reaction RN is fuel of a different reaction RNN
4276 SERIES_SPECIES_LOOP: DO NS = 1,N_TRACKED_SPECIES
4277 PRODUCTS_IF: IF (RN%NU(NS) > 0._EB) THEN
4278
4279 RNN_LOOP: DO NRR = 1,N_REACTIONS
4280 RNN => REACTION(NRR)
4281 IF (RN%FAST_CHEMISTRY) THEN
4282 IF (NRR == NR) CYCLE RNN_LOOP
4283 IF (NS == RNN%FUEL_SMIX_INDEX) THEN
4284 N_SERIES_REACTIONS = N_SERIES_REACTIONS + 1
4285 RN%SERIES_REACTION = .TRUE.
4286 ENDDO
4287 ENDDO
4288 ENDDO RNN_LOOP
4289
4290 ENDDO PRODUCTS_IF
4291 ENDDO SERIES_SPECIES_LOOP
4292
4293 ENDDO FAST_CHEM_IF
4294 ENDDO SERIES_REACTION_LOOP
4295 !Sesa--end added as in 6.2.0
4296
4297 ! Correct CP_BAR array if H.F is redefined.
4298 IF (ANY(REDEFINE_H.F)) THEN
4299 DO NS=1,N_TRACKED_SPECIES
4300 IF (.NOT.REDEFINE_H.F(NS)) CYCLE
4301 CPBAR_Z(0,NS) = CPBAR_Z(0,NS) + SPECIES_MIXTURE(NS)%H.F - H.F_OLD(NS)
4302 DO J=1,5000
4303 CPBAR_Z(J,NS) = CPBAR_Z(J,NS) + (SPECIES_MIXTURE(NS)%H.F - H.F_OLD(NS))/REAL(J,EB)
4304 ENDDO
4305 ENDDO
4306 ELSE
4307 IF (SIMPLE_CHEMISTRY .AND. CONSTANT_SPECIFIC_HEAT_RATIO) THEN
4308 H.F_OLD = 0._EB
4309 I_HFRT = INT(H.F_REFERENCE_TEMPERATURE)
4310 D_HFRT = H.F_REFERENCE_TEMPERATURE - REAL(I_HFRT,EB)
4311 RN => REACTION(1)
4312 DO NS=1,N_TRACKED_SPECIES
4313 H.F_OLD(NS) = RN%NU(NS)*SPECIES_MIXTURE(NS)%MW*(CPBAR_Z(I_HFRT,NS)+D_HFRT*(CPBAR_Z(I_HFRT+1,NS)-CPBAR_Z(I_HFRT,NS)
4314 )))
4315 ENDDO
4316 H.F_OLD = H.F_OLD*H.F_REFERENCE_TEMPERATURE
4317 D_H = -SUM(H.F_OLD)/SPECIES_MIXTURE(RN%FUEL_SMIX_INDEX)%MW-RN%HEAT_OF_COMBUSTION
4318 CPBAR_Z(0,RN%FUEL_SMIX_INDEX) = CPBAR_Z(0,RN%FUEL_SMIX_INDEX) - D_H
4319 DO J=1,5000
4319 CPBAR_Z(J,RN%FUEL_SMIX_INDEX) = CPBAR_Z(J,RN%FUEL_SMIX_INDEX) -D_H/REAL(J,EB)
4320 ENDDO

```

```

4321 ENDIF
4322 ENDIF
4323
4324 END SUBROUTINE PROC.REAC.2
4325
4326
4327 SUBROUTINE READ.PART
4328
4329 USE MATH.FUNCTIONS, ONLY : GET.RAMP.INDEX
4330 USE DEVICE.VARIABLES, ONLY : PROPERTY.TYPE
4331 USE RADCONS, ONLY : MIE.NDG
4332 USE MATH.FUNCTIONS, ONLY : GET.TABLE.INDEX
4333 INTEGER :: SAMPLING.FACTOR,N,NN,ILPC,IPC,RGB(3),N.STRATA,N.LAGRANGIAN.CLASSES.READ
4334 REAL(EB) :: DIAMETER, GAMMAD,AGE,INITIAL.TEMPERATURE,HEAT.OF.COMBUSTION, &
4335 VERTICAL.VELOCITY,HORIZONTAL.VELOCITY,MAXIMUM.DIAMETER,MINIMUM.DIAMETER,SIGMA.D, &
4336 SURFACE.TENSION,BREAKUP.RATIO,BREAKUP.GAMMA.D,BREAKUP.SIGMA.D,&
4337 DENSE.VOLUME.FRACTION,REAL.REFRACTIVE.INDEX,COMPLEX.REFRACTIVE.INDEX,RUNNING.AVERAGE.FACTOR
4338 REAL(EB) :: DRAG.COEFFICIENT(3),FREE.AREA.FRACTION,PERMEABILITY(3),POROUS.VOLUME.FRACTION
4339 REAL(EB), DIMENSION(3,10) :: ORIENTATION
4340 REAL(EB), DIMENSION(3) :: OR.TEMP
4341 CHARACTER(LABEL.LENGTH) :: SPEC.ID,DEVC.ID,CTRL.ID,QUANTITIES(1:10),QUANTITIES.SPEC.ID(1:10),SURF.ID,DRAG.LAW,
4342 PROP.ID, &
4343 RADIATIVE.PROPERTY.TABLE='null',CNF.RAMP.ID='null',BREAKUP.CNF.RAMP.ID='null',DISTRIBUTION,BREAKUP.DISTRIBUTION
4344 CHARACTER(25) :: COLOR
4345 LOGICAL :: TARGET.ONLY,MASSLESS,STATIC,MONODISPERSE,BREAKUP,CHECK.DISTRIBUTION,TURBULENT.DISPERSION,&
4346 PERIODIC.X,PERIODIC.Y,PERIODIC.Z,SECOND.ORDER.PARTICLE.TRANSPORT
4347 TYPE(LAGRANGIAN.PARTICLE.CLASS.TYPE), POINTER :: LPC=>NULL()
4348 NAMELIST /PART/ AGE,BREAKUP,BREAKUP.CNF.RAMP.ID,BREAKUP.DISTRIBUTION,BREAKUP.GAMMA.D,BREAKUP.RATIO,&
4349 BREAKUP.SIGMA.D,CHECK.DISTRIBUTION,CNF.RAMP.ID,COLOR,COMPLEX.REFRACTIVE.INDEX,&
4350 CTRL.ID,DENSE.VOLUME.FRACTION,&
4351 DEVC.ID,DIAMETER,DISTRIBUTION,DRAG.COEFFICIENT,DRAG.LAW,FREE.AREA.FRACTION,FYI,GAMMA.D,HEAT.OF.COMBUSTION,&
4352 HORIZONTAL.VELOCITY.ID,INITIAL.TEMPERATURE,MASSLESS,MAXIMUM.DIAMETER,MINIMUM.DIAMETER,MONODISPERSE,&
4353 N.STRATA,ORIENTATION,PERMEABILITY,PERIODIC.X,PERIODIC.Y,PERIODIC.Z,POROUS.VOLUME.FRACTION,PROP.ID,QUANTITIES,&
4354 QUANTITIES.SPEC.ID,RADIATIVE.PROPERTY.TABLE,REAL.REFRACTIVE.INDEX,RGB,RUNNING.AVERAGE.FACTOR,&
4355 SAMPLING.FACTOR,SECOND.ORDER.PARTICLE.TRANSPORT,SIGMA.D,SPEC.ID,STATIC,&
4356 SURFACE.TENSION,SURF.ID,TARGET.ONLY,TURBULENT.DISPERSION,VERTICAL.VELOCITY
4357
4358 ! Determine total number of PART lines in the input file
4359
4360 REWIND(LU.INPUT) ; INPUT.FILE.LINE.NUMBER = 0
4361 N.LAGRANGIAN.CLASSES = 0
4362 N.LAGRANGIAN.CLASSES.READ = 0
4363
4364 COUNT.PART.LOOP: DO
4365 CALL CHECKREAD('PART',LU.INPUT,IOS)
4366 IF (IOS==1) EXIT COUNT.PART.LOOP
4367 READ(LU.INPUT,PART,END=219,ERR=220,IOSTAT=IOS)
4368 N.LAGRANGIAN.CLASSES.READ = N.LAGRANGIAN.CLASSES.READ + 1
4369 220 IF (IOS>0) THEN ; CALL SHUTDOWN('ERROR: Problem with PART line') ; RETURN ; ENDIF
4370 ENDDO COUNT.PART.LOOP
4371 219 REWIND(LU.INPUT) ; INPUT.FILE.LINE.NUMBER = 0
4372
4373 ! Add reserved INIT lines to account for devices for 'RADIATIVE HEAT FLUX GAS' or 'ADIABATIC SURFACE TEMPERATURE
4374 GAS'
4375
4376 N.LAGRANGIAN.CLASSES = N.LAGRANGIAN.CLASSES.READ
4377
4378 IF (TARGET.PARTICLES.INCLUDED) N.LAGRANGIAN.CLASSES = N.LAGRANGIAN.CLASSES + 1
4379
4380 ! Allocate the derived type array to hold information about the particle classes
4381
4382 IF (N.LAGRANGIAN.CLASSES>0) PARTICLE.FILE = .TRUE.
4383 ALLOCATE(LAGRANGIAN.PARTICLE.CLASS(N.LAGRANGIAN.CLASSES),STAT=IZERO)
4384 CALL ChkMemErr('READ','N.LAGRANGIAN.CLASSES',IZERO)
4385
4386 N.LP.ARRAY.INDICES = 0
4387 ILPC = 0
4388
4389 READ.PART.LOOP: DO N=1,N.LAGRANGIAN.CLASSES
4390
4391 ! Read the PART line from the input file or set up special PARTICLE.CLASS class for water PARTICLES or tracers
4392
4393 IF (N<=N.LAGRANGIAN.CLASSES.READ) THEN
4394 CALL CHECKREAD('PART',LU.INPUT,IOS)
4395 IF (IOS==1) EXIT READ.PART.LOOP
4396 CALL SET.PART.DEFAULTS
4397 READ(LU.INPUT,PART)
4398
4399 ELSE
4400
4401 ! Create a class of particles that is just a target
4402
4403 CALL SET.PART.DEFAULTS
4404 WRITE(ID,'(A)') 'RESERVED TARGET PARTICLE'
4405 TARGET.ONLY = .TRUE.
4406 STATIC = .TRUE.

```

## Source Code files for edited portions of FDS

```

4407 ORIENTATION(1:3,1) = (/1._EB , 0._EB , 0._EB/) ! This is just a dummy orientation
4408
4409 ENDIF
4410
4411 LPC => LAGRANGIAN.PARTICLE.CLASS(N)
4412
4413 ! Identify the different types of Lagrangian particles , like massless tracers , droplets , etc.
4414
4415 IF (SURF_ID/= 'null') THEN
4416 SOLID.PARTICLES = .TRUE.
4417 IF (CNF.RAMP_ID== 'null') MONODISPERSE = .TRUE.
4418 LPC%$SOLID.PARTICLE = .TRUE.
4419 IF (SAMPLING.FACTOR<=0) SAMPLING.FACTOR = 1
4420 IF (DIAMETER>0._EB) THEN
4421 WRITE(MESSAGE, '(A,I0,A)') 'ERROR: PART ',N,' cannot have both a specified DIAMETER and a SURF.ID.'
4422 CALL SHUTDOWN(MESSAGE) ; RETURN
4423 ENDIF
4424 ENDIF
4425
4426 IF (TARGET_ONLY) THEN
4427 LPC%$MASSLESS.TARGET = .TRUE.
4428 SURF_ID = 'MASSLESS TARGET'
4429 SOLID.PARTICLES = .TRUE.
4430 IF (CNF.RAMP_ID== 'null') MONODISPERSE = .TRUE.
4431 STATIC = .TRUE.
4432 IF (SAMPLING.FACTOR<=0) SAMPLING.FACTOR = 1
4433 ENDIF
4434
4435 IF (SPEC_ID/= 'null') THEN
4436 SURF_ID = 'DROPLET'
4437 LPC%$LIQUID.DROPLET = .TRUE.
4438 IF (SAMPLING.FACTOR<=0) SAMPLING.FACTOR = 10
4439 IF (DIAMETER<=0._EB .AND. CNF.RAMP_ID== 'null') THEN
4440 WRITE(MESSAGE, '(A,I0,A)') 'ERROR: PART ',N,' requires a specified DIAMETER.'
4441 CALL SHUTDOWN(MESSAGE) ; RETURN
4442 ENDIF
4443 IF (MASSLESS) THEN
4444 WRITE(MESSAGE, '(A)') 'ERROR: Cannot have MASSLESS=.TRUE. with evaporating PARTICLES'
4445 CALL SHUTDOWN(MESSAGE) ; RETURN
4446 ENDIF
4447 ENDIF
4448
4449 IF (MASSLESS) THEN
4450 LPC%$MASSLESS.TRACER = .TRUE.
4451 DIAMETER = 0._EB
4452 SURF_ID = 'MASSLESS TRACER'
4453 IF (SAMPLING.FACTOR<=0) SAMPLING.FACTOR = 1
4454 ENDIF
4455
4456 ! If particle class has no ID at this point , stop.
4457
4458 IF (SURF_ID== 'null') THEN
4459 WRITE(MESSAGE, '(A,I0,A)') 'ERROR: PART ',N,' needs a SURF.ID.'
4460 CALL SHUTDOWN(MESSAGE) ; RETURN
4461 ENDIF
4462
4463 ! If particle class has no ID at this point , stop.
4464
4465 DO I=1,10
4466 IF (QUANTITIES(I)== 'MASS FLUX' .AND. QUANTITIES.SPEC_ID(I)== 'null') THEN
4467 WRITE(MESSAGE, '(A)') 'ERROR: PART QUANTITIES 'MASS FLUX' requires QUANTITIES.SPEC_ID.'
4468 CALL SHUTDOWN(MESSAGE) ; RETURN
4469 ENDIF
4470 ENDDO
4471
4472 ! Set default colors for Smokeview. Water droplets are BLUE. Fuel droplets are YELLOW. Everything else is BLACK.
4473
4474 IF (TRIM(SPEC_ID)== 'WATER VAPOR') THEN
4475 IF (ANY(RGB<0) .AND. COLOR== 'null') COLOR= 'BLUE'
4476 ENDIF
4477
4478 IF (SIMPLE.CHEMISTRY) THEN
4479 IF (TRIM(SPEC_ID)==TRIM(REACTION(1)%FUEL)) THEN
4480 IF (ANY(RGB<0) .AND. COLOR== 'null') COLOR= 'YELLOW'
4481 ENDIF
4482 ENDIF
4483
4484 IF (ANY(RGB<0) .AND. COLOR== 'null') COLOR = 'BLACK'
4485
4486 IF (COLOR /= 'null') CALL COLOR2RGB(RGB,COLOR)
4487
4488 ! Determine if the SPEC_ID is OK
4489 LPC%$SPEC_ID = SPEC_ID
4490 IF (LPC%$LIQUID.DROPLET) THEN
4491 DO NN=1,N.TRACKED.SPECIES
4492 IF (TRIM(SPECIES.MIXTURE(NN)%ID)==TRIM(LPC%$SPEC_ID)) THEN
4493 LPC%$Z_INDEX = NN
4494 SPECIES.MIXTURE(NN)%EVAPORATING = .TRUE.

```



Source Code files for edited portions of FDS

```

4495 EXIT
4496 ENDDO
4497 ENDDO
4498 IF (LPC%Z_INDEX < 0) THEN
4499 WRITE(MESSAGE,'(A,A,A)') 'ERROR: PART SPEC.ID ',TRIM(LPC%SPEC.ID),' not found'
4500 CALL SHUTDOWN(MESSAGE) ; RETURN
4501 ENDDO
4502 IF (SPECIES_MIXTURE(LPC%Z_INDEX)%SINGLE_SPEC_INDEX < 0) THEN
4503 WRITE(MESSAGE,'(A,I0,A)') 'ERROR: PART line ',N,'. Particles cannot evaporate to a lumped species.'
4504 CALL SHUTDOWN(MESSAGE) ; RETURN
4505 ELSE
4506 LPC%Y_INDEX = SPECIES_MIXTURE(LPC%Z_INDEX)%SINGLE_SPEC_INDEX
4507 ENDDO
4508 IF (SPECIES(LPC%Y_INDEX)%DENSITY_LIQUID > 0._EB) LPC%DENSITY=SPECIES(LPC%Y_INDEX)%DENSITY_LIQUID
4509 ENDDO
4510
4511 ! Arrays for particle size distribution
4512
4513 IF (MONODISPERSE) THEN
4514 LPC%N_STRATA = 1
4515 ELSE
4516 LPC%N_STRATA = N_STRATA
4517 ENDDO
4518
4519 IF (DIAMETER > 0._EB .OR. CNF_RAMP_ID/= 'null') THEN
4520 ALLOCATE(LPC%CNF(0:NDC),STAT=IZERO)
4521 CALL ChkMemErr('READ','CNF',IZERO)
4522 ALLOCATE(LPC%CVF(0:NDC),STAT=IZERO)
4523 CALL ChkMemErr('READ','CVF',IZERO)
4524 ALLOCATE(LPC%R_CNF(0:NDC),STAT=IZERO)
4525 CALL ChkMemErr('READ','R_CNF',IZERO)
4526 ALLOCATE(LPC%IL_CNF(LPC%N_STRATA),STAT=IZERO)
4527 CALL ChkMemErr('READ','IL_CNF',IZERO)
4528 ALLOCATE(LPC%IU_CNF(LPC%N_STRATA),STAT=IZERO)
4529 CALL ChkMemErr('READ','IU_CNF',IZERO)
4530 ALLOCATE(LPC%W_CNF(LPC%N_STRATA),STAT=IZERO)
4531 CALL ChkMemErr('READ','W_CNF',IZERO)
4532 ENDDO
4533
4534 ! Arrays related to particle break-up model
4535
4536 IF (BREAKUP) THEN
4537 ALLOCATE(LPC%BREAKUP_CNF(0:NDC),STAT=IZERO)
4538 CALL ChkMemErr('READ','BREAKUP_CNF',IZERO)
4539 ALLOCATE(LPC%BREAKUP_R_CNF(0:NDC),STAT=IZERO)
4540 CALL ChkMemErr('READ','BREAKUP_R_CNF',IZERO)
4541 ALLOCATE(LPC%BREAKUP_CVF(0:NDC),STAT=IZERO)
4542 CALL ChkMemErr('READ','BREAKUP_CVF',IZERO)
4543 ENDDO
4544
4545 ! Radiative property table
4546
4547 IF (RADIATIVE_PROPERTY_TABLE /= 'null') THEN
4548 CALL GET_TABLE_INDEX(RADIATIVE_PROPERTY_TABLE, PART_RADIATIVE_PROPERTY, LPC%RADIATIVE_PROPERTY_INDEX)
4549 LPC%RADIATIVE_PROPERTY_TABLE_ID = RADIATIVE_PROPERTY_TABLE
4550 ELSE
4551 LPC%RADIATIVE_PROPERTY_INDEX = 0
4552 ENDDO
4553
4554 ! Assign property data to LAGRANGIAN.PARTICLE.CLASS class
4555
4556 LPC%ID = ID
4557 LPC%BREAKUP = BREAKUP
4558 LPC%BREAKUP_RATIO = BREAKUP_RATIO
4559 LPC%BREAKUP_GAMMA = BREAKUP_GAMMA
4560 IF (BREAKUP_SIGMA_D > 0._EB) THEN
4561 LPC%BREAKUP_SIGMA = BREAKUP_SIGMA_D
4562 ELSE
4563 ! per tech guide, sigma*gamma=1.15 smoothly joins Rosin-Rammler and lognormal distributions
4564 LPC%BREAKUP_SIGMA = 1.15._EB/BREAKUP_GAMMA
4565 ENDDO
4566 LPC%CTRL_ID = CTRL_ID
4567 LPC%DENSE_VOLUME_FRACTION = DENSE_VOLUME_FRACTION
4568 LPC%DEVC_ID = DEVC_ID
4569 LPC%TMP_INITIAL = INITIAL_TEMPERATURE + TMPM
4570 LPC%SAMPLING = SAMPLING_FACTOR
4571 LPC%RGB = RGB
4572 LPC%DIAMETER = DIAMETER*1.E-6.EB
4573 LPC%MEAN_DROPLET_VOLUME = FOTHP*(0.5.EB*LPC%DIAMETER)**3 ! recomputed for distributions
4574 LPC%MAXIMUM_DIAMETER = MAXIMUM_DIAMETER*1.E-6.EB
4575 IF (MINIMUM_DIAMETER < 0._EB) THEN
4576 MINIMUM_DIAMETER = 0.005.EB*DIAMETER
4577 ENDDO
4578 LPC%MINIMUM_DIAMETER = MINIMUM_DIAMETER*1.E-6.EB
4579 LPC%KILL_RADIUS = MINIMUM_DIAMETER*0.5.EB*1.E-6.EB*0.171.EB ! 0.171 ???
4580 LPC%MONODISPERSE = MONODISPERSE
4581 LPC%PERIODIC_X = PERIODIC_X
4582 LPC%PERIODIC_Y = PERIODIC_Y

```

Source Code files for edited portions of FDS

```

4583 LPC%PERIODIC_Z           = PERIODIC_Z
4584 LPC%PROP_ID             = PROP_ID
4585 LPC%QUANTITIES         = QUANTITIES
4586 LPC%QUANTITIES_SPEC_ID = QUANTITIES_SPEC_ID
4587 LPC%GAMMA              = GAMMAD
4588 IF ( SIGMA_D > 0._EB ) THEN
4589   LPC%SIGMA             = SIGMA_D
4590 ELSE
4591   LPC%SIGMA             = 1.15_EB/GAMMAD
4592 END IF
4593 LPC%DISTRIBUTION        = DISTRIBUTION
4594 LPC%CHECK_DISTRIBUTION = CHECK_DISTRIBUTION
4595 LPC%BREAKUP_DISTRIBUTION = BREAKUP.DISTRIBUTION
4596 LPC%CNF_RAMP_ID        = CNF_RAMP_ID
4597 LPC%BREAKUP.CNF_RAMP_ID = BREAKUP.CNF_RAMP_ID
4598
4599 IF (LPC%CNF_RAMP_ID/= 'null') THEN
4600 CALL GET_RAMP_INDEX(LPC%CNF_RAMP_ID, 'DIAMETER', LPC%CNF_RAMP_INDEX)
4601 ENDIF
4602 IF (LPC%BREAKUP.CNF_RAMP_ID/= 'null') THEN
4603 CALL GET_RAMP_INDEX(LPC%BREAKUP.CNF_RAMP_ID, 'DIAMETER', LPC%BREAKUP.CNF_RAMP_INDEX)
4604 ENDIF
4605
4606 LPC%TMP_INITIAL        = INITIAL_TEMPERATURE + TMFM
4607 LPC%REAL_REFRACTIVE_INDEX = REAL_REFRACTIVE_INDEX
4608 LPC%COMPLEX_REFRACTIVE_INDEX = COMPLEX_REFRACTIVE_INDEX
4609 IF (LPC%REAL_REFRACTIVE_INDEX <= 0._EB .OR. LPC%COMPLEX_REFRACTIVE_INDEX < 0._EB) THEN
4610 WRITE(MESSAGE, '(A,A)') 'Bad refractive index on PART line ', LPC%ID
4611 CALL SHUTDOWN(MESSAGE) ; RETURN
4612 ENDIF
4613 LPC%HEAT_OF_COMBUSTION = HEAT_OF_COMBUSTION*1000._EB
4614 LPC%FTPR              = FOTHP1*LPC%DENSITY
4615 LPC%KILL_MASS         = LPC%FTPR*LPC%KILL_RADIUS**3
4616 LPC%LIFETIME          = AGE
4617 LPC%TURBULENT_DISPERSION = TURBULENT_DISPERSION
4618 LPC%STATIC            = STATIC
4619 LPC%SPEC_ID           = SPEC_ID
4620 LPC%SURF_ID           = SURF_ID
4621 LPC%SURF_INDEX        = -1
4622 LPC%SURFACE_TENSION   = SURFACE_TENSION
4623 LPC%ADJUST_EVAPORATION = 1._EB ! If H.O.C>0. this parameter will have to be reset later
4624 LPC%VERTICAL_VELOCITY = VERTICAL_VELOCITY
4625 LPC%HORIZONTAL_VELOCITY = HORIZONTAL_VELOCITY
4626 LPC%SECOND_ORDER_PARTICLE_TRANSPORT = SECOND_ORDER_PARTICLE_TRANSPORT
4627 LPC%DRAG_COEFFICIENT  = DRAG_COEFFICIENT
4628 IF (DRAG_COEFFICIENT(1)>=0._EB .AND. DRAGLAW=='SPHERE') DRAGLAW = 'USER'
4629
4630 ! Count and process the number of orientations for the particle
4631
4632 LPC%N_ORIENTATION = 0
4633
4634 DO NN=1,10
4635 IF (ANY(ABS(ORIENTATION(1:3, NN))>TWO_EPSILON_EB)) LPC%N_ORIENTATION = LPC%N_ORIENTATION + 1
4636 ENDDO
4637 IF (TRIM(DRAGLAW)=='SCREEN' .AND. LPC%N_ORIENTATION/=1) THEN
4638 WRITE(MESSAGE, '(A,I0,A)') 'ERROR: PART line ', N, '. Must specify exactly one ORIENTATION for SCREEN drag law.'
4639 CALL SHUTDOWN(MESSAGE) ; RETURN
4640 ENDIF
4641
4642 IF (LPC%N_ORIENTATION>0) THEN
4643 ALLOCATE(LPC%SOLID_ANGLE(1:LPC%N_ORIENTATION))
4644 LPC%SOLID_ANGLE = 4._EB*PI
4645 LPC%ORIENTATION_INDEX = N_ORIENTATION.VECTOR + 1
4646 DO NN=1,LPC%N_ORIENTATION
4647 OR_TEMP(1:3) = ORIENTATION(1:3, NN)
4648 N_ORIENTATION.VECTOR = N_ORIENTATION.VECTOR + 1
4649 IF (N_ORIENTATION.VECTOR>UBOUND(ORIENTATION_VECTOR, DIM=2)) THEN
4650 ORIENTATION_VECTOR => REALLOCATE2D(ORIENTATION_VECTOR, 1, 3, 1, N_ORIENTATION.VECTOR+10)
4651 ENDIF
4652 IF (ALL(ABS(OR_TEMP(1:3))<TWO_EPSILON_EB)) THEN
4653 WRITE(MESSAGE, '(A,I0,A)') 'ERROR: PART line ', N, '. All components of ORIENTATION are zero.'
4654 CALL SHUTDOWN(MESSAGE) ; RETURN
4655 ENDIF
4656 ORIENTATION_VECTOR(1:3, N_ORIENTATION.VECTOR) = ORIENTATION(1:3, NN) / NORM2(OR_TEMP)
4657 ENDDO
4658 ENDIF
4659 LPC%FREE_AREA_FRACTION = FREE_AREA_FRACTION
4660
4661 SELECT CASE(DRAGLAW)
4662 CASE('SPHERE')
4663   LPC%DRAGLAW = SPHERE_DRAG
4664 CASE('CYLINDER')
4665   LPC%DRAGLAW = CYLINDER_DRAG
4666 CASE('USER')
4667   LPC%DRAGLAW = USER_DRAG
4668 CASE('SCREEN')
4669   LPC%DRAGLAW = SCREEN_DRAG
4670 LPC%PERMEABILITY(1:3) = 3.44E-9_EB*LPC%FREE_AREA_FRACTION**1.6_EB

```

Source Code files for edited portions of FDS

```

4671 LPC%DRAG.COEFFICIENT(1:3) = 4.30E-2.EB*LPC%FREE.AREA.FRACTION**2.13.EB
4672 CASE('POROUS MEDIA')
4673 IF (ANY(DRAG.COEFFICIENT<TWO.EPSILON.EB) .OR. ANY(PERMEABILITY<TWO.EPSILON.EB)) THEN
4674 WRITE(MESSAGE,'(A,I0,A)') 'ERROR: PART line ',N,&
4675 ' For POROUS MEDIA must specify all compoents for DRAG.COEFFICIENT and PERMIABILITY.'
4676 CALL SHUTDOWN(MESSAGE) ; RETURN
4677 ENDF
4678 LPC%DRAGLAW = POROUS.DRAG
4679 LPC%PERMEABILITY = PERMEABILITY
4680 LPC%POROUS.VOLUME.FRACTION = POROUS.VOLUME.FRACTION
4681 CASE DEFAULT
4682 WRITE(MESSAGE,'(A)') 'ERROR: unrecognized drag law on PART line'
4683 CALL SHUTDOWN(MESSAGE) ; RETURN
4684 END SELECT
4685
4686 ! Determine the number of slots to create in the particle evaporation and radiation arrays
4687
4688 IF (LPC%LIQUID.DROPLET .OR. LPC%SOLID.PARTICLE) THEN
4689 N_LP_ARRAY_INDICES = N_LP_ARRAY_INDICES + 1
4690 LPC%ARRAY_INDEX = N_LP_ARRAY_INDICES
4691 IF (LPC%SOLID.PARTICLE .AND. RUNNING.AVERAGE.FACTOR<0.EB) LPC%RUNNING.AVERAGE.FACTOR = 0.0.EB
4692 IF (LPC%LIQUID.DROPLET .AND. RUNNING.AVERAGE.FACTOR<0.EB) LPC%RUNNING.AVERAGE.FACTOR = 0.5.EB
4693 ENDF
4694
4695 ENDDO READ.PART.LOOP
4696
4697 ! Allocate radiation arrays
4698
4699 PLOOP2: DO ILPC=1,N,LAGRANGIAN.CLASSES
4700 LPC=>LAGRANGIAN.PARTICLE.CLASS(ILPC)
4701 IF (LPC%LIQUID.DROPLET) THEN
4702 ALLOCATE(LPC%WQABS(0:MIE.NDG,1:NUMBER.SPECTRAL.BANDS))
4703 CALL ChkMemErr('INIT','WQABS',IZERO)
4704 LPC%WQABS = 0.EB
4705 ALLOCATE(LPC%WQSCA(0:MIE.NDG,1:NUMBER.SPECTRAL.BANDS))
4706 CALL ChkMemErr('INIT','WQSCA',IZERO)
4707 LPC%WQSCA = 0.EB
4708 ALLOCATE(LPC%R50(0:MIE.NDG))
4709 CALL ChkMemErr('INIT','R50',IZERO)
4710 LPC%R50 = 0.EB
4711 ENDF
4712 ENDDO PLOOP2
4713
4714 ! Determine output quantities
4715
4716 DO ILPC=1,N,LAGRANGIAN.CLASSES
4717 LPC=>LAGRANGIAN.PARTICLE.CLASS(ILPC)
4718 LPC%N.QUANTITIES = 0
4719 IF (ANY(LPC%QUANTITIES/= 'null')) THEN
4720 QUANTITIES.LOOP: DO N=1,10
4721 IF (LPC%QUANTITIES(N)='null') CYCLE QUANTITIES.LOOP
4722 LPC%N.QUANTITIES = LPC%N.QUANTITIES + 1
4723 CALL GET.QUANTITY.INDEX(LPC%SMOKEVIEW.LABEL(LPC%N.QUANTITIES),LPC%SMOKEVIEW.BAR.LABEL(LPC%N.QUANTITIES), &
4724 LPC%QUANTITIES.INDEX(LPC%N.QUANTITIES),LDUM(1), &
4725 LPC%QUANTITIES.Y.INDEX(LPC%N.QUANTITIES),LPC%QUANTITIES.Z.INDEX(LPC%N.QUANTITIES),&
4726 LDUM(4),LDUM(5),LDUM(6),LDUM(7), 'PART', &
4727 LPC%QUANTITIES(N), 'null',LPC%QUANTITIES.SPEC.ID(N), 'null', 'null', 'null', 'null')
4728 ENDDO QUANTITIES.LOOP
4729 ENDF
4730 ENDDO
4731
4732 CONTAINS
4733
4734
4735 SUBROUTINE SET.PART.DEFAULTS
4736
4737 BREAKUP = .FALSE.
4738 BREAKUP.RATIO = 3.EB/7.EB ! ratio of child Sauter mean to parent size in Bag breakup regime
4739 BREAKUP.GAMMA.D = 2.4.EB
4740 BREAKUP.SIGMA.D = -99999.9.EB
4741 CTRL.ID = 'null'
4742 DENSE.VOLUME.FRACTION = 1.E-5.EB ! Limiting volume fraction for drag reduction
4743 DEVC.ID = 'null'
4744 INITIAL.TEMPERATURE = TMPA - TMFM ! C
4745 HEAT.OF.COMBUSTION = -1.EB ! kJ/kg
4746 DIAMETER = -1.EB !
4747 MAXIMUM.DIAMETER = 1.E9.EB ! microns, meant to be infinitely large and not used
4748 MINIMUM.DIAMETER = -1.EB ! microns, below which the PARTICLE evaporates in one time step
4749 MONODISPERSE = .FALSE.
4750 N.STRATA = 6
4751 GAMMAD = 2.4.EB
4752 SIGMA.D = -99999.9.EB
4753 AGE = 1.E6.EB ! s
4754 ID = 'null'
4755 PERIODIC.X = .FALSE.
4756 PERIODIC.Y = .FALSE.
4757 PERIODIC.Z = .FALSE.
4758 PROP.ID = 'null'

```

```

4759 | ORIENTATION           = 0. _EB
4760 | QUANTITIES            = 'null'
4761 | QUANTITIES_SPEC_ID   = 'null'
4762 | RADIATIVE_PROPERTY_TABLE = 'null'
4763 | RGB                   = -1
4764 | SPEC_ID               = 'null'
4765 | SURF_ID               = 'null'
4766 | SURFACE_TENSION      = 72.8E-3_EB ! N/m, applies for water
4767 | COLOR                 = 'null'
4768 | SAMPLING_FACTOR      = -1
4769 | STATIC                = .FALSE.
4770 | MASSLESS              = .FALSE.
4771 | TARGET_ONLY          = .FALSE.
4772 | TURBULENT_DISPERSION = .FALSE.
4773 | REAL_REFRACTIVE_INDEX = 1.33_EB
4774 | RUNNING_AVERAGE_FACTOR = -1. _EB
4775 | COMPLEX_REFRACTIVE_INDEX = 0.01_EB
4776 | VERTICAL_VELOCITY    = 0.5_EB
4777 | HORIZONTAL_VELOCITY  = 0.2_EB
4778 | DRAGLAW              = 'SPHERE'
4779 | DRAG_COEFFICIENT     = -1. _EB
4780 | PERMEABILITY         = -1. _EB
4781 | DISTRIBUTION         = 'ROSIN-RAMMLER-LOGNORMAL'
4782 | CNF_RAMP_ID          = 'null'
4783 | CHECK_DISTRIBUTION   = .FALSE.
4784 | BREAKUP_DISTRIBUTION = 'ROSIN-RAMMLER-LOGNORMAL'
4785 | BREAKUP_CNF_RAMP_ID = 'null'
4786 | FREE_AREA_FRACTION  = 0.5_EB
4787 | SECOND_ORDER_PARTICLE_TRANSPORT = .FALSE.
4788 |
4789 | END SUBROUTINE SET_PART_DEFAULTS
4790 |
4791 | END SUBROUTINE READ_PART
4792 |
4793 |
4794 | SUBROUTINE PROC_PART
4795 |
4796 | USE PROPERTY_DATA, ONLY: JANAF_TABLE_LIQUID
4797 | USE MATH_FUNCTIONS, ONLY: EVALUATE_RAMP
4798 | INTEGER :: N, NN, J, ITMP, LMELT, L_BOIL
4799 | REAL(EB) :: H.L., H.V., CPBAR, H.G.S., H.L.REF, TMP_REF, TMP_MELT, TMP_V, TMP_WGT, DENSITY, MASS, VOLUME, R.O., R.I., &
4800 | MU_LIQUID, K_LIQUID, BETA_LIQUID
4801 | TYPE(LAGRANGIAN_PARTICLE_CLASS_TYPE), POINTER :: LPC=>NULL()
4802 | TYPE(SPECIES_TYPE), POINTER :: SS=>NULL()
4803 | TYPE(SURFACE_TYPE), POINTER :: SF=>NULL()
4804 |
4805 | IF (N_LAGRANGIAN_CLASSES == 0) RETURN
4806 |
4807 | PART_LOOP: DO N=1, N_LAGRANGIAN_CLASSES
4808 |
4809 | LPC => LAGRANGIAN_PARTICLE_CLASS(N)
4810 | SF => SURFACE(LPC%SURF_INDEX)
4811 |
4812 | ! Assign device or controller
4813 |
4814 | CALL SEARCH_CONTROLLER('PART', LPC%CTRL_ID, LPC%DEV_ID, LPC%DEV_INDEX, LPC%CTRL_INDEX, N)
4815 |
4816 | ! Get density if the particles are liquid droplets or have mass
4817 |
4818 | IF (LPC%LIQUID_DROPLET) THEN
4819 | CALL JANAF_TABLE_LIQUID(1, CPBAR, H.V., H.L., TMP_REF, TMP_MELT, TMP_V, SPECIES(LPC%Y_INDEX)%ID, LPC%FUEL, DENSITY, &
4820 | MU_LIQUID, K_LIQUID, BETA_LIQUID)
4821 | IF (LPC%DENSITY < 0. _EB) THEN
4822 | LPC%DENSITY = DENSITY
4823 | LPC%FTPR = FOTH*PI*DENSITY
4824 | LPC%KILL_MASS = LPC%FTPR*LPC%KILL_RADIUS**3
4825 | IF (SPECIES(LPC%Y_INDEX)%DENSITY_LIQUID < 0. _EB) SPECIES(LPC%Y_INDEX)%DENSITY_LIQUID = DENSITY
4826 | ENDF
4827 | IF (SPECIES(LPC%Y_INDEX)%MU_LIQUID < 0. _EB) SPECIES(LPC%Y_INDEX)%MU_LIQUID = MU_LIQUID
4828 | IF (SPECIES(LPC%Y_INDEX)%K_LIQUID < 0. _EB) SPECIES(LPC%Y_INDEX)%K_LIQUID = K_LIQUID
4829 | IF (SPECIES(LPC%Y_INDEX)%BETA_LIQUID < 0. _EB) SPECIES(LPC%Y_INDEX)%BETA_LIQUID = BETA_LIQUID
4830 | ENDF
4831 |
4832 | IF (SP%THERMALLY_THICK) THEN
4833 | MASS = 0. _EB
4834 | VOLUME = 0. _EB
4835 | DO NN=1, SP%N_CELLS_INI
4836 | SELECT CASE (SP%GEOMETRY)
4837 | CASE (SURF_CARTESIAN)
4838 | MASS = MASS + (SP%X_S(NN)-SP%X_S(NN-1))*SUM(SP%RHO_0(NN, 1: SP%N_MATL))
4839 | VOLUME = VOLUME + (SP%X_S(NN)-SP%X_S(NN-1))
4840 | CASE (SURF_CYLINDRICAL)
4841 | R.I = SP%INNER_RADIUS + SP%THICKNESS - SP%X_S(NN)
4842 | R.O = SP%INNER_RADIUS + SP%THICKNESS - SP%X_S(NN-1)
4843 | MASS = MASS + (R.O**2 - R.I**2)*SUM(SP%RHO_0(NN, 1: SP%N_MATL))
4844 | VOLUME = VOLUME + (R.O**2 - R.I**2)
4845 | CASE (SURF_SPHERICAL)
4846 | R.I = SP%INNER_RADIUS + SP%THICKNESS - SP%X_S(NN)

```

Source Code files for edited portions of FDS

```

4847 R.O = SP%INNER_RADIUS + SP%THICKNESS - SP%X_S(NN-1)
4848 MASS = MASS + (R.O**3-R.I**3)*SUM(SP%RHO.0(NN,1:SP%N.MATL))
4849 VOLUME = VOLUME + (R.O**3-R.I**3)
4850 END SELECT
4851 ENDDO
4852 LPC%DENSITY = MASS/VOLUME
4853 LPC%FTPR = FOTH*PI*LPC%DENSITY
4854 ENDIF
4855
4856 IF (SP%HEAT.TRANSFER.MODEL>0) THEN ; CALL SHUTDOWN('ERROR: HEAT.TRANSFER.MODEL not appropriate for PART') ;
      RETURN ; ENDIF
4857
4858 ! Set the flag to do particle exchanges between meshes
4859
4860 OMESH.PARTICLES=.TRUE.
4861
4862 ! Only process DROPLETs
4863
4864 SURF_OR_SPEC: IF (LPC%SURF_INDEX==DROPLET_SURF_INDEX) THEN
4865
4866 SS => SPECIES(LPC%Y_INDEX)
4867
4868 IZERO = 0
4869 IF (.NOT.ALLOCATED(SS%C.P.L)) ALLOCATE(SS%C.P.L(0:5000),STAT=IZERO)
4870 CALL ChkMemErr('PROC.PART', 'SS%C.P.L', IZERO)
4871 SS%C.P.L=SS%SPECIFIC.HEAT.LIQUID
4872 IF (.NOT.ALLOCATED(SS%C.P.L.BAR)) ALLOCATE(SS%C.P.L.BAR(0:5000),STAT=IZERO)
4873 CALL ChkMemErr('PROC.PART', 'SS%C.P.L.BAR', IZERO)
4874 IF (.NOT.ALLOCATED(SS%H.L)) ALLOCATE(SS%H.L(0:5000),STAT=IZERO)
4875 CALL ChkMemErr('PROC.PART', 'SS%H.L', IZERO)
4876 IF (.NOT.ALLOCATED(SS%H.V)) ALLOCATE(SS%H.V(0:5000),STAT=IZERO)
4877 CALL ChkMemErr('PROC.PART', 'SS%H.V', IZERO)
4878
4879 TMP_REF = -1._EB
4880 TMP_MELT = -1._EB
4881 TMP_V = -1._EB
4882 DO J = 1, 5000
4883 IF (SS%C.P.L(J) > 0._EB) THEN
4884 SS%H.L(J) = (REAL(J,EB)-SS%TMP_MELT)*SS%C.P.L(J)
4885 IF (J==1) THEN
4886 CALL JANAF.TABLE.LIQUID (J,CPBAR,H.V,H.L,TMP_REF,TMP_MELT,TMP_V,SS%PROP_ID,LPC%FUEL,DENSITY,&
4887 MULLIQUID,K.LIQUID,BETA.LIQUID)
4888 IF (SS%H.V.REFERENCE.TEMPERATURE < 0._EB) SS%H.V.REFERENCE.TEMPERATURE=TMP_REF
4889 IF (SS%TMP_V < 0._EB) SS%TMP_V = TMP_V
4890 IF (SS%TMP_V < 0._EB) THEN
4891 WRITE(MESSAGE,'(A,A,A)') 'ERROR: SPEC ',TRIM(SS%D),' requires a VAPORIZATION.TEMPERATURE'
4892 CALL SHUTDOWN(MESSAGE) ; RETURN
4893 ENDIF
4894 IF (SS%TMP_MELT < 0._EB) SS%TMP_MELT = TMP_MELT
4895 IF (SS%TMP_MELT < 0._EB) THEN
4896 WRITE(MESSAGE,'(A,A,A)') 'ERROR: PARTicle class ',TRIM(SS%D),' requires a TMP_MELT'
4897 CALL SHUTDOWN(MESSAGE) ; RETURN
4898 ENDIF
4899 IF (LPC%DENSITY < 0._EB) LPC%DENSITY = DENSITY
4900 IF (LPC%DENSITY < 0._EB) THEN
4901 WRITE(MESSAGE,'(A,A,A)') 'ERROR: PARTicle class ',TRIM(SS%D),' requires a density'
4902 CALL SHUTDOWN(MESSAGE) ; RETURN
4903 ENDIF
4904 LPC%FTPR = FOTH*PI*LPC%DENSITY
4905 ENDIF
4906 ELSE
4907 CALL JANAF.TABLE.LIQUID (J,SS%C.P.L(J),H.V,H.L,TMP_REF,TMP_MELT,TMP_V,SS%D,LPC%FUEL,DENSITY,&
4908 MULLIQUID,K.LIQUID,BETA.LIQUID)
4909 IF (SS%RAMP_CP.L.INDEX>0) SS%C.P.L(J) = EVALUATE.RAMP(REAL(J,EB),1._EB,SS%RAMP_CP.L.INDEX)*1000._EB
4910 IF (J==1) THEN
4911 IF (SS%H.V.REFERENCE.TEMPERATURE < 0._EB) SS%H.V.REFERENCE.TEMPERATURE=TMP_REF
4912 IF (SS%TMP_V < 0._EB) SS%TMP_V = TMP_V
4913 IF (SS%TMP_V < 0._EB) THEN
4914 WRITE(MESSAGE,'(A,A,A)') 'ERROR: PARTicle class ',TRIM(SS%D),' requires a TMP.V'
4915 CALL SHUTDOWN(MESSAGE) ; RETURN
4916 ENDIF
4917 IF (SS%TMP_MELT < 0._EB) SS%TMP_MELT = TMP_MELT
4918 IF (SS%TMP_MELT < 0._EB) THEN
4919 WRITE(MESSAGE,'(A,A,A)') 'ERROR: PARTicle class ',TRIM(SS%D),' requires a TMP_MELT'
4920 CALL SHUTDOWN(MESSAGE) ; RETURN
4921 ENDIF
4922 IF (LPC%DENSITY < 0._EB) LPC%DENSITY = DENSITY
4923 IF (LPC%DENSITY < 0._EB) THEN
4924 WRITE(MESSAGE,'(A,A,A)') 'ERROR: PARTicle class ',TRIM(SS%D),' requires a density'
4925 CALL SHUTDOWN(MESSAGE) ; RETURN
4926 ENDIF
4927 LPC%FTPR = FOTH*PI*LPC%DENSITY
4928 IF (SS%C.P.L(J) < 0._EB) THEN
4929 WRITE(MESSAGE,'(A,A,A)') 'ERROR: PARTicle class ',TRIM(SS%D),' requires CP, H.V, TMP_MELT, TMP.V, and T_REF'
4930 CALL SHUTDOWN(MESSAGE) ; RETURN
4931 ENDIF
4932 SS%H.L(J) = H.L + SS%C.P.L(J)
4933 ELSE

```

Source Code files for edited portions of FDS

```

4934 SS%HLL(J) = SS%HLL(J-1) + 0.5_EB*(SS%C.P.L(J)+SS%C.P.L(J-1))
4935 ENDIF
4936 ENDIF
4937 END DO
4938
4939 SS%C.P.L(0) = SS%C.P.L(1)
4940 SS%HLL(0) = SS%HLL(1) - SS%C.P.L(1)
4941
4942 ! Adjust liquid H.L to force H.V at H.V.REFERENCE.TEMPERATURE
4943
4944 IF (SS%HEAT.OF.VAPORIZATION > 0._EB) H.V = SS%HEAT.OF.VAPORIZATION
4945 ITMP = INT(SS%H.V.REFERENCE.TEMPERATURE)
4946 TMP.WGT = SS%H.V.REFERENCE.TEMPERATURE - REAL(ITMP,EB)
4947 H.L.REF = SS%H.L(ITMP)+TMP.WGT*(SS%H.L(ITMP+1)-SS%H.L(ITMP))
4948 H.G.S.REF=(CPBAR.Z(ITMP,LPC%Z.INDEX)+TMP.WGT*(CPBAR.Z(ITMP+1,LPC%Z.INDEX)-CPBAR.Z(ITMP,LPC%Z.INDEX)))*&
4949 SS%H.V.REFERENCE.TEMPERATURE
4950 SS%H.L = SS%H.L + (H.G.S.REF - H.L.REF) - H.V
4951 I.MELT = INT(SS%TMP.MELT) - 1
4952 I.BOIL = INT(SS%TMP.V) + 1
4953
4954 ! Determine the properties of the PARTICLE
4955
4956 DO J=1,5000
4957 H.G.S = CPBAR.Z(J,LPC%Z.INDEX)*REAL(J,EB)
4958 SS%HLV(J) = H.G.S - SS%HLL(J)
4959 IF (SS%HLV(J) < 0._EB .AND. J > I.MELT .AND. J < I.BOIL) THEN
4960 WRITE(MESSAGE,'(A,A,A)') 'ERROR: PARTicle class ',TRIM(SS%ID), ' H.V(T) < 0. Check inputs for C.P gas and C.P
liquid'
4961 CALL SHUTDOWN(MESSAGE) ; RETURN
4962 ENDIF
4963 IF (J==1) THEN
4964 SS%C.P.L.BAR(J) = SS%H.L(J)
4965 ELSE
4966 SS%C.P.L.BAR(J) = SS%H.L(J) / REAL(J,EB)
4967 ENDIF
4968 ENDDO
4969
4970 SS%HLV(0) = SS%HLV(1)
4971 SS%C.P.L.BAR(0) = SS%H.L(1)
4972 TMPMIN = MIN(TMPMIN,SS%TMP.MELT)
4973
4974 SS%PR.LIQUID = SS%MU.LIQUID*SS%C.P.L(NINT(TMPA))/SS%K.LIQUID
4975
4976 ENDIF SURF.OR.SPEC
4977
4978 ! Adjust the evaporation rate of fuel PARTICLES to account for difference in HoC.
4979
4980 IF (LPC%HEAT.OF.COMBUSTION > 0._EB) LPC%ADJUST.EVAPORATION = LPC%HEAT.OF.COMBUSTION/REACTION(1)%
HEAT.OF.COMBUSTION
4981
4982 ENDDO PART.LOOP
4983
4984 END SUBROUTINE PROC.PART
4985
4986 SUBROUTINE READ.PROP
4987
4988 USE DEVICE.VARIABLES
4989 USE MATH.FUNCTIONS, ONLY : GET.RAMP.INDEX,GET.TABLE.INDEX
4990 USE PHYSICAL.FUNCTIONS, ONLY : SPRAY.ANGLE.DISTRIBUTION
4991 REAL(EB) :: ACTIVATION.OBSCURATION,ACTIVATION.TEMPERATURE,ALPHA.C,ALPHA.E,BETA.C,BETA.E, &
4992 BEAD.DIAMETER, BEAD.EMISSIVITY, BEAD.SPECIFIC.HEAT, BEAD.DENSITY, &
4993 BEAD.HEAT.TRANSFER.COEFFICIENT, HEAT.TRANSFER.COEFFICIENT, DIAMETER, DENSITY, SPECIFIC.HEAT, &
4994 C.FACTOR, CHARACTERISTIC.VELOCITY, ORIFICE.DIAMETER, DROPLET.VELOCITY, EMISSIVITY, &
4995 PARTICLE.VELOCITY, FLOW.RATE, FLOW.TAU, GAUGE.EMISSIVITY, GAUGE.TEMPERATURE, INITIAL.TEMPERATURE, K.FACTOR,&
4996 LENGTH, SPRAY.ANGLE(2,2), OFFSET, OPERATING.PRESSURE, RTI, PDPA.START, PDPA.END, PDPA.RADIUS, MASS.FLOW.RATE,&
4997 SPRAY.PATTERN.MU, SPRAY.PATTERN.BETA, &
4998 PDPA.HISTOGRAM.LIMITS(2), &
4999 P0,PX(3),PXX(3,3)
5000 EQUIVALENCE(PARTICLE.VELOCITY, DROPLET.VELOCITY)
5001 EQUIVALENCE(EMISSIVITY, BEAD.EMISSIVITY)
5002 EQUIVALENCE(HEAT.TRANSFER.COEFFICIENT, BEAD.HEAT.TRANSFER.COEFFICIENT)
5003 EQUIVALENCE(DENSITY, BEAD.DENSITY)
5004 EQUIVALENCE(DIAMETER, BEAD.DIAMETER)
5005 EQUIVALENCE(SPECIFIC.HEAT, BEAD.SPECIFIC.HEAT)
5006 INTEGER :: I, N, NN, PDPA.M, PDPA.N, PARTICLES.PER.SECOND, VELOCITY.COMPONENT, PDPA.HISTOGRAM.NBINS, FED.ACTIVITY
5007 LOGICAL :: PDPA.INTEGRATE, PDPA.NORMALIZE, PDPA.HISTOGRAM, PDPA.HISTOGRAM.CUMULATIVE
5008 EQUIVALENCE(LENGTH, ALPHA.C)
5009 CHARACTER(LABEL.LENGTH) :: SMOKEVIEW.ID(SMOKEVIEW.OBJECTS.DIMENSION), QUANTITY='null', PART.ID='null', FLOW.RAMP='
null', &
5010 SPRAY.PATTERN.TABLE='null', SPEC.ID='null', &
5011 PRESSURE.RAMP='null', SMOKEVIEW.PARAMETERS(SMOKEVIEW.OBJECTS.DIMENSION), &
5012 SPRAY.PATTERN.SHAPE='GAUSSIAN'
5013 TYPE (PROPERTY.TYPE), POINTER :: PY=>NULL()
5014
5015 NAMELIST /PROP/ ACTIVATION.OBSCURATION,ACTIVATION.TEMPERATURE,ALPHA.C,ALPHA.E,BETA.C,BETA.E, FED.ACTIVITY,&
5016 CHARACTERISTIC.VELOCITY, C.FACTOR, DENSITY, DIAMETER, EMISSIVITY, FLOW.RAMP, FLOW.RATE, FLOW.TAU,&
5017 GAUGE.EMISSIVITY, GAUGE.TEMPERATURE, HEAT.TRANSFER.COEFFICIENT, ID,&

```

## Source Code files for edited portions of FDS

```

5019 INITIAL_TEMPERATURE,K_FACTOR,LENGTH,MASS_FLOW_RATE,OFFSET,OPERATING_PRESSURE,ORIFICE_DIAMETER,P0,&
5020 PARTICLES_PER_SECOND,&
5021 PARTICLE_VELOCITY,PART_ID,PDPA_END,PDPA_HISTOGRAM, PDPA_HISTOGRAM_LIMITS,PDPA_HISTOGRAM_NBINS,&
5022 PDPA_HISTOGRAM_CUMULATIVE,PDPA_INTEGRATE,PDPA_M,PDPA_N,PDPA_NORMALIZE,PDPA_RADIUS,&
5023 PDPA_START,PRESSURE_RAMP,PX,PXX,QUANTITY,RTI,SMOKEVIEW_ID,SMOKEVIEW_PARAMETERS,SPEC_ID,SPECIFIC_HEAT,SPRAY_ANGLE
    ,&
5024 SPRAY_PATTERN_BETA,SPRAY_PATTERN_MU,SPRAY_PATTERN_SHAPE,SPRAY_PATTERN_TABLE,VELOCITY_COMPONENT,&
5025 BEAD_EMISSIVITY, BEAD_HEAT_TRANSFER_COEFFICIENT, DROPLET_VELOCITY,& ! Backward compatability
5026 BEAD_DENSITY, BEAD_DIAMETER, BEAD_SPECIFIC_HEAT ! Backward compatability
5027
5028 ! Count the PROP lines in the input file. Note how many of these are cables.
5029
5030 N_PROP=0
5031 REWIND(LU_INPUT) ; INPUT_FILE_LINE_NUMBER = 0
5032 COUNT_PROP_LOOP: DO
5033 CALL CHECKREAD('PROP',LU_INPUT,IOS)
5034 IF (IOS==1) EXIT COUNT_PROP_LOOP
5035 READ(LU_INPUT,PROP,ERR=34,IOSTAT=IOS)
5036 N_PROP = N_PROP + 1
5037 34 IF (IOS>0) THEN
5038 WRITE(MESSAGE,'(A,I0,A,I0)') 'ERROR: Problem with PROP number', N_PROP+1,' , line number',INPUT_FILE_LINE_NUMBER
5039 CALL SHUTDOWN(MESSAGE) ; RETURN
5040 ENDIF
5041 ENDDO COUNT_PROP_LOOP
5042
5043 ! Allocate the PROPERTY derived types
5044
5045 ALLOCATE(PROPERTY(0:N_PROP),STAT=IZERO)
5046 CALL ChkMemErr('READ','PROPERTY',IZERO)
5047
5048 ! Read the PROP lines in the order listed in the input file
5049
5050 REWIND(LU_INPUT) ; INPUT_FILE_LINE_NUMBER = 0
5051
5052 READ_PROP_LOOP: DO N=0,N_PROP
5053
5054 CALL CHECKREAD('PROP',LU_INPUT,IOS) ! Look for PROP lines in the input file
5055 CALL SET_PROP_DEFAULTS ! Reset PROP NAMELIST parameters to default values
5056 IF (N > 0) READ(LU_INPUT,PROP)
5057
5058 ! Pack PROP parameters into the appropriate property derived types
5059
5060 PY => PROPERTY(N)
5061 PY%ACTIVATION_OBSCURATION = ACTIVATION_OBSCURATION
5062 PY%ACTIVATION_TEMPERATURE = ACTIVATION_TEMPERATURE ! NOTE: Act_Temp remains in degrees C. It is just a
    SETPOINT.
5063 PY%ALPHA_C = ALPHA_C
5064 PY%ALPHA_E = ALPHA_E
5065 PY%BETA_C = BETA_C
5066 PY%BETA_E = BETA_E
5067 PY%DENSITY = DENSITY
5068 PY%DIAMETER = DIAMETER
5069 PY%EMISSIVITY = EMISSIVITY
5070 PY%HEAT_TRANSFER_COEFFICIENT= HEAT_TRANSFER_COEFFICIENT
5071 PY%SPECIFIC_HEAT = SPECIFIC_HEAT*1000..EB/TIME.SHRIK_FACTOR
5072 PY%C_FACTOR = C_FACTOR
5073 PY%CHARACTERISTIC_VELOCITY = CHARACTERISTIC_VELOCITY
5074 PY%GAUGE_EMISSIVITY = GAUGE_EMISSIVITY
5075 PY%GAUGE_TEMPERATURE = GAUGE_TEMPERATURE + TFMF
5076 PY%ID = ID
5077 PY%INITIAL_TEMPERATURE = INITIAL_TEMPERATURE + TFMF
5078 PY%PARTICLES_PER_SECOND = PARTICLES_PER_SECOND
5079 PY%OFFSET = OFFSET
5080 PY%OPERATING_PRESSURE = OPERATING_PRESSURE
5081 PY%PART_ID = PART_ID
5082 PY%QUANTITY = QUANTITY
5083 IF (PY%PART_ID/='null' .AND. PY%QUANTITY == 'null' ) PY%QUANTITY = 'NOZZLE'
5084 PY%RTI = RTI
5085 IF (SMOKEVIEW_ID(1)/='null') THEN
5086 PY%SMOKEVIEW_ID = SMOKEVIEW_ID
5087 PY%N_SMOKEVIEW_IDS = 0
5088 DO NN=1,SMOKEVIEW_OBJECTS_DIMENSION
5089 IF (SMOKEVIEW_ID(NN)/='null') PY%N_SMOKEVIEW_IDS = PY%N_SMOKEVIEW_IDS + 1
5090 ENDDO
5091 ELSE
5092 PY%N_SMOKEVIEW_IDS = 1
5093 SELECT CASE(PY%QUANTITY)
5094 CASE DEFAULT
5095 PY%SMOKEVIEW_ID(1) = 'sensor'
5096 CASE('SPRINKLER LINK TEMPERATURE')
5097 PY%SMOKEVIEW_ID(1) = 'sprinkler.pendent'
5098 CASE('NOZZLE')
5099 PY%SMOKEVIEW_ID(1) = 'nozzle'
5100 CASE('LINK TEMPERATURE')
5101 PY%SMOKEVIEW_ID(1) = 'heat.detector'
5102 CASE('spot obscuration','CHAMBER_OBSCURATION')
5103 PY%SMOKEVIEW_ID(1) = 'smoke.detector'
5104 CASE('THERMOCOUPLE')

```



Source Code files for edited portions of FDS

```

5105 PY%SMOKEVIEW_ID(1) = 'thermocouple'
5106 END SELECT
5107 ENDIF
5108 PY%SMOKEVIEW_PARAMETERS = SMOKEVIEW_PARAMETERS
5109 PY%N.SMOKEVIEW_PARAMETERS = 0
5110 DO I=1,SMOKEVIEW_OBJECTS_DIMENSION
5111 IF (PY%SMOKEVIEW_PARAMETERS(I)/='null') PY%N.SMOKEVIEW_PARAMETERS = PY%N.SMOKEVIEW_PARAMETERS + 1
5112 ENDDO
5113 PY%SPEC_ID = SPEC_ID
5114 IF (PART_ID/='null' .AND. SPRAY_PATTERN_TABLE /= 'null') THEN
5115 CALL GET_TABLE_INDEX(SPRAY_PATTERN_TABLE,SPRAY_PATTERN,PY%SPRAY_PATTERN_INDEX)
5116 PY%TABLE_ID = SPRAY_PATTERN_TABLE
5117 ELSE
5118 PY%SPRAY_PATTERN_INDEX = 0
5119 ENDIF
5120 PY%SPRAY_ANGLE = SPRAY_ANGLE*PI/180..EB
5121 IF (ANY(PY%SPRAY_ANGLE(1:2,2)<0)) PY%SPRAY_ANGLE(1:2,2)=PY%SPRAY_ANGLE(1:2,1)
5122 SPRAY_PATTERN_MU=SPRAY_PATTERN_MU*PI/180..EB
5123 IF (PART_ID/='null' .AND. SPRAY_PATTERN_TABLE == 'null') THEN
5124 ALLOCATE(PY%SPRAY_LON_CDF(0:NDC2),PY%SPRAY_LON(0:NDC2),PY%SPRAY_LAT(0:NDC2),PY%SPRAY_LAT_CDF(0:NDC2,0:NDC2))
5125 IF (SPRAY_PATTERN_MU<0..EB) THEN
5126 IF (SPRAY_ANGLE(1,1)>0..EB) THEN
5127 SPRAY_PATTERN_MU=0.5..EB*SUM(PY%SPRAY_ANGLE(1:2,1))
5128 ELSE
5129 SPRAY_PATTERN_MU=0..EB
5130 ENDIF
5131 ENDIF
5132 CALL SPRAY_ANGLE_DISTRIBUTION(PY%SPRAY_LON,PY%SPRAY_LAT,PY%SPRAY_LON_CDF,PY%SPRAY_LAT_CDF, &
5133 SPRAY_PATTERN_BETA,SPRAY_PATTERN_MU,PY%SPRAY_ANGLE &
5134 ,SPRAY_PATTERN_SHAPE,NDC2)
5135 ENDIF
5136
5137 ! PDPA model
5138
5139 PY%PDPA_START = PDPA_START
5140 PY%PDPA_END = PDPA_END
5141 PY%PDPA_RADIUS = PDPA_RADIUS
5142 PY%PDPA_M = PDPA_M
5143 PY%PDPA_N = PDPA_N
5144 PY%PDPA_INTEGRATE = PDPA_INTEGRATE
5145 PY%PDPA_NORMALIZE = PDPA_NORMALIZE
5146 IF (TRIM(PY%QUANTITY) == 'NUMBER CONCENTRATION') THEN
5147 PY%PDPA_M = 0
5148 PY%PDPA_N = 0
5149 ENDIF
5150 IF ((TRIM(PY%QUANTITY) == 'MASS CONCENTRATION') .OR. &
5151 (TRIM(PY%QUANTITY) == 'ENTHALPY') .OR. &
5152 (TRIM(PY%QUANTITY) == 'PARTICLE FLUX X') .OR. &
5153 (TRIM(PY%QUANTITY) == 'PARTICLE FLUX Y') .OR. &
5154 (TRIM(PY%QUANTITY) == 'PARTICLE FLUX Z')) THEN
5155 PY%PDPA_M = 3
5156 PY%PDPA_N = 0
5157 ENDIF
5158
5159 ! Histograms of PDPA data
5160 PY%PDPA_HISTOGRAM = PDPA_HISTOGRAM
5161 PY%PDPA_HISTOGRAM_NBINS = PDPA_HISTOGRAM_NBINS
5162 PY%PDPA_HISTOGRAM_LIMITS = PDPA_HISTOGRAM_LIMITS
5163 PY%PDPA_HISTOGRAM_CUMULATIVE = PDPA_HISTOGRAM_CUMULATIVE
5164 IF (PDPA_HISTOGRAM) THEN
5165 IF (PDPA_HISTOGRAM_NBINS<2) THEN
5166 WRITE(MESSAGE,'(A,A,A)') 'ERROR: Problem with PROP ',TRIM(PY%ID),', PDPA_HISTOGRAM needs PDPA_NBINS>2'
5167 CALL SHUTDOWN(MESSAGE) ; RETURN
5168 ENDIF
5169 IF (ABS(PDPA_HISTOGRAM_LIMITS(1)-PDPA_HISTOGRAM_LIMITS(2)) < TWO_EPSILON_EB) THEN
5170 WRITE(MESSAGE,'(A,A,A)') 'ERROR: Problem with PROP ',TRIM(PY%ID),', PDPA_HISTOGRAM needs PDPA_HISTOGRAM_LIMITS'
5171 CALL SHUTDOWN(MESSAGE) ; RETURN
5172 ENDIF
5173
5174 ENDIF
5175
5176 PY%FED_ACTIVITY = FED_ACTIVITY
5177 IF (FED_ACTIVITY < 1 .OR. FED_ACTIVITY > 3) THEN
5178 WRITE(MESSAGE,'(A,A,A,10)') 'ERROR: Problem with PROP ',TRIM(PY%ID),', FED_ACTIVITY out of range: ',FED_ACTIVITY
5179 CALL SHUTDOWN(MESSAGE) ; RETURN
5180 ENDIF
5181
5182 PATCH_VELOCITY_IF: IF (VELOCITY_COMPONENT>0) THEN
5183 IF (VELOCITY_COMPONENT > 3) THEN
5184 WRITE(MESSAGE,'(A,A,A,10)') 'ERROR: Problem with PROP ',TRIM(PY%ID),', VELOCITY_COMPONENT > 3: ',
VELOCITY_COMPONENT
5185 CALL SHUTDOWN(MESSAGE) ; RETURN
5186 ENDIF
5187 IF (P0<-1.E9..EB) THEN
5188 WRITE(MESSAGE,'(A,A,A)') 'ERROR: Problem with PROP ',TRIM(PY%ID),', VELOCITY_PATCH requires P0'
5189 CALL SHUTDOWN(MESSAGE) ; RETURN
5190 ENDIF
5191

```



Source Code files for edited portions of FDS

```

5192 PY%I_VEL = VELOCITY_COMPONENT
5193 PY%P0 = P0 ! value at origin of Taylor expansion
5194 DO J=1,3
5195 PY%PX(J) = PX(J) ! first derivative of P evaluated at origin
5196 DO I=1,3
5197 IF (I>J) PXX(I,J)=PXX(J,I) ! make symmetric
5198 PY%PXX(I,J) = PXX(I,J) ! second derivative of P evaluated at origin
5199 ENDDO
5200 ENDDO
5201 ENDIF PATCH.VELOCITY_IF
5202
5203 ! Set flow variables
5204 PY%MASS.FLOW_RATE =MASS.FLOW_RATE
5205 PY%FLOW_RATE =FLOW_RATE
5206
5207 IF (PART_ID/= 'null' .AND. PRESSURE_RAMP /= 'null') THEN
5208 CALL GET_RAMP_INDEX(PRESSURE_RAMP, 'PRESSURE', PY%PRESSURE_RAMP_INDEX)
5209 ELSE
5210 PY%PRESSURE_RAMP_INDEX = 0
5211 ENDIF
5212
5213 ! Check sufficient input
5214
5215 IF (PY%PRESSURE_RAMP_INDEX == 0 .AND. FLOW_RATE > 0._EB) THEN
5216 IF (K_FACTOR < 0._EB) K_FACTOR = 10.0_EB
5217 ENDIF
5218
5219 IF (PART_ID /= 'null' .AND. ABS(PDPA_RADIUS) <= TWO_EPSILON_EB) THEN
5220 IF (MASS_FLOW_RATE > 0._EB) THEN
5221 PY%MASS_FLOW_RATE = MASS_FLOW_RATE
5222 IF (ABS(PARTICLE_VELOCITY) <= TWO_EPSILON_EB) THEN
5223 WRITE(MESSAGE, '(A,A,A)') 'ERROR: Problem with PROP ', TRIM(PY%ID), ', must specify PARTICLE_VELOCITY with
MASS_FLOW_RATE'
5224 CALL SHUTDOWN(MESSAGE) ; RETURN
5225 ELSE
5226 PY%PARTICLE_VELOCITY = PARTICLE_VELOCITY
5227 ENDIF
5228 ELSE
5229 IF ((FLOW_RATE > 0._EB .AND. K_FACTOR <= 0._EB .AND. OPERATING_PRESSURE <= 0._EB) .OR. &
5230 (FLOW_RATE < 0._EB .AND. K_FACTOR >= 0._EB .AND. OPERATING_PRESSURE <= 0._EB) .OR. &
5231 (FLOW_RATE < 0._EB .AND. K_FACTOR <= 0._EB .AND. OPERATING_PRESSURE > 0._EB)) THEN
5232 WRITE(MESSAGE, '(A,A,A)') 'ERROR: Problem with PROP ', TRIM(PY%ID), ', too few flow parameters'
5233 CALL SHUTDOWN(MESSAGE) ; RETURN
5234 ENDIF
5235 IF (K_FACTOR < 0._EB .AND. OPERATING_PRESSURE > 0._EB) K_FACTOR = FLOW_RATE/SQRT(OPERATING_PRESSURE)
5236 IF (FLOW_RATE < 0._EB .AND. OPERATING_PRESSURE > 0._EB) FLOW_RATE = K_FACTOR*SQRT(OPERATING_PRESSURE)
5237 IF (OPERATING_PRESSURE < 0._EB .AND. K_FACTOR > 0._EB) OPERATING_PRESSURE = (FLOW_RATE/K_FACTOR)**2
5238 PY%K_FACTOR = K_FACTOR
5239 PY%FLOW_RATE = FLOW_RATE
5240 PY%OPERATING_PRESSURE = OPERATING_PRESSURE
5241
5242 IF (PARTICLE_VELOCITY <= TWO_EPSILON_EB .AND. ORIFICE_DIAMETER <= TWO_EPSILON_EB .AND. &
5243 PRESSURE_RAMP = 'null' .AND. SPRAY_PATTERN_TABLE = 'null') THEN
5244 WRITE(MESSAGE, '(A,A,A)') 'WARNING: PROP ', TRIM(PY%ID), ' PARTICLE velocity is not defined.'
5245 IF (MYID=0) WRITE(LU_ERR, '(A)') TRIM(MESSAGE)
5246 ENDIF
5247
5248 IF (PARTICLE_VELOCITY > 0._EB) THEN
5249 PY%PARTICLE_VELOCITY = PARTICLE_VELOCITY
5250 ELSEIF ((ORIFICE_DIAMETER > 0._EB) .AND. (FLOW_RATE > 0._EB)) THEN
5251 PY%PARTICLE_VELOCITY = (FLOW_RATE/60._EB/1000._EB)/(PI*(ORIFICE_DIAMETER/2._EB)**2)
5252 ENDIF
5253 ENDIF
5254 ENDIF
5255 IF (FLOW_RAMP /= 'null') THEN
5256 CALL GET_RAMP_INDEX(FLOW_RAMP, 'TIME', PY%FLOW_RAMP_INDEX)
5257 ELSE
5258 PY%FLOW_RAMP_INDEX = 0
5259 ENDIF
5260 IF (ABS(FLOW_TAU) > TWO_EPSILON_EB) THEN
5261 PY%FLOW_TAU = FLOW_TAU/TIME_SHRINK_FACTOR
5262 IF (FLOW_TAU > 0._EB) PY%FLOW_RAMP_INDEX = TANHRAMP
5263 IF (FLOW_TAU < 0._EB) PY%FLOW_RAMP_INDEX = TSQR_RAMP
5264 ENDIF
5265
5266 ! Check for SPEC_ID
5267
5268 IF (PY%SPEC_ID /= 'null') THEN
5269 CALL GET_SPEC_OR_SMIX_INDEX(PY%SPEC_ID, PY%Y_INDEX, PY%Z_INDEX)
5270 IF (PY%Z_INDEX >= 0 .AND. PY%Y_INDEX >= 1) THEN
5271 IF (TRIM(PY%QUANTITY) = 'DIFFUSIVITY') THEN
5272 PY%Y_INDEX = -999
5273 ELSE
5274 PY%Z_INDEX = -999
5275 ENDIF
5276 ENDIF
5277 IF (PY%Y_INDEX < 1 .AND. PY%Z_INDEX < 0) THEN
5278 WRITE(MESSAGE, '(A,A,A)') 'ERROR: PROP SPEC_ID ', TRIM(PY%SPEC_ID), ' not found'

```

```

5279 CALL SHUTDOWN(MESSAGE) ; RETURN
5280 ENDIF
5281 ENDIF
5282
5283 ENDDO READ_PROP_LOOP
5284
5285
5286 CONTAINS
5287
5288
5289 SUBROUTINE SET_PROP_DEFAULTS
5290
5291 ACTIVATION_OBSCURATION = 3.24_EB ! %/m
5292 ACTIVATION_TEMPERATURE = -273.15_EB ! C
5293 ALPHA_C = 1.8_EB ! m, Heskestad Length Scale
5294 ALPHA_E = 0.0_EB
5295 BETA_C = -1.0_EB
5296 BETA_E = -1.0_EB
5297 DENSITY = 8908._EB ! kg/m3 (Nickel)
5298 DIAMETER = 0.001 ! m
5299 EMISSIVITY = 0.85_EB
5300 HEAT_TRANSFER_COEFFICIENT = -1._EB ! W/m2/K
5301 SPECIFIC_HEAT = 0.44_EB ! kJ/kg/K (Nickel)
5302 C_FACTOR = 0.0_EB
5303 CHARACTERISTIC_VELOCITY = 1.0_EB ! m/s
5304 PARTICLE_VELOCITY = 0._EB ! m/s
5305 PARTICLES_PER_SECOND = 5000
5306 !DT_INSERT = 0.01 ! s
5307 FLOW_RATE = -1._EB ! L/min
5308 MASS_FLOW_RATE = -1._EB
5309 FLOW_RAMP = 'null'
5310 FLOW_TAU = 0._EB
5311 GAUGE_EMISSIVITY = 1._EB
5312 GAUGE_TEMPERATURE = TMPA - TMFM
5313 INITIAL_TEMPERATURE = TMPA - TMFM
5314 ID = 'null'
5315 K_FACTOR = -1.0_EB ! L/min/bar**0.5
5316 MASS_FLOW_RATE = -1._EB ! kg/s
5317 OFFSET = 0.05_EB ! m
5318 OPERATING_PRESSURE = -1.0_EB ! bar
5319 ORIFICE_DIAMETER = 0.0_EB ! m
5320 PART_ID = 'null'
5321 PDPA_START = T_BEGIN
5322 PDPA_END = T_END + 1.0_EB
5323 PDPA_RADIUS = 0.0_EB
5324 PDPA_M = 0
5325 PDPA_N = 0
5326 PDPA_HISTOGRAM = .FALSE.
5327 PDPA_HISTOGRAM_CUMULATIVE = .FALSE.
5328 PDPA_HISTOGRAM_NBINS = -1
5329 PDPA_HISTOGRAM_LIMITS = 0._EB
5330 PDPA_HISTOGRAM_CUMULATIVE = .FALSE.
5331 PDPA_INTEGRATE = .TRUE.
5332 PDPA_NORMALIZE = .TRUE.
5333 PRESSURE_RAMP = 'null'
5334 P0 = -1.E10_EB
5335 PX = 0._EB
5336 PXX = 0._EB
5337 QUANTITY = 'null'
5338 RTI = 100._EB ! (ms)**0.5
5339 SMOKEVIEW_ID = 'null'
5340 SMOKEVIEW_PARAMETERS = 'null'
5341 SPEC_ID = 'null'
5342 SPRAY_ANGLE(1,1) = 60._EB ! degrees
5343 SPRAY_ANGLE(2,1) = 75._EB ! degrees
5344 SPRAY_ANGLE(1,2) = -999._EB ! degrees
5345 SPRAY_ANGLE(2,2) = -999._EB ! degrees
5346 SPRAY_PATTERN_TABLE = 'null'
5347 SPRAY_PATTERN_SHAPE = 'GAUSSIAN'
5348 SPRAY_PATTERN_MU = -1._EB
5349 SPRAY_PATTERN_BETA = 5.0_EB
5350 FED_ACTIVITY = 2 ! light work
5351 VELOCITY_COMPONENT = 0
5352 END SUBROUTINE SET_PROP_DEFAULTS
5353
5354 END SUBROUTINE READ_PROP
5355
5356
5357
5358 SUBROUTINE PROC_PROP
5359 USE DEVICE_VARIABLES
5360 REAL(EB) :: TOTAL_FLOWRATE, SUBTOTAL_FLOWRATE
5361 INTEGER :: N, NN, N.V.FACTORS, ILPC
5362 LOGICAL :: TABLE_NORMED(1:N, TABLE)
5363 TYPE (PROPERTY_TYPE), POINTER :: PY=>NULL()
5364 TYPE (TABLE_TYPE), POINTER :: TA=>NULL()
5365 TYPE (LAGRANGIAN_PARTICLE_CLASS_TYPE), POINTER :: LPC=>NULL()
5366

```

```

5367 TABLENORMED = .FALSE.
5368
5369 PROP_LOOP: DO N=0,N_PROP
5370 PY => PROPERTY(N)
5371
5372 ! Assign PART_INDEX to Device PROPERTY array
5373
5374 IF (PY%PART_ID/='null') THEN
5375
5376 DO ILPC=1,N_LAGRANGIAN_CLASSES
5377 LPC => LAGRANGIAN_PARTICLE_CLASS(ILPC)
5378 IF (LPC%ID==PY%PART_ID) THEN
5379 PY%PART_INDEX = ILPC
5380 EXIT
5381 ENDIF
5382 ENDDO
5383
5384 IF (PY%PART_INDEX<0) THEN
5385 WRITE(MESSAGE,'(A,I0,A)') 'ERROR: PART_ID for PROP ',N,' not found'
5386 CALL SHUTDOWN(MESSAGE) ; RETURN
5387 ENDIF
5388
5389 IF (LPC%ID==PY%PART_ID .AND. LPC%MASSLESS_TRACER) THEN
5390 IF (.NOT.(TRIM(PY%QUANTITY)== 'NUMBER CONCENTRATION' .OR. &
5391 TRIM(PY%QUANTITY)== 'U-VELOCITY' .OR. &
5392 TRIM(PY%QUANTITY)== 'V-VELOCITY' .OR. &
5393 TRIM(PY%QUANTITY)== 'W-VELOCITY' .OR. &
5394 TRIM(PY%QUANTITY)== 'VELOCITY')) THEN
5395 WRITE(MESSAGE,'(A,I0,A)') 'ERROR: PART_ID for PROP ',N,' cannot refer to MASSLESS particles'
5396 CALL SHUTDOWN(MESSAGE) ; RETURN
5397 ENDIF
5398 ENDIF
5399
5400 PARTICLE_FILE=.TRUE.
5401
5402 ENDIF
5403
5404 ! Set up spinkler distribution if needed
5405
5406 IF (PY%SPRAY_PATTERN_INDEX > 0) THEN
5407 TA => TABLES(PY%SPRAY_PATTERN_INDEX)
5408 ALLOCATE(PY%TABLE_ROW(1:TA%NUMBER_ROWS))
5409 TOTAL_FLOWRATE=0._EB
5410 SUBTOTAL_FLOWRATE=0._EB
5411 DO NN=1,TA%NUMBER_ROWS
5412 IF (TA%TABLE_DATA(NN,6) <=0._EB) THEN
5413 WRITE(MESSAGE,'(A,A,A,I0)') 'ERROR: Spray Pattern Table, ',TRIM(PY%TABLE_ID),' , massflux <= 0 for line ',NN
5414 CALL SHUTDOWN(MESSAGE) ; RETURN
5415 ENDIF
5416 TOTAL_FLOWRATE = TOTAL_FLOWRATE + TA%TABLE_DATA(NN,6)
5417 ENDDO
5418 IF (TABLE_NORMED(PY%SPRAY_PATTERN_INDEX)) THEN
5419 DO NN=1,TA%NUMBER_ROWS
5420 SUBTOTAL_FLOWRATE = SUBTOTAL_FLOWRATE + TA%TABLE_DATA(NN,6)
5421 PY%TABLE_ROW(NN) = SUBTOTAL_FLOWRATE/TOTAL_FLOWRATE
5422 ENDDO
5423 ELSE
5424 DO NN=1,TA%NUMBER_ROWS
5425 TA%TABLE_DATA(NN,1) = TA%TABLE_DATA(NN,1) * P1/180._EB
5426 TA%TABLE_DATA(NN,2) = TA%TABLE_DATA(NN,2) * P1/180._EB
5427 TA%TABLE_DATA(NN,3) = TA%TABLE_DATA(NN,3) * P1/180._EB
5428 TA%TABLE_DATA(NN,4) = TA%TABLE_DATA(NN,4) * P1/180._EB
5429 SUBTOTAL_FLOWRATE = SUBTOTAL_FLOWRATE + TA%TABLE_DATA(NN,6)
5430 PY%TABLE_ROW(NN) = SUBTOTAL_FLOWRATE/TOTAL_FLOWRATE
5431 ENDDO
5432 TABLE_NORMED(PY%SPRAY_PATTERN_INDEX) = .TRUE.
5433 ENDIF
5434 PY%TABLE_ROW(TA%NUMBER_ROWS) = 1._EB
5435 END IF
5436
5437 ! Set up pressure dependence
5438 IF (PY%PRESSURE_RAMP_INDEX > 0) THEN
5439 IF (PY%SPRAY_PATTERN_INDEX > 0) THEN
5440 N_V_FACTORS = TA%NUMBER_ROWS
5441 ELSE
5442 N_V_FACTORS = 1
5443 ENDIF
5444 ALLOCATE(PY%V_FACTOR(1:N_V_FACTORS))
5445 IF (PY%SPRAY_PATTERN_INDEX > 0) THEN
5446 DO NN=1,TA%NUMBER_ROWS
5447 PY%V_FACTOR(NN) = TA%TABLE_DATA(NN,5)/SQRT(PY%OPERATING_PRESSURE)
5448 ENDDO
5449 ELSE
5450 PY%V_FACTOR = PY%PARTICLE_VELOCITY/SQRT(PY%OPERATING_PRESSURE)
5451 ENDIF
5452 ENDIF
5453
5454 ENDDO PROP_LOOP

```

```

5455
5456 END SUBROUTINE PROC_PROP
5457
5458
5459 SUBROUTINE READ_MATL
5460
5461
5462 USE MATH_FUNCTIONS, ONLY : GET_RAMP_INDEX
5463 CHARACTER(LABEL_LENGTH) :: CONDUCTIVITY_RAMP, SPECIFIC_HEAT_RAMP
5464 CHARACTER(LABEL_LENGTH) :: SPEC_ID(MAX_SPECIES, MAX_REACTIONS)
5465 REAL(EB) :: EMISSIVITY, CONDUCTIVITY, SPECIFIC_HEAT, DENSITY, ABSORPTION_COEFFICIENT, BOILING_TEMPERATURE, &
5466 PEAK_REACTION_RATE, POROSITY
5467 REAL(EB), DIMENSION(MAX_MATERIALS, MAX_REACTIONS) :: NU_MATL
5468 REAL(EB), DIMENSION(MAX_REACTIONS) :: A, E, HEATING_RATE, PYROLYSIS_RANGE, HEAT_OF_REACTION, &
5469 N_S, N_T, N_O2, REFERENCE_RATE, REFERENCE_TEMPERATURE, THRESHOLD_TEMPERATURE, HEAT_OF_COMBUSTION, &
5470 THRESHOLD_SIGN, GAS_DIFFUSION_DEPTH, NU_O2, BETA_CHAR
5471 REAL(EB) :: NU_SPEC(MAX_SPECIES, MAX_REACTIONS)
5472 LOGICAL, DIMENSION(MAX_REACTIONS) :: PCR
5473 LOGICAL :: ALLOW_SHRINKING, ALLOW_SWELLING, VEGETATION
5474 CHARACTER(25) :: COLOR
5475 INTEGER :: RGB(3)
5476 CHARACTER(LABEL_LENGTH), DIMENSION(MAX_MATERIALS, MAX_REACTIONS) :: MATL_ID
5477 INTEGER :: N, NN, NNN, IOS, NR, N_REACTIONS
5478 NAMELIST /MATL/ A, ABSORPTION_COEFFICIENT, ALLOW_SHRINKING, ALLOW_SWELLING, BETA_CHAR, BOILING_TEMPERATURE, COLOR,
CONDUCTIVITY, &
CONDUCTIVITY_RAMP, DENSITY, E, EMISSIVITY, FYI, &
GAS_DIFFUSION_DEPTH, HEATING_RATE, HEAT_OF_COMBUSTION, HEAT_OF_REACTION, ID, MATL_ID, NU_MATL, NU_SPEC, N_REACTIONS, &
N_S, N_T, N_O2, NU_O2, PCR, POROSITY, PYROLYSIS_RANGE, REFERENCE_RATE, REFERENCE_TEMPERATURE, RGB, &
SPECIFIC_HEAT, SPECIFIC_HEAT_RAMP, SPEC_ID, THRESHOLD_SIGN, THRESHOLD_TEMPERATURE, VEGETATION
5479
5480 ! Count the MATL lines in the input file
5481
5482 REWIND(LU_INPUT) ; INPUT_FILE_LINE_NUMBER = 0
5483 N_MATL = 0
5484 COUNT_MATL_LOOP: DO
5485 CALL CHECKREAD('MATL', LU_INPUT, IOS)
5486 IF (IOS==1) EXIT COUNT_MATL_LOOP
5487 READ(LU_INPUT, MATL, ERR=34, IOSTAT=IOS)
5488 N_MATL = N_MATL + 1
5489 MATL_NAME(N_MATL) = ID
5490 34 IF (IOS>0) THEN
5491 WRITE(MESSAGE, '(A,I0,A,I0)') 'ERROR: Problem with MATL number', N_MATL+1, ', line number', INPUT_FILE_LINE_NUMBER
5492 CALL SHUTDOWN(MESSAGE) ; RETURN
5493 ENDIF
5494 ENDDO COUNT_MATL_LOOP
5495
5496 ! Allocate the MATERIAL derived type
5497
5498 ALLOCATE(MATERIAL(1:N_MATL), STAT=IZERO)
5499 CALL ChkMemErr('READ', 'MATERIAL', IZERO)
5500
5501 ! Read the MATL lines in the order listed in the input file
5502
5503 REWIND(LU_INPUT) ; INPUT_FILE_LINE_NUMBER = 0
5504
5505 READ_MATL_LOOP: DO N=1, N_MATL
5506
5507 ML => MATERIAL(N)
5508
5509 ! Read user defined MATL lines
5510
5511 CALL CHECKREAD('MATL', LU_INPUT, IOS)
5512 CALL SET_MATL_DEFAULTS
5513 READ(LU_INPUT, MATL)
5514
5515 ! Do some error checking on the inputs
5516
5517 NOT_BOILING: IF (BOILING_TEMPERATURE>4000..EB) THEN
5518
5519 IF ( ( ANY(THRESHOLD_TEMPERATURE>-TMFM) .OR. ANY(REFERENCE_TEMPERATURE>-TMFM) .OR. ANY(A>=0..EB) .OR. ANY(E>=0..EB) .OR. &
5520 ANY(ABS(HEAT_OF_REACTION)>TWO_EPSILON_EB) ) .AND. N_REACTIONS==0) THEN
5521 WRITE(MESSAGE, '(A,A,A)') 'ERROR: Problem with MATL number ', TRIM(ID), ', A reaction parameter is used, but
5522 N_REACTIONS=0'
5523 CALL SHUTDOWN(MESSAGE) ; RETURN
5524 ENDIF
5525
5526 DO NR=1, N_REACTIONS
5527 IF (REFERENCE_TEMPERATURE(NR)<-TMFM .AND. (E(NR)< 0..EB .OR. A(NR)<0..EB)) THEN
5528 WRITE(MESSAGE, '(A,A,A,I0,A)') 'ERROR: Problem with MATL ', TRIM(ID), ', REAC ', NR, '. Set REFERENCE_TEMPERATURE or E
5529 , A'
5530 CALL SHUTDOWN(MESSAGE) ; RETURN
5531 ENDIF
5532 IF (ABS(SUM(NU_MATL(:, NR)))<=TWO_EPSILON_EB .AND. ABS(SUM(NU_SPEC(:, NR)))<=TWO_EPSILON_EB) THEN
5533 WRITE(MESSAGE, '(A,A,A,I0,A)') 'WARNING: MATL ', TRIM(ID), ', REAC ', NR, '. No product yields (NUs) set'
5534 IF (MYID==0) WRITE(LU_ERR, '(A)') TRIM(MESSAGE)
5535 ENDIF
5536 ENDDO
5537
5538

```

Source Code files for edited portions of FDS

```

5539
5540 ELSE NOT_BOILING ! Is liquid
5541
5542 N_REACTIONS = 1
5543 IF (ABS(HEAT_OF_REACTION(1))<=TWO_EPSILON_EB) THEN
5544 WRITE(MESSAGE,'(A,A,A)') 'ERROR: HEAT_OF_REACTION should be greater than zero for liquid MAIL ',TRIM(ID)
5545 CALL SHUTDOWN(MESSAGE) ; RETURN
5546 ENDIF
5547
5548 ENDIF NOT_BOILING
5549
5550 ! Error checking for thermal properties
5551
5552 IF (ABS(DENSITY) <=TWO_EPSILON_EB ) THEN
5553 WRITE(MESSAGE,'(A,A,A)') 'ERROR: Problem with MAIL ',TRIM(ID),': DENSITY=0'
5554 CALL SHUTDOWN(MESSAGE) ; RETURN
5555 ENDIF
5556 IF (ABS(CONDUCTIVITY) <=TWO_EPSILON_EB .AND. CONDUCTIVITY_RAMP == 'null') THEN
5557 WRITE(MESSAGE,'(A,A,A)') 'ERROR: Problem with MAIL ',TRIM(ID),': CONDUCTIVITY = 0'
5558 CALL SHUTDOWN(MESSAGE) ; RETURN
5559 ENDIF
5560 IF (ABS(SPECIFIC_HEAT)<=TWO_EPSILON_EB .AND. SPECIFIC_HEAT_RAMP == 'null') THEN
5561 WRITE(MESSAGE,'(A,A,A)') 'ERROR: Problem with MAIL ',TRIM(ID),': SPECIFIC_HEAT = 0'
5562 CALL SHUTDOWN(MESSAGE) ; RETURN
5563 ENDIF
5564 IF (SPECIFIC_HEAT > 10._EB) WRITE(LU_ERR,'(A,A)') 'WARNING: SPECIFIC_HEAT units are kJ/kg/K check MAIL ',TRIM(ID)
5565
5566 ! Pack MAIL parameters into the MATERIAL derived type
5567
5568 IF (COLOR/= 'null') THEN
5569 CALL COLOR2RGB(RGB,COLOR)
5570 ENDIF
5571 IF (ANY(RGB< 0)) THEN
5572 RGB(1) = 255
5573 RGB(2) = 204
5574 RGB(3) = 102
5575 ENDIF
5576 M%RGB = RGB
5577
5578 M%A(:) = A(:)
5579
5580 ALLOCATE(M%ADJUST_BURN_RATE(N_TRACKED_SPECIES,MAX(1,N_REACTIONS)),STAT=IZERO)
5581 CALL ChkMemErr('READ','MATERIAL',IZERO)
5582 M%ADJUST_BURN_RATE = 1._EB
5583 M%ALLOW_SHRINKING = ALLOW_SHRINKING
5584 M%ALLOW_SWELLING = ALLOW_SWELLING
5585 M%BETA_CHAR(:) = BETA_CHAR(:)
5586 M%C_S = 1000._EB*SPECIFIC_HEAT/TIME_SHRINK_FACTOR
5587 M%E(:) = 1000._EB*E(:)
5588 M%EMISSIVITY = EMISSIVITY
5589 M%FYI = FYI
5590 M%GAS_DIFFUSION_DEPTH(:) = GAS_DIFFUSION_DEPTH(:)
5591 M%HEAT_OF_COMBUSTION = 1000._EB*HEAT_OF_COMBUSTION
5592 M%H_R(:) = 1000._EB*HEAT_OF_REACTION(:)
5593 M%ID = ID
5594 M%KAPPA_S = ABSORPTION_COEFFICIENT
5595 M%K_S = CONDUCTIVITY
5596 M%N_REACTIONS = N_REACTIONS
5597 M%N_O2(:) = N_O2(:)
5598 M%NU_O2(:) = NU_O2(:)
5599 M%N_S(:) = N_S(:)
5600 M%N_T(:) = N_T(:)
5601 M%NU_RESIDUE = NU_MAIL
5602 M%NU_SPEC = NU_SPEC
5603 M%POROSITY = POROSITY
5604 M%SPEC_ID = SPEC_ID
5605 M%RAMP_C_S = SPECIFIC_HEAT_RAMP
5606 M%RAMP_K_S = CONDUCTIVITY_RAMP
5607 M%RHOS = DENSITY ! This is bulk density of pure material.
5608 M%RESIDUE_MATL_NAME = MATL_ID
5609 M%HEATING_RATE(:) = HEATING_RATE(:)/60._EB
5610 M%PYROLYSIS_RANGE(:) = PYROLYSIS_RANGE(:)
5611 M%PCR(:) = PCR(:)
5612 M%TMP_BOIL = BOILING_TEMPERATURE + TMFM
5613 M%TMP_THR(:) = THRESHOLD_TEMPERATURE(:) + TMFM
5614 M%TMP_REF(:) = REFERENCE_TEMPERATURE(:) + TMFM
5615 M%THR_SIGN(:) = THRESHOLD_SIGN
5616 M%RATE_REF(:) = REFERENCE_RATE(:)
5617
5618 ALLOCATE(M%NU_GAS(N_TRACKED_SPECIES,N_REACTIONS),STAT=IZERO)
5619 CALL ChkMemErr('READ','MATERIAL',IZERO)
5620 M%NU_GAS=0._EB
5621
5622 ! Additional logic
5623
5624 IF (BOILING_TEMPERATURE<5000._EB) THEN
5625 M%PYROLYSIS_MODEL = PYROLYSIS_LIQUID
5626 M%N_REACTIONS = 1

```

```

5627 ELSEIF (VEGETATION) THEN
5628   M%PYROLYSIS_MODEL = PYROLYSIS_VEGETATION
5629   M%ALLOW_SHRINKING = .FALSE.
5630   M%ALLOW_SWELLING = .FALSE.
5631 ELSE
5632   M%PYROLYSIS_MODEL = PYROLYSIS_SOLID
5633 ENDIF
5634
5635 IF (N_REACTIONS==0) M%PYROLYSIS_MODEL = PYROLYSIS_NONE
5636
5637 IF (M%RAMP_K_S/= 'null') THEN
5638   CALL GET_RAMP_INDEX(M%RAMP_K_S, 'TEMPERATURE', NR)
5639   M%K_S = -NR
5640 ENDIF
5641
5642 IF (M%RAMP_C_S/= 'null') THEN
5643   CALL GET_RAMP_INDEX(M%RAMP_C_S, 'TEMPERATURE', NR)
5644   M%C_S = -NR
5645 ENDIF
5646
5647 ! Determine A and E if REFERENCE_TEMPERATURE is specified
5648
5649 DO NR=1, M%N_REACTIONS
5650   IF (M%TMP_REF(NR) > 0. .EB) THEN
5651     IF (M%RATE_REF(NR) > 0. .EB) THEN
5652       PEAK_REACTION_RATE = M%RATE_REF(NR)
5653     ELSE
5654       PEAK_REACTION_RATE = 2. .EB * M%HEATING_RATE(NR) * (1. .EB - SUM(M%NU_RESIDUE(: , NR))) / M%PYROLYSIS_RANGE(NR)
5655     ENDIF
5656     M%E(NR) = EXP(1. .EB) * PEAK_REACTION_RATE * R0 * M%TMP_REF(NR) ** 2 / M%HEATING_RATE(NR)
5657     M%A(NR) = EXP(1. .EB) * PEAK_REACTION_RATE * EXP(M%E(NR)) / (R0 * M%TMP_REF(NR))
5658   ENDIF
5659
5660   M%N_RESIDUE(NR) = 0
5661   DO NN=1, MAX_MATERIALS
5662     IF (M%RESIDUE_MATL_NAME(NN, NR) /= 'null') M%N_RESIDUE(NR) = M%N_RESIDUE(NR) + 1
5663   ENDDO
5664 ENDDO
5665
5666 ENDDO READ_MATL_LOOP
5667
5668 ! Assign a material index to the RESIDUES
5669
5670 DO N=1, N_MATL
5671   ML => MATERIAL(N)
5672   M%RESIDUE_MATL_INDEX = 0
5673   DO NR=1, M%N_REACTIONS
5674     DO NN=1, M%N_RESIDUE(NR)
5675       DO NNN=1, N_MATL
5676         IF (MATL_NAME(NNN) == M%RESIDUE_MATL_NAME(NN, NR)) M%RESIDUE_MATL_INDEX(NN, NR) = NNN
5677       ENDDO
5678       IF (M%RESIDUE_MATL_INDEX(NN, NR) == 0 .AND. M%NU_RESIDUE(NN, NR) > 0. .EB) THEN
5679         WRITE(MESSAGE, '(5A)') 'ERROR: Residue ', TRIM(M%RESIDUE_MATL_NAME(NN, NR)), ' of ', TRIM(MATL_NAME(N)), ' is not
                    defined.'
5680         CALL SHUTDOWN(MESSAGE) ; RETURN
5681       ENDIF
5682     ENDDO
5683   ENDDO
5684 ENDDO
5685
5686 ! Check for duplicate names
5687
5688 IF (N_MATL > 1) THEN
5689   DO N=1, N_MATL-1
5690     DO NN=N+1, N_MATL
5691       IF (MATL_NAME(N) == MATL_NAME(NN)) THEN
5692         WRITE(MESSAGE, '(A,A)') 'ERROR: Duplicate material name: ', TRIM(MATL_NAME(N))
5693         CALL SHUTDOWN(MESSAGE) ; RETURN
5694       ENDIF
5695     ENDDO
5696   ENDDO
5697 ENDIF
5698
5699 ! Check material porosity values
5700
5701 DO N=1, N_MATL
5702   ML => MATERIAL(N)
5703   IF ((1.0 .EB - M%POROSITY) < 1.0E-5 .EB) THEN
5704     WRITE(MESSAGE, '(2A)') 'ERROR: Too high porosity for material ', TRIM(MATL_NAME(N))
5705     CALL SHUTDOWN(MESSAGE) ; RETURN
5706   ENDIF
5707   ! M%RHO_NONPOROUS = DENSITY / (1. .EB - POROSITY)
5708 ENDDO
5709
5710 CONTAINS
5711
5712 SUBROUTINE SET_MATL_DEFAULTS
5713

```

Source Code files for edited portions of FDS

```

5714 A = -1._EB ! 1/s
5715 ABSORPTION_COEFFICIENT = 5.0E4.EB ! 1/m, corresponds to 99.3% drop within 1E-4 m distance.
5716 ALLOW_SHRINKING = .TRUE.
5717 ALLOW_SWELLING = .TRUE.
5718 BOILING_TEMPERATURE = 5000._EB ! C
5719 BETA_CHAR = 0.2.EB
5720 COLOR = 'null'
5721 RGB = -1
5722 CONDUCTIVITY = 0.0.EB ! W/m/K
5723 CONDUCTIVITY_RAMP = 'null'
5724 DENSITY = 0._EB ! kg/m3
5725 E = -1._EB ! kJ/kmol
5726 EMISSIVITY = 0.9.EB
5727 FYI = 'null'
5728 GAS_DIFFUSION_DEPTH = 0.001.EB ! m
5729 HEAT_OF_COMBUSTION = -1._EB ! kJ/kg
5730 HEAT_OF_REACTION = 0._EB ! kJ/kg
5731 ID = 'null'
5732 THRESHOLD_TEMPERATURE = -1MPM ! 0 K
5733 THRESHOLD_SIGN = 1.0
5734 N_REACTIONS = 0
5735 N_O2 = 0._EB
5736 NU_O2 = 0._EB
5737 N_S = 1._EB
5738 N_T = 0._EB
5739 NU_SPEC = 0._EB
5740 NU_MATL = 0._EB
5741 PCR = .FALSE.
5742 POROSITY = 0._EB
5743 REFERENCE_RATE = -1._EB
5744 REFERENCE_TEMPERATURE = -1000._EB
5745 MATL_ID = 'null'
5746 SPECIFIC_HEAT = 0.0.EB ! kJ/kg/K
5747 SPECIFIC_HEAT_RAMP = 'null'
5748 SPEC_ID = 'null'
5749 HEATING_RATE = 5._EB ! K/min
5750 PYROLYSIS_RANGE = 80._EB ! K or C
5751 VEGETATION = .FALSE.
5752
5753 END SUBROUTINE SET_MATL_DEFAULTS
5754
5755 END SUBROUTINE READ_MATL
5756
5757
5758
5759 SUBROUTINE PROC_MATL
5760
5761 ! Process Materials — do some additional set-up work with materials
5762
5763 INTEGER :: N, J, JJ, NS, NS2, NR, Z_INDEX(N, TRACKED_SPECIES, MAX_REACTIONS)
5764
5765 PROC_MATL_LOOP: DO N=1, N_MATL
5766
5767 ML => MATERIAL(N)
5768
5769 ! Convert M%NU_SPEC(I, ORDINAL, I, REACTION) and M%SPEC_ID(I, ORDINAL, I, REACTION) to M%NU_GAS(I, SPECIES, I, REACTION)
5770
5771 Z_INDEX = -1
5772 DO NR=1, M%N_REACTIONS
5773 DO NS=1, MAX_SPECIES
5774
5775 IF (TRIM(M%SPEC_ID(NS, NR))=='null' .AND. M%NU_SPEC(NS, NR)>TWO_EPSILON_EB) THEN
5776 WRITE(MESSAGE, '(A,A,A,I0,A,I0)') 'ERROR: MATL ', TRIM(MATL_NAME(N)), ' requires a SPEC_ID for the ', &
5777 NS, 'th yield of reaction ', NR
5778 CALL SHUTDOWN(MESSAGE) ; RETURN
5779 ENDIF
5780 IF (TRIM(M%SPEC_ID(NS, NR))=='null') EXIT
5781 IF (NS==2 .AND. M%PYROLYSIS_MODEL==PYROLYSIS_LIQUID) THEN
5782 WRITE(MESSAGE, '(A,A,A,A)') 'ERROR: MATL ', TRIM(MATL_NAME(N)), ' can only specify one SPEC_ID for a liquid'
5783 CALL SHUTDOWN(MESSAGE) ; RETURN
5784 ENDIF
5785 DO NS2=1, N_TRACKED_SPECIES
5786 IF (TRIM(M%SPEC_ID(NS, NR))==TRIM(SPECIES_MIXTURE(NS2)%ID)) THEN
5787 Z_INDEX(NS, NR) = NS2
5788 M%NU_GAS(Z_INDEX(NS, NR), NR) = M%NU_SPEC(NS, NR)
5789 EXIT
5790 ENDIF
5791 ENDDO
5792 IF (Z_INDEX(NS, NR)==-1) THEN
5793 WRITE(MESSAGE, '(A,A,A,A,A,A)') 'ERROR: SPECIES ', TRIM(M%SPEC_ID(NS, NR)), &
5794 ' corresponding to MATL ', TRIM(MATL_NAME(N)), ' is not a tracked species'
5795 CALL SHUTDOWN(MESSAGE) ; RETURN
5796 ENDIF
5797
5798 ENDDO
5799 ENDDO
5800

```

Source Code files for edited portions of FDS

```

5801 ! Adjust burn rate if heat of combustion is different from the gas phase reaction value
5802
5803 IF (N.REACTIONS>0) THEN
5804 RN => REACTION(1)
5805 DO NS = 1,N_TRACKED_SPECIES
5806 DO J=0,MAX(1,ML%N.REACTIONS)
5807 JJ = MAX(J,1)
5808 IF (ML%HEAT_OF_COMBUSTION(JJ)>0._EB .AND. RN%HEAT_OF_COMBUSTION>0._EB) &
5809 ML%ADJUST_BURN_RATE(NS, JJ) = ML%HEAT_OF_COMBUSTION(JJ)/RN%HEAT_OF_COMBUSTION
5810 ENDDO
5811 ENDDO
5812 ENDF
5813
5814 ! Check units of specific heat
5815
5816 IF (ML%RAMP_C.S/='null') THEN
5817 NR = -NINT(ML%C.S)
5818 IF (.NOT.RAMPS(NR)%DEP.VAR.UNITS.CONVERTED) THEN
5819 RAMPS(NR)%INTERPOLATED.DATA(:) = RAMPS(NR)%INTERPOLATED.DATA(:) * 1000._EB/TIME.SHRIK.FACTOR
5820 RAMPS(NR)%DEP.VAR.UNITS.CONVERTED = .TRUE.
5821 ENDF
5822 IF (RAMPS(NR)%DEPENDENT.DATA(1) > 10._EB) &
5823 WRITE(LU_ERR,'(A,A)') 'WARNING: SPECIFIC_HEAT units are kJ/kg/K check MAIL ',TRIM(ID)
5824 ENDF
5825
5826 ENDDO PROC.MATL_LOOP
5827
5828 END SUBROUTINE PROC.MATL
5829
5830
5831 SUBROUTINE READ.SURF
5832
5833 USE MATH.FUNCTIONS, ONLY : GET_RAMP_INDEX
5834 USE DEVICE.VARIABLES, ONLY : PROPERTY_TYPE
5835 CHARACTER(LABEL_LENGTH) :: PART_ID,RAMP.MF(MAX_SPECIES),RAMP.Q,RAMP.V,RAMP.T,MATL.ID(MAXLAYERS,MAXMATERIALS),&
5836 PROFILE,BACKING,GEOMETRY,NAME.LIST(MAXMATERIALS*MAXLAYERS),EXTERNAL.FLUX.RAMP,RAMP.EF,RAMP.PART,&
5837 SPEC.ID(MAX_SPECIES),RAMP.T.I,RAMP.V.X,RAMP.V.Y,RAMP.V.Z
5838 EQUIVALENCE(EXTERNAL.FLUX.RAMP,RAMP.EF)
5839 LOGICAL :: ADIABATIC,BURN.AWAY,FREE.SLIP,NO.SLIP,CONVERT.VOLUME.TO.MASS
5840 CHARACTER(60) :: TEXTURE.MAP,HEAT.TRANSFER.MODEL
5841 CHARACTER(25) :: COLOR
5842 REAL(EB) :: TAU.Q,TAU.V,TAU.T,TAU.MF(MAX_SPECIES),HRR.PUA,MLR.PUA,TEXTURE.WIDTH,TEXTURE.HEIGHT,VEL.T(2),&
5843 TAU.EXTERNAL.FLUX,TAU.EF,E.COEFFICIENT,VOLUME.FLOW,VOLUME.FLUX,&
5844 TMP.FRONT,TMP.INNER(MAXLAYERS),THICKNESS(MAXLAYERS),VEL,VEL.BULK,INTERNAL.HEAT.SOURCE(MAXLAYERS),&
5845 MASS.FLUX(MAX_SPECIES),Z0,PLE,CONVECTIVE.HEAT.FLUX,PARTICLE.MASS.FLUX,&
5846 TRANSPARENCY,EXTERNAL.FLUX,TMP.BACK,MASS.FLUX.TOTAL,MASS.FLUX.VAR,STRETCH.FACTOR(MAXLAYERS),
5847 CONVECTION.LENGTH.SCALE,&
5848 MATL.MASS.FRACTION(MAXLAYERS,MAXMATERIALS),CELL.SIZE.FACTOR,MAX.PRESSURE,&
5849 IGNITION.TEMPERATURE,HEAT.OF.VAPORIZATION,NET.HEAT.FLUX,LAYER.DIVIDE,&
5850 ROUGHNESS,RADIUS,INNER.RADIUS,LENGTH,WIDTH,DT.INSERT,HEAT.TRANSFER.COEFFICIENT,HEAT.TRANSFER.COEFFICIENT.BACK,&
5851 TAU.PART,EMISSIVITY,EMISSIVITY.BACK,EMISSIVITY.DEFAULT,SPREAD.RATE,XYZ(3),MINIMUM.LAYER.THICKNESS,VEL.GRAD,&
5852 MASS.FRACTION(MAX_SPECIES),MASS.TRANSFER.COEFFICIENT,&
5853 C.FORCED.CONSTANT,C.FORCED.PR.EXP,C.FORCED.RE,C.FORCED.RE.EXP,C.VERTICAL,C.HORIZONTAL,ZETA.FRONT,&
5854 AUTO.IGNITION.TEMPERATURE
5855 EQUIVALENCE(TAU.EXTERNAL.FLUX,TAU.EF)
5856 INTEGER :: NPPC,N,IOS,NL,NN,NNN,NRM,N.LIST,N.LIST2,INDEX.LIST(MAXMATERIALS.TOTAL),LEAK.PATH(2),DUCT.PATH(2),RGB
5857 (3),&
5858 NR,IL,N.CELLS.MAX
5859 INTEGER :: VEGETATION.LAYERS,N.LAYER.CELLS.MAX(MAXLAYERS)
5860 REAL(EB) :: VEGETATION.CDRAG,VEGETATION.CHAR.FRACTION,VEGETATION.ELEMENT.DENSITY,VEGETATION.HEIGHT,&
5861 VEGETATION.INITIAL.TEMP,VEGETATION.LOAD,VEGETATION.LSET.IGNITE.TIME,VEG.LSET.QCON,VEGETATION.MOISTURE,&
5862 VEGETATION.SVRATIO,&
5863 FIRELINE.MLR.MAX,VEGETATION.GROUND.TEMP,VEG.LSET.ROS.HEAD,VEG.LSET.ROS.FLANK,VEG.LSET.ROS.BACK,&
5864 VEG.LSET.WIND.EXP,VEG.LSET.BETA,VEG.LSET.HT,VEG.LSET.SIGMA,VEG.LSET.ELLIPSE.HEAD
5865 LOGICAL :: VEGETATION,VEGETATION.NO.BURN,VEGETATION.LINEAR.DEGRAD,VEGETATION.ARRHENIUS.DEGRAD,
5866 VEG.LEVEL.SET.SPREAD,&
5867 DEFAULT,EVAC.DEFAULT,VEG.LSET.ELLIPSE,VEG.LSET.TAN2,TGA.ANALYSIS,COMPUTE.EMISSIVITY,COMPUTE.EMISSIVITY.BACK,&
5868 HIT3D
5869
5870 NAMELIST /SURF/ ADIABATIC,AUTO.IGNITION.TEMPERATURE,&
5871 BACKING,BURN.AWAY,CELL.SIZE.FACTOR,C.FORCED.CONSTANT,C.FORCED.PR.EXP,C.FORCED.RE,C.FORCED.RE.EXP,&
5872 C.HORIZONTAL,C.VERTICAL,COLOR,&
5873 CONVECTION.LENGTH.SCALE,CONVECTIVE.HEAT.FLUX,CONVERT.VOLUME.TO.MASS,DEFAULT,&
5874 DT.INSERT,EMISSIVITY,EMISSIVITY.BACK,EVAC.DEFAULT,EXTERNAL.FLUX,E.COEFFICIENT,FIRELINE.MLR.MAX,&
5875 FREE.SLIP,FYI,GEOMETRY,HEAT.OF.VAPORIZATION,HEAT.TRANSFER.COEFFICIENT,HEAT.TRANSFER.COEFFICIENT.BACK,&
5876 HEAT.TRANSFER.MODEL,HRR.PUA,HIT3D,ID,IGNITION.TEMPERATURE,INNER.RADIUS,INTERNAL.HEAT.SOURCE,LAYER.DIVIDE,&
5877 LEAK.PATH,LENGTH,MASS.FLUX,MASS.FLUX.TOTAL,MASS.FLUX.VAR,MASS.FRACTION,MASS.TRANSFER.COEFFICIENT,MATL.ID,&
5878 MATL.MASS.FRACTION,MINIMUM.LAYER.THICKNESS,MLR.PUA,N.CELLS.MAX,N.LAYER.CELLS.MAX,NET.HEAT.FLUX,&
5879 NO.SLIP,NPPC,PARTICLE.MASS.FLUX,PART.ID,PLE,PROFILE,RADIUS,RAMP.EF,RAMP.MF,RAMP.PART,RAMP.Q,RAMP.T,RAMP.T.I,&
5880 RAMP.V,RAMP.V.X,RAMP.V.Y,RAMP.V.Z,&
5881 RGB,ROUGHNESS,SPEC.ID,SPREAD.RATE,STRETCH.FACTOR,&
5882 TAU.EF,TAU.MF,TAU.PART,TAU.Q,TAU.T,TAU.V,TEXTURE.HEIGHT,TEXTURE.MAP,TEXTURE.WIDTH,&
5883 TGA.ANALYSIS,TGA.FINAL.TEMPERATURE,TGA.HEATING.RATE,THICKNESS,&
5884 TMP.BACK,TMP.FRONT,TMP.INNER,TRANSPARENCY,VEGETATION,VEGETATION.ARRHENIUS.DEGRAD,VEGETATION.CDRAG,&
5885 VEGETATION.CHAR.FRACTION,VEGETATION.ELEMENT.DENSITY,VEGETATION.GROUND.TEMP,VEGETATION.HEIGHT,&
5886 VEGETATION.INITIAL.TEMP,VEGETATION.LAYERS,VEGETATION.LINEAR.DEGRAD,VEGETATION.LOAD,VEGETATION.LSET.IGNITE.TIME,&
5887 VEG.LSET.QCON,VEGETATION.MOISTURE,VEGETATION.NO.BURN,VEGETATION.SVRATIO,VEG.LEVEL.SET.SPREAD,&
5888 VEG.LSET.ROS.BACK,VEG.LSET.ROS.FLANK,VEG.LSET.ROS.HEAD,VEG.LSET.WIND.EXP,&

```



## Source Code files for edited portions of FDS

```

5886 VEG.LSET.SIGMA,VEG.LSET.HT,VEG.LSET.BETA,VEG.LSET.ELLIPSE,VEG.LSET.TAN2,VEG.LSET.ELLIPSE_HEAD,&
5887 VEL,VEL.BULK,VEL.GRAD,VEL.T,VOLUME.FLOW,WIDTH,XYZ,Z0,ZETA.FRONT,&
5888 EXTERNAL.FLUX.RAMP,TAU.EXTERNAL.FLUX,VOLUME.FLUX ! Backwards compatability??
5889
5890 ! Count the SURF lines in the input file
5891
5892 REWIND(LU.INPUT) ; INPUT_FILE.LINE.NUMBER = 0
5893 N_SURF = 0
5894 COUNT_SURF_LOOP: DO
5895 HRRPUA = 0._EB
5896 MLRPUA = 0._EB
5897 CALL CHECKREAD('SURF',LU.INPUT,IOS)
5898 IF (IOS==1) EXIT COUNT_SURF_LOOP
5899 READ(LU.INPUT,SURF,ERR=34,IOSTAT=IOS)
5900 N_SURF = N_SURF + 1
5901 34 IF (IOS>0) THEN
5902 WRITE(MESSAGE,'(A,I0,A,I0)') 'ERROR: Problem with SURF number', N_SURF+1,', line number',INPUT_FILE.LINE.NUMBER
5903 CALL SHUTDOWN(MESSAGE) ; RETURN
5904 ENDIF
5905 ENDDO COUNT_SURF_LOOP
5906
5907 ! Allocate the SURFACE derived type, leaving space for SURF entries not defined explicitly by the user
5908
5909 N_SURF_RESERVED = 11
5910 ALLOCATE(SURFACE(0:N_SURF+N_SURF_RESERVED),STAT=IZERO)
5911 CALL ChkMemErr('READ','SURFACE',IZERO)
5912
5913 ! Count the SURF lines in the input file
5914
5915 REWIND(LU.INPUT) ; INPUT_FILE.LINE.NUMBER = 0
5916 NN = 0
5917 COUNT_SURF_LOOP_AGAIN: DO
5918 CALL CHECKREAD('SURF',LU.INPUT,IOS)
5919 IF (IOS==1) EXIT COUNT_SURF_LOOP_AGAIN
5920 READ(LU.INPUT,SURF)
5921 NN = NN+1
5922 SURFACE(NN)%ID = ID
5923 ENDDO COUNT_SURF_LOOP_AGAIN
5924
5925 ! Add extra surface types to the list that has already been compiled
5926
5927 INERT_SURF_INDEX           = 0
5928 OPEN_SURF_INDEX           = N_SURF + 1
5929 MIRROR_SURF_INDEX        = N_SURF + 2
5930 INTERPOLATED_SURF_INDEX  = N_SURF + 3
5931 PERIODIC_SURF_INDEX      = N_SURF + 4
5932 HVAC_SURF_INDEX          = N_SURF + 5
5933 MASSLESS_TRACER_SURF_INDEX = N_SURF + 6
5934 DROPLET_SURF_INDEX       = N_SURF + 7
5935 VEGETATION_SURF_INDEX    = N_SURF + 8
5936 EVACUATION_SURF_INDEX    = N_SURF + 9
5937 MASSLESS_TARGET_SURF_INDEX = N_SURF + 10
5938 PERIODIC_WIND_SURF_INDEX = N_SURF + 11
5939
5940 N_SURF = N_SURF + N_SURF_RESERVED
5941
5942 SURFACE(INERT_SURF_INDEX)%ID           = 'INERT'
5943 SURFACE(OPEN_SURF_INDEX)%ID           = 'OPEN'
5944 SURFACE(MIRROR_SURF_INDEX)%ID         = 'MIRROR'
5945 SURFACE(INTERPOLATED_SURF_INDEX)%ID   = 'INTERPOLATED'
5946 SURFACE(PERIODIC_SURF_INDEX)%ID       = 'PERIODIC'
5947 SURFACE(HVAC_SURF_INDEX)%ID           = 'HVAC'
5948 SURFACE(MASSLESS_TRACER_SURF_INDEX)%ID = 'MASSLESS TRACER'
5949 SURFACE(DROPLET_SURF_INDEX)%ID        = 'DROPLET'
5950 SURFACE(VEGETATION_SURF_INDEX)%ID     = 'VEGETATION'
5951 SURFACE(EVACUATION_SURF_INDEX)%ID     = 'EVACUATION.OUTFLOW'
5952 SURFACE(MASSLESS_TARGET_SURF_INDEX)%ID = 'MASSLESS TARGET'
5953 SURFACE(PERIODIC_WIND_SURF_INDEX)%ID  = 'PERIODIC WIND'
5954
5955 SURFACE(0)%USER_DEFINED = .FALSE.
5956 SURFACE(N_SURF-N_SURF_RESERVED+1:N_SURF)%USER_DEFINED = .FALSE.
5957
5958 ! Check if SURF_DEFAULT exists
5959
5960 CALL CHECK_SURF_NAME(SURF_DEFAULT,EX)
5961 IF (.NOT.EX) THEN
5962 WRITE(MESSAGE,'(A)') 'ERROR: SURF_DEFAULT not found'
5963 CALL SHUTDOWN(MESSAGE) ; RETURN
5964 ENDIF
5965
5966 ! Add evacuation boundary type if necessary
5967
5968 CALL CHECK_SURF_NAME(EVAC_SURF_DEFAULT,EX)
5969 IF (.NOT.EX) THEN
5970 WRITE(MESSAGE,'(A)') 'ERROR: EVAC_SURF_DEFAULT not found'
5971 CALL SHUTDOWN(MESSAGE) ; RETURN
5972 ENDIF
5973

```

Source Code files for edited portions of FDS

```

5974 ! Read the SURF lines
5975
5976 REWIND(LU_INPUT) ; INPUT_FILE.LINE_NUMBER = 0
5977 READ_SURF_LOOP: DO N=0,N_SURF
5978
5979 SF => SURFACE(N)
5980
5981 ! Allocate arrays associated with the SURF line
5982
5983 ALLOCATE(SP%MASS_FRACTION(1:N_TRACKED_SPECIES),STAT=IZERO)
5984 CALL ChkMemErr('READ','SURFACE',IZERO) ; SP%MASS_FRACTION = 0._EB
5985 ALLOCATE(SP%MASS_FLUX(1:N_TRACKED_SPECIES),STAT=IZERO)
5986 CALL ChkMemErr('READ','SURFACE',IZERO) ; SP%MASS_FLUX = 0._EB
5987 ALLOCATE(SP%TAU(-5:N_TRACKED_SPECIES),STAT=IZERO)
5988 CALL ChkMemErr('READ','SURFACE',IZERO) ; SP%TAU = 0._EB
5989 ALLOCATE(SP%ADJUST_BURN_RATE(-5:N_TRACKED_SPECIES),STAT=IZERO)
5990 CALL ChkMemErr('READ','SURFACE',IZERO) ; SP%ADJUST_BURN_RATE = 1._EB
5991 ALLOCATE(SP%RAMP_INDEX(-8:N_TRACKED_SPECIES),STAT=IZERO)
5992 CALL ChkMemErr('READ','SURFACE',IZERO) ; SP%RAMP_INDEX = 0
5993 ALLOCATE(SP%RAMP_MF(1:N_TRACKED_SPECIES),STAT=IZERO)
5994 CALL ChkMemErr('READ','SURFACE',IZERO) ; SP%RAMP_MF = 'null'
5995
5996 ! Read the user defined SURF lines
5997
5998 CALL SET_SURF_DEFAULTS
5999
6000 IF (SP%USER_DEFINED) THEN
6001 CALL CHECKREAD('SURF',LU_INPUT,IOS)
6002 READ(LU_INPUT,SURF)
6003 ENDIF
6004
6005 ! Check to make sure that a DEFAULT SURF has an ID
6006
6007 IF (DEFAULT) THEN
6008 IF (ID=='null') ID = 'DEFAULT SURF'
6009 SURF_DEFAULT = TRIM(ID)
6010 ENDIF
6011
6012 IF (EVAC_DEFAULT) EVAC_SURF_DEFAULT = TRIM(ID)
6013
6014 ! Look for special TGA_ANALYSIS=.TRUE. to indicate that only a TGA analysis is to be done
6015
6016 IF (TGA_ANALYSIS) THEN
6017 GEOMETRY = 'CARTESIAN'
6018 LENGTH = 0.1
6019 WIDTH = 0.1
6020 BACKING = 'INSULATED'
6021 IF (THICKNESS(2)>0._EB) THEN
6022 WRITE(MESSAGE, '(A)') 'ERROR: IF TGA_ANALYSIS=.TRUE., the surface can only be one layer thick'
6023 CALL SHUTDOWN(MESSAGE) ; RETURN
6024 ENDIF
6025 THICKNESS = 1.E-6._EB
6026 HEAT_TRANSFER_COEFFICIENT = 1000._EB
6027 MINIMUM_LAYER_THICKNESS = 1.E-12._EB
6028 TGA_SURF_INDEX = N
6029 INITIAL_RADIATION_ITERATIONS = 0
6030 ENDIF
6031
6032 ! Vegetation parameters
6033
6034 IF (VEGETATION) WFDS_BNDRY_FUEL = .TRUE.
6035
6036 ! Level set vegetation fire spread specific
6037 SP%VEG_LSET_SPREAD = VEG_LEVEL_SET_SPREAD
6038 SP%VEG_LSET_ROS_HEAD = VEG_LSET_ROS_HEAD !head fire rate of spread m/s
6039 SP%VEG_LSET_ELLIPSE_HEAD = VEG_LSET_ELLIPSE_HEAD !no-wind, no-slope ros for elliptical model in level set
6040 SP%VEG_LSET_ROS_FLANK = VEG_LSET_ROS_FLANK !flank fire rate of spread
6041 SP%VEG_LSET_ROS_BACK = VEG_LSET_ROS_BACK !back fire rate of spread
6042 SP%VEG_LSET_WIND_EXP = VEG_LSET_WIND_EXP !exponent on wind cosine in ROS formula
6043 SP%VEG_LSET_SIGMA = VEG_LSET_SIGMA * 0.01 !SAV for Farsite emulation in LSET converted to 1/cm
6044 SP%VEG_LSET_HT = VEG_LSET_HT
6045 SP%VEG_LSET_BETA = VEG_LSET_BETA
6046 SP%VEG_LSET_ELLIPSE = VEG_LSET_ELLIPSE
6047 SP%VEG_LSET_TAN2 = VEG_LSET_TAN2
6048
6049 ! Boundary Vegetation specific
6050
6051 SP%VEGETATION = VEGETATION !T or F
6052 SP%VEG_NO_BURN = VEGETATION_NO_BURN
6053 IF (WIND_ONLY) SP%VEG_NO_BURN = .TRUE.
6054 ! IF (SP%VEGETATION) ADIABATIC = .TRUE.
6055 SP%VEG_CHARFRAC = VEGETATION_CHAR_FRACTION
6056 SP%VEG_MOISTURE = VEGETATION_MOISTURE
6057 SP%VEG_HEIGHT = VEGETATION_HEIGHT
6058 SP%VEG_INITIAL_TEMP = VEGETATION_INITIAL_TEMP
6059 SP%VEG_GROUND_TEMP = VEGETATION_GROUND_TEMP
6060 IF (ABS(VEGETATION_GROUND_TEMP+99._EB)>TWO_EPSILON_EB) SP%VEG_GROUND_ZERO_RAD = .FALSE.
6061 SP%VEG_LOAD = VEGETATION_LOAD

```

Source Code files for edited portions of FDS

```

6062 SP%FIRELINE_MLR_MAX = FIRELINE_MLR_MAX
6063 ! SP%VEG_DEHYDRATION_RATE_MAX = SRF.VEG.DEHYDRATION_RATE_MAX
6064 SP%VEG_PACKING = VEGETATION_LOAD/VEGETATION_HEIGHT/VEGETATION_ELEMENT_DENSITY
6065 SP%VEG_SVRATIO = VEGETATION_SVRATIO
6066 SP%VEG_KAPPA = 0.25_EB*VEGETATION_SVRATIO*SP%VEG_PACKING
6067 SP%NVEGL = INT(1. + VEGETATION_HEIGHT*3._EB*SP%VEG_KAPPA)
6068 IF (VEGETATION_LAYERS > 0) SP%NVEGL = VEGETATION_LAYERS
6069 SP%VEG_DRAG_INI = VEGETATION_CD*SP%VEG_PACKING*SP%VEG_SVRATIO
6070 SP%VEG_LSET_IGNITE_T = VEGETATION_LSET_IGNITE_TIME
6071 IF (SP%VEG_LSET_IGNITE_T > -1._EB) SP%VEG_LSET_SPREAD = .TRUE.
6072 SP%VEG_LSET_QCON = -VEG.LSET.QCON*1000._EB !convert from kW/m^2 to W/m^2
6073 SP%VEG_LINEAR_DEGRAD = VEGETATION_LINEAR_DEGRAD
6074 SP%VEG_ARRHENIUS_DEGRAD = VEGETATION_ARRHENIUS_DEGRAD
6075 IF (VEGETATION_ARRHENIUS_DEGRAD) SP%VEG_LINEAR_DEGRAD = .FALSE.
6076
6077 ALLOCATE(SP%VEG_FUEL_FLUX_L(SP%NVEGL),STAT=IZERO)
6078 CALL ChkMemErr('READ.SURF','VEG.FUEL_FLUX.L',IZERO)
6079 ALLOCATE(SP%VEG_MOIST_FLUX_L(SP%NVEGL),STAT=IZERO)
6080 CALL ChkMemErr('READ.SURF','VEG.MOIST_FLUX.L',IZERO)
6081 ALLOCATE(SP%VEG_DIVQNET_L(SP%NVEGL),STAT=IZERO)
6082 CALL ChkMemErr('READ.SURF','VEG.DIVQNET',IZERO)
6083
6084 ALLOCATE(SP%VEG_FINCM_RADFCT_L(0:SP%NVEGL),STAT=IZERO) !add index for mult veg
6085 CALL ChkMemErr('READ.SURF','VEG.FINCM_RADFCT.L',IZERO)
6086 ALLOCATE(SP%VEG_FINCP_RADFCT_L(0:SP%NVEGL),STAT=IZERO)
6087 CALL ChkMemErr('READ','VEG.FINCP_RADFCT.L',IZERO)
6088
6089 ALLOCATE(SP%VEG_SEMISSP_RADFCT_L(0:SP%NVEGL,0:SP%NVEGL),STAT=IZERO) !add index for mult veg
6090 CALL ChkMemErr('READ','VEG.SEMISSP_RADFCT.L',IZERO)
6091 ALLOCATE(SP%VEG_SEMISSM_RADFCT_L(0:SP%NVEGL,0:SP%NVEGL),STAT=IZERO)
6092 CALL ChkMemErr('READ','VEG.SEMISSM_RADFCT.L',IZERO)
6093
6094 ! If a RADIUS is specified, consider it the same as THICKNESS(1)
6095
6096 IF (RADIUS>0._EB) THICKNESS(1) = RADIUS
6097
6098 ! Check SURF parameters for potential problems
6099
6100 LAYER_LOOP: DO IL=1,MAXLAYERS
6101 IF ((ADIABATIC.OR.NET_HEAT_FLUX<1.E12_EB.OR.ABS(CONVECTIVE_HEAT_FLUX)>TWO_EPSILON_EB.OR.TMP_FRONT>TMP) &
6102 .AND. MATL_ID(IL,1)/='null') THEN
6103 WRITE(MESSAGE,'(A)') 'ERROR: SURF '//TRIM(SP%D)//' cannot have a specified flux or temperature and a MATL_ID'
6104 CALL SHUTDOWN(MESSAGE) ; RETURN
6105 ENDIF
6106 IF (THICKNESS(IL)<=0._EB .AND. MATL_ID(IL,1)/='null') THEN
6107 WRITE(MESSAGE,'(A,I0)') 'ERROR: SURF '//TRIM(SP%D)//' must have a specified THICKNESS for Layer ',IL
6108 CALL SHUTDOWN(MESSAGE) ; RETURN
6109 ENDIF
6110 ENDDO LAYER_LOOP
6111
6112 IF ((GEOMETRY=='CYLINDRICAL' .OR. GEOMETRY=='SPHERICAL') .AND. RADIUS<0._EB .AND. THICKNESS(1)<0._EB) THEN
6113 WRITE(MESSAGE,'(A,A,A)') 'ERROR: SURF ',TRIM(SP%D),' needs a RADIUS or THICKNESS'
6114 CALL SHUTDOWN(MESSAGE) ; RETURN
6115 ENDIF
6116
6117 ! Identify the default SURF
6118
6119 IF (ID==SURF_DEFAULT) DEFAULT_SURF_INDEX = N
6120
6121 ! Pack SURF parameters into the SURFACE derived type
6122
6123 SF => SURFACE(N)
6124 SP%ADIABATIC = ADIABATIC
6125 SP%AUTO_IGNITION_TEMPERATURE = AUTO_IGNITION_TEMPERATURE + TMP
6126 SELECT CASE(BACKING)
6127 CASE('VOID')
6128 SP%BACKING = VOID
6129 CASE('INSULATED')
6130 SP%BACKING = INSULATED
6131 CASE('EXPOSED')
6132 SP%BACKING = EXPOSED
6133 CASE DEFAULT
6134 WRITE(MESSAGE,'(A)') 'ERROR: SURF '//TRIM(SP%D)//', BACKING '//TRIM(BACKING)//' not recognized'
6135 CALL SHUTDOWN(MESSAGE) ; RETURN
6136 END SELECT
6137 SP%BURN_AWAY = BURN_AWAY
6138 SP%CELL_SIZE_FACTOR = CELL_SIZE_FACTOR
6139 SP%CONVECTIVE_HEAT_FLUX = 1000._EB*CONVECTIVE_HEAT_FLUX
6140 SP%C_FORCED_CONSTANT = C_FORCED_CONSTANT
6141 SP%C_FORCED_PR_EXP = C_FORCED_PR_EXP
6142 SP%C_FORCED_RE = C_FORCED_RE
6143 SP%C_FORCED_RE_EXP = C_FORCED_RE_EXP
6144 SP%C_HORIZONTAL = C_HORIZONTAL
6145 SP%C_VERTICAL = C_VERTICAL
6146 SP%CONV_LENGTH = CONVECTION_LENGTH_SCALE
6147 SP%CONVERT_VOLUME_TO_MASS = CONVERT_VOLUME_TO_MASS
6148 IF (SP%CONVERT_VOLUME_TO_MASS .AND. TMP_FRONT<0._EB) THEN
6149 WRITE(MESSAGE,'(A,A,A)') 'ERROR: SURF ',TRIM(SP%D),' must specify TMP_FRONT for CONVERT_VOLUME_TO_MASS'

```

Source Code files for edited portions of FDS

```

6150 CALL SHUTDOWN(MESSAGE) ; RETURN
6151 ENDIF
6152 SP%NET_HEAT_FLUX = 1000._EB*NET_HEAT_FLUX
6153 SP%DUCT_PATH = DUCT_PATH
6154 SP%DT_INSERT = DT_INSERT
6155 SP%E_COEFFICIENT = E_COEFFICIENT
6156 SP%EMISSIVITY = EMISSIVITY
6157 SP%EMISSIVITY_BACK = EMISSIVITY_BACK
6158 SP%FIRE_SPREAD_RATE = SPREAD_RATE / TIME_SHRINK_FACTOR
6159 SP%FREE_SLIP = FREE_SLIP
6160 SP%NO_SLIP = NO_SLIP
6161 SP%FYI = FYI
6162 SP%EXTERNAL_FLUX = 1000._EB*EXTERNAL_FLUX
6163 SP%INNER_RADIUS = INNER_RADIUS
6164 SELECT CASE(GEOMETRY)
6165 CASE('CARTESIAN')
6166 SP%GEOMETRY = SURF_CARTESIAN
6167 IF (SP%WIDTH>0._EB) SP%BACKING = INSULATED
6168 CASE('CYLINDRICAL')
6169 SP%GEOMETRY = SURF_CYLINDRICAL
6170 IF (SP%INNER_RADIUS<TWO_EPSILON_EB) SP%BACKING = INSULATED
6171 CASE('SPHERICAL')
6172 SP%GEOMETRY = SURF_SPHERICAL
6173 IF (SP%INNER_RADIUS<TWO_EPSILON_EB) SP%BACKING = INSULATED
6174 CASE DEFAULT
6175 WRITE(MESSAGE,'(A,A,A)') 'ERROR: SURF ',TRIM(SP%D),' GEOMETRY not recognized'
6176 CALL SHUTDOWN(MESSAGE) ; RETURN
6177 END SELECT
6178 SP%HLV = 1000._EB*HEAT_OF_VAPORIZATION
6179 SELECT CASE(HEAT_TRANSFER_MODEL)
6180 CASE DEFAULT
6181 IF (ABS(C_FORCED_CONSTANT)>TWO_EPSILON_EB .OR. ABS(C_FORCED_RE)>TWO_EPSILON_EB) THEN
6182 SP%HEAT_TRANSFER_MODEL = H_CUSTOM
6183 ELSE
6184 SP%HEAT_TRANSFER_MODEL = H_DEFAULT
6185 ENDIF
6186 CASE('LOGLAW','LOG LAW')
6187 SP%HEAT_TRANSFER_MODEL = H_LOGLAW
6188 CASE('ABL')
6189 SP%HEAT_TRANSFER_MODEL = H_ABL
6190 CASE('RAYLEIGH')
6191 SP%HEAT_TRANSFER_MODEL = H_RAYLEIGH
6192 CASE('YUAN')
6193 SP%HEAT_TRANSFER_MODEL = H_YUAN
6194 END SELECT
6195 SP%HRRPUA = 1000._EB*HRRPUA
6196 SP%MLRPUA = MLRPUA
6197 SP%LAYER_DIVIDE = LAYER_DIVIDE
6198 IF (LEAK_PATH(2) < LEAK_PATH(1)) THEN
6199 SP%LEAK_PATH(2) = LEAK_PATH(1)
6200 SP%LEAK_PATH(1) = LEAK_PATH(2)
6201 ELSE
6202 SP%LEAK_PATH = LEAK_PATH
6203 ENDIF
6204 SP%LENGTH = LENGTH
6205 SP%MASS_FLUX = 0._EB
6206 SP%MASS_FLUX_VAR = MASS_FLUX_VAR
6207 SP%MASS_FRACTION = 0._EB
6208 SP%MAX_PRESSURE = MAX_PRESSURE
6209 SP%MINIMUM_LAYER_THICKNESS = MINIMUM_LAYER_THICKNESS
6210 SP%N_CELLS_MAX = N_CELLS_MAX
6211 IF (ANY(N_LAYER_CELLS_MAX<1)) THEN
6212 WRITE(MESSAGE,'(A,A,A)') 'ERROR: SURF ',TRIM(SP%D),' N_LAYER_CELLS_MAX must be >= 2'
6213 CALL SHUTDOWN(MESSAGE) ; RETURN
6214 ENDIF
6215 SP%N_LAYER_CELLS_MAX = N_LAYER_CELLS_MAX+1
6216 SP%NRA = NUMBER_RADIATION_ANGLES
6217 SP%NSB = NUMBER_SPECTRAL_BANDS
6218 ALLOCATE(SP%PARTICLE_INSERT_CLOCK(NMESHES),STAT=IZERO)
6219 CALL ChkMemErr('READ','PARTICLE_INSERT_CLOCK',IZERO)
6220 SP%PARTICLE_INSERT_CLOCK = T_BEGIN
6221 SP%NPPC = NPPC
6222 SP%PARTICLE_MASS_FLUX = PARTICLE_MASS_FLUX
6223 SP%PART_ID = PART_ID
6224 SP%PLE = PLE
6225 SELECT CASE (PROFILE)
6226 CASE('null')
6227 SP%PROFILE = 0
6228 CASE('ATMOSPHERIC')
6229 SP%PROFILE = ATMOSPHERIC_PROFILE
6230 CASE('PARABOLIC')
6231 SP%PROFILE = PARABOLIC_PROFILE
6232 CASE('BOUNDARY_LAYER')
6233 SP%PROFILE = BOUNDARY_LAYER_PROFILE
6234 CASE('RAMP')
6235 SP%PROFILE = RAMP_PROFILE
6236 END SELECT
6237 SP%RAMP_EF = EXTERNAL_FLUX_RAMP

```

Source Code files for edited portions of FDS

```

6238 SP%RAMP.MF           = 'null'
6239 SP%RAMP.Q            = RAMP.Q
6240 SP%RAMP.V            = RAMP.V
6241 SP%RAMP.T            = RAMP.T
6242 SP%RAMP.T.I         = RAMP.T.I
6243 SP%RAMP.PART        = RAMP.PART
6244 SP%RAMP.V.X         = RAMP.V.X
6245 SP%RAMP.V.Y         = RAMP.V.Y
6246 SP%RAMP.V.Z         = RAMP.V.Z
6247 IF (COLOR/= 'null') THEN
6248 IF (COLOR== 'INVISIBLE') THEN
6249 TRANSPARENCY = 0. .EB
6250 ELSE
6251 CALL COLOR2RGB(RGB,COLOR)
6252 ENDIF
6253 ENDIF
6254 IF (ANY(RGB< 0)) THEN
6255 RGB(1) = 255
6256 RGB(2) = 204
6257 RGB(3) = 102
6258 ENDIF
6259 IF (SP%ID== "OPEN") THEN
6260 RGB(1) = 255
6261 RGB(2) = 0
6262 RGB(3) = 255
6263 ENDIF
6264 SP%RGB               = RGB
6265 SP%ROUGHNESS        = ROUGHNESS
6266 SP%TRANSPARENCY     = TRANSPARENCY
6267 SP%STRETCH_FACTOR   = STRETCH_FACTOR
6268 SP%STRETCH_FACTOR   = MAX(1.0 .EB , SP%STRETCH_FACTOR)
6269 SP%TAU (TIME.HEAT)  = TAU.Q/TIME.SHRINK_FACTOR
6270 SP%TAU (TIME.VELO)  = TAU.V/TIME.SHRINK_FACTOR
6271 SP%TAU (TIME.TEMP)  = TAU.T/TIME.SHRINK_FACTOR
6272 SP%TAU (TIME.EFLUX) = TAU.EXTERNAL_FLUX/TIME.SHRINK_FACTOR
6273 SP%TAU (TIME.PART)  = TAU.PART/TIME.SHRINK_FACTOR
6274 SP%TEXTURE_MAP      = TEXTURE_MAP
6275 SP%TEXTURE_WIDTH    = TEXTURE_WIDTH
6276 SP%TEXTURE_HEIGHT  = TEXTURE_HEIGHT
6277 SP%THERMALLY_THICK_HT3D = HT3D
6278 SP%TMP_IGN         = IGNITION_TEMPERATURE + TMPM
6279 SP%VEL              = VEL
6280 SP%VEL.BULK         = VEL.BULK
6281 SP%VEL.GRAD         = VEL.GRAD
6282 SP%VEL.T            = VEL.T
6283 SP%VOLUME.FLOW     = VOLUME.FLOW
6284 SP%WIDTH            = WIDTH
6285 SP%Z0               = Z0
6286 SP%ZETA.FRONT       = ZETA.FRONT
6287 IF (HEAT_TRANSFER_COEFFICIENT.BACK < 0. .EB) HEAT_TRANSFER_COEFFICIENT.BACK=HEAT_TRANSFER_COEFFICIENT
6288 SP%H.FIXED          = HEAT_TRANSFER_COEFFICIENT
6289 SP%H.FIXED.B        = HEAT_TRANSFER_COEFFICIENT.BACK
6290 SP%H.M.FIXED        = MASS_TRANSFER_COEFFICIENT
6291 SP%XYZ              = XYZ
6292
6293 ! Convert inflowing MASS_FLUX_TOTAL to MASS_FLUX
6294
6295 IF (MASS_FLUX_TOTAL >= 0. .EB) THEN
6296 SP%MASS_FLUX_TOTAL = MASS_FLUX_TOTAL
6297 ELSE
6298 WRITE (MESSAGE, '(A,A,A)') 'ERROR: Problem with SURF: ', TRIM(SP%ID), &
6299 ' . MASS_FLUX_TOTAL should only be used for outflow. Use MASS_FLUX for inflow'
6300 CALL SHUTDOWN(MESSAGE) ; RETURN
6301 ENDIF
6302
6303 ! Error checking
6304
6305 IF (DEFAULT .AND. &
6306 (TRIM(ID)== 'OPEN' .OR. &
6307 TRIM(ID)== 'MIRROR' .OR. &
6308 TRIM(ID)== 'INTERPOLATED' .OR. &
6309 TRIM(ID)== 'PERIODIC' .OR. &
6310 TRIM(ID)== 'HVAC' .OR. &
6311 TRIM(ID)== 'MASSLESS TRACER' .OR. &
6312 TRIM(ID)== 'DROPLET' .OR. &
6313 TRIM(ID)== 'VEGETATION' .OR. &
6314 TRIM(ID)== 'EVACUATION.OUTFLOW' .OR. &
6315 TRIM(ID)== 'MASSLESS TARGET') ) THEN
6316 WRITE (MESSAGE, '(A,A,A)') 'ERROR: Problem with SURF: ', TRIM(SP%ID), '. Cannot set predefined SURF as DEFAULT'
6317 CALL SHUTDOWN(MESSAGE) ; RETURN
6318 ENDIF
6319
6320 IF (ABS(VOLUME_FLOW)>0. .EB) THEN
6321 WRITE (MESSAGE, '(A,A,A)') 'ERROR: Problem with SURF: ', TRIM(SP%ID), '. VOLUME_FLOW is deprecated; use VOLUME_FLOW'
6322 CALL SHUTDOWN(MESSAGE) ; RETURN
6323 ENDIF
6324
6325 IF (ANY(MASS_FLUX>0. .EB) .AND. ANY(MASS_FRACTION>0. .EB)) THEN

```

Source Code files for edited portions of FDS

```

6326 WRITE (MESSAGE,'(A,A,A)') 'ERROR: Problem with SURF: ',TRIM(SP%D),'. Cannot use both MASS_FLUX and MASS_FRACTION
6327 CALL SHUTDOWN(MESSAGE) ; RETURN
6328 ENDIF
6329
6330 IF (ANY(MASS_FLUX<0._EB) .OR. PARTICLE_MASS_FLUX<0._EB) THEN
6331 WRITE (MESSAGE,'(A,A,A)') 'ERROR: Problem with SURF: ',TRIM(SP%D),'. MASS_FLUX cannot be less than zero'
6332 CALL SHUTDOWN(MESSAGE) ; RETURN
6333 ENDIF
6334
6335 IF (ANY(MASS_FLUX>0._EB) .AND. ABS(VEL)>TWO_EPSILON_EB) THEN
6336 WRITE (MESSAGE,'(A,A,A)') 'ERROR: Problem with SURF: ',TRIM(SP%D),'. Cannot use both MASS_FLUX and VEL'
6337 CALL SHUTDOWN(MESSAGE) ; RETURN
6338 ENDIF
6339
6340 IF (ANY(MASS_FLUX>0._EB) .AND. ABS(MASS_FLUX_TOTAL)>TWO_EPSILON_EB) THEN
6341 WRITE (MESSAGE,'(A,A,A)') 'ERROR: Problem with SURF: ',TRIM(SP%D),'. Cannot use both MASS_FLUX and
        MASS_FLUX_TOTAL'
6342 CALL SHUTDOWN(MESSAGE) ; RETURN
6343 ENDIF
6344
6345 IF (ABS(MASS_FLUX_TOTAL)>TWO_EPSILON_EB .AND. ABS(VEL)>TWO_EPSILON_EB) THEN
6346 WRITE (MESSAGE,'(A,A,A)') 'ERROR: Problem with SURF: ',TRIM(SP%D),'. Cannot use both MASS_FLUX_TOTAL and VEL'
6347 CALL SHUTDOWN(MESSAGE) ; RETURN
6348 ENDIF
6349
6350 IF (ANY(MASS_FRACTION<0._EB)) THEN
6351 WRITE (MESSAGE,'(A,A,A)') 'ERROR: Problem with SURF: ',TRIM(SP%D),'. Cannot use a negative MASS_FRACTION'
6352 CALL SHUTDOWN(MESSAGE) ; RETURN
6353 ENDIF
6354
6355 IF (ANY(MASS_FLUX/=0._EB) .OR. ANY(MASS_FRACTION>0._EB)) THEN
6356 IF (SPEC_ID(1)=='null') THEN
6357 WRITE (MESSAGE,'(A,A,A)') 'ERROR: Problem with SURF: ',TRIM(SP%D),&
6358 '. Must define SPEC_ID when using MASS_FLUX or MASS_FRACTION'
6359 CALL SHUTDOWN(MESSAGE) ; RETURN
6360 ELSE
6361 DO NN=1,MAX_SPECIES
6362 IF (TRIM(SPEC_ID(NN))=='null') EXIT
6363 DO NNN=1,N_TRACKED_SPECIES
6364 IF (TRIM(SPECIES_MIXTURE(NN)%D)==TRIM(SPEC_ID(NN))) THEN
6365 SP%MASS_FLUX(NNN) = MASS_FLUX(NN)
6366 SP%MASS_FRACTION(NNN) = MASS_FRACTION(NN)
6367 SP%TAU(NNN) = TAU_MF(NN)/TIME_SHRINK_FACTOR
6368 SP%RAMP_MF(NNN) = RAMP_MF(NN)
6369 EXIT
6370 ENDIF
6371 IF (NN==N_TRACKED_SPECIES) THEN
6372 WRITE(MESSAGE,'(A,A,A,A)') 'ERROR: Problem with SURF: ',TRIM(SP%D), ' SPEC ',TRIM(SPEC_ID(NN)), ' not found'
6373 CALL SHUTDOWN(MESSAGE) ; RETURN
6374 ENDIF
6375 ENDDO
6376 ENDDO
6377 ENDIF
6378 IF (SUM(SP%MASS_FRACTION) > TWO_EPSILON_EB) THEN
6379 IF (SUM(SP%MASS_FRACTION) > 1._EB) THEN
6380 WRITE (MESSAGE,'(A,A,A)') 'ERROR: Problem with SURF: ',TRIM(SP%D),'. SUM(MASS_FRACTION) > 1'
6381 CALL SHUTDOWN(MESSAGE) ; RETURN
6382 ENDIF
6383 IF (SP%MASS_FRACTION(1) > 0._EB) THEN
6384 WRITE (MESSAGE,'(A,A,A)') 'ERROR: Problem with SURF: ',TRIM(SP%D), &
6385 '. Cannot use background species with MASS_FRACTION.'
6386 CALL SHUTDOWN(MESSAGE) ; RETURN
6387 ENDIF
6388 SP%MASS_FRACTION(1) = 1._EB - SUM(SP%MASS_FRACTION(2:N_TRACKED_SPECIES))
6389 ENDIF
6390 ENDIF
6391
6392 IF (SP%HEAT_TRANSFER_MODEL==H_RAYLEIGH .AND. GRAV<TWO_EPSILON_EB) THEN
6393 WRITE (MESSAGE,'(A,A,A)') 'ERROR: Problem with SURF: ',TRIM(SP%D),'. Cannot use a RAYLEIGH model with GRAV=0'
6394 CALL SHUTDOWN(MESSAGE) ; RETURN
6395 ENDIF
6396
6397 ! Set various logical parameters
6398
6399 IF (ABS(SP%VEL_T(1))>TWO_EPSILON_EB .OR. ABS(SP%VEL_T(2))>TWO_EPSILON_EB) SP%SPECIFIED_TANGENTIAL_VELOCITY = .
        TRUE.
6400
6401 ! Count the number of layers for the surface, and compile a LIST of all material names and indices
6402
6403 COMPUTE_EMISSIVITY = .FALSE.
6404 COMPUTE_EMISSIVITY_BACK = .FALSE.
6405 IF (SP%EMISSIVITY <0._EB) COMPUTE_EMISSIVITY = .TRUE.
6406 IF (SP%EMISSIVITY_BACK<0._EB) COMPUTE_EMISSIVITY_BACK = .TRUE.
6407
6408 SP%N_LAYERS = 0
6409 N_LIST = 0
6410 NAME_LIST = 'null'

```

```

6411 SP%THICKNESS = 0._EB
6412 SP%LAYER.MATL.INDEX = 0
6413 SP%LAYER.DENSITY = 0._EB
6414 INDEX.LIST = -1
6415 ALLOCATE(SP%LAYER.THICKNESS(MAX.LAYERS))
6416 SP%LAYER.THICKNESS = 0._EB
6417 COUNT.LAYERS: DO NL=1,MAX.LAYERS
6418 IF (THICKNESS(NL) < 0._EB) EXIT COUNT.LAYERS
6419 SP%N.LAYERS = SP%N.LAYERS + 1
6420 SP%LAYER.THICKNESS(NL) = THICKNESS(NL)
6421 SP%N.LAYER.MATL(NL) = 0
6422 EMISSIVITY = 0._EB
6423 COUNT.LAYER.MATL: DO NN=1,MAX.MATERIALS
6424 IF (MATL.ID(NL,NN) == 'null') CYCLE COUNT.LAYER.MATL
6425 N.LIST = N.LIST + 1
6426 NAME.LIST(N.LIST) = MATL.ID(NL,NN)
6427 SP%N.LAYER.MATL(NL) = SP%N.LAYER.MATL(NL) + 1
6428 SP%LAYER.MATL.NAME(NL,NN) = MATL.ID(NL,NN)
6429 SP%LAYER.MATL.FRAC(NL,NN) = MATL.MASS.FRAC(NL,NN)
6430 DO NNN=1,N.MATL
6431 IF (MATL.NAME(NNN)==NAME.LIST(N.LIST)) THEN
6432 INDEX.LIST(N.LIST) = NNN
6433 SP%LAYER.MATL.INDEX(NL,NN) = NNN
6434 SP%LAYER.DENSITY(NL) = SP%LAYER.DENSITY(NL)+SP%LAYER.MATL.FRAC(NL,NN)/MATERIAL(NNN)%RHO.S
6435 EMISSIVITY = EMISSIVITY + &
6436 MATERIAL(NNN)%EMISSIVITY*SP%LAYER.MATL.FRAC(NL,NN)/MATERIAL(NNN)%RHO.S ! volume based
6437 ENDF
6438 ENDDO
6439 IF (INDEX.LIST(N.LIST) < 0) THEN
6440 WRITE(MESSAGE, '(A,A,A,A,A,A)') 'ERROR: MATL.ID ',TRIM(NAME.LIST(N.LIST)),', on SURF: ',TRIM(SP%ID),', does not
exist'
6441 CALL SHUTDOWN(MESSAGE) ; RETURN
6442 ENDF
6443 ENDDO COUNT.LAYER.MATL
6444 IF (SP%LAYER.DENSITY(NL) > 0._EB) SP%LAYER.DENSITY(NL) = 1./SP%LAYER.DENSITY(NL)
6445 IF (COMPUTE.EMISSIVITY.BACK) SP%EMISSIVITY.BACK = EMISSIVITY*SP%LAYER.DENSITY(NL)
6446 IF (NL==1 .AND. COMPUTE.EMISSIVITY) SP%EMISSIVITY = EMISSIVITY*SP%LAYER.DENSITY(NL)
6447 SP%THICKNESS = SP%THICKNESS + SP%LAYER.THICKNESS(NL)
6448 ENDDO COUNT.LAYERS
6449
6450 ! Set emissivity to default value if no other method applies.
6451
6452 IF (SP%EMISSIVITY < 0._EB) SP%EMISSIVITY = EMISSIVITY.DEFAULT
6453 IF (SP%EMISSIVITY.BACK < 0._EB) SP%EMISSIVITY.BACK = EMISSIVITY.DEFAULT
6454
6455 ! Define mass flux division point
6456
6457 IF (SP%LAYER.DIVIDE < 0._EB) THEN
6458 IF (SP%BACKING==EXPOSED) THEN
6459 SP%LAYER.DIVIDE = 0.5._EB * REAL(SP%N.LAYERS,EB)
6460 ELSE
6461 SP%LAYER.DIVIDE = REAL(SP%N.LAYERS+1)
6462 ENDF
6463 ENDF
6464
6465 ! Add residue materials
6466
6467 DO I = 1,MAX.STEPS ! repeat the residue loop to find chained reactions - allows MAX.STEPS steps
6468 N.LIST2 = N.LIST
6469 DO NN = 1, N.LIST2
6470 MI=>MATERIAL(INDEX.LIST(NN))
6471 DO NR=1,MI%N.REACTIONS
6472 DO NNN=1,MI%N.RESIDUE(NR)
6473 IF (MI%RESIDUE.MATL.NAME(NNN,NR) == 'null') CYCLE
6474 IF (ANY(NAME.LIST==MI%RESIDUE.MATL.NAME(NNN,NR))) CYCLE
6475 N.LIST = N.LIST + 1
6476 IF (N.LIST>MAX.MATERIALS.TOTAL) THEN ; CALL SHUTDOWN('ERROR: Too many materials in the surface.') ; RETURN ;
ENDF
6477 NAME.LIST(N.LIST) = MI%RESIDUE.MATL.NAME(NNN,NR)
6478 INDEX.LIST(N.LIST) = MI%RESIDUE.MATL.INDEX(NNN,NR)
6479 ENDDO
6480 ENDDO
6481 ENDDO
6482 ENDDO
6483
6484 ! Eliminate multiply counted materials from the list
6485
6486 N.LIST2 = N.LIST
6487 WEED.MATL.LIST: DO NN=1,N.LIST
6488 DO NNN=1,NN-1
6489 IF (NAME.LIST(NNN)==NAME.LIST(NN)) THEN
6490 NAME.LIST(NN) = 'null'
6491 INDEX.LIST(NN) = 0
6492 N.LIST2 = N.LIST2-1
6493 CYCLE WEED.MATL.LIST
6494 ENDF
6495 ENDDO
6496 ENDDO WEED.MATL.LIST

```



Source Code files for edited portions of FDS

```

6497
6498 ! Allocate parameters indexed by layer
6499
6500 IF (TMP_FRONT >= -TMPM) TMPMIN = MIN(TMPMIN,TMP_FRONT+TMPM)
6501 IF (TMP_BACK >= -TMPM) TMPMIN = MIN(TMPMIN,TMP_BACK+TMPM)
6502 IF (ASSUMED.GAS.TEMPERATURE >= 0._EB) TMPMIN = MIN(TMPMIN,ASSUMED.GAS.TEMPERATURE)
6503
6504 SP%N.MATL = N.LIST2
6505 SP%THERMALLY.THICK = .FALSE.
6506 IF (SP%LAYER.DENSITY(1) > 0._EB) THEN
6507 SP%THERMALLY.THICK = .TRUE.
6508 SP%TMP_INNER = TMP_INNER + TMPM
6509 IF (SP%TMP_INNER(1) >= 0._EB) THEN
6510 SP%TMP_FRONT = SP%TMP_INNER(1)
6511 SP%TAU(TIME.TEMP) = 0._EB
6512 ELSE
6513 SP%TMP_FRONT = TMP_FRONT + TMPM
6514 ENDIF
6515 SP%TMP_BACK = TMP_BACK + TMPM
6516 ALLOCATE(SP%N.LAYER.CELLS(SP%N.LAYERS)) ! The number of cells in each layer
6517 ALLOCATE(SP%MIN.DIFFUSIVITY(SP%N.LAYERS)) ! The smallest diffusivity of materials in each layer
6518 ALLOCATE(SP%MATL.NAME(SP%N.MATL)) ! The list of all material names associated with the surface
6519 ALLOCATE(SP%MATL.INDEX(SP%N.MATL)) ! The list of all material indices associated with the surface
6520 ALLOCATE(SP%RESIDUE.INDEX(SP%N.MATL,MAX.MATERIALS,MAX.REACTIONS))! Each material associated with the surface has
    a RESIDUE
6521 ALLOCATE(SP%INTERNAL.HEAT.SOURCE(SP%N.LAYERS)) ! Volumetric source term set by the user
6522 ELSE
6523 SP%TMP_FRONT = TMP_FRONT + TMPM
6524 SP%TMP_INNER = SP%TMP_FRONT
6525 SP%TMP_BACK = SP%TMP_FRONT
6526 ENDIF
6527 DO NN = 1,SP%N.LAYERS
6528 IF (TMP_INNER(NN) >= -TMPM) TMPMIN = MIN(TMPMIN,TMP_INNER(NN)+TMPM)
6529 ENDDO
6530
6531 ! Store the names and indices of all materials associated with the surface
6532
6533 NNN = 0
6534 DO NN=1,N.LIST
6535 IF (NAME.LIST(NN) /= 'null') THEN
6536 NNN = NNN + 1
6537 SP%MATL.NAME(NNN) = NAME.LIST(NN)
6538 SP%MATL.INDEX(NNN) = INDEX.LIST(NN)
6539 ENDIF
6540 ENDDO
6541
6542 ! Store the RESIDUE indices
6543
6544 DO NN=1,SP%N.MATL
6545 ML => MATERIAL(SP%MATL.INDEX(NN))
6546 IF (M%N.REACTIONS>0 .AND. SP%TMP_IJGN<5000._EB) THEN
6547 WRITE(MESSAGE,'(A)') 'ERROR: SURF '//TRIM(SP%ID)// ' cannot have a REACting MATL and IGNITION.TEMPERATURE'
6548 CALL SHUTDOWN(MESSAGE) ; RETURN
6549 ENDIF
6550 DO NR=1,M%N.REACTIONS
6551 DO NRM=1,M%N.RESIDUE(NR)
6552 DO NNN=1,SP%N.MATL
6553 IF (M%RESIDUE.MATL.INDEX(NRM,NR)==SP%MATL.INDEX(NNN)) SP%RESIDUE.INDEX(NN,NRM,NR) = NNN
6554 ENDDO
6555 ENDDO
6556 ENDDO
6557 ENDDO
6558
6559 ! Specified source term
6560
6561 IF (SP%N.LAYERS>0 .AND. SP%LAYER.DENSITY(1)>0._EB) THEN
6562 SP%INTERNAL.HEAT.SOURCE(1:SP%N.LAYERS) = 1000._EB*INTERNAL.HEAT.SOURCE(1:SP%N.LAYERS)
6563 IF (MAXVAL(ABS(SP%INTERNAL.HEAT.SOURCE)) > TWO.EPSILON.EB) SP%SPECIFIED.HEAT.SOURCE = .TRUE.
6564 ENDIF
6565
6566 ! Thermal boundary conditions
6567
6568 IF (SP%ADIABATIC .AND. (SP%NET.HEAT.FLUX < 1.E12.EB .OR. ABS(SP%CONVECTIVE.HEAT.FLUX)>TWO.EPSILON.EB)) THEN
6569 WRITE(MESSAGE,'(A)') 'ERROR: SURF '//TRIM(SP%ID)//&
6570 ' cannot have both ADIABATIC and NET.HEAT.FLUX or CONVECTIVE.HEAT.FLUX'
6571 CALL SHUTDOWN(MESSAGE) ; RETURN
6572 ENDIF
6573 IF (SP%NET.HEAT.FLUX < 1.E12.EB .AND. ABS(SP%CONVECTIVE.HEAT.FLUX)>TWO.EPSILON.EB) THEN
6574 WRITE(MESSAGE,'(A)') 'ERROR: SURF '//TRIM(SP%ID)// ' cannot have both NET.HEAT.FLUX or CONVECTIVE.HEAT.FLUX'
6575 CALL SHUTDOWN(MESSAGE) ; RETURN
6576 ENDIF
6577 IF (SP%THERMALLY.THICK.HT3D) THEN
6578 IF ( SP%NET.HEAT.FLUX < 1.E12.EB .OR. ABS(SP%CONVECTIVE.HEAT.FLUX) > TWO.EPSILON.EB .OR. TMP_FRONT >= -TMPM )
        THEN
6579 WRITE(MESSAGE,'(A)') 'ERROR: SURF '//TRIM(SP%ID)// ' cannot have HT3D with specified TMP or FLUX bc'
6580 CALL SHUTDOWN(MESSAGE) ; RETURN
6581 ENDIF
6582 ENDIF

```



```

6583
6584 SP%THERMAL_BC_INDEX = SPECIFIED.TEMPERATURE
6585 IF (SP%ADIABATIC) THEN
6586 SP%THERMAL_BC_INDEX = NET_FLUX_BC
6587 SP%NET_HEAT_FLUX = 0._EB
6588 SP%EMISSIVITY = 1._EB
6589 ENDIF
6590 IF (SP%NET_HEAT_FLUX < 1.E12_EB) SP%THERMAL_BC_INDEX = NET_FLUX_BC
6591 IF (ABS(SP%CONVECTIVE_HEAT_FLUX)>TWO_EPSILON_EB) SP%THERMAL_BC_INDEX = CONVECTIVE_FLUX_BC
6592 IF (SP%THERMALLY_THICK) SP%THERMAL_BC_INDEX = THERMALLY_THICK
6593 IF (SP%THERMALLY_THICK_HT3D) SP%THERMAL_BC_INDEX = THERMALLY_THICK_HT3D
6594 IF (SP%PROFILE==ATMOSPHERIC_PROFILE) SP%THERMAL_BC_INDEX = INFLOW_OUTFLOW
6595 IF (SP%VEGETATION) SP%THERMAL_BC_INDEX = VEG_BNDRY_FUEL
6596
6597 ! Boundary layer profile
6598
6599 IF (SP%PROFILE==BOUNDARY_LAYER_PROFILE) THEN
6600 IF ( ABS(VEL_BULK)>ABS(VEL) ) THEN
6601 WRITE(MESSAGE,'(A)') 'ERROR: SURF '//TRIM(SP%ID)// ' VEL_BULK invalid, must have VEL_BULK <= VEL'
6602 CALL SHUTDOWN(MESSAGE) ; RETURN
6603 ENDIF
6604 ENDIF
6605
6606 ! Set convection length scale automatically for spheres. Set to 1 m for everything else.
6607
6608 IF (SP%CONV_LENGTH<0._EB) THEN
6609 SELECT CASE(SP%GEOMETRY)
6610 CASE(SURF_SPHERICAL) ; SP%CONV_LENGTH = 2._EB*(SP%INNER_RADIUS+SP%THICKNESS)
6611 CASE(SURF_CYLINDRICAL) ; SP%CONV_LENGTH = 2._EB+SP%THICKNESS
6612 CASE DEFAULT ; SP%CONV_LENGTH = 1._EB
6613 END SELECT
6614 ENDIF
6615
6616 ! Determine if REIGNITION_MODEL is to be used
6617
6618 IF (SP%AUTO_IGNITION_TEMPERATURE < 1.E20_EB ) REIGNITION_MODEL = .TRUE.
6619
6620 ! Ramps
6621
6622 IF (SP%RAMP_Q/= 'null') THEN
6623 CALL GET_RAMP_INDEX(SP%RAMP_Q, 'TIME', NR)
6624 SP%RAMP_INDEX(TIME_HEAT) = NR
6625 ELSE
6626 IF (SP%TAU(TIME_HEAT) > 0._EB) SP%RAMP_INDEX(TIME_HEAT) = TANHRAMP
6627 IF (SP%TAU(TIME_HEAT) < 0._EB) SP%RAMP_INDEX(TIME_HEAT) = TSQR_RAMP
6628 ENDIF
6629
6630 IF (SP%RAMP_V/= 'null') THEN
6631 CALL GET_RAMP_INDEX(SP%RAMP_V, 'TIME', NR)
6632 SP%RAMP_INDEX(TIME_VELO) = NR
6633 ELSE
6634 IF (SP%TAU(TIME_VELO) > 0._EB) SP%RAMP_INDEX(TIME_VELO) = TANHRAMP
6635 IF (SP%TAU(TIME_VELO) < 0._EB) SP%RAMP_INDEX(TIME_VELO) = TSQR_RAMP
6636 ENDIF
6637
6638 IF (SP%RAMP_T/= 'null') THEN
6639 CALL GET_RAMP_INDEX(SP%RAMP_T, 'TIME', NR)
6640 SP%RAMP_INDEX(TIME_TEMP) = NR
6641 ELSE
6642 IF (SP%TAU(TIME_TEMP) > 0._EB) SP%RAMP_INDEX(TIME_TEMP) = TANHRAMP
6643 IF (SP%TAU(TIME_TEMP) < 0._EB) SP%RAMP_INDEX(TIME_TEMP) = TSQR_RAMP
6644 ENDIF
6645
6646 IF (SP%RAMP_T_I/= 'null') THEN
6647 CALL GET_RAMP_INDEX(SP%RAMP_T_I, 'TIME', NR)
6648 SP%RAMP_INDEX(T_I_INDEX) = NR
6649 ENDIF
6650
6651 IF (SP%RAMP_EF/= 'null') THEN
6652 CALL GET_RAMP_INDEX(SP%RAMP_EF, 'TIME', NR)
6653 SP%RAMP_INDEX(TIME_EFLUX) = NR
6654 ELSE
6655 IF (SP%TAU(TIME_EFLUX) > 0._EB) SP%RAMP_INDEX(TIME_EFLUX) = TANHRAMP
6656 IF (SP%TAU(TIME_EFLUX) < 0._EB) SP%RAMP_INDEX(TIME_EFLUX) = TSQR_RAMP
6657 ENDIF
6658
6659 IF (SP%RAMP_PART/= 'null') THEN
6660 CALL GET_RAMP_INDEX(SP%RAMP_PART, 'TIME', NR)
6661 SP%RAMP_INDEX(TIME_PART) = NR
6662 ELSE
6663 IF (SP%TAU(TIME_PART) > 0._EB) SP%RAMP_INDEX(TIME_PART) = TANHRAMP
6664 IF (SP%TAU(TIME_PART) < 0._EB) SP%RAMP_INDEX(TIME_PART) = TSQR_RAMP
6665 ENDIF
6666
6667 IF (SP%RAMP_V_X/= 'null') THEN
6668 CALL GET_RAMP_INDEX(SP%RAMP_V_X, 'PROFILE', NR)
6669 SP%RAMP_INDEX(VELO_PROF_X) = NR
6670 ENDIF

```

```

6671
6672 IF (SP%RAMP.V.Y/= 'null') THEN
6673 CALL GET_RAMP_INDEX(SP%RAMP.V.Y, 'PROFILE', NR)
6674 SP%RAMP_INDEX(VELO_PROF.Y) = NR
6675 ENDIF
6676
6677 IF (SP%RAMP.V.Z/= 'null') THEN
6678 CALL GET_RAMP_INDEX(SP%RAMP.V.Z, 'PROFILE', NR)
6679 SP%RAMP_INDEX(VELO_PROF.Z) = NR
6680 ENDIF
6681
6682 ENDDO READ_SURF_LOOP
6683
6684 CONTAINS
6685
6686 SUBROUTINE SET_SURF_DEFAULTS
6687
6688 ADIABATIC = .FALSE.
6689 AUTO_IGNITION_TEMPERATURE = 1.E20_EB
6690 BACKING = 'EXPOSED'
6691 BURN_AWAY = .FALSE.
6692 CELL_SIZE_FACTOR = 1.0
6693 C_FORCED_CONSTANT = 0._EB
6694 C_FORCED_PR_EXP = 0._EB
6695 C_FORCED_RE = 0._EB
6696 C_FORCED_RE_EXP = 0._EB
6697 C_VERTICAL = 1.31_EB ! Vertical free convection (Holman, Table 7-2)
6698 C_HORIZONTAL = 1.52_EB ! Horizontal free convection
6699 COLOR = 'null'
6700 CONVECTIVE_HEAT_FLUX = 0._EB
6701 CONVECTION_LENGTH_SCALE = -1._EB
6702 CONVERT_VOLUME_TO_MASS = .FALSE.
6703 NET_HEAT_FLUX = 1.E12_EB
6704 DEFAULT = .FALSE.
6705 DT_INSERT = 0.01_EB
6706 DUCT_PATH = 0
6707 E_COEFFICIENT = 0._EB
6708 EMISSIVITY = -1._EB
6709 EMISSIVITY_DEFAULT = 0.9_EB
6710 EMISSIVITY_BACK = -1._EB
6711 EVAC_DEFAULT = .FALSE.
6712 EXTERNAL_FLUX = 0._EB
6713 EXTERNAL_FLUX_RAMP = 'null'
6714 FREE_SLIP = .FALSE.
6715 NO_SLIP = .FALSE.
6716 FYI = 'null'
6717 GEOMETRY = 'CARTESIAN'
6718 HEAT_OF_VAPORIZATION = 0._EB
6719 HEAT_TRANSFER_MODEL = 'null'
6720 HEAT_TRANSFER_COEFFICIENT = -1._EB
6721 HEAT_TRANSFER_COEFFICIENT_BACK = -1._EB
6722 MASS_TRANSFER_COEFFICIENT = -1._EB
6723 HRRPUA = 0._EB
6724 HT3D = .FALSE.
6725 ID = 'null'
6726 IGNITION_TEMPERATURE = 5000._EB
6727 INNER_RADIUS = 0._EB
6728 INTERNAL_HEAT_SOURCE = 0._EB
6729 LAYER_DIVIDE = -1._EB
6730 LEAK_PATH = -1
6731 LENGTH = -1._EB
6732 MASS_FLUX = 0._EB
6733 MASS_FLUX_TOTAL = 0._EB
6734 MASS_FLUX_VAR = -1._EB
6735 MASS_FRACTION = 0._EB
6736 MATL_ID = 'null'
6737 MATL_MASS_FRACTION = 0._EB
6738 MATL_MASS_FRACTION(:,1) = 1._EB
6739 MAX_PRESSURE = 1.E12_EB
6740 MINIMUM_LAYER_THICKNESS = 1.E-6_EB
6741 MLRPUA = 0._EB
6742 N_CELLS_MAX = 0
6743 N_LAYER_CELLS_MAX = 999
6744 NPPC = 1
6745 PARTICLE_MASS_FLUX = 0._EB
6746 PART_ID = 'null'
6747 PLE = 0.3_EB
6748 PROFILE = 'null'
6749 RADIUS = -1._EB
6750 RAMP_MF = 'null'
6751 RAMP_Q = 'null'
6752 RAMP_V = 'null'
6753 RAMP_T = 'null'
6754 RAMP_T_I = 'null'
6755 RAMP_PART = 'null'
6756 RAMP_V_X = 'null'
6757 RAMP_V_Y = 'null'
6758 RAMP_V_Z = 'null'

```

```

6759 RGB = -1
6760 IF (LES) ROUGHNESS = 0._EB !4.5E-5.EB ! meters, commercial steel
6761 IF (DNS) ROUGHNESS = 0._EB
6762 SPEC.ID = 'null'
6763 SPREAD.RATE = -1._EB
6764 STRETCH.FACTOR = 2._EB
6765 TAU.MF = 1._EB
6766 TAU.Q = 1._EB
6767 TAU.V = 1._EB
6768 TAU.T = 1._EB
6769 TAU.PART = 1._EB
6770 TAU.EXTERNAL.FLUX = 0.001.EB
6771 TEXTURE.MAP = 'null'
6772 TEXTURE.WIDTH = 1._EB
6773 TEXTURE.HEIGHT = 1._EB
6774 TGA.ANALYSIS = .FALSE.
6775 THICKNESS = -1._EB
6776 TMP.BACK = -TMPM-1._EB
6777 TMP.FRONT = -TMPM-1._EB
6778 TMP.INNER = -TMPM-1._EB
6779 TRANSPARENCY = 1._EB
6780 VEL = 0._EB
6781 VEL.BULK = 0._EB
6782 VEL.GRAD = -999999._EB
6783 VEL.T = 0._EB
6784 VOLUME.FLUX = 0._EB ! deprecated
6785 VOLUME.FLOW = 0._EB
6786 WIDTH = -1._EB
6787 XYZ = -1.E6.EB
6788 ZO = 10._EB
6789 ZETA.FRONT = INITIAL.UNMIXED.FRACTION
6790
6791 VEGETATION = .FALSE.
6792 VEGETATION.NO.BURN = .FALSE.
6793 VEGETATION.CDRAG = 1.0.EB
6794 VEGETATION.CHAR.FRACTION = 0.20.EB
6795 VEGETATION.ELEMENT.DENSITY = 512._EB !kg/m^3
6796 VEGETATION.HEIGHT = 0.50.EB !m
6797 VEGETATION.INITIAL.TEMP = TMPA-TMPM
6798 VEGETATION.GROUND.TEMP = -99._EB
6799 VEGETATION.LOAD = 0.30.EB !kg/m^2
6800 FIRELINE.MLR.MAX = 999. !kg/m/s w*R*(1-ChiChar)
6801 !SRF.VEG.DEHYDRATION.RATE.MAX = 999. !kg/m^2/s
6802 VEGETATION.LAYERS = 0
6803 VEGETATION.MOISTURE = 0.06.EB
6804 VEGETATION.SVRATIO = 12000.EB !1/m
6805 VEGETATION.L.SET.IGNITE.TIME = -1._EB
6806 VEGETATION.LINEAR.DEGRAD = .TRUE.
6807 VEGETATION.ARRHENIUS.DEGRAD = .FALSE.
6808 VEG.L.SET.ROS.HEAD = 0.0.EB
6809 VEG.L.SET.ELLIPSE.HEAD = 0.0.EB
6810 VEG.L.SET.ROS.FLANK = 0.0.EB
6811 VEG.L.SET.ROS.BACK = 0.0.EB
6812 VEG.LEVEL.SET.SPREAD = .FALSE.
6813 VEG.L.SET.WIND.EXP = 1.0.EB
6814 VEG.L.SET.ELLIPSE = .FALSE.
6815 VEG.L.SET.TAN2 = .FALSE.
6816 VEG.L.SET.HT = 0.0.EB
6817 VEG.L.SET.BETA = 0.0.EB
6818 VEG.L.SET.SIGMA = 0.0.EB
6819 VEG.L.SET.QCON = 0.0.EB
6820
6821 END SUBROUTINE SET_SURF_DEFAULTS
6822
6823 END SUBROUTINE READ_SURF
6824
6825 SUBROUTINE PROC_SURF_1
6826
6827 ! Go through the SURF types and process
6828
6829 USE MATH_FUNCTIONS, ONLY : GET_RAMP_INDEX
6830 INTEGER :: N, NSPC, NR, ILPC
6831 TYPE (LAGRANGIAN_PARTICLE_CLASS_TYPE), POINTER :: LPC=>NULL()
6832
6833 PROCESS_SURF_LOOP: DO N=0, N_SURF
6834
6835 SF => SURFACE(N)
6836
6837 ! Get ramps for the surface mass fraction and flux
6838
6839 DO NSPC=1, N_TRACKED_SPECIES
6840 IF (TRIM(SF%RAMP.MF(NSPC))/= 'null') THEN
6841 CALL GET_RAMP_INDEX(SF%RAMP.MF(NSPC), 'TIME', NR)
6842 SF%RAMP_INDEX(NSPC) = NR
6843 ELSE
6844 IF (SF%TAU(NSPC) > 0._EB) SF%RAMP_INDEX(NSPC) = TANHRAMP
6845 IF (SF%TAU(NSPC) < 0._EB) SF%RAMP_INDEX(NSPC) = TSQR.RAMP
6846

```

```

6847 ENDF
6848 ENDDO
6849
6850 ! Look for particle classes that use SURF for property info
6851
6852 DO ILPC=1,N,LAGRANGIAN.CLASSES
6853
6854   LPC=>LAGRANGIAN.PARTICLE.CLASS(ILPC)
6855
6856   IF (LPC%SURF_ID==SP%ID) THEN
6857     LPC%SURF_INDEX = N
6858
6859     IF (.NOT. LPC%SOLID_PARTICLE) CYCLE
6860     IF (LPC%DRAGLAW==SCREEN.DRAG) CYCLE
6861     SELECT CASE (SP%GEOMETRY)
6862     CASE(SURF.CARTESIAN)
6863       IF (SP%THICKNESS<=0..EB) THEN
6864         WRITE(MESSAGE,'(A,A,A)') 'ERROR: SURF ',TRIM(SP%ID),' needs a THICKNESS'
6865         CALL SHUTDOWN(MESSAGE) ; RETURN
6866       ENDF
6867       IF (.NOT. LPC%DRAGLAW==POROUS.DRAG) THEN
6868         IF (SP%LENGTH<=0..EB) THEN
6869           WRITE(MESSAGE,'(A,A,A)') 'ERROR: SURF ',TRIM(SP%ID),' needs a LENGTH'
6870           CALL SHUTDOWN(MESSAGE) ; RETURN
6871         ENDF
6872         IF (SP%WIDTH<=0..EB) THEN
6873           WRITE(MESSAGE,'(A,A,A)') 'ERROR: SURF ',TRIM(SP%ID),' needs a WIDTH'
6874           CALL SHUTDOWN(MESSAGE) ; RETURN
6875         ENDF
6876       ENDF
6877       CASE(SURF.CYLINDRICAL)
6878       IF (.NOT. LPC%DRAGLAW==POROUS.DRAG) THEN
6879         IF (SP%LENGTH <0..EB) THEN
6880           WRITE(MESSAGE,'(A,A,A)') 'ERROR: SURF ',TRIM(SP%ID),' needs a LENGTH'
6881           CALL SHUTDOWN(MESSAGE) ; RETURN
6882         ENDF
6883       ENDF
6884     END SELECT
6885   ENDF
6886 ENDDO
6887
6888 ENDDO PROCESS_SURF_LOOP
6889
6890 ! If a particle class uses a SURF line, make sure the SURF ID exists
6891
6892 DO ILPC=1,N,LAGRANGIAN.CLASSES
6893   LPC=>LAGRANGIAN.PARTICLE.CLASS(ILPC)
6894   IF (LPC%SURF_INDEX<0) THEN
6895     WRITE(MESSAGE,'(A,A,A)') 'ERROR: SURF ',TRIM(LPC%SURF_ID),' not found'
6896     CALL SHUTDOWN(MESSAGE) ; RETURN
6897   ENDF
6898 ENDDO
6899
6900 END SUBROUTINE PROC_SURF_1
6901
6902
6903 SUBROUTINE PROC_SURF_2
6904
6905 ! Go through the SURF types and process
6906
6907 INTEGER :: ILPC,N,NN,NNN,NL
6908 REAL(EB) :: ADJUSTED_LAYER_DENSITY, R_L(0:MAXLAYERS)
6909 INTEGER :: IVEG.L, HVEG.L, L_FUEL, L_GRAD
6910 REAL(EB) :: DETA.VEG, DZVEG.L, ETA.H, ETAFM.VEG, ETAFP.VEG
6911 LOGICAL :: BURNING, BLOWING, SUCKING
6912
6913 TYPE(LAGRANGIAN.PARTICLE.CLASS.TYPE), POINTER :: LPC=>NULL()
6914
6915 PROCESS_SURF_LOOP: DO N=0,N.SURF
6916
6917   SF => SURFACE(N)
6918   IF (SP%THERMALLY_THICK) ML => MATERIAL(SP%LAYER.MATL_INDEX(1,1))
6919
6920   SELECT CASE(SP%GEOMETRY)
6921   CASE(SURF.CARTESIAN) ; L_GRAD = 1
6922   CASE(SURF.CYLINDRICAL) ; L_GRAD = 2
6923   CASE(SURF.SPHERICAL) ; L_GRAD = 3
6924   END SELECT
6925
6926 ! Particle Information
6927
6928   SP%PART_INDEX = 0
6929   IF (SP%PART_ID/= 'null') THEN
6930     DO ILPC=1,N,LAGRANGIAN.CLASSES
6931       LPC=>LAGRANGIAN.PARTICLE.CLASS(ILPC)
6932       IF (LPC%ID==SP%PART_ID) SP%PART_INDEX = ILPC
6933     ENDDO
6934   IF (SP%PART_INDEX==0) THEN

```

Source Code files for edited portions of FDS

```

6935 WRITE(MESSAGE,'(A)') 'ERROR: PART_ID '//TRIM(SP%PART_ID)//' not found'
6936 CALL SHUTDOWN(MESSAGE) ; RETURN
6937 ENDIF
6938 PARTICLE_FILE=.TRUE.
6939 ENDIF
6940
6941 ! Determine if surface has internal radiation
6942
6943 SP%INTERNAL_RADIATION = .FALSE.
6944 DO NL=1,SP%N_LAYERS
6945 DO NN =1,SP%N_LAYER_MATL(NL)
6946 ML => MATERIAL(SP%LAYER_MATL_INDEX(NL,NN))
6947 IF (ML%KAPPA_S<5.0E4.EB) SP%INTERNAL_RADIATION = .TRUE.
6948 ENDDO
6949 ENDDO
6950
6951 ! In case of internal radiation, do not allow zero-emissivity
6952
6953 IF (SP%INTERNAL_RADIATION) THEN
6954 DO NL=1,SP%N_LAYERS
6955 DO NN =1,SP%N_LAYER_MATL(NL)
6956 ML => MATERIAL(SP%LAYER_MATL_INDEX(NL,NN))
6957 IF (ML%EMISSIVITY == 0..EB) THEN
6958 WRITE(MESSAGE,'(A)') 'ERROR: Zero emissivity of MAIL '//TRIM(MAILNAME(SP%LAYER_MATL_INDEX(NL,NN)))/' &
6959 ' is inconsistent with internal radiation in SURF '//TRIM(SP%D)//'. '
6960 CALL SHUTDOWN(MESSAGE) ; RETURN
6961 ENDIF
6962 ENDDO
6963 ENDDO
6964 ENDIF
6965
6966 ! Determine if the surface is combustible/burning
6967
6968 SP%PYROLYSIS_MODEL = PYROLYSIS_NONE
6969 BURNING = .FALSE.
6970 DO NL=1,SP%N_LAYERS
6971 DO NN=1,SP%N_LAYER_MATL(NL)
6972 NNN = SP%LAYER_MATL_INDEX(NL,NN)
6973 ML => MATERIAL(NNN)
6974 IF (ML%PYROLYSIS_MODEL/=PYROLYSIS_NONE) THEN
6975 SP%PYROLYSIS_MODEL = PYROLYSIS_PREDICTED
6976 SP%STRETCH_FACTOR(NL) = 1..EB
6977 IF (N_REACTIONS>0) THEN
6978 IF (REACTION(1)%FUEL_SMIX_INDEX>=0) THEN
6979 IF (ANY(ML%NU_SPEC(REACTION(1)%FUEL_SMIX_INDEX,:))>0..EB) THEN
6980 BURNING = .TRUE.
6981 SP%TAU(TIME_HEAT) = 0..EB
6982 ENDIF
6983 ENDIF
6984 ENDIF
6985 ENDIF
6986 ENDDO
6987 ENDDO
6988
6989 IF (SP%HRRPUA>0..EB .OR. SP%MLRPUA>0..EB) THEN
6990 IF (SP%PYROLYSIS_MODEL==PYROLYSIS_PREDICTED) THEN
6991 WRITE(MESSAGE,'(A)') 'ERROR: SURF '//TRIM(SP%D)//' has a specified HRRPUA or MLRPUA plus another pyrolysis model
6992 CALL SHUTDOWN(MESSAGE) ; RETURN
6993 ENDIF
6994 IF (N_REACTIONS > 1) THEN
6995 WRITE(MESSAGE,'(A)') 'ERROR: SURF '//TRIM(SP%D)//' has HRRPUA or MLRPUA set and there is more than one reaction'
6996 CALL SHUTDOWN(MESSAGE) ; RETURN
6997 ENDIF
6998 BURNING = .TRUE.
6999 SP%PYROLYSIS_MODEL = PYROLYSIS_SPECIFIED
7000 ENDIF
7001
7002 IF (BURNING .AND. N_REACTIONS==0) THEN
7003 WRITE(MESSAGE,'(A)') 'ERROR: SURF '//TRIM(SP%D)//' indicates burning, but there is no REAC line'
7004 CALL SHUTDOWN(MESSAGE) ; RETURN
7005 ENDIF
7006
7007 ! Make decisions based on whether there is forced ventilation at the surface
7008
7009 BLOWING = .FALSE.
7010 SUCKING = .FALSE.
7011 IF (SP%VEL<0..EB .OR. SP%VOLUME_FLOW<0..EB .OR. SP%MASS_FLUX_TOTAL < 0..EB) BLOWING = .TRUE.
7012 IF (SP%VEL>0..EB .OR. SP%VOLUME_FLOW>0..EB .OR. SP%MASS_FLUX_TOTAL > 0..EB) SUCKING = .TRUE.
7013 IF (BLOWING .OR. SUCKING) SP%SPECIFIED_NORMAL_VELOCITY = .TRUE.
7014 IF (SUCKING) SP%FREE_SLIP = .TRUE.
7015
7016 IF (BURNING .AND. (BLOWING .OR. SUCKING)) THEN
7017 WRITE(MESSAGE,'(A)') 'ERROR: SURF '//TRIM(SP%D)//' cannot have a specified velocity or volume flux'
7018 CALL SHUTDOWN(MESSAGE) ; RETURN
7019 ENDIF
7020
7021 ! Neumann for normal component of velocity

```

```

7022
7023 IF (SP%VEL_GRAD > -999998._EB) THEN
7024 SP%SPECIFIED.NORMAL_GRADIENT = .TRUE.
7025 SP%SPECIFIED.NORMAL_VELOCITY = .FALSE.
7026 ENDIF
7027
7028 ! Set predefined HRRPUA
7029
7030 BURNING_IF: IF (BURNING .AND. .NOT.ALL(EVACUATION_ONLY)) THEN
7031 IF (SP%HRRPUA>0._EB) THEN
7032 RN => REACTION(1)
7033 SP%MASS_FLUX(RN%FUEL_SMIX_INDEX) = SP%HRRPUA/RN%HOC_COMPLETE
7034 ENDIF
7035 IF (SP%MLRPUA>0._EB) THEN
7036 RN => REACTION(1)
7037 SP%MASS_FLUX(RN%FUEL_SMIX_INDEX) = SP%MLRPUA
7038 ENDIF
7039 I_FUEL = REACTION(1)%FUEL_SMIX_INDEX
7040 IF (SP%N_LAYERS > 0 .AND. SP%THERMALLY_THICK) THEN
7041 SP%ADJUST_BURN_RATE(I_FUEL) = MATERIAL(SP%MATL_INDEX(1))%ADJUST_BURN_RATE(I_FUEL,1)
7042 SP%MASS_FLUX(I_FUEL) = SP%MASS_FLUX(I_FUEL)/SP%ADJUST_BURN_RATE(I_FUEL) ! This is the true burning rate of the
      fuel.
7043 ENDIF
7044 SP%TAU(I_FUEL) = SP%TAU(TIME_HEAT)
7045 SP%RAMP_MF(I_FUEL) = SP%RAMP_Q
7046 SP%RAMP_INDEX(I_FUEL) = SP%RAMP_INDEX(TIME_HEAT)
7047 ENDIF BURNING_IF
7048
7049 ! Compute surface density
7050
7051 SP%SURFACE_DENSITY = 0._EB
7052 R.L(0) = SP%THICKNESS
7053 DO NL=1,SP%N_LAYERS
7054 ADJUSTED_LAYER_DENSITY = 0._EB
7055 MATL_LOOP:DO NN=1,SP%N_LAYER_MATL(NL)
7056 NNN = SP%LAYER_MATL_INDEX(NL,NN)
7057 ML => MATERIAL(NNN)
7058 ADJUSTED_LAYER_DENSITY = ADJUSTED_LAYER_DENSITY + SP%LAYER_MATL_FRAC(NL,NN)/ML%RHO_S
7059 ENDDO MATL_LOOP
7060 IF (ADJUSTED_LAYER_DENSITY > 0._EB) ADJUSTED_LAYER_DENSITY = 1./ADJUSTED_LAYER_DENSITY
7061 R.L(NL) = R.L(NL-1)-SP%LAYER_THICKNESS(NL)
7062 SP%SURFACE_DENSITY = SP%SURFACE_DENSITY + ADJUSTED_LAYER_DENSITY * &
7063 (R.L(NL-1)**L_GRAD-R.L(NL)**L_GRAD)/(REAL(L_GRAD,_EB)*SP%THICKNESS**(L_GRAD-1))
7064 ENDDO
7065
7066 IF ((ABS(SP%SURFACE_DENSITY) <= TWO_EPSILON_EB) .AND. SP%BURN_AWAY) THEN
7067 WRITE(MESSAGE,'(A,A)') 'WARNING: SURF ',TRIM(SP%ID),' has BURN_AWAY set but zero combustible density'
7068 IF (MYID==0) WRITE(LU_ERR,'(A)') TRIM(MESSAGE)
7069 ENDIF
7070
7071 ! Ignition Time
7072
7073 SP%T_IGN = T_BEGIN
7074 IF (SP%TMP_IGN<5000._EB) SP%T_IGN = HUGE(T_END)
7075 IF (SP%PYROLYSIS_MODEL==PYROLYSIS_PREDICTED) SP%T_IGN = HUGE(T_END)
7076
7077 ! Species Arrays and Method of Mass Transfer (SPECIES_BC_INDEX)
7078
7079 SP%SPECIES_BC_INDEX = NO_MASS_FLUX
7080
7081 IF (ANY(SP%MASS_FRACTION>0._EB) .AND. (ANY(ABS(SP%MASS_FLUX)>TWO_EPSILON_EB) .OR. SP%PYROLYSIS_MODEL/=
      PYROLYSIS_NONE)) THEN
7082 WRITE(MESSAGE,'(A)') 'ERROR: SURF '//TRIM(SP%ID)//' cannot specify mass fraction with mass flux and/or pyrolysis'
7083 CALL SHUTDOWN(MESSAGE) ; RETURN
7084 ENDIF
7085 IF (ANY(SP%MASS_FRACTION>0._EB) .AND. SUCKING) THEN
7086 WRITE(MESSAGE,'(A)') 'ERROR: SURF '//TRIM(SP%ID)//' cannot specify both mass fraction and outflow velocity'
7087 CALL SHUTDOWN(MESSAGE) ; RETURN
7088 ENDIF
7089 IF (ANY(SP%LEAK_PATH>=0) .AND. (BLOWING .OR. SUCKING .OR. SP%PYROLYSIS_MODEL/= PYROLYSIS_NONE)) THEN
7090 WRITE(MESSAGE,'(A)') 'ERROR: SURF '//TRIM(SP%ID)//' cannot leak and specify flow or pyrolysis at the same time'
7091 CALL SHUTDOWN(MESSAGE) ; RETURN
7092 ENDIF
7093 IF (ANY(ABS(SP%MASS_FLUX)>TWO_EPSILON_EB) .AND. (BLOWING .OR. SUCKING)) THEN
7094 WRITE(MESSAGE,'(A)') 'ERROR: SURF '//TRIM(SP%ID)//' cannot have both a mass flux and specified velocity'
7095 CALL SHUTDOWN(MESSAGE) ; RETURN
7096 ENDIF
7097
7098 IF (BLOWING .OR. SUCKING) SP%SPECIES_BC_INDEX = SPECIFIED_MASS_FRACTION
7099 IF (ANY(SP%MASS_FRACTION>0._EB)) SP%SPECIES_BC_INDEX = SPECIFIED_MASS_FRACTION
7100 IF (ANY(ABS(SP%MASS_FLUX)>TWO_EPSILON_EB) .OR. &
7101 SP%PYROLYSIS_MODEL==PYROLYSIS_PREDICTED) SP%SPECIES_BC_INDEX = SPECIFIED_MASS_FLUX
7102
7103 IF (SP%SPECIES_BC_INDEX==SPECIFIED_MASS_FRACTION) THEN
7104 IF (ALL(ABS(SP%MASS_FRACTION)< TWO_EPSILON_EB)) &
7105 SP%MASS_FRACTION(1:N_TRACKED_SPECIES) = SPECIES_MIXTURE(1:N_TRACKED_SPECIES)%ZZO
7106 ENDIF
7107

```

```

7108 ! Boundary fuel model for vegetation
7109
7110 IF (SP%VEGETATION) SP%SPECIES_BC_INDEX = SPECIFIED_MASS_FLUX
7111
7112 ! Texture map info
7113
7114 SP%SURF_TYPE = 0
7115 IF (SP%TEXTURE_MAP/= 'null') SP%SURF_TYPE = 1
7116
7117 ! Set BCs for various boundary types
7118
7119 SP%VELOCITY_BC_INDEX = WALL_MODEL_BC
7120 IF (DNS) SP%VELOCITY_BC_INDEX = NO_SLIP_BC
7121 IF (SP%FREE_SLIP) SP%VELOCITY_BC_INDEX = FREE_SLIP_BC
7122 IF (SP%NO_SLIP) SP%VELOCITY_BC_INDEX = NO_SLIP_BC
7123
7124 IF (N==OPEN_SURF_INDEX) THEN
7125 SP%THERMAL_BC_INDEX = INFLOW_OUTFLOW
7126 SP%SPECIES_BC_INDEX = INFLOW_OUTFLOW_MASS_FLUX
7127 SP%VELOCITY_BC_INDEX = FREE_SLIP_BC
7128 SP%SURF_TYPE = 2
7129 SP%EMISSIVITY = 1.0
7130 ENDIF
7131 IF (N==MIRROR_SURF_INDEX) THEN
7132 SP%THERMAL_BC_INDEX = NO_CONVECTION
7133 SP%SPECIES_BC_INDEX = NO_MASS_FLUX
7134 SP%VELOCITY_BC_INDEX = FREE_SLIP_BC
7135 SP%SURF_TYPE = -2
7136 SP%EMISSIVITY = 0.0
7137 ENDIF
7138 IF (N==INTERPOLATED_SURF_INDEX) THEN
7139 SP%THERMAL_BC_INDEX = INTERPOLATED_BC
7140 SP%SPECIES_BC_INDEX = INTERPOLATED_BC
7141 SP%VELOCITY_BC_INDEX = INTERPOLATED_VELOCITY_BC
7142 ENDIF
7143 IF (N==PERIODIC_SURF_INDEX) THEN
7144 SP%THERMAL_BC_INDEX = INTERPOLATED_BC
7145 SP%SPECIES_BC_INDEX = INTERPOLATED_BC
7146 SP%VELOCITY_BC_INDEX = INTERPOLATED_VELOCITY_BC
7147 ENDIF
7148 IF (N==PERIODIC_WIND_SURF_INDEX) THEN
7149 SP%THERMAL_BC_INDEX = INFLOW_OUTFLOW
7150 SP%SPECIES_BC_INDEX = INFLOW_OUTFLOW_MASS_FLUX
7151 SP%VELOCITY_BC_INDEX = INTERPOLATED_VELOCITY_BC
7152 ENDIF
7153 IF (N==HVAC_SURF_INDEX) THEN
7154 SP%THERMAL_BC_INDEX = HVAC_BOUNDARY
7155 SP%SPECIES_BC_INDEX = HVAC_BOUNDARY
7156 ENDIF
7157 IF (N==MASSLESS_TRACER_SURF_INDEX) THEN
7158 SP%NSA = 1
7159 SP%NSB = 1
7160 ENDIF
7161 IF (N==DROPLET_SURF_INDEX) THEN
7162 SP%NSA = 1
7163 SP%NSB = 1
7164 ENDIF
7165 IF (N==VEGETATION_SURF_INDEX) THEN
7166 SP%NSA = 1
7167 SP%NSB = 1
7168 ENDIF
7169 IF (N==EVACUATION_SURF_INDEX) THEN
7170 SP%THERMAL_BC_INDEX = INFLOW_OUTFLOW
7171 SP%SPECIES_BC_INDEX = SPECIFIED_MASS_FRACTION
7172 SP%SPECIFIED_NORMAL_VELOCITY = .TRUE.
7173 SP%FREE_SLIP = .TRUE.
7174 SP%VELOCITY_BC_INDEX = FREE_SLIP_BC
7175 SP%VEL = +0.000001.0 ! VEL
7176 SP%TAU(TIME_VELO) = 0.1.0 ! TAU.V
7177 SP%RAMP_INDEX(TIME_VELO) = TANHRAMP
7178 ENDIF
7179 IF (N==MASSLESS_TARGET_SURF_INDEX) THEN
7180 SP%EMISSIVITY = 1.0
7181 ENDIF
7182
7183 ! Do not allow N_LAYERS or N_CELLS_INI to be zero
7184
7185 IF (.NOT. SP%THERMALLY_THICK) THEN
7186 SP%N_LAYERS = 1
7187 SP%N_CELLS_MAX = 1
7188 SP%N_CELLS_INI = 1
7189 SP%N_MATL = 1
7190 ALLOCATE(SP%N_LAYER_CELLS(SP%N_LAYERS))
7191 ALLOCATE(SP%X_S(0:SP%N_CELLS_MAX))
7192 SP%X_S(0) = 0.0
7193 SP%X_S(1) = SP%THICKNESS
7194 ALLOCATE(SP%RHO_0(0:SP%N_CELLS_MAX+1,SP%N_MATL))
7195 SP%RHO_0 = 0.0

```

```

7196 SP%TMP_INNER(:) = TMPA
7197 ENDIF
7198
7199 ! Boundary surface vegetation
7200
7201 DZVEG.L = SP%VEG.HEIGHT/REAL(SP%NVEG.L,EB)
7202 DETA.VEG = SP%VEG.KAPPA*DZVEG.L
7203
7204 ! Factors for computing decay of +/- incident fluxes
7205
7206 SP%VEG.FINCM_RADFCT.L(:) = 0.0_EB
7207 SP%VEG.FINCP_RADFCT.L(:) = 0.0_EB
7208 ETA.H = SP%VEG.KAPPA*SP%VEG.HEIGHT
7209 DO IVEG.L = 0,SP%NVEG.L
7210   ETAFM.VEG = IVEG.L*DETA.VEG
7211   ETAFP.VEG = ETA.H - ETAFM.VEG
7212   SP%VEG.FINCM_RADFCT.L(IVEG.L) = EXP(-ETAFM.VEG)
7213   SP%VEG.FINCP_RADFCT.L(IVEG.L) = EXP(-ETAFP.VEG)
7214 ENDDO
7215
7216 ! Integrand for computing +/- self emission fluxes
7217
7218 SP%VEG.SEMISSP_RADFCT.L(:, :) = 0.0_EB
7219 SP%VEG.SEMISSM_RADFCT.L(:, :) = 0.0_EB
7220 ! q+
7221 DO IIVEG.L = 0,SP%NVEG.L !grid coordinate
7222   DO IVEG.L = IIVEG.L,SP%NVEG.L !integrand index
7223     ETAFM.VEG = (IVEG.L-IIVEG.L)*DETA.VEG
7224     ETAFP.VEG = ETAFM.VEG + DETA.VEG
7225     SP%VEG.SEMISSP_RADFCT.L(IVEG.L,IIVEG.L) = EXP(-ETAFM.VEG) - EXP(-ETAFP.VEG)
7226   ENDDO
7227   ENDDO
7228 ! q-
7229 DO IIVEG.L = 0,SP%NVEG.L
7230   DO IVEG.L = 1,IIVEG.L
7231     ETAFM.VEG = (IIVEG.L-IIVEG.L)*DETA.VEG
7232     ETAFP.VEG = ETAFM.VEG + DETA.VEG
7233     SP%VEG.SEMISSM_RADFCT.L(IVEG.L,IIVEG.L) = EXP(-ETAFM.VEG) - EXP(-ETAFP.VEG)
7234   ENDDO
7235   ENDDO
7236
7237 ENDDO PROCESS_SURF_LOOP
7238
7239 END SUBROUTINE PROC_SURF_2
7240
7241
7242
7243 SUBROUTINE PROC_WALL
7244
7245 ! Set up 1-D grids and arrays for thermally-thick calcs
7246
7247 USE GEOMETRY.FUNCTIONS
7248 USE MATH.FUNCTIONS, ONLY: EVALUATE_RAMP
7249
7250 INTEGER :: SURF_INDEX,N,NL,II,IL,NN,N,CELLS_MAX
7251 REAL(EB) :: K.S_0,C.S_0,SMALLEST_CELL_SIZE(MAXLAYERS),SWELL_RATIO,DENSITY_MAX,DENSITY_MIN
7252
7253 ! Calculate ambient temperature thermal DIFFUSIVITY for each MATERIAL, to be used in determining number of solid
       cells
7254
7255 DO N=1,N_MATL
7256   ML => MATERIAL(N)
7257   IF (ML%K.S > 0._EB) THEN
7258     K.S_0 = ML%K.S
7259   ELSE
7260     K.S_0 = EVALUATE_RAMP(TMPA,0._EB,-NINT(ML%K.S))
7261   ENDIF
7262   IF (ML%C.S > 0._EB) THEN
7263     C.S_0 = ML%C.S
7264   ELSE
7265     C.S_0 = EVALUATE_RAMP(TMPA,0._EB,-NINT(ML%C.S))*1000._EB
7266   ENDIF
7267   ML%DIFFUSIVITY = K.S_0/(C.S_0*ML%RHOS)
7268   ENDDO
7269
7270 NWP_MAX = 0 ! For some utility arrays, need to know the greatest number of points of all surface types
7271
7272 ! Loop through all surfaces, looking for those that are thermally-thick (have layers).
7273 ! Compute smallest cell size for each layer such that internal cells double in size.
7274 ! Each layer should have an odd number of cells.
7275
7276 SURF_GRID_LOOP: DO SURF_INDEX=0,N_SURF
7277
7278   SF => SURFACE(SURF_INDEX)
7279   IF (SF%THERMAL_BC_INDEX /= THERMALLY_THICK) CYCLE SURF_GRID_LOOP
7280
7281   ! Compute number of points per layer, and then sum up to get total points for the surface
7282

```



Source Code files for edited portions of FDS

```

7283 SP%N_CELLS.INI = 0
7284 N_CELLS.MAX = 0
7285
7286 LAYER_LOOP: DO NL=1,SP%N_LAYERS
7287
7288 SP%MIN_DIFFUSIVITY(NL) = 1000000. .EB
7289 DO N = 1,SP%N_LAYER.MATL(NL)
7290 ML => MATERIAL(SP%LAYER.MATL.INDEX(NL,N))
7291 SP%MIN_DIFFUSIVITY(NL) = MIN(SP%MIN_DIFFUSIVITY(NL),ML%DIFFUSIVITY)
7292 ENDDO
7293
7294 DENSITY_MAX = 0. .EB
7295 DENSITY_MIN = 10000000. .EB
7296 DO N = 1,SP%N.MATL
7297 ML => MATERIAL(SP%MATL.INDEX(N))
7298 DO NN = 1,SP%N.LAYER.MATL(NL)
7299 IF ((ML%PYROLYSIS.MODEL==PYROLYSIS.SOLID.OR.ML%PYROLYSIS.MODEL==PYROLYSIS.VEGETATION) .AND. &
7300 SP%LAYER.MATL.INDEX(NL,NN)==SP%MATL.INDEX(N)) THEN
7301 DENSITY_MAX = MAX(DENSITY_MAX, SP%LAYER.MATL.FRAC(NL,NN)*SP%LAYER.DENSITY(NL))
7302 ENDFIF
7303 ENDDO
7304 DENSITY_MIN = MIN(DENSITY_MIN,ML%RHO.S)
7305 ENDDO
7306
7307 SWELL_RATIO = 1. .EB
7308 IF (SP%PYROLYSIS.MODEL==PYROLYSIS.PREDICTED .AND. DENSITY_MIN>TWO_EPSILON.EB) SWELL_RATIO = DENSITY_MAX/
7309 DENSITY_MIN
7310 SWELL_RATIO = MAX(1.0.EB, SWELL_RATIO)
7311
7312 ! Get highest possible number of cells for this layer
7313 CALL GET_N_LAYER_CELLS(SP%MIN_DIFFUSIVITY(NL),SWELL_RATIO*SP%LAYER.THICKNESS(NL),SP%STRETCH_FACTOR(NL), &
7314 SP%CELL_SIZE_FACTOR,SP%N_LAYER_CELLS.MAX(NL),SP%N_LAYER_CELLS(NL),SMALLEST_CELL_SIZE(NL))
7315 N_CELLS.MAX = N_CELLS.MAX + SP%N_LAYER_CELLS(NL)
7316
7317 ! Get initial number of cells for this layer
7318
7319 CALL GET_N_LAYER_CELLS(SP%MIN_DIFFUSIVITY(NL),SP%LAYER.THICKNESS(NL),SP%STRETCH_FACTOR(NL), &
7320 SP%CELL_SIZE_FACTOR,SP%N_LAYER_CELLS.MAX(NL),SP%N_LAYER_CELLS(NL),SMALLEST_CELL_SIZE(NL))
7321 SP%N_CELLS.INI= SP%N_CELLS.INI + SP%N_LAYER_CELLS(NL)
7322
7323 ENDDO LAYER_LOOP
7324
7325 IF (SP%N_CELLS.MAX==0) SP%N_CELLS.MAX = N_CELLS.MAX
7326
7327 ! Allocate arrays to hold x_s, 1/dx_s (center to center, RDXN), 1/dx_s (edge to edge, RDX)
7328
7329 NWP_MAX = MAX(NWP_MAX,SP%N_CELLS.MAX)
7330 ALLOCATE(SP%DX(1:SP%N_CELLS.MAX))
7331 ALLOCATE(SP%RDX(0:SP%N_CELLS.MAX+1))
7332 ALLOCATE(SP%RDXN(0:SP%N_CELLS.MAX))
7333 ALLOCATE(SP%DX.WGT(0:SP%N_CELLS.MAX))
7334 ALLOCATE(SP%X_S(0:SP%N_CELLS.MAX))
7335 ALLOCATE(SP%LAYER.INDEX(0:SP%N_CELLS.MAX+1))
7336 ALLOCATE(SP%MF.FRAC(1:SP%N_CELLS.MAX))
7337 ALLOCATE(SP%RHO.0(0:SP%N_CELLS.MAX+1,SP%N.MATL))
7338
7339 ! Compute node coordinates
7340
7341 CALL GET_WALL_NODE_COORDINATES(SP%N_CELLS.INI,SP%N_LAYERS,SP%N_LAYER_CELLS, &
7342 SMALLEST_CELL_SIZE(1:SP%N_LAYERS),SP%STRETCH_FACTOR(1:SP%N_LAYERS),SP%X_S)
7343
7344 CALL GET_WALL_NODE_WEIGHTS(SP%N_CELLS.INI,SP%N_LAYERS,SP%N_LAYER_CELLS,SP%LAYER.THICKNESS,SP%GEOMETRY, &
7345 SP%X_S,SP%LAYER.DIVIDE,SP%DX,SP%RDX,SP%RDXN,SP%DX.WGT,SP%DXF,SP%DXB,SP%LAYER.INDEX,SP%MF.FRAC,SP%INNER.RADIUS)
7346
7347 ! Initialize the material densities of the solid
7348
7349 SP%RHO.0 = 0. .EB
7350
7351 DO II=0,SP%N_CELLS.INI+1
7352 IL = SP%LAYER.INDEX(IL)
7353 IF (SP%TMP INNER(IL)<=0. .EB) SP%TMP INNER(IL) = TMPA
7354 DO NN=1,SP%N_LAYER.MATL(IL)
7355 DO N=1,SP%N.MATL
7356 IF (SP%LAYER.MATL.INDEX(IL,NN)==SP%MATL.INDEX(N)) &
7357 SP%RHO.0(IL,N) = SP%LAYER.MATL.FRAC(IL,NN)*SP%LAYER.DENSITY(IL)
7358 ENDDO
7359 ENDDO
7360 ENDDO
7361
7362 ENDDO SURF_GRID_LOOP
7363
7364 ALLOCATE(AAS(NWP_MAX),STAT=IZERO)
7365 CALL ChkMemErr('INIT','AAS',IZERO)
7366 ALLOCATE(CCS(NWP_MAX),STAT=IZERO)
7367 CALL ChkMemErr('INIT','CCS',IZERO)
7368 ALLOCATE(BBS(NWP_MAX),STAT=IZERO)
7369 CALL ChkMemErr('INIT','BBS',IZERO)

```

```

7370 ALLOCATE(DDS(NWP:MAX) ,STAT=IZERO)
7371 CALL ChkMemErr('INIT' , 'DDS' , IZERO)
7372 ALLOCATE(DDT(NWP:MAX) ,STAT=IZERO)
7373 CALL ChkMemErr('INIT' , 'DDT' , IZERO)
7374 ALLOCATE(K_S(0:NWP:MAX+1) ,STAT=IZERO)
7375 CALL ChkMemErr('INIT' , 'K_S' , IZERO)
7376 ALLOCATE(C_S(0:NWP:MAX+1) ,STAT=IZERO)
7377 CALL ChkMemErr('INIT' , 'C_S' , IZERO)
7378 ALLOCATE(Q_S(1:NWP:MAX) ,STAT=IZERO)
7379 CALL ChkMemErr('INIT' , 'Q_S' , IZERO)
7380 ALLOCATE(RHO_S(0:NWP:MAX+1) ,STAT=IZERO)
7381 CALL ChkMemErr('INIT' , 'RHO_S' , IZERO)
7382 ALLOCATE(RHOCCBAR(1:NWP:MAX) ,STAT=IZERO)
7383 CALL ChkMemErr('INIT' , 'RHOCCBAR' , IZERO)
7384 ALLOCATE(KAPPA_S(1:NWP:MAX) ,STAT=IZERO)
7385 CALL ChkMemErr('INIT' , 'KAPPA_S' , IZERO)
7386 ALLOCATE(X_S_NEW(0:NWP:MAX) ,STAT=IZERO)
7387 CALL ChkMemErr('INIT' , 'X_S_NEW' , IZERO)
7388 ALLOCATE(DX_S(1:NWP:MAX) ,STAT=IZERO)
7389 CALL ChkMemErr('INIT' , 'DX_S' , IZERO)
7390 ALLOCATE(RDX_S(0:NWP:MAX+1) ,STAT=IZERO)
7391 CALL ChkMemErr('INIT' , 'RDX_S' , IZERO)
7392 ALLOCATE(RDXN_S(0:NWP:MAX) ,STAT=IZERO)
7393 CALL ChkMemErr('INIT' , 'RDXN_S' , IZERO)
7394 ALLOCATE(R_S(0:NWP:MAX) ,STAT=IZERO)
7395 CALL ChkMemErr('INIT' , 'R_S' , IZERO)
7396 ALLOCATE(R_S_NEW(0:NWP:MAX) ,STAT=IZERO)
7397 CALL ChkMemErr('INIT' , 'R_S_NEW' , IZERO)
7398 ALLOCATE(DX.WGT_S(0:NWP:MAX) ,STAT=IZERO)
7399 CALL ChkMemErr('INIT' , 'DX.WGT_S' , IZERO)
7400 ALLOCATE(LAYER_INDEX(0:NWP:MAX+1) ,STAT=IZERO)
7401 CALL ChkMemErr('INIT' , 'LAYER_INDEX' , IZERO)
7402 ALLOCATE(MF_FRAC(1:NWP:MAX) ,STAT=IZERO)
7403 CALL ChkMemErr('INIT' , 'MF_FRAC' , IZERO)
7404 ALLOCATE(REGRID_FACTOR(1:NWP:MAX) ,STAT=IZERO)
7405 CALL ChkMemErr('INIT' , 'REGRID_FACTOR' , IZERO)
7406
7407 END SUBROUTINE PROC.WALL
7408
7409 SUBROUTINE READ.PRES
7410
7411
7412 USE SCRC, ONLY: SCARC.METHOD , SCARC.KRYLOV , SCARC.MULTIGRID , SCARC.SMOOTH , SCARC.PRECON, &
7413 SCARC.COARSE , SCARC.INITIAL , SCARC.ACCURACY , SCARC.DEBUG , &
7414 SCARC.MULTIGRID.CYCLE , SCARC.MULTIGRID.LEVEL , SCARC.MULTIGRID.COARSENING , &
7415 SCARC.MULTIGRID.ITERATIONS , SCARC.MULTIGRID.ACCURACY , SCARC.MULTIGRID.INTERPOL, &
7416 SCARC.KRYLOV.ITERATIONS , SCARC.KRYLOV.ACCURACY, &
7417 SCARC.SMOOTH.ITERATIONS , SCARC.SMOOTH.ACCURACY , SCARC.SMOOTH.OMEGA, &
7418 SCARC.PRECON.ITERATIONS , SCARC.PRECON.ACCURACY , SCARC.PRECON.OMEGA, &
7419 SCARC.COARSE.ITERATIONS , SCARC.COARSE.ACCURACY
7420
7421 CHARACTER(60) :: SOLVER='FFT'
7422
7423 NAMELIST /PRES/ CHECK_POISSON, FISHPAK_BC, ITERATION_SUSPEND_FACTOR, LAPLACE_PRESSURE_CORRECTION, &
7424 MAX_PRESSURE_ITERATIONS, PRESSURE_RELAX_TIME, PRESSURE_TOLERANCE, RELAXATION_FACTOR, &
7425 SCARC.METHOD , SCARC.KRYLOV , SCARC.MULTIGRID , SCARC.SMOOTH , SCARC.PRECON, &
7426 SCARC.COARSE , SCARC.INITIAL , SCARC.ACCURACY , SCARC.DEBUG , &
7427 SCARC.MULTIGRID.CYCLE, SCARC.MULTIGRID.LEVEL, SCARC.MULTIGRID.COARSENING , &
7428 SCARC.MULTIGRID.ITERATIONS , SCARC.MULTIGRID.ACCURACY , SCARC.MULTIGRID.INTERPOL, &
7429 SCARC.KRYLOV.ITERATIONS , SCARC.KRYLOV.ACCURACY, &
7430 SCARC.SMOOTH.ITERATIONS , SCARC.SMOOTH.ACCURACY , SCARC.SMOOTH.OMEGA, &
7431 SCARC.PRECON.ITERATIONS , SCARC.PRECON.ACCURACY , SCARC.PRECON.OMEGA, &
7432 SCARC.COARSE.ITERATIONS , SCARC.COARSE.ACCURACY, &
7433 SOLVER, SUSPEND_PRESSURE_ITERATIONS, VELOCITY_TOLERANCE
7434
7435 ! Read the single PRES line
7436
7437 REWIND(LU.INPUT) ; INPUT_FILE_LINE_NUMBER = 0
7438 READ_LOOP: DO
7439 CALL CHECKREAD('PRES' , LU.INPUT, IOS)
7440 IF (IOS==1) EXIT READ_LOOP
7441 READ(LU.INPUT, PRES, END=23, ERR=24, IOSTAT=IOS)
7442 24 IF (IOS>0) THEN
7443 CALL SHUTDOWN('ERROR: Problem with PRES line') ; RETURN
7444 ENDF
7445 ENDDO READ_LOOP
7446 23 REWIND(LU.INPUT) ; INPUT_FILE_LINE_NUMBER = 0
7447
7448 ! Given the chosen SOLVER, define internal variable PRES.METHOD:
7449
7450 SELECT CASE(TRIM(SOLVER))
7451 CASE('SCARC')
7452 PRES.METHOD = 'SCARC'
7453 ITERATE_PRESSURE = .FALSE.
7454 IF (SCARC.METHOD == 'null') SCARC.METHOD = 'KRYLOV' ! Taken as default for SCARC when SOLVER is SCARC and
7455 ! SCARC.METHOD is not defined.
7456 CASE('GLMAT')
7457 PRES.METHOD = 'GLMAT'

```

```

7458 GLMAT.SOLVER = .TRUE.
7459 PRES.ON.WHOLE.DOMAIN = .FALSE.
7460
7461 CASE('GLMAT IBM')
7462 PRES.METHOD = 'GLMAT'
7463 GLMAT.SOLVER = .TRUE.
7464 PRES.ON.WHOLE.DOMAIN = .TRUE.
7465
7466 CASE DEFAULT
7467 ! Nothing to do. By default PRES.METHOD is set to 'FFT' in cons.f90
7468 END SELECT
7469
7470 ! Determine how many pressure iterations to perform per half time step.
7471
7472 IF (VELOCITY.TOLERANCE>100..EB) THEN
7473 ITERATE.PRESSURE = .FALSE.
7474 ELSE
7475 ITERATE.PRESSURE = .TRUE.
7476 IF (VELOCITY.TOLERANCE>TWO.EPSILON.EB .OR. PRESSURE.TOLERANCE>TWO.EPSILON.EB .OR. MAX.PRESSURE.ITERATIONS/=10) &
7477 SUSPEND.PRESSURE.ITERATIONS=.FALSE.
7478 IF (VELOCITY.TOLERANCE<TWO.EPSILON.EB) VELOCITY.TOLERANCE = 0.5..EB*CHARACTERISTIC.CELL.SIZE
7479 IF (PRESSURE.TOLERANCE<TWO.EPSILON.EB) PRESSURE.TOLERANCE = 20.0..EB/CHARACTERISTIC.CELL.SIZE**2
7480 ENDIF
7481
7482 IF (NMESHES>1 .AND. ANY(FISHPAK.BC==FISHPAK.BC.PERIODIC)) THEN
7483 CALL SHUTDOWN('ERROR: Cannot use FISHPAK.BC.PERIODIC with NMESHES>1') ; RETURN
7484 ENDIF
7485
7486 IF (ANY(FISHPAK.BC>0)) THEN
7487 CALL SHUTDOWN('ERROR: Cannot have FISHPAK.BC>0') ; RETURN
7488 ENDIF
7489
7490 END SUBROUTINE READ_PRES
7491
7492
7493 SUBROUTINE READ_RADI
7494
7495 USE RADCONS
7496 REAL(EB) :: BAND.LIMITS(MAX.NUMBER.SPECTRAL.BANDS+1), RADIATIVE.FRACTION=-1..EB
7497 NAMELIST /RADI/ ANGLE.INCREMENT, BAND.LIMITS, C.MAX, C.MIN, INITIAL.RADIATION.ITERATIONS, KAPPA0, NMIEANG,
7498 NUMBER.RADIATION.ANGLES,&
7499 PATH.LENGTH,&
7500 QR.CLIP, RADIATION, RADIATION.ITERATIONS, RADIATIVE.FRACTION, RADTMP, RTE.SOURCE.CORRECTION, TIME.STEP.INCREMENT,&
7501 WIDE.BAND.MODEL, MIE.MINIMUM.DIAMETER, MIE.MAXIMUM.DIAMETER, MIE.INDG, &
7502 NUMBER.INITIAL.ITERATIONS !, RADIATIVE.FRACTION.1 ! Backward compatibility Sesa-added RADIATIVE.FRACTION.1
7503 REAL(EB) THETA.LOW, THETA.UP
7504 INTEGER NRA, N
7505
7506 ! Set default values
7507 BAND.LIMITS(:) = -1..EB
7508
7509
7510 INITIAL.RADIATION.ITERATIONS = 3
7511 NUMBER.RADIATION.ANGLES = 100
7512 TIME.STEP.INCREMENT = 3
7513 IF (TWO.D) THEN
7514 NUMBER.RADIATION.ANGLES = 60
7515 TIME.STEP.INCREMENT = 2
7516 ENDIF
7517
7518 KAPPA0 = 0..EB
7519 RADTMP = 900..EB
7520 WIDE.BAND.MODEL = .FALSE.
7521 NMIEANG = 15
7522 PATH.LENGTH = -1.0..EB ! calculate path based on the geometry
7523 ANGLE.INCREMENT = -1
7524 MIE.MAXIMUM.DIAMETER = 0..EB
7525 MIE.MINIMUM.DIAMETER = 0..EB
7526 MIE.INDG = 50
7527 QR.CLIP = 10..EB ! kW/m3, lower bound for radiation source correction
7528
7529 ! Read radiation parameters
7530
7531 REWIND(LU.INPUT) ; INPUT.FILE.LINE.NUMBER = 0
7532 READ_LOOP: DO
7533 CALL CHECKREAD('RADI', LU.INPUT, IOS)
7534 IF (IOS==1) EXIT READ_LOOP
7535 READ(LU.INPUT, RADI, END=23, ERR=24, IOSTAT=IOS)
7536 24 IF (IOS>0) THEN
7537 CALL SHUTDOWN('ERROR: Problem with RADI line') ; RETURN
7538 ENDIF
7539 ENDDO READ_LOOP
7540 23 REWIND(LU.INPUT) ; INPUT.FILE.LINE.NUMBER = 0
7541
7542 IF (RADIATIVE.FRACTION >=0..EB) THEN
7543 CALL SHUTDOWN('ERROR: RADIATIVE.FRACTION is now specified on REAC.')
7544 RETURN

```

```

7545 ENDIF
7546
7547 RADIMP = RADIMP + TMPM
7548 QR_CLIP = QR_CLIP*1000._EB ! kW/m3 to W/m3
7549
7550 ! Define band parameters
7551
7552 IF (WIDE_BAND_MODEL) THEN
7553 NUMBER_SPECTRAL_BANDS = 6
7554 TIME_STEP_INCREMENT=MAX(1,TIME_STEP_INCREMENT)
7555 ANGLE_INCREMENT = 1
7556 UIIDIM=NUMBER_SPECTRAL_BANDS
7557 ELSE
7558 NUMBER_SPECTRAL_BANDS = 1
7559 IF (ANGLE_INCREMENT < 0) ANGLE_INCREMENT = MAX(1,MIN(5,NUMBER_RADIATION_ANGLES/15))
7560 UIIDIM = ANGLE_INCREMENT
7561 ENDIF
7562
7563 ! Define custom wavelength band limits
7564
7565 IF (ANY(BAND_LIMITS>0._EB)) THEN
7566 NUMBER_SPECTRAL_BANDS = COUNT(BAND_LIMITS>0._EB) - 1
7567 IF (NUMBER_SPECTRAL_BANDS<2) THEN ; CALL SHUTDOWN('ERROR: Need more spectral band limits. '); RETURN ; ENDIF
7568 IF (ANY((BAND_LIMITS(2:NUMBER_SPECTRAL_BANDS+1)-BAND_LIMITS(1:NUMBER_SPECTRAL_BANDS))<0._EB)) THEN
7569 CALL SHUTDOWN('ERROR: Spectral band limits should be given in ascending order. ')
7570 RETURN
7571 ENDIF
7572 ALLOCATE(WL_HIGH(1:NUMBER_SPECTRAL_BANDS))
7573 ALLOCATE(WL_LOW(1:NUMBER_SPECTRAL_BANDS))
7574 DO I=1,NUMBER_SPECTRAL_BANDS
7575 WL_LOW(I) = BAND_LIMITS(I)
7576 WL_HIGH(I)= BAND_LIMITS(I+1)
7577 ENDDO
7578
7579 TIME_STEP_INCREMENT=MAX(1,TIME_STEP_INCREMENT)
7580 ANGLE_INCREMENT = 1
7581 UIIDIM=NUMBER_SPECTRAL_BANDS
7582 ENDIF
7583
7584 ! Calculate actual number of radiation angles and determine the angular discretization
7585
7586 IF (.NOT.RADIATION) THEN
7587
7588 NUMBER_RADIATION_ANGLES = 1
7589 INITIAL_RADIATION_ITERATIONS = 1
7590
7591 ELSE
7592
7593 NRA = NUMBER_RADIATION_ANGLES
7594
7595 ! Determine the number of polar angles (theta)
7596
7597 IF (CYLINDRICAL) THEN
7598 NRT = NINT(SQRT(REAL(NRA)))
7599 ELSEIF (TWO_D) THEN
7600 NRT = 1
7601 ELSE
7602 NRT = 2*NINT(0.5._EB*1.17*REAL(NRA)**(1._EB/2.26))
7603 ENDIF
7604
7605 ! Determine number of azimuthal angles (phi)
7606
7607 ALLOCATE(NRP(1:NRT),STAT=IZERO)
7608 CALL ChkMemErr('INIT','NRP',IZERO)
7609
7610 N = 0
7611 DO I=1,NRT
7612 IF (CYLINDRICAL) THEN
7613 NRP(I) = NINT(REAL(NRA)/(REAL(NRT)))
7614 ELSEIF (TWO_D) THEN
7615 NRP(I) = 4*NINT(0.25._EB*REAL(NRA))
7616 ELSE
7617 THETA_LOW = PI*REAL(I-1)/REAL(NRT)
7618 THETA_UP = PI*REAL(I)/REAL(NRT)
7619 NRP(I) = NINT(0.5._EB*REAL(NRA)*(COS(THETA_LOW)-COS(THETA_UP)))
7620 NRP(I) = MAX(4,NRP(I))
7621 NRP(I) = 4*NINT(0.25._EB*REAL(NRP(I)))
7622 ENDIF
7623 N = N + NRP(I)
7624 ENDDO
7625 NUMBER_RADIATION_ANGLES = N
7626
7627 ENDIF
7628
7629 END SUBROUTINE READ_RADI
7630
7631
7632 SUBROUTINE READ_CLIP

```

```

7633
7634 REAL(EB) :: MAXIMUM.DENSITY,MINIMUM.DENSITY,MINIMUM.TEMPERATURE,MAXIMUM.TEMPERATURE
7635 NAMELIST /CLIP/ FYI,MAXIMUM.DENSITY,MAXIMUM.TEMPERATURE,MINIMUM.DENSITY,MINIMUM.TEMPERATURE
7636
7637 ! Check for user-defined mins and maxes.
7638
7639 MINIMUM.DENSITY      = -999..EB
7640 MAXIMUM.DENSITY      = -999..EB
7641 MINIMUM.TEMPERATURE = -999..EB
7642 MAXIMUM.TEMPERATURE = -999..EB
7643
7644 REWIND(LU.INPUT) ; INPUT_FILE.LINE_NUMBER = 0
7645 CLIP_LOOP: DO
7646 CALL CHECKREAD('CLIP',LU.INPUT,IOS)
7647 IF (IOS==1) EXIT CLIP_LOOP
7648 READ(LU.INPUT,CLIP,END=431,ERR=432,IOSTAT=IOS)
7649 432 IF (IOS>0) THEN ; CALL SHUTDOWN('ERROR: Problem with CLIP line') ; RETURN ; ENDIF
7650 ENDDO CLIP_LOOP
7651 431 REWIND(LU.INPUT) ; INPUT_FILE.LINE_NUMBER = 0
7652
7653 IF (MINIMUM.TEMPERATURE>=TMPM) TMPMIN = MINIMUM.TEMPERATURE + TMPM
7654 IF (MAXIMUM.TEMPERATURE>=TMPM) TMPMAX = MAXIMUM.TEMPERATURE + TMPM
7655
7656 IF (TMPMAX > 5000..EB) THEN ; CALL SHUTDOWN('MAXIMUM.TEMPERATURE cannot be greater than 4726.85 C (5000 K)') ;
    RETURN ; ENDIF
7657
7658 IF (MINIMUM.DENSITY>0..EB) THEN
7659 RHOMIN = MINIMUM.DENSITY
7660 ELSE
7661 RHOMIN = MIN(0.1..EB*RHOA,P.INF*MW.MIN/(R0*TMPMAX))
7662 ENDIF
7663
7664 IF (MAXIMUM.DENSITY>0..EB) THEN
7665 RHOMAX = MAXIMUM.DENSITY
7666 ELSE
7667 RHOMAX = 3.0..EB*P.INF*MW.MAX/(R0*MAX(TMPMIN,1..EB))
7668 ENDIF
7669
7670 END SUBROUTINE READ_CLIP
7671
7672
7673 SUBROUTINE READ_RAMP
7674
7675 REAL(EB) :: X,Z,T,F,FM
7676 INTEGER :: I,II,_NN,N,NUMBER.INTERPOLATION.POINTS,N.RES.RAMP
7677 CHARACTER(LABEL.LENGTH) :: DEVC.ID,CTRL.ID
7678 TYPE(RAMPS.TYPE), POINTER :: RP
7679 TYPE(RESERVED.RAMPS.TYPE), POINTER :: RRP
7680 NAMELIST /RAMP/ CTRL.ID,DEVC.ID,F,FYI,ID,NUMBER.INTERPOLATION.POINTS,T,X,Z
7681
7682 IF (N.RAMP==0) RETURN
7683 ALLOCATE(RAMPS(N.RAMP),STAT=IZERO)
7684 CALL ChkMemErr('READ','RAMPS',IZERO)
7685
7686 ! Count the number of points in each ramp
7687
7688 N.RES.RAMP = 0
7689
7690 COUNT_RAMP.POINTS: DO N=1,N.RAMP
7691
7692 RP => RAMPS(N)
7693
7694 IF (RAMP.ID(N)(1:5)=='RSRVD') THEN
7695 N.RES.RAMP = N.RES.RAMP + 1
7696 RRP => RESERVED.RAMPS(N.RES.RAMP)
7697 RP%RESERVED = .TRUE.
7698 RP%NUMBER.DATA.POINTS = RRP%NUMBER.DATA.POINTS
7699 ELSE
7700 REWIND(LU.INPUT) ; INPUT_FILE.LINE_NUMBER = 0
7701 RP%NUMBER.DATA.POINTS = 0
7702 SEARCH_LOOP: DO
7703 CALL CHECKREAD('RAMP',LU.INPUT,IOS)
7704 IF (IOS==1) EXIT SEARCH_LOOP
7705 READ(LU.INPUT,NML=RAMP,ERR=56,IOSTAT=IOS)
7706 IF (ID/=RAMP.ID(N)) CYCLE SEARCH_LOOP
7707 RP%NUMBER.DATA.POINTS = RP%NUMBER.DATA.POINTS + 1
7708 56 IF (IOS>0) THEN ; CALL SHUTDOWN('ERROR: Problem with RAMP '//TRIM(RAMP.ID(N)) ) ; RETURN ; ENDIF
7709 ENDDO SEARCH_LOOP
7710 ENDIF
7711
7712 IF (RP%NUMBER.DATA.POINTS<2) THEN
7713 IF (RP%NUMBER.DATA.POINTS==0) WRITE(MESSAGE,'(A,A,A)') 'ERROR: RAMP ',TRIM(RAMP.ID(N)), ' not found'
7714 IF (RP%NUMBER.DATA.POINTS==1) WRITE(MESSAGE,'(A,A,A)') 'ERROR: RAMP ',TRIM(RAMP.ID(N)), ' has only one point'
7715 CALL SHUTDOWN(MESSAGE) ; RETURN
7716 ENDIF
7717
7718 ENDDO COUNT_RAMP.POINTS
7719

```

```

7720 ! Read and process the ramp functions
7721
7722 N_RES_RAMP = 0
7723
7724 READ_RAMP_LOOP: DO N=1,N_RAMP
7725
7726 RP => RAMPS(N)
7727
7728 RP%DEVC_ID = 'null'
7729 RP%CTRL_ID = 'null'
7730 ALLOCATE(RP%INDEPENDENT_DATA(1:RP%NUMBER_DATA_POINTS))
7731 ALLOCATE(RP%DEPENDENT_DATA(1:RP%NUMBER_DATA_POINTS))
7732 NUMBER_INTERPOLATION_POINTS=5000
7733
7734 IF (RP%RESERVED) THEN
7735
7736 N_RES_RAMP = N_RES_RAMP + 1
7737 RRP => RESERVED_RAMPS(N_RES_RAMP)
7738 RP%INDEPENDENT_DATA(1:RP%NUMBER_DATA_POINTS) = RRP%INDEPENDENT_DATA(1:RRP%NUMBER_DATA_POINTS)
7739 RP%DEPENDENT_DATA(1:RP%NUMBER_DATA_POINTS) = RRP%DEPENDENT_DATA(1:RRP%NUMBER_DATA_POINTS)
7740 RP%NUMBER_INTERPOLATION_POINTS = NUMBER_INTERPOLATION_POINTS
7741
7742 ELSE
7743
7744 REWIND(LU_INPUT) ; INPUT_FILE_LINE_NUMBER = 0
7745 NN = 0
7746 SEARCH_LOOP2: DO
7747 DEVC_ID = 'null'
7748 CTRL_ID = 'null'
7749 X = -1.E6_EB
7750 Z = -1.E6_EB
7751 CALL CHECKREAD('RAMP',LU_INPUT,IOS)
7752 IF (IOS==1) EXIT SEARCH_LOOP2
7753 READ(LU_INPUT,RAMP)
7754 IF (ID/=RAMP_ID(N)) CYCLE SEARCH_LOOP2
7755 IF (RP%DEVC_ID == 'null') RP%DEVC_ID = DEVC_ID
7756 IF (RP%CTRL_ID == 'null') RP%CTRL_ID = CTRL_ID
7757 IF (X>-1.E5_EB) THEN
7758 RAMP_TYPE(N) = 'X COORDINATE'
7759 SPATIAL_GRAVITY_VARIATION = .TRUE.
7760 STRATIFICATION = .FALSE.
7761 T = X
7762 ENDF
7763 IF (Z>-1.E5_EB) THEN
7764 RAMP_TYPE(N) = 'Z COORDINATE'
7765 T = Z
7766 ENDF
7767 IF (RAMP_TYPE(N) == 'TEMPERATURE') T = T + TMFM
7768 IF (RAMP_TYPE(N) == 'TIME') T = T_BEGIN + (T-T_BEGIN)/TIME_SHRINK_FACTOR
7769 NN = NN+1
7770 RP%INDEPENDENT_DATA(NN) = T
7771 IF (NN>1) THEN
7772 IF (T<RP%INDEPENDENT_DATA(NN-1)) THEN
7773 WRITE(MESSAGE,'(A,A,A)') 'ERROR: RAMP ',TRIM(RAMP_ID(N)), ' variable T must be monotonically increasing'
7774 CALL SHUTDOWN(MESSAGE) ; RETURN
7775 ENDF
7776 ENDF
7777 RP%DEPENDENT_DATA(NN) = F
7778 RP%NUMBER_INTERPOLATION_POINTS = NUMBER_INTERPOLATION_POINTS
7779 ENDDO SEARCH_LOOP2
7780
7781 ENDF
7782
7783 RP%T_MIN = MINVAL(RP%INDEPENDENT_DATA)
7784 RP%T_MAX = MAXVAL(RP%INDEPENDENT_DATA)
7785 RP%SPAN = RP%T_MAX - RP%T_MIN
7786
7787 ENDDO READ_RAMP_LOOP
7788
7789 ! Set up interpolated ramp values in INTERPOLATED_DATA and get control or device index
7790
7791 DO N=1,N_RAMP
7792 RP => RAMPS(N)
7793 RP%RDT = REAL(RP%NUMBER_INTERPOLATION_POINTS,EB)/RP%SPAN
7794 ALLOCATE(RAMPS(N)%INTERPOLATED_DATA(0:RP%NUMBER_INTERPOLATION_POINTS+1))
7795 RAMPS(N)%INTERPOLATED_DATA(0) = RP%DEPENDENT_DATA(1)
7796 DO I=1,RP%NUMBER_INTERPOLATION_POINTS-1
7797 TM = RP%INDEPENDENT_DATA(1) + REAL(I,EB)/RP%RDT
7798 TLOOP: DO II=1,RP%NUMBER_DATA_POINTS-1
7799 IF (TM>RP%INDEPENDENT_DATA(II) .AND. TM<RP%INDEPENDENT_DATA(II+1)) THEN
7800 RP%INTERPOLATED_DATA(I) = RP%DEPENDENT_DATA(II) + (TM-RP%INDEPENDENT_DATA(II)) * &
7801 (RP%DEPENDENT_DATA(II+1)-RP%DEPENDENT_DATA(II))/(RP%INDEPENDENT_DATA(II+1)-RP%INDEPENDENT_DATA(II))
7802 EXIT TLOOP
7803 ENDF
7804 ENDDO TLOOP
7805 ENDDO
7806 RP%INTERPOLATED_DATA(RP%NUMBER_INTERPOLATION_POINTS) = RP%DEPENDENT_DATA(RP%NUMBER_DATA_POINTS)
7807 RP%INTERPOLATED_DATA(RP%NUMBER_INTERPOLATION_POINTS+1) = RP%DEPENDENT_DATA(RP%NUMBER_DATA_POINTS)

```

```

7808
7809 ! Get Device or Control Index
7810
7811 CALL SEARCH_CONTROLLER( 'RAMP', RP%CTRL_ID, RP%DEVC_ID, RP%DEVC_INDEX, RP%CTRL_INDEX, N)
7812
7813 ENDDO
7814
7815 END SUBROUTINE READ_RAMP
7816
7817
7818 SUBROUTINE READ_TABL
7819 REAL(EB) :: TABLE_DATA(9)
7820 INTEGER :: NN, N, I, J
7821 TYPE(TABLES_TYPE), POINTER :: TA=>NULL()
7822 NAMELIST /TABL/ FYI_ID, TABLE_DATA
7823
7824 IF (N_TABLE==0) RETURN
7825
7826 ALLOCATE(TABLES(N_TABLE), STAT=IZERO)
7827 CALL ChkMemErr( 'READ', 'TABLES', IZERO)
7828
7829 ! Count the number of points in each table
7830
7831 REWIND(LU_INPUT) ; INPUT_FILE_LINE_NUMBER = 0
7832 COUNT_TABLE_POINTS: DO N=1, N_TABLE
7833 TA => TABLES(N)
7834 REWIND(LU_INPUT) ; INPUT_FILE_LINE_NUMBER = 0
7835 TA%NUMBER_ROWS = 0
7836 SELECT CASE (TABLE_TYPE(N))
7837 CASE (SPRAY_PATTERN)
7838 TA%NUMBER_COLUMNS = 6
7839 CASE (PART_RADIATIVE_PROPERTY)
7840 TA%NUMBER_COLUMNS = 3
7841 CASE (TABLE_2D_TYPE, FLAME_SPEED_TABLE)
7842 TA%NUMBER_COLUMNS = 3
7843 END SELECT
7844 SEARCHLOOP: DO
7845 CALL CHECKREAD( 'TABL', LU_INPUT, IOS)
7846 IF (IOS==1) EXIT SEARCHLOOP
7847 TABLE_DATA = -999._EB
7848 READ(LU_INPUT, NML=TABL, ERR=56, IOSTAT=IOS)
7849 IF (ID/=TABLE_ID(N)) CYCLE SEARCHLOOP
7850 TA%NUMBER_ROWS = TA%NUMBER_ROWS + 1
7851 MESSAGE= 'null'
7852 SELECT CASE(TABLE_TYPE(N))
7853 CASE (SPRAY_PATTERN)
7854 IF (TABLE_DATA(1)<0._EB .OR. TABLE_DATA(1)>180._EB) THEN
7855 WRITE(MESSAGE, '(A,I0,A,A,A)') 'ERROR: Row ', TA%NUMBER_ROWS, ' of ', TRIM(TABLE_ID(N)), ' has a bad 1st latitude'
7856 CALL SHUTDOWN(MESSAGE) ; RETURN
7857 ENDIF
7858 IF (TABLE_DATA(2)<TABLE_DATA(1) .OR. TABLE_DATA(2)>180._EB) THEN
7859 WRITE(MESSAGE, '(A,I0,A,A,A)') 'ERROR: Row ', TA%NUMBER_ROWS, ' of ', TRIM(TABLE_ID(N)), ' has a bad 2nd latitude'
7860 CALL SHUTDOWN(MESSAGE) ; RETURN
7861 ENDIF
7862 IF (TABLE_DATA(3)<-180._EB .OR. TABLE_DATA(3)>360._EB) THEN
7863 WRITE(MESSAGE, '(A,I0,A,A,A)') 'ERROR: Row ', TA%NUMBER_ROWS, ' of ', TRIM(TABLE_ID(N)), ' has a bad 1st longitude'
7864 CALL SHUTDOWN(MESSAGE) ; RETURN
7865 ENDIF
7866 IF (TABLE_DATA(4)<TABLE_DATA(3) .OR. TABLE_DATA(4)>360._EB) THEN
7867 WRITE(MESSAGE, '(A,I0,A,A,A)') 'ERROR: Row ', TA%NUMBER_ROWS, ' of ', TRIM(TABLE_ID(N)), ' has a bad 2nd longitude'
7868 CALL SHUTDOWN(MESSAGE) ; RETURN
7869 ENDIF
7870 IF (TABLE_DATA(5)<0._EB) THEN
7871 WRITE(MESSAGE, '(A,I0,A,A,A)') 'ERROR: Row ', TA%NUMBER_ROWS, ' of ', TRIM(TABLE_ID(N)), ' has a bad velocity'
7872 CALL SHUTDOWN(MESSAGE) ; RETURN
7873 ENDIF
7874 IF (TABLE_DATA(6)<0._EB) THEN
7875 WRITE(MESSAGE, '(A,I0,A,A,A)') 'ERROR: Row ', TA%NUMBER_ROWS, ' of ', TRIM(TABLE_ID(N)), ' has a bad mass flow'
7876 CALL SHUTDOWN(MESSAGE) ; RETURN
7877 ENDIF
7878 CASE (PART_RADIATIVE_PROPERTY)
7879 IF (TABLE_DATA(1)<0._EB) THEN
7880 WRITE(MESSAGE, '(A,I0,A,A,A)') 'ERROR: Row ', TA%NUMBER_ROWS, ' of ', TRIM(TABLE_ID(N)), ' has a bad wave length'
7881 CALL SHUTDOWN(MESSAGE) ; RETURN
7882 ENDIF
7883 IF (TABLE_DATA(2)<=0._EB) THEN
7884 WRITE(MESSAGE, '(A,I0,A,A,A)') 'ERROR: Row ', TA%NUMBER_ROWS, ' of ', TRIM(TABLE_ID(N)), ' has a bad real index'
7885 CALL SHUTDOWN(MESSAGE) ; RETURN
7886 ENDIF
7887 IF (TABLE_DATA(3)< 0._EB) THEN
7888 WRITE(MESSAGE, '(A,I0,A,A,A)') 'ERROR: Row ', TA%NUMBER_ROWS, ' of ', TRIM(TABLE_ID(N)), ' has a bad complex index'
7889 CALL SHUTDOWN(MESSAGE) ; RETURN
7890 ENDIF
7891 CASE (TABLE_2D_TYPE, FLAME_SPEED_TABLE)
7892 IF (TA%NUMBER_ROWS == 1) THEN
7893 IF (INT(TABLE_DATA(1)) <= 0._EB) THEN
7894 WRITE(MESSAGE, '(A,I0,A,A,A)') 'ERROR: Row ', TA%NUMBER_ROWS, ' of ', TRIM(TABLE_ID(N)), '&'
7895 ' has < 1 x entries'

```

Source Code files for edited portions of FDS

```

7896 CALL SHUTDOWN(MESSAGE) ; RETURN
7897 ENDIF
7898 IF (INT(TABLE_DATA(2)) < 0.._EB) THEN
7899 WRITE(MESSAGE, '(A,I0,A,A,A)') 'ERROR: Row ',TA%NUMBERROWS,' of ',TRIM(TABLE_ID(N)),&
7900 ' has < 1 y entries'
7901 CALL SHUTDOWN(MESSAGE) ; RETURN
7902 ENDIF
7903 ENDIF
7904 END SELECT
7905
7906 56 IF (IOS>0) THEN ; CALL SHUTDOWN('ERROR: Problem with TABLE '//TRIM(TABLE_ID(N)) ) ; RETURN ; ENDIF
7907 ENDDO SEARCH_LOOP
7908 IF (TA%NUMBERROWS<=0) THEN
7909 WRITE(MESSAGE, '(A,A,A)') 'ERROR: TABLE ',TRIM(TABLE_ID(N)), ' not found'
7910 CALL SHUTDOWN(MESSAGE) ; RETURN
7911 ENDIF
7912 IF (TABLE_TYPE(N) == TABLE_2D_TYPE .OR. TABLE_TYPE(N) == FLAME.SPEED.TABLE) THEN
7913 IF (TA%NUMBERROWS<=1) THEN
7914 WRITE(MESSAGE, '(A,A,A)') 'ERROR: 2D TABLE ',TRIM(TABLE_ID(N)), ' must have at least one row of data'
7915 CALL SHUTDOWN(MESSAGE) ; RETURN
7916 ENDIF
7917 ENDIF
7918 ENDDO COUNT_TABLE_POINTS
7919
7920 READ_TABL_LOOP: DO N=1,N.TABLE
7921 TA => TABLE(N)
7922 ALLOCATE(TA%TABLE_DATA(TA%NUMBERROWS,TA%NUMBERCOLUMNS),STAT=IZERO)
7923 CALL ChkMemErr('READ',TA%TABLE_DATA,IZERO)
7924 REWIND(LU.INPUT) ; INPUT_FILE_LINE_NUMBER = 0
7925 NN = 0
7926 SEARCH_LOOP2: DO
7927 CALL CHECKREAD('TABL',LU.INPUT,IOS)
7928 IF (IOS==1) EXIT SEARCH_LOOP2
7929 READ(LU.INPUT,TABL)
7930 IF (ID/=TABLE_ID(N)) CYCLE SEARCH_LOOP2
7931 NN = NN+1
7932 TA%TABLE_DATA(NN,:) = TABLE_DATA(1:TA%NUMBERCOLUMNS)
7933 ENDDO SEARCH_LOOP2
7934 TABLE_2D_IF: IF (TABLE_TYPE(N)==TABLE_2D_TYPE) THEN
7935 IF (TA%NUMBERROWS-1/=INT(TA%TABLE_DATA(1,1))*INT(TA%TABLE_DATA(1,2))) THEN
7936 WRITE(MESSAGE, '(A,A,A)') 'ERROR: 2D TABLE ',TRIM(TABLE_ID(N)), ' is not rectangular'
7937 CALL SHUTDOWN(MESSAGE) ; RETURN
7938 ENDIF
7939 TA%X = MINVAL(TA%TABLE_DATA(2:TA%NUMBERROWS,1),1)
7940 TA%X = MAXVAL(TA%TABLE_DATA(2:TA%NUMBERROWS,1),1)
7941 TA%Y = MINVAL(TA%TABLE_DATA(2:TA%NUMBERROWS,2),1)
7942 TA%Y = MAXVAL(TA%TABLE_DATA(2:TA%NUMBERROWS,2),1)
7943 ALLOCATE(TA%X(INT(TA%TABLE_DATA(1,1))),STAT=IZERO)
7944 CALL ChkMemErr('READ',TA%X,IZERO)
7945 ALLOCATE(TA%Y(INT(TA%TABLE_DATA(1,2))),STAT=IZERO)
7946 CALL ChkMemErr('READ',TA%Y,IZERO)
7947 ALLOCATE(TA%Z(INT(TA%TABLE_DATA(1,1)),INT(TA%TABLE_DATA(1,2))),STAT=IZERO)
7948 CALL ChkMemErr('READ',TA%Z,IZERO)
7949 NN = 1
7950 TA%NUMBERROWS = INT(TA%TABLE_DATA(1,1))
7951 TA%NUMBERCOLUMNS = INT(TA%TABLE_DATA(1,2))
7952 DO I = 1, TA%NUMBERROWS
7953 DO J = 1, TA%NUMBERCOLUMNS
7954 NN = NN + 1
7955 IF (J==1) THEN
7956 TA%X(1)=TA%TABLE_DATA(NN,1)
7957 ELSE
7958 IF (TA%TABLE_DATA(NN,1) /= TA%X(1)) THEN
7959 WRITE(MESSAGE, '(A,A,A)') 'ERROR: 2D TABLE ',TRIM(TABLE_ID(N)), ' x value must be the same for each row'
7960 CALL SHUTDOWN(MESSAGE) ; RETURN
7961 ENDIF
7962 ENDIF
7963 IF (I==1) THEN
7964 TA%Y(J)=TA%TABLE_DATA(NN,2)
7965 ELSE
7966 IF (TA%TABLE_DATA(NN,2) /= TA%Y(J)) THEN
7967 WRITE(MESSAGE, '(A,A,A)') 'ERROR: 2D TABLE ',TRIM(TABLE_ID(N)), ' y value must be the same for each column'
7968 CALL SHUTDOWN(MESSAGE) ; RETURN
7969 ENDIF
7970 ENDIF
7971 TA%Z(I,J) = TA%TABLE_DATA(NN,3)
7972 ENDDO
7973 ENDDO
7974 IF (TABLE_TYPE(N)==FLAME.SPEED.TABLE) TA%Y=TA%Y+IMFM
7975 ENDIF TABLE_2D_IF
7976 ENDDO READ_TABL_LOOP
7977
7978 END SUBROUTINE READ_TABL
7979
7980 SUBROUTINE READ_OBST
7981
7982 USE GEOMETRY_FUNCTIONS, ONLY: BLOCK_CELL
7983

```



Source Code files for edited portions of FDS

```

7984 TYPE(OBSTRUCTION_TYPE), POINTER :: OB2=>NULL(), OB1=>NULL()
7985 TYPE(MULTIPLIER_TYPE), POINTER :: MR=>NULL()
7986 TYPE(OBSTRUCTION_TYPE), DIMENSION(:), ALLOCATABLE, TARGET :: TEMP_OBSTRUCTION
7987 INTEGER :: NM,NCM,N_OBST,O,NNN,IC,N,NN,NNNN,N_NEW,OBST,RGB(3),N_OBST_NEW,II,JJ,KK,EVAC,N
7988 CHARACTER(LABEL_LENGTH) :: ID,DEVC.ID,PROP.ID,SURF.ID,SURF.IDS(3),SURF.ID6(6),CTRL.ID,MULT.ID,MATL.ID
7989 CHARACTER(60) :: MESH.ID
7990 CHARACTER(25) :: COLOR
7991 LOGICAL :: EVACUATION_OBST,OVERLAY
7992 REAL(EB) :: TRANSPARENCY,XB1,XB2,XB3,XB4,XB5,XB6,BULK.DENSITY,VOL.ADJUSTED,VOL.SPECIFIED,UNDIVIDED.INPUT.AREA(3)
    &
7993 INTERNAL.HEAT.SOURCE
7994 LOGICAL :: EMBEDDED,THICKEN,PERMIT.HOLE,ALLOW.VENT,EVACUATION,REMOVABLE,BNDF.FACE(-3:3),BNDF.OBST,OUTLINE,
    NOTERRAIN,HT3D
7995 NAMELIST /OBST/ ALLOW.VENT,BNDF.FACE,BNDF.OBST,BULK.DENSITY,&
7996 COLOR,CTRL.ID,DEVC.ID,EVACUATION,FY1,HT3D,ID,INTERNAL.HEAT.SOURCE,MATL.ID,MESH.ID,MULT.ID,NOTERRAIN,&
7997 OUTLINE,OVERLAY,PERMIT.HOLE,PROP.ID,REMOVABLE,RGB,SURF.ID,SURF.ID6,SURF.IDS,TEXTURE.ORIGIN,THICKEN,&
7998 TRANSPARENCY,XB
7999
8000 MESHLOOP: DO NM=1,NMESHES
8001
8002 IF (PROCESS(NM)/=MYID .AND. MYID/=EVAC.PROCESS) CYCLE MESHLOOP
8003
8004 M=>MESHES(NM)
8005 CALL POINT_TO_MESH(NM)
8006 ! Count OBST lines
8007
8008 REWIND(LU.INPUT) ; INPUT.FILE.LINE.NUMBER = 0
8009 N.OBST = 0
8010 COUNT_OBST_LOOP: DO
8011 CALL CHECKREAD('OBST',LU.INPUT,IOS)
8012 IF (IOS==1) EXIT COUNT_OBST_LOOP
8013 MULT.ID = 'null'
8014 READ(LU.INPUT,NML=OBST,END=1,ERR=2,IOSTAT=IOS)
8015 N.OBST_NEW = 0
8016 IF (MULT.ID=='null') THEN
8017 N.OBST_NEW = 1
8018 ELSE
8019 DO N=1,N.MULT
8020 MR => MULTIPLIER(N)
8021 IF (MULT.ID==MR%ID) N.OBST_NEW = MR%N.COPIES
8022 ENDDO
8023 IF (N.OBST_NEW==0) THEN
8024 WRITE(MESSAGE,'(A,A,A,10,A,10)') 'ERROR: MULT line ', TRIM(MULT.ID), ' not found on OBST ', N.OBST+1,&
8025 ', line number',INPUT.FILE.LINE.NUMBER
8026 CALL SHUTDOWN(MESSAGE) ; RETURN
8027 ENDIF
8028 ENDF
8029 N.OBST = N.OBST + N.OBST_NEW
8030 2 IF (IOS>0) THEN
8031 WRITE(MESSAGE,'(A,10,A,10)') 'ERROR: Problem with OBST number',N.OBST+1,', line number',INPUT.FILE.LINE.NUMBER
8032 CALL SHUTDOWN(MESSAGE) ; RETURN
8033 ENDF
8034 ENDDO COUNT_OBST_LOOP
8035 1 REWIND(LU.INPUT) ; INPUT.FILE.LINE.NUMBER = 0
8036
8037 IF (EVACUATION_ONLY(NM)) CALL DEFINE_EVACUATION_OBSTS(NM,1,0)
8038
8039 ! Allocate OBSTRUCTION array
8040
8041 ALLOCATE(M%OBSTRUCTION(0:N.OBST),STAT=IZERO)
8042 CALL ChkMemErr('READ','OBSTRUCTION',IZERO)
8043 OBSTRUCTION=>M%OBSTRUCTION
8044
8045 N = 0
8046 N.OBST.O = N.OBST
8047 EVAC.N = 1
8048
8049 READ_OBST_LOOP: DO NN=1,N.OBST.O
8050
8051 ID = 'null'
8052 MATL.ID = 'null' ! for HT3D only
8053 MULT.ID = 'null'
8054 PROP.ID = 'null'
8055 SURF.ID = 'null'
8056 SURF.IDS = 'null'
8057 SURF.ID6 = 'null'
8058 COLOR = 'null'
8059 MESH.ID = 'null'
8060 RGB = -1
8061 BULK.DENSITY= -1._EB
8062 HT3D = .FALSE.
8063 INTERNAL.HEAT.SOURCE = 0._EB ! for HT3D only
8064 TRANSPARENCY= 1._EB
8065 BNDF.FACE = BNDF.DEFAULT
8066 BNDF.OBST = BNDF.DEFAULT
8067 NOTERRAIN = .FALSE.
8068 THICKEN = THICKEN.OBSTRUCTIONS
8069 OUTLINE = .FALSE.

```

Source Code files for edited portions of FDS

```

8070 OVERLAY = .TRUE.
8071 TEXTURE_ORIGIN = -999..EB
8072 DEVC.ID = 'null'
8073 CTRL.ID = 'null'
8074 PERMIT.HOLE = .TRUE.
8075 ALLOW.VENT = .TRUE.
8076 REMOVABLE = .TRUE.
8077 XB = -9.E30..EB
8078 IF (.NOT.EVACUATION_ONLY(NM)) EVACUATION = .FALSE.
8079 IF ( EVACUATION_ONLY(NM)) EVACUATION = .TRUE.
8080 IF ( EVACUATION_ONLY(NM)) REMOVABLE = .FALSE.
8081
8082 ! Read the OBST line
8083
8084 EVACUATION.OBST = .FALSE.
8085 IF (EVACUATION_ONLY(NM)) CALL DEFINE.EVACUATION.OBSTS(NM,2,EVAC.N)
8086 EVACUATION.OBSTS: IF (.NOT. EVACUATION.OBST) THEN
8087 CALL CHECKREAD('OBST',LU.INPUT,IOS)
8088 IF (IOS==1) EXIT READ.OBST.LOOP
8089 READ(LU.INPUT,OBST,END=35)
8090 END IF EVACUATION.OBSTS
8091
8092 ! Reorder OBST coordinates if necessary
8093
8094 CALL CHECK.XB(XB)
8095
8096 ! If any obstruction is to do 3D heat transfer (HT3D), set a global parameter
8097
8098 IF (HT3D) SOLID.HT3D = .TRUE.
8099
8100 ! No device and controls for evacuation obstructions
8101
8102 IF (EVACUATION_ONLY(NM)) THEN
8103 DEVC.ID = 'null'
8104 CTRL.ID = 'null'
8105 PROP.ID = 'null'
8106 END IF
8107
8108 ! Loop over all possible multiples of the OBST
8109
8110 MR => MULTIPLIER(0)
8111 DO NNN=1,NMULT
8112 IF (MULT.ID==MULTIPLIER(NNN)%ID) MR => MULTIPLIER(NNN)
8113 ENDDO
8114
8115 K.MULT.LOOP: DO KK=MR%K.LOWER,MR%K.UPPER
8116 J.MULT.LOOP: DO JJ=MR%J.LOWER,MR%J.UPPER
8117 I.MULT.LOOP: DO II=MR%I.LOWER,MR%I.UPPER
8118
8119 IF (.NOT.MR%SEQUENTIAL) THEN
8120 XB1 = XB(1) + MR%DX0 + II*MR%DXB(1)
8121 XB2 = XB(2) + MR%DX0 + II*MR%DXB(2)
8122 XB3 = XB(3) + MR%DY0 + JJ*MR%DXB(3)
8123 XB4 = XB(4) + MR%DY0 + JJ*MR%DXB(4)
8124 XB5 = XB(5) + MR%DZ0 + KK*MR%DXB(5)
8125 XB6 = XB(6) + MR%DZ0 + KK*MR%DXB(6)
8126 ELSE
8127 XB1 = XB(1) + MR%DX0 + II*MR%DXB(1)
8128 XB2 = XB(2) + MR%DX0 + II*MR%DXB(2)
8129 XB3 = XB(3) + MR%DY0 + II*MR%DXB(3)
8130 XB4 = XB(4) + MR%DY0 + II*MR%DXB(4)
8131 XB5 = XB(5) + MR%DZ0 + II*MR%DXB(5)
8132 XB6 = XB(6) + MR%DZ0 + II*MR%DXB(6)
8133 ENDIF
8134
8135 ! Increase the OBST counter
8136
8137 N = N + 1
8138
8139 ! Evacuation criteria
8140
8141 EVAC.N = EVAC.N + 1
8142 IF (MESH.ID/=MESH.NAME(NM) .AND. MESH.ID/'null') THEN
8143 N = N-1
8144 N.OBST = N.OBST-1
8145 CYCLE I.MULT.LOOP
8146 ENDIF
8147
8148 IF ((.NOT.EVACUATION .AND. EVACUATION_ONLY(NM)) .OR. (EVACUATION .AND. .NOT.EVACUATION_ONLY(NM))) THEN
8149 N = N-1
8150 N.OBST = N.OBST-1
8151 CYCLE I.MULT.LOOP
8152 ENDIF
8153
8154 ! Look for obstructions that are within a half grid cell of the current mesh. If the obstruction is thin and has
8155 ! the THICKEN attribute, look for it within an entire grid cell.
8156

```

Source Code files for edited portions of FDS

```

8157 IF ( (XB2>=XS-0.5.EB*DX(0) .AND. XB2<XS) .OR. (THICKEN .AND. 0.5.EB*(XB1+XB2)>=XS-DX(0) .AND. XB2<XS) ) THEN
8158 XB1 = XS
8159 XB2 = XS
8160 THICKEN = .FALSE.
8161 ENDIF
8162 IF ( (XB1<XF+0.5.EB*DX(1BP1) .AND. XB1>XF) .OR. (THICKEN .AND. 0.5.EB*(XB1+XB2)< XF+DX(1BP1) .AND. XB1>XF) ) THEN
8163 XB1 = XF
8164 XB2 = XF
8165 THICKEN = .FALSE.
8166 ENDIF
8167 IF ( (XB4>=YS-0.5.EB*DY(0) .AND. XB4<YS) .OR. (THICKEN .AND. 0.5.EB*(XB3+XB4)>=YS-DY(0) .AND. XB4<YS) ) THEN
8168 XB3 = YS
8169 XB4 = YS
8170 THICKEN = .FALSE.
8171 ENDIF
8172 IF ( (XB3<YF+0.5.EB*DY(JBP1) .AND. XB3>YF) .OR. (THICKEN .AND. 0.5.EB*(XB3+XB4)< YF+DY(JBP1) .AND. XB3>YF) ) THEN
8173 XB3 = YF
8174 XB4 = YF
8175 THICKEN = .FALSE.
8176 ENDIF
8177 IF ( ((XB6>=ZS-0.5.EB*DZ(0) .AND. XB6<ZS) .OR. (THICKEN .AND. 0.5.EB*(XB5+XB6)>=ZS-DZ(0) .AND. XB6<ZS) ) .AND
      &
8178 .NOT.EVACUATION_ONLY(NM) ) THEN
8179 XB5 = ZS
8180 XB6 = ZS
8181 THICKEN = .FALSE.
8182 ENDIF
8183 IF ( ((XB5<ZF+0.5.EB*DZ(KBP1) .AND. XB5>ZF) .OR. (THICKEN .AND. 0.5.EB*(XB5+XB6)< ZF+DZ(KBP1) .AND. XB5>ZF) ) .AND
      &
8184 .NOT.EVACUATION_ONLY(NM) ) THEN
8185 XB5 = ZF
8186 XB6 = ZF
8187 THICKEN = .FALSE.
8188 ENDIF
8189
8190 ! Save the original, undivided obstruction face areas.
8191
8192 UNDIVIDED_INPUT_AREA(1) = (XB4-XB3)*(XB6-XB5)
8193 UNDIVIDED_INPUT_AREA(2) = (XB2-XB1)*(XB6-XB5)
8194 UNDIVIDED_INPUT_AREA(3) = (XB2-XB1)*(XB4-XB3)
8195
8196 ! Throw out obstructions that are not within computational domain
8197
8198 XB1 = MAX(XB1,XS)
8199 XB2 = MIN(XB2,XF)
8200 XB3 = MAX(XB3,YS)
8201 XB4 = MIN(XB4,YF)
8202 XB5 = MAX(XB5,ZS)
8203 XB6 = MIN(XB6,ZF)
8204 IF (XB1>XF .OR. XB2<XS .OR. XB3>YF .OR. XB4<YS .OR. XB5>ZF .OR. XB6<ZS) THEN
8205 N = N-1
8206 N.OBST = N.OBST-1
8207 CYCLE LMULT_LOOP
8208 ENDIF
8209
8210 ! Begin processing of OBSTRUCTION
8211
8212 OB=>OBSTRUCTION(N)
8213
8214 OB%UNDIVIDED_INPUT_AREA(1) = UNDIVIDED_INPUT_AREA(1)
8215 OB%UNDIVIDED_INPUT_AREA(2) = UNDIVIDED_INPUT_AREA(2)
8216 OB%UNDIVIDED_INPUT_AREA(3) = UNDIVIDED_INPUT_AREA(3)
8217
8218 OB%X1 = XB1
8219 OB%X2 = XB2
8220 OB%Y1 = XB3
8221 OB%Y2 = XB4
8222 OB%Z1 = XB5
8223 OB%Z2 = XB6
8224
8225 OB%NOTERRAIN = NOTERRAIN
8226
8227 ! Thicken evacuation mesh obstructions in the z direction
8228
8229 IF (EVACUATION_ONLY(NM) .AND. EVACUATION) THEN
8230 OB%Z1 = ZS
8231 OB%Z2 = ZF
8232 XB5 = ZS
8233 XB6 = ZF
8234 ENDIF
8235
8236 ! Determine the indices of the obstruction according to cell edges, not centers.
8237
8238 OB%I1 = NINT( GINV(XB1-XS,1,NM)*RDXI )
8239 OB%I2 = NINT( GINV(XB2-XS,1,NM)*RDXI )
8240 OB%J1 = NINT( GINV(XB3-YS,2,NM)*RDETA )
8241 OB%J2 = NINT( GINV(XB4-YS,2,NM)*RDETA )
8242 OB%K1 = NINT( GINV(XB5-ZS,3,NM)*RDZETA )

```

Source Code files for edited portions of FDS

```

8243 OB%k2 = NINT( GINV(XB6-ZS,3 ,NM)*RDZETA )
8244
8245 ! If desired , thicken small obstructions
8246
8247 IF (THICKEN) THEN
8248 IF (OB%i1==OB%i2) THEN
8249 OB%i1 = INT(GINV(.5 _EB*(XB1+XB2)-XS,1 ,NM)*RDXI)
8250 OB%i2 = MIN(OB%i1+1,IBAR)
8251 ENDIF
8252 IF (OB%j1==OB%j2) THEN
8253 OB%j1 = INT(GINV(.5 _EB*(XB3+XB4)-YS,2 ,NM)*RDETA)
8254 OB%j2 = MIN(OB%j1+1,JBAR)
8255 ENDIF
8256 IF (OB%k1==OB%k2) THEN
8257 OB%k1 = INT(GINV(.5 _EB*(XB5+XB6)-ZS,3 ,NM)*RDZETA)
8258 OB%k2 = MIN(OB%k1+1,KBAR)
8259 ENDIF
8260 ELSE
8261 !Don't allow thickening if an OBST straddles the midpoint and is small compared to grid cell
8262 IF (GINV(XB2-XS,1 ,NM)-GINV(XB1-XS,1 ,NM)<0.25_EB/RDXI .AND. OB%i1 /= OB%i2) THEN
8263 IF (GINV(XB1-XS,1 ,NM)-REAL(OB%i1 ,EB) < REAL(OB%i2 ,EB) - GINV(XB2-XS,1 ,NM)) THEN
8264 OB%i2=OB%i1
8265 ELSE
8266 OB%i1=OB%i2
8267 ENDIF
8268 ENDIF
8269 IF (GINV(XB4-YS,2 ,NM)-GINV(XB3-YS,2 ,NM)<0.25_EB/RDETA .AND. OB%j1 /= OB%j2) THEN
8270 IF (GINV(XB3-XS,2 ,NM)-REAL(OB%j1 ,EB) < REAL(OB%j2 ,EB) - GINV(XB4-YS,2 ,NM)) THEN
8271 OB%j2=OB%j1
8272 ELSE
8273 OB%j1=OB%j2
8274 ENDIF
8275 ENDIF
8276 IF (GINV(XB6-ZS,3 ,NM)-GINV(XB5-ZS,3 ,NM)<0.25_EB/RDZETA .AND. OB%k1 /= OB%k2) THEN
8277 IF (GINV(XB5-ZS,3 ,NM)-REAL(OB%i1 ,EB) < REAL(OB%i2 ,EB) - GINV(XB6-ZS,3 ,NM)) THEN
8278 OB%k2=OB%k1
8279 ELSE
8280 OB%k1=OB%k2
8281 ENDIF
8282 ENDIF
8283 ENDIF
8284
8285 ! Throw out obstructions that are too small
8286
8287 IF ((OB%i1==OB%i2 .AND. OB%j1==OB%j2) .OR. (OB%i1==OB%i2 .AND. OB%k1==OB%k2) .OR. (OB%j1==OB%j2 .AND. OB%k1==OB%k2))
      THEN
8288 N = N-1
8289 N_OBST= N_OBST-1
8290 CYCLE 1.MULT_LOOP
8291 ENDIF
8292
8293 ! Check to see if obstruction is completely embedded in another
8294
8295 EMBEDDED = .FALSE.
8296 EMBED_LOOP: DO NNN=1,N-1
8297 OB2=>OBSTRUCTION(NNN)
8298 IF (OB%i1>OB2%i1 .AND. OB%i2<OB2%i2 .AND. &
8299 OB%j1>OB2%j1 .AND. OB%j2<OB2%j2 .AND. &
8300 OB%k1>OB2%k1 .AND. OB%k2<OB2%k2) THEN
8301 EMBEDDED = .TRUE.
8302 EXIT EMBED_LOOP
8303 ENDIF
8304 ENDDO EMBED_LOOP
8305
8306 IF (EMBEDDED .AND. DEVC.ID=='null' .AND. REMOVABLE .AND. CTRL.ID=='null' ) THEN
8307 N = N-1
8308 N_OBST= N_OBST-1
8309 CYCLE 1.MULT_LOOP
8310 ENDIF
8311
8312 ! Check if the SURF IDs exist
8313
8314 IF (EVACUATION_ONLY(NM)) SURF.ID=EVAC.SURF.DEFAULT
8315
8316 IF (SURF.ID/='null') CALL CHECK_SURF_NAME(SURF.ID,EX)
8317 IF (.NOT.EX) THEN
8318 WRITE(MESSAGE,'(A,A,A)') 'ERROR: SURF_ID ',TRIM(SURF.ID),' does not exist'
8319 CALL SHUTDOWN(MESSAGE) ; RETURN
8320 ENDIF
8321
8322 DO NNN=1,3
8323 IF (EVACUATION_ONLY(NM)) SURF.IDS(NNN)=EVAC.SURF.DEFAULT
8324 IF (SURF.IDS(NNN)/='null') CALL CHECK_SURF_NAME(SURF.IDS(NNN),EX)
8325 IF (.NOT.EX) THEN
8326 WRITE(MESSAGE,'(A,A,A)') 'ERROR: SURF_ID ',TRIM(SURF.IDS(NNN)),' does not exist'
8327 CALL SHUTDOWN(MESSAGE) ; RETURN
8328 ENDIF
8329 ENDDO

```

Source Code files for edited portions of FDS

```

8330
8331 DO NNN=1,6
8332 IF (EVACUATION_ONLY(NM)) SURF_ID6(NNN)=EVAC_SURF_DEFAULT
8333 IF (SURF_ID6(NNN)/='null') CALL CHECK_SURF_NAME(SURF_ID6(NNN),EX)
8334 IF (.NOT.EX) THEN
8335 WRITE(MESSAGE,'(A,A,A)') 'ERROR: SURF_ID ',TRIM(SURF_ID6(NNN)), ' does not exist'
8336 CALL SHUTDOWN(MESSAGE) ; RETURN
8337 ENDF
8338 ENDDO
8339
8340 ! Save boundary condition info for obstacles
8341
8342 OB%SURF_INDEX(:) = DEFAULT_SURF_INDEX
8343
8344 NNN = 0
8345 DO NNN=0,N_SURF
8346 IF (SURF_ID ==SURFACE(NNN)%ID) OB%SURF_INDEX(:) = NNN
8347 IF (SURF_IDS(1)==SURFACE(NNN)%ID) OB%SURF_INDEX(3) = NNN
8348 IF (SURF_IDS(2)==SURFACE(NNN)%ID) OB%SURF_INDEX(-2:2) = NNN
8349 IF (SURF_IDS(3)==SURFACE(NNN)%ID) OB%SURF_INDEX(-3) = NNN
8350 IF (SURF_ID6(1)==SURFACE(NNN)%ID) OB%SURF_INDEX(-1) = NNN
8351 IF (SURF_ID6(2)==SURFACE(NNN)%ID) OB%SURF_INDEX( 1) = NNN
8352 IF (SURF_ID6(3)==SURFACE(NNN)%ID) OB%SURF_INDEX(-2) = NNN
8353 IF (SURF_ID6(4)==SURFACE(NNN)%ID) OB%SURF_INDEX( 2) = NNN
8354 IF (SURF_ID6(5)==SURFACE(NNN)%ID) OB%SURF_INDEX(-3) = NNN
8355 IF (SURF_ID6(6)==SURFACE(NNN)%ID) OB%SURF_INDEX( 3) = NNN
8356 IF (TRIM(SURFACE(NNN)%ID)==TRIM(EVAC_SURF_DEFAULT)) NNN = NNN
8357 ENDDO
8358
8359 ! Fire + evacuation calculation: draw obsts as outlines by default
8360
8361 IF (.NOT.OUTLINE .AND. EVACUATION_ONLY(NM) .AND. .NOT.ALL(EVACUATION_ONLY)) THEN
8362 IF (SURFACE(NNN)%TRANSPARENCY < 0.99999.EB .AND. .NOT.OUTLINE) THEN
8363 OUTLINE = .FALSE.
8364 ELSE
8365 OUTLINE = .TRUE.
8366 ENDF
8367 ENDF
8368
8369 ! Determine if the OBST is CONSUMABLE
8370
8371 FACELOOP: DO NNN=-3,3
8372 IF (NNN==0) CYCLE FACELOOP
8373 IF (SURFACE(OB%SURF_INDEX(NNN))%BURN_AWAY) OB%CONSUMABLE = .TRUE.
8374 ENDDO FACELOOP
8375
8376 ! Calculate the increase or decrease in the obstruction volume over the user-specified
8377
8378 VOL_SPECIFIED = (OB%X2-OB%X1)*(OB%Y2-OB%Y1)*(OB%Z2-OB%Z1)
8379 VOL_ADJUSTED = (X(OB%I2)-X(OB%I1))*(Y(OB%J2)-Y(OB%J1))*(Z(OB%K2)-Z(OB%K1))
8380 IF (VOL_SPECIFIED > 0.EB .AND. .NOT.EVACUATION_ONLY(NM)) THEN
8381 OB%VOLUME_ADJUST = VOL_ADJUSTED/VOL_SPECIFIED
8382 ELSE
8383 OB%VOLUME_ADJUST = 0.EB
8384 ENDF
8385
8386 ! Creation and removal logic
8387
8388 OB%DEVC_ID = DEVC_ID
8389 OB%CTRL_ID = CTRL_ID
8390 OB%HIDDEN = .FALSE.
8391
8392 ! Property ID
8393
8394 OB%PROP_ID = PROP_ID
8395
8396 CALL SEARCH_CONTROLLER('OBST',CTRL_ID,DEVC_ID,OB%DEVC_INDEX,OB%CTRL_INDEX,N)
8397 IF (DEVC_ID /= 'null' .OR. CTRL_ID /= 'null') OB%REMOVABLE = .TRUE.
8398
8399 IF (OB%CONSUMABLE .AND. .NOT.EVACUATION_ONLY(NM)) OB%REMOVABLE = .TRUE.
8400
8401 ! Choose obstruction color index
8402
8403 SELECT CASE (COLOR)
8404 CASE ('INVISIBLE')
8405 OB%COLOR_INDICATOR = -3
8406 RGB(1) = 255
8407 RGB(2) = 204
8408 RGB(3) = 102
8409 TRANSPARENCY = 0.EB
8410 CASE ('null')
8411 IF (ANY(RGB<0)) THEN
8412 OB%COLOR_INDICATOR = -1
8413 ELSE
8414 OB%COLOR_INDICATOR = -3
8415 ENDF
8416 CASE DEFAULT
8417 CALL COLOR2RGB(RGB,COLOR)

```

Source Code files for edited portions of FDS

```

8418 OB%COLOR_INDICATOR = -3
8419 END SELECT
8420 OB%RGB = RGB
8421 OB%TRANSPARENCY = TRANSPARENCY
8422
8423 ! Miscellaneous assignments
8424
8425 OB%TEXTURE(:) = TEXTURE_ORIGIN(:) ! Origin of texture map
8426 OB%ORDINAL = NN ! Order of OBST in original input file
8427 OB%PERMIT_HOLE = PERMIT_HOLE
8428 OB%ALLOW_VENT = ALLOW_VENT
8429 OB%OVERLAY = OVERLAY
8430
8431 ! Only allow the use of BULK_DENSITY if the obstruction has a non-zero volume
8432
8433 IF (EVACUATION_ONLY(NM)) BULK_DENSITY = -1.EB
8434 OB%BULK_DENSITY = BULK_DENSITY
8435 IF (ABS(OB%VOLUME_ADJUST)<TWO_EPSILON_EB .AND. OB%BULK_DENSITY>0.EB) OB%BULK_DENSITY = -1.EB
8436
8437 ! Error traps and warnings for HT3D
8438
8439 IF (.NOT.HT3D .AND. ABS(INTERNAL_HEAT_SOURCE)>TWO_EPSILON_EB) THEN
8440 WRITE(MESSAGE,'(A,I0,A)') 'ERROR: Problem with OBST number ',NN,', INTERNAL_HEAT_SOURCE requires HT3D=T.'
8441 CALL SHUTDOWN(MESSAGE) ; RETURN
8442 ENDIF
8443
8444 ! No HT3D for EVAC or zero volume OBST
8445
8446 IF (EVACUATION_ONLY(NM)) HT3D=.FALSE.
8447 OB%HT3D = HT3D
8448 IF (OB%HT3D .AND. ABS(OB%VOLUME_ADJUST)<TWO_EPSILON_EB) THEN
8449 WRITE(LU_ERR,'(A,I0,A)') 'WARNING: OBST number ',NN,' has zero volume, consider THICKEN=T, HT3D set to F.'
8450 OB%HT3D=.FALSE. ! later add capability for 2D lateral ht on thin obst
8451 ENDIF
8452
8453 IF (OB%HT3D .AND. TRIM(MATL_ID)=='null') THEN
8454 WRITE(MESSAGE,'(A,I0,A)') 'ERROR: Problem with OBST number ',NN,', HT3D requires MATL_ID.'
8455 CALL SHUTDOWN(MESSAGE) ; RETURN
8456 ENDIF
8457
8458 ! Set MATL_INDEX for HT3D
8459
8460 IF (OB%HT3D) THEN
8461 OB%MATL_ID = MATL_ID
8462 DO NNN=1,N_MATL
8463 ML=>MATERIAL(NNN)
8464 IF (TRIM(OB%MATL_ID)==TRIM(ML%ID)) THEN
8465 OB%MATL_INDEX=NNN
8466 OB%BULK_DENSITY=ML%RHOS
8467 EXIT
8468 ENDIF
8469 ENDDO
8470 IF (OB%MATL_INDEX<0) THEN
8471 WRITE(MESSAGE,'(A,I0,A)') 'ERROR: Problem with OBST number ',NN,', MATL_ID not found.'
8472 CALL SHUTDOWN(MESSAGE) ; RETURN
8473 ENDIF
8474 OB%INTERNAL_HEAT_SOURCE = INTERNAL_HEAT_SOURCE * 1000.EB ! W/m^3
8475 ENDIF
8476
8477 ! Make obstruction invisible if it's within a finer mesh
8478
8479 DO NCM=1,NM-1
8480 IF (EVACUATION_ONLY(NCM)) CYCLE
8481 IF (EVACUATION_ONLY(NM)) CYCLE
8482 IF (XB1>MESHES(NCM)%XS .AND. XB2<MESHES(NCM)%XF .AND. &
8483 XB3>MESHES(NCM)%YS .AND. XB4<MESHES(NCM)%YF .AND. &
8484 XB5>MESHES(NCM)%ZS .AND. XB6<MESHES(NCM)%ZF) OB%COLOR_INDICATOR=-2
8485 ENDDO
8486
8487 ! Prevent drawing of boundary info if desired
8488
8489 IF (BNDF_DEFAULT) THEN
8490 OB%SHOW_BNDF(:) = BNDF_FACE(:)
8491 IF (.NOT.BNDF_OBST) OB%SHOW_BNDF(:) = .FALSE.
8492 ELSE
8493 OB%SHOW_BNDF(:) = BNDF_FACE(:)
8494 IF (BNDF_OBST) OB%SHOW_BNDF(:) = .TRUE.
8495 ENDIF
8496 IF (EVACUATION_ONLY(NM)) OB%SHOW_BNDF(:) = .FALSE.
8497
8498 ! In Smokeview, draw the outline of the obstruction
8499
8500 IF (OUTLINE) OB%TYPE_INDICATOR = 2
8501
8502 ENDDO I.MULT_LOOP
8503 ENDDO J.MULT_LOOP
8504 ENDDO K.MULT_LOOP
8505

```

Source Code files for edited portions of FDS

```

8506 ENDDO READ_OBST_LOOP
8507 35 REWIND(LU.INPUT) ; INPUT_FILE_LINE_NUMBER = 0
8508
8509 ENDDO MESH_LOOP
8510
8511 ! Read HOLES and cut out blocks
8512
8513 CALL READ_HOLE
8514
8515 ! Look for OBSTRUCTIONS that are meant to BURN_AWAY and break them up into single cell blocks
8516
8517 MESH_LOOP_2: DO NM=1,NMESHES
8518
8519 IF (PROCESS(NM)/=MYID .AND. MYID/=EVAC.PROCESS) CYCLE MESH_LOOP_2
8520
8521 M=>MESHES(NM)
8522 CALL POINT_TO_MESH(NM)
8523
8524 N_OBST_O = N_OBST
8525 DO N=1,N_OBST_O
8526 OB => OBSTRUCTION(N)
8527 IF (OB%CONSUMABLE .AND. .NOT.EVACUATION_ONLY(NM)) THEN
8528
8529 N_NEW_OBST = MAX(1,OB%I2-OB%I1)*MAX(1,OB%J2-OB%J1)*MAX(1,OB%K2-OB%K1)
8530 IF (N_NEW_OBST > 1) THEN
8531
8532 ! Create a temporary array of obstructions with the same properties as the one being replaced, except coordinates
8533
8534 ALLOCATE(TEMP_OBSTRUCTION(N_NEW_OBST))
8535 TEMP_OBSTRUCTION = OBSTRUCTION(N)
8536 NN = 0
8537 DO K=OB%K1,MAX(OB%K1,OB%K2-1)
8538 DO J=OB%J1,MAX(OB%J1,OB%J2-1)
8539 DO I=OB%I1,MAX(OB%I1,OB%I2-1)
8540 NN = NN + 1
8541 OBT=>TEMP_OBSTRUCTION(NN)
8542 OBT%I1 = I
8543 OBT%I2 = MIN(I+1,OB%I2)
8544 OBT%J1 = J
8545 OBT%J2 = MIN(J+1,OB%J2)
8546 OBT%K1 = K
8547 OBT%K2 = MIN(K+1,OB%K2)
8548 OBT%X1 = M%X(OBT%I1)
8549 OBT%X2 = M%X(OBT%I2)
8550 OBT%Y1 = M%Y(OBT%J1)
8551 OBT%Y2 = M%Y(OBT%J2)
8552 OBT%Z1 = M%Z(OBT%K1)
8553 OBT%Z2 = M%Z(OBT%K2)
8554 ENDDO
8555 ENDDO
8556 ENDDO
8557
8558 CALL RE_ALLOCATE_OBST(NM,N_OBST,N_NEW_OBST-1)
8559 OBSTRUCTION=>M%OBSTRUCTION
8560 OBSTRUCTION(N) = TEMP_OBSTRUCTION(1)
8561 OBSTRUCTION(N_OBST+1:N_OBST+N_NEW_OBST-1) = TEMP_OBSTRUCTION(2:N_NEW_OBST)
8562 N_OBST = N_OBST + N_NEW_OBST-1
8563 DEALLOCATE(TEMP_OBSTRUCTION)
8564
8565 ENDIF
8566 ENDIF
8567 ENDDO
8568
8569 ENDDO MESH_LOOP_2
8570
8571 ! Allocate the number of cells for each mesh that are SOLID or border a boundary
8572
8573 ALLOCATE(CELL_COUNT(NMESHES)) ; CELL_COUNT = 0
8574
8575 ! Go through all meshes, recording which cells are solid
8576
8577 MESH_LOOP_3: DO NM=1,NMESHES
8578
8579 IF (PROCESS(NM)/=MYID .AND. MYID/=EVAC.PROCESS) CYCLE MESH_LOOP_3
8580
8581 M=>MESHES(NM)
8582 CALL POINT_TO_MESH(NM)
8583
8584 ! Compute areas of obstruction faces, both actual (AB0) and FDS approximated (AB)
8585
8586 DO N=1,N_OBST
8587 OB=>OBSTRUCTION(N)
8588 OB%INPUT_AREA(1) = (OB%Y2-OB%Y1)*(OB%Z2-OB%Z1)
8589 OB%INPUT_AREA(2) = (OB%X2-OB%X1)*(OB%Z2-OB%Z1)
8590 OB%INPUT_AREA(3) = (OB%X2-OB%X1)*(OB%Y2-OB%Y1)
8591 OB%FDS_AREA(1) = (Y(OB%J2)-Y(OB%J1))*(Z(OB%K2)-Z(OB%K1))
8592 OB%FDS_AREA(2) = (X(OB%I2)-X(OB%I1))*(Z(OB%K2)-Z(OB%K1))
8593 OB%FDS_AREA(3) = (X(OB%I2)-X(OB%I1))*(Y(OB%J2)-Y(OB%J1))

```

```

8594 OB%DIMENSIONS(1) = OB%i2 - OB%i1
8595 OB%DIMENSIONS(2) = OB%j2 - OB%j1
8596 OB%DIMENSIONS(3) = OB%k2 - OB%k1
8597 ENDDO
8598
8599 ! Create main blockage index array (ICA)
8600
8601 ALLOCATE(M%CELL_INDEX(0:IBP1,0:JBP1,0:KBP1),STAT=IZERO)
8602 CALL ChkMemErr('READ','CELL_INDEX',IZERO) ; CELL_INDEX=>M%CELL_INDEX ; CELL_INDEX = 0
8603
8604 DO K=0,KBP1
8605 IF (EVACUATION_ONLY(NM) .AND. .NOT.(K==1)) CYCLE
8606 DO J=0,JBP1
8607 DO I=0,I
8608 IF (CELL_INDEX(I,J,K)==0) THEN
8609 CELL_COUNT(NM) = CELL_COUNT(NM) + 1
8610 CELL_INDEX(I,J,K) = CELL_COUNT(NM)
8611 ENDF
8612 ENDDO
8613 DO I=IBAR,IBP1
8614 IF (CELL_INDEX(I,J,K)==0) THEN
8615 CELL_COUNT(NM) = CELL_COUNT(NM) + 1
8616 CELL_INDEX(I,J,K) = CELL_COUNT(NM)
8617 ENDF
8618 ENDDO
8619 ENDDO
8620 ENDDO
8621
8622 DO K=0,KBP1
8623 IF (EVACUATION_ONLY(NM) .AND. .NOT.(K==1)) CYCLE
8624 DO I=0,IBP1
8625 DO J=0,1
8626 IF (CELL_INDEX(I,J,K)==0) THEN
8627 CELL_COUNT(NM) = CELL_COUNT(NM) + 1
8628 CELL_INDEX(I,J,K) = CELL_COUNT(NM)
8629 ENDF
8630 ENDDO
8631 DO J=JBAR,JBP1
8632 IF (CELL_INDEX(I,J,K)==0) THEN
8633 CELL_COUNT(NM) = CELL_COUNT(NM) + 1
8634 CELL_INDEX(I,J,K) = CELL_COUNT(NM)
8635 ENDF
8636 ENDDO
8637 ENDDO
8638 ENDDO
8639
8640 DO J=0,JBP1
8641 DO I=0,IBP1
8642 DO K=0,1
8643 IF (EVACUATION_ONLY(NM) .AND. .NOT.(K==1)) CYCLE
8644 IF (CELL_INDEX(I,J,K)==0) THEN
8645 CELL_COUNT(NM) = CELL_COUNT(NM) + 1
8646 CELL_INDEX(I,J,K) = CELL_COUNT(NM)
8647 ENDF
8648 ENDDO
8649 DO K=KBAR,KBP1
8650 IF (EVACUATION_ONLY(NM) .AND. .NOT.(K==1)) CYCLE
8651 IF (CELL_INDEX(I,J,K)==0) THEN
8652 CELL_COUNT(NM) = CELL_COUNT(NM) + 1
8653 CELL_INDEX(I,J,K) = CELL_COUNT(NM)
8654 ENDF
8655 ENDDO
8656 ENDDO
8657 ENDDO
8658
8659 DO N=1,N_OBST
8660 OB->OBSTRUCTION(N)
8661 DO K=OB%k1,OB%k2+1
8662 IF (EVACUATION_ONLY(NM) .AND. .NOT.(K==1)) CYCLE
8663 DO J=OB%j1,OB%j2+1
8664 DO I=OB%i1,OB%i2+1
8665 IF (CELL_INDEX(I,J,K)==0) THEN
8666 CELL_COUNT(NM) = CELL_COUNT(NM) + 1
8667 CELL_INDEX(I,J,K) = CELL_COUNT(NM)
8668 ENDF
8669 ENDDO
8670 ENDDO
8671 ENDDO
8672 ENDDO
8673
8674 ! Store in SOLID which cells are solid and which are not
8675
8676 ALLOCATE(M%SOLID(0:CELL_COUNT(NM)),STAT=IZERO)
8677 CALL ChkMemErr('READ','SOLID',IZERO)
8678 M%SOLID = .FALSE.
8679 ALLOCATE(M%EXTERIOR(0:CELL_COUNT(NM)),STAT=IZERO)
8680 CALL ChkMemErr('READ','EXTERIOR',IZERO)
8681 M%EXTERIOR = .FALSE.

```



```

8682 SOLID=>M%$SOLID
8683 ALLOCATE(M%OBST_INDEX.C(0:CELL_COUNT(NM)),STAT=IZERO)
8684 CALL ChkMemErr('READ','OBST_INDEX.C',IZERO)
8685 M%OBST_INDEX.C = 0
8686 OBST_INDEX.C=>M%OBST_INDEX.C
8687
8688 ! Make all exterior cells solid
8689
8690 CALL BLOCK_CELL(NM, 0, 0, 0, JBP1, 0, KBP1, 1, 0)
8691 CALL BLOCK_CELL(NM, IBP1, IBP1, 0, JBP1, 0, KBP1, 1, 0)
8692 CALL BLOCK_CELL(NM, 0, IBP1, 0, 0, 0, KBP1, 1, 0)
8693 CALL BLOCK_CELL(NM, 0, IBP1, JBP1, JBP1, 0, KBP1, 1, 0)
8694 CALL BLOCK_CELL(NM, 0, IBP1, 0, JBP1, 0, 0, 1, 0)
8695 CALL BLOCK_CELL(NM, 0, IBP1, 0, JBP1, KBP1, KBP1, 1, 0)
8696
8697 ! Block off cells filled by obstructions
8698
8699 DO N=1, N_OBST
8700 OB=>OBSTRUCTION(N)
8701 CALL BLOCK_CELL(NM, OB%I1+1, OB%I2, OB%J1+1, OB%J2, OB%K1+1, OB%K2, 1, N)
8702 ENDDO
8703
8704 ! Create arrays to hold cell indices
8705
8706 ALLOCATE(M%I_CELL(CELL_COUNT(NM)),STAT=IZERO)
8707 CALL ChkMemErr('READ','I_CELL',IZERO)
8708 M%I_CELL = -1
8709 ALLOCATE(M%J_CELL(CELL_COUNT(NM)),STAT=IZERO)
8710 CALL ChkMemErr('READ','J_CELL',IZERO)
8711 M%J_CELL = -1
8712 ALLOCATE(M%K_CELL(CELL_COUNT(NM)),STAT=IZERO)
8713 CALL ChkMemErr('READ','K_CELL',IZERO)
8714 M%K_CELL = -1
8715 I_CELL=>M%I_CELL
8716 J_CELL=>M%J_CELL
8717 K_CELL=>M%K_CELL
8718
8719 DO K=0, KBP1
8720 DO J=0, JBP1
8721 DO I=0, IBP1
8722 IC = CELL_INDEX(I, J, K)
8723 IF (IC>0) THEN
8724 I_CELL(IC) = I
8725 J_CELL(IC) = J
8726 K_CELL(IC) = K
8727 ENDF
8728 ENDDO
8729 ENDDO
8730 ENDDO
8731
8732 ENDDO MESH_LOOP_3
8733
8734 CONTAINS
8735
8736 SUBROUTINE DEFINE_EVACUATION_OBSTS(NM, IMODE, EVAC_N)
8737 !
8738 ! Define the evacuation OBSTs for the doors/exits, if needed. A VENT should always
8739 ! be defined on an OBST that is at least one grid cell thick or the VENT should be
8740 ! on the outer boundary of the evacuation mesh, which is by default solid.
8741 ! The core of the STRS meshes are also defined.
8742 !
8743 USE EVAC, ONLY: N_DOORS, N_EXITS, N_CO_EXITS, EVAC_EMESH_EXITS_TYPE, EMESH_EXITS, &
8744 N_STRS, EMESH_STAIRS, EVAC_EMESH_STAIRS_TYPE
8745 IMPLICIT NONE
8746 ! Passed variables
8747 INTEGER, INTENT(IN) :: NM, IMODE, EVAC_N
8748 ! Local variables
8749 INTEGER :: N, N_END, I1, I2, J1, J2
8750 REAL(EB) :: TINY
8751
8752 TINY = 0.1_EB*MIN(MESHES(NM)%DXI, MESHES(NM)%DELTA)
8753 N_END = N_EXITS - N_CO_EXITS + N_DOORS
8754 IMODE_1_1F: IF (IMODE==1) THEN
8755 NEND_LOOP_1: DO N = 1, N_END
8756
8757 IF (.NOT.EMESH_EXITS(N)%DEFINE_MESH) CYCLE NEND_LOOP_1
8758 IF (EMESH_EXITS(N)%MESH==NM .OR. EMESH_EXITS(N)%MAINMESH==NM) THEN
8759 EMESH_EXITS(N)%L_OBST = 0
8760
8761 ! Move EMESH_EXITS(N)%XB to mesh cell boundaries
8762 EMESH_EXITS(N)%XB(1) = MAX(EMESH_EXITS(N)%XB(1), MESHES(NM)%XS)
8763 EMESH_EXITS(N)%XB(2) = MIN(EMESH_EXITS(N)%XB(2), MESHES(NM)%XF)
8764 EMESH_EXITS(N)%XB(3) = MAX(EMESH_EXITS(N)%XB(3), MESHES(NM)%YS)
8765 EMESH_EXITS(N)%XB(4) = MIN(EMESH_EXITS(N)%XB(4), MESHES(NM)%YF)
8766
8767 I1 = NINT(GINV(EMESH_EXITS(N)%XB(1)-MESHES(NM)%XS, 1, NM)*MESHES(NM)%RDXI)
8768 I2 = NINT(GINV(EMESH_EXITS(N)%XB(2)-MESHES(NM)%XS, 1, NM)*MESHES(NM)%RDXI)
8769 J1 = NINT(GINV(EMESH_EXITS(N)%XB(3)-MESHES(NM)%YS, 2, NM)*MESHES(NM)%RDETA)

```

Source Code files for edited portions of FDS

```

8770 J2 = NINT(GINV(EMESH.EXIT(N)%XB(4)-MESHES(NM)%YS,2,NM)*MESHES(NM)%RDETA)
8771
8772 EMESH.EXIT(N)%XB(1) = MESHES(NM)%X(I1)
8773 EMESH.EXIT(N)%XB(2) = MESHES(NM)%X(I2)
8774 EMESH.EXIT(N)%XB(3) = MESHES(NM)%Y(J1)
8775 EMESH.EXIT(N)%XB(4) = MESHES(NM)%Y(J2)
8776
8777 ! Check if the exit/door is at the mesh boundary, then no OBST is needed.
8778 SELECT CASE (EMESH.EXIT(N)%IOR)
8779 CASE (-1)
8780 IF (MESHES(NM)%XS >= EMESH.EXIT(N)%XB(1)-TINY) CYCLE NEND_LOOP_1
8781 CASE (+1)
8782 IF (MESHES(NM)%XF <= EMESH.EXIT(N)%XB(2)+TINY) CYCLE NEND_LOOP_1
8783 CASE (-2)
8784 IF (MESHES(NM)%YS >= EMESH.EXIT(N)%XB(3)-TINY) CYCLE NEND_LOOP_1
8785 CASE (+2)
8786 IF (MESHES(NM)%YF <= EMESH.EXIT(N)%XB(4)+TINY) CYCLE NEND_LOOP_1
8787 END SELECT
8788 N.OBST = N.OBST + 1
8789 EMESH.EXIT(N)%L_OBST = N.OBST
8790 EVACUATION.OBST = .TRUE.
8791 END IF
8792 END DO NEND_LOOP_1
8793
8794 NSTRS_LOOP_1: DO N = 1, N_STRS
8795
8796 IF (.NOT.EMESH.STAIRS(N)%DEFINE_MESH) CYCLE NSTRS_LOOP_1
8797 IF (EMESH.STAIRS(N)%MESH==NM) THEN
8798
8799 ! Move EMESH.STAIRS(N)%XB.CORE to mesh cell boundaries
8800 EMESH.STAIRS(N)%XB.CORE(1) = MAX(EMESH.STAIRS(N)%XB.CORE(1),MESHES(NM)%XS)
8801 EMESH.STAIRS(N)%XB.CORE(2) = MIN(EMESH.STAIRS(N)%XB.CORE(2),MESHES(NM)%XF)
8802 EMESH.STAIRS(N)%XB.CORE(3) = MAX(EMESH.STAIRS(N)%XB.CORE(3),MESHES(NM)%YS)
8803 EMESH.STAIRS(N)%XB.CORE(4) = MIN(EMESH.STAIRS(N)%XB.CORE(4),MESHES(NM)%YF)
8804
8805 I1 = NINT(GINV(EMESH.STAIRS(N)%XB.CORE(1)-MESHES(NM)%XS,1,NM)*MESHES(NM)%RDXI)
8806 I2 = NINT(GINV(EMESH.STAIRS(N)%XB.CORE(2)-MESHES(NM)%XS,1,NM)*MESHES(NM)%RDXI)
8807 J1 = NINT(GINV(EMESH.STAIRS(N)%XB.CORE(3)-MESHES(NM)%YS,2,NM)*MESHES(NM)%RDETA)
8808 J2 = NINT(GINV(EMESH.STAIRS(N)%XB.CORE(4)-MESHES(NM)%YS,2,NM)*MESHES(NM)%RDETA)
8809
8810 EMESH.STAIRS(N)%XB.CORE(1) = MESHES(NM)%X(I1)
8811 EMESH.STAIRS(N)%XB.CORE(2) = MESHES(NM)%X(I2)
8812 EMESH.STAIRS(N)%XB.CORE(3) = MESHES(NM)%Y(J1)
8813 EMESH.STAIRS(N)%XB.CORE(4) = MESHES(NM)%Y(J2)
8814
8815 N.OBST = N.OBST + 1
8816 EMESH.STAIRS(N)%L_OBST = N.OBST
8817 EVACUATION.OBST = .TRUE.
8818 END IF
8819 END DO NSTRS_LOOP_1
8820
8821 END IF IMODE_1.IF
8822
8823 IMODE_2.IF: IF (IMODE==2) THEN
8824 NEND_LOOP_2: DO N = 1, N_END
8825 IF (.NOT.EMESH.EXIT(N)%DEFINE_MESH) CYCLE NEND_LOOP_2
8826 IF (EMESH.EXIT(N)%L_OBST==EVAC_N .AND. (EMESH.EXIT(N)%MESH==NM .OR. EMESH.EXIT(N)%MAINMESH==NM)) THEN
8827 EVACUATION.OBST = .TRUE.
8828 EVACUATION = .TRUE.
8829 REMOVABLE = .FALSE.
8830 THICKEN = .TRUE.
8831 PERMIT_HOLE = .FALSE.
8832 ALLOW_VENT = .TRUE.
8833 MESH.ID = TRIM(MESHNAME(NM))
8834 XB(1) = EMESH.EXIT(N)%XB(1)
8835 XB(2) = EMESH.EXIT(N)%XB(2)
8836 XB(3) = EMESH.EXIT(N)%XB(3)
8837 XB(4) = EMESH.EXIT(N)%XB(4)
8838 XB(5) = EMESH.EXIT(N)%XB(5)
8839 XB(6) = EMESH.EXIT(N)%XB(6)
8840 SELECT CASE (EMESH.EXIT(N)%IOR)
8841 CASE (-1)
8842 XB(1) = MAX(MESHES(NM)%XS, XB(1) - 0.49.EB*MESHES(NM)%DXI)
8843 CASE (+1)
8844 XB(2) = MIN(MESHES(NM)%XF, XB(2) + 0.49.EB*MESHES(NM)%DXI)
8845 CASE (-2)
8846 XB(3) = MAX(MESHES(NM)%YS, XB(3) - 0.49.EB*MESHES(NM)%DETA)
8847 CASE (+2)
8848 XB(4) = MIN(MESHES(NM)%YF, XB(4) + 0.49.EB*MESHES(NM)%DETA)
8849 END SELECT
8850 RGB(:) = EMESH.EXIT(N)%RGB(:)
8851 ID = TRIM('Eobst_' // TRIM(MESHNAME(NM)))
8852 END IF
8853 END DO NEND_LOOP_2
8854
8855 NSTRS_LOOP_2: DO N = 1, N_STRS
8856 IF (.NOT.EMESH.STAIRS(N)%DEFINE_MESH) CYCLE NSTRS_LOOP_2
8857 IF (EMESH.STAIRS(N)%L_OBST==EVAC_N .AND. EMESH.STAIRS(N)%MESH==NM) THEN

```

```

8858 EVACUATION.OBST = .TRUE.
8859 EVACUATION = .TRUE.
8860 REMOVABLE = .FALSE.
8861 ! THICKEN = .TRUE.
8862 PERMIT.HOLE = .FALSE.
8863 ALLOW.VENT = .FALSE.
8864 MESH.ID = TRIM(MESHNAME(NM))
8865 XB(1) = EMESH.STAIRS(N)%XB.CORE(1)
8866 XB(2) = EMESH.STAIRS(N)%XB.CORE(2)
8867 XB(3) = EMESH.STAIRS(N)%XB.CORE(3)
8868 XB(4) = EMESH.STAIRS(N)%XB.CORE(4)
8869 XB(5) = EMESH.STAIRS(N)%XB(5)
8870 XB(6) = EMESH.STAIRS(N)%XB(6)
8871 RGB(:) = EMESH.STAIRS(N)%RGB(:)
8872 ID = TRIM('Obst.' // TRIM(MESHNAME(NM)))
8873 END IF
8874 END DO NSTRS.LOOP.2
8875
8876 END IF IMODE.2.IF
8877
8878 RETURN
8879 END SUBROUTINE DEFINE.EVACUATION.OBSTS
8880
8881 END SUBROUTINE READ.OBST
8882
8883
8884 SUBROUTINE READ.HOLE
8885
8886 CHARACTER(LABEL.LENGTH) :: DEVC.ID,CTRL.ID,MULT.ID
8887 CHARACTER(60) :: MESH.ID
8888 CHARACTER(25) :: COLOR
8889 LOGICAL :: EVACUATION.HOLE,EVACUATION.BLOCK.WIND
8890 INTEGER :: NM,N.HOLE,NN,NDO,N,I1,I2,J1,J2,K1,K2,RGB(3),N.HOLE.NEW,N.HOLE.O,I1,J1,JK,NNN,DEVC.INDEX.O,CTRL.INDEX.O
8891 REAL(EB) :: X1,X2,Y1,Y2,Z1,Z2,TRANSPARENCY
8892 NAMELIST /HOLE/ BLOCK.WIND,COLOR,CTRL.ID,DEVC.ID,EVACUATION,FYI.ID,MESH.ID,MULT.ID,RGB,TRANSPARENCY,XB
8893 REAL(EB),ALLOCATABLE,DIMENSION(:,:) :: TEMP.XB
8894 LOGICAL,ALLOCATABLE,DIMENSION(:) :: CONTROLLED
8895 TYPE(OBSTRUCTION.TYPE),ALLOCATABLE,DIMENSION(:) :: TEMP.OBST
8896 TYPE(MULTIPLIER.TYPE),POINTER :: MR=>NULL()
8897 LOGICAL,ALLOCATABLE,DIMENSION(:) :: TEMP.HOLE.EVAC
8898
8899 ALLOCATE(TEMP.OBST(0:6))
8900
8901 N.HOLE = 0
8902 N.HOLE.O = 0
8903 REWIND(LU.INPUT) ; INPUT.FILE.LINE.NUMBER = 0
8904
8905 COUNT.LOOP: DO
8906 CALL CHECKREAD('HOLE',LU.INPUT,IOS)
8907 IF (IOS==1) EXIT COUNT.LOOP
8908 MULT.ID = 'null'
8909 READ(LU.INPUT,NML=HOLE,END=1,ERR=2,IOSTAT=IOS)
8910 N.HOLE.O = N.HOLE.O + 1
8911 N.HOLE.NEW = 0
8912 IF (MULT.ID=='null') THEN
8913 N.HOLE.NEW = 1
8914 ELSE
8915 DO N=1,N.MULT
8916 MR => MULTIPLIER(N)
8917 IF (MULT.ID==MR%ID) N.HOLE.NEW = MR%N.COPIES
8918 ENDDO
8919 IF (N.HOLE.NEW==0) THEN
8920 WRITE(MESSAGE,'(A,A,A,10)') 'ERROR: MULT line ',TRIM(MULT.ID),' not found on HOLE line',N.HOLE.O
8921 CALL SHUTDOWN(MESSAGE) ; RETURN
8922 ENDIF
8923 ENDIF
8924 N.HOLE = N.HOLE + N.HOLE.NEW
8925 2 IF (IOS>0) THEN
8926 WRITE(MESSAGE,'(A,10,A,10)') 'ERROR: Problem with HOLE number',N.HOLE.O+1,', line number',INPUT.FILE.LINE.NUMBER
8927 CALL SHUTDOWN(MESSAGE) ; RETURN
8928 ENDIF
8929 ENDDO COUNT.LOOP
8930 1 REWIND(LU.INPUT) ; INPUT.FILE.LINE.NUMBER = 0
8931
8932 CALL DEFINE.EVACUATION.HOLES(1)
8933
8934 ALLOCATE (TEMP.XB(N.HOLE.O,6))
8935 TEMP.XB = 0..EB
8936 ALLOCATE (CONTROLLED(N.HOLE.O))
8937 CONTROLLED = .FALSE.
8938
8939 ! TEMP.HOLE.EVAC(:) indicates if the given HOLE is to be used in the EVACUATION routine
8940
8941 IF (ANY(EVACUATION.ONLY)) THEN
8942 ALLOCATE(TEMP.HOLE.EVAC(1:N.HOLE.O))
8943 READ.HOLE.EVAC.LOOP: DO N=1,N.HOLE.O
8944 EVACUATION.HOLE = .FALSE.
8945 CALL DEFINE.EVACUATION.HOLES(2)

```

```

8946 EVACUATION = .TRUE.
8947 IF (.NOT.EVACUATION.HOLE) THEN
8948 CALL CHECKREAD('HOLE',LU_INPUT,IOS)
8949 IF (IOS==1) EXIT READ.HOLE.EVAC.LOOP
8950 READ(LU_INPUT,HOLE)
8951 END IF
8952 TEMP.HOLE.EVAC(N) = EVACUATION
8953 ENDDO READ.HOLE.EVAC.LOOP
8954 REWIND(LU_INPUT) ; INPUT_FILE.LINE.NUMBER = 0
8955 ENDIF
8956
8957 READ.HOLE.LOOP: DO N=1,N.HOLE.O
8958
8959 ! Set default values for the HOLE namelist parameters
8960
8961 BLOCK.WIND = .FALSE.
8962 DEVC.ID = 'null'
8963 CTRL.ID = 'null'
8964 ID = 'null'
8965 MESH.ID = 'null'
8966 MULT.ID = 'null'
8967 COLOR = 'null'
8968 RGB = -1
8969 TRANSPARENCY = 1..EB
8970 EVACUATION = .FALSE.
8971 XB = -9.E30.EB
8972
8973 ! Read the HOLE line
8974
8975 EVACUATION.HOLE = .FALSE.
8976 IF (ANY(EVACUATION_ONLY)) CALL DEFINE.EVACUATION_HOLES(3)
8977 EVACUATION.HOLES: IF (.NOT. EVACUATION.HOLE) THEN
8978 CALL CHECKREAD('HOLE',LU_INPUT,IOS)
8979 IF (IOS==1) EXIT READ.HOLE.LOOP
8980 READ(LU_INPUT,HOLE)
8981 END IF EVACUATION.HOLES
8982
8983 ! Re-order coordinates , if necessary
8984
8985 CALL CHECK.XB(XB)
8986 TEMP.XB(N,:) = XB
8987 ! Check for overlap if controlled
8988 IF (DEVC.ID/'null' .OR. CTRL.ID/'null') CONTROLLED(N) = .TRUE.
8989 IF (N>1) THEN
8990 DO NN = 1,N-1
8991 IF (TEMP.XB(NN,1) >= XB(2) .OR. TEMP.XB(NN,3) >= XB(4) .OR. TEMP.XB(NN,5) >= XB(6) .OR. &
8992 TEMP.XB(NN,2) <= XB(1) .OR. TEMP.XB(NN,4) <= XB(3) .OR. TEMP.XB(NN,6) <= XB(5)) CYCLE
8993 IF ((TEMP.XB(NN,1) <= XB(2) .AND. TEMP.XB(NN,2) >= XB(1)) .AND. &
8994 (TEMP.XB(NN,3) <= XB(4) .AND. TEMP.XB(NN,4) >= XB(3)) .AND. &
8995 (TEMP.XB(NN,5) <= XB(6) .AND. TEMP.XB(NN,6) >= XB(5))) THEN
8996 IF (CONTROLLED(N) .OR. CONTROLLED(NN)) THEN
8997 WRITE(MESSAGE,'(A,I0,A,I0)') 'ERROR: Cannot overlap HOLES with a DEVC.ID or CTRL.ID. HOLE number',N,HOLE.O+1,&
8998 ', line number',INPUT_FILE.LINE.NUMBER
8999 CALL SHUTDOWN(MESSAGE) ; RETURN
9000 ENDIF
9001 ENDIF
9002 ENDDO
9003 ENDIF
9004
9005 ! Loop over all the meshes to determine where the HOLE is
9006
9007 MESH.LOOP: DO NM=1,NMESHES
9008
9009 IF (PROCESS(NM)/=MYID .AND. MYID/=EVAC.PROCESS) CYCLE MESH.LOOP
9010
9011 M=>MESHES(NM)
9012 CALL POINT.TO.MESH(NM)
9013
9014 ! Evacuation criteria
9015
9016 IF (MESH.ID/'null' .AND. MESH.ID/=MESHNAME(NM)) CYCLE MESH.LOOP
9017 IF (EVACUATION .AND. .NOT.EVACUATION_ONLY(NM)) CYCLE MESH.LOOP
9018 IF (EVACUATION_ONLY(NM)) THEN
9019 IF (.NOT.TEMP.HOLE.EVAC(N)) CYCLE MESH.LOOP
9020 DEVC.ID = 'null'
9021 CTRL.ID = 'null'
9022 ENDIF
9023
9024 ! Loop over all possible multiples of the HOLE
9025
9026 MR => MULTIPLIER(0)
9027 DO NNN=1,NMULT
9028 IF (MULT.ID==MULTIPLIER(NNN)%ID) MR => MULTIPLIER(NNN)
9029 ENDDO
9030
9031 K.MULT.LOOP: DO KK=MR%K.LOWER,MR%K.UPPER
9032 J.MULT.LOOP: DO JJ=MR%J.LOWER,MR%J.UPPER
9033 I.MULT.LOOP: DO II=MR%I.LOWER,MR%I.UPPER

```

```

9034
9035 IF (.NOT.MR%SEQUENTIAL) THEN
9036 X1 = XB(1) + MR%DX0 + I1*MR%DXB(1)
9037 X2 = XB(2) + MR%DX0 + I1*MR%DXB(2)
9038 Y1 = XB(3) + MR%DY0 + JJ*MR%DXB(3)
9039 Y2 = XB(4) + MR%DY0 + JJ*MR%DXB(4)
9040 Z1 = XB(5) + MR%DZ0 + KK*MR%DXB(5)
9041 Z2 = XB(6) + MR%DZ0 + KK*MR%DXB(6)
9042 ELSE
9043 X1 = XB(1) + MR%DX0 + I1*MR%DXB(1)
9044 X2 = XB(2) + MR%DX0 + I1*MR%DXB(2)
9045 Y1 = XB(3) + MR%DY0 + I1*MR%DXB(3)
9046 Y2 = XB(4) + MR%DY0 + I1*MR%DXB(4)
9047 Z1 = XB(5) + MR%DZ0 + I1*MR%DXB(5)
9048 Z2 = XB(6) + MR%DZ0 + I1*MR%DXB(6)
9049 ENDIF
9050
9051 ! Check if hole is contained within the current mesh
9052
9053 IF (X1>=XF .OR. X2<=XS .OR. Y1>YF .OR. Y2<=YS .OR. Z1>ZF .OR. Z2<=ZS) CYCLE LMULT_LOOP
9054
9055 ! Assign mesh-limited bounds
9056
9057 X1 = MAX(X1,XS-0.001_EB*DX(0))
9058 X2 = MIN(X2,XF+0.001_EB*DX(1BP1))
9059 Y1 = MAX(Y1,YS-0.001_EB*DY(0))
9060 Y2 = MIN(Y2,YF+0.001_EB*DY(1BP1))
9061 Z1 = MAX(Z1,ZS-0.001_EB*DZ(0))
9062 Z2 = MIN(Z2,ZF+0.001_EB*DZ(1BP1))
9063
9064 I1 = NINT( GINV(X1-XS,1,NM)*RDXI )
9065 I2 = NINT( GINV(X2-XS,1,NM)*RDXI )
9066 J1 = NINT( GINV(Y1-YS,2,NM)*RDETA )
9067 J2 = NINT( GINV(Y2-YS,2,NM)*RDETA )
9068 K1 = NINT( GINV(Z1-ZS,3,NM)*RDZETA )
9069 K2 = NINT( GINV(Z2-ZS,3,NM)*RDZETA )
9070
9071 ! Remove mean forcing in hole region
9072
9073 IF (ANY(MEAN_FORCING) .AND. BLOCK.WIND) THEN
9074 DO K=K1,K2+1
9075 DO J=J1,J2+1
9076 DO I=I1,I2+1
9077 MB%MEAN_FORCING.CELL(I,J,K) = .FALSE.
9078 ENDDO
9079 ENDDO
9080 ENDDO
9081 CYCLE LMULT_LOOP
9082 ENDIF
9083
9084 NN=0
9085 OBST_LOOP: DO
9086 NN=NN+1
9087 IF (NN>N_OBST) EXIT OBST_LOOP
9088 OB=>OBSTRUCTION(NN)
9089 DEVC_INDEX_O = OB%DEVC_INDEX
9090 CTRL_INDEX_O = OB%CTRL_INDEX
9091 IF (.NOT.OB%PERMIT_HOLE) CYCLE OBST_LOOP
9092
9093 ! TEMP_OBST(0) is the intersection of HOLE and OBST
9094
9095 TEMP_OBST(0) = OBSTRUCTION(NN)
9096
9097 TEMP_OBST(0)%I1 = MAX(I1,OB%I1)
9098 TEMP_OBST(0)%I2 = MIN(I2,OB%I2)
9099 TEMP_OBST(0)%J1 = MAX(J1,OB%J1)
9100 TEMP_OBST(0)%J2 = MIN(J2,OB%J2)
9101 TEMP_OBST(0)%K1 = MAX(K1,OB%K1)
9102 TEMP_OBST(0)%K2 = MIN(K2,OB%K2)
9103
9104 TEMP_OBST(0)%X1 = MAX(X1,OB%X1)
9105 TEMP_OBST(0)%X2 = MIN(X2,OB%X2)
9106 TEMP_OBST(0)%Y1 = MAX(Y1,OB%Y1)
9107 TEMP_OBST(0)%Y2 = MIN(Y2,OB%Y2)
9108 TEMP_OBST(0)%Z1 = MAX(Z1,OB%Z1)
9109 TEMP_OBST(0)%Z2 = MIN(Z2,OB%Z2)
9110
9111 ! Ignore OBSTs that do not intersect with HOLE or are merely sliced by the hole.
9112
9113 IF (TEMP_OBST(0)%I2-TEMP_OBST(0)%I1<0 .OR. TEMP_OBST(0)%J2-TEMP_OBST(0)%J1<0 .OR. &
9114 TEMP_OBST(0)%K2-TEMP_OBST(0)%K1<0) CYCLE OBST_LOOP
9115 IF (TEMP_OBST(0)%I2-TEMP_OBST(0)%I1==0) THEN
9116 IF (OB%I1<TEMP_OBST(0)%I1 .OR. OB%I2>TEMP_OBST(0)%I2) CYCLE OBST_LOOP
9117 ENDIF
9118 IF (TEMP_OBST(0)%J2-TEMP_OBST(0)%J1==0) THEN
9119 IF (OB%J1<TEMP_OBST(0)%J1 .OR. OB%J2>TEMP_OBST(0)%J2) CYCLE OBST_LOOP
9120 ENDIF
9121 IF (TEMP_OBST(0)%K2-TEMP_OBST(0)%K1==0) THEN

```

Source Code files for edited portions of FDS

```

9122 IF (OB%K1<TEMP_OBST(0)%K1 .OR. OB%K2>TEMP_OBST(0)%K2) CYCLE OBST_LOOP
9123 ENDF
9124
9125 IF (TEMP_OBST(0)%X2<=X1 .OR. TEMP_OBST(0)%X1>=X2 .OR. TEMP_OBST(0)%Y2<=Y1 .OR. TEMP_OBST(0)%Y1>=Y2 .OR. &
9126 TEMP_OBST(0)%Z2<=Z1 .OR. TEMP_OBST(0)%Z1>=Z2) CYCLE OBST_LOOP
9127
9128 ! Start counting new OBSTs that need to be created
9129
9130 NDO=0
9131
9132 IF ((OB%I1<I1 .AND. I1<OB%I2) .OR. (XB(1)>=XS .AND. I1==0 .AND. OB%I1==0)) THEN
9133 NDO=NDO+1
9134 TEMP_OBST(NDO)=OBSTRUCTION(NN)
9135 TEMP_OBST(NDO)%I1 = OB%I1
9136 TEMP_OBST(NDO)%I2 = I1
9137 TEMP_OBST(NDO)%X1 = OB%X1
9138 TEMP_OBST(NDO)%X2 = X1
9139 ENDF
9140
9141 IF ((OB%I1<I2 .AND. I2<OB%I2) .OR. (XB(2)<=XF .AND. I2==IBAR .AND. OB%I2==IBAR)) THEN
9142 NDO=NDO+1
9143 TEMP_OBST(NDO)=OBSTRUCTION(NN)
9144 TEMP_OBST(NDO)%I1 = I2
9145 TEMP_OBST(NDO)%I2 = OB%I2
9146 TEMP_OBST(NDO)%X1 = X2
9147 TEMP_OBST(NDO)%X2 = OB%X2
9148 ENDF
9149
9150 IF ((OB%J1<J1 .AND. J1<OB%J2) .OR. (XB(3)>=YS .AND. J1==0 .AND. OB%J1==0)) THEN
9151 NDO=NDO+1
9152 TEMP_OBST(NDO)=OBSTRUCTION(NN)
9153 TEMP_OBST(NDO)%I1 = MAX(I1 ,OB%I1 )
9154 TEMP_OBST(NDO)%I2 = MIN(I2 ,OB%I2 )
9155 TEMP_OBST(NDO)%X1 = MAX(X1 ,OB%X1 )
9156 TEMP_OBST(NDO)%X2 = MIN(X2 ,OB%X2 )
9157 TEMP_OBST(NDO)%J1 = OB%J1
9158 TEMP_OBST(NDO)%J2 = J1
9159 TEMP_OBST(NDO)%Y1 = OB%Y1
9160 TEMP_OBST(NDO)%Y2 = Y1
9161 ENDF
9162
9163 IF ((OB%J1<J2 .AND. J2<OB%J2) .OR. (XB(4)<=YF .AND. J2==JBAR .AND. OB%J2==JBAR)) THEN
9164 NDO=NDO+1
9165 TEMP_OBST(NDO)=OBSTRUCTION(NN)
9166 TEMP_OBST(NDO)%I1 = MAX(I1 ,OB%I1 )
9167 TEMP_OBST(NDO)%I2 = MIN(I2 ,OB%I2 )
9168 TEMP_OBST(NDO)%X1 = MAX(X1 ,OB%X1 )
9169 TEMP_OBST(NDO)%X2 = MIN(X2 ,OB%X2 )
9170 TEMP_OBST(NDO)%J1 = J2
9171 TEMP_OBST(NDO)%J2 = OB%J2
9172 TEMP_OBST(NDO)%Y1 = Y2
9173 TEMP_OBST(NDO)%Y2 = OB%Y2
9174 ENDF
9175
9176 IF ((OB%K1<K1 .AND. K1<OB%K2) .OR. (XB(5)>=ZS .AND. K1==0 .AND. OB%K1==0)) THEN
9177 NDO=NDO+1
9178 TEMP_OBST(NDO)=OBSTRUCTION(NN)
9179 TEMP_OBST(NDO)%I1 = MAX(I1 ,OB%I1 )
9180 TEMP_OBST(NDO)%I2 = MIN(I2 ,OB%I2 )
9181 TEMP_OBST(NDO)%X1 = MAX(X1 ,OB%X1 )
9182 TEMP_OBST(NDO)%X2 = MIN(X2 ,OB%X2 )
9183 TEMP_OBST(NDO)%J1 = MAX(J1 ,OB%J1 )
9184 TEMP_OBST(NDO)%J2 = MIN(J2 ,OB%J2 )
9185 TEMP_OBST(NDO)%Y1 = MAX(Y1 ,OB%Y1 )
9186 TEMP_OBST(NDO)%Y2 = MIN(Y2 ,OB%Y2 )
9187 TEMP_OBST(NDO)%K1 = OB%K1
9188 TEMP_OBST(NDO)%K2 = K1
9189 TEMP_OBST(NDO)%Z1 = OB%Z1
9190 TEMP_OBST(NDO)%Z2 = Z1
9191 ENDF
9192
9193 IF ((OB%K1<K2 .AND. K2<OB%K2) .OR. (XB(6)<=ZF .AND. K2==KBAR .AND. OB%K2==KBAR)) THEN
9194 NDO=NDO+1
9195 TEMP_OBST(NDO)=OBSTRUCTION(NN)
9196 TEMP_OBST(NDO)%I1 = MAX(I1 ,OB%I1 )
9197 TEMP_OBST(NDO)%I2 = MIN(I2 ,OB%I2 )
9198 TEMP_OBST(NDO)%X1 = MAX(X1 ,OB%X1 )
9199 TEMP_OBST(NDO)%X2 = MIN(X2 ,OB%X2 )
9200 TEMP_OBST(NDO)%J1 = MAX(J1 ,OB%J1 )
9201 TEMP_OBST(NDO)%J2 = MIN(J2 ,OB%J2 )
9202 TEMP_OBST(NDO)%Y1 = MAX(Y1 ,OB%Y1 )
9203 TEMP_OBST(NDO)%Y2 = MIN(Y2 ,OB%Y2 )
9204 TEMP_OBST(NDO)%K1 = K2
9205 TEMP_OBST(NDO)%K2 = OB%K2
9206 TEMP_OBST(NDO)%Z1 = Z2
9207 TEMP_OBST(NDO)%Z2 = OB%Z2
9208 ENDF
9209

```

## Source Code files for edited portions of FDS

```

9210  ! Maintain ordinal rank of original obstruction, but negate it. This will be a code for Smokeview.
9211
9212  TEMP_OBST(:)%ORDINAL = -OB%ORDINAL
9213
9214  ! Re-allocate space of new OBSTs, or remove entry for dead OBST
9215
9216  NEW_OBST_IF: IF (NDO>0) THEN
9217  CALL RE_ALLOCATE_OBST(NM,N_OBST,NDO)
9218  OBSTRUCTION=>M%OBSTRUCTION
9219  OBSTRUCTION(N_OBST+1:N_OBST+NDO) = TEMP_OBST(1:NDO)
9220  N_OBST = N_OBST + NDO
9221  ENDFIF NEW_OBST_IF
9222
9223  ! If the HOLE is to be created or removed, save it in OBSTRUCTION(NN), the original OBST that was broken up
9224
9225  DEVC_OR_CTRL: IF (DEVC_ID/='null' .OR. CTRL_ID/='null') THEN
9226
9227  OBSTRUCTION(NN) = TEMP_OBST(0)
9228  OB => OBSTRUCTION(NN)
9229  OB%DEVC_INDEX_O = DEVC_INDEX_O
9230  OB%CTRL_INDEX_O = CTRL_INDEX_O
9231  OB%DEVC_ID = DEVC_ID
9232  OB%CTRL_ID = CTRL_ID
9233  CALL SEARCH_CONTROLLER('HOLE',CTRL_ID,DEVC_ID,OB%DEVC_INDEX,OB%CTRL_INDEX,N)
9234  IF (DEVC_ID/='null' .OR. CTRL_ID /='null') THEN
9235  OB%REMOVABLE = .TRUE.
9236  OB%HOLE_FILLER = .TRUE.
9237  IF (DEVC_ID/='null') OB%CTRL_INDEX = -1
9238  IF (CTRL_ID/='null') OB%DEVC_INDEX = -1
9239  ENDFIF
9240  IF (OB%CONSUMABLE) OB%REMOVABLE = .TRUE.
9241
9242  SELECT CASE (COLOR)
9243  CASE ('INVISIBLE')
9244  OB%COLOR_INDICATOR = -3
9245  OB%RGB(1) = 255
9246  OB%RGB(2) = 204
9247  OB%RGB(3) = 102
9248  OB%TRANSPARENCY = 0. .EB
9249  CASE ('null')
9250  IF (ANY(RGB>0)) THEN
9251  OB%COLOR_INDICATOR = -3
9252  OB%RGB = RGB
9253  OB%TRANSPARENCY = TRANSPARENCY
9254  ENDFIF
9255  CASE DEFAULT
9256  CALL COLOR2RGB(RGB,COLOR)
9257  OB%COLOR_INDICATOR = -3
9258  OB%RGB = RGB
9259  OB%TRANSPARENCY = TRANSPARENCY
9260  END SELECT
9261
9262  ELSE DEVC_OR_CTRL
9263
9264  OBSTRUCTION(NN) = OBSTRUCTION(N_OBST)
9265  N_OBST = N_OBST-1
9266  NN = NN-1
9267
9268  ENDFIF DEVC_OR_CTRL
9269
9270  ENDDO OBST_LOOP
9271  ENDDO I_MULT_LOOP
9272  ENDDO J_MULT_LOOP
9273  ENDDO K_MULT_LOOP
9274  ENDDO MESH_LOOP
9275  ENDDO READ_HOLE_LOOP
9276
9277  REWIND(LU_INPUT) ; INPUT_FILE_LINE_NUMBER = 0
9278
9279  IF (ANY(EVACUATION_ONLY)) DEALLOCATE(TEMP_HOLE_EVAC)
9280  DEALLOCATE(TEMP_OBST)
9281  DEALLOCATE (CONTROLLED)
9282  DEALLOCATE (TEMP_XB)
9283
9284  CONTAINS
9285
9286  SUBROUTINE DEFINE_EVACUATION_HOLES(IMODE)
9287  !
9288  ! Clear the STRS meshes by a hole with size of XB of the stairs.
9289  ! The core is put there applying permit_hole=false.
9290  USE EVAC, ONLY: N_STRS, EMESH_STAIRS, EVAC_EMESH_STAIRS_TYPE
9291  IMPLICIT NONE
9292  ! Passed variables
9293  INTEGER, INTENT(IN) :: IMODE
9294  ! Local variables
9295  INTEGER :: I
9296  REAL(EB) :: TINY_X, TINY_Y, TINY_Z
9297

```

```

9298 IF (.NOT.ANY(EVACUATION_ONLY)) RETURN
9299 IMODE.1.IF: IF (IMODE==1) THEN
9300 NSTRS_LOOP.1: DO I = 1, N_STRS
9301 IF (.NOT.EMESH_STAIRS(1)%DEFINE_MESH) CYCLE NSTRS_LOOP.1
9302 N_HOLE.O = N_HOLE.O + 1
9303 N_HOLE = N_HOLE + 1 ! No mult for evacuation strs meshes
9304 EMESH_STAIRS(1)%L_HOLE = N_HOLE.O
9305 EVACUATION_HOLE = .TRUE.
9306 END DO NSTRS_LOOP.1
9307 END IF IMODE.1.IF
9308
9309 IMODE.2.IF: IF (IMODE==2) THEN
9310 EVACUATION_HOLE = .FALSE.
9311 NSTRS_LOOP.2: DO I = 1, N_STRS
9312 IF (.NOT.EMESH_STAIRS(1)%DEFINE_MESH) CYCLE NSTRS_LOOP.2
9313 IF (.NOT.EMESH_STAIRS(1)%L_HOLE==N) CYCLE NSTRS_LOOP.2
9314 EVACUATION_HOLE = .TRUE.
9315 EXIT NSTRS_LOOP.2
9316 END DO NSTRS_LOOP.2
9317 END IF IMODE.2.IF
9318
9319 IMODE.3.IF: IF (IMODE==3) THEN
9320 EVACUATION_HOLE = .FALSE.
9321 NSTRS_LOOP.3: DO I = 1, N_STRS
9322 IF (.NOT.EMESH_STAIRS(1)%DEFINE_MESH) CYCLE NSTRS_LOOP.3
9323 IF (.NOT.EMESH_STAIRS(1)%L_HOLE==N) CYCLE NSTRS_LOOP.3
9324 EVACUATION_HOLE = .TRUE.
9325 RGB = EMESH_STAIRS(1)%RGB
9326 XB = EMESH_STAIRS(1)%XB
9327 TINY_X = 0.01_EB*(EMESH_STAIRS(1)%XB(2)-EMESH_STAIRS(1)%XB(1))/EMESH_STAIRS(1)%BAR
9328 TINY_Y = 0.01_EB*(EMESH_STAIRS(1)%XB(4)-EMESH_STAIRS(1)%XB(3))/EMESH_STAIRS(1)%BAR
9329 TINY_Z = 0.01_EB
9330 XB(1) = XB(1)-TINY_X ; XB(2) = XB(2)+TINY_X
9331 XB(3) = XB(3)-TINY_Y ; XB(4) = XB(4)+TINY_Y
9332 XB(5) = XB(5)-TINY_Z ; XB(6) = XB(6)+TINY_Z
9333 EVACUATION = .TRUE.
9334 MESH_ID = TRIM(MESHNAME(EMESH_STAIRS(1)%MESH))
9335 EXIT NSTRS_LOOP.3
9336 END DO NSTRS_LOOP.3
9337 END IF IMODE.3.IF
9338
9339 END SUBROUTINE DEFINE_EVACUATION_HOLES
9340
9341 END SUBROUTINE READ_HOLE
9342
9343
9344 SUBROUTINE RE_ALLOCATE_OBST(NM,N_OBST,NDO)
9345
9346 TYPE (OBSTRUCTION_TYPE), ALLOCATABLE, DIMENSION(:) :: DUMMY
9347 INTEGER, INTENT(IN) :: NM,NDO,N_OBST
9348 TYPE (MESH_TYPE), POINTER :: M=>NULL()
9349 M=>MESHES(NM)
9350 ALLOCATE(DUMMY(0:N_OBST))
9351 DUMMY(0:N_OBST) = M%OBSTRUCTION(0:N_OBST)
9352 DEALLOCATE(M%OBSTRUCTION)
9353 ALLOCATE(M%OBSTRUCTION(0:N_OBST+NDO))
9354 M%OBSTRUCTION(0:N_OBST) = DUMMY(0:N_OBST)
9355 DEALLOCATE(DUMMY)
9356 END SUBROUTINE RE_ALLOCATE_OBST
9357
9358
9359 SUBROUTINE READ_VENT
9360
9361 USE GEOMETRY_FUNCTIONS, ONLY : BLOCK_CELL,CIRCLE_CELL_INTERSECTION_AREA
9362 USE DEVICE_VARIABLES, ONLY : DEVICE
9363 USE CONTROL_VARIABLES, ONLY : CONTROL
9364 USE MATH_FUNCTIONS, ONLY : GET_RAMP_INDEX
9365
9366 INTEGER :: N,NN,NNM,NNN,N_VENT_O,IOR,I1,I2,J1,J2,K1,K2,RGB(3),N_EDDY,N_VENT_NEW,II,JJ,KK
9367 REAL(EB) :: SPREAD_RATE,TRANSPARENCY,XYZ(3),TMP_EXTERIOR,DYNAMIC_PRESSURE,XB1,XB2,XB3,XB4,XB5,XB6, &
9368 REYNOLDS_STRESS(3,3),L_EDDY,VEL_RMS,L_EDDY_IJ(3,3),UWV(3),RADIUS
9369 CHARACTER(LABEL_LENGTH) :: ID,DEV_C_ID,CTRL_ID,SURF_ID,PRESSURE_RAMP,TMP_EXTERIOR_RAMP,MULT_ID
9370 CHARACTER(60) :: MESH_ID
9371 CHARACTER(25) :: COLOR
9372 TYPE(MULTIPLIER_TYPE), POINTER :: MR
9373 LOGICAL :: REJECT_VENT,EVACUATION,OUTLINE,EVACUATION_VENT,WIND
9374 NAMELIST /VENT/ COLOR,CTRL_ID,DEV_C_ID,DYNAMIC_PRESSURE,EVACUATION,FYI,ID,IOR,L_EDDY,L_EDDY_IJ,MB,MESH_ID,MULT_ID,
9375 N_EDDY,OUTLINE,&
9376 PBX,PBY,PBZ,PRESSURE_RAMP,RADIUS,REYNOLDS_STRESS,RGB,SPREAD_RATE,SURF_ID,TEXTURE_ORIGIN,TMP_EXTERIOR,&
9377 TMP_EXTERIOR_RAMP,TRANSPARENCY,UWV,VEL_RMS,WIND,XB,XYZ
9378
9379 MESH_LOOP.1: DO NM=1,NMESHES
9380
9381 M=>MESHES(NM)
9382 CALL POINT_TO_MESH(NM)
9383
9384 REWIND(LU_INPUT) ; INPUT_FILE_LINE_NUMBER = 0
9385 N_VENT = 0

```



```

9385 COUNT.VENT_LOOP: DO
9386 CALL CHECKREAD('VENT',LU_INPUT,IOS)
9387 IF (IOS==1) EXIT COUNT.VENT_LOOP
9388 ID = 'null'
9389 MULT_ID = 'null'
9390 SURF_ID = 'null'
9391 READ(LU_INPUT,NML=VENT,END=3,ERR=4,IOSTAT=IOS)
9392 N.VENT_NEW = 0
9393 IF (MULT_ID=='null') THEN
9394 N.VENT_NEW = 1
9395 ELSE
9396 IF (SURF_ID=='HVAC') THEN
9397 WRITE(MESSAGE,'(A,I0,A,I0)') 'ERROR: Cannot use MULT with an HVAC VENT, VENT ', N.VENT+1,&
9398 ', line number',INPUT_FILE.LINE_NUMBER
9399 CALL SHUTDOWN(MESSAGE) ; RETURN
9400 ENDIF
9401 DO N=1,N_MULT
9402 MR => MULTIPLIER(N)
9403 IF (MULT_ID==MR%ID) N.VENT_NEW = MR%N_COPIES
9404 ENDDO
9405 IF (N.VENT_NEW==0) THEN
9406 WRITE(MESSAGE,'(A,A,A,I0,A,I0)') 'ERROR: MULT line ', TRIM(MULT_ID),' not found on VENT ', N.VENT+1,&
9407 ', line number',INPUT_FILE.LINE_NUMBER
9408 CALL SHUTDOWN(MESSAGE) ; RETURN
9409 ENDIF
9410 ENDIF
9411 IF (SURF_ID=='HVAC' .AND. ID=='null') THEN
9412 WRITE(MESSAGE,'(A,I0,A,I0)') 'ERROR: must specify an ID for an HVAC VENT, VENT ', N.VENT+1,&
9413 ', line number',INPUT_FILE.LINE_NUMBER
9414 CALL SHUTDOWN(MESSAGE) ; RETURN
9415 ENDIF
9416 N.VENT = N.VENT + N.VENT_NEW
9417 4 IF (IOS>0) THEN
9418 WRITE(MESSAGE,'(A,I0,A,I0)') 'ERROR: Problem with VENT ',N.VENT+1,', line number',INPUT_FILE.LINE_NUMBER
9419 CALL SHUTDOWN(MESSAGE) ; RETURN
9420 ENDIF
9421 ENDDO COUNT.VENT_LOOP
9422 3 REWIND(LU_INPUT) ; INPUT_FILE.LINE_NUMBER = 0
9423
9424 IF (EVACUATION_ONLY(NM)) CALL DEFINE_EVACUATION_VENTS(NM,1)
9425
9426 IF (TWO_D) N.VENT = N.VENT + 2
9427 IF (CYLINDRICAL .AND. M%XS<=TWO_EPSILON_EB) N.VENT = N.VENT + 1
9428 IF (EVACUATION_ONLY(NM)) N.VENT = N.VENT + 2
9429
9430 ALLOCATE(M%VENTS(N.VENT),STAT=IZERO)
9431 CALL ChkMemErr('READ','VENTS',IZERO)
9432 VENTS->M%VENTS
9433
9434 N.VENT_O = N.VENT
9435 N = 0
9436
9437 REWIND(LU_INPUT) ; INPUT_FILE.LINE_NUMBER = 0
9438 READ.VENT_LOOP: DO NN=1,N.VENT_O
9439
9440 IOR = 0
9441 MB = 'null'
9442 PBX = -1.E6_EB
9443 PBY = -1.E6_EB
9444 PBZ = -1.E6_EB
9445 SURF_ID = 'null'
9446 COLOR = 'null'
9447 MESH_ID = 'null'
9448 MULT_ID = 'null'
9449 ID = 'null'
9450 RGB = -1
9451 TRANSPARENCY = 1._EB
9452 DYNAMIC_PRESSURE = 0._EB
9453 PRESSURE_RAMP = 'null'
9454 XYZ = -1.E6_EB
9455 SPREAD_RATE = -1._EB
9456 TMP_EXTERIOR = -1000.
9457 TMP_EXTERIOR_RAMP = 'null'
9458 TEXTURE_ORIGIN = -999._EB
9459 OUTLINE = .FALSE.
9460 DEVC_ID = 'null'
9461 CTRL_ID = 'null'
9462 EVACUATION = .FALSE.
9463 N_EDDY=0
9464 L_EDDY=0._EB
9465 L_EDDY_IJ=0._EB
9466 VEL_RMS=0._EB
9467 REYNOLDS_STRESS=0._EB
9468 UWW = -1.E12_EB
9469 RADIUS = -1._EB
9470 WIND = .FALSE.
9471
9472 IF (NN==N.VENT_O-2 .AND. CYLINDRICAL .AND. XS<=TWO_EPSILON_EB) MB='XMIN'

```

Source Code files for edited portions of FDS

```

9473 IF (NN==N.VENT_O-1 .AND. TWO.D) MB= 'YMIN'
9474 IF (NN==N.VENT_O .AND. TWO.D) MB= 'YMAX'
9475 IF (NN==N.VENT_O-1 .AND. EVACUATION_ONLY(NM)) MB= 'ZMIN'
9476 IF (NN==N.VENT_O .AND. EVACUATION_ONLY(NM)) MB= 'ZMAX'
9477
9478 IF (MB='null') THEN
9479 EVACUATION.VENT = .FALSE.
9480 IF (EVACUATION_ONLY(NM)) CALL DEFINE_EVACUATION_VENTS(NM,2)
9481 EVACUATION.VENTS: IF (.NOT. EVACUATION.VENT) THEN
9482 CALL CHECKREAD('VENT',LU.INPUT,IOS)
9483 IF (IOS==1) EXIT READ_VENT_LOOP
9484 READ(LU.INPUT,VENT,END=37) ! Read in info for VENT N
9485 END IF EVACUATION.VENTS
9486 ELSE
9487 SURF.ID = 'MIRROR'
9488 ENDIF
9489
9490 IF (MESH.ID/'null' .AND. MESH.ID/=MESHNAME(NM)) CYCLE READ_VENT_LOOP
9491
9492 IF (PBX>-1.E5.EB .OR. PBY>-1.E5.EB .OR. PBZ>-1.E5.EB) THEN
9493 IF (MULT.ID/'null') THEN
9494 WRITE(MESSAGE,'(A,I0,A)') 'ERROR: MULT.ID cannot be applied to VENT',NN,' because it uses PBX, PBY or PBZ.'
9495 CALL SHUTDOWN(MESSAGE) ; RETURN
9496 ENDIF
9497 XB(1) = XS
9498 XB(2) = XF
9499 XB(3) = YS
9500 XB(4) = YF
9501 XB(5) = ZS
9502 XB(6) = ZF
9503 IF (PBX>-1.E5.EB) XB(1:2) = PBX
9504 IF (PBY>-1.E5.EB) XB(3:4) = PBY
9505 IF (PBZ>-1.E5.EB) XB(5:6) = PBZ
9506 ENDIF
9507
9508 IF (MB/'null') THEN
9509 IF (NMESHES>1 .AND. SURF.ID=='PERIODIC') THEN
9510 WRITE(MESSAGE,'(A,I0,A)') 'ERROR: Use PBX,PBY,PBZ or XB for VENT',NN,' multi-mesh PERIODIC boundary'
9511 CALL SHUTDOWN(MESSAGE) ; RETURN
9512 ENDIF
9513 IF (MULT.ID/'null') THEN
9514 WRITE(MESSAGE,'(A,I0,A)') 'ERROR: MULT.ID cannot be applied to VENT',NN,' because it uses MB.'
9515 CALL SHUTDOWN(MESSAGE) ; RETURN
9516 ENDIF
9517 XB(1) = XS
9518 XB(2) = XF
9519 XB(3) = YS
9520 XB(4) = YF
9521 XB(5) = ZS
9522 XB(6) = ZF
9523 SELECT CASE (MB)
9524 CASE ('XMIN')
9525 XB(2) = XS
9526 CASE ('XMAX')
9527 XB(1) = XF
9528 CASE ('YMIN')
9529 XB(4) = YS
9530 CASE ('YMAX')
9531 XB(3) = YF
9532 CASE ('ZMIN')
9533 XB(6) = ZS
9534 CASE ('ZMAX')
9535 XB(5) = ZF
9536 CASE DEFAULT
9537 WRITE(MESSAGE,'(A,I0,A)') 'ERROR: MB specified for VENT',NN,' is not XMIN, XMAX, YMIN, YMAX, ZMIN, or ZMAX'
9538 CALL SHUTDOWN(MESSAGE) ; RETURN
9539 END SELECT
9540 ENDIF
9541
9542 ! Check that the vent is properly specified
9543
9544 IF (ABS(XB(3)-XB(4))<=SPACING(XB(4)) .AND. TWO.D .AND. NN<N.VENT_O-1) THEN
9545 IF (ID/'null')WRITE(MESSAGE,'(A,I0,A)') 'ERROR: VENT ',NN,' cannot be specified on a y boundary in a 2D
calculation'
9546 IF (ID/'null')WRITE(MESSAGE,'(A,A,A)') 'ERROR: VENT ',TRIM(ID),' cannot be specified on a y boundary in a 2D
calculation'
9547 CALL SHUTDOWN(MESSAGE) ; RETURN
9548 ENDIF
9549
9550 IF (ABS(XB(1)-XB(2))>SPACING(XB(2)) .AND. ABS(XB(3)-XB(4))>SPACING(XB(4)) .AND. ABS(XB(5)-XB(6))>SPACING(XB(6))
) THEN
9551 IF (ID/'null') WRITE(MESSAGE,'(A,I0,A)') 'ERROR: VENT ',NN,' must be a plane'
9552 IF (ID/'null') WRITE(MESSAGE,'(A,A,A)') 'ERROR: VENT ',TRIM(ID),' must be a plane'
9553 CALL SHUTDOWN(MESSAGE) ; RETURN
9554 ENDIF
9555
9556 CALL CHECK_XB(XB)
9557

```

```

9558 IF (ALL(EVACUATION_ONLY)) THEN
9559 DEVC_ID = 'null'
9560 CTRL_ID = 'null'
9561 END IF
9562
9563 ! Loop over all possible multiples of the OBST
9564
9565 MR => MULTIPLIER(0)
9566 DO NNN=1,NMULT
9567 IF (MULT_ID==MULTIPLIER(NNN)%ID) MR => MULTIPLIER(NNN)
9568 ENDDO
9569
9570 K_MULT_LOOP: DO KK=MR%K_LOWER,MR%K_UPPER
9571 J_MULT_LOOP: DO JJ=MR%J_LOWER,MR%J_UPPER
9572 L_MULT_LOOP: DO II=MR%L_LOWER,MR%L_UPPER
9573
9574 REJECT_VENT = .FALSE.
9575
9576 IF (.NOT.MR%SEQUENTIAL) THEN
9577 XB1 = XB(1) + MR%DX0 + II*MR%DXB(1)
9578 XB2 = XB(2) + MR%DX0 + II*MR%DXB(2)
9579 XB3 = XB(3) + MR%DY0 + JJ*MR%DXB(3)
9580 XB4 = XB(4) + MR%DY0 + JJ*MR%DXB(4)
9581 XB5 = XB(5) + MR%DZ0 + KK*MR%DXB(5)
9582 XB6 = XB(6) + MR%DZ0 + KK*MR%DXB(6)
9583 ELSE
9584 XB1 = XB(1) + MR%DX0 + II*MR%DXB(1)
9585 XB2 = XB(2) + MR%DX0 + II*MR%DXB(2)
9586 XB3 = XB(3) + MR%DY0 + II*MR%DXB(3)
9587 XB4 = XB(4) + MR%DY0 + II*MR%DXB(4)
9588 XB5 = XB(5) + MR%DZ0 + II*MR%DXB(5)
9589 XB6 = XB(6) + MR%DZ0 + II*MR%DXB(6)
9590 ENDIF
9591
9592 ! Increase the VENT counter
9593
9594 N = N + 1
9595
9596 VT=>VENTS(N)
9597
9598 IF (ABS(XB1-XB2)<=-SPACING(XB2)) VT%UNDIVIDED.INPUT_AREA = (XB4-XB3)*(XB6-XB5)
9599 IF (ABS(XB3-XB4)<=-SPACING(XB4)) VT%UNDIVIDED.INPUT_AREA = (XB2-XB1)*(XB6-XB5)
9600 IF (ABS(XB5-XB6)<=-SPACING(XB6)) VT%UNDIVIDED.INPUT_AREA = (XB2-XB1)*(XB4-XB3)
9601 IF (RADIUS>0..EB) VT%UNDIVIDED.INPUT_AREA = P1*RADIUS**2
9602
9603 VT%X1_ORIG = XB1
9604 VT%X2_ORIG = XB2
9605 VT%Y1_ORIG = XB3
9606 VT%Y2_ORIG = XB4
9607 VT%Z1_ORIG = XB5
9608 VT%Z2_ORIG = XB6
9609
9610 XB1 = MAX(XB1, XS)
9611 XB2 = MIN(XB2, XF)
9612 XB3 = MAX(XB3, YS)
9613 XB4 = MIN(XB4, YF)
9614 XB5 = MAX(XB5, ZS)
9615 XB6 = MIN(XB6, ZF)
9616
9617 IF ((XB1-XF)>SPACING(XF) .OR. (XS-XB2)>SPACING(XS) .OR. &
9618 (XB3-YF)>SPACING(YF) .OR. (YS-XB4)>SPACING(YS) .OR. &
9619 (XB5-ZF)>SPACING(ZF) .OR. (ZS-XB6)>SPACING(ZS)) REJECT_VENT = .TRUE.
9620
9621 VT%i1 = MAX(0, NINT(GINV(XB1-XS,1,NM)*RDXI))
9622 VT%i2 = MIN(IBAR, NINT(GINV(XB2-XS,1,NM)*RDXI))
9623 VT%j1 = MAX(0, NINT(GINV(XB3-YS,2,NM)*RDETA))
9624 VT%j2 = MIN(JBAR, NINT(GINV(XB4-YS,2,NM)*RDETA))
9625 VT%k1 = MAX(0, NINT(GINV(XB5-ZS,3,NM)*RDZETA))
9626 VT%k2 = MIN(KBAR, NINT(GINV(XB6-ZS,3,NM)*RDZETA))
9627
9628 ! Thicken evacuation mesh vents in the z direction
9629
9630 IF (EVACUATION_ONLY(NM) .AND. EVACUATION .AND. VT%k1==VT%k2 .AND. .NOT.REJECT_VENT) THEN
9631 VT%k1 = INT(GINV(.5..EB*(XB5+XB6)-ZS,3,NM)*RDZETA)
9632 VT%k2 = KBAR
9633 XB5 = ZS
9634 XB6 = ZF
9635 IF (ABS(XB1-XB2)>SPACING(XB2) .AND. ABS(XB3-XB4)>SPACING(XB4)) THEN
9636 IF (ID=='null') WRITE(MESSAGE,'(A,I0,A)') 'ERROR: Evacuation VENT ',NN, ' must be a vertical plane'
9637 IF (ID!='null') WRITE(MESSAGE,'(A,A,A)') 'ERROR: Evacuation VENT ',TRIM(ID), ' must be a vertical plane'
9638 CALL SHUIDOWN(MESSAGE) ; RETURN
9639 ENDIF
9640 ENDIF
9641
9642 IF (ABS(XB1-XB2)<=-SPACING(XB2)) THEN
9643 IF (VT%i1==VT%j2 .OR. VT%k1==VT%k2) REJECT_VENT=.TRUE.
9644 IF (VT%i1>IBAR .OR. VT%i2<0) REJECT_VENT=.TRUE.
9645 ENDIF

```

```

9646 IF (ABS(XB3-XB4)<=SPACING(XB4) ) THEN
9647 IF (VT%I1==VT%I2 .OR. VT%K1==VT%K2) REJECT_VENT=.TRUE.
9648 IF (VT%J1>JBAR .OR. VT%J2<0) REJECT_VENT=.TRUE.
9649 ENDIF
9650 IF (ABS(XB5-XB6)<=SPACING(XB6) ) THEN
9651 IF (VT%I1==VT%I2 .OR. VT%J1==VT%J2) REJECT_VENT=.TRUE.
9652 IF (VT%K1>KBAR .OR. VT%K2<0) REJECT_VENT=.TRUE.
9653 ENDIF
9654
9655 ! Evacuation criteria
9656
9657 IF (.NOT.EVACUATION .AND. EVACUATION_ONLY(NM)) REJECT_VENT=.TRUE.
9658 IF (EVACUATION .AND. .NOT.EVACUATION_ONLY(NM)) REJECT_VENT=.TRUE.
9659
9660 IF (ALL(EVACUATION_ONLY)) THEN
9661 DEVC_ID = 'null'
9662 CTRL_ID = 'null'
9663 END IF
9664
9665 ! If the VENT is to be rejected
9666
9667 IF (REJECT_VENT) THEN
9668 N = N-1
9669 N_VENT = N_VENT-1
9670 CYCLE LMULT_LOOP
9671 ENDIF
9672
9673 ! Vent area
9674
9675 VT%X1 = XB1
9676 VT%X2 = XB2
9677 VT%Y1 = XB3
9678 VT%Y2 = XB4
9679 VT%Z1 = XB5
9680 VT%Z2 = XB6
9681
9682 IF (ABS(XB1-XB2)<=SPACING(XB2) ) VT%INPUT_AREA = (XB4-XB3)*(XB6-XB5)
9683 IF (ABS(XB3-XB4)<=SPACING(XB4) ) VT%INPUT_AREA = (XB2-XB1)*(XB6-XB5)
9684 IF (ABS(XB5-XB6)<=SPACING(XB6) ) VT%INPUT_AREA = (XB2-XB1)*(XB4-XB3)
9685
9686 ! Check the SURF_ID against the list of SURF's
9687
9688 CALL CHECK_SURF_NAME(SURF_ID,EX)
9689 IF (.NOT.EX) THEN
9690 WRITE(MESSAGE,'(A,A,A,I0,A,I0)') 'ERROR: SURF_ID ',TRIM(SURF_ID),' not found for VENT ',N_VENT,&
9691 ', line number ',INPUT_FILE_LINE_NUMBER
9692 CALL SHUTDOWN(MESSAGE) ; RETURN
9693 ENDIF
9694
9695 ! Assign SURF_INDEX, Index of the Boundary Condition
9696
9697 VT%SURF_INDEX = DEFAULT_SURF_INDEX
9698 DO NNN=0,N_SURF
9699 IF (SURF_ID==SURFACE(NNN)%ID) VT%SURF_INDEX = NNN
9700 ENDDO
9701
9702 IF (SURF_ID=='OPEN') VT%TYPE_INDICATOR = 2
9703 IF (SURF_ID=='MIRROR' .OR. SURF_ID=='PERIODIC') VT%TYPE_INDICATOR = -2
9704 IF ((MB/='null' .OR. PBX>-1.E5.EB .OR. PBY>-1.E5.EB .OR. PBZ>-1.E5.EB) .AND. SURF_ID=='OPEN') VT%TYPE_INDICATOR
9705 =-2
9706 IF (SURF_ID=='PERIODIC' .AND. WIND) VT%SURF_INDEX = PERIODIC_WIND_SURF_INDEX
9707
9708 VT%BOUNDARY_TYPE = SOLID_BOUNDARY
9709 IF (VT%SURF_INDEX==OPEN_SURF_INDEX) VT%BOUNDARY_TYPE = OPEN_BOUNDARY
9710 IF (VT%SURF_INDEX==MIRROR_SURF_INDEX) VT%BOUNDARY_TYPE = MIRROR_BOUNDARY
9711 IF (VT%SURF_INDEX==PERIODIC_SURF_INDEX) VT%BOUNDARY_TYPE = PERIODIC_BOUNDARY
9712 IF (VT%SURF_INDEX==PERIODIC_WIND_SURF_INDEX) VT%BOUNDARY_TYPE = PERIODIC_BOUNDARY
9713 IF (VT%SURF_INDEX==HVAC_SURF_INDEX) VT%BOUNDARY_TYPE = HVAC_BOUNDARY
9714
9715 VT%IOR = IOR
9716 VT%ORDINAL = NN
9717
9718 ! Activate and Deactivate logic
9719
9720 VT%ACTIVATED = .TRUE.
9721 VT%DEVC_ID = DEVC_ID
9722 VT%CTRL_ID = CTRL_ID
9723 VT%ID = ID
9724 CALL SEARCH_CONTROLLER('VENT',CTRL_ID,DEVC_ID,VT%DEVC_INDEX,VT%CTRL_INDEX,N)
9725 IF (DEVC_ID /= 'null') THEN
9726 IF (.NOT.DEVICE(VT%DEVC_INDEX)%INITIAL_STATE) VT%ACTIVATED = .FALSE.
9727 ENDIF
9728 IF (CTRL_ID /= 'null') THEN
9729 IF (.NOT.CONTROL(VT%CTRL_INDEX)%INITIAL_STATE) VT%ACTIVATED = .FALSE.
9730 ENDIF
9731
9732 IF ( (VT%BOUNDARY_TYPE==OPEN_BOUNDARY .OR. VT%BOUNDARY_TYPE==MIRROR_BOUNDARY .OR. &
9733 VT%BOUNDARY_TYPE==PERIODIC_BOUNDARY) .AND. &

```

## Source Code files for edited portions of FDS

```

9733 (VT%DEVIC.ID /= 'null' .OR. VT%CTRL.ID /= 'null' ) THEN
9734 IF (ID=='null') WRITE(MESSAGE,'(A,I0,A,I0)') 'ERROR: VENT ',NN, &
9735 ' cannot be controlled by a device, line number ',INPUT.FILE.LINE.NUMBER
9736 IF (ID/='null') WRITE(MESSAGE,'(A,A,A)') 'ERROR: VENT ',TRIM(ID),' cannot be controlled by a device'
9737 CALL SHUTDOWN(MESSAGE) ; RETURN
9738 ENDIF
9739
9740 ! Set the VENT color index
9741
9742 SELECT CASE(COLOR)
9743 CASE( 'INVISIBLE' )
9744 VT%COLOR.INDICATOR = 8
9745 TRANSPARENCY = 0..EB
9746 CASE('null')
9747 VT%COLOR.INDICATOR = 99
9748 CASE DEFAULT
9749 VT%COLOR.INDICATOR = 99
9750 CALL COLOR2RGB(RGB,COLOR)
9751 END SELECT
9752 IF (VT%COLOR.INDICATOR==8) VT%TYPE.INDICATOR = -2
9753 IF (OUTLINE) VT%TYPE.INDICATOR = 2
9754 VT%RGB = RGB
9755 VT%TRANSPARENCY = TRANSPARENCY
9756
9757 ! Parameters for specified spread of a fire over a VENT
9758
9759 VT%X0 = XYZ(1)
9760 VT%Y0 = XYZ(2)
9761 VT%Z0 = XYZ(3)
9762 VT%FIRE.SPREAD.RATE = SPREAD.RATE / TIME.SHRIK.FACTOR
9763
9764 ! Circular VENT
9765
9766 IF (RADIUS>0..EB) THEN
9767 IF (ANY(XYZ<-1.E5.EB)) THEN
9768 WRITE(MESSAGE,'(A,I0,A)') 'ERROR: VENT ',NN,' requires center point XYZ'
9769 CALL SHUTDOWN(MESSAGE) ; RETURN
9770 ENDIF
9771 VT%RADIUS = RADIUS
9772 ENDIF
9773
9774 ! Dynamic Pressure
9775
9776 VT%DYNAMIC.PRESSURE = DYNAMIC.PRESSURE
9777 IF (PRESSURE.RAMP/='null') CALL GET_RAMP_INDEX(PRESSURE.RAMP,'TIME',VT%PRESSURE.RAMP.INDEX)
9778
9779 ! Synthetic Eddy Method
9780
9781 VT%N.EDDY = N.EDDY
9782 IF (L.EDDY>TWO.EPSILON.EB) THEN
9783 VT%SIGMA.IJ = L.EDDY
9784 ELSE
9785 VT%SIGMA.IJ = L.EDDY.IJ ! Modified SEM (Jarrin , Ch. 7)
9786 VT%SIGMA.IJ = MAX(VT%SIGMA.IJ,1.E-10.EB)
9787 ENDIF
9788 IF (VEL.RMS>0..EB) THEN
9789 VT%R.IJ = 0..EB
9790 VT%R.IJ(1,1)=VEL.RMS**2
9791 VT%R.IJ(2,2)=VEL.RMS**2
9792 VT%R.IJ(3,3)=VEL.RMS**2
9793 ELSE
9794 VT%R.IJ = REYNOLDS.STRESS
9795 VT%R.IJ = MAX(VT%R.IJ,1.E-10.EB)
9796 ENDIF
9797
9798 ! Check SEM parameters
9799
9800 IF (N.EDDY>0) THEN
9801 SYNTHETIC.EDDY.METHOD = .TRUE.
9802 IF (ANY(VT%SIGMA.IJ<TWO.EPSILON.EB)) THEN
9803 WRITE(MESSAGE,'(A,I0,A)') 'ERROR: VENT ',NN,' L.EDDY = 0 in Synthetic Eddy Method'
9804 CALL SHUTDOWN(MESSAGE) ; RETURN
9805 ENDIF
9806 IF (ALL(ABS(VT%R.IJ)<TWO.EPSILON.EB)) THEN
9807 WRITE(MESSAGE,'(A,I0,A)') 'ERROR: VENT ',NN,' VEL.RMS (or Reynolds Stress) = 0 in Synthetic Eddy Method'
9808 CALL SHUTDOWN(MESSAGE) ; RETURN
9809 ENDIF
9810 IF (TRIM(SURF.ID)=='HVAC') THEN
9811 WRITE(MESSAGE,'(A,I0,A)') 'ERROR: VENT ',NN,' Synthetic Eddy Method not permitted with HVAC'
9812 CALL SHUTDOWN(MESSAGE) ; RETURN
9813 ENDIF
9814 ENDIF
9815
9816 ! Miscellaneous
9817
9818 VT%TMP.EXTERIOR = TMP.EXTERIOR + TMPM
9819 IF (VT%TMP.EXTERIOR>0..EB) TMPMIN = MIN(TMPMIN,VT%TMP.EXTERIOR)
9820 IF (TMP.EXTERIOR.RAMP/='null') CALL GET_RAMP_INDEX(TMP.EXTERIOR.RAMP,'TIME',VT%TMP.EXTERIOR.RAMP.INDEX)

```

```

9821 VT%TEXTURE(:) = TEXTURE.ORIGIN(:)
9822
9823
9824 VT%UW = UW
9825 IF (ALL(VT%UW > -1.E12.EB)) THEN
9826 VT%UW = VT%UW/SQRT(VT%UW(1)**2+VT%UW(2)**2+VT%UW(3)**2)
9827 ENDIF
9828
9829 ENDDO LMULT_LOOP
9830 ENDDO JMULT_LOOP
9831 ENDDO KMULT_LOOP
9832
9833 ENDDO READ.VENT_LOOP
9834 37 REWIND(LU.INPUT) ; INPUT_FILE.LINE_NUMBER = 0
9835
9836 ENDDO MESH_LOOP.1
9837
9838 ! Go through all the meshes again, but this time only if PROCESS(NM)==MYID
9839
9840 MESH_LOOP.2: DO NM=1,NMESHES
9841
9842 IF (PROCESS(NM)/=MYID) CYCLE MESH_LOOP.2
9843
9844 M=>MESHERS(NM)
9845 CALL POINT.TO_MESH(NM)
9846
9847 ! Get total number of vents (needed for detailed wind BC)
9848
9849 N.VENT.TOTAL = N.VENT.TOTAL + N.VENT
9850
9851 ! Check vents and assign orientations
9852
9853 VENT_LOOP.2: DO N=1,N.VENT
9854
9855 VT => VENTS(N)
9856
9857 I1 = MAX(0,VT%i1)
9858 I2 = MIN(IBAR,VT%i2)
9859 J1 = MAX(0,VT%j1)
9860 J2 = MIN(JBAR,VT%j2)
9861 K1 = MAX(0,VT%k1)
9862 K2 = MIN(KBAR,VT%k2)
9863
9864 IF (VT%IOR==0) THEN
9865 IF (I1== 0 .AND. I2==0) VT%IOR = 1
9866 IF (I1==IBAR .AND. I2==IBAR) VT%IOR = -1
9867 IF (J1== 0 .AND. J2==0) VT%IOR = 2
9868 IF (J1==JBAR .AND. J2==JBAR) VT%IOR = -2
9869 IF (K1== 0 .AND. K2==0) VT%IOR = 3
9870 IF (K1==KBAR .AND. K2==KBAR) VT%IOR = -3
9871 ENDIF
9872
9873 ORIENTATION_IF: IF (VT%IOR==0) THEN
9874 IF (I1==I2) THEN
9875 DO K=K1+1,K2
9876 DO J=J1+1,J2
9877 IF (.NOT.SOLID(CELL_INDEX(I2+1,J,K))) VT%IOR = 1
9878 IF (.NOT.SOLID(CELL_INDEX(I2 ,J,K))) VT%IOR = -1
9879 ENDDO
9880 ENDDO
9881 ENDIF
9882 IF (J1==J2) THEN
9883 DO K=K1+1,K2
9884 DO I=I1+1,I2
9885 IF (.NOT.SOLID(CELL_INDEX(I,J2+1,K))) VT%IOR = 2
9886 IF (.NOT.SOLID(CELL_INDEX(I,J2 ,K))) VT%IOR = -2
9887 ENDDO
9888 ENDDO
9889 ENDIF
9890 IF (K1==K2) THEN
9891 DO J=J1+1,J2
9892 DO I=I1+1,I2
9893 IF (.NOT.SOLID(CELL_INDEX(I,J,K2+1))) VT%IOR = 3
9894 IF (.NOT.SOLID(CELL_INDEX(I,J,K2 ))) VT%IOR = -3
9895 ENDDO
9896 ENDDO
9897 ENDIF
9898 ENDIF ORIENTATION_IF
9899
9900 IF (VT%IOR==0) THEN
9901 WRITE(MESSAGE,'(A,I0,A,I0)') 'ERROR: Specify orientation of VENT ',VT%ORDINAL, ', MESH NUMBER',NM
9902 CALL SHUTDOWN(MESSAGE) ; RETURN
9903 ENDIF
9904
9905 ! Other error messages for VENTS
9906
9907 SELECT CASE(ABS(VT%IOR))
9908 CASE(1)

```

Source Code files for edited portions of FDS

```

9909 IF (I1 >=1 .AND. I1 <=IBM1) THEN
9910 IF (VT%BOUNDARY.TYPE==OPEN.BOUNDARY.OR.VT%BOUNDARY.TYPE==MIRROR.BOUNDARY.OR.VT%BOUNDARY.TYPE==PERIODIC.BOUNDARY)
    THEN
9911 WRITE(MESSAGE,'(A,I0,A)') 'ERROR: OPEN, MIRROR, OR PERIODIC VENT ',VT%ORDINAL, ' must be an exterior boundary.'
9912 CALL SHUTDOWN(MESSAGE) ; RETURN
9913 ENDIF
9914 IF (VT%BOUNDARY.TYPE/=HVAC.BOUNDARY) VT%BOUNDARY.TYPE = SOLID.BOUNDARY
9915 IF (.NOT.SOLID(CELL_INDEX(I2+1,J2,K2)) .AND. .NOT.SOLID(CELL_INDEX(I2,J2,K2))) THEN
9916 WRITE(MESSAGE,'(A,I0,A)') 'ERROR: VENT ',VT%ORDINAL, ' must be attached to a solid obstruction'
9917 CALL SHUTDOWN(MESSAGE) ; RETURN
9918 ENDIF
9919 ENDIF
9920 CASE(2)
9921 IF (J1 >=1 .AND. J1 <=JBM1) THEN
9922 IF (VT%BOUNDARY.TYPE==OPEN.BOUNDARY.OR.VT%BOUNDARY.TYPE==MIRROR.BOUNDARY.OR.VT%BOUNDARY.TYPE==PERIODIC.BOUNDARY)
    THEN
9923 WRITE(MESSAGE,'(A,I0,A)') 'ERROR: OPEN, MIRROR, OR PERIODIC VENT ',VT%ORDINAL, ' must be an exterior boundary.'
9924 CALL SHUTDOWN(MESSAGE) ; RETURN
9925 ENDIF
9926 IF (VT%BOUNDARY.TYPE/=HVAC.BOUNDARY) VT%BOUNDARY.TYPE = SOLID.BOUNDARY
9927 IF (.NOT.SOLID(CELL_INDEX(I2,J2+1,K2)) .AND. .NOT.SOLID(CELL_INDEX(I2,J2,K2))) THEN
9928 WRITE(MESSAGE,'(A,I0,A)') 'ERROR: VENT ',VT%ORDINAL, ' must be attached to a solid obstruction'
9929 CALL SHUTDOWN(MESSAGE) ; RETURN
9930 ENDIF
9931 ENDIF
9932 CASE(3)
9933 IF (K1 >=1 .AND. K1 <=KBM1) THEN
9934 IF (VT%BOUNDARY.TYPE==OPEN.BOUNDARY.OR.VT%BOUNDARY.TYPE==MIRROR.BOUNDARY.OR.VT%BOUNDARY.TYPE==PERIODIC.BOUNDARY)
    THEN
9935 WRITE(MESSAGE,'(A,I0,A)') 'ERROR: OPEN, MIRROR, OR PERIODIC VENT ',VT%ORDINAL, ' must be an exterior boundary.'
9936 CALL SHUTDOWN(MESSAGE) ; RETURN
9937 ENDIF
9938 IF (VT%BOUNDARY.TYPE/=HVAC.BOUNDARY) VT%BOUNDARY.TYPE = SOLID.BOUNDARY
9939 IF (.NOT.SOLID(CELL_INDEX(I2,J2,K2+1)) .AND. .NOT.SOLID(CELL_INDEX(I2,J2,K2))) THEN
9940 WRITE(MESSAGE,'(A,I0,A)') 'ERROR: VENT ',VT%ORDINAL, ' must be attached to a solid obstruction'
9941 CALL SHUTDOWN(MESSAGE) ; RETURN
9942 ENDIF
9943 ENDIF
9944 END SELECT
9945
9946 ! Open up boundary cells if it is an open vent
9947
9948 IF (VT%BOUNDARY.TYPE==OPEN.BOUNDARY) THEN
9949 SELECT CASE(VT%IOR)
9950 CASE( 1)
9951 CALL BLOCK.CELL(NM, 0, 0,J1+1, J2,K1+1, K2,0,0)
9952 CASE(-1)
9953 CALL BLOCK.CELL(NM,IBP1,IBP1,J1+1, J2,K1+1, K2,0,0)
9954 CASE( 2)
9955 CALL BLOCK.CELL(NM,I1+1, I2, 0, 0,K1+1, K2,0,0)
9956 CASE(-2)
9957 CALL BLOCK.CELL(NM,I1+1, I2,JBP1,JBP1,K1+1, K2,0,0)
9958 CASE( 3)
9959 CALL BLOCK.CELL(NM,I1+1, I2,J1+1, J2, 0, 0,0,0)
9960 CASE(-3)
9961 CALL BLOCK.CELL(NM,I1+1, I2,J1+1, J2,KBP1,KBP1,0,0)
9962 END SELECT
9963 ENDIF
9964
9965 ! Check UWW
9966 IF (ABS(VT%UW(ABS(VT%IOR))) < TWO_EPSILON.EB) THEN
9967 WRITE(MESSAGE,'(A,I0,A)') 'ERROR: VENT ',VT%ORDINAL, ' cannot have normal component of UWW equal to 0'
9968 CALL SHUTDOWN(MESSAGE) ; RETURN
9969 ENDIF
9970
9971 ENDDO VENT_LOOP.2
9972
9973 ! Compute vent areas and check for passive openings
9974
9975 VENT_LOOP.3: DO N=1,N.VENT
9976
9977 VT => VENTS(N)
9978
9979 IF (VT%SURF_INDEX==HVAC.SURF_INDEX .AND. N>1) THEN
9980 DO NN=1,N-1
9981 IF (TRIM(VT%ID)==TRIM(VENTS(NN)%ID) .AND. VENTS(NN)%SURF_INDEX==HVAC.SURF_INDEX) THEN
9982 WRITE(MESSAGE,'(A,A)') 'ERROR: Two HVAC VENTS have the same ID. VENT ID: ',TRIM(VT%ID)
9983 CALL SHUTDOWN(MESSAGE) ; RETURN
9984 ENDIF
9985 ENDDO
9986 ENDIF
9987
9988 VT%FDS_AREA = 0._EB
9989 IF (VT%RADIUS>0._EB) VT%INPUT_AREA = 0._EB
9990
9991 I1 = VT%I1
9992 I2 = VT%I2
9993 J1 = VT%J1

```

```

9994 J2 = VT%J2
9995 K1 = VT%K1
9996 K2 = VT%K2
9997
9998 VT%GHOST_CELLS_ONLY = .TRUE.
9999
10000 SELECT CASE(ABS(VT%IOR))
10001 CASE(1)
10002 DO K=K1+1,K2
10003 DO J=J1+1,J2
10004 IF (J>=1 .AND. J<=JBAR .AND. K>=1 .AND. K<=KBAR) VT%GHOST_CELLS_ONLY = .FALSE.
10005 IF ( VT%RADIUS>0..EB) THEN
10006 VT%INPUT_AREA = VT%INPUT_AREA + CIRCLE_CELL_INTERSECTION_AREA(VT%Y0,VT%Z0,VT%RADIUS,Y(J-1),Y(J),Z(K-1),Z(K))
10007 IF (((YC(J)-VT%Y0)**2+(ZC(K)-VT%Z0)**2)>VT%RADIUS**2) CYCLE
10008 ENDF
10009 VT%FDS_AREA = VT%FDS_AREA + DY(J)*DZ(K)
10010 ENDDO
10011 ENDDO
10012 CASE(2)
10013 DO K=K1+1,K2
10014 DO I=I1+1,I2
10015 IF (I>=1 .AND. I<=IBAR .AND. K>=1 .AND. K<=KBAR) VT%GHOST_CELLS_ONLY = .FALSE.
10016 IF ( VT%RADIUS>0..EB) THEN
10017 VT%INPUT_AREA = VT%INPUT_AREA + CIRCLE_CELL_INTERSECTION_AREA(VT%X0,VT%Z0,VT%RADIUS,X(I-1),X(I),Z(K-1),Z(K))
10018 IF (((XC(I)-VT%X0)**2+(ZC(K)-VT%Z0)**2)>VT%RADIUS**2) CYCLE
10019 ENDF
10020 VT%FDS_AREA = VT%FDS_AREA + DX(I)*DZ(K)
10021 ENDDO
10022 ENDDO
10023 CASE(3)
10024 DO J=J1+1,J2
10025 DO I=I1+1,I2
10026 IF (I>=1 .AND. I<=IBAR .AND. J>=1 .AND. J<=JBAR) VT%GHOST_CELLS_ONLY = .FALSE.
10027 IF ( VT%RADIUS>0..EB) THEN
10028 VT%INPUT_AREA = VT%INPUT_AREA + CIRCLE_CELL_INTERSECTION_AREA(VT%X0,VT%Y0,VT%RADIUS,X(I-1),X(I),Y(J-1),Y(J))
10029 IF (((XC(I)-VT%X0)**2+(YC(J)-VT%Y0)**2)>VT%RADIUS**2) CYCLE
10030 ENDF
10031 VT%FDS_AREA = VT%FDS_AREA + DX(I)*DY(J)
10032 ENDDO
10033 ENDDO
10034 END SELECT
10035
10036 ENDDO VENT_LOOP_3
10037
10038 ENDDO MESH_LOOP_2
10039
10040 CONTAINS
10041
10042 SUBROUTINE DEFINE_EVACUATION_VENTS(NM,IMODE)
10043 !
10044 ! Define the evacuation outflow VENTS for the doors/exits.
10045 !
10046 USE EVAC, ONLY: N.DOORS, N.EXITS, N.CO.EXITS, EVAC.EMESH.EXITS.TYPE, EMESH.EXITS
10047 IMPLICIT NONE
10048 ! Passed variables
10049 INTEGER, INTENT(IN) :: NM, IMODE
10050 ! Local variables
10051 INTEGER :: N, N.END
10052
10053 N.END = N.EXITS - N.CO.EXITS + N.DOORS
10054 IMODE_1_IF: IF (IMODE==1) THEN
10055 NEND_LOOP_1: DO N = 1, N.END
10056 IF (.NOT.EMESH.EXITS(N)%DEFINE_MESH) CYCLE NEND_LOOP_1
10057 IF (EMESH.EXITS(N)%IMESH==NM .OR. EMESH.EXITS(N)%MAINMESH==NM) THEN
10058 N.VENT = N.VENT + 1
10059 EMESH.EXITS(N)%L.VENT = N.VENT
10060 EVACUATION_VENT = .TRUE.
10061 EVACUATION = .TRUE.
10062 END IF
10063 END DO NEND_LOOP_1
10064 END IF IMODE_1_IF
10065
10066 IMODE_2_IF: IF (IMODE==2) THEN
10067 ! Evacuation VENTS (for the outflow vents) need: XB, EVACUATION, RGB, MESH_ID, SURF_ID, IOR
10068 NEND_LOOP_2: DO N = 1, N.END
10069 IF (.NOT.EMESH.EXITS(N)%DEFINE_MESH) CYCLE NEND_LOOP_2
10070 IF (EMESH.EXITS(N)%L.VENT==NN .AND. (EMESH.EXITS(N)%IMESH==NM .OR. EMESH.EXITS(N)%MAINMESH==NM)) THEN
10071 EVACUATION_VENT = .TRUE.
10072 EVACUATION = .TRUE.
10073 SURF_ID = 'EVACUATION.OUTFLOW'
10074 MESH_ID = TRIM(MESHNAME(NM))
10075 XB(1) = EMESH.EXITS(N)%XB(1)
10076 XB(2) = EMESH.EXITS(N)%XB(2)
10077 XB(3) = EMESH.EXITS(N)%XB(3)
10078 XB(4) = EMESH.EXITS(N)%XB(4)
10079 XB(5) = EMESH.EXITS(N)%XB(5)
10080 XB(6) = EMESH.EXITS(N)%XB(6)
10081 RGB(:) = EMESH.EXITS(N)%RGB(:)

```



```

10082 ID = TRIM('Event.' // TRIM(MESHNAME(NM)))
10083 END IF
10084 END DO NEND_LOOP_2
10085 END IF IMODE_2_IF
10086
10087 RETURN
10088 END SUBROUTINE DEFINE_EVACUATION_VENTS
10089
10090 END SUBROUTINE READ_VENT
10091
10092
10093 SUBROUTINE READ_INIT
10094
10095 USE PHYSICAL_FUNCTIONS, ONLY: GET_SPECIFIC_GAS_CONSTANT
10096 USE COMP_FUNCTIONS, ONLY: GET_FILE_NUMBER
10097 USE DEVICE_VARIABLES, ONLY: DEVICE_TYPE, DEVICE, N_DEVC
10098 REAL (EB) :: DIAMETER, TEMPERATURE, DENSITY, RR_SUM, ZZ_GET(1:N_TRACKED_SPECIES), MASS_PER_VOLUME, &
10099 MASS_PER_TIME, DT_INSERT, UMW(3), HRRPUV, XYZ(3), DX, DY, DZ, HEIGHT, RADIUS, MASS_FRACTION(MAX_SPECIES), &
10100 PARTICLE_WEIGHT_FACTOR, AUTO_IGNITION, TEMPERATURE, VOLUME_FRACTION(MAX_SPECIES)
10101 INTEGER :: NM, NN, NNN, II, JJ, KK, NS, NS2, NUMBER_INITIAL_PARTICLES, N_PARTICLES, N_INIT_NEW, N_INIT_READ,
N_PARTICLES_PER_CELL
10102 LOGICAL :: CELL_CENTERED
10103 EQUIVALENCE(NUMBER_INITIAL_PARTICLES, N_PARTICLES)
10104 CHARACTER(LABEL_LENGTH) :: ID, CTRL_ID, DEVC_ID, PART_ID, SHAPE, MULT_ID, SPEC_ID(1:MAX_SPECIES)
10105 TYPE(INITIALIZATION_TYPE), POINTER :: IN=>NULL()
10106 TYPE(MULTIPLIER_TYPE), POINTER :: MR=>NULL()
10107 TYPE(LAGRANGIAN_PARTICLE_CLASS_TYPE), POINTER :: LPC=>NULL()
10108 TYPE(DEVICE_TYPE), POINTER :: DV
10109 NAMELIST /INIT/ AUTO_IGNITION, TEMPERATURE, CELL_CENTERED, CTRL_ID, DENSITY, DEVC_ID, DIAMETER, DT_INSERT, DX, DY, DZ, &
10110 HEIGHT, HRRPUV, ID, MASS_FRACTION, &
10111 MASS_PER_TIME, MASS_PER_VOLUME, MULT_ID, N_PARTICLES, N_PARTICLES_PER_CELL, PART_ID, PARTICLE_WEIGHT_FACTOR, &
10112 RADIUS, SHAPE, SPEC_ID, TEMPERATURE, UMW, VOLUME_FRACTION, XB, XYZ, &
10113 NUMBER_INITIAL_PARTICLES !Backwards computability
10114
10115 N_INIT = 0
10116 N_INIT_READ = 0
10117 REWIND(LU_INPUT) ; INPUT_FILE_LINE_NUMBER = 0
10118
10119 COUNT_LOOP: DO
10120 CALL CHECKREAD('INIT', LU_INPUT, IOS)
10121 IF (IOS==1) EXIT COUNT_LOOP
10122 MULT_ID = 'null'
10123 READ(LU_INPUT, NMI=INIT, END=11, ERR=12, IOSTAT=IOS)
10124 N_INIT_READ = N_INIT_READ + 1
10125 12 IF (IOS>0) THEN
10126 WRITE(MESSAGE, '(A,I0,A,I0)') 'ERROR: Problem with INIT number ', N_INIT_READ+1, ', line number',
INPUT_FILE_LINE_NUMBER
10127 CALL SHUTDOWN(MESSAGE) ; RETURN
10128 ENDIF
10129 N_INIT_NEW = 0
10130 IF (MULT_ID=='null') THEN
10131 N_INIT_NEW = 1
10132 ELSE
10133 DO N=1, N_MULT
10134 MR => MULTIPLIER(N)
10135 IF (MULT_ID==MR%ID) N_INIT_NEW = MR%N_COPIES
10136 ENDDO
10137 IF (N_INIT_NEW==0) THEN
10138 WRITE(MESSAGE, '(A,A,A,I0)') 'ERROR: MULT line ', TRIM(MULT_ID), ' not found on INIT line', N_INIT_READ
10139 CALL SHUTDOWN(MESSAGE) ; RETURN
10140 ENDIF
10141 ENDIF
10142 N_INIT = N_INIT + N_INIT_NEW
10143 ENDDO COUNT_LOOP
10144 11 REWIND(LU_INPUT) ; INPUT_FILE_LINE_NUMBER = 0
10145
10146 ! Add reserved INIT lines
10147
10148 N_INIT = N_INIT + N_INIT_RESERVED
10149
10150 ! If there are no INIT lines, return
10151
10152 IF (N_INIT==0) RETURN
10153
10154 ALLOCATE(INITIALIZATION(N_INIT), STAT=IZERO)
10155 CALL ChkMemErr('READ', 'INITIALIZATION', IZERO)
10156
10157 DO NN=1, N_INIT
10158 ALLOCATE(INITIALIZATION(NN)%MASS_FRACTION(N_TRACKED_SPECIES), STAT=IZERO)
10159 CALL ChkMemErr('READ', 'INITIALIZATION', IZERO)
10160 INITIALIZATION(NN)%MASS_FRACTION=0. _EB
10161 ENDDO
10162
10163 NN = 0
10164
10165 INIT_LOOP: DO N=1, N_INIT_READ+N_INIT_RESERVED
10166
10167 IF (N<=N_INIT_READ) THEN

```

Source Code files for edited portions of FDS

```

10168
10169 ! Read in the INIT lines
10170
10171 CALL CHECKREAD('INIT',LU,INPUT,IOS)
10172 IF (IOS==1) EXIT INIT_LOOP
10173 CALL SET_INIT_DEFAULTS
10174 READ(LU,INPUT,INIT)
10175 IF (ANY(MASS_FRACTION>0..EB) .AND. ANY(VOLUME_FRACTION>0..EB)) THEN
10176 WRITE(MESSAGE,'(A,I0..A)') 'ERROR: INIT line ', N, ". Do not specify both MASS_FRACTION and VOLUME_FRACTION."
10177 CALL SHUTDOWN(MESSAGE) ; RETURN
10178 ENDIF
10179
10180 ELSE
10181
10182 ! Use information from DEVC line to create an INIT line for 'RADIATIVE HEAT FLUX GAS' or 'ADIABATIC SURFACE
      TEMPERATURE GAS'
10183
10184 CALL SET_INIT_DEFAULTS
10185 DV => DEVICE(INIT_RESERVED(N-N_INIT_READ)%DEVC_INDEX)
10186 DX = INIT_RESERVED(N-N_INIT_READ)%DX
10187 DY = INIT_RESERVED(N-N_INIT_READ)%DY
10188 DZ = INIT_RESERVED(N-N_INIT_READ)%DZ
10189 WRITE(PART_ID,'(A)') 'RESERVED TARGET PARTICLE'
10190 N_PARTICLES = INIT_RESERVED(N-N_INIT_READ)%N_PARTICLES
10191 XB(1) = DV%X
10192 XB(2) = DV%X + (N_PARTICLES-1)*DX
10193 XB(3) = DV%Y
10194 XB(4) = DV%Y + (N_PARTICLES-1)*DY
10195 XB(5) = DV%Z
10196 XB(6) = DV%Z + (N_PARTICLES-1)*DZ
10197 ID = DV%D
10198 ENDIF
10199
10200 ! Transform XYZ into XB if necessary, and move XYZ points off of mesh boundaries.
10201
10202 IF (ANY(XYZ>-100000..EB)) THEN
10203
10204 MESH_LOOP: DO NM=1,N_MESHES
10205 IF (EVACUATION_ONLY(NM)) CYCLE MESH_LOOP
10206 M=>MESHES(NM)
10207 IF (XYZ(1)>=M%X .AND. XYZ(1)<=M%XF .AND. XYZ(2)>=M%YS .AND. XYZ(2)<=M%YF .AND. XYZ(3)>=M%ZS .AND. XYZ(3)<=M%ZF)
      THEN
10208 IF (ABS(XYZ(1)-M%X)<TWO_EPSILON.EB) XYZ(1) = XYZ(1) + 0.01.EB*M%DXI
10209 IF (ABS(XYZ(1)-M%XF)<TWO_EPSILON.EB) XYZ(1) = XYZ(1) - 0.01.EB*M%DXI
10210 IF (ABS(XYZ(2)-M%YS)<TWO_EPSILON.EB) XYZ(2) = XYZ(2) + 0.01.EB*M%DZETA
10211 IF (ABS(XYZ(2)-M%YF)<TWO_EPSILON.EB) XYZ(2) = XYZ(2) - 0.01.EB*M%DZETA
10212 IF (ABS(XYZ(3)-M%ZS)<TWO_EPSILON.EB) XYZ(3) = XYZ(3) + 0.01.EB*M%DZETA
10213 IF (ABS(XYZ(3)-M%ZF)<TWO_EPSILON.EB) XYZ(3) = XYZ(3) - 0.01.EB*M%DZETA
10214 EXIT MESH_LOOP
10215 ENDIF
10216 ENDDO MESH_LOOP
10217
10218 XB(1:2) = XYZ(1)
10219 XB(3:4) = XYZ(2)
10220 XB(5:6) = XYZ(3)
10221 ENDIF
10222
10223 ! If an offset has been specified, set the SHAPE to LINE.
10224
10225 IF (DX>0..EB .OR. DY>0..EB .OR. DZ>0..EB) SHAPE = 'LINE'
10226
10227 IF (N_PARTICLES>0 .AND. SHAPE=='LINE') THEN
10228 XB(2) = XB(1) + DX*(N_PARTICLES-1)
10229 XB(4) = XB(3) + DY*(N_PARTICLES-1)
10230 XB(6) = XB(5) + DZ*(N_PARTICLES-1)
10231 ENDIF
10232
10233 ! Create a box around a CONE
10234
10235 IF (SHAPE=='CONE' .OR. SHAPE=='RING') THEN
10236 XB(1) = XYZ(1) - RADIUS
10237 XB(2) = XYZ(1) + RADIUS
10238 XB(3) = XYZ(2) - RADIUS
10239 XB(4) = XYZ(2) + RADIUS
10240 XB(5) = XYZ(3)
10241 XB(6) = XYZ(3) + HEIGHT
10242 IF (SHAPE=='RING') XB(6) = XB(5)
10243 ENDIF
10244
10245 ! Reorder XB coordinates if necessary
10246
10247 CALL CHECK_XB(XB)
10248
10249 ! Loop over all possible multiples of the INIT
10250
10251 MR => MULTIPLIER(0)
10252 DO NNN=1,N_MULT
10253 IF (MULT_ID==MULTIPLIER(NNN)%ID) MR => MULTIPLIER(NNN)

```

Source Code files for edited portions of FDS

```

10254 ENDDO
10255
10256 NNN = 0
10257 K.MULT.LOOP: DO KK=MR%K_LOWER,MR%K_UPPER
10258 J.MULT.LOOP: DO JJ=MR%J_LOWER,MR%J_UPPER
10259 I.MULT.LOOP: DO II=MR%I_LOWER,MR%I_UPPER
10260
10261 NNN = NNN + 1 ! Counter for MULT INIT lines
10262
10263 NN = NN + 1
10264 IN => INITIALIZATION(NN)
10265
10266 ! Store the input parameters
10267
10268 IF (.NOT.MR%SEQUENTIAL) THEN
10269 IN%X1 = XB (1) + MR%DX0 + II*MR%DXB(1)
10270 IN%X2 = XB (2) + MR%DX0 + II*MR%DXB(2)
10271 IN%Y1 = XB (3) + MR%DY0 + JJ*MR%DXB(3)
10272 IN%Y2 = XB (4) + MR%DY0 + JJ*MR%DXB(4)
10273 IN%Z1 = XB (5) + MR%DZ0 + KK*MR%DXB(5)
10274 IN%Z2 = XB (6) + MR%DZ0 + KK*MR%DXB(6)
10275 ELSE
10276 IN%X1 = XB (1) + MR%DX0 + II*MR%DXB(1)
10277 IN%X2 = XB (2) + MR%DX0 + II*MR%DXB(2)
10278 IN%Y1 = XB (3) + MR%DY0 + II*MR%DXB(3)
10279 IN%Y2 = XB (4) + MR%DY0 + II*MR%DXB(4)
10280 IN%Z1 = XB (5) + MR%DZ0 + II*MR%DXB(5)
10281 IN%Z2 = XB (6) + MR%DZ0 + II*MR%DXB(6)
10282 ENDIF
10283
10284 IF (MR%N_COPIES>1) THEN
10285 WRITE(IN%D, '(A,A,I5.5)') TRIM(ID), '-', NNN
10286 ELSE
10287 IN%D = ID
10288 ENDIF
10289
10290 IN%CELL_CENTERED = CELL_CENTERED
10291 IN%DIAMETER = DIAMETER*1.E-6.EB
10292 IN%DX = DX
10293 IN%DY = DY
10294 IN%DZ = DZ
10295 IN%CTRL.ID = CTRL.ID
10296 IN%DEVC.ID = DEVC.ID
10297 CALL SEARCH.CONTROLLER('INIT',IN%CTRL.ID,IN%DEVC.ID,IN%DEVC.INDEX,IN%CTRL.INDEX,N)
10298 IN%VOLUME = (IN%X2-IN%X1)*(IN%Y2-IN%Y1)*(IN%Z2-IN%Z1)
10299 IN%TEMPERATURE = TEMPERATURE + TMPM
10300 IN%DENSITY = DENSITY
10301 IN%SHAPE = SHAPE
10302 IN%HEIGHT = HEIGHT
10303 IN%RADIUS = RADIUS
10304 IN%HRRPUV = HRRPUV*1000._EB
10305 IN%AUT = AUTO.IGNITION.TEMPERATURE + TMPM
10306 IF (HRRPUV > TWO.EPSILON.EB) INIT.HRRPUV = .TRUE.
10307 IF (DENSITY > 0._EB) RHOMAX = MAX(RHOMAX,IN%DENSITY)
10308 IF (AUTO.IGNITION.TEMPERATURE < 1.E20.EB) REIGNITION.MODEL = .TRUE.
10309
10310 SPEC_INIT_IF: IF (ANY(MASS.FRACTION > 0._EB)) THEN
10311 IF (SPEC.ID(1) == 'null') THEN
10312 WRITE(MESSAGE, '(A,I0,A,A)') 'ERROR: Problem with INIT number ', N, '. SPEC.ID must be used with MASS.FRACTION'
10313 CALL SHUTDOWN(MESSAGE) ; RETURN
10314 ENDIF
10315 DO NS=1,MAX.SPECIES
10316 IF (SPEC.ID(NS) == 'null') EXIT
10317 DO NS2=1,N.TRACKED.SPECIES
10318 IF (NS2>0 .AND. TRIM(SPEC.ID(NS)) == TRIM(SPECIES.MIXTURE(NS2)%ID)) THEN
10319 IN%MASS.FRACTION(NS2) = MASS.FRACTION(NS)
10320 EXIT
10321 ENDIF
10322 IF (NS2 == N.TRACKED.SPECIES) THEN
10323 WRITE(MESSAGE, '(A,I0,A,A,A)') 'ERROR: Problem with INIT number ', N, ' tracked species ', &
10324 TRIM(SPEC.ID(NS)), ' not found'
10325 CALL SHUTDOWN(MESSAGE) ; RETURN
10326 ENDIF
10327 ENDDO
10328 ENDDO
10329
10330 IF (SUM(IN%MASS.FRACTION) > 1._EB) THEN
10331 WRITE(MESSAGE, '(A,I0,A,A)') 'ERROR: Problem with INIT number ', N, '. Sum of specified mass fractions > 1'
10332 CALL SHUTDOWN(MESSAGE) ; RETURN
10333 ENDIF
10334 IF (IN%MASS.FRACTION(1) <= TWO.EPSILON.EB) THEN
10335 IN%MASS.FRACTION(1) = 1._EB - SUM(IN%MASS.FRACTION(2:N.TRACKED.SPECIES))
10336 ELSE
10337 WRITE(MESSAGE, '(A,I0,A,A)') 'ERROR: Problem with INIT number ', N, &
10338 '. Cannot specify background species for MASS.FRACTION'
10339 CALL SHUTDOWN(MESSAGE) ; RETURN
10340 ENDIF
10341 ZZ.GET(1:N.TRACKED.SPECIES) = IN%MASS.FRACTION(1:N.TRACKED.SPECIES)

```

Source Code files for edited portions of FDS

```

10342 CALL GET_SPECIFIC_GAS_CONSTANT (ZZ.GET,RRSUM)
10343
10344 ELSEIF (ANY(VOLUME.FRACTION>0..EB)) THEN SPEC_INIT_IF
10345 IF (SUM(VOLUME.FRACTION) > 1..EB) THEN
10346 WRITE(MESSAGE,'(A,I0,A,A)') 'ERROR: Problem with INIT number ',N,'. Sum of specified volume fractions > 1'
10347 CALL SHUTDOWN(MESSAGE) ; RETURN
10348 ENDIF
10349 IF (SPEC_ID(1)=='null') THEN
10350 WRITE(MESSAGE,'(A,I0,A,A)') 'ERROR: Problem with INIT number ',N,'. SPEC.ID must be used with VOLUME.FRACTION'
10351 CALL SHUTDOWN(MESSAGE) ; RETURN
10352 ENDIF
10353 DO NS=1,MAX_SPECIES
10354 IF (SPEC_ID(NS)=='null') EXIT
10355 DO NS2=1,N.TRACKED_SPECIES
10356 IF (NS2>0 .AND. TRIM(SPEC_ID(NS))=TRIM(SPECIES.MIXTURE(NS2)%ID)) THEN
10357 MASS_FRACTION(NS2)=VOLUME.FRACTION(NS)*SPECIES.MIXTURE(NS2)%MW
10358 EXIT
10359 ENDIF
10360 IF (NS2==N.TRACKED_SPECIES) THEN
10361 WRITE(MESSAGE,'(A,I0,A,A,A)') 'ERROR: Problem with INIT number ',N,' tracked species ',&
10362 TRIM(SPEC_ID(NS)), ' not found'
10363 CALL SHUTDOWN(MESSAGE) ; RETURN
10364 ENDIF
10365 ENDDO
10366 ENDDO
10367 IF (MASS_FRACTION(1) <=TWO.EPSILON.EB) THEN
10368 MASS_FRACTION(1) = (1..EB-SUM(VOLUME.FRACTION))*SPECIES.MIXTURE(1)%MW
10369 ELSE
10370 WRITE(MESSAGE,'(A,I0,A,A)') 'ERROR: Problem with INIT number ',N,&
10371 '. Cannot specify background species for VOLUME.FRACTION'
10372 CALL SHUTDOWN(MESSAGE) ; RETURN
10373 ENDIF
10374 MASS_FRACTION(1:N.TRACKED_SPECIES) = MASS_FRACTION(1:N.TRACKED_SPECIES)/SUM(MASS_FRACTION(1:N.TRACKED_SPECIES))
10375 IN%MASS_FRACTION(1:N.TRACKED_SPECIES) = MASS_FRACTION(1:N.TRACKED_SPECIES)
10376 ZZ.GET(1:N.TRACKED_SPECIES) = IN%MASS_FRACTION(1:N.TRACKED_SPECIES)
10377 CALL GET_SPECIFIC_GAS_CONSTANT (ZZ.GET,RRSUM)
10378
10379 ELSE SPEC_INIT_IF
10380 IN%MASS_FRACTION(1:N.TRACKED_SPECIES) = SPECIES.MIXTURE(1:N.TRACKED_SPECIES)%ZZO
10381 RR.SUM = R.SUM0
10382
10383 ENDIF SPEC_INIT_IF
10384
10385 IF (TEMPERATURE > 0..EB) TMPMIN = MIN(TMPMIN,IN%TEMPERATURE)
10386
10387 IF (IN%TEMPERATURE > 0..EB .AND. IN%DENSITY < 0..EB) THEN
10388 IN%DENSITY = P.INF/(IN%TEMPERATURE*RR.SUM)
10389 IN%ADJUST.DENSITY = .TRUE.
10390 ENDIF
10391 IF (IN%TEMPERATURE < 0..EB .AND. IN%DENSITY > 0..EB) THEN
10392 IN%TEMPERATURE = P.INF/(IN%DENSITY*RR.SUM)
10393 IN%ADJUST.TEMPERATURE = .TRUE.
10394 ENDIF
10395 IF (IN%TEMPERATURE < 0..EB .AND. IN%DENSITY < 0..EB) THEN
10396 IN%TEMPERATURE = TMPA
10397 IN%DENSITY = P.INF/(IN%TEMPERATURE*RR.SUM)
10398 IN%ADJUST.DENSITY = .TRUE.
10399 ENDIF
10400
10401 ! Special case where INIT is used to introduce a block of particles
10402
10403 IN%MASS.PER.TIME = MASS.PER.TIME
10404 IN%MASS.PER.VOLUME = MASS.PER.VOLUME
10405
10406 IF (N.PARTICLES.PER.CELL>0 .AND. N.PARTICLES>0) THEN
10407 WRITE(MESSAGE,'(A,I0,A)') 'ERROR: INIT ',N,' Cannot use both N.PARTICLES and N.PARTICLES.PER.CELL'
10408 CALL SHUTDOWN(MESSAGE) ; RETURN
10409 ENDIF
10410
10411 IN%N.PARTICLES = N.PARTICLES
10412 IN%N.PARTICLES.PER.CELL = N.PARTICLES.PER.CELL
10413 IN%PARTICLE.WEIGHT.FACTOR = PARTICLE.WEIGHT.FACTOR
10414
10415 IF ( IN%MASS.PER.VOLUME>0..EB .AND. IN%VOLUME<=TWO.EPSILON.EB) THEN
10416 WRITE(MESSAGE,'(A,I0,A)') 'ERROR: INIT ',N,' XB has no volume'
10417 CALL SHUTDOWN(MESSAGE) ; RETURN
10418 ENDIF
10419
10420 IN%DT.INSERT = DT.INSERT
10421 IF (DT.INSERT>0..EB) IN%SINGLE.INSERTION = .FALSE.
10422
10423 ! Set up a clock to keep track of particle insertions
10424
10425 ALLOCATE(IN%PARTICLE.INSERT.CLOCK (NMESHES) ,STAT=IZERO)
10426 CALL ChkMemErr('READ','PARTICLE.INSERT.CLOCK',IZERO)
10427 IN%PARTICLE.INSERT.CLOCK = T.BEGIN
10428
10429 ALLOCATE(IN%ALREADY.INSERTED (NMESHES) ,STAT=IZERO)

```

Source Code files for edited portions of FDS

```

10430 CALL ChkMemErr('READ','ALREADY_INSERTED',IZERO)
10431 IN%ALREADY_INSERTED = .FALSE.
10432
10433 ! Assign an index to identify the particle class
10434
10435 PART_ID_IF: IF (PART_ID/= 'null') THEN
10436
10437 DO NS=1,N_LAGRANGIAN_CLASSES
10438 IF (PART_ID==LAGRANGIAN.PARTICLE.CLASS(NS)%ID) THEN
10439 IN%PART_INDEX = NS
10440 PARTICLE_FILE = .TRUE.
10441 EXIT
10442 ENDF
10443 ENDDO
10444 IF (IN%PART_INDEX<1) THEN
10445 WRITE(MESSAGE,'(A,A,A)') 'ERROR: PART_ID ',TRIM(PART_ID),' does not exist'
10446 CALL SHUTDOWN(MESSAGE) ; RETURN
10447 ENDF
10448 LPC => LAGRANGIAN.PARTICLE.CLASS(IN%PART_INDEX)
10449 IN%N_PARTICLES = N_PARTICLES*MAX(1,LPC%N_ORIENTATION)
10450 IN%N_PARTICLES_PER_CELL = N_PARTICLES_PER_CELL*MAX(1,LPC%N_ORIENTATION)
10451 IF (IN%MASS_PER_TIME>0._EB .OR. IN%MASS_PER_VOLUME>0._EB) THEN
10452 IF (LPC%DENSITY < 0._EB) THEN
10453 WRITE(MESSAGE,'(A,A,A)') 'INIT ERROR: PARTicle class ',TRIM(LPC%ID),' requires a density'
10454 CALL SHUTDOWN(MESSAGE) ; RETURN
10455 ENDF
10456 ENDF
10457
10458 ! Make sure that all particles are inside of the domain
10459 IF ( LPC%PERIODIC_X .AND. (IN%X2>=XF.MAX .OR. IN%X1<=XS.MIN) ) THEN
10460 WRITE(MESSAGE,'(A,I0,A,A)') 'ERROR: Problem with INIT number ',N,'. Particle at boundary or outside of domain.'
10461 CALL SHUTDOWN(MESSAGE) ; RETURN
10462 ENDF
10463 IF ( LPC%PERIODIC_Y .AND. (IN%Y2>=YF.MAX .OR. IN%Y1<=YS.MIN) ) THEN
10464 WRITE(MESSAGE,'(A,I0,A,A)') 'ERROR: Problem with INIT number ',N,'. Particle at boundary or outside of domain.'
10465 CALL SHUTDOWN(MESSAGE) ; RETURN
10466 ENDF
10467 IF ( LPC%PERIODIC_Z .AND. (IN%Z2>=ZF.MAX .OR. IN%Z1<=ZS.MIN) ) THEN
10468 WRITE(MESSAGE,'(A,I0,A,A)') 'ERROR: Problem with INIT number ',N,'. Particle at boundary or outside of domain.'
10469 CALL SHUTDOWN(MESSAGE) ; RETURN
10470 ENDF
10471
10472 ENDF PART_ID_IF
10473
10474 ! Initial velocity components
10475
10476 IN%U0 = UW(1)
10477 IN%V0 = UW(2)
10478 IN%W0 = UW(3)
10479
10480 ENDDO I.MULT_LOOP
10481 ENDDO J.MULT_LOOP
10482 ENDDO K.MULT_LOOP
10483
10484 ENDDO INIT_LOOP
10485
10486 ! Check if there are any devices that refer to INIT lines
10487
10488 DEVICE_LOOP: DO NN=1,N_DEVC
10489 DV => DEVICE(NN)
10490 IF (DV%INIT_ID=='null') CYCLE
10491 DO I=1,N_INIT
10492 IN => INITIALIZATION(I)
10493 IF (IN%D==DV%INIT_ID) CYCLE DEVICE_LOOP
10494 ENDDO
10495 WRITE(MESSAGE,'(A,A,A)') 'ERROR: The INIT_ID for DEVC ',TRIM(DV%ID),' cannot be found.'
10496 CALL SHUTDOWN(MESSAGE) ; RETURN
10497 ENDDO DEVICE_LOOP
10498
10499 ! Rewind the input file and return
10500
10501 REWIND(LU.INPUT) ; INPUT_FILE.LINE_NUMBER = 0
10502
10503 CONTAINS
10504
10505 SUBROUTINE SET_INIT_DEFAULTS
10506
10507 ! Set default values
10508
10509 AUTO_IGNITION_TEMPERATURE = 1.E20_EB
10510 CELL_CENTERED = .FALSE.
10511 CTRL_ID = 'null'
10512 DENSITY = -1000._EB
10513 DEVC_ID = 'null'
10514 DIAMETER = -1._EB
10515 DT_INSERT = -1._EB
10516 DX = 0._EB

```

```

10518 | DY                = 0..EB
10519 | DZ                = 0..EB
10520 | HEIGHT           = -1..EB
10521 | HRRPUV           = 0..EB
10522 | ID               = 'null'
10523 | MASS.FRACTION    = 0..EB
10524 | MASS.PER.TIME    = -1..EB
10525 | MASS.PER.VOLUME  = -1..EB
10526 | MULT.ID          = 'null'
10527 | N.PARTICLES      = 0
10528 | N.PARTICLES.PER.CELL = 0
10529 | PARTICLE.WEIGHT.FACTOR = 1.0..EB
10530 | PART.ID          = 'null'
10531 | RADIUS           = -1..EB
10532 | SHAPE            = 'BLOCK'
10533 | SPEC.ID          = 'null'
10534 | TEMPERATURE      = -1000..EB
10535 | UWW              = 0..EB
10536 | VOLUME.FRACTION = 0..EB
10537 | XB(1)            = -1000000..EB
10538 | XB(2)            = 1000000..EB
10539 | XB(3)            = -1000000..EB
10540 | XB(4)            = 1000000..EB
10541 | XB(5)            = -1000000..EB
10542 | XB(6)            = 1000000..EB
10543 | XYZ              = -1000000..EB
10544 |
10545 | END SUBROUTINE SET_INIT_DEFAULTS
10546 |
10547 | END SUBROUTINE READ_INIT
10548 |
10549 |
10550 | SUBROUTINE READ_ZONE
10551 |
10552 | REAL(EB), ALLOCATABLE, DIMENSION(:) :: LEAK_AREA, LEAK_REFERENCE_PRESSURE, LEAK_PRESSURE_EXPONENT
10553 | INTEGER :: N,NM,NN,N_EVAC_ZONE,N_EVAC_MESH,NM_EVAC
10554 | LOGICAL :: SEALED, READ_ZONE_LINES, PERIODIC
10555 | CHARACTER(LABEL_LENGTH) :: ID
10556 | NAMELIST /ZONE/ ID, LEAK_AREA, LEAK_PRESSURE_EXPONENT, LEAK_REFERENCE_PRESSURE, XB, PERIODIC
10557 |
10558 | ALLOCATE (LEAK_AREA(0:MAX_LEAK_PATHS))
10559 | ALLOCATE (LEAK_REFERENCE_PRESSURE(0:MAX_LEAK_PATHS))
10560 | ALLOCATE (LEAK_PRESSURE_EXPONENT(0:MAX_LEAK_PATHS))
10561 |
10562 | N_ZONE = 0
10563 | REWIND(LU_INPUT) ; INPUT_FILE_LINE_NUMBER = 0
10564 | COUNT_ZONE_LOOP: DO
10565 | CALL CHECKREAD('ZONE', LU_INPUT, IOS)
10566 | IF (IOS==1) EXIT COUNT_ZONE_LOOP
10567 | READ(LU_INPUT, NML=ZONE, END=11, ERR=12, IOSTAT=IOS)
10568 | N_ZONE = N_ZONE + 1
10569 | 12 IF (IOS>0) THEN
10570 | WRITE(MESSAGE, '(A,I0,A,I0)') 'ERROR: Problem with ZONE number ', N_ZONE+1, ', line number', INPUT_FILE_LINE_NUMBER
10571 | CALL SHUTDOWN(MESSAGE) ; RETURN
10572 | ENDIF
10573 | ENDDO COUNT_ZONE_LOOP
10574 | 11 REWIND(LU_INPUT) ; INPUT_FILE_LINE_NUMBER = 0
10575 |
10576 | ! Check to see if there are any OPEN vents. If there are not, and there are no declared pressure ZONES, stop with
10577 | ! an error.
10578 | SEALED = .TRUE.
10579 |
10580 | N_EVAC_ZONE = 0
10581 | DO NM=1, NMESHES
10582 | IF (.NOT. EVACUATION_ONLY(NM)) THEN
10583 | M => MESHES(NM)
10584 | DO N=1, M%N_VENT
10585 | VT => M%VENTS(N)
10586 | IF (VT%BOUNDARY_TYPE/=OPEN_BOUNDARY) SEALED = .FALSE.
10587 | IF (VT%BOUNDARY_TYPE/=PERIODIC_BOUNDARY) SEALED = .FALSE.
10588 | ENDDO
10589 | ELSE
10590 | IF (EVACUATION_SKIP(NM)) N_EVAC_ZONE = N_EVAC_ZONE + 1
10591 | END IF
10592 | ENDDO
10593 |
10594 | ! If the whole domain lacks on OPEN or PERIODIC boundary, assume it to be one big pressure zone
10595 |
10596 | READ_ZONE_LINES = .TRUE.
10597 | IF (SEALED .AND. N_ZONE==0) THEN
10598 | N_ZONE = 1
10599 | READ_ZONE_LINES = .FALSE.
10600 | ENDIF
10601 |
10602 | IF (ANY(EVACUATION_SKIP)) THEN
10603 | IF (READ_ZONE_LINES) THEN
10604 | N_ZONE = N_ZONE + N_EVAC_ZONE

```

```

10605 ELSE
10606 N_ZONE = N_EVAC_ZONE
10607 ENDIF
10608 END IF
10609
10610 ! Make sure that there are no leak paths to undefined pressure ZONES
10611
10612 DO N=0,N_SURF
10613 SF => SURFACE(N)
10614 IF (SP%LEAK_PATH(1)>(N_ZONE-N_EVAC_ZONE) .OR. SP%LEAK_PATH(2)>(N_ZONE-N_EVAC_ZONE)) SP%LEAK_PATH = -1
10615 ENDDO
10616
10617 ! Allocate array to indicate if pressure ZONES are connected
10618
10619 ALLOCATE(CONNECTED_ZONES(0:N_ZONE,0:N_ZONE,N_MESHES),STAT=IZERO)
10620 CALL ChkMemErr('READ','CONNECTED_ZONES',IZERO)
10621 CONNECTED_ZONES = .FALSE.
10622
10623 ! If there are no ZONE lines, return
10624
10625 IF (N_ZONE==0) RETURN
10626
10627 ! Allocate ZONE arrays
10628
10629 ALLOCATE(P_ZONE(N_ZONE),STAT=IZERO)
10630 CALL ChkMemErr('READ','P_ZONE',IZERO)
10631
10632 ! Read in and process ZONE lines
10633
10634 READ_ZONE_LOOP: DO N=1,N_ZONE-N_EVAC_ZONE
10635
10636 ALLOCATE(P_ZONE(N)%LEAK_AREA(0:N_ZONE),STAT=IZERO)
10637 CALL ChkMemErr('READ','LEAK_AREA',IZERO)
10638 ALLOCATE(P_ZONE(N)%LEAK_PRESSURE_EXPONENT(0:N_ZONE),STAT=IZERO)
10639 CALL ChkMemErr('READ','LEAK_PRESSURE_EXPONENT',IZERO)
10640 ALLOCATE(P_ZONE(N)%LEAK_REFERENCE_PRESSURE(0:N_ZONE),STAT=IZERO)
10641 CALL ChkMemErr('READ','LEAK_REFERENCE_PRESSURE',IZERO)
10642
10643 IF (N<1000) WRITE(ID,'(A,I3)') 'ZONE.',N
10644 IF (N<100) WRITE(ID,'(A,I2)') 'ZONE.',N
10645 IF (N<10) WRITE(ID,'(A,I1)') 'ZONE.',N
10646 LEAK_AREA = 0._EB
10647 LEAK_REFERENCE_PRESSURE = 4._EB
10648 LEAK_PRESSURE_EXPONENT = 0.5._EB
10649 XB(1) = -1000000._EB
10650 XB(2) = 1000000._EB
10651 XB(3) = -1000000._EB
10652 XB(4) = 1000000._EB
10653 XB(5) = -1000000._EB
10654 XB(6) = 1000000._EB
10655 PERIODIC = .FALSE.
10656
10657 IF (READ_ZONE_LINES) THEN
10658 CALL CHECKREAD('ZONE',LU_INPUT,IOS)
10659 IF (IOS==1) EXIT READ_ZONE_LOOP
10660 READ(LU_INPUT,ZONE)
10661 ENDIF
10662
10663 CALL CHECK_XB(XB)
10664
10665 P_ZONE(N)%ID = ID
10666 P_ZONE(N)%LEAK_AREA(0:N_ZONE) = LEAK_AREA(0:N_ZONE)
10667 P_ZONE(N)%LEAK_REFERENCE_PRESSURE(0:N_ZONE) = LEAK_REFERENCE_PRESSURE(0:N_ZONE)
10668 P_ZONE(N)%LEAK_PRESSURE_EXPONENT(0:N_ZONE) = LEAK_PRESSURE_EXPONENT(0:N_ZONE)
10669 P_ZONE(N)%X1 = XB(1)
10670 P_ZONE(N)%X2 = XB(2)
10671 P_ZONE(N)%Y1 = XB(3)
10672 P_ZONE(N)%Y2 = XB(4)
10673 P_ZONE(N)%Z1 = XB(5)
10674 P_ZONE(N)%Z2 = XB(6)
10675 P_ZONE(N)%EVACUATION = .FALSE.
10676 P_ZONE(N)%PERIODIC = PERIODIC
10677 IF (N > 1) THEN
10678 DO NN = 1,N-1
10679 IF (P_ZONE(NN)%LEAK_AREA(N) > 0._EB) THEN
10680 IF (P_ZONE(N)%LEAK_AREA(NN) > 0._EB) THEN
10681 WRITE(MESSAGE,'(A,I0,A,I0)') 'ERROR: LEAK_AREA specified twice for ZONE ',N,' and ',NN
10682 CALL SHUTDOWN(MESSAGE); RETURN
10683 ELSE
10684 P_ZONE(N)%LEAK_AREA(NN) = P_ZONE(NN)%LEAK_AREA(N)
10685 P_ZONE(N)%LEAK_REFERENCE_PRESSURE(NN) = P_ZONE(NN)%LEAK_REFERENCE_PRESSURE(N)
10686 P_ZONE(N)%LEAK_PRESSURE_EXPONENT(NN) = P_ZONE(NN)%LEAK_PRESSURE_EXPONENT(N)
10687 ENDIF
10688 ENDIF
10689 ENDDO
10690 ENDIF
10691
10692 ENDDO READ_ZONE_LOOP

```

```

10693
10694 READ.EVACUATION_ZONE_LOOP: DO N=N_ZONE-N.EVAC_ZONE+1,N_ZONE
10695 ALLOCATE(P_ZONE(N)%LEAK_AREA(0:N_ZONE),STAT=IZERO)
10696 CALL ChkMemErr('READ','LEAK_AREA',IZERO)
10697
10698 WRITE(ID,'(A,I2.2)') 'ZONE',N
10699 LEAK_AREA = 0._EB
10700 N.EVAC_MESH = 0
10701 NM.EVAC = 0
10702 EVAC_MESH_NUMBER: DO NM=1,NMESHES
10703 IF (EVACUATION_SKIP(NM)) THEN
10704 N.EVAC_MESH = N.EVAC_MESH + 1
10705 NM.EVAC = NM
10706 IF (N.EVAC_MESH == N-(N_ZONE-N.EVAC_ZONE)) EXIT EVAC_MESH_NUMBER
10707 END IF
10708 ENDDO EVAC_MESH_NUMBER
10709
10710 XB(1) = 0.5._EB*(MESHES(NM.EVAC)%XS+MESHES(NM.EVAC)%XF)
10711 XB(2) = 0.5._EB*(MESHES(NM.EVAC)%XS+MESHES(NM.EVAC)%XF)
10712 XB(3) = 0.5._EB*(MESHES(NM.EVAC)%YS+MESHES(NM.EVAC)%YF)
10713 XB(4) = 0.5._EB*(MESHES(NM.EVAC)%YS+MESHES(NM.EVAC)%YF)
10714 XB(5) = MESHES(NM.EVAC)%ZS
10715 XB(6) = MESHES(NM.EVAC)%ZF
10716
10717 P_ZONE(N)%ID = ID
10718 P_ZONE(N)%ID = TRIM('EvacPzone.') // TRIM(MESH_NAME(NM.EVAC))
10719 P_ZONE(N)%LEAK_AREA(0:N_ZONE) = LEAK_AREA(0:N_ZONE)
10720 P_ZONE(N)%X1 = XB(1)-0.5._EB
10721 P_ZONE(N)%X2 = XB(2)+0.5._EB
10722 P_ZONE(N)%Y1 = XB(3)-0.5._EB
10723 P_ZONE(N)%Y2 = XB(4)+0.5._EB
10724 P_ZONE(N)%Z1 = XB(5)
10725 P_ZONE(N)%Z2 = XB(6)
10726 P_ZONE(N)%EVACUATION = .TRUE.
10727 P_ZONE(N)%MESH_INDEX = NM.EVAC
10728
10729 ENDDO READ.EVACUATION_ZONE_LOOP
10730
10731 REWIND(LU_INPUT) ; INPUT_FILE_LINE_NUMBER = 0
10732
10733 DEALLOCATE (LEAK_AREA)
10734
10735 END SUBROUTINE READ_ZONE
10736
10737
10738 SUBROUTINE READ_DEVC
10739
10740 ! Just read in the DEViCes and the store the info in DEVICE()
10741
10742 USE DEVICE_VARIABLES, ONLY: DEVICE_TYPE, DEVICE, N.DEVC, N.DEVC.TIME, N.DEVC.LINE, MAX.DEVC.LINE.POINTS,
10743 DEVC.PIPE.OPERATING
10744 INTEGER :: NN,NM,MESH_NUMBER,N.DEVC.READ,IOR,TRIP_DIRECTION,VELO_INDEX,POINTS,I.POINT,PIPE_INDEX,
10745 ORIENTATION_INDEX, &
10746 ORIENTATION_NUMBER, STATISTICS.LOCATION_INDEX, GHOST_CELL_IOR(3)
10747 REAL(EB) :: DEPTH,ORIENTATION(3),ROTATION,SETPOINT,FLOWRATE,BYPASS_FLOWRATE,DELAY,XYZ(3),CONVERSION_FACTOR,
10748 SMOOTHING_FACTOR,&
10749 OR_TEMP(3),QUANTITY_RANGE(2),STATISTICS.START,COORD_FACTOR
10750 CHARACTER(LABEL_LENGTH) :: QUANTITY,QUANTITY2,PROP_ID,CTRL_ID,DEVC_ID,INIT_ID,SURF_ID,STATISTICS,PART_ID,MATL_ID,
10751 SPEC_ID,UNITS, &
10752 DUCT_ID,NODE_ID(2),R_ID,X_ID,Y_ID,Z_ID,NO.UPDATE.DEVC_ID,NO.UPDATE.CTRL_ID,REAC_ID
10753 LOGICAL :: INITIAL_STATE,LATCH,DRY,TIME_AVERAGED,EVACUATION,HIDE.COORDINATES,RELATIVE,OUTPUT,
10754 NEW_ORIENTATION.VECTOR,TIME.HISTORY,&
10755 LINE.DEVICE
10756 TYPE (DEVICE_TYPE), POINTER :: DV=>NULL()
10757 NAMELIST /DEVC/ BYPASS_FLOWRATE,CONVERSION_FACTOR,COORD_FACTOR,CTRL_ID,DELAY,DEPTH,DEVC_ID,DRY,DUCT_ID,EVACUATION
10758 ,FLOWRATE,FYI,&
10759 GHOST_CELL_IOR,HIDE.COORDINATES,ID,INITIAL_STATE,INIT_ID,IOR,LATCH,MATL_ID,NODE_ID, &
10760 NO.UPDATE.DEVC_ID,NO.UPDATE.CTRL_ID,ORIENTATION,ORIENTATION_NUMBER,OUTPUT,PART_ID,PIPE_INDEX,POINTS,&
10761 PROP_ID,QUANTITY,QUANTITY2,QUANTITY_RANGE,&
10762 REAC_ID,RELATIVE,R_ID,ROTATION,SETPOINT,SMOOTHING_FACTOR,SPEC_ID,STATISTICS,STATISTICS.START,SURF_ID,&
10763 TIME.AVERAGED,TIME.HISTORY,TRIP_DIRECTION,UNITS,VELO_INDEX,XB,XYZ,X_ID,Y_ID,Z_ID
10764
10765 ! Read the input file and count the number of DEVC lines
10766
10767 N.DEVC = 0
10768 N.DEVC.READ = 0
10769 N.DEVC.TIME = 0
10770 N.DEVC.LINE = 0
10771
10772 REWIND(LU_INPUT) ; INPUT_FILE_LINE_NUMBER = 0
10773 COUNT.DEVC_LOOP: DO
10774 CALL CHECKREAD('DEVC',LU_INPUT,IOS)
10775 IF (IOS==1) EXIT COUNT.DEVC_LOOP
10776 POINTS = 1
10777 TIME.HISTORY = .FALSE.
10778 READ(LU_INPUT,NML=DEVC,END=11,ERR=12,Iostat=IOS)
10779 N.DEVC = N.DEVC + POINTS
10780 N.DEVC.READ = N.DEVC.READ + 1

```



## Source Code files for edited portions of FDS

```

10775 IF (POINTS>1 .AND. .NOT.TIME.HISTORY) MAX.DEVC.LINE.POINTS = MAX(MAX.DEVC.LINE.POINTS,POINTS)
10776 12 IF (IOS>0) THEN
10777 WRITE(MESSAGE,'(A,I0,A,I0)') 'ERROR: Problem with DEVC number ',N.DEVC.READ+1,', line number',
      INPUT.FILE.LINE.NUMBER
10778 CALL SHUTDOWN(MESSAGE) ; RETURN
10779 ENDIF
10780 ENDDO COUNT.DEVC.LOOP
10781 11 REWIND(LU.INPUT) ; INPUT.FILE.LINE.NUMBER = 0
10782
10783 IF (N.DEVC==0) RETURN
10784
10785 ! Allocate DEVICE array to hold all information for each device
10786
10787 ALLOCATE(DEVICE(N.DEVC),STAT=IZERO) ; CALL ChkMemErr('READ','DEVICE',IZERO)
10788
10789 ! Speeical case for QUANTITY='RADIATIVE HEAT FLUX GAS' or 'ADIABATIC SURFACE TEMPERATURE GAS'
10790
10791 ALLOCATE(INIT.RESERVED(N.DEVC),STAT=IZERO) ; CALL ChkMemErr('READ','INIT.RESERVED',IZERO)
10792 N.INIT.RESERVED = 0
10793
10794 ! Read in the DEVC lines, keeping track of TIME-history devices, and LINE array devices
10795
10796 N.DEVC = 0
10797
10798 READ.DEVC.LOOP: DO NN=1,N.DEVC.READ
10799
10800 CALL CHECKREAD('DEVC',LU.INPUT,IOS)
10801 IF (IOS==1) EXIT READ.DEVC.LOOP
10802 CALL SET.DEVC.DEFAULTS
10803 READ(LU.INPUT,DEVC)
10804
10805 IF (QUANTITY.RANGE(2) <= QUANTITY.RANGE(1)) THEN
10806 WRITE(MESSAGE,'(A,A,A)') 'ERROR: DEVC ',TRIM(ID),' has QUANTITY.RANGE(2) <= QUANTITY.RANGE(1)'
10807 CALL SHUTDOWN(MESSAGE) ; RETURN
10808 ENDIF
10809
10810 ! Determine if the device is a steady-state "line" device or the usual time-history device.
10811
10812 LINE.DEVICE = .FALSE.
10813 IF (POINTS>1 .AND. .NOT.TIME.HISTORY) LINE.DEVICE = .TRUE.
10814
10815 ! Parse 'MAXLOC X', etc.
10816
10817 IF (STATISTICS(1:6)== 'MAXLOC' .OR. STATISTICS(1:6)== 'MINLOC') THEN
10818 IF (STATISTICS(8:8)== 'X') STATISTICS.LOCATION.INDEX = 1
10819 IF (STATISTICS(8:8)== 'Y') STATISTICS.LOCATION.INDEX = 2
10820 IF (STATISTICS(8:8)== 'Z') STATISTICS.LOCATION.INDEX = 3
10821 IF (STATISTICS(1:6)== 'MAXLOC') STATISTICS = 'MAX'
10822 IF (STATISTICS(1:6)== 'MINLOC') STATISTICS = 'MIN'
10823 UNITS = 'm'
10824 ENDIF
10825
10826 ! Error statement involving POINTS
10827
10828 IF (POINTS>1 .AND. ANY(XB<-1.E5.EB) .AND. INIT.ID=='null') THEN
10829 WRITE(MESSAGE,'(A,A,A)') 'ERROR: DEVC ',TRIM(ID),' must have coordinates given in terms of XB'
10830 CALL SHUTDOWN(MESSAGE) ; RETURN
10831 ENDIF
10832 IF (LINE.DEVICE .AND. TRIM(STATISTICS)/='null' .AND. &
10833 TRIM(STATISTICS)/='RMS' .AND. &
10834 TRIM(STATISTICS)/='COV' .AND. &
10835 TRIM(STATISTICS)/='CORRcoef' .AND. &
10836 TRIM(STATISTICS)/='TIME MIN' .AND. &
10837 TRIM(STATISTICS)/='TIME MAX') THEN
10838 WRITE(MESSAGE,'(A,A,A)') 'ERROR: DEVC ',TRIM(ID),' cannot use a steady-state line device and STATISTICS at the
      same time'
10839 CALL SHUTDOWN(MESSAGE) ; RETURN
10840 ENDIF
10841 IF (.NOT.LINE.DEVICE .AND. (TRIM(STATISTICS)=='TIME MIN' .OR. &
10842 TRIM(STATISTICS)=='TIME MAX')) THEN
10843 WRITE(MESSAGE,'(A,A,A)') 'ERROR: DEVC ',TRIM(ID),' cannot compute STATISTICS. Set POINTS>1.'
10844 CALL SHUTDOWN(MESSAGE) ; RETURN
10845 ENDIF
10846
10847 ! Make ORIENTATION consistent with IOR
10848
10849 SELECT CASE(IOR)
10850 CASE( 1) ; ORIENTATION=(/ 1..EB, 0..EB, 0..EB/)
10851 CASE(-1) ; ORIENTATION=(/ -1..EB, 0..EB, 0..EB/)
10852 CASE( 2) ; ORIENTATION=(/ 0..EB, 1..EB, 0..EB/)
10853 CASE(-2) ; ORIENTATION=(/ 0..EB, -1..EB, 0..EB/)
10854 CASE( 3) ; ORIENTATION=(/ 0..EB, 0..EB, 1..EB/)
10855 CASE(-3) ; ORIENTATION=(/ 0..EB, 0..EB, -1..EB/)
10856 END SELECT
10857
10858 ! Add ORIENTATION to global list
10859
10860 NEW.ORIENTATION.VECTOR = .TRUE.

```

```

10861 ORIENTATION_INDEX = 0
10862
10863 DO I=1,N_ORIENTATION_VECTOR
10864 IF (ORIENTATION(1)==ORIENTATION_VECTOR(1,I) .AND. &
10865 ORIENTATION(2)==ORIENTATION_VECTOR(2,I) .AND. &
10866 ORIENTATION(3)==ORIENTATION_VECTOR(3,I)) THEN
10867 NEW_ORIENTATION_VECTOR = .FALSE.
10868 ORIENTATION_INDEX = I
10869 EXIT
10870 ENDIF
10871 ENDDO
10872
10873 IF (NEW_ORIENTATION_VECTOR) THEN
10874 OR_TEMP(1:3) = ORIENTATION(1:3)
10875 N_ORIENTATION_VECTOR = N_ORIENTATION_VECTOR + 1
10876 IF (N_ORIENTATION_VECTOR<UBOUND(ORIENTATION_VECTOR,DIM=2)) THEN
10877 ORIENTATION_VECTOR => REALLOCATED2D(ORIENTATION_VECTOR,1,3,1,N_ORIENTATION_VECTOR+10)
10878 ENDIF
10879 IF (ALL(ABS(OR_TEMP(1:3))<TWO_EPSILON_EB)) THEN
10880 WRITE(MESSAGE,'(A,A,A)') 'ERROR: DEVC ',TRIM(ID),' All components of ORIENTATION are zero.'
10881 CALL SHUTDOWN(MESSAGE) ; RETURN
10882 ENDIF
10883 ORIENTATION_VECTOR(1:3,N_ORIENTATION_VECTOR) = ORIENTATION(1:3) / NORM2(OR_TEMP)
10884 ORIENTATION_INDEX = N_ORIENTATION_VECTOR
10885 ENDIF
10886
10887 ! Check if there are any devices with specified XB that do not fall within a mesh.
10888
10889 IF (POINTS==1 .AND. XB(1)>-1.E5_EB) THEN
10890
10891 IF (QUANTITY/= 'PATH OBSCURATION' .AND. QUANTITY/= 'TRANSMISSION') CALL CHECK_XB(XB)
10892
10893 BAD = .TRUE.
10894 CHECK_MESH_LOOP: DO NM=1,N_MESHES
10895 IF (EVACUATION_ONLY(NM)) CYCLE CHECK_MESH_LOOP
10896 M=>MESHES(NM)
10897 IF (XB(1)>=M%XS .AND. XB(2)<=M%XF .AND. XB(3)>=M%YS .AND. XB(4)<=M%YF .AND. XB(5)>=M%ZS .AND. XB(6)<=M%ZF) THEN
10898 BAD = .FALSE.
10899 EXIT CHECK_MESH_LOOP
10900 ENDIF
10901 ENDDO CHECK_MESH_LOOP
10902
10903 IF (BAD .AND. .NOT.ALL(EVACUATION_ONLY)) THEN
10904 WRITE(MESSAGE,'(A,A,A)') 'ERROR: XB for DEVC ',TRIM(ID),' must be completely within a mesh.'
10905 CALL SHUTDOWN(MESSAGE) ; RETURN
10906 ENDIF
10907
10908 ENDIF
10909
10910 ! Process the point devices along a line, if necessary
10911
10912 POINTS_LOOP: DO I_POINT=1,POINTS
10913
10914 IF (XB(1)>-1.E5_EB) THEN
10915 IF (TRIM(QUANTITY)='VELOCITY PATCH') THEN
10916 IF (XYZ(1) <-1.E5_EB) THEN
10917 XYZ(1) = XB(1) + (XB(2)-XB(1))/2._EB
10918 XYZ(2) = XB(3) + (XB(4)-XB(3))/2._EB
10919 XYZ(3) = XB(5) + (XB(6)-XB(5))/2._EB
10920 ENDIF
10921 ELSE
10922 IF (POINTS > 1) THEN
10923 XYZ(1) = XB(1) + (XB(2)-XB(1))*REAL(I_POINT-1,EB)/REAL(MAX(POINTS-1,1),EB)
10924 XYZ(2) = XB(3) + (XB(4)-XB(3))*REAL(I_POINT-1,EB)/REAL(MAX(POINTS-1,1),EB)
10925 XYZ(3) = XB(5) + (XB(6)-XB(5))*REAL(I_POINT-1,EB)/REAL(MAX(POINTS-1,1),EB)
10926 ELSE
10927 XYZ(1) = XB(1) + (XB(2)-XB(1))/2._EB
10928 XYZ(2) = XB(3) + (XB(4)-XB(3))/2._EB
10929 XYZ(3) = XB(5) + (XB(6)-XB(5))/2._EB
10930 ENDIF
10931 ENDIF
10932 ELSE
10933 IF (XYZ(1) < -1.E5_EB .AND. DUCT_ID=='null' .AND. NODE_ID(1)=='null' .AND. INIT_ID=='null') THEN
10934 WRITE(MESSAGE,'(A,A,A)') 'ERROR: DEVC ',TRIM(ID),' must have coordinates, even if it is not a point quantity'
10935 CALL SHUTDOWN(MESSAGE) ; RETURN
10936 ENDIF
10937 ENDIF
10938
10939 ! Determine which mesh the device is in
10940
10941 BAD = .TRUE.
10942 MESH_LOOP: DO NM=1,N_MESHES
10943 IF (EVACUATION_ONLY(NM)) CYCLE MESH_LOOP
10944 M=>MESHES(NM)
10945 IF (XYZ(1)>=M%XS .AND. XYZ(1)<=M%XF .AND. XYZ(2)>=M%YS .AND. XYZ(2)<=M%YF .AND. XYZ(3)>=M%ZS .AND. XYZ(3)<=M%ZF)
10946 THEN
10947 IF (ABS(XYZ(1)-M%XS)<TWO_EPSILON_EB) XYZ(1) = XYZ(1) + 0.01_EB*M%DXI
10948 IF (ABS(XYZ(1)-M%XF)<TWO_EPSILON_EB) XYZ(1) = XYZ(1) - 0.01_EB*M%DXI

```

Source Code files for edited portions of FDS

```

10948 IF (ABS(XYZ(2)-M%YS)<TWO_EPSILON.EB) XYZ(2) = XYZ(2) + 0.01.EB*M%DETA
10949 IF (ABS(XYZ(2)-M%YF)<TWO_EPSILON.EB) XYZ(2) = XYZ(2) - 0.01.EB*M%DETA
10950 IF (ABS(XYZ(3)-M%ZS)<TWO_EPSILON.EB) XYZ(3) = XYZ(3) + 0.01.EB*M%DZETA
10951 IF (ABS(XYZ(3)-M%ZF)<TWO_EPSILON.EB) XYZ(3) = XYZ(3) - 0.01.EB*M%DZETA
10952 MESHNUMBER = NM
10953 BAD = .FALSE.
10954 EXIT MESHLOOP
10955 ENDIF
10956 ENDDO MESHLOOP
10957
10958 ! Process EVAC meshes
10959
10960 EVACUATION.MESHLOOP: DO NM=1,NMESHERS
10961 IF (.NOT.EVACUATION_SKIP(NM)) CYCLE EVACUATION.MESHLOOP
10962 M=>MESHERS(NM)
10963 IF (XYZ(1)>=M%XS .AND. XYZ(1)<=M%XF .AND. XYZ(2)>=M%YS .AND. XYZ(2)<=M%YF .AND. XYZ(3)>=M%ZS .AND. XYZ(3)<=M%ZF)
10964 THEN
10965 IF (BAD) MESHNUMBER = NM
10966 IF (.NOT.BAD .AND. EVACUATION .AND. QUANTITY=='TIME' .AND. SETPOINT<=T_BEGIN) THEN
10967 MESHNUMBER = NM
10968 BAD = .FALSE.
10969 END IF
10970 EXIT EVACUATION.MESHLOOP
10971 ENDIF
10972 ENDDO EVACUATION.MESHLOOP
10973
10974 ! Make sure there is either a QUANTITY or PROP.ID for the DEVICE
10975
10976 IF (QUANTITY=='null' .AND. PROP.ID=='null') THEN
10977 WRITE(MESSAGE, '(A,A,A)') 'ERROR: DEVC ',TRIM(ID), ' must have either an output QUANTITY or PROP.ID'
10978 CALL SHUTDOWN(MESSAGE) ; RETURN
10979 ENDIF
10980
10981 IF (BAD) THEN
10982 IF (DUCT.ID/'null' .OR. NODE.ID(1)/='null' .OR. INIT.ID/'null') THEN
10983 XYZ(1) = MESHERS(1)%XS
10984 XYZ(2) = MESHERS(1)%YS
10985 XYZ(3) = MESHERS(1)%ZS
10986 MESHNUMBER = 1
10987 ELSE
10988 IF (ALL(EVACUATION_ONLY)) CYCLE READ.DEVC_LOOP
10989 WRITE(MESSAGE, '(A,A,A)') 'WARNING: DEVC ',TRIM(ID), ' is not within any mesh.'
10990 IF (MYID=0) WRITE(LU_ERR, '(A)') TRIM(MESSAGE)
10991 CYCLE READ.DEVC_LOOP
10992 ENDIF
10993 ENDIF
10994
10995 ! Don't print out clocks
10996
10997 IF (QUANTITY=='TIME' .AND. NO.UPDATE.DEVC.ID=='null' .AND. NO.UPDATE.CTRL.ID=='null') OUTPUT = .FALSE.
10998
10999 ! Determine if the DEVC is a TIME or LINE device
11000
11001 IF (.NOT.LINE.DEVICE .AND. OUTPUT) N.DEVC.TIME = N.DEVC.TIME + 1
11002 IF (LINE.DEVICE .AND. L.POINT==1) N.DEVC.LINE = N.DEVC.LINE + 1
11003
11004 ! Assign properties to the DEVICE array
11005
11006 N.DEVC = N.DEVC + 1
11007 DV => DEVICE(N.DEVC)
11008
11009 DV%RELATIVE = RELATIVE
11010 DV%CONVERSION_FACTOR = CONVERSION_FACTOR
11011 DV%COORD_FACTOR = COORD_FACTOR
11012 DV%DEPTH = DEPTH
11013 DV%IOR = IOR
11014 IF (POINTS>1 .AND. POINTS<=99 .AND. .NOT.LINE.DEVICE) THEN
11015 WRITE(DV%ID, '(A,A,I2.2)') TRIM(ID), '-',L.POINT
11016 ELSEIF (POINTS>99 .AND. POINTS<=999 .AND. .NOT.LINE.DEVICE) THEN
11017 WRITE(DV%ID, '(A,A,I3.3)') TRIM(ID), '-',L.POINT
11018 ELSEIF (POINTS>999 .AND. .NOT.LINE.DEVICE) THEN
11019 WRITE(DV%ID, '(A,A,I6.6)') TRIM(ID), '-',L.POINT
11020 ELSE
11021 DV%ID = ID
11022 ENDIF
11023 IF (LINE.DEVICE) DV%LINE = N.DEVC.LINE
11024 DV%POINT = L.POINT
11025 DV%MESH = MESHNUMBER
11026 DV%ORDINAL = NN
11027 DV%ORIENTATION_INDEX = ORIENTATION_INDEX
11028 DV%PROP.ID = PROP.ID
11029 DV%DEVC.ID = DEVC.ID
11030 DV%CTRL.ID = CTRL.ID
11031 DV%SURF.ID = SURF.ID
11032 DV%PART.ID = PART.ID
11033 DV%MATL.ID = MATL.ID
11034 DV%SPEC.ID = SPEC.ID

```

Source Code files for edited portions of FDS

```

11035 DV%DUCT.ID           = DUCT.ID
11036 DV%INIT.ID          = INIT.ID
11037 DV%NODE.ID         = NODE.ID
11038 DV%REAC.ID         = REAC.ID
11039 DV%QUANTITY         = QUANTITY
11040 DV%QUANTITY2        = QUANTITY2
11041 DV%ROTATION         = ROTATION*TWOP/360..EB
11042 DV%SETPOINT        = SETPOINT
11043 DV%LATCH            = LATCH
11044 DV%OUTPUT           = OUTPUT
11045 DV%ORIENTATION.NUMBER = ORIENTATION.NUMBER
11046 DV%TRIP.DIRECTION  = TRIP.DIRECTION
11047 DV%INITIAL.STATE   = INITIAL.STATE
11048 DV%CURRENT.STATE    = INITIAL.STATE
11049 DV%PRIOR.STATE      = INITIAL.STATE
11050 DV%FLOWRATE         = FLOWRATE
11051 DV%BYPASS.FLOWRATE = BYPASS.FLOWRATE
11052 DV%SMOOTHING.FACTOR = SMOOTHING.FACTOR
11053 DV%STATISTICS       = STATISTICS
11054 DV%STATISTICS.LOCATION.INDEX = STATISTICS.LOCATION.INDEX
11055 DV%STATISTICS.START = STATISTICS.START
11056 DV%TIME.AVERAGED   = TIME.AVERAGED
11057 DV%SURF.INDEX      = 0
11058 DV%UNITS            = UNITS
11059 DV%DELAY            = DELAY / TIME.SHRIK.FACTOR
11060 DV%X1               = XB(1)
11061 DV%X2               = XB(2)
11062 DV%Y1               = XB(3)
11063 DV%Y2               = XB(4)
11064 DV%Z1               = XB(5)
11065 DV%Z2               = XB(6)
11066 DV%X                 = XYZ(1)
11067 DV%Y                 = XYZ(2)
11068 DV%Z                 = XYZ(3)
11069 IF (X.ID=='null') X.ID = TRIM(ID)//'-x'
11070 IF (Y.ID=='null') Y.ID = TRIM(ID)//'-y'
11071 IF (Z.ID=='null') Z.ID = TRIM(ID)//'-z'
11072 DV%R.ID             = R.ID
11073 DV%X.ID             = X.ID
11074 DV%Y.ID             = Y.ID
11075 DV%Z.ID             = Z.ID
11076 DV%DRY              = DRY
11077 DV%EVACUATION       = EVACUATION
11078 DV%VELO.INDEX       = VELO.INDEX
11079 DV%PIPE.INDEX       = PIPE.INDEX
11080 DV%NO.UPDATE.DEVC.ID = NO.UPDATE.DEVC.ID
11081 DV%NO.UPDATE.CTRL.ID = NO.UPDATE.CTRL.ID
11082 DV%QUANTITY.RANGE   = QUANTITY.RANGE
11083 DV%GHOST.CELL.IOR   = GHOST.CELL.IOR
11084
11085 IF (LINE.DEVICE) THEN
11086 IF (.NOT.HIDE.COORDINATES) THEN
11087 IF (ABS(XB(1)-XB(2))> SPACING(XB(2)) .AND. ABS(XB(3)-XB(4))<=SPACING(XB(4)) .AND. &
11088 ABS(XB(5)-XB(6))<=SPACING(XB(6))) DV%LINE.COORD.CODE = 1
11089 IF (ABS(XB(1)-XB(2))<=SPACING(XB(2)) .AND. ABS(XB(3)-XB(4))> SPACING(XB(4)) .AND. &
11090 ABS(XB(5)-XB(6))<=SPACING(XB(6))) DV%LINE.COORD.CODE = 2
11091 IF (ABS(XB(1)-XB(2))<=SPACING(XB(2)) .AND. ABS(XB(3)-XB(4))<=SPACING(XB(4)) .AND. &
11092 ABS(XB(5)-XB(6))> SPACING(XB(6))) DV%LINE.COORD.CODE = 3
11093 IF (ABS(XB(1)-XB(2))> SPACING(XB(2)) .AND. ABS(XB(3)-XB(4))> SPACING(XB(4)) .AND. &
11094 ABS(XB(5)-XB(6))<=SPACING(XB(6))) DV%LINE.COORD.CODE = 12
11095 IF (ABS(XB(1)-XB(2))> SPACING(XB(2)) .AND. ABS(XB(3)-XB(4))<=SPACING(XB(4)) .AND. &
11096 ABS(XB(5)-XB(6))> SPACING(XB(6))) DV%LINE.COORD.CODE = 13
11097 IF (ABS(XB(1)-XB(2))<=SPACING(XB(2)) .AND. ABS(XB(3)-XB(4))> SPACING(XB(4)) .AND. &
11098 ABS(XB(5)-XB(6))> SPACING(XB(6))) DV%LINE.COORD.CODE = 23
11099 IF (DV%R.ID/'null') DV%LINE.COORD.CODE = 4 ! Special case where radial coordinates are requested
11100 ELSE
11101 DV%LINE.COORD.CODE = 0
11102 ENDF
11103 ENDF
11104
11105 ! Special case for QUANTITY='RADIATIVE HEAT FLUX GAS' or 'ADIABATIC SURFACE TEMPERATURE GAS'.
11106 ! Save information to create INIT line.
11107
11108 IF (DV%QUANTITY=='RADIATIVE HEAT FLUX GAS' .OR. &
11109 DV%QUANTITY=='RADIANCE' .OR. &
11110 DV%QUANTITY=='ADIABATIC SURFACE TEMPERATURE GAS') THEN
11111 DV%INIT.ID = DV%ID
11112 TARGET.PARTICLES.INCLUDED = .TRUE.
11113 IF (DV%POINT==1) THEN
11114 N_INIT.RESERVED = N_INIT.RESERVED + 1
11115 INIT.RESERVED(N_INIT.RESERVED)%DEVC.INDEX = N.DEVC
11116 INIT.RESERVED(N_INIT.RESERVED)%N.PARTICLES = POINTS
11117 IF (POINTS>1) THEN
11118 INIT.RESERVED(N_INIT.RESERVED)%XYZ(1) = XB(1)
11119 INIT.RESERVED(N_INIT.RESERVED)%XYZ(2) = XB(3)
11120 INIT.RESERVED(N_INIT.RESERVED)%XYZ(3) = XB(5)
11121 INIT.RESERVED(N_INIT.RESERVED)%DX = (XB(2)-XB(1))/(REAL(POINTS,EB) -1)
11122 INIT.RESERVED(N_INIT.RESERVED)%DY = (XB(4)-XB(3))/(REAL(POINTS,EB) -1)

```

Source Code files for edited portions of FDS

```

11123 INIT_RESERVED(N_INIT_RESERVED)%dZ = (XB(6)-XB(5))/(REAL(POINTS,EB)-1)
11124 ENDIF
11125 ENDIF
11126 ENDIF
11127
11128 ENDDO POINTS_LOOP
11129
11130 ! Coordinates for non-point devices
11131
11132 IF ((XB(1)>-1.E5_EB.OR.STATISTICS/='null') .AND. POINTS==1) THEN
11133 NM = DV%MESH
11134 M=>MESSES(NM)
11135 XB(1) = MAX(XB(1),M%XS)
11136 XB(2) = MIN(XB(2),M%XF)
11137 XB(3) = MAX(XB(3),M%YS)
11138 XB(4) = MIN(XB(4),M%YF)
11139 XB(5) = MAX(XB(5),M%ZS)
11140 XB(6) = MIN(XB(6),M%ZF)
11141 DV%X1 = XB(1)
11142 DV%X2 = XB(2)
11143 DV%Y1 = XB(3)
11144 DV%Y2 = XB(4)
11145 DV%Z1 = XB(5)
11146 DV%Z2 = XB(6)
11147 DV%I1 = NINT( GINV(XB(1)-M%XS,1,NM)*M%RDXI )
11148 DV%I2 = NINT( GINV(XB(2)-M%XF,1,NM)*M%RDXI )
11149 DV%J1 = NINT( GINV(XB(3)-M%YS,2,NM)*M%RDZETA )
11150 DV%J2 = NINT( GINV(XB(4)-M%YF,2,NM)*M%RDZETA )
11151 DV%K1 = NINT( GINV(XB(5)-M%ZS,3,NM)*M%RDZETA )
11152 DV%K2 = NINT( GINV(XB(6)-M%ZF,3,NM)*M%RDZETA )
11153 IF (DV%I1<DV%I2) DV%I1 = DV%I1 + 1
11154 IF (DV%J1<DV%J2) DV%J1 = DV%J1 + 1
11155 IF (DV%K1<DV%K2) DV%K1 = DV%K1 + 1
11156 IF (ABS(XB(1)-XB(2))<=SPACING(XB(2)) .AND. IOR==0) DV%IOR_ASSUMED = 1
11157 IF (ABS(XB(3)-XB(4))<=SPACING(XB(4)) .AND. IOR==0) DV%IOR_ASSUMED = 2
11158 IF (ABS(XB(5)-XB(6))<=SPACING(XB(6)) .AND. IOR==0) DV%IOR_ASSUMED = 3
11159 ENDIF
11160
11161 IF (TRIM(DV%QUANTITY) == 'CHEMISTRY SUBITERATIONS') OUTPUT_CHEM_IT = .TRUE.
11162
11163 IF (TRIM(DV%QUANTITY) == 'REAC SOURCE TERM' .OR. TRIM(DV%QUANTITY) == 'HRRPUV REAC') REAC_SOURCE_CHECK=.TRUE.
11164
11165 IF (TRIM(QUANTITY)=='SOLID CELL Q-S') STORE_Q_DOT_PPP_S = .TRUE.
11166
11167 IF (TRIM(QUANTITY)=='DUDT' .OR. TRIM(QUANTITY)=='DVDI' .OR. TRIM(QUANTITY)=='DWDI') STORE_OLD_VELOCITY=.TRUE.
11168
11169 ENDDO READ_DEVC_LOOP
11170
11171 ALLOCATE (DEVC_PIPE_OPERATING(MAXVAL(DEVICE%PIPE_INDEX)))
11172 DEVC_PIPE_OPERATING = 0
11173
11174 REWIND(LU_INPUT) ; INPUT_FILE_LINE_NUMBER = 0
11175
11176 CONTAINS
11177
11178 SUBROUTINE SET_DEVC_DEFAULTS
11179
11180 RELATIVE = .FALSE.
11181 CONVERSION_FACTOR = 1._EB
11182 COORD_FACTOR = 1._EB
11183 DEPTH = 0._EB
11184 IOR = 0
11185 ID = 'null'
11186 ORIENTATION(1:3) = (/0._EB,0._EB,-1._EB/)
11187 PROP_ID = 'null'
11188 CTRL_ID = 'null'
11189 DEVC_ID = 'null'
11190 SURF_ID = 'null'
11191 PART_ID = 'null'
11192 MATL_ID = 'null'
11193 SPEC_ID = 'null'
11194 DUCT_ID = 'null'
11195 INIT_ID = 'null'
11196 NODE_ID = 'null'
11197 REAC_ID = 'null'
11198 FLOWRATE = 0._EB
11199 DELAY = 0._EB
11200 BYPASS_FLOWRATE = 0._EB
11201 QUANTITY = 'null'
11202 QUANTITY2 = 'null'
11203 ROTATION = 0._EB
11204 XB(1) = -1.E6_EB
11205 XB(2) = 1.E6_EB
11206 XB(3) = -1.E6_EB
11207 XB(4) = 1.E6_EB
11208 XB(5) = -1.E6_EB
11209 XB(6) = 1.E6_EB
11210 INITIAL_STATE = .FALSE.

```

```

11211 LATCH = .TRUE.
11212 OUTPUT = .TRUE.
11213 ORIENTATION_NUMBER = 1
11214 POINTS = 1
11215 SETPOINT = 1.E20.EB
11216 SMOOTHING_FACTOR = 0.EB
11217 STATISTICS = 'null'
11218 STATISTICS_START = -1.E20.EB
11219 TRIP_DIRECTION = 1
11220 TIME_AVERAGED = .TRUE.
11221 TIME_HISTORY = .FALSE.
11222 UNITS = 'null'
11223 VELO_INDEX = 0
11224 XYZ = -1.E6.EB
11225 R_ID = 'null'
11226 X_ID = 'null'
11227 Y_ID = 'null'
11228 Z_ID = 'null'
11229 HIDE_COORDINATES = .FALSE.
11230 DRY = .FALSE.
11231 EVACUATION = .FALSE.
11232 PIPE_INDEX = 1
11233 NO_UPDATE_DEVC_ID = 'null'
11234 NO_UPDATE_CTRL_ID = 'null'
11235 QUANTITY_RANGE(1) = -1.E50.EB
11236 QUANTITY_RANGE(2) = 1.E50.EB
11237 STATISTICS_LOCATION_INDEX = 0
11238 GHOST_CELL_IOR(1:3) = (/0,0,0/)
11239
11240 END SUBROUTINE SET_DEVC_DEFAULTS
11241
11242 END SUBROUTINE READ_DEVC
11243
11244
11245 SUBROUTINE READ_CTRL
11246
11247 ! Just read in the ConTRoL parameters and store in the array CONTROL
11248
11249 USE CONTROL_VARIABLES
11250 USE MATH_FUNCTIONS, ONLY : GET_RAMP_INDEX
11251
11252 LOGICAL :: INITIAL_STATE, LATCH, EVACUATION
11253 INTEGER :: CYCLES, N, NC, TRIP_DIRECTION
11254 REAL (EB) :: SETPOINT(2), DELAY, CYCLE_TIME_CONSTANT, PROPORTIONAL_GAIN, INTEGRAL_GAIN, DIFFERENTIAL_GAIN,
TARGET_VALUE
11255 CHARACTER (LABEL_LENGTH) :: ID, FUNCTION_TYPE, INPUT_ID(40), RAMP_ID, ON_BOUND
11256 TYPE (CONTROL_TYPE), POINTER :: CF=>NULL()
11257 NAMELIST /CTRL/ CONSTANT, CYCLES, CYCLE_TIME, DELAY, DIFFERENTIAL_GAIN, EVACUATION, FUNCTION_TYPE, ID, INITIAL_STATE,
INTEGRAL_GAIN, &
INPUT_ID, LATCH, N, ON_BOUND, PROPORTIONAL_GAIN, RAMP_ID, &
SETPOINT, TARGET_VALUE, TRIP_DIRECTION
11258
11259 N_CTRL = 0
11260 REWIND(LU_INPUT) ; INPUT_FILE_LINE_NUMBER = 0
11261 COUNT_CTRL_LOOP: DO
11262 CALL CHECKREAD('CTRL', LU_INPUT, IOS)
11263 IF (IOS==1) EXIT COUNT_CTRL_LOOP
11264 READ(LU_INPUT, NML=CTRL, END=11, ERR=12, IOSTAT=IOS)
11265 N_CTRL = N_CTRL + 1
11266 12 IF (IOS > 0) THEN
11267 WRITE(MESSAGE, '(A,I0,A,I0)') 'ERROR: Problem with CTRL number ', N_CTRL+1, ', line number', INPUT_FILE_LINE_NUMBER
11268 CALL SHUTDOWN(MESSAGE) ; RETURN
11269 ENDF
11270 ENDDO COUNT_CTRL_LOOP
11271 11 REWIND(LU_INPUT) ; INPUT_FILE_LINE_NUMBER = 0
11272
11273 IF (N_CTRL==0) RETURN
11274
11275 ! Allocate CONTROL array and set initial values of all to 0
11276
11277 ALLOCATE(CONTROL(N_CTRL), STAT=IZERO)
11278 CALL ChkMemErr('READ', 'CONTROL', IZERO)
11279
11280 ! Read in the CTRL lines
11281
11282 READ_CTRL_LOOP: DO NC=1, N_CTRL
11283
11284 CALL CHECKREAD('CTRL', LU_INPUT, IOS)
11285 IF (IOS==1) EXIT READ_CTRL_LOOP
11286 CALL SET_CTRL_DEFAULTS
11287 READ(LU_INPUT, CTRL)
11288
11289 ! Make sure there is either a FUNCTION_TYPE type for the CTRL
11290
11291 IF (FUNCTION_TYPE=='null') THEN
11292 WRITE(MESSAGE, '(A,I0,A)') 'ERROR: CTRL ', NC, ' must have a FUNCTION_TYPE'
11293 CALL SHUTDOWN(MESSAGE) ; RETURN
11294 ENDF
11295
11296

```

```

11297
11298 ! Assign properties to the CONTROL array
11299
11300 CF => CONTROL(NC)
11301 CP%CONSTANT = CONSTANT
11302 CP%ID = ID
11303 CP%LATCH = LATCH
11304 CP%INITIAL_STATE = INITIAL_STATE
11305 CP%CURRENT_STATE = INITIAL_STATE
11306 CP%PRIOR_STATE = INITIAL_STATE
11307 CP%SETPOINT = SETPOINT
11308 CP%DELAY = DELAY / TIME_SHRINK_FACTOR
11309 CP%CYCLE_TIME = CYCLE_TIME
11310 CP%CYCLES = CYCLES
11311 CP%RAMP_ID = RAMP_ID
11312 CP%N = N
11313 CP%INPUT_ID = INPUT_ID
11314 CP%EVACUATION = EVACUATION
11315 CP%TRIP_DIRECTION = TRIP_DIRECTION
11316 CP%PROPORTIONAL_GAIN = PROPORTIONAL_GAIN
11317 CP%INTEGRAL_GAIN = INTEGRAL_GAIN
11318 CP%DIFFERENTIAL_GAIN = DIFFERENTIAL_GAIN
11319 CP%TARGET_VALUE = TARGET_VALUE
11320 IF (ONBOUND=='UPPER') THEN
11321 CP%ONBOUND = 1
11322 ELSE
11323 CP%ONBOUND = -1
11324 ENDIF
11325 !Assign control index
11326 SELECT CASE(FUNCTION_TYPE)
11327 CASE('ALL')
11328 CP%CONTROL_INDEX = AND_GATE
11329 CASE('ANY')
11330 CP%CONTROL_INDEX = OR_GATE
11331 CASE('ONLY')
11332 CP%CONTROL_INDEX = XOR_GATE
11333 CASE('AT_LEAST')
11334 CP%CONTROL_INDEX = X_OF_N_GATE
11335 CASE('TIME_DELAY')
11336 CP%CONTROL_INDEX = TIME_DELAY
11337 CASE('DEADBAND')
11338 CP%CONTROL_INDEX = DEADBAND
11339 CASE('CYCLING')
11340 CP%CONTROL_INDEX = CYCLING
11341 CASE('CUSTOM')
11342 CP%CONTROL_INDEX = CUSTOM
11343 CALL GET_RAMP_INDEX(RAMP_ID, 'CONTROL', CP%RAMP_INDEX)
11344 CP%LATCH = .FALSE.
11345 CASE('KILL')
11346 CP%CONTROL_INDEX = KILL
11347 CASE('RESTART')
11348 CP%CONTROL_INDEX = CORE_DUMP
11349 CASE('SUM')
11350 CP%CONTROL_INDEX = CF_SUM
11351 CASE('SUBTRACT')
11352 CP%CONTROL_INDEX = CF_SUBTRACT
11353 CASE('MULTIPLY')
11354 CP%CONTROL_INDEX = CF_MULTIPLY
11355 CASE('DIVIDE')
11356 CP%CONTROL_INDEX = CF_DIVIDE
11357 CASE('POWER')
11358 CP%CONTROL_INDEX = CF_POWER
11359 CASE('EXP')
11360 CP%CONTROL_INDEX = CF_EXP
11361 CASE('LOG')
11362 CP%CONTROL_INDEX = CF_LOG
11363 CASE('SIN')
11364 CP%CONTROL_INDEX = CF_SIN
11365 CASE('COS')
11366 CP%CONTROL_INDEX = CF_COS
11367 CASE('ASIN')
11368 CP%CONTROL_INDEX = CF_ASIN
11369 CASE('ACOS')
11370 CP%CONTROL_INDEX = CF_ACOS
11371 CASE('PID')
11372 CP%CONTROL_INDEX = CF_PID
11373 IF (CP%TARGET_VALUE < -1.E30.EB) THEN
11374 WRITE(MESSAGE, '(A,I0,A)') 'ERROR: CTRL ', NC, ', PID controller must be given a TARGET.VALUE'
11375 CALL SHUTDOWN(MESSAGE) ; RETURN
11376 ENDIF
11377 CASE DEFAULT
11378 WRITE(MESSAGE, '(A,I0,A)') 'ERROR: CTRL ', NC, ' FUNCTION_TYPE not recognized'
11379 CALL SHUTDOWN(MESSAGE) ; RETURN
11380 END SELECT
11381
11382 ENDDO READ_CTRL_LOOP
11383 REWIND(LU_INPUT) ; INPUT_FILE_LINE_NUMBER = 0
11384

```

```

11385 CONTAINS
11386
11387 SUBROUTINE SET_CTRL_DEFAULTS
11388 CONSTANT = -9.E30_EB
11389 ID = 'null'
11390 LATCH = .TRUE.
11391 INITIAL_STATE = .FALSE.
11392 SETPOINT = 1.E30_EB
11393 DELAY = 0._EB
11394 CYCLE_TIME = 1000000._EB
11395 CYCLES = 1
11396 FUNCTION_TYPE = 'null'
11397 RAMP_ID = 'null'
11398 INPUT_ID = 'null'
11399 ONBOUND = 'LOWER'
11400 N = 1
11401 EVACUATION = .FALSE.
11402 TRIP_DIRECTION = 1
11403 PROPORTIONAL_GAIN = 1._EB
11404 INTEGRAL_GAIN = 0._EB
11405 DIFFERENTIAL_GAIN = 0._EB
11406 TARGET_VALUE = 0._EB
11407
11408 END SUBROUTINE SET_CTRL_DEFAULTS
11409
11410 END SUBROUTINE READ_CTRL
11411
11412
11413 SUBROUTINE PROC_CTRL
11414
11415 ! Process the CONTROL function parameters
11416
11417 USE CONTROL_VARIABLES
11418 USE DEVICE_VARIABLES, ONLY: N_DEVC, DEVICE
11419 LOGICAL :: CONSTANT_SPECIFIED, TSP_WARNING=.FALSE.
11420 INTEGER :: NC, NN, NNN
11421 TYPE (CONTROL_TYPE), POINTER :: CF=>NULL()
11422
11423 PROC_CTRL_LOOP: DO NC = 1, N_CTRL
11424
11425 CF => CONTROL(NC)
11426 CONSTANT_SPECIFIED = .FALSE.
11427 IF (CP%CONTROL_INDEX== TIME_DELAY) TSP_WARNING=.TRUE.
11428 ! setup input array
11429
11430 CP%N_INPUTS = 0
11431 INPUT_COUNT: DO NN=1,40
11432 IF (CP%INPUT_ID(NN)=='null') EXIT INPUT_COUNT
11433 END DO INPUT_COUNT
11434 CP%N_INPUTS=NN-1
11435 IF (CP%N_INPUTS==0) THEN
11436 WRITE(MESSAGE,'(A,I0,A)') 'ERROR: CTRL ',NC,' must have at least one input'
11437 CALL SHUTDOWN(MESSAGE) ; RETURN
11438 ENDIF
11439 SELECT CASE (CP%CONTROL_INDEX)
11440 CASE (CF_SUBTRACT,CF_DIVIDE,CF_POWER)
11441 IF (CP%N_INPUTS /= 2) THEN
11442 WRITE(MESSAGE,'(A,I0,A)') 'ERROR: CTRL ',NC,' must have at only two inputs'
11443 CALL SHUTDOWN(MESSAGE) ; RETURN
11444 ENDIF
11445 CASE (CF_SUM,CF_MULTIPLY)
11446 CASE DEFAULT
11447 IF (ANY(CP%INPUT_ID=='CONSTANT')) THEN
11448 WRITE(MESSAGE,'(A,I0,A)') 'ERROR: CTRL ',NC,' the INPUT_ID of CONSTANT cannot be used'
11449 CALL SHUTDOWN(MESSAGE) ; RETURN
11450 ENDIF
11451 END SELECT
11452 ALLOCATE (CP%INPUT(CP%N_INPUTS),STAT=IZERO)
11453 CALL ChkMemErr('READ',CP%INPUT,IZERO)
11454 ALLOCATE (CP%INPUT_TYPE(CP%N_INPUTS),STAT=IZERO)
11455 CALL ChkMemErr('READ',CP%INPUT_TYPE,IZERO)
11456 CP%INPUT_TYPE = -1
11457
11458 BUILD_INPUT: DO NN = 1, CP%N_INPUTS
11459 IF (TRIM(CP%INPUT_ID(NN))/=TRIM(CP%ID)) THEN
11460 WRITE(MESSAGE,'(A,I0,A)') 'ERROR: CTRL ',NC,' cannot use a control function as an input to itself'
11461 CALL SHUTDOWN(MESSAGE) ; RETURN
11462 ENDIF
11463 IF (CP%INPUT_ID(NN)=='CONSTANT') THEN
11464 IF (CONSTANT_SPECIFIED) THEN
11465 WRITE(MESSAGE,'(A,I0,A)') 'ERROR: CTRL ',NC,' can only specify one input as a constant value'
11466 CALL SHUTDOWN(MESSAGE) ; RETURN
11467 ENDIF
11468 IF (CP%CONSTANT < -8.E30_EB) THEN
11469 WRITE(MESSAGE,'(A,I0,A)') 'ERROR: CTRL ',NC,' has the INPUT_ID CONSTANT but no constant value was specified'
11470 CALL SHUTDOWN(MESSAGE) ; RETURN
11471 ENDIF
11472 CP%INPUT_TYPE(NN) = CONSTANT_INPUT

```



Source Code files for edited portions of FDS

```

11473 CONSTANT_SPECIFIED = .TRUE.
11474 CYCLE BUILD.INPUT
11475 ENDIF
11476 CTRL_LOOP: DO NNN = 1, N_CTRL
11477 IF (CONTROL(NNN)%ID == CP%INPUT_ID(NN)) THEN
11478 CP%INPUT(NN) = NNN
11479 CP%INPUT_TYPE(NN) = CONTROL.INPUT
11480 IF (CP%CONTROL_INDEX == CUSTOM) THEN
11481 WRITE(MESSAGE, '(A,I0,A)') 'ERROR: CUSTOM CTRL ',NC,' cannot have another CTRL as input'
11482 CALL SHUTDOWN(MESSAGE) ; RETURN
11483 ENDIF
11484 EXIT CTRL_LOOP
11485 ENDIF
11486 END DO CTRL_LOOP
11487 DEVC_LOOP: DO NNN = 1, N_DEVC
11488 IF (DEVICE(NNN)%ID == CP%INPUT_ID(NN)) THEN
11489 IF (DEVICE(NNN)%OUTPUT_INDEX==41 .OR. DEVICE(NNN)%OUTPUT2_INDEX==41) TSF_WARNING=.TRUE.
11490 IF (CP%INPUT_TYPE(NN) > 0) THEN
11491 WRITE(MESSAGE, '(A,I0,A,I0,A)') 'ERROR: CTRL ',NC,' input ',NN,' is the ID for both a DEVC and a CTRL'
11492 CALL SHUTDOWN(MESSAGE) ; RETURN
11493 ENDIF
11494 CP%INPUT(NN) = NNN
11495 CP%INPUT_TYPE(NN) = DEVICE.INPUT
11496 EXIT DEVC_LOOP
11497 ENDIF
11498 END DO DEVC_LOOP
11499 IF (CP%INPUT_TYPE(NN) > 0) CYCLE BUILD.INPUT
11500 WRITE(MESSAGE, '(A,I0,A,A)') 'ERROR: CTRL ',NC,' cannot locate item for input ', TRIM(CP%INPUT_ID(NN))
11501 CALL SHUTDOWN(MESSAGE) ; RETURN
11502 IF (ALL(EVACUATION_ONLY)) CYCLE BUILD.INPUT
11503 END DO BUILD.INPUT
11504
11505 END DO PROC_CTRL_LOOP
11506
11507 IF (ABS(TIME_SHRINK_FACTOR-1..EB)>TWO_EPSILON_EB .AND. TSF_WARNING) THEN
11508 IF (MYID==0) WRITE(LU_ERR, '(A)') 'WARNING: One or more time based CTRL functions are being used with
    TIME_SHRINK_FACTOR'
11509 ENDIF
11510
11511 END SUBROUTINE PROC_CTRL
11512
11513
11514 SUBROUTINE PROC_OBST
11515
11516 USE GEOMETRY_FUNCTIONS, ONLY: BLOCK_CELL
11517 INTEGER :: NM,N,I,J,K,IS,JS,KS,IC1,IC2
11518
11519 MESH_LOOP: DO NM=1,NMESHES
11520
11521 IF (PROCESS(NM)/=MYID) CYCLE MESH_LOOP
11522 IF (EVACUATION_ONLY(NM)) CYCLE MESH_LOOP
11523
11524 M=>MESHES(NM)
11525 CALL POINT_TO_MESH(NM)
11526
11527 ! Assign a property index to the obstruction for use in Smokeview
11528
11529 DO N=1,N_OBST
11530 OB=>OBSTRUCTION(N)
11531 IF (OB%PROP_ID /= 'null') THEN
11532 CALL GET_PROPERTY_INDEX(OB%PROP_INDEX, 'OBST', OB%PROP_ID)
11533 ENDIF
11534 ENDDO
11535
11536 ! Make mesh edge cells not solid if cells on either side are not solid
11537
11538 DO K=0,KBP1,KBP1
11539 IF (K==0) THEN ; KS=1 ; ELSE ; KS=-1 ; ENDIF
11540 DO J=0,JBP1,JBP1
11541 IF (J==0) THEN ; JS=1 ; ELSE ; JS=-1 ; ENDIF
11542 DO I=1,IBAR
11543 IC1 = CELL_INDEX(I,J+JS,K) ; IC2 = CELL_INDEX(I,J,K+KS)
11544 IF (.NOT.SOLID(IC1) .AND. .NOT.SOLID(IC2)) CALL BLOCK_CELL(NM,I,I,J,J,K,K,0,0)
11545 ENDDO
11546 ENDDO
11547 ENDDO
11548
11549 DO K=0,KBP1,KBP1
11550 IF (K==0) THEN ; KS=1 ; ELSE ; KS=-1 ; ENDIF
11551 DO I=0,IBP1,IBP1
11552 IF (I==0) THEN ; IS=1 ; ELSE ; IS=-1 ; ENDIF
11553 DO J=1,JBAR
11554 IC1 = CELL_INDEX(I+IS,J,K) ; IC2 = CELL_INDEX(I,J,K+KS)
11555 IF (.NOT.SOLID(IC1) .AND. .NOT.SOLID(IC2)) CALL BLOCK_CELL(NM,I,I,J,J,K,K,0,0)
11556 ENDDO
11557 ENDDO
11558 ENDDO
11559

```

Source Code files for edited portions of FDS

```

11560 DO J=0,JB1,JB1
11561 IF (J==0) THEN ; JS=1 ; ELSE ; JS=-1 ; ENDIF
11562 DO I=0,IB1,IB1
11563 IF (I==0) THEN ; IS=1 ; ELSE ; IS=-1 ; ENDIF
11564 DO K=1,KBAR
11565 IC1 = CELL.INDEX(I+IS,J,K) ; IC2 = CELL.INDEX(I,J+JS,K)
11566 IF (.NOT.SOLID(IC1) .AND. .NOT.SOLID(IC2)) CALL BLOCK.CELL(NM,I,I,J,J,K,K,0,0)
11567 ENDDO
11568 ENDDO
11569 ENDDO
11570
11571 ENDDO MESHLOOP
11572
11573
11574 END SUBROUTINE PROC.OBST
11575
11576 SUBROUTINE PROC.DEVC(DT)
11577
11578 ! Process the DEViCes
11579
11580 USE COMP.FUNCTIONS, ONLY : CHANGE.UNITS
11581 USE CONTROL.VARIABLES
11582 USE DEVICE.VARIABLES, ONLY : DEVICE.TYPE, DEVICE, N.DEVC, PROPERTY, PROPERTY.TYPE, MAX.PDPA.HISTOGRAM.NBINS,
11583 N.PDPA.HISTOGRAM
11584
11585 REAL(EB), INTENT(IN) :: DT
11586 INTEGER :: N,NN,NNN,NM,QUANTITY.INDEX,QUANTITY2.INDEX,MAXCELLS,I,J,K
11587 REAL(EB) :: XX,YY,ZZ,XX1,YY1,ZZ1,DISTANCE,SCANDISTANCE,DX,DY,DZ
11588 TYPE (DEVICE.TYPE), POINTER :: DV=>NULL()
11589
11590 IF (N.DEVC==0) RETURN
11591
11592 ! Set initial values for DEViCes
11593
11594 DEVICE(1:N.DEVC)%VALUE = 0._EB
11595 DEVICE(1:N.DEVC)%TIME.INTERVAL = 0._EB
11596
11597 PROC.DEVC.LOOP: DO N=1,N.DEVC
11598
11599 DV => DEVICE(N)
11600
11601 ! Check for HVAC outputs with no HVAC inputs
11602
11603 IF ((DV%DUCT.ID/'null' .OR. DV%NODE.ID(1)/='null') .AND. .NOT. HVAC.SOLVE) THEN
11604 WRITE(MESSAGE,'(A)') 'ERROR: HVAC outputs specified with no HVAC inputs'
11605 CALL SHUTDOWN(MESSAGE) ; RETURN
11606 ENDIF
11607
11608 ! If the Device has a SURF.ID, get the SURF.INDEX
11609
11610 IF (DV%SURF.ID/'null') THEN
11611 DO NN=1,N.SURF
11612 IF (SURFACE(NN)%ID==DV%SURF.ID) DV%SURF.INDEX = NN
11613 ENDDO
11614 ENDIF
11615
11616 ! Check if the device PROPERTY exists and is appropriate
11617
11618 DV%PROP.INDEX = 0
11619 IF (DV%PROP.ID /='null') THEN
11620 CALL GET_PROPERTY.INDEX(DV%PROP.INDEX,'DEVC',DV%PROP.ID)
11621 IF (DV%QUANTITY=='null' .AND. PROPERTY(DV%PROP.INDEX)%QUANTITY=='null') THEN
11622 WRITE(MESSAGE,'(5A)') 'ERROR: DEVC ',TRIM(DV%ID),' or DEVC PROPerTy ',TRIM(DV%PROP.ID),' must have a QUANTITY'
11623 CALL SHUTDOWN(MESSAGE) ; RETURN
11624 ENDIF
11625 IF (DV%QUANTITY=='null' .AND. PROPERTY(DV%PROP.INDEX)%QUANTITY/='null') DV%QUANTITY = PROPERTY(DV%PROP.INDEX)%
11626 QUANTITY
11627 ENDIF
11628
11629 ! Check if the output QUANTITY exists and is appropriate
11630
11631 QUANTITY.IF: IF (DV%QUANTITY /='null') THEN
11632 CALL GET_QUANTITY.INDEX(DV%SMOKEVIEW.LABEL,DV%SMOKEVIEW.BAR.LABEL,QUANTITY.INDEX,LDUM(1), &
11633 DV%Y.INDEX,DV%Z.INDEX,DV%PART.INDEX,DV%DUCT.INDEX,DV%NODE.INDEX(1),DV%REAC.INDEX,'DEVC', &
11634 DV%QUANTITY,'null',DV%SPEC.ID,DV%PART.ID,DV%DUCT.ID,DV%NODE.ID(1),DV%REAC.ID)
11635
11636 IF (DV%QUANTITY=='CONTROL' .OR. DV%QUANTITY=='CONTROL VALUE') UPDATE.DEVICES.AGAIN = .TRUE.
11637
11638 IF (OUTPUT.QUANTITY(QUANTITY.INDEX)%INTEGRATED .AND. DV%X1<=-1.E6.EB) THEN
11639 WRITE(MESSAGE,'(3A)') 'ERROR: DEVC QUANTITY ',TRIM(DV%QUANTITY),' requires coordinates using XB'
11640 CALL SHUTDOWN(MESSAGE) ; RETURN
11641 ENDIF
11642
11643 IF (QUANTITY.INDEX<0 .AND. DV%IOR==0 .AND. (DV%STATISTICS=='null'.OR.DV%LINE>0) .AND. DV%INIT.ID=='null') THEN
11644 WRITE(MESSAGE,'(A,A,A)') 'ERROR: Specify orientation of DEVC ',TRIM(DV%ID),' using the parameter IOR'
11645 CALL SHUTDOWN(MESSAGE) ; RETURN
11646 ENDIF

```

Source Code files for edited portions of FDS

```

11646
11647 IF (QUANTITY_INDEX < 0 .AND. (DV%STATISTICS=='MASS MEAN' .OR. DV%STATISTICS=='VOLUME MEAN' .OR. &
11648 DV%STATISTICS=='VOLUME INTEGRAL' .OR. DV%STATISTICS=='MASS INTEGRAL' .OR. &
11649 DV%STATISTICS=='AREA INTEGRAL' .OR. DV%STATISTICS=='MASS' .OR. &
11650 DV%STATISTICS=='VOLUME')) THEN
11651 WRITE(MESSAGE,'(A,A)') 'ERROR: Invalid STATISTICS specified for wall DEVC ',TRIM(DV%ID)
11652 CALL SHUTDOWN(MESSAGE) ; RETURN
11653 ENDIF
11654
11655 IF (QUANTITY_INDEX > 0 .AND. DV%STATISTICS=='SURFACE INTEGRAL' .OR. DV%STATISTICS=='SURFACE AREA') THEN
11656 WRITE(MESSAGE,'(A,A)') 'ERROR: Invalid STATISTICS specified for gas DEVC ',TRIM(DV%ID)
11657 CALL SHUTDOWN(MESSAGE) ; RETURN
11658 ENDIF
11659
11660 IF (QUANTITY_INDEX > 0 .AND. &
11661 DV%STATISTICS/='null' .AND. &
11662 DV%STATISTICS/='TIME INTEGRAL' .AND. &
11663 DV%STATISTICS/='RMS' .AND. &
11664 DV%STATISTICS/='COV' .AND. &
11665 DV%STATISTICS/='CORRcoef' .AND. &
11666 DV%STATISTICS/='TIME MIN' .AND. &
11667 DV%STATISTICS/='TIME MAX' .AND. DV%I1 < 0) THEN
11668 WRITE(MESSAGE,'(A,A)') 'ERROR: XB required when geometrical STATISTICS specified for gas DEVC ',TRIM(DV%ID)
11669 CALL SHUTDOWN(MESSAGE) ; RETURN
11670 ENDIF
11671
11672 IF (TRIM(DV%QUANTITY)=='NODE PRESSURE DIFFERENCE') THEN
11673 CALL GET_QUANTITY_INDEX(DV%SMOKEVIEW_LABEL,DV%SMOKEVIEW_BAR_LABEL,QUANTITY_INDEX,LDUM(1), &
11674 DV%Y_INDEX,DV%Z_INDEX,DV%PART_INDEX,DV%DUCT_INDEX,DV%NODE_INDEX(2),LDUM(2),'DEVC', &
11675 DV%QUANTITY,'null',DV%SPEC_ID,DV%PART_ID,DV%DUCT_ID,DV%NODE_ID(2),'null')
11676 IF (DV%NODE_INDEX(1)=DV%NODE_INDEX(2)) THEN
11677 WRITE(MESSAGE,'(A,A)') 'ERROR: NODE PRESSURE DIFFERENCE node 1 = node 2 ',TRIM(DV%ID)
11678 CALL SHUTDOWN(MESSAGE) ; RETURN
11679 ENDIF
11680 ENDIF
11681
11682 IF (OUTPUT_QUANTITY(QUANTITY_INDEX)%INTEGRATED_PARTICLES) DEVC_PARTICLE_FLUX = .TRUE.
11683
11684 IF (ANY(ABS(DV%GHOST_CELL_IOR)>0)) THEN
11685 IF (DV%STATISTICS/='null') THEN
11686 WRITE(MESSAGE,'(A,A)') 'ERROR: Statistics not appropriate for GHOST_CELL_IOR DEVC ',TRIM(DV%ID)
11687 CALL SHUTDOWN(MESSAGE) ; RETURN
11688 ENDIF
11689 ENDIF
11690
11691 ENDIF QUANTITY_IF
11692
11693 ! Even if the device is not in a mesh that is handled by the current MPI process, assign its unit.
11694
11695 DV%OUTPUT_INDEX = QUANTITY_INDEX
11696 DV%QUANTITY = OUTPUT_QUANTITY(QUANTITY_INDEX)%NAME
11697 IF (DV%UNITS=='null') DV%UNITS = OUTPUT_QUANTITY(DV%OUTPUT_INDEX)%UNITS
11698
11699 ! Only process the device if it belongs to the current MPI process.
11700
11701 ! IF (PROCESS(DV%MESH)/=MYID) CYCLE PROC.DEVC_LOOP
11702
11703 ! Assign properties to the DEVICE array
11704
11705 M => MESHES(DV%MESH)
11706
11707 DV%T_CHANGE = 1.E7_EB
11708 DV%i = MAX( 1 , MIN( M%IBAR , FLOOR(GINV(DV%X-M%XS,1,DV%MESH)*M%RD XI) +1 ) )
11709 DV%j = MAX( 1 , MIN( M%JBAR , FLOOR(GINV(DV%Y-M%YS,2,DV%MESH)*M%RD ETA) +1 ) )
11710 DV%k = MAX( 1 , MIN( M%KBAR , FLOOR(GINV(DV%Z-M%ZS,3,DV%MESH)*M%RD ZETA)+1 ) )
11711 DV%CTRL_INDEX = 0
11712 DV%T = T_BEGIN
11713 DV%TMP_L = TMPA
11714 DV%TL_VALUE = 0..EB
11715 DV%TL_T = 0..EB
11716
11717 ! COVariance and CORRelation COEFFicient STATISTICS requiring QUANTITY2
11718
11719 QUANTITY2_IF: IF (TRIM(DV%QUANTITY2)/='null') THEN
11720 IF (TRIM(DV%STATISTICS)/='COV' .AND. TRIM(DV%STATISTICS)/='CORRcoef') THEN
11721 WRITE(MESSAGE,'(A,A)') 'ERROR: QUANTITY2 only applicable to COV and CORRcoef STATISTICS for DEVC ',TRIM(DV%ID)
11722 CALL SHUTDOWN(MESSAGE) ; RETURN
11723 ENDIF
11724 IF (DV%RELATIVE) THEN
11725 WRITE(MESSAGE,'(A,A)') 'ERROR: RELATIVE not applicable for COV and CORRcoef STATISTICS for DEVC ',TRIM(DV%ID)
11726 CALL SHUTDOWN(MESSAGE) ; RETURN
11727 ENDIF
11728 DV%RELATIVE=.FALSE.
11729 CALL GET_QUANTITY_INDEX(DV%SMOKEVIEW_LABEL,DV%SMOKEVIEW_BAR_LABEL,QUANTITY2_INDEX,LDUM(1), &
11730 DV%Y_INDEX,DV%Z_INDEX,DV%PART_INDEX,DV%DUCT_INDEX,DV%NODE_INDEX(1),DV%REAC_INDEX,'DEVC', &
11731 DV%QUANTITY2,'null',DV%SPEC_ID,DV%PART_ID,DV%DUCT_ID,DV%NODE_ID(1),DV%REAC_ID)
11732 DV%OUTPUT2_INDEX = QUANTITY2_INDEX
11733 DV%QUANTITY2 = OUTPUT_QUANTITY(QUANTITY2_INDEX)%NAME

```

Source Code files for edited portions of FDS

```

11734 DV%SMOKEVIEW_LABEL = TRIM(DV%QUANTITY) // ' //TRIM(DV%QUANTITY2) // ' //TRIM(DV%STATISTICS)
11735 DV%SMOKEVIEW_BAR_LABEL = TRIM(OUTPUT_QUANTITY(DV%OUTPUT_INDEX)%SHORT_NAME) // ' // &
11736 TRIM(OUTPUT_QUANTITY(DV%OUTPUT2_INDEX)%SHORT_NAME) // ' //TRIM(DV%STATISTICS)
11737 SELECT CASE(TRIM(DV%STATISTICS))
11738 CASE('COV')
11739 DV%UNITS = TRIM(OUTPUT_QUANTITY(DV%OUTPUT_INDEX)%UNITS) // '* //TRIM(OUTPUT_QUANTITY(DV%OUTPUT2_INDEX)%UNITS)
11740 CASE('CORRcoef')
11741 DV%UNITS = ''
11742 END SELECT
11743 ENDIF QUANTITY2_IF
11744
11745 ! Initialize histogram
11746
11747 IF (PROPERTY(DV%PROP_INDEX)%PDPA_HISTOGRAM) THEN
11748 ALLOCATE(DV%PDPA_HISTOGRAM_COUNTS(PROPERTY(DV%PROP_INDEX)%PDPA_HISTOGRAM_NBINS))
11749 DV%PDPA_HISTOGRAM_COUNTS(:) = 0._EB
11750 N_PDPA_HISTOGRAM = N_PDPA_HISTOGRAM + 1
11751 MAX_PDPA_HISTOGRAM_NBINS = MAX(MAX_PDPA_HISTOGRAM_NBINS, PROPERTY(DV%PROP_INDEX)%PDPA_HISTOGRAM_NBINS)
11752 ENDIF
11753
11754 ! Do initialization of special models
11755
11756 SPECIAL_QUANTITIES: SELECT CASE (DV%QUANTITY)
11757
11758 CASE ('CHAMBER_OBSCURATION')
11759
11760 IF (DV%PROP_INDEX < 1) THEN
11761 WRITE(MESSAGE, '(A,A,A) 'ERROR: DEVC ', TRIM(DV%ID), ' is a smoke detector and must have a PROP_ID'
11762 CALL SHUTDOWN(MESSAGE) ; RETURN
11763 ENDIF
11764 IF (PROPERTY(DV%PROP_INDEX)%Y_INDEX < 0 .AND. PROPERTY(DV%PROP_INDEX)%Z_INDEX < 0) THEN
11765 IF (SOOT_INDEX < 1) THEN
11766 WRITE(MESSAGE, '(A,A,A) 'ERROR: DEVC ', TRIM(DV%ID), ' is a smoke detector and requires a smoke source'
11767 CALL SHUTDOWN(MESSAGE) ; RETURN
11768 ELSE
11769 PROPERTY(DV%PROP_INDEX)%Y_INDEX = SOOT_INDEX
11770 ENDIF
11771 ENDIF
11772 ALLOCATE(DV%T_E(-1:1000))
11773 ALLOCATE(DV%Y_E(-1:1000))
11774 DV%T_E = T_BEGIN - DT
11775 DV%Y_E = 0._EB
11776 DV%N_T_E = -1
11777 DV%Y_C = 0._EB
11778 DV%SETPOINT = PROPERTY(DV%PROP_INDEX)%ACTIVATION_TEMPERATURE
11779 IF (PROPERTY(DV%PROP_INDEX)%Y_INDEX > 0) DV%Y_INDEX = PROPERTY(DV%PROP_INDEX)%Y_INDEX
11780 IF (PROPERTY(DV%PROP_INDEX)%Z_INDEX > 0) DV%Z_INDEX = PROPERTY(DV%PROP_INDEX)%Z_INDEX
11781
11782 CASE ('LINK_TEMPERATURE', 'SPRINKLER_LINK_TEMPERATURE')
11783
11784 IF (DV%PROP_INDEX < 1) THEN
11785 WRITE(MESSAGE, '(A,A,A) 'ERROR: DEVC ', TRIM(DV%ID), ' must have a PROP_ID'
11786 CALL SHUTDOWN(MESSAGE) ; RETURN
11787 ENDIF
11788 IF (PROPERTY(DV%PROP_INDEX)%ACTIVATION_TEMPERATURE <= -273.15_EB) THEN
11789 WRITE(MESSAGE, '(A,A) 'ERROR: ACTIVATION_TEMPERATURE needed for PROP ', TRIM(DV%PROP_ID)
11790 CALL SHUTDOWN(MESSAGE) ; RETURN
11791 ENDIF
11792
11793 DV%SETPOINT = PROPERTY(DV%PROP_INDEX)%ACTIVATION_TEMPERATURE
11794 DV%TMP_L = PROPERTY(DV%PROP_INDEX)%INITIAL_TEMPERATURE
11795
11796 CASE ('THERMOCOUPLE')
11797
11798 IF (DV%STATISTICS == 'MAX' .OR. DV%STATISTICS == 'MIN' .OR. DV%STATISTICS == 'MASS MEAN' .OR. DV%STATISTICS == 'VOLUME MEAN' .OR
11799 &
11800 DV%STATISTICS == 'MASS INTEGRAL' .OR. DV%STATISTICS == 'VOLUME INTEGRAL' .OR. DV%STATISTICS == 'MEAN' .OR. &
11801 DV%STATISTICS == 'AREA INTEGRAL') THEN
11802 WRITE(MESSAGE, '(A,A,A) 'ERROR: DEVC ', TRIM(DV%ID), ' cannot use volume STATISTICS'
11803 CALL SHUTDOWN(MESSAGE) ; RETURN
11804 ENDIF
11805 DV%TMP_L = PROPERTY(DV%PROP_INDEX)%INITIAL_TEMPERATURE
11806
11807 CASE ('SOLID_DENSITY')
11808
11809 IF (DV%MATL_ID == 'null') THEN
11810 WRITE(MESSAGE, '(A,A,A) 'ERROR: DEVC ', TRIM(DV%ID), ' must have a MATL_ID'
11811 CALL SHUTDOWN(MESSAGE) ; RETURN
11812 ENDIF
11813
11814 CASE ('LAYER_HEIGHT', 'UPPER_TEMPERATURE', 'LOWER_TEMPERATURE')
11815
11816 DV%k1 = MAX(1, DV%k1)
11817 DV%k2 = MIN(M%KBAR, DV%k2)
11818
11819 CASE ('TRANSMISSION', 'PATH_OBSCURATION')
11820
11821 IF (DV%PROP_INDEX > 0) THEN

```

Source Code files for edited portions of FDS

```

11821 IF (PROPERTY(DV%PROP_INDEX)%Y_INDEX<1 .AND. PROPERTY(DV%PROP_INDEX)%Z_INDEX<1) THEN
11822 IF (SOOT_INDEX<1) THEN
11823 WRITE(MESSAGE,'(A,A,A)') 'ERROR: DEVC ',TRIM(DV%D), ' is a smoke detector and requires a smoke source'
11824 CALL SHUTDOWN(MESSAGE) ; RETURN
11825 ELSE
11826 PROPERTY(DV%PROP_INDEX)%Y_INDEX = SOOT_INDEX
11827 ENDIF
11828 ENDIF
11829 ELSE
11830 IF (SOOT_INDEX <=0) THEN
11831 WRITE(MESSAGE,'(A,A,A)') 'ERROR: DEVC ',TRIM(DV%D), ' is a smoke detector and requires a smoke source'
11832 CALL SHUTDOWN(MESSAGE) ; RETURN
11833 ENDIF
11834 ENDIF
11835 IF (PROPERTY(DV%PROP_INDEX)%Y_INDEX>0) DV%Y_INDEX = PROPERTY(DV%PROP_INDEX)%Y_INDEX
11836 IF (PROPERTY(DV%PROP_INDEX)%Z_INDEX>0) DV%Z_INDEX = PROPERTY(DV%PROP_INDEX)%Z_INDEX
11837 NM = DV%MESH
11838 M=>MESHES(NM)
11839 DISTANCE = SQRT((DV%X1-DV%X2)**2 + (DV%Y1-DV%Y2)**2 + (DV%Z1-DV%Z2)**2)
11840 SCANDISTANCE = 0.0001.EB * DISTANCE
11841 DX = (DV%X2-DV%X1) * 0.0001.EB
11842 DY = (DV%Y2-DV%Y1) * 0.0001.EB
11843 DZ = (DV%Z2-DV%Z1) * 0.0001.EB
11844 XX = DV%X1
11845 YY = DV%Y1
11846 ZZ = DV%Z1
11847 MAXCELLS = 2*MAX(M%IBAR,M%JBAR,M%KBAR)
11848 ALLOCATE(DV%I_PATH(MAXCELLS))
11849 ALLOCATE(DV%J_PATH(MAXCELLS))
11850 ALLOCATE(DV%K_PATH(MAXCELLS))
11851 ALLOCATE(DV%D_PATH(MAXCELLS))
11852 DV%D_PATH = 0..EB
11853 DV%I_PATH = MIN(M%IBAR , INT(GINV(DV%X1-M%X,1,NM)*M%RDXI) + 1)
11854 DV%J_PATH = MIN(M%JBAR , INT(GINV(DV%Y1-M%Y,2,NM)*M%RDZETA) + 1)
11855 DV%K_PATH = MIN(M%KBAR , INT(GINV(DV%Z1-M%Z,3,NM)*M%RDZETA) + 1)
11856 DV%N_PATH = 1
11857 NN = 1
11858 DO NNN=1,10000
11859 XX = XX + DX
11860 I = MIN(M%IBAR , INT(GINV(XX-M%X,1,NM)*M%RDXI) + 1)
11861 YY = YY + DY
11862 J = MIN(M%JBAR , INT(GINV(YY-M%Y,2,NM)*M%RDZETA) + 1)
11863 ZZ = ZZ + DZ
11864 K = MIN(M%KBAR , INT(GINV(ZZ-M%Z,3,NM)*M%RDZETA) + 1)
11865 IF (I==DV%I_PATH(NN) .AND. J==DV%J_PATH(NN) .AND. K==DV%K_PATH(NN)) THEN
11866 DV%D_PATH(NN) = DV%D_PATH(NN) + SCANDISTANCE
11867 ELSE
11868 NN = NN + 1
11869 DV%I_PATH(NN) = I
11870 DV%J_PATH(NN) = J
11871 DV%K_PATH(NN) = K
11872 XX1 = DX
11873 YY1 = DY
11874 ZZ1 = DZ
11875 IF (PROCESS(DV%MESH)==MYID) THEN
11876 IF (I<DV%I_PATH(NN-1)) XX1 = XX-M%X(DV%I_PATH(NN-1)-1)
11877 IF (I>DV%I_PATH(NN-1)) XX1 = XX-M%X(DV%I_PATH(NN-1))
11878 IF (J<DV%J_PATH(NN-1)) YY1 = YY-M%Y(DV%J_PATH(NN-1)-1)
11879 IF (J>DV%J_PATH(NN-1)) YY1 = YY-M%Y(DV%J_PATH(NN-1))
11880 IF (K<DV%K_PATH(NN-1)) ZZ1 = ZZ-M%Z(DV%K_PATH(NN-1)-1)
11881 IF (K>DV%K_PATH(NN-1)) ZZ1 = ZZ-M%Z(DV%K_PATH(NN-1))
11882 ENDIF
11883 DV%D_PATH(NN) = SCANDISTANCE - SQRT(XX1**2+YY1**2+ZZ1**2)
11884 DV%D_PATH(NN-1) = DV%D_PATH(NN-1) + SCANDISTANCE - DV%D_PATH(NN)
11885 ENDIF
11886 ENDDO
11887 DV%N_PATH = NN
11888
11889 CASE ('CONTROL')
11890
11891 DO NN=1,N_CTRL
11892 IF (CONTROL(NN)%ID==DV%CTRL_ID) DV%CTRL_INDEX = NN
11893 ENDDO
11894 IF (DV%CTRL_ID/='null' .AND. DV%CTRL_INDEX<=0) THEN
11895 WRITE(MESSAGE,'(A,A,A)') 'ERROR: CONTROL ',TRIM(DV%CTRL_ID), ' does not exist'
11896 CALL SHUTDOWN(MESSAGE) ; RETURN
11897 ENDIF
11898 DV%SETPOINT = 0.5
11899 DV%TRIP_DIRECTION = 1
11900
11901 CASE ('CONTROL VALUE')
11902
11903 DO NN=1,N_CTRL
11904 IF (CONTROL(NN)%ID==DV%CTRL_ID) DV%CTRL_INDEX = NN
11905 ENDDO
11906 IF (DV%CTRL_ID/='null' .AND. DV%CTRL_INDEX<=0) THEN
11907 WRITE(MESSAGE,'(A,A,A)') 'ERROR: CONTROL ',TRIM(DV%CTRL_ID), ' does not exist'
11908 CALL SHUTDOWN(MESSAGE) ; RETURN

```

Source Code files for edited portions of FDS

```

11909   ENDIF
11910
11911   CASE ('ASPIRATION')
11912
11913   ! Check either for a specified SMOKE SPECies, or if simple chemistry model is being used
11914   IF (DV%PROP_INDEX>0) THEN
11915   IF (PROPERTY(DV%PROP_INDEX)%Y_INDEX<1 .AND. PROPERTY(DV%PROP_INDEX)%Z_INDEX<1 .AND. SOOT_INDEX<1) THEN
11916   WRITE(MESSAGE, '(A,A,A)') 'ERROR: DEVC ',TRIM(DV%ID), ' is a smoke detector and requires a smoke source'
11917   CALL SHUTDOWN(MESSAGE) ; RETURN
11918   ENDIF
11919   ENDIF
11920   ! Count number of inputs for detector and verify that input is DENSITY with a specified SPEC.ID for smoke
11921   NNN = 0
11922   DO NN=1,N.DEVC
11923   IF (DEVICE(NN)%DEV_C_ID==DV%ID) THEN
11924   IF (DEVICE(NN)%QUANTITY/= 'DENSITY' .OR. DEVICE(NN)%SPEC_ID=='null') THEN
11925   WRITE(MESSAGE, '(A,A,A)') &
11926   'ERROR: DEVICE ',TRIM(DEVICE(NN)%ID), ' must use QUANTITY='DENSITY' and a SPEC.ID'
11927   CALL SHUTDOWN(MESSAGE) ; RETURN
11928   ENDIF
11929   NNN = NNN + 1
11930   ENDIF
11931   ENDDO
11932   ALLOCATE(DV%DEV_C_INDEX(NNN),STAT=IZERO)
11933   CALL ChkMemErr('READ', 'DV%DEV_C_INDEX', IZERO)
11934   DV%DEV_C_INDEX = -1
11935   ALLOCATE(DV%Y_SOOT(NNN,0:100))
11936   CALL ChkMemErr('READ', 'DV%Y_SOOT', IZERO)
11937   DV%Y_SOOT = 0._EB
11938   ALLOCATE(DV%TIME_ARRAY(0:100))
11939   CALL ChkMemErr('READ', 'DV%TIME_ARRAY', IZERO)
11940   DV%TIME_ARRAY = 0._EB
11941   DV%TOTAL_FLOWRATE = DV%BYPASS_FLOWRATE
11942   DV%DT = -1._EB
11943   DV%N_INPUTS = NNN
11944   NNN = 1
11945   DO NN=1,N.DEVC
11946   IF (DEVICE(NN)%DEV_C_ID==DV%ID) THEN
11947   DV%TOTAL_FLOWRATE = DV%TOTAL_FLOWRATE + DEVICE(NN)%FLOWRATE
11948   DV%DT = MAX(DV%DT, DEVICE(NN)%DELAY)
11949   IF (NN > N) THEN
11950   WRITE(MESSAGE, '(A,A,A)') 'ERROR: ASPIRATION DEVICE ',TRIM(DV%ID), ' is not listed after all its inputs'
11951   CALL SHUTDOWN(MESSAGE) ; RETURN
11952   ENDIF
11953   DV%DEV_C_INDEX(NNN) = NN
11954   NNN = NNN + 1
11955   ENDIF
11956   ENDDO
11957   DV%DT = DV%DT * 0.01._EB
11958
11959   CASE ('FED')
11960   IF (DV%STATISTICS /= 'null' .AND. DV%STATISTICS /= 'TIME INTEGRAL') THEN
11961   WRITE(MESSAGE, '(A)') 'ERROR: Only TIME INTEGRAL statistics can be used with FED devices'
11962   CALL SHUTDOWN(MESSAGE) ; RETURN
11963   ENDIF
11964   DV%STATISTICS = 'TIME INTEGRAL'
11965   IF (DV%PROP_INDEX>0) THEN
11966   IF (PROPERTY(DV%PROP_INDEX)%FED_ACTIVITY<1 .AND. PROPERTY(DV%PROP_INDEX)%FED_ACTIVITY>3) THEN
11967   WRITE(MESSAGE, '(A,A,A)') 'ERROR: DEVC ',TRIM(DV%ID), ' is a FED detector and requires an activity'
11968   CALL SHUTDOWN(MESSAGE) ; RETURN
11969   ENDIF
11970   DV%FED_ACTIVITY = PROPERTY(DV%PROP_INDEX)%FED_ACTIVITY
11971   ENDIF
11972
11973   CASE ('VELOCITY PATCH')
11974   PATCH_VELOCITY = .TRUE.
11975   ALLOCATE(DV%DEV_C_INDEX(1),STAT=IZERO)
11976   DV%DEV_C_INDEX(1) = 0
11977   DO NN=1,N.DEVC
11978   IF (DEVICE(NN)%ID==DV%DEV_C_ID) DV%DEV_C_INDEX(1) = NN
11979   ENDDO
11980   IF (DV%DEV_C_INDEX(1)==0) THEN
11981   WRITE(MESSAGE, '(A)') 'ERROR: A VELOCITY PATCH DEVC line needs a DEVC.ID to control it'
11982   CALL SHUTDOWN(MESSAGE) ; RETURN
11983   ENDIF
11984
11985   END SELECT SPECIAL_QUANTITIES
11986
11987   IF (DV%STATISTICS/='null') CALL CHANGE_UNITS(DV%QUANTITY, DV%UNITS, DV%STATISTICS, MYID, LU_ERR)
11988
11989   IF (DV%LINE == 0 .AND. (DV%STATISTICS=='RMS' .OR. DV%STATISTICS=='COV' .OR. DV%STATISTICS=='CORRcoef')) THEN
11990   DV%TIME_AVERAGED=.FALSE.
11991   IF (DV%STATISTICS.START < T.BEGIN) DV%STATISTICS.START = T.BEGIN
11992   ENDIF
11993
11994   IF (DV%NO_UPDATE.DEVC_ID/='null' .OR. DV%NO_UPDATE.CTRL_ID/='null') &
11995   CALL SEARCH_CONTROLLER('DEVC', DV%NO_UPDATE.CTRL_ID, DV%NO_UPDATE.DEVC_ID, DV%NO_UPDATE.DEVC_INDEX, DV%
NO_UPDATE.CTRL_INDEX, N)

```

```

11996
11997 ENDDO PROC.DEVC.LOOP
11998
11999 END SUBROUTINE PROC.DEVC
12000
12001
12002 SUBROUTINE READ.PROF
12003
12004 INTEGER :: N,NM,MESHNUMBER,NN,N.PROFO,IOR,FORMATINDEX
12005 REAL(EB) :: XYZ(3)
12006 CHARACTER(LABELLENGTH) :: QUANTITY
12007 TYPE (PROFILE_TYPE), POINTER :: PF=>NULL()
12008 NAMELIST /PROF/ FORMATINDEX,FYI,ID,IOR,QUANTITY,XYZ
12009
12010 N.PROF = 0
12011 REWIND(LU.INPUT) ; INPUT_FILE.LINE.NUMBER = 0
12012 COUNT.PROF.LOOP: DO
12013 CALL CHECKREAD('PROF',LU.INPUT,IOS)
12014 IF (IOS==1) EXIT COUNT.PROF.LOOP
12015 READ(LU.INPUT,NML=PROF,END=11,ERR=12,IOSTAT=IOS)
12016 N.PROF = N.PROF + 1
12017 12 IF (IOS>0) THEN
12018 WRITE(MESSAGE,'(A,I0,A,I0)') 'ERROR: Problem with PROF number ',N.PROF+1,', line number',INPUT_FILE.LINE.NUMBER
12019 CALL SHUTDOWN(MESSAGE) ; RETURN
12020 ENDIF
12021 ENDDO COUNT.PROF.LOOP
12022 11 REWIND(LU.INPUT) ; INPUT_FILE.LINE.NUMBER = 0
12023
12024 IF (N.PROF==0) RETURN
12025
12026 ALLOCATE(PROFILE(N.PROF),STAT=IZERO)
12027 CALL ChkMemErr('READ','PROFILE',IZERO)
12028
12029 PROFILE(1:N.PROF)%QUANTITY = 'TEMPERATURE'
12030 PROFILE(1:N.PROF)%IOR = 0
12031 PROFILE(1:N.PROF)%IW = 0
12032
12033 N.PROFO = N.PROF
12034 N = 0
12035
12036 PROF.LOOP: DO NN=1,N.PROFO
12037 N = N+1
12038 FORMATINDEX = 1
12039 IOR = 0
12040 SELECT CASE(N)
12041 CASE(1:9)
12042 WRITE(ID,'(A,I1)') 'PROFILE ',N
12043 CASE(10:99)
12044 WRITE(ID,'(A,I2)') 'PROFILE ',N
12045 CASE(100:999)
12046 WRITE(ID,'(A,I3)') 'PROFILE ',N
12047 END SELECT
12048
12049 CALL CHECKREAD('PROF',LU.INPUT,IOS)
12050 IF (IOS==1) EXIT PROF.LOOP
12051 READ(LU.INPUT,PROF)
12052
12053 ! Check for bad PROF quantities or coordinates
12054
12055 IF (IOR==0) THEN
12056 WRITE(MESSAGE,'(A,I0,A)') 'ERROR: Specify orientation of PROF ',N,', using the parameter IOR'
12057 CALL SHUTDOWN(MESSAGE) ; RETURN
12058 ENDIF
12059
12060 BAD = .FALSE.
12061
12062 MESHLOOP: DO NM=1,NMESHES
12063 IF (.NOT.EVACUATIONONLY(NM)) THEN
12064 M=>MESHES(NM)
12065 IF (XYZ(1)>=M%XS .AND. XYZ(1)<=M%XF .AND. XYZ(2)>=M%YS .AND. XYZ(2)<=M%YF .AND. XYZ(3)>=M%ZS .AND. XYZ(3)<=M%ZF)
12066 THEN
12067 MESHNUMBER = NM
12068 EXIT MESHLOOP
12069 ENDIF
12070 ENDIF
12071 IF (NM==NMESHES) BAD = .TRUE.
12072 ENDDO MESHLOOP
12073
12074 IF (BAD) THEN
12075 N = N-1
12076 N.PROF = N.PROF-1
12077 CYCLE PROF.LOOP
12078 ENDIF
12079 ! Assign parameters to the PROFILE array
12080 PF => PROFILE(N)
12081 PP%FORMATINDEX = FORMATINDEX
12082

```

```

12083 PP%ORDINAL = NN
12084 PP%MESH = MESHNUMBER
12085 PP%ID = ID
12086 PP%QUANTITY = QUANTITY
12087 PP%X = XYZ(1)
12088 PP%Y = XYZ(2)
12089 PP%Z = XYZ(3)
12090 PP%IOR = IOR
12091
12092 ENDDO PROF_LOOP
12093 REWIND(LU_INPUT) ; INPUT_FILE_LINE_NUMBER = 0
12094
12095 END SUBROUTINE READ_PROF
12096
12097
12098
12099 SUBROUTINE READ_ISOF
12100
12101 REAL(EB) :: VALUE(10)
12102 CHARACTER(LABEL_LENGTH) :: QUANTITY,SPEC_ID
12103 INTEGER :: N,VELO_INDEX
12104 TYPE(ISOSURFACE_FILE_TYPE), POINTER :: IS=>NULL()
12105 NAMELIST /ISOF/ FYI,QUANTITY,SPEC_ID,VALUE,VELO_INDEX
12106
12107 N_ISOF = 0
12108 REWIND(LU_INPUT) ; INPUT_FILE_LINE_NUMBER = 0
12109 COUNT_ISOF_LOOP: DO
12110 CALL CHECKREAD('ISOF',LU_INPUT,IOS)
12111 IF (IOS==1) EXIT COUNT_ISOF_LOOP
12112 READ(LU_INPUT,NML=ISOF,END=9,ERR=10,IOSTAT=IOS)
12113 N_ISOF = N_ISOF + 1
12114 10 IF (IOS>0) THEN
12115 WRITE(MESSAGE,'(A,I0,A,I0)') 'ERROR: Problem with ISOF number ',N_ISOF,', line number',INPUT_FILE_LINE_NUMBER
12116 CALL SHUTDOWN(MESSAGE) ; RETURN
12117 ENDIF
12118 ENDDO COUNT_ISOF_LOOP
12119 9 REWIND(LU_INPUT) ; INPUT_FILE_LINE_NUMBER = 0
12120
12121 ALLOCATE(ISOSURFACE_FILE(N_ISOF),STAT=IZERO)
12122 CALL ChkMemErr('READ','ISOSURFACE_FILE',IZERO)
12123
12124 READ_ISOF_LOOP: DO N=1,N_ISOF
12125 IS => ISOSURFACE_FILE(N)
12126 QUANTITY = 'null'
12127 SPEC_ID = 'null'
12128 VALUE = -999..EB
12129 VELO_INDEX = 0
12130
12131 CALL CHECKREAD('ISOF',LU_INPUT,IOS)
12132 IF (IOS==1) EXIT READ_ISOF_LOOP
12133 READ(LU_INPUT,ISOF)
12134
12135 IS%VELO_INDEX = VELO_INDEX
12136
12137 CALL GET_QUANTITY_INDEX(IS%SMOKEVIEW_LABEL,IS%SMOKEVIEW_BAR_LABEL,IS%INDEX,LDUM(1), &
12138 IS%Y_INDEX,IS%Z_INDEX,LDUM(2),LDUM(3),LDUM(4),LDUM(5),'ISOF', &
12139 QUANTITY,'null',SPEC_ID,'null','null','null','null')
12140
12141 VALUE_LOOP: DO I=1,10
12142 IF (VALUE(I)<=-998..EB) EXIT VALUE_LOOP
12143 IS%N_VALUES = I
12144 IS%VALUE(I) = REAL(VALUE(I),FB)
12145 ENDDO VALUE_LOOP
12146
12147 ENDDO READ_ISOF_LOOP
12148
12149 REWIND(LU_INPUT) ; INPUT_FILE_LINE_NUMBER = 0
12150
12151 END SUBROUTINE READ_ISOF
12152
12153
12154 SUBROUTINE READ_SLICF
12155
12156 REAL(EB) :: MAXIMUM_VALUE,MINIMUM_VALUE
12157 REAL(EB) :: AGL_SLICE
12158 INTEGER :: N,NN,NM,MESHNUMBER,N_SLICF_O,NITER,ITER,VELO_INDEX,IOR
12159 LOGICAL :: VECTOR,CELL_CENTERED,FACE_CENTERED,FIRE_LINE,EVACUATION,LEVEL_SET_FIRE_LINE
12160 CHARACTER(LABEL_LENGTH) :: QUANTITY,SPEC_ID,PART_ID,QUANTITY2,PROP_ID,REAC_ID,SLICETYPE
12161 REAL(EB),PARAMETER :: TOL=1.E-10..EB
12162 REAL(EB) :: DELX,DELY,DELZ,SMV_OFFSET
12163 TYPE(SLICE_TYPE), POINTER :: SL=>NULL()
12164 NAMELIST /SLICF/ AGL_SLICE,CELL_CENTERED,EVACUATION,FACE_CENTERED,FIRE_LINE,FYI,ID,IOR,LEVEL_SET_FIRE_LINE,
12165 MAXIMUM_VALUE,&
12166 MESHNUMBER,MINIMUM_VALUE,PART_ID,PBX,PBY,PBZ,PROP_ID,QUANTITY,QUANTITY2,REAC_ID,SLICETYPE,SMV_OFFSET,SPEC_ID,&
12167 VECTOR,VELO_INDEX,XB
12168
12169 MESH_LOOP: DO NM=1,NMESHES

```



```

12170 IF (MYID/=PROCESS(NM)) CYCLE MESHLOOP
12171
12172 M=>MESHERS(NM)
12173 CALL POINT_TO_MESH(NM)
12174
12175 N_SLCF = 0
12176 N_SLCF_O = 0
12177 REWIND(LU_INPUT) ; INPUT_FILE_LINE_NUMBER = 0
12178 COUNT_SLCF_LOOP: DO
12179 VECTOR = .FALSE.
12180 EVACUATION = .FALSE.
12181 MESHNUMBER=NM
12182 CALL CHECKREAD('SLCF',LU_INPUT,IOS)
12183 IF (IOS==1) EXIT COUNT_SLCF_LOOP
12184 READ(LU_INPUT,NML=SLCF,END=9,ERR=10,IOSTAT=IOS)
12185 N_SLCF_O = N_SLCF_O + 1
12186 IF (MESHNUMBER/=NM) CYCLE COUNT_SLCF_LOOP
12187 IF (.NOT.EVACUATION_ONLY(NM) .AND. EVACUATION) CYCLE COUNT_SLCF_LOOP
12188 IF ( EVACUATION_ONLY(NM) .AND. .NOT.EVACUATION) CYCLE COUNT_SLCF_LOOP
12189 IF (EVACUATION_ONLY(NM) .AND. .NOT.EVACUATION_SKIP(NM)) CYCLE COUNT_SLCF_LOOP
12190 N_SLCF = N_SLCF + 1
12191 IF (VECTOR .AND. TWO.D) N_SLCF = N_SLCF + 2
12192 IF (VECTOR .AND. .NOT. TWO.D) N_SLCF = N_SLCF + 3
12193 10 IF (IOS>0) THEN
12194 WRITE(MESSAGE,'(A,10,A,10)') 'ERROR: Problem with SLCF number ',N_SLCF_O+1,', line number',INPUT_FILE_LINE_NUMBER
12195 CALL SHUTDOWN(MESSAGE) ; RETURN
12196 ENDIF
12197 ENDDO COUNT_SLCF_LOOP
12198 9 CONTINUE
12199
12200 ALLOCATE(M%SLICE(N_SLCF),STAT=IZERO)
12201 CALL ChkMemErr('READ','ISP1',IZERO)
12202 CALL POINT_TO_MESH(NM) ! Reset the pointers after the allocation
12203
12204 N = 0
12205 N_TERRAIN_SLCF = 0
12206
12207 REWIND(LU_INPUT) ; INPUT_FILE_LINE_NUMBER = 0
12208 SLCF_LOOP: DO NN=1,N_SLCF_O
12209 QUANTITY = 'null'
12210 QUANTITY2 = 'null'
12211 SMV_OFFSET = 0.0_EB
12212 PBX = -1.E6_EB
12213 PBY = -1.E6_EB
12214 PBZ = -1.E6_EB
12215 VECTOR = .FALSE.
12216 ID = 'null'
12217 MESHNUMBER=NM
12218 MINIMUMVALUE = 0. _EB
12219 MAXIMUMVALUE = 0. _EB
12220 AGL_SLICE = -1. _EB
12221 REAC_ID = 'null'
12222 SPEC_ID = 'null'
12223 PART_ID = 'null'
12224 PROP_ID = 'null'
12225 SLICETYPE = 'STRUCTURED'
12226 IOR = 0
12227 CELL_CENTERED = .FALSE.
12228 FACE_CENTERED = .FALSE.
12229 FIRE_LINE=.FALSE.
12230 EVACUATION = .FALSE.
12231 VELO_INDEX = 0
12232 LEVEL_SET_FIRE_LINE = .FALSE.
12233
12234 CALL CHECKREAD('SLCF',LU_INPUT,IOS)
12235 IF (IOS==1) EXIT SLCF_LOOP
12236 READ(LU_INPUT,SLCF)
12237 IF (MESHNUMBER/=NM) CYCLE SLCF_LOOP
12238 IF (.NOT.EVACUATION_ONLY(NM) .AND. EVACUATION) CYCLE SLCF_LOOP
12239 IF ( EVACUATION_ONLY(NM) .AND. .NOT.EVACUATION) CYCLE SLCF_LOOP
12240 IF (EVACUATION_ONLY(NM) .AND. .NOT.EVACUATION_SKIP(NM)) CYCLE SLCF_LOOP
12241 IF (CELL_CENTERED .AND. FACE_CENTERED) FACE_CENTERED = .FALSE.
12242
12243 IF (PBX>-1.E5_EB .OR. PBY>-1.E5_EB .OR. PBZ>-1.E5_EB) THEN
12244 XB(1) = XS
12245 XB(2) = XF
12246 XB(3) = YS
12247 XB(4) = YF
12248 XB(5) = ZS
12249 XB(6) = ZF
12250 IF (PBX>-1.E5_EB) XB(1:2) = PBX
12251 IF (PBY>-1.E5_EB) XB(3:4) = PBY
12252 IF (PBZ>-1.E5_EB) XB(5:6) = PBZ
12253 ENDIF
12254
12255 CALL CHECK_XB(XB)
12256
12257 XB(1) = MAX(XB(1),XS)

```

```

12258 XB(2) = MIN(XB(2),XF)
12259 XB(3) = MAX(XB(3),YS)
12260 XB(4) = MIN(XB(4),YF)
12261 XB(5) = MAX(XB(5),ZS)
12262 XB(6) = MIN(XB(6),ZF)
12263 IF (IOR == 0) THEN ! determine slice orientation if not already specified
12264 DELX = ABS(XB(1)-XB(2))
12265 DELY = ABS(XB(3)-XB(4))
12266 DELZ = ABS(XB(5)-XB(6))
12267 IF ( DELX < MIN(DELY,DELZ) )THEN
12268 IOR = 1
12269 ELSE IF ( DELY < MIN(DELY,DELZ) )THEN
12270 IOR = 2
12271 ELSE
12272 IOR = 3
12273 ENDIF
12274 ENDIF
12275
12276 ! Reject a slice if it is beyond the bounds of the current mesh
12277
12278 IF (XB(1)>XF .OR. XB(2)<XS .OR. XB(3)>YF .OR. XB(4)<YS .OR. XB(5)>ZF .OR. XB(6)<ZS) THEN
12279 N_SLCF = N_SLCF - 1
12280 IF (VECTOR .AND. TWO.D) N_SLCF = N_SLCF - 2
12281 IF (VECTOR .AND. .NOT. TWO.D) N_SLCF = N_SLCF - 3
12282 CYCLE SLCF-LOOP
12283 ENDIF
12284
12285 ! Process vector quantities
12286
12287 NITER = 1
12288 IF (VECTOR .AND. TWO.D) NITER = 3
12289 IF (VECTOR .AND. .NOT. TWO.D) NITER = 4
12290
12291 VECTORLOOP: DO ITER=1,NITER
12292 N = N + 1
12293 SL=>SLICE(N)
12294 SL%ID = ID
12295 SL%SLICETYPE = TRIM(SLICETYPE)
12296 SL%IOR = IOR
12297 IF ((FACE.CENTERED .OR. CELL.CENTERED) .AND. NITER==1) THEN ! scalar raw data
12298 DO I=1,IBAR
12299 IF ( ABS(XB(1)-XC(I)) < 0.5.EB*DX(I) + TOL ) SL%i1 = I
12300 IF ( ABS(XB(2)-XC(I)) < 0.5.EB*DX(I) + TOL ) SL%i2 = I
12301 ENDDO
12302 DO J=1,JBAR
12303 IF ( ABS(XB(3)-YC(J)) < 0.5.EB*DY(J) + TOL ) SL%j1 = J
12304 IF ( ABS(XB(4)-YC(J)) < 0.5.EB*DY(J) + TOL ) SL%j2 = J
12305 ENDDO
12306 DO K=1,KBAR
12307 IF ( ABS(XB(5)-ZC(K)) < 0.5.EB*DZ(K) + TOL ) SL%k1 = K
12308 IF ( ABS(XB(6)-ZC(K)) < 0.5.EB*DZ(K) + TOL ) SL%k2 = K
12309 ENDDO
12310 IF (SL%i1<SL%i2) SL%i1=SL%i1-1
12311 IF (SL%j1<SL%j2) SL%j1=SL%j1-1
12312 IF (SL%k1<SL%k2) SL%k1=SL%k1-1
12313 ELSE
12314 SL%i1 = NINT( GINV(XB(1)-XS,1,NM)*RDXI )
12315 SL%i2 = NINT( GINV(XB(2)-XS,1,NM)*RDXI )
12316 SL%j1 = NINT( GINV(XB(3)-YS,2,NM)*RDETA )
12317 SL%j2 = NINT( GINV(XB(4)-YS,2,NM)*RDETA )
12318 SL%k1 = NINT( GINV(XB(5)-ZS,3,NM)*RDZETA )
12319 SL%k2 = NINT( GINV(XB(6)-ZS,3,NM)*RDZETA )
12320 ENDIF
12321 SL%MINMAX(1) = REAL(MINIMUMVALUE,FB)
12322 SL%MINMAX(2) = REAL(MAXIMUMVALUE,FB)
12323 IF (ITER==2) QUANTITY = 'U-VELOCITY'
12324 IF (ITER==3 .AND. .NOT. TWO.D) QUANTITY = 'V-VELOCITY'
12325 IF (ITER==3 .AND. TWO.D) QUANTITY = 'W-VELOCITY'
12326 IF (ITER==4) QUANTITY = 'W-VELOCITY'
12327 IF (ITER==1 .AND. FIRE.LINE) QUANTITY = 'TEMPERATURE'
12328 IF (ITER==1 .AND. LEVEL.SET_FIRE.LINE) QUANTITY = 'TEMPERATURE'
12329 IF (ITER==1) THEN
12330 SL%FIRE.LINE = FIRE.LINE
12331 SL%LEVEL.SET_FIRE.LINE = LEVEL.SET_FIRE.LINE
12332 IF (LEVEL.SET_FIRE.LINE .AND. .NOT. VEG.LEVEL.SET) THEN
12333 WRITE(MESSAGE,'(A)') "ERROR: VEG.LEVEL.SET must be TRUE on MISC line to run the LS model"
12334 CALL SHUTDOWN(MESSAGE) ; RETURN
12335 ENDIF
12336 ELSE
12337 SL%FIRE.LINE = .FALSE.
12338 SL%LEVEL.SET_FIRE.LINE = .FALSE.
12339 SPEC_ID = 'null'
12340 ENDIF
12341 SL%VELO.INDEX = VELO.INDEX
12342 CALL GET_QUANTITY.INDEX(SL%SMOKEVIEW.LABEL,SL%SMOKEVIEW.BAR.LABEL,SL%INDEX,SL%INDEX2, &
12343 SL%Y.INDEX,SL%Z.INDEX,SL%PART.INDEX,LDUM(1),LDUM(2),SL%REAC.INDEX,'SLCF', &
12344 QUANTITY,QUANTITY2,SPEC_ID,PART_ID,'null','null',REAC.ID)
12345

```

## Source Code files for edited portions of FDS

```

12346 ! If the user needs to do a particle flux calculation , detect that here .
12347
12348 IF (OUTPUT_QUANTITY(SL%INDEX)%INTEGRATED_PARTICLES) SLCF_PARTICLE_FLUX = .TRUE.
12349
12350 ! For terrain slices , AGL=above ground level
12351 ! FIRE_LINE==.TRUE. means a terrain slice at one grid cell above ground with quantity temperature .
12352 ! Smokeview will display only regions where temperature is above 200 C. This is currently hard wired .
12353 ! LEVEL_SET_FIRE_LINE = .TRUE. will create a slice file !!! this is not fully functional !!!
12354
12355 IF (ITER == 1 .AND. (AGL_SLICE > -1._EB .OR. FIRE_LINE .OR. LEVEL_SET_FIRE_LINE ) ) THEN
12356 SL%TERRAIN_SLICE = .TRUE.
12357 IF (FIRE_LINE) THEN
12358 SL%SMOKEVIEW_LABEL = "Fire line"
12359 SL%SMOKEVIEW_BAR_LABEL = "Fire_line"
12360 ENDIF
12361 IF (LEVEL_SET_FIRE_LINE) THEN
12362 SL%SMOKEVIEW_LABEL = "Level Set Fire line"
12363 SL%SMOKEVIEW_BAR_LABEL = "LS.Fire_line"
12364 ENDIF
12365 IF (AGL_SLICE <= -1._EB .AND. FIRE_LINE) AGL_SLICE = M%Z(1) - M%Z(0)
12366 IF (AGL_SLICE <= -1._EB .AND. LEVEL_SET_FIRE_LINE) AGL_SLICE = 0._EB
12367 SL%SLICE_AGL = AGL_SLICE
12368 N_TERRAIN_SLCF = N_TERRAIN_SLCF + 1
12369 ENDIF
12370 IF (ITER==2 .OR. ITER==3 .OR. ITER ==4) THEN
12371 IF (SLICE(N-1)%TERRAIN_SLICE) THEN
12372 SL%TERRAIN_SLICE = .TRUE.
12373 SL%SLICE_AGL = SLICE(N-1)%SLICE_AGL
12374 N_TERRAIN_SLCF = N_TERRAIN_SLCF + 1
12375 ENDIF
12376 ENDIF
12377
12378 ! Disable cell centered for velocity
12379
12380 ! IF (QUANTITY=='VELOCITY' .OR. &
12381 ! QUANTITY=='U-VELOCITY' .OR. &
12382 ! QUANTITY=='V-VELOCITY' .OR. &
12383 ! QUANTITY=='W-VELOCITY') THEN
12384 ! CELL_CENTERED = .FALSE.
12385 ! ENDIF
12386 SL%CELL_CENTERED = CELL_CENTERED
12387 SL%FACE_CENTERED = FACE_CENTERED
12388
12389 ! Check if the slcf PROPERTY exists (for FED_ACTIVITY input)
12390
12391 SL%PROP_INDEX = 0
12392 IF (PROP_ID /= 'null') THEN
12393 CALL GET_PROPERTY_INDEX(SL%PROP_INDEX, 'SLCF', PROP_ID)
12394 ENDIF
12395
12396 SL%SMV_OFFSET = SMV_OFFSET
12397
12398 ENDDO VECTORLOOP
12399
12400 IF (TRIM(QUANTITY)=='CHEMISTRY SUBITERATIONS') OUTPUT_CHEM_IT = .TRUE.
12401
12402 IF (TRIM(QUANTITY)=='REAC SOURCE TERM' .OR. TRIM(QUANTITY)=='HRRPUV REAC') REAC_SOURCE_CHECK = .TRUE.
12403
12404 IF (TRIM(QUANTITY)=='H PRIME' .AND. .NOT.EXTERNAL_BOUNDARY_CORRECTION) THEN
12405 WRITE(MESSAGE, '(A,I0,A)') 'ERROR: Problem with SCLF ', NN, ', H PRIME requires EXTERNAL_BOUNDARY_CORRECTION=T on
MISC'
12406 CALL SHUTDOWN(MESSAGE) ; RETURN
12407 ENDIF
12408
12409 IF (TRIM(QUANTITY)=='SOLID CELL Q-S') STORE_Q_DOT_PPP_S = .TRUE.
12410
12411 IF (TRIM(QUANTITY)=='SUBGRID TEMPERATURE CORRECTION' .AND. .NOT.CORRECT_SUBGRID_TEMPERATURE) THEN
12412 WRITE(MESSAGE, '(A,I0,A)') 'ERROR: Promblem with SCLF ', NN, ', requires CORRECT_SUBGRID_TEMPERATURE=T on MISC'
12413 CALL SHUTDOWN(MESSAGE) ; RETURN
12414 ENDIF
12415
12416 IF (TRIM(QUANTITY)=='DUDT' .OR. TRIM(QUANTITY)=='DVDI' .OR. TRIM(QUANTITY)=='DWDI') STORE_OLD_VELOCITY=.TRUE.
12417
12418 ENDDO SLCF_LOOP
12419
12420 ALLOCATE(M%K_AGL_SLICE(0:IBP1,0:JBP1,N_TERRAIN_SLCF),STAT=IZERO)
12421 CALL ChkMemErr('READ', 'K_AGL_SLICE', IZERO)
12422 M%K_AGL_SLICE = 0
12423 N = 0
12424 DO NN = 1, N_SLCF
12425 SL=>SLICE(NN)
12426 IF (SL%TERRAIN_SLICE) THEN
12427 TERRAIN_CASE = .TRUE.
12428 N = N + 1
12429 M%K_AGL_SLICE(0:IBP1,0:JBP1,N) = INT(SL%SLICE_AGL*M%RDZ(1))
12430 ! Subtract one because bottom of domain will be accounted for when cycling through walls cells
12431 M%K_AGL_SLICE(0:IBP1,0:JBP1,N) = MAX(0,M%K_AGL_SLICE(0:IBP1,0:JBP1,N)-1)
12432 ENDIF

```

```

12433 ENDDO
12434
12435 N.SLCF.MAX = MAX(N.SLCF.MAX,N.SLCF)
12436
12437 IF (VEG_LEVEL_SET) THEN
12438 ALLOCATE(M%LS.Z.TERRAIN(0:IBP1,0:JBP1),STAT=IZERO) ; CALL ChkMemErr('READ','LS.Z.TERRAIN',IZERO)
12439 ENDF
12440
12441 ENDDO MESHLOOP
12442
12443 END SUBROUTINE READ.SLCF
12444
12445
12446 SUBROUTINE READ.BNDF
12447
12448 USE DEVICE_VARIABLES
12449 USE COMP_FUNCTIONS, ONLY : CHANGE_UNITS
12450 INTEGER :: N
12451 LOGICAL :: CELL_CENTERED
12452 CHARACTER(LABEL_LENGTH) :: QUANTITY,PROP_ID,SPEC_ID,PART_ID,STATISTICS
12453 NAMELIST /BNDF/ CELL_CENTERED,FYI,PART_ID,PROP_ID,QUANTITY,SPEC_ID,STATISTICS
12454 TYPE(BOUNDARY_FILE_TYPE), POINTER :: BF=>NULL()
12455
12456 N.BNDF = 0
12457 REWIND(LU.INPUT) ; INPUT_FILE_LINE_NUMBER = 0
12458 COUNT.BNDF.LOOP: DO
12459 CALL CHECKREAD('BNDF',LU.INPUT,IOS)
12460 IF (IOS==1) EXIT COUNT.BNDF.LOOP
12461 READ(LU.INPUT,NML=BNDF,END=209,ERR=210,IOSTAT=IOS)
12462 N.BNDF = N.BNDF + 1
12463 210 IF (IOS>0) THEN
12464 WRITE(MESSAGE,'(A,I0,A,I0)') 'ERROR: Problem with BNDF number ',N.BNDF+1,', line number',INPUT_FILE_LINE_NUMBER
12465 CALL SHUTDOWN(MESSAGE) ; RETURN
12466 ENDF
12467 ENDDO COUNT.BNDF.LOOP
12468 209 REWIND(LU.INPUT) ; INPUT_FILE_LINE_NUMBER = 0
12469
12470 ALLOCATE(BOUNDARY_FILE(N.BNDF),STAT=IZERO)
12471 CALL ChkMemErr('READ','BOUNDARY_FILE',IZERO)
12472
12473 BNDF.TIME_INTEGRALS = 0
12474
12475 READ.BNDF.LOOP: DO N=1,N.BNDF
12476 BF => BOUNDARY_FILE(N)
12477 CELL_CENTERED = .FALSE.
12478 PART_ID = 'null'
12479 PROP_ID = 'null'
12480 SPEC_ID = 'null'
12481 STATISTICS = 'null'
12482 QUANTITY = 'WALLTEMPERATURE'
12483 CALL CHECKREAD('BNDF',LU.INPUT,IOS)
12484 IF (IOS==1) EXIT READ.BNDF.LOOP
12485 READ(LU.INPUT,BNDF)
12486
12487 IF (TRIM(QUANTITY)=='AMPUAZ' .OR. TRIM(QUANTITY)=='CPUAZ' .OR. TRIM(QUANTITY)=='MPUAZ') THEN
12488 IF (N.LP_ARRAY_INDICES == 0) THEN
12489 WRITE(MESSAGE,'(A,I0)') 'ERROR: CPUAZ, MPUAZ, and AMPUAZ require liquid droplets. BNDF line ',N
12490 CALL SHUTDOWN(MESSAGE) ; RETURN
12491 ELSE
12492 IF (.NOT. ALL(LAGRANGIAN_PARTICLE_CLASS%LIQUID_DROPLET)) THEN
12493 WRITE(MESSAGE,'(A,I0)') 'ERROR: CPUAZ, MPUAZ, and AMPUAZ require liquid droplets. BNDF line ',N
12494 CALL SHUTDOWN(MESSAGE) ; RETURN
12495 ENDF
12496 ENDF
12497 ENDF
12498
12499 ! Look to see if output QUANTITY exists
12500
12501 CALL GET_QUANTITY_INDEX(BP%SMOKEVIEW_LABEL,BP%SMOKEVIEW_BAR_LABEL,BP%INDEX,LDUM(1), &
12502 BP%Y_INDEX,BP%Z_INDEX,BP%PART_INDEX,LDUM(2),LDUM(3),LDUM(4),'BNDF', &
12503 QUANTITY,'null',SPEC_ID,PART_ID,'null','null','null')
12504
12505 BP%UNITS = OUTPUT_QUANTITY(BP%INDEX)%UNITS
12506
12507 ! Assign miscellaneous attributes to the boundary file
12508
12509 BP%CELL_CENTERED = CELL_CENTERED
12510
12511 ! Check to see if PROP_ID exists
12512
12513 BP%PROP_INDEX = 0
12514 IF (PROP_ID=='null') CALL GET_PROPERTY_INDEX(BP%PROP_INDEX,'BNDF',PROP_ID)
12515
12516 ! Check to see if the QUANTITY is to be time integrated
12517
12518 IF (STATISTICS=='TIME INTEGRAL') THEN
12519 BNDF.TIME_INTEGRALS = BNDF.TIME_INTEGRALS + 1
12520 BP%TIME_INTEGRAL_INDEX = BNDF.TIME_INTEGRALS

```

Source Code files for edited portions of FDS

```

12521 CALL CHANGE.UNITS(QUANTITY, BP%UNITS, STATISTICS, MYID, LU_ERR)
12522 ENDIF
12523
12524 ENDDO READ.BNDF_LOOP
12525 REWIND(LU.INPUT) ; INPUT_FILE.LINE.NUMBER = 0
12526
12527 END SUBROUTINE READ.BNDF
12528
12529
12530 SUBROUTINE READ.BNDE
12531
12532 USE DEVICE.VARIABLES
12533 INTEGER :: N
12534 LOGICAL :: CELL_CENTERED
12535 CHARACTER(LABEL.LENGTH) :: QUANTITY, PROP_ID, SPEC_ID, PART_ID
12536 NAMELIST /BNDE/ CELL_CENTERED, FYI, PART_ID, PROP_ID, QUANTITY, SPEC_ID
12537 TYPE(BOUNDARY_ELEMENT_FILE_TYPE), POINTER :: BE=>NULL()
12538
12539 N.BNDE = 0
12540 REWIND(LU.INPUT) ; INPUT_FILE.LINE.NUMBER = 0
12541 COUNT.BNDE_LOOP: DO
12542 CALL CHECKREAD('BNDE', LU.INPUT, IOS)
12543 IF (IOS==1) EXIT COUNT.BNDE_LOOP
12544 READ(LU.INPUT, NML=BNDE, END=309, ERR=310, IOSTAT=IOS)
12545 N.BNDE = N.BNDE + 1
12546 310 IF (IOS>0) THEN
12547 WRITE(MESSAGE, '(A,I0,A,I0)') 'ERROR: Problem with BNDE number ', N.BNDE+1, ', line number', INPUT_FILE.LINE.NUMBER
12548 CALL SHUTDOWN(MESSAGE) ; RETURN
12549 ENDIF
12550 ENDDO COUNT.BNDE_LOOP
12551 309 REWIND(LU.INPUT) ; INPUT_FILE.LINE.NUMBER = 0
12552
12553 ALLOCATE(BOUNDARY_ELEMENT_FILE(N.BNDE), STAT=IZERO)
12554 CALL ChkMemErr('READ', 'BOUNDARY_ELEMENT_FILE', IZERO)
12555
12556 READ.BNDE_LOOP: DO N=1, N.BNDE
12557 BE => BOUNDARY_ELEMENT_FILE(N)
12558 CELL_CENTERED = .TRUE.
12559 PART_ID = 'null'
12560 PROP_ID = 'null'
12561 SPEC_ID = 'null'
12562 QUANTITY = 'WALL TEMPERATURE'
12563 CALL CHECKREAD('BNDE', LU.INPUT, IOS)
12564 IF (IOS==1) EXIT READ.BNDE_LOOP
12565 READ(LU.INPUT, BNDE)
12566
12567 ! Look to see if output QUANTITY exists
12568
12569 CALL GET_QUANTITY_INDEX(BE%SMOKEVIEW_LABEL, BE%SMOKEVIEW_BAR_LABEL, BE%INDEX, LDUM(1), &
12570 BE%Y_INDEX, BE%Z_INDEX, BE%PART_INDEX, LDUM(2), LDUM(3), LDUM(4), 'BNDE', &
12571 QUANTITY, 'null', SPEC_ID, PART_ID, 'null', 'null', 'null')
12572
12573 ! Assign miscellaneous attributes to the boundary file
12574
12575 BE%CELL_CENTERED = CELL_CENTERED
12576
12577 ! Check to see if PROP.ID exists
12578
12579 BE%PROP_INDEX = 0
12580 IF (PROP_ID/= 'null') CALL GET_PROPERTY_INDEX(BE%PROP_INDEX, 'BNDE', PROP_ID)
12581
12582 ENDDO READ.BNDE_LOOP
12583 REWIND(LU.INPUT) ; INPUT_FILE.LINE.NUMBER = 0
12584
12585 END SUBROUTINE READ.BNDE
12586
12587
12588 SUBROUTINE CHECK_SURF_NAME(NAME, EXISTS)
12589
12590 LOGICAL, INTENT(OUT) :: EXISTS
12591 CHARACTER(*), INTENT(IN) :: NAME
12592 INTEGER :: NS
12593
12594 EXISTS = .FALSE.
12595 DO NS=0, N_SURF
12596 IF (NAME==SURFACE(NS)%ID) EXISTS = .TRUE.
12597 ENDDO
12598
12599 END SUBROUTINE CHECK_SURF_NAME
12600
12601
12602 SUBROUTINE GET_QUANTITY_INDEX(SMOKEVIEW_LABEL, SMOKEVIEW_BAR_LABEL, OUTPUT_INDEX, OUTPUT2_INDEX, &
12603 Y_INDEX, Z_INDEX, PART_INDEX, DUCT_INDEX, NODE_INDEX, REAC_INDEX, OUTTYPE, &
12604 QUANTITY, QUANTITY2, SPEC_ID_IN, PART_ID, DUCT_ID, NODE_ID, REAC_ID)
12605 CHARACTER(*), INTENT(INOUT) :: QUANTITY
12606 CHARACTER(*), INTENT(OUT) :: SMOKEVIEW_LABEL, SMOKEVIEW_BAR_LABEL
12607 CHARACTER(*) :: SPEC_ID_IN, PART_ID, DUCT_ID, NODE_ID
12608 CHARACTER(LABEL.LENGTH) :: SPEC_ID

```

Source Code files for edited portions of FDS

```

12609 CHARACTER(*), INTENT(IN) :: OUTTYPE,QUANTITY2,REAC_ID
12610 INTEGER, INTENT(OUT) :: OUTPUT_INDEX,Y_INDEX,Z_INDEX,PART_INDEX,DUCT_INDEX,NODE_INDEX,REAC_INDEX,OUTPUT2_INDEX
12611 INTEGER :: ND,NS,NN,NR,N_PLUS,N_MINUS
12612
12613 ! Backward compatibility
12614
12615 IF (QUANTITY=='VEG.TEMPERATURE') QUANTITY='PARTICLE TEMPERATURE'
12616
12617 IF (QUANTITY=='oxygen') THEN
12618 QUANTITY = 'VOLUME FRACTION'
12619 SPEC_ID.IN = 'OXYGEN'
12620 ENDF
12621 IF (QUANTITY=='carbon monoxide') THEN
12622 QUANTITY = 'VOLUME FRACTION'
12623 SPEC_ID.IN = 'CARBON MONOXIDE'
12624 ENDF
12625 IF (QUANTITY=='carbon dioxide') THEN
12626 QUANTITY = 'VOLUME FRACTION'
12627 SPEC_ID.IN = 'CARBON DIOXIDE'
12628 ENDF
12629 IF (QUANTITY=='soot') THEN
12630 QUANTITY = 'VOLUME FRACTION'
12631 SPEC_ID.IN = 'SOOT'
12632 ENDF
12633 IF (QUANTITY=='soot density') THEN
12634 QUANTITY = 'DENSITY'
12635 SPEC_ID.IN = 'SOOT'
12636 ENDF
12637 IF (QUANTITY=='fuel') THEN
12638 QUANTITY = 'VOLUME FRACTION'
12639 WRITE(SPEC_ID.IN,'(A)') REACTION(1)%FUEL
12640 ENDF
12641
12642 DO ND=N_OUTPUT_QUANTITIES,N_OUTPUT_QUANTITIES
12643 IF (QUANTITY==OUTPUT_QUANTITY(ND)%OLDNAME) QUANTITY = OUTPUT_QUANTITY(ND)%NAME
12644 ENDDO
12645
12646 ! Initialize indices
12647
12648 Y_INDEX = -1
12649 Z_INDEX = -1
12650
12651 SPEC_ID = SPEC_ID.IN
12652
12653 IF (QUANTITY=='OPTICAL DENSITY') .AND. SPEC_ID=='null') SPEC_ID='SOOT'
12654 IF (QUANTITY=='EXTINCTION COEFFICIENT') .AND. SPEC_ID=='null') SPEC_ID='SOOT'
12655 IF (QUANTITY=='AEROSOL VOLUME FRACTION') .AND. SPEC_ID=='null') SPEC_ID='SOOT'
12656 IF (QUANTITY=='VISIBILITY') .AND. SPEC_ID=='null') SPEC_ID='SOOT'
12657
12658 PART_INDEX = 0
12659 DUCT_INDEX = 0
12660 NODE_INDEX = 0
12661 OUTPUT2_INDEX = 0
12662 REAC_INDEX = 0
12663
12664 ! Look for the appropriate SPEC or SMIX index
12665
12666 IF (SPEC_ID/='null') THEN
12667 CALL GET_SPEC_OR_SMIX_INDEX(SPEC_ID,Y_INDEX,Z_INDEX)
12668 IF (Z_INDEX>=0 .AND. Y_INDEX>=1) THEN
12669 IF (TRIM(QUANTITY)=='DIFFUSIVITY') THEN
12670 Y_INDEX=-999
12671 ELSE
12672 Z_INDEX=-999
12673 ENDF
12674 ENDF
12675 IF (Z_INDEX<0 .AND. Y_INDEX<1) THEN
12676 WRITE(MESSAGE,'(A,A,A,A)') 'ERROR: SPEC_ID ',TRIM(SPEC_ID),' is not explicitly specified for QUANTITY ',TRIM(
QUANTITY)
12677 CALL SHUTDOWN(MESSAGE) ; RETURN
12678 ENDF
12679 ENDF
12680
12681 ! Assign HVAC indexes
12682
12683 IF (DUCT_ID/='null') THEN
12684 DO ND = 1, N_DUCTS
12685 IF (DUCT_ID==DUCT(ND)%ID) THEN
12686 DUCT_INDEX = ND
12687 EXIT
12688 ENDF
12689 ENDDO
12690 ENDF
12691
12692 IF (NODE_ID/='null') THEN
12693 DO NN = 1, N_DUCTNODES
12694 IF (NODE_ID==DUCTNODE(NN)%ID) THEN
12695 NODE_INDEX = NN

```

```

12696 EXIT
12697 ENDIF
12698 ENDDO
12699 ENDIF
12700
12701 IF (TRIM(QUANTITY)=='FILTER LOADING') THEN
12702 Y_INDEX = -999
12703 DO NS = 1, N_TRACKED_SPECIES
12704 IF (TRIM(SPECIES_MIXTURE(NS)%ID)=='TRIM(SPEC_ID)) THEN
12705 Z_INDEX = NS
12706 EXIT
12707 ENDIF
12708 ENDDO
12709 IF (Z_INDEX < 0) THEN
12710 WRITE(MESSAGE, '(A,A,A)') 'ERROR: FILTER LOADING. ', TRIM(SPEC_ID), ' is not a tracked species'
12711 CALL SHUTDOWN(MESSAGE) ; RETURN
12712 ENDIF
12713 ENDIF
12714
12715
12716 IF (TRIM(QUANTITY)=='EQUILIBRIUM VAPOR FRACTION' .OR. TRIM(QUANTITY)=='EQUILIBRIUM TEMPERATURE') THEN
12717 Y_INDEX = -999
12718 DO NS = 1, N_TRACKED_SPECIES
12719 IF (TRIM(SPECIES_MIXTURE(NS)%ID)=='TRIM(SPEC_ID)) THEN
12720 Z_INDEX = NS
12721 EXIT
12722 ENDIF
12723 ENDDO
12724 IF (Z_INDEX < 0) THEN
12725 WRITE(MESSAGE, '(A,A,A)') 'ERROR: EQUILIBRIUM VAPOR FRACTION. ', TRIM(SPEC_ID), ' is not a tracked species'
12726 CALL SHUTDOWN(MESSAGE) ; RETURN
12727 ENDIF
12728 IF (.NOT. SPECIES_MIXTURE(Z_INDEX)%EVAPORATING) THEN
12729 WRITE(MESSAGE, '(A,A,A)') 'ERROR: EQUILIBRIUM VAPOR FRACTION. ', TRIM(SPEC_ID), ' is not an evaporating species'
12730 CALL SHUTDOWN(MESSAGE) ; RETURN
12731 ENDIF
12732 ENDIF
12733
12734 IF (TRIM(QUANTITY)=='MIXTURE FRACTION') THEN
12735 IF (N_REACTIONS/=1) THEN
12736 WRITE(MESSAGE, '(A)') 'ERROR: MIXTURE FRACTION requires one and only one REAC input'
12737 CALL SHUTDOWN(MESSAGE) ; RETURN
12738 ENDIF
12739 N_PLUS = 0
12740 N_MINUS = 0
12741 DO NN = 1, N_TRACKED_SPECIES
12742 IF (REACTION(1)%NU(NN) > 0) THEN
12743 N_PLUS = N_PLUS + 1
12744 Z_INDEX = NN
12745 ELSEIF (REACTION(1)%NU(NN) < 0) THEN
12746 N_MINUS = N_MINUS + 1
12747 ENDIF
12748 ENDDO
12749 IF (N_PLUS/=1 .AND. N_MINUS/=2) THEN
12750 WRITE(MESSAGE, '(A)') 'ERROR: MIXTURE FRACTION requires REAC of the form A + B -> C'
12751 CALL SHUTDOWN(MESSAGE) ; RETURN
12752 ENDIF
12753 ENDIF
12754
12755 IF (TRIM(QUANTITY)=='HRRPUV REAC') THEN
12756 DO NR = 1, N_REACTIONS
12757 IF (TRIM(REAC_ID)=='TRIM(REACTION(NR)%ID)) REAC_INDEX = NR
12758 ENDDO
12759 IF (REAC_INDEX==0) THEN
12760 WRITE(MESSAGE, '(3A)') 'ERROR: Output QUANTITY ', TRIM(QUANTITY), ' requires a REAC.ID'
12761 CALL SHUTDOWN(MESSAGE) ; RETURN
12762 ENDIF
12763 ENDIF
12764
12765 ! Assign PART_INDEX when PART_ID is specified
12766
12767 IF (PART_ID/'null') THEN
12768 DO NS=1, N_LAGRANGIAN_CLASSES
12769 IF (PART_ID=='LAGRANGIAN.PARTICLE.CLASS(NS)%ID') THEN
12770 PART_INDEX = NS
12771 EXIT
12772 ENDIF
12773 ENDDO
12774 ENDIF
12775
12776 ! Loop over all possible output quantities and assign an index number to match the desired QUANTITY
12777
12778 DO ND=N_OUTPUT_QUANTITIES, N_OUTPUT_QUANTITIES
12779 IF (OUTPUT_QUANTITY(ND)%NAME=='null') CYCLE
12780 IF (QUANTITY2=='OUTPUT_QUANTITY(ND)%NAME') THEN
12781 OUTPUT2_INDEX=ND
12782
12783

```

## Source Code files for edited portions of FDS

```

12784 IF (OUTPUT_QUANTITY(ND)%$SPEC_ID.REQUIRED .AND. (Y_INDEX<1 .AND. Z_INDEX<0)) THEN
12785 WRITE(MESSAGE,'(3A)') 'ERROR: Output QUANTITY2 ',TRIM(QUANTITY2),' requires a SPEC.ID'
12786 CALL SHUTDOWN(MESSAGE) ; RETURN
12787 ENDIF
12788
12789 ! QUANTITY2 only works with SLCF at the moment
12790 IF (.NOT.OUTPUT_QUANTITY(ND)%$SLCF_APPROPRIATE) THEN
12791 WRITE(MESSAGE,'(3A)') 'ERROR: The QUANTITY2 ',TRIM(QUANTITY2),' is not appropriate for SLCF'
12792 CALL SHUTDOWN(MESSAGE) ; RETURN
12793 ENDIF
12794
12795 ENDIF
12796 ENDDO
12797
12798 QUANTITY_INDEX_LOOP: DO ND=-N.OUTPUT_QUANTITIES,N.OUTPUT_QUANTITIES
12799
12800 QUANTITY_IF: IF (QUANTITY==OUTPUT_QUANTITY(ND)%NAME) THEN
12801
12802 OUTPUT_INDEX = ND
12803
12804 IF (OUTPUT_QUANTITY(ND)%QUANTITY2.REQUIRED .AND. OUTPUT2_INDEX==0) THEN
12805 WRITE(MESSAGE,'(3A)') 'ERROR: Output QUANTITY ',TRIM(QUANTITY),' requires a QUANTITY2'
12806 CALL SHUTDOWN(MESSAGE) ; RETURN
12807 ENDIF
12808
12809 IF (OUTPUT_QUANTITY(ND)%$SPEC_ID.REQUIRED .AND. (Y_INDEX<1 .AND. Z_INDEX<0)) THEN
12810 IF (SPEC_ID=='null') THEN
12811 WRITE(MESSAGE,'(3A)') 'ERROR: Output QUANTITY ',TRIM(QUANTITY),' requires a SPEC.ID'
12812 ELSE
12813 WRITE(MESSAGE,'(5A)') 'ERROR: Output QUANTITY ',TRIM(QUANTITY),' . SPEC.ID ',TRIM(SPEC_ID),' not found.'
12814 ENDIF
12815 CALL SHUTDOWN(MESSAGE) ; RETURN
12816 ENDIF
12817
12818 IF (OUTPUT_QUANTITY(ND)%$PART_ID.REQUIRED .AND. PART_INDEX<1) THEN
12819 IF (PART_ID=='null') THEN
12820 WRITE(MESSAGE,'(3A)') 'ERROR: Output QUANTITY ',TRIM(QUANTITY),' requires a PART.ID'
12821 ELSE
12822 WRITE(MESSAGE,'(5A)') 'ERROR: Output QUANTITY ',TRIM(QUANTITY),' . PART.ID ',TRIM(PART_ID),' not found.'
12823 ENDIF
12824 CALL SHUTDOWN(MESSAGE) ; RETURN
12825 ENDIF
12826
12827 IF (OUTPUT_QUANTITY(ND)%$DUCT_ID.REQUIRED .AND. DUCT_INDEX<1) THEN
12828 IF (DUCT_ID=='null') THEN
12829 WRITE(MESSAGE,'(3A)') 'ERROR: Output QUANTITY ',TRIM(QUANTITY),' requires a DUCT.ID'
12830 ELSE
12831 WRITE(MESSAGE,'(5A)') 'ERROR: Output QUANTITY ',TRIM(QUANTITY),' . DUCT.ID ',TRIM(DUCT_ID),' not found.'
12832 ENDIF
12833 CALL SHUTDOWN(MESSAGE) ; RETURN
12834 ENDIF
12835
12836 IF (OUTPUT_QUANTITY(ND)%$NODE_ID.REQUIRED .AND. NODE_INDEX<1) THEN
12837 IF (NODE_ID=='null') THEN
12838 WRITE(MESSAGE,'(3A)') 'ERROR: Output QUANTITY ',TRIM(QUANTITY),' requires a NODE.ID'
12839 ELSE
12840 WRITE(MESSAGE,'(5A)') 'ERROR: Output QUANTITY ',TRIM(QUANTITY),' . NODE.ID ',TRIM(NODE_ID),' not found.'
12841 ENDIF
12842 CALL SHUTDOWN(MESSAGE) ; RETURN
12843 ENDIF
12844
12845 IF (( QUANTITY=='RELATIVE HUMIDITY' .OR. QUANTITY=='HUMIDITY') .AND. H2O_INDEX==0) THEN
12846 WRITE(MESSAGE,'(A)') 'ERROR: RELATIVE HUMIDITY and HUMIDITY require SPEC=WATER VAPOR'
12847 CALL SHUTDOWN(MESSAGE) ; RETURN
12848 END IF
12849
12850 IF (TRIM(QUANTITY)=='DIFFUSIVITY' .AND. DNS .AND. Z_INDEX < 0) THEN
12851 WRITE(MESSAGE,'(A)') 'ERROR: DIFFUSIVITY requires a tracked species SPEC.ID when using DNS'
12852 CALL SHUTDOWN(MESSAGE) ; RETURN
12853 ENDIF
12854
12855 IF (TRIM(QUANTITY)=='SURFACE DEPOSITION') THEN
12856 Y_INDEX = -999
12857 DO NS=1,N.TRACKED_SPECIES
12858 IF (TRIM(SPEC_ID)==TRIM(SPECIES_MIXTURE(NS)%ID)) THEN
12859 Z_INDEX = NS
12860 EXIT
12861 ENDIF
12862 ENDDO
12863 IF (Z_INDEX < 0) THEN
12864 DO NS=1,N.SPECIES
12865 IF (TRIM(SPEC_ID)==TRIM(SPECIES(NS)%ID)) THEN
12866 Y_INDEX = NS
12867 EXIT
12868 ENDIF
12869 ENDDO
12870 IF (Y_INDEX < 0) THEN
12871 WRITE(MESSAGE,'(A,A,A,A)') 'ERROR: SURFACE DEPOSITION for ',TRIM(SPEC_ID),' is invalid as species', &

```



Source Code files for edited portions of FDS

```

12872 ' is not a tracked species'
12873 CALL SHUTDOWN(MESSAGE) ; RETURN
12874 ELSE
12875 IF (SPECIES(Y_INDEX)%MODE /= AEROSOL_SPECIES) THEN
12876 WRITE(MESSAGE, '(A,A,A,A)') 'ERROR: SURFACE DEPOSITION for ', TRIM(SPEC_ID), ' is invalid as species', &
12877 ' is not an aerosol species'
12878 CALL SHUTDOWN(MESSAGE) ; RETURN
12879 ENDIF
12880 IF (SPECIES(Y_INDEX)%AWMINDEX < 0) THEN
12881 N_SURFACE.DENSITY_SPECIES = N_SURFACE.DENSITY_SPECIES + 1
12882 SPECIES(Y_INDEX)%AWMINDEX = N_SURFACE.DENSITY_SPECIES
12883 ENDIF
12884 ENDIF
12885 ELSEIF (Z_INDEX==0) THEN
12886 WRITE(MESSAGE, '(A)') 'ERROR: Cannot select background species for deposition'
12887 CALL SHUTDOWN(MESSAGE) ; RETURN
12888 ELSE
12889 IF (.NOT. SPECIES_MIXTURE(Z_INDEX)%DEPOSITING) THEN
12890 WRITE(MESSAGE, '(A,A,A)') 'ERROR: SURFACE DEPOSITION for ', TRIM(SPEC_ID), ' is not an aerosol tracked species'
12891 CALL SHUTDOWN(MESSAGE) ; RETURN
12892 ENDIF
12893 IF (SPECIES_MIXTURE(Z_INDEX)%AWMINDEX < 0) THEN
12894 N_SURFACE.DENSITY_SPECIES = N_SURFACE.DENSITY_SPECIES + 1
12895 SPECIES_MIXTURE(Z_INDEX)%AWMINDEX = N_SURFACE.DENSITY_SPECIES
12896 ENDIF
12897 ENDIF
12898 ENDIF
12899
12900 IF (TRIM(QUANTITY)== 'MPUV.Z' .OR. TRIM(QUANTITY)== 'ADD.Z' .OR. TRIM(QUANTITY)== 'ADT.Z' .OR. TRIM(QUANTITY)== '
12901 ADA.Z' .OR. &
12902 TRIM(QUANTITY)== 'QABS.Z' .OR. TRIM(QUANTITY)== 'QSCA.Z' .OR. TRIM(QUANTITY)== 'MPUA.Z' .OR. TRIM(QUANTITY)== 'CPUA.Z
12903 ' .OR. &
12904 TRIM(QUANTITY)== 'AMPUA.Z') THEN
12905 IF (N_LAGRANGIAN_CLASSES==0) THEN
12906 WRITE(MESSAGE, '(3A)') 'ERROR: The QUANTITY ', TRIM(QUANTITY), ' requires liquid droplets'
12907 CALL SHUTDOWN(MESSAGE) ; RETURN
12908 ELSE
12909 IF (.NOT. ALL(LAGRANGIAN_PARTICLE_CLASS%LIQUID_DROPLET)) THEN
12910 WRITE(MESSAGE, '(3A)') 'ERROR: The QUANTITY ', TRIM(QUANTITY), ' requires liquid droplets'
12911 CALL SHUTDOWN(MESSAGE) ; RETURN
12912 ENDIF
12913 ENDIF
12914 ENDIF
12915 ENDIF
12916
12917 SELECT CASE (TRIM(OUTTYPE))
12918 CASE ('SLCF')
12919 ! Throw out bad slices
12920 IF (.NOT. OUTPUT_QUANTITY(ND)%SLCF_APPROPRIATE) THEN
12921 WRITE(MESSAGE, '(3A)') 'ERROR: The QUANTITY ', TRIM(QUANTITY), ' is not appropriate for SLCF'
12922 CALL SHUTDOWN(MESSAGE) ; RETURN
12923 ENDIF
12924 CASE ('DEVC')
12925 IF (.NOT. OUTPUT_QUANTITY(ND)%DEVC_APPROPRIATE) THEN
12926 WRITE(MESSAGE, '(3A)') 'ERROR: The QUANTITY ', TRIM(QUANTITY), ' is not appropriate for DEVC'
12927 CALL SHUTDOWN(MESSAGE) ; RETURN
12928 ENDIF
12929 CASE ('PART')
12930 IF (.NOT. OUTPUT_QUANTITY(ND)%PART_APPROPRIATE) THEN
12931 WRITE(MESSAGE, '(3A)') 'ERROR: ', TRIM(QUANTITY), ' is not a particle output QUANTITY'
12932 CALL SHUTDOWN(MESSAGE) ; RETURN
12933 ENDIF
12934 CASE ('BNDF')
12935 IF (.NOT. OUTPUT_QUANTITY(ND)%BNDF_APPROPRIATE) THEN
12936 WRITE(MESSAGE, '(3A)') 'ERROR: The QUANTITY ', TRIM(QUANTITY), ' is not appropriate for BNDF'
12937 CALL SHUTDOWN(MESSAGE) ; RETURN
12938 ENDIF
12939 CASE ('BNDE')
12940 IF (QUANTITY== 'AMPUA' .OR. QUANTITY== 'AMPUA.Z') ACCUMULATE_WATER = .TRUE.
12941 IF (.NOT. OUTPUT_QUANTITY(ND)%BNDE_APPROPRIATE) THEN
12942 WRITE(MESSAGE, '(3A)') 'ERROR: The QUANTITY ', TRIM(QUANTITY), ' is not appropriate for BNDE'
12943 CALL SHUTDOWN(MESSAGE) ; RETURN
12944 ENDIF
12945 CASE ('ISOF')
12946 IF (QUANTITY== 'AMPUA' .OR. QUANTITY== 'AMPUA.Z') ACCUMULATE_WATER = .TRUE.
12947 IF (.NOT. OUTPUT_QUANTITY(ND)%ISOF_APPROPRIATE) THEN
12948 WRITE(MESSAGE, '(3A)') 'ERROR: ISOF quantity ', TRIM(QUANTITY), ' not appropriate for isosurface'
12949 CALL SHUTDOWN(MESSAGE) ; RETURN
12950 ENDIF
12951 CASE ('PLOT3D')
12952 IF (OUTPUT_QUANTITY(ND)%SOLID_PHASE) THEN
12953 WRITE(MESSAGE, '(5A)') 'ERROR: ', TRIM(OUTTYPE), ' QUANTITY ', TRIM(QUANTITY), ' not appropriate for gas phase'
12954 CALL SHUTDOWN(MESSAGE) ; RETURN
12955 ENDIF
12956 IF (.NOT. OUTPUT_QUANTITY(ND)%SLCF_APPROPRIATE) THEN
12957 WRITE(MESSAGE, '(5A)') 'ERROR: ', TRIM(OUTTYPE), ' QUANTITY ', TRIM(QUANTITY), ' not appropriate for Plot3D'
12958 CALL SHUTDOWN(MESSAGE) ; RETURN
12959 ENDIF

```

Source Code files for edited portions of FDS

```

12958 CASE ('SMOKE3D')
12959 IF (SMOKE3D .AND. (.NOT.OUTPUT_QUANTITY(ND)%MASS_FRACTION .AND. ND/=11)) THEN
12960 WRITE(MESSAGE,'(5A)') 'ERROR: ',TRIM(OUTTYPE),' QUANTITY ',TRIM(QUANTITY), ' must be a mass fraction'
12961 CALL SHUTDOWN(MESSAGE) ; RETURN
12962 ENDIF
12963 CASE DEFAULT
12964 END SELECT
12965
12966 ! Assign Smokeview Label
12967
12968 IF (Z_INDEX>=0) THEN
12969 IF (TRIM(QUANTITY)='MIXTURE_FRACTION') THEN
12970 SMOKEVIEW_LABEL = TRIM(QUANTITY)
12971 SMOKEVIEW_BAR_LABEL = TRIM(OUTPUT_QUANTITY(ND)%SHORT_NAME)
12972 ELSE
12973 SMOKEVIEW_LABEL = TRIM(SPECIES_MIXTURE(Z_INDEX)%ID)//' '//TRIM(QUANTITY)
12974 SMOKEVIEW_BAR_LABEL = TRIM(OUTPUT_QUANTITY(ND)%SHORT_NAME)//'-'//TRIM(SPECIES_MIXTURE(Z_INDEX)%ID)
12975 ENDIF
12976 ELSEIF (Y_INDEX>0) THEN
12977 SMOKEVIEW_LABEL = TRIM(SPECIES(Y_INDEX)%ID)//' '//TRIM(QUANTITY)
12978 SMOKEVIEW_BAR_LABEL = TRIM(OUTPUT_QUANTITY(ND)%SHORT_NAME)//'-'//TRIM(SPECIES(Y_INDEX)%FORMULA)
12979 ELSEIF (PART_INDEX>0) THEN
12980 SMOKEVIEW_LABEL = TRIM(LAGRANGIAN_PARTICLE_CLASS(PART_INDEX)%ID)//' '//TRIM(QUANTITY)
12981 SMOKEVIEW_BAR_LABEL = TRIM(OUTPUT_QUANTITY(ND)%SHORT_NAME)
12982 ELSEIF (OUTPUT2_INDEX/=0) THEN
12983 SMOKEVIEW_LABEL = TRIM(QUANTITY)//' '//TRIM(QUANTITY2)
12984 SMOKEVIEW_BAR_LABEL = TRIM(OUTPUT_QUANTITY(ND)%SHORT_NAME)//' '//TRIM(OUTPUT_QUANTITY(OUTPUT2_INDEX)%SHORT_NAME)
12985 ELSEIF (REAC_INDEX/=0) THEN
12986 SMOKEVIEW_LABEL = TRIM(QUANTITY)//' '//TRIM(REACTION(REAC_INDEX)%ID)
12987 SMOKEVIEW_BAR_LABEL = TRIM(OUTPUT_QUANTITY(ND)%SHORT_NAME)//' '//TRIM(REACTION(REAC_INDEX)%ID)
12988 ELSE
12989 SMOKEVIEW_LABEL = TRIM(QUANTITY)
12990 SMOKEVIEW_BAR_LABEL = TRIM(OUTPUT_QUANTITY(ND)%SHORT_NAME)
12991 ENDIF
12992 RETURN
12993 ENDIF QUANTITY_IF
12994
12995 ENDDO QUANTITY_INDEX_LOOP
12996
12997 ! If no match for desired QUANTITY is found, stop the job
12998
12999 WRITE(MESSAGE,'(5A)') 'ERROR: ',TRIM(OUTTYPE),' QUANTITY ',TRIM(QUANTITY), ' not found'
13000 CALL SHUTDOWN(MESSAGE) ; RETURN
13001
13002 END SUBROUTINE GET_QUANTITY_INDEX
13003
13004 SUBROUTINE GET_SPEC_OR_SMIX_INDEX(SPEC_ID,Y_INDX,Z_INDX)
13005
13006 ! Find the appropriate SPEC or SMIX index for the given SPEC_ID
13007
13008 CHARACTER(*) , INTENT(IN) :: SPEC_ID
13009 INTEGER , INTENT(OUT) :: Y_INDX,Z_INDX
13010 INTEGER :: NS
13011
13012 Y_INDX = -999
13013 Z_INDX = -999
13014
13015 DO NS=1,N_SPECIES
13016 IF (TRIM(SPEC_ID)==TRIM(SPECIES(NS)%ID)) THEN
13017 Y_INDX = NS
13018 EXIT
13019 ENDIF
13020 ENDDO
13021
13022 DO NS=1,N_TRACKED_SPECIES
13023 IF (TRIM(SPEC_ID)==TRIM(SPECIES_MIXTURE(NS)%ID)) THEN
13024 Z_INDX = NS
13025 RETURN
13026 ENDIF
13027 ENDDO
13028
13029 END SUBROUTINE GET_SPEC_OR_SMIX_INDEX
13030
13031 SUBROUTINE GET_PROPERTY_INDEX(P_INDEX,OUTTYPE,PROP_ID)
13032
13033 USE DEVICE_VARIABLES
13034 CHARACTER(*) , INTENT(IN) :: PROP_ID
13035 CHARACTER(*) , INTENT(IN) :: OUTTYPE
13036 INTEGER , INTENT(INOUT) :: P_INDEX
13037 INTEGER :: NN
13038
13039 DO NN=1,N_PROP
13040 IF (PROP_ID==PROPERTY(NN)%ID) THEN
13041 P_INDEX = NN

```

Source Code files for edited portions of FDS

```

13046 SELECT CASE (TRIM(OUTTYPE))
13047 CASE ('SLCF')
13048 CASE ('DEVC')
13049 CASE ('PART')
13050 CASE ('OBST')
13051 CASE ('BNDF')
13052 CASE ('PLOT3D')
13053 CASE DEFAULT
13054 END SELECT
13055 RETURN
13056 ENDIF
13057 ENDDO
13058
13059 WRITE(MESSAGE,'(5A)') 'ERROR: ',TRIM(OUTTYPE),' PROP_ID ',TRIM(PROP_ID),' not found'
13060 CALL SHUTDOWN(MESSAGE) ; RETURN
13061
13062 END SUBROUTINE GET_PROPERTY_INDEX
13063
13064 SUBROUTINE READ.CSVF
13065 USE OUTPUT.DATA
13066
13067 CHARACTER(256) :: CSVFILE,UVWFILE='null'
13068 NAMELIST /CSVF/ CSVFILE,UVWFILE
13069
13070 N.CSVF=0
13071 REWIND(LU.INPUT) ; INPUT_FILE.LINE_NUMBER = 0
13072 COUNT.CSVF.LOOP: DO
13073 CALL CHECKREAD('CSVF',LU.INPUT,IOS)
13074 IF (IOS==1) EXIT COUNT.CSVF.LOOP
13075 READ(LU.INPUT,NML=CSVF,END=16,ERR=17,IOSTAT=IOS)
13076 N.CSVF=N.CSVF+1
13077 16 IF (IOS>0) THEN ; CALL SHUTDOWN('ERROR: problem with CSVF line') ; RETURN ; ENDIF
13078 ENDDO COUNT.CSVF.LOOP
13079 17 REWIND(LU.INPUT) ; INPUT_FILE.LINE_NUMBER = 0
13080
13081 IF (N.CSVF==0) RETURN
13082
13083 ! Allocate CSVFINFO array
13084
13085 ALLOCATE(CSVFINFO(N.CSVF),STAT=IZERO)
13086 CALL ChkMemErr('READ','CSVF',IZERO)
13087
13088 READ.CSVF.LOOP: DO I=1,N.CSVF
13089
13090 CALL CHECKREAD('CSVF',LU.INPUT,IOS)
13091 IF (IOS==1) EXIT READ.CSVF.LOOP
13092
13093 ! Read the CSVF line
13094
13095 READ(LU.INPUT,CSVF,END=37)
13096
13097 CSVFINFO(I)%CSVFILE = TRIM(CSVFILE)
13098 IF (TRIM(UVWFILE)/='null'.AND. I>NMESHES) THEN
13099 CALL SHUTDOWN('Problem with CSVF line: UVWFILE must be in order with MESH.') ; RETURN
13100 ELSE
13101 CSVFINFO(I)%UVWFILE = UVWFILE
13102 UVW.RESTART = .TRUE.
13103 ENDIF
13104
13105 ENDDO READ.CSVF.LOOP
13106 37 REWIND(LU.INPUT) ; INPUT_FILE.LINE_NUMBER = 0
13107
13108 END SUBROUTINE READ.CSVF
13109
13110
13111 SUBROUTINE CALC.H2O.HV
13112 USE PROPERTY_DATA, ONLY: JANAF_TABLE,JANAF_TABLE.LIQUID
13113 CHARACTER(LABEL_LENGTH) :: WATER_VAPOR='WATER VAPOR'
13114 INTEGER :: I
13115 REAL(EB) :: CP.G,CP.G.O,CP.L,CP.L.O,H.G,H.L,H.G.O,H.L.O,G.F,RCON,H.V,T.R,T.M,T.B,DENSITY,MU.LIQUID,K.LIQUID,
13116 BETA.LIQUID
13117 LOGICAL :: FUEL
13118
13119 DO I=0,5000
13120 CALL JANAF_TABLE (I,CP.G,H.G.O,WATER_VAPOR,RCON,FUEL,G.F)
13121 CALL JANAF_TABLE.LIQUID (I,CP.L,H.V,H.L.O,T.R,T.M,T.B,WATER_VAPOR,FUEL,DENSITY,MU.LIQUID,K.LIQUID,BETA.LIQUID)
13122 IF (I==0) THEN
13123 H.G = H.G.O
13124 H.L = H.L.O
13125 ELSE
13126 H.G = H.G + 0.5_EB*(CP.G+CP.G.O)
13127 H.L = H.L + 0.5_EB*(CP.L+CP.L.O)
13128 ENDIF
13129 H.V.H2O(I) = H.G-H.L
13130 CP.L.O=CP.L
13131 END DO
13132

```

```

13133 END SUBROUTINE CALC_H2O_HV
13134
13135 !Sesa
13136 SUBROUTINE read_pen
13137
13138 integer :: I1
13139
13140 !creating a namelist group name PENA
13141 NAMELIST /PENA/ penalizationParameter, blendingParameter, dampingParameter, &
13142 penXmin, penXmax, penYmin, penYmax, penZmin, penZmax, &
13143 mX, mY, mZ, b, dataFileName, pena.I, pena.J, pena.K!,penU0, penV0, penW0,
13144
13145 !opening the input file and reading the PENA namelist
13146 OPEN(LU_INPUT,FILE=FN_INPUT,ACTION='READ')
13147
13148 npen=0
13149 pendat_size=0
13150
13151 !counting the number of penalisation regions mentioned in input file
13152 COUNT.PEN_LOOP: DO
13153
13154 CALL CHECKREAD('PENA',LU_INPUT,IOS)
13155 IF (IOS==1) EXIT COUNT.PEN_LOOP
13156 READ(LU_INPUT,NML=PENA,END=66,ERR=17,IOSTAT=IOS)
13157 npen=npen+1
13158 pendat_size(npen)=(pena.I +1)*(pena.J +1)*(pena.K +1)
13159 ENDDO COUNT.PEN_LOOP
13160
13161 66 if (npen.gt.0) allocate(pendat(npen,(15+3*(maxval(pendat_size))))))
13162
13163 pendat=0d0
13164
13165 REWIND(LU_INPUT)
13166
13167 !reading the penalisation regions
13168 READ.PEN_LOOP: DO I1=1, npen
13169
13170 !make sure everything is read fresh
13171 penalizationParameter=0d0
13172 penXmin=0d0
13173 penXmax=0d0
13174 penYmin=0d0
13175 penYmax=0d0
13176 penZmin=0d0
13177 penZmax=0d0
13178 penU0=0d0
13179 penV0=0d0
13180 penW0=0d0
13181 blendingParameter=0d0
13182 mX=0d0
13183 mY=0d0
13184 mZ=0d0
13185 b=0d0
13186 dampingParameter=0d0
13187
13188 READ(LU_INPUT,NML=PENA,END=16,ERR=17,IOSTAT=IOS)
13189 !opening and reading the csv files
13190 OPEN(UNIT=fileread,FILE=dataFileName,STATUS='OLD',FORM='FORMATTED',ACTION='READ',IOSTAT=IERROR)
13191
13192 IF (IERROR/=0) THEN
13193 MESSAGE = 'ERROR: Problem with the csv file '
13194 CALL SHUTDOWN(MESSAGE)
13195 RETURN
13196 ENDIF
13197
13198 READ(fileread,*) !skipping the first line
13199 READ(fileread,*) penXmin,penXmax,penYmin,penYmax,penZmin,penZmax,mX,mY,mZ,b,pena.I,pena.J,pena.K
13200 READ(fileread,*) timestep
13201
13202 if (ALLOCATED(penU0)) deallocate(penU0)
13203 allocate(penU0((pena.I+1),(pena.J+1),(pena.K+1)))
13204
13205 if (ALLOCATED(penV0)) deallocate(penV0)
13206 allocate(penV0((pena.I+1),(pena.J+1),(pena.K+1)))
13207
13208 if (ALLOCATED(penW0)) deallocate(penW0)
13209 allocate(penW0((pena.I+1),(pena.J+1),(pena.K+1)))
13210
13211 DO penZ=1,pena.K+1
13212 DO penY=1,pena.J+1
13213 DO penX=1,pena.I+1
13214 READ(fileread,*,IOSTAT=IERROR) penU0(penX,penY,penZ),penV0(penX,penY,penZ),penW0(penX,penY,penZ)
13215
13216 IF (IERROR/=0) THEN
13217 penU0(penX,penY,penZ)=0..EB
13218 penV0(penX,penY,penZ)=0..EB
13219 penW0(penX,penY,penZ)=0..EB
13220 ENDIF

```

```

13221 ENDDO
13222 ENDDO
13223 ENDDO
13224
13225
13226 pendat(11,:)=(/ penalizationParameter, blendingParameter, dampingParameter, &
13227 penXmin, penXmax, penYmin, penYmax, penZmin, penZmax, &
13228 mX, mY, mZ, b, timestep, &
13229 penU0(:,:,:), penV0(:,:,:), penW0(:,:,:)/)
13230
13231 16 IF (IOS>0) THEN ; CALL SHUTDOWN('ERROR: problem with PENALIZATION line') ; RETURN ;ENDIF
13232
13233 ENDDO READ.PEN.LOOP
13234
13235 close(fileread)
13236 17 close(LU.INPUT)
13237
13238 END SUBROUTINE read_pen
13239
13240 SUBROUTINE read_trunks
13241 !open the input file again look for the &trnk line
13242 !read it and close the file
13243 namelist /trnk/ ntrunks, trunks, eta, trnk_min, trnk_max
13244 namelist /tloc/ trnk_loc
13245
13246 !open the file with the trunk details in it
13247 OPEN(LU.INPUT,FILE=FN.INPUT,ACTION='READ')
13248
13249 COUNT.TRUNK.LOOP: DO
13250 CALL CHECKREAD('TRNK',LU.INPUT,IOS)
13251 IF (IOS==1) EXIT COUNT.TRUNK.LOOP
13252 READ(LU.INPUT,NML=trnk,END=16,ERR=17,IOSTAT=IOS)
13253 16 IF (IOS>0) THEN ; CALL SHUTDOWN('ERROR: problem with TRNK line') ; RETURN ;ENDIF
13254
13255 ENDDO COUNT.TRUNK.LOOP
13256
13257 rewind(LU.INPUT)
13258 allocate(trnk_loc(2,ntrunks))
13259
13260 GET.TRUNK.LOOP: DO
13261 CALL CHECKREAD('TLOC',LU.INPUT,IOS)
13262 IF (IOS==1) EXIT GET.TRUNK.LOOP
13263 READ(LU.INPUT,NML=tloc,END=18,ERR=17,IOSTAT=IOS)
13264 18 IF (IOS>0) THEN ; CALL SHUTDOWN('ERROR: problem with TLOC line') ; RETURN ;ENDIF
13265 ENDDO GET.TRUNK.LOOP
13266
13267 17 close(LU.INPUT)
13268
13269 END SUBROUTINE read_trunks
13270
13271 END MODULE READ.INPUT
13272

```

### A.3 *velo.f90*

```

1 |
2 | MODULE VELO
3 |
4 | ! Module computes the velocity flux terms, baroclinic torque correction terms, and performs the CFL Check
5 |
6 | USE PRECISION.PARAMETERS
7 | USE GLOBAL.CONSTANTS
8 | USE MESH.POINTERS
9 | USE COMP.FUNCTIONS, ONLY: SECOND
10 |
11 | IMPLICIT NONE
12 | PRIVATE
13 |
14 | PUBLIC COMPUTE.VELOCITY.FLUX, VELOCITY.PREDICTOR, VELOCITY.CORRECTOR, NO.FLUX, BAROCLINIC.CORRECTION, &
15 | MATCH.VELOCITY, MATCH.VELOCITY.FLUX, VELOCITY_BC, COMPUTE.VISCOSITY, VISCOSITY_BC
16 | PRIVATE VELOCITY.FLUX, VELOCITY.FLUX.CYLINDRICAL
17 |
18 |
19 | CONTAINS
20 |
21 |
22 | SUBROUTINE COMPUTE.VELOCITY.FLUX(T,DT,NM,FUNCTION.CODE)
23 |
24 | REAL(EB), INTENT(IN) :: T,DT
25 | REAL(EB) :: TNOW
26 | INTEGER, INTENT(IN) :: NM,FUNCTION.CODE
27 |
28 | IF (SOLID.PHASE.ONLY .OR. FREEZE.VELOCITY) RETURN
29 |

```

```

30 TNOW = SECOND()
31
32 SELECT CASE(FUNCTION_CODE)
33 CASE(1)
34 IF (PREDICTOR .OR. COMPUTE_VISCOSITY_TWICE) CALL COMPUTE_VISCOSITY(T,NM)
35 CASE(2)
36 BAROCLINIC_TERMS_ATTACHED = .FALSE.
37 IF (PREDICTOR .OR. COMPUTE_VISCOSITY_TWICE) CALL VISCOSITY_BC(NM)
38 IF (.NOT. CYLINDRICAL) CALL VELOCITY_FLUX(T,DT,NM)
39 IF ( CYLINDRICAL) CALL VELOCITY_FLUX_CYLINDRICAL(T,NM)
40 END SELECT
41
42 T_USED(4) = T_USED(4) + SECOND() - TNOW
43 END SUBROUTINE COMPUTE_VELOCITY_FLUX
44
45
46 SUBROUTINE COMPUTE_VISCOSITY(T,NM)
47
48 USE PHYSICAL_FUNCTIONS, ONLY: GET_VISCOSITY, LES_FILTER_WIDTH_FUNCTION, GET_POTENTIAL_TEMPERATURE
49 USE TURBULENCE, ONLY: VARDEN_DYNMAG, TEST_FILTER, FILL_EDGES, WALL_MODEL, RNG_EDDY_VISCOSITY, WALE_VISCOSITY
50 USE MATH_FUNCTIONS, ONLY: EVALUATE_RAMP
51 REAL(EB), INTENT(IN) :: T
52 INTEGER, INTENT(IN) :: NM
53 REAL(EB) :: ZZ_GET(1:N_TRACKED_SPECIES), NU_EDDY, DELTA, KSGS, U2, V2, W2, AA, A_IJ(3,3), BB, B_IJ(3,3), &
54 DUDX, DUDY, DUDZ, DVDX, DVDY, DVDZ, DWDX, DWDY, DWDZ, MU_EFF, SLIP_COEF, VEL_GAS, VEL_T, RAMP_T, TSI, &
55 VDF, LS, THETA_0, THETA_1, THETA_2, DTDZBAR, WGT
56 REAL(EB), PARAMETER :: RAPLUS=1..EB/26..EB, C_LS=0.76_EB
57 INTEGER :: I, J, K, IIG, JIG, KKG, II, JJ, KK, IW, IOR
58 REAL(EB), POINTER, DIMENSION(:,:,:) :: RHOP=>NULL(), UP=>NULL(), VP=>NULL(), WP=>NULL(), &
59 UP_HAT=>NULL(), VP_HAT=>NULL(), WP_HAT=>NULL(), &
60 UV=>NULL(), VW=>NULL(), WW=>NULL(), DIDZ=>NULL()
61 REAL(EB), POINTER, DIMENSION(:,:,:) :: ZZP=>NULL()
62 INTEGER, POINTER, DIMENSION(:,:,:) :: CELL_COUNTER=>NULL()
63 TYPE(WALL_TYPE), POINTER :: WC=>NULL()
64 TYPE(SURFACE_TYPE), POINTER :: SF=>NULL()
65
66 IF (EVACUATION_ONLY(NM)) RETURN ! No need to update viscosity, use initial one
67
68 CALL POINT_TO_MESH(NM)
69
70 IF (PREDICTOR) THEN
71 RHOP => RHO
72 UV => U
73 VW => V
74 WW => W
75 ZZP => ZZ
76 ELSE
77 RHOP => RHOS
78 UV => US
79 VW => VS
80 WW => WS
81 ZZP => ZS
82 ENDIF
83
84 ! Compute viscosity for DNS using primitive species
85
86 !$OMP PARALLEL DO FIRSTPRIVATE(ZZ_GET) SCHEDULE(guided)
87 DO K=1,KBAR
88 DO J=1,JBAR
89 DO I=1,IBAR
90 IF (SOLID(CELL_INDEX(I,J,K))) CYCLE
91 ZZ_GET(1:N_TRACKED_SPECIES) = ZZP(I,J,K,1:N_TRACKED_SPECIES)
92 CALL GET_VISCOSITY(ZZ_GET, MU_DNS(I,J,K), TMP(I,J,K))
93 ENDDO
94 ENDDO
95 ENDDO
96 !$OMP END PARALLEL DO
97
98 CALL COMPUTE_STRAIN_RATE(NM)
99
100 SELECT_TURB: SELECT CASE (TURB_MODEL)
101
102 CASE (NO_TURB_MODEL)
103
104 MU = MU_DNS
105
106 CASE (CONSMAG, DYNMAG) SELECT_TURB ! Smagorinsky (1963) eddy viscosity
107
108 IF (PREDICTOR .AND. TURB_MODEL/=DYNMAG) CALL VARDEN_DYNMAG(NM) ! dynamic procedure, Moin et al. (1991)
109
110 DO K=1,KBAR
111 DO J=1,JBAR
112 DO I=1,IBAR
113 IF (SOLID(CELL_INDEX(I,J,K))) CYCLE
114 MU(I,J,K) = MU_DNS(I,J,K) + RHOP(I,J,K)*CSD2(I,J,K)*STRAIN_RATE(I,J,K)
115 ENDDO
116 ENDDO
117 ENDDO

```

Source Code files for edited portions of FDS

```

118
119 CASE (DEARDORFF) SELECT_TURB ! Deardorff (1980) eddy viscosity model (current default)
120
121 ! Velocities relative to the p-cell center
122
123 UP => WORK1
124 VP => WORK2
125 WP => WORK3
126 UP=0._EB
127 VP=0._EB
128 WP=0._EB
129
130 DO K=1,KBAR
131 DO J=1,JBAR
132 DO I=1,IBAR
133 IF (SOLID(CELL_INDEX(I,J,K))) CYCLE
134 UP(I,J,K) = 0.5_EB*(UU(I,J,K) + UU(I-1,J,K))
135 VP(I,J,K) = 0.5_EB*(VV(I,J,K) + VV(I,J-1,K))
136 WP(I,J,K) = 0.5_EB*(WW(I,J,K) + WW(I,J,K-1))
137 ENDDO
138 ENDDO
139 ENDDO
140
141 ! fill mesh boundary ghost cells
142
143 DO IW=1,N_EXTERNAL_WALL_CELLS
144 WC=>WALL(IW)
145 SELECT CASE(WC%BOUNDARY_TYPE)
146 CASE(INTERPOLATED_BOUNDARY)
147 II = WC%ONE_D%II
148 JJ = WC%ONE_D%JJ
149 KK = WC%ONE_D%KK
150 UP(II,JJ,KK) = U_GHOST(IW)
151 VP(II,JJ,KK) = V_GHOST(IW)
152 WP(II,JJ,KK) = W_GHOST(IW)
153 CASE(OPEN_BOUNDARY,MIRROR_BOUNDARY)
154 II = WC%ONE_D%II
155 JJ = WC%ONE_D%JJ
156 KK = WC%ONE_D%KK
157 IIG = WC%ONE_D%IIG
158 JJG = WC%ONE_D%JJG
159 KKG = WC%ONE_D%KKG
160 UP(II,JJ,KK) = UP(IIG,JJG,KKG)
161 VP(II,JJ,KK) = VP(IIG,JJG,KKG)
162 WP(II,JJ,KK) = WP(IIG,JJG,KKG)
163 END SELECT
164 ENDDO
165
166 ! fill edge and corner ghost cells
167
168 CALL FILL_EDGES(UP)
169 CALL FILL_EDGES(VP)
170 CALL FILL_EDGES(WP)
171
172 UP.HAT => WORK4
173 VP.HAT => WORK5
174 WP.HAT => WORK6
175 UP.HAT=0._EB
176 VP.HAT=0._EB
177 WP.HAT=0._EB
178
179 CALL TEST_FILTER(UP,HAT,UP)
180 CALL TEST_FILTER(VP,HAT,VP)
181 CALL TEST_FILTER(WP,HAT,WP)
182
183 POTENTIAL_TEMPERATURE_IF: IF (.NOT.POTENTIAL_TEMPERATURE_CORRECTION) THEN
184 !$OMP PARALLEL DO PRIVATE(Delta, KSGS, NUEDDY) SCHEDULE(static)
185 DO K=1,KBAR
186 DO J=1,JBAR
187 DO I=1,IBAR
188 IF (SOLID(CELL_INDEX(I,J,K))) CYCLE
189 Delta = LES_FILTER_WIDTH_FUNCTION(DX(I),DY(J),DZ(K))
190 KSGS = 0.5_EB*( (UP(I,J,K)-UP.HAT(I,J,K))**2 + (VP(I,J,K)-VP.HAT(I,J,K))**2 + (WP(I,J,K)-WP.HAT(I,J,K))**2 )
191 NUEDDY = C_DEARDORFF*Delta*SQR(KSGS)
192 MU(I,J,K) = MU_DNS(I,J,K) + RHOP(I,J,K)*NUEDDY
193 ENDDO
194 ENDDO
195 ENDDO
196 !$OMP END PARALLEL DO
197 ELSE POTENTIAL_TEMPERATURE_IF
198 DTDZ => WORK7
199 DTDZ = 0._EB
200 DO K=0,KBAR
201 DO J=0,JBAR
202 DO I=0,IBAR
203 THETA.1 = GET_POTENTIAL_TEMPERATURE(TMP(I,J,K),ZC(K))
204 THETA.2 = GET_POTENTIAL_TEMPERATURE(TMP(I,J,K+1),ZC(K+1))
205 DTDZ(I,J,K) = (THETA.2-THETA.1)*RDZN(K)

```

```

206 ENDDO
207 ENDDO
208 ENDDO
209 DO K=1,KBAR
210 DO J=1,JBAR
211 DO I=1,IBAR
212 IF (SOLID(CELL_INDEX(I,J,K))) CYCLE
213 DELTA = LES_FILTER_WIDTH_FUNCTION(DX(I),DY(J),DZ(K))
214 LS = DELTA
215 KSGS = 0.5*_EB*( (UP(I,J,K)-UP_HAT(I,J,K))**2 + (VP(I,J,K)-VP_HAT(I,J,K))**2 + (WP(I,J,K)-WP_HAT(I,J,K))**2 )
216 DTDZBAR = 0.5*_EB*(DTDZ(I,J,K)+DTDZ(I,J,K+1))
217 IF (DTDZBAR>0._EB) THEN
218 THETA_0 = GET_POTENTIAL_TEMPERATURE(TMP_0(K),ZC(K))
219 LS = C.LS*SQRT(KSGS)/SQRT(ABS(GVEC(3))/THETA_0*DTDZBAR) ! von Schoenberg Eq. (3.19)
220 ENDF
221 NUEDDY = C.DEARDORFF*MIN(LS,DELTA)*SQRT(KSGS)
222 MU(I,J,K) = MUDNS(I,J,K) + RHOP(I,J,K)*NUEDDY
223 ENDDO
224 ENDDO
225 ENDDO
226 ENDF POTENTIAL_TEMPERATURE_IF
227
228 CASE (VREMAN) SELECT_TURB ! Vreman (2004) eddy viscosity model (experimental)
229
230 ! A. W. Vreman. An eddy-viscosity subgrid-scale model for turbulent shear flow: Algebraic theory and applications
231 ! Phys. Fluids, 16(10):3670-3681, 2004.
232
233 DO K=1,KBAR
234 DO J=1,JBAR
235 DO I=1,IBAR
236 IF (SOLID(CELL_INDEX(I,J,K))) CYCLE
237 DUDX = RDX(I)*(UU(I,J,K)-UU(I-1,J,K))
238 DVDY = RDY(J)*(VV(I,J,K)-VV(I,J-1,K))
239 DWDZ = RDZ(K)*(WW(I,J,K)-WW(I,J,K-1))
240 DUDY = 0.25*_EB*RDY(J)*(UU(I,J+1,K)-UU(I,J-1,K)+UU(I-1,J+1,K)-UU(I-1,J-1,K))
241 DUDZ = 0.25*_EB*RDZ(K)*(UU(I,J,K+1)-UU(I,J,K-1)+UU(I-1,J,K+1)-UU(I-1,J,K-1))
242 DVDX = 0.25*_EB*RDX(I)*(VV(I+1,J,K)-VV(I-1,J,K)+VV(I+1,J-1,K)-VV(I-1,J-1,K))
243 DVDZ = 0.25*_EB*RDZ(K)*(VV(I,J,K+1)-VV(I,J,K-1)+VV(I,J-1,K+1)-VV(I,J-1,K-1))
244 DWDY = 0.25*_EB*RDX(I)*(WW(I+1,J,K)-WW(I-1,J,K)+WW(I+1,J,K-1)-WW(I-1,J,K-1))
245 DWDY = 0.25*_EB*RDY(J)*(WW(I,J+1,K)-WW(I,J-1,K)+WW(I,J+1,K-1)-WW(I,J-1,K-1))
246
247 ! Vreman, Eq. (6)
248 A_IJ(1,1)=DUDX; A_IJ(2,1)=DUDY; A_IJ(3,1)=DUDZ
249 A_IJ(1,2)=DVDX; A_IJ(2,2)=DVDY; A_IJ(3,2)=DVDZ
250 A_IJ(1,3)=DWDX; A_IJ(2,3)=DWDY; A_IJ(3,3)=DWDZ
251
252 AA=0._EB
253 DO JJ=1,3
254 DO II=1,3
255 AA = AA + A_IJ(II,JJ)*A_IJ(II,JJ)
256 ENDDO
257 ENDDO
258
259 ! Vreman, Eq. (7)
260 B_IJ(1,1)=(DX(I)*A_IJ(1,1))**2 + (DY(J)*A_IJ(2,1))**2 + (DZ(K)*A_IJ(3,1))**2
261 B_IJ(2,2)=(DX(I)*A_IJ(1,2))**2 + (DY(J)*A_IJ(2,2))**2 + (DZ(K)*A_IJ(3,2))**2
262 B_IJ(3,3)=(DX(I)*A_IJ(1,3))**2 + (DY(J)*A_IJ(2,3))**2 + (DZ(K)*A_IJ(3,3))**2
263
264 B_IJ(1,2)=DX(I)**2*A_IJ(1,1)*A_IJ(1,2) + DY(J)**2*A_IJ(2,1)*A_IJ(2,2) + DZ(K)**2*A_IJ(3,1)*A_IJ(3,2)
265 B_IJ(1,3)=DX(I)**2*A_IJ(1,1)*A_IJ(1,3) + DY(J)**2*A_IJ(2,1)*A_IJ(2,3) + DZ(K)**2*A_IJ(3,1)*A_IJ(3,3)
266 B_IJ(2,3)=DX(I)**2*A_IJ(1,2)*A_IJ(1,3) + DY(J)**2*A_IJ(2,2)*A_IJ(2,3) + DZ(K)**2*A_IJ(3,2)*A_IJ(3,3)
267
268 BB = B_IJ(1,1)*B_IJ(2,2) - B_IJ(1,2)**2 &
269 + B_IJ(1,1)*B_IJ(3,3) - B_IJ(1,3)**2 &
270 + B_IJ(2,2)*B_IJ(3,3) - B_IJ(2,3)**2 ! Vreman, Eq. (8)
271
272 IF (ABS(AA)>TWO_EPSILON_EB .AND. BB>TWO_EPSILON_EB) THEN
273 NUEDDY = CVREMAN*SQRT(BB/AA) ! Vreman, Eq. (5)
274 ELSE
275 NUEDDY=0._EB
276 ENDF
277
278 MU(I,J,K) = MUDNS(I,J,K) + RHOP(I,J,K)*NUEDDY
279
280 ENDDO
281 ENDDO
282 ENDDO
283
284 CASE (RNG) SELECT_TURB
285
286 ! A. Yakhot, S. A. Orszag, V. Yakhot, and M. Israeli. Renormalization Group Formulation of Large-Eddy Simulation.
287 ! Journal of Scientific Computing, 1(1):1-51, 1989.
288
289 DO K=1,KBAR
290 DO J=1,JBAR
291 DO I=1,IBAR
292 IF (SOLID(CELL_INDEX(I,J,K))) CYCLE

```



```

293 DELTA = LES_FILTER_WIDTH_FUNCTION(DX(1),DY(J),DZ(K))
294 CALL RNG.EDDY_VISCOSITY(MU.EFF,MU.DNS(I,J,K),RHOP(I,J,K),STRAIN_RATE(I,J,K),DELTA)
295 MU(I,J,K) = MU.EFF
296 ENDDO
297 ENDDO
298 ENDDO
299
300 CASE (WALE) SELECT_TURB
301
302 DO K=1,KBAR
303 DO J=1,JBAR
304 DO I=1,IBAR
305 IF (SOLID(CELL_INDEX(I,J,K))) CYCLE
306 DELTA = LES_FILTER_WIDTH_FUNCTION(DX(1),DY(J),DZ(K))
307 ! compute velocity gradient tensor
308 DUDX = RDX(I)*(UU(I,J,K)-UU(I-1,J,K))
309 DVDY = RDY(J)*(VV(I,J,K)-VV(I,J-1,K))
310 DWDZ = RDZ(K)*(WW(I,J,K)-WW(I,J,K-1))
311 DUDY = 0.25_EB*RDY(J)*(UU(I,J+1,K)-UU(I,J-1,K)+UU(I-1,J+1,K)-UU(I-1,J-1,K))
312 DUDZ = 0.25_EB*RDZ(K)*(UU(I,J,K+1)-UU(I,J,K-1)+UU(I-1,J,K+1)-UU(I-1,J,K-1))
313 DVDX = 0.25_EB*RDX(I)*(VV(I+1,J,K)-VV(I-1,J,K)+VV(I+1,J-1,K)-VV(I-1,J-1,K))
314 DVZD = 0.25_EB*RDZ(K)*(VV(I,J,K+1)-VV(I,J,K-1)+VV(I,J-1,K+1)-VV(I,J-1,K-1))
315 DWDX = 0.25_EB*RDX(I)*(WW(I+1,J,K)-WW(I-1,J,K)+WW(I+1,J,K-1)-WW(I-1,J,K-1))
316 DWDY = 0.25_EB*RDY(J)*(WW(I,J+1,K)-WW(I,J-1,K)+WW(I,J+1,K-1)-WW(I,J-1,K-1))
317 A_IJ(1,1)=DUDX; A_IJ(1,2)=DUDY; A_IJ(1,3)=DUDZ
318 A_IJ(2,1)=DVDX; A_IJ(2,2)=DVDY; A_IJ(2,3)=DVZD
319 A_IJ(3,1)=DWDX; A_IJ(3,2)=DWDY; A_IJ(3,3)=DWDZ
320
321 CALL WALE_VISCOSITY(NU.EDDY, A_IJ, DELTA)
322
323 MU(I,J,K) = MU.DNS(I,J,K) + RHOP(I,J,K)*NU.EDDY
324 ENDDO
325 ENDDO
326 ENDDO
327
328 END SELECT SELECT_TURB
329
330 ! Compute resolved kinetic energy per unit mass
331
332 DO K=1,KBAR
333 DO J=1,JBAR
334 DO I=1,IBAR
335 IF (SOLID(CELL_INDEX(I,J,K))) CYCLE
336 U2 = 0.25_EB*(UU(I-1,J,K)+UU(I,J,K))**2
337 V2 = 0.25_EB*(VV(I,J-1,K)+VV(I,J,K))**2
338 W2 = 0.25_EB*(WW(I,J,K-1)+WW(I,J,K))**2
339 KRES(I,J,K) = 0.5_EB*(U2+V2+W2)
340 ENDDO
341 ENDDO
342 ENDDO
343
344 ! Mirror viscosity into solids and exterior boundary cells
345
346 CELL_COUNTER => IWORK1 ; CELL_COUNTER = 0
347
348 WALL_LOOP: DO IW=1,N.EXTERNAL_WALL.CELLS+N.INTERNAL_WALL.CELLS
349
350 WC=>WALL(IW)
351 IF (WC%BOUNDARY.TYPE==NULL_BOUNDARY) CYCLE WALL_LOOP
352 II = WC%ONE.D%II
353 JJ = WC%ONE.D%JJ
354 KK = WC%ONE.D%KK
355 IOR = WC%ONE.D%IOR
356 IIG = WC%ONE.D%IIG
357 JIG = WC%ONE.D%JIG
358 KIG = WC%ONE.D%KIG
359 SF=>SURFACE(WC%SURF_INDEX)
360
361 SELECT CASE(WC%BOUNDARY.TYPE)
362
363 CASE(SOLID.BOUNDARY)
364
365 IF (ABS(SP%T_IGN-T.BEGIN)<=SPACING(SP%T_IGN) .AND. SP%RAMP_INDEX(TIME.VELO)>=1) THEN
366 TSI = T
367 ELSE
368 TSI = T-SP%T_IGN
369 ENDF
370 RAMP.T = EVALUATE_RAMP(TSI, SP%TAU(TIME.VELO), SP%RAMP_INDEX(TIME.VELO))
371 VEL.T = RAMP.T*SQRT(SP%VEL.T(1)**2 + SP%VEL.T(2)**2)
372
373 SELECT CASE(ABS(IOR))
374 CASE(1)
375 VEL.GAS = SQRT( 0.25_EB*( (VV(IIG,JIG,KIG)+VV(IIG,JIG-1,KIG))**2 + (WW(IIG,JIG,KIG)+WW(IIG,JIG,KIG-1))**2 ) )
376 CASE(2)
377 VEL.GAS = SQRT( 0.25_EB*( (UU(IIG,JIG,KIG)+UU(IIG-1,JIG,KIG))**2 + (WW(IIG,JIG,KIG)+WW(IIG,JIG,KIG-1))**2 ) )
378 CASE(3)
379 VEL.GAS = SQRT( 0.25_EB*( (UU(IIG,JIG,KIG)+UU(IIG-1,JIG,KIG))**2 + (VV(IIG,JIG,KIG)+VV(IIG,JIG-1,KIG))**2 ) )
380 END SELECT

```

```

381
382 CALL WALLMODEL(SLIP_COEF,WC%U_TAU,WC%Y_PLUS,VEL_GAS-VEL_T,&
383 MUDNS(IIG,JJG,KKG)/RHO(IIG,JJG,KKG),1._EB/WC%ONE.D%RDN,SURFACE(WC%SURF_INDEX)%ROUGHNESS)
384
385 IF (LES) THEN
386 DELTA = LES.FILTER_WIDTH_FUNCTION(DX(IIG),DY(JJG),DZ(KKG))
387 SELECT CASE(NEAR_WALL_TURB_MODEL)
388 CASE DEFAULT ! Constant Smagorinsky with Van Driest damping
389 VDF = 1._EB-EXP(-WC%Y_PLUS*RAPLUS)
390 NU_EDDY = (VDF*C.SMAGORINSKY*DELTA)**2*STRAIN_RATE(IIG,JJG,KKG)
391 CASE(WALE)
392 ! compute velocity gradient tensor
393 DUDX = RDX(IIG)*(UU(IIG,JJG,KKG)-UU(IIG-1,JJG,KKG))
394 DVDY = RDY(JJG)*(VV(IIG,JJG,KKG)-VV(IIG,JJG-1,KKG))
395 DWDZ = RDZ(KKG)*(WW(IIG,JJG,KKG)-WW(IIG,JJG,KKG-1))
396 DUDZ = 0.25._EB*RDY(JJG)*(UU(IIG,JJG+1,KKG)-UU(IIG,JJG-1,KKG))+UU(IIG-1,JJG+1,KKG)-UU(IIG-1,JJG-1,KKG))
397 DVDX = 0.25._EB*RDX(IIG)*(VV(IIG+1,JJG,KKG)-VV(IIG-1,JJG,KKG))+VV(IIG+1,JJG-1,KKG)-VV(IIG-1,JJG-1,KKG))
398 DVZDZ = 0.25._EB*RDZ(KKG)*(VV(IIG,JJG,KKG+1)-VV(IIG,JJG,KKG-1))+VV(IIG,JJG-1,KKG+1)-VV(IIG,JJG-1,KKG-1))
399 DWDY = 0.25._EB*RDX(IIG)*(WW(IIG+1,JJG,KKG)-WW(IIG-1,JJG,KKG))+WW(IIG+1,JJG,KKG-1)-WW(IIG-1,JJG,KKG-1))
400 DWDZ = 0.25._EB*RDY(JJG)*(WW(IIG,JJG+1,KKG)-WW(IIG,JJG-1,KKG))+WW(IIG,JJG+1,KKG-1)-WW(IIG,JJG-1,KKG-1))
401 A_IJ(1,1)=DUDX; A_IJ(1,2)=DVDY; A_IJ(1,3)=DUDZ
402 A_IJ(2,1)=DVDX; A_IJ(2,2)=DVZDZ; A_IJ(2,3)=DWDZ
403 A_IJ(3,1)=DWDY; A_IJ(3,2)=DWDZ; A_IJ(3,3)=DWDZ
404 CALL WALE_VISCOSITY(NU_EDDY,A_IJ,DELTA)
405 END SELECT
406 IF (CELL_COUNTER(IIG,JJG,KKG)==0) MU(IIG,JJG,KKG) = 0._EB
407 CELL_COUNTER(IIG,JJG,KKG) = CELL_COUNTER(IIG,JJG,KKG) + 1
408 WGT = 1._EB/REAL(CELL_COUNTER(IIG,JJG,KKG),EB)
409 MU(IIG,JJG,KKG) = (1._EB-WGT)*MU(IIG,JJG,KKG) + WGT*(MUDNS(IIG,JJG,KKG) + RHOP(IIG,JJG,KKG)*NU_EDDY)
410 ELSE
411 MU(IIG,JJG,KKG) = MUDNS(IIG,JJG,KKG)
412 ENDIF
413
414 IF (SOLID(CELL_INDEX(I,J,K))) MU(I,J,K) = MU(IIG,JJG,KKG)
415
416 CASE(OPEN_BOUNDARY,MIRROR_BOUNDARY)
417
418 MU(I,J,K) = MU(IIG,JJG,KKG)
419 KRES(I,J,K) = KRES(IIG,JJG,KKG)
420
421 END SELECT
422
423 ENDDO WALL_LOOP
424
425 MU( 0,0:JBP1, 0) = MU( 1,0:JBP1,1)
426 MU(IBP1,0:JBP1, 0) = MU(IBAR,0:JBP1,1)
427 MU(IBP1,0:JBP1,KBP1) = MU(IBAR,0:JBP1,KBAR)
428 MU( 0,0:JBP1,KBP1) = MU( 1,0:JBP1,KBAR)
429 MU(0:IBP1, 0, 0) = MU(0:IBP1, 1,1)
430 MU(0:IBP1,JBP1,0) = MU(0:IBP1,JBAR,1)
431 MU(0:IBP1,JBP1,KBP1) = MU(0:IBP1,JBAR,KBAR)
432 MU(0:IBP1,0,KBP1) = MU(0:IBP1, 1,KBAR)
433 MU(0, 0,0:KBP1) = MU( 1, 1,0:KBP1)
434 MU(IBP1,0,0:KBP1) = MU(IBAR, 1,0:KBP1)
435 MU(IBP1,JBP1,0:KBP1) = MU(IBAR,JBAR,0:KBP1)
436 MU(0,JBP1,0:KBP1) = MU( 1,JBAR,0:KBP1)
437
438 END SUBROUTINE COMPUTE_VISCOSITY
439
440
441 SUBROUTINE COMPUTE_STRAIN_RATE(NM)
442
443 INTEGER, INTENT(IN) :: NM
444 REAL(EB) :: DUDX,DUDY,DUDZ,DVDX,DVDY,DVDZ,DWDX,DWDY,DWDZ,S11,S22,S33,S12,S13,S23,ONTHDIV
445 INTEGER :: I,J,K,IOR,IIG,JJG,KKG,IW,SURF_INDEX
446 REAL(EB), POINTER, DIMENSION(:,:,:) :: UU=>NULL(),VV=>NULL(),WW=>NULL()
447 TYPE(WALL_TYPE), POINTER :: WC=>NULL()
448
449 CALL POINT_TO_MESH(NM)
450
451 IF (PREDICTOR) THEN
452 UU => U
453 VV => V
454 WW => W
455 ELSE
456 UU => US
457 VV => VS
458 WW => WS
459 ENDIF
460
461 SELECT CASE (TURB_MODEL)
462 CASE DEFAULT
463 DO K=1,KBAR
464 DO J=1,JBAR
465 DO I=1,IBAR
466 IF (SOLID(CELL_INDEX(I,J,K))) CYCLE
467 DUDX = RDX(I)*(UU(I,J,K)-UU(I-1,J,K))

```

```

469 DVDY = RDY(J)*(VV(I,J,K)-VV(I,J-1,K))
470 DWDZ = RDZ(K)*(WW(I,J,K)-WW(I,J,K-1))
471 DUDY = 0.25_EB*RDY(J)*(UU(I,J+1,K)-UU(I,J-1,K)+UU(I-1,J+1,K)-UU(I-1,J-1,K))
472 DUDZ = 0.25_EB*RDZ(K)*(UU(I,J,K+1)-UU(I,J,K-1)+UU(I-1,J,K+1)-UU(I-1,J,K-1))
473 DVDX = 0.25_EB*RDY(I)*(VV(I+1,J,K)-VV(I-1,J,K)+VV(I+1,J-1,K)-VV(I-1,J-1,K))
474 DVDZ = 0.25_EB*RDZ(K)*(VV(I,J,K+1)-VV(I,J,K-1)+VV(I-1,J,K+1)-VV(I-1,J,K-1))
475 DWDX = 0.25_EB*RDY(I)*(WW(I+1,J,K)-WW(I-1,J,K)+WW(I+1,J,K-1)-WW(I-1,J,K-1))
476 DWDY = 0.25_EB*RDY(J)*(WW(I,J+1,K)-WW(I,J-1,K)+WW(I+1,J,K-1)-WW(I+1,J-1,K-1))
477 ONTHDIV = ONTH*(DUDX+DVDY+DWDZ)
478 S11 = DUDX - ONTHDIV
479 S22 = DVDY - ONTHDIV
480 S33 = DWDZ - ONTHDIV
481 S12 = 0.5_EB*(DUDY+DVDX)
482 S13 = 0.5_EB*(DUDZ+DWDX)
483 S23 = 0.5_EB*(DVDZ+DWDY)
484 STRAIN_RATE(I,J,K) = SQRT(2._EB*(S11**2 + S22**2 + S33**2 + 2._EB*(S12**2 + S13**2 + S23**2)))
485 ENDDO
486 ENDDO
487 ENDDO
488 CASE (DEARDORFF)
489 ! Here we omit the 3D loop, we only need the wall cell values of STRAIN_RATE
490 END SELECT
491
492 WALLLOOP: DO IW=1,N_EXTERNAL_WALL_CELLS+N_INTERNAL_WALL_CELLS
493 WC=>WALL(IW)
494 IF (WC%BOUNDARY_TYPE/=SOLID_BOUNDARY) CYCLE WALLLOOP
495
496 SURF_INDEX = WC%SURF_INDEX
497 IIG = WC%ONE_D%IIG
498 JJG = WC%ONE_D%JJG
499 KKG = WC%ONE_D%KKG
500 IOR = WC%ONE_D%IOR
501
502 ! Handle the case where OBST lives on an external boundary
503 IF (IW>N_EXTERNAL_WALL_CELLS) THEN
504 SELECT CASE(IOR)
505 CASE( 1); IF (IIG>IBAR) CYCLE WALLLOOP
506 CASE(-1); IF (IIG<1) CYCLE WALLLOOP
507 CASE( 2); IF (JJG>JBAR) CYCLE WALLLOOP
508 CASE(-2); IF (JJG<1) CYCLE WALLLOOP
509 CASE( 3); IF (KKG>KBAR) CYCLE WALLLOOP
510 CASE(-3); IF (KKG<1) CYCLE WALLLOOP
511 END SELECT
512 ENDF
513
514 DUDX = RDX(IIG)*(UU(IIG,JJG,KKG)-UU(IIG-1,JJG,KKG))
515 DVDY = RDY(JJG)*(VV(IIG,JJG,KKG)-VV(IIG,JJG-1,KKG))
516 DWDZ = RDZ(KKG)*(WW(IIG,JJG,KKG)-WW(IIG,JJG,KKG-1))
517 ONTHDIV = ONTH*(DUDX+DVDY+DWDZ)
518 S11 = DUDX - ONTHDIV
519 S22 = DVDY - ONTHDIV
520 S33 = DWDZ - ONTHDIV
521
522 DUDY = 0.25_EB*RDY(JJG)*(UU(IIG,JJG+1,KKG)-UU(IIG,JJG-1,KKG)+UU(IIG-1,JJG+1,KKG)-UU(IIG-1,JJG-1,KKG))
523 DUDZ = 0.25_EB*RDZ(KKG)*(UU(IIG,JJG,KKG+1)-UU(IIG,JJG,KKG-1)+UU(IIG-1,JJG,KKG+1)-UU(IIG-1,JJG,KKG-1))
524 DVDX = 0.25_EB*RDY(IIG)*(VV(IIG+1,JJG,KKG)-VV(IIG-1,JJG,KKG)+VV(IIG+1,JJG-1,KKG)-VV(IIG-1,JJG-1,KKG))
525 DVDZ = 0.25_EB*RDZ(KKG)*(VV(IIG,JJG,KKG+1)-VV(IIG,JJG,KKG-1)+VV(IIG,JJG-1,KKG+1)-VV(IIG,JJG-1,KKG-1))
526 DWDX = 0.25_EB*RDY(IIG)*(WW(IIG+1,JJG,KKG)-WW(IIG-1,JJG,KKG)+WW(IIG+1,JJG,KKG-1)-WW(IIG-1,JJG,KKG-1))
527 DWDY = 0.25_EB*RDY(JJG)*(WW(IIG,JJG+1,KKG)-WW(IIG,JJG-1,KKG)+WW(IIG,JJG+1,KKG-1)-WW(IIG,JJG-1,KKG-1))
528
529 S12 = 0.5_EB*(DUDY+DVDX)
530 S13 = 0.5_EB*(DUDZ+DWDX)
531 S23 = 0.5_EB*(DVDZ+DWDY)
532
533 STRAIN_RATE(IIG,JJG,KKG) = SQRT(2._EB*(S11**2 + S22**2 + S33**2 + 2._EB*(S12**2 + S13**2 + S23**2)))
534 ENDDO WALLLOOP
535
536 END SUBROUTINE COMPUTE_STRAIN_RATE
537
538
539 SUBROUTINE VISCOSITY_BC(NM)
540
541 ! Specify ghost cell values of the viscosity array MI
542
543 INTEGER, INTENT(IN) :: NM
544 REAL(EB) :: MU_OTHER, DP_OTHER, KRES_OTHER
545 INTEGER :: II, JJ, KK, IW, IO, JJO, KKO, NOM, N_INT_CELLS
546 TYPE(WALL_TYPE), POINTER :: WC=>NULL()
547 TYPE(EXTERNAL_WALL_TYPE), POINTER :: EWC=>NULL()
548
549 CALL POINT_TO_MESH(NM)
550
551 ! Mirror viscosity into solids and exterior boundary cells
552
553 WALLLOOP: DO IW=1,N_EXTERNAL_WALL_CELLS
554 WC=>WALL(IW)
555 EWC=>EXTERNAL_WALL(IW)
556 IF (EWC%NM/=0) CYCLE WALLLOOP

```

```

557 II = WC%ONE.D%II
558 JJ = WC%ONE.D%JJ
559 KK = WC%ONE.D%KK
560 NCM = EWC%NCM
561 MU.OTHER = 0. _EB
562 DP.OTHER = 0. _EB
563 KRES.OTHER = 0. _EB
564 DO KKO=EWC%KKO.MIN,EWC%KKO.MAX
565 DO JJO=EWC%JJO.MIN,EWC%JJO.MAX
566 DO IIO=EWC%iIO.MIN,EWC%iIO.MAX
567 MU.OTHER = MU.OTHER + OMESH(NCM)%MU(IIO,JJO,KKO)
568 KRES.OTHER = KRES.OTHER + OMESH(NCM)%KRES(IIO,JJO,KKO)
569 IF (PREDICTOR) THEN
570 DP.OTHER = DP.OTHER + OMESH(NCM)%D(IIO,JJO,KKO)
571 ELSE
572 DP.OTHER = DP.OTHER + OMESH(NCM)%DS(IIO,JJO,KKO)
573 ENDIF
574 ENDDO
575 ENDDO
576 ENDDO
577 N_INT_CELLS = (EWC%iIO.MAX-EWC%iIO.MIN+1) * (EWC%JJO.MAX-EWC%JJO.MIN+1) * (EWC%KKO.MAX-EWC%KKO.MIN+1)
578 MU.OTHER = MU.OTHER/REAL(N_INT_CELLS,EB)
579 KRES.OTHER = KRES.OTHER/REAL(N_INT_CELLS,EB)
580 DP.OTHER = DP.OTHER/REAL(N_INT_CELLS,EB)
581 MU(II,JJ,KK) = MU.OTHER
582 KRES(II,JJ,KK) = KRES.OTHER
583 IF (PREDICTOR) THEN
584 D(II,JJ,KK) = DP.OTHER
585 ELSE
586 DS(II,JJ,KK) = DP.OTHER
587 ENDIF
588 ENDDO WALL_LOOP
589
590 END SUBROUTINE VISCOSITY_BC
591
592
593 SUBROUTINE VELOCITY_FLUX(T,DT,NM)
594
595 ! Compute convective and diffusive terms of the momentum equations
596
597 USE MATH_FUNCTIONS, ONLY: EVALUATE_RAMP
598 USE COMPLEX_GEOMETRY, ONLY: CCIBM_VELOCITY_FLUX
599
600 ! Ses
601 USE MESH_POINTERS
602 USE penalization
603
604 INTEGER, INTENT(IN) :: NM
605 REAL(EB), INTENT(IN) :: T,DT
606 REAL(EB) :: MUX,MUY,MUZ,UP,UM,VP,VM,WP,WM,VIRM,OMXP,OMXM,OMYP,OMYM,OMZP,OMZM,TXYP,TXYM,TXZP,TXZM,TYZP,TYZM, &
607 DTXYDY,DTXZDZ,DTYZDZ,DTXYDX,DTXZDX,DTYZDY, &
608 DUDX,DVDY,DWDZ,DUDY,DUDZ,DVDX,DVDZ,DWDX,DWDY, &
609 VOMZ,WOMY,UOMY,VOMX,UOMZ,WOMX &
610 RHOP,GX(0:IBAR.MAX),GY(0:IBAR.MAX),GZ(0:IBAR.MAX),TXXP,TXXM,TYYP,TYEM,TZZP,TZZM,DTXXDX,DTYYDY,DTZZDZ, &
611 DUMMY=0. _EB
612 REAL(EB) :: VEG_LMAG
613 INTEGER :: I,J,K,IEXP,IEXM,IEYP,IEYM,IEZM,IC,IC1,IC2
614 REAL(EB), POINTER, DIMENSION(:,:,:): :: TXY=>NULL(),TXZ=>NULL(),TYZ=>NULL(),OMX=>NULL(),OMY=>NULL(),OMZ=>NULL(), &
615 UV=>NULL(),VV=>NULL(),WV=>NULL(),RHOP=>NULL(),DP=>NULL()
616
617 ! Ses
618 integer :: I1
619 double precision :: xloc,yloc
620
621 CALL POINT_TO_MESH(NM)
622
623 IF (PREDICTOR) THEN
624 UU => U
625 VV => V
626 WW => W
627 DP => D
628 RHOP => RHO
629 ELSE
630 UU => US
631 VV => VS
632 WW => WS
633 DP => DS
634 RHOP => RHOS
635 ENDIF
636
637 TXY => WORK1
638 TXZ => WORK2
639 TYZ => WORK3
640 OMX => WORK4
641 OMY => WORK5
642 OMZ => WORK6
643
644

```

```

645 ! Compute vorticity and stress tensor components
646
647 !$OMP PARALLEL DO PRIVATE(DUDY, DVDX, DUDZ, DWDX, DVDZ, DWDY, &
648 !$OMP& MLX, MUY, MUZ) SCHEDULE(STATIC)
649 DO K=0,KBAR
650 DO J=0,JBAR
651 DO I=0,IBAR
652 DUDY = RDYN(J)*(UU(I,J+1,K)-UU(I,J,K))
653 DVDX = RDXN(1)*(VV(I+1,J,K)-VV(I,J,K))
654 DUDZ = RDZN(K)*(UU(I,J,K+1)-UU(I,J,K))
655 DWDX = RDXN(1)*(WW(I+1,J,K)-WW(I,J,K))
656 DVDZ = RDZN(K)*(VV(I,J,K+1)-VV(I,J,K))
657 DWDY = RDYN(J)*(WW(I,J+1,K)-WW(I,J,K))
658 OMX(I,J,K) = DWDY - DVDZ
659 OMY(I,J,K) = DUDZ - DWDX
660 OMZ(I,J,K) = DVDX - DUDY
661 MLX = 0.25_EB*(MU(I,J+1,K)+MU(I,J,K)+MU(I,J,K+1)+MU(I,J+1,K+1))
662 MUY = 0.25_EB*(MU(I+1,J,K)+MU(I,J,K)+MU(I,J,K+1)+MU(I+1,J,K+1))
663 MUZ = 0.25_EB*(MU(I+1,J,K)+MU(I,J,K)+MU(I,J+1,K)+MU(I+1,J+1,K))
664 TXY(I,J,K) = MUZ*(DVDX + DUDY)
665 TXZ(I,J,K) = MUY*(DUDZ + DWDX)
666 TYZ(I,J,K) = MLX*(DVDZ + DWDY)
667 ENDDO
668 ENDDO
669 ENDDO
670 !$OMP END PARALLEL DO
671
672 ! Wannier Flow (Stokes flow) test case
673
674 IF (PERIODIC.TEST==5) THEN
675 OMX=0._EB
676 OMY=0._EB
677 OMZ=0._EB
678 OME.E = -1.E6_EB
679 ENDIF
680
681 ! Compute gravity components
682
683 IF (.NOT.SPATIAL_GRAVITY_VARIATION) THEN
684 GX(0:IBAR) = EVALUATE_RAMP(T,DUMMY,LRAMP.GX)*GVEC(1)
685 GY(0:IBAR) = EVALUATE_RAMP(T,DUMMY,LRAMP.GY)*GVEC(2)
686 GZ(0:IBAR) = EVALUATE_RAMP(T,DUMMY,LRAMP.GZ)*GVEC(3)
687 ELSE
688 DO I=0,IBAR
689 GX(I) = EVALUATE_RAMP(X(I),DUMMY,LRAMP.GX)*GVEC(1)
690 GY(I) = EVALUATE_RAMP(X(I),DUMMY,LRAMP.GY)*GVEC(2)
691 GZ(I) = EVALUATE_RAMP(X(I),DUMMY,LRAMP.GZ)*GVEC(3)
692 ENDDO
693 ENDIF
694
695 ! Compute x-direction flux term FVX
696
697 !$OMP PARALLEL PRIVATE(WP, WM, VP, VM, UP, UM, &
698 !$OMP& OMXP, OMXM, OMP, OMM, OMZP, OMZM, &
699 !$OMP& TXZP, TXZM, TXYP, TXYM, TYZP, TYZM, &
700 !$OMP& IC, IEXP, IEXM, IEYP, IEYM, IEZP, IEZM, &
701 !$OMP& RRHO, DUDX, DVDY, DWDX, VTRM)
702 !$OMP DO SCHEDULE(static) &
703 !$OMP& PRIVATE(WOMY, WOMZ, TXXP, TXXM, DTXXDX, DTXYDY, DTXZDZ)
704 DO K=1,KBAR
705 DO J=1,JBAR
706 DO I=0,IBAR
707 WP = WW(I,J,K) + WW(I+1,J,K)
708 WM = WW(I,J,K-1) + WW(I+1,J,K-1)
709 VP = VV(I,J,K) + VV(I+1,J,K)
710 VM = VV(I,J-1,K) + VV(I+1,J-1,K)
711 OMP = OMY(I,J,K)
712 OMM = OMY(I,J,K-1)
713 OMZP = OMZ(I,J,K)
714 OMZM = OMZ(I,J-1,K)
715 TXZP = TXZ(I,J,K)
716 TXZM = TXZ(I,J,K-1)
717 TXYP = TXY(I,J,K)
718 TXYM = TXY(I,J-1,K)
719 IC = CELL_INDEX(I,J,K)
720 IEYP = EDGE_INDEX(8,IC)
721 IEYM = EDGE_INDEX(6,IC)
722 IEZP = EDGE_INDEX(12,IC)
723 IEZM = EDGE_INDEX(10,IC)
724 IF (OME.E(-1,IEYP)>-1.E5_EB) THEN
725 OMP = OME.E(-1,IEYP)
726 TXZP = TAU.E(-1,IEYP)
727 ENDIF
728 IF (OME.E(1,IEYM)>-1.E5_EB) THEN
729 OMM = OME.E(1,IEYM)
730 TXZM = TAU.E(1,IEYM)
731 ENDIF
732 IF (OME.E(-2,IEZP)>-1.E5_EB) THEN

```

Source Code files for edited portions of FDS

```

733 OMZP = OME.E(-2,IEZP)
734 TXYP = TAU.E(-2,IEZP)
735 ENDIF
736 IF (OME.E( 2,IEZM)>-1.E5.EB) THEN
737 OMZM = OME.E( 2,IEZM)
738 TXYM = TAU.E( 2,IEZM)
739 ENDIF
740 WCMY = WP*OMYP + WM*OMYM
741 VCMZ = VP*OMZP + VM*OMZM
742 RRHO = 2.*EB/(RHOP(1,J,K)+RHOP(1+1,J,K))
743 DVDY = (VV(1+1,J,K)-VV(1+1,J-1,K))*RDY(J)
744 DWDZ = (WW(1+1,J,K)-WW(1+1,J,K-1))*RDZ(K)
745 TXXP = MU(1+1,J,K)*(FOIH*DP(1+1,J,K) - 2.*EB*(DVDY+DWDZ) )
746 DVDY = (VV(1,J,K)-VV(1,J-1,K))*RDY(J)
747 DWDZ = (WW(1,J,K)-WW(1,J,K-1))*RDZ(K)
748 TXXM = MU(1,J,K) *(FOIH*DP(1,J,K) - 2.*EB*(DVDY+DWDZ) )
749 DTXDX= RDXN(1)*(TXXP-TXXM)
750 DTYDY= RDY(J) *(TXYP-TXYM)
751 DTXDZ= RDZ(K) *(TXZP-TXXM)
752 VIRM = DTXDX + DTYDY + DTXDZ
753 FVX(1,J,K) = 0.25.EB*(WCMY - VCMZ) - GX(1) + RRHO*(GX(1)*RHO.0(K) - VIRM)
754 ENDDO
755 ENDDO
756 ENDDO
757 !$OMP END DO NOWAIT
758
759 ! Compute y-direction flux term FVY
760
761 !$OMP DO SCHEDULE(static) &
762 !$OMP PRIVATE(WCMX, UOMZ, TYYP, TYXM, DTXYDX, DTYDY, DTYZDZ)
763 DO K=1,KBAR
764 DO J=0,JBAR
765 DO I=1,I BAR
766 UP = UU(1,J,K) + UU(1,J+1,K)
767 UM = UU(1-1,J,K) + UU(1-1,J+1,K)
768 WP = WW(1,J,K) + WW(1,J+1,K)
769 WM = WW(1,J,K-1) + WW(1,J+1,K-1)
770 OMPX = OMX(1,J,K)
771 OMMX = OMX(1,J,K-1)
772 OMZP = OMZ(1,J,K)
773 OMZM = OMZ(1-1,J,K)
774 TYZP = TYZ(1,J,K)
775 TYZM = TYZ(1,J,K-1)
776 TXYP = TXY(1,J,K)
777 TXYM = TXY(1-1,J,K)
778 IC = CELL.INDEX(1,J,K)
779 IEXP = EDGE.INDEX(4,IC)
780 IEXM = EDGE.INDEX(2,IC)
781 IEZP = EDGE.INDEX(12,IC)
782 IEZM = EDGE.INDEX(11,IC)
783 IF (OME.E(-2,IEXP)>-1.E5.EB) THEN
784 OMPX = OME.E(-2,IEXP)
785 TYZP = TAU.E(-2,IEXP)
786 ENDIF
787 IF (OME.E( 2,IEXM)>-1.E5.EB) THEN
788 OMMX = OME.E( 2,IEXM)
789 TYZM = TAU.E( 2,IEXM)
790 ENDIF
791 IF (OME.E(-1,IEZP)>-1.E5.EB) THEN
792 OMZP = OME.E(-1,IEZP)
793 TXYP = TAU.E(-1,IEZP)
794 ENDIF
795 IF (OME.E( 1,IEZM)>-1.E5.EB) THEN
796 OMZM = OME.E( 1,IEZM)
797 TXYM = TAU.E( 1,IEZM)
798 ENDIF
799 WCMX = WP*OMXP + WM*OMXM
800 UOMZ = UP*OMZP + UM*OMZM
801 RRHO = 2.*EB/(RHOP(1,J,K)+RHOP(1,J+1,K))
802 DUDX = (UU(1,J+1,K)-UU(1-1,J+1,K))*RDX(1)
803 DWDZ = (WW(1,J+1,K)-WW(1,J+1,K-1))*RDZ(K)
804 TYYP = MU(1,J+1,K)*(FOIH*DP(1,J+1,K) - 2.*EB*(DUDX+DWDZ) )
805 DUDX = (UU(1,J,K)-UU(1-1,J,K))*RDX(1)
806 DWDZ = (WW(1,J,K)-WW(1,J,K-1))*RDZ(K)
807 TYXM = MU(1,J,K) *(FOIH*DP(1,J,K) - 2.*EB*(DUDX+DWDZ) )
808 DTXYDX= RDX(1) *(TXYP-TXYM)
809 DTYDY= RDY(J) *(TYYP-TYXM)
810 DTYZDZ= RDZ(K) *(TYZP-TYXM)
811 VIRM = DTXYDX + DTYDY + DTYZDZ
812 FVY(1,J,K) = 0.25.EB*(UOMZ - WCMX) - GY(1) + RRHO*(GY(1)*RHO.0(K) - VIRM)
813 ENDDO
814 ENDDO
815 ENDDO
816 !$OMP END DO NOWAIT
817
818 ! Compute z-direction flux term FVZ
819
820 !$OMP DO SCHEDULE(static) &

```

Source Code files for edited portions of FDS

```

821  !&PRIVATE(UOMY, VOMX, TZPZ, TZMZ, DTZXDX, DTZYDY, DTZZDZ)
822  DO K=0,KBAR
823  DO J=1,JBAR
824  DO I=1,IBAR
825  UP = UU(I,J,K) + UU(I,J,K+1)
826  UM = UU(I-1,J,K) + UU(I-1,J,K+1)
827  VP = VV(I,J,K) + VV(I,J,K+1)
828  VM = VV(I,J-1,K) + VV(I,J-1,K+1)
829  OMP = OM(I,J,K)
830  OMM = OM(I-1,J,K)
831  OMP = OM(I,J,K)
832  OMM = OM(I,J-1,K)
833  TZP = TZ(I,J,K)
834  TZM = TZ(I-1,J,K)
835  TYZ = TYZ(I,J,K)
836  TYM = TYZ(I,J-1,K)
837  IC = CELL.INDEX(I,J,K)
838  IEXP = EDGE.INDEX(4,IC)
839  IEXM = EDGE.INDEX(3,IC)
840  IEYP = EDGE.INDEX(8,IC)
841  IEYM = EDGE.INDEX(7,IC)
842  IF (OME.E(-1,IEXP)>-1.E5.EB) THEN
843  OMP = OME.E(-1,IEXP)
844  TYZ = TAU.E(-1,IEXP)
845  ENDIF
846  IF (OME.E( 1,IEXM)>-1.E5.EB) THEN
847  OMM = OME.E( 1,IEXM)
848  TYM = TAU.E( 1,IEXM)
849  ENDIF
850  IF (OME.E(-2,IEYP)>-1.E5.EB) THEN
851  OMP = OME.E(-2,IEYP)
852  TZP = TAU.E(-2,IEYP)
853  ENDIF
854  IF (OME.E( 2,IEYM)>-1.E5.EB) THEN
855  OMM = OME.E( 2,IEYM)
856  TZM = TAU.E( 2,IEYM)
857  ENDIF
858  UOMY = UP*OMP + UM*OMM
859  VOMX = VP*OMP + VM*OMM
860  RRHO = 2. .EB/(RHOP(I,J,K)+RHOP(I,J,K+1))
861  DUDX = (UU(I,J,K+1)-UU(I-1,J,K+1))*RDX(I)
862  DVDY = (VV(I,J,K+1)-VV(I,J-1,K+1))*RDY(J)
863  TZPZ = MU(I,J,K+1)*(FOIH*DP(I,J,K+1) - 2. .EB*(DUDX+DVDY))
864  DUDX = (UU(I,J,K)-UU(I-1,J,K))*RDX(I)
865  DVDY = (VV(I,J,K)-VV(I,J-1,K))*RDY(J)
866  TZMZ = MU(I,J,K) *(FOIH*DP(I,J,K) - 2. .EB*(DUDX+DVDY))
867  DTZXDX= RDX(I) *(TZP-TMZ)
868  DTZYDY= RDY(J) *(TYZ-TYM)
869  DTZZDZ= RDN(K) *(TZP-TMZ)
870  VIRM = DTZXDX + DTZYDY + DTZZDZ
871  FVZ(I,J,K) = 0.25.EB*(VOMX - UOMY) - CZ(I) + RRHO*(CZ(I)*0.5.EB*(RHO.0(K)+RHO.0(K+1)) - VIRM)
872  ENDDO
873  ENDDO
874  ENDDO
875  !&END DO NOWAIT
876  !&END PARALLEL
877
878  IF (EVACUATION.ONLY(NM)) THEN
879  FVZ = 0. .EB
880  RETURN
881  END IF
882
883  ! Additional force terms
884
885  IF (ANY(MEAN.FORCING)) CALL MOMENTUMNUDGING ! Mean forcing
886  IF (ANY(ABS(FVEC)>TWO.EPSILON.EB)) CALL DIRECT_FORCE ! Direct force
887  IF (ANY(ABS(OVEC)>TWO.EPSILON.EB)) CALL CORIOLIS_FORCE ! Coriolis force
888
889  ! Ses
890  do ll=1,size(pendat,1)
891
892  cntr=0
893
894  if (pendat(ll,14).ge.0) then
895  if (pendat(ll,14).le.floor(T)) then
896  if (ll.le.4) then
897
898  DO K=0,KBAR
899  DO J=0,JBAR
900  DO I=0,IBAR
901  !penalisation region start
902  if (pendat(ll,13).eq.1) then
903  if ((Z(K).le.pendat(ll,9)).and.(Z(K).ge.pendat(ll,8))) then
904  if ((Y(J).le.pendat(ll,7)).and.(Y(J).ge.pendat(ll,6))) then
905  if ((X(I).le.pendat(ll,5)).and.(X(I).ge.pendat(ll,4))) then
906
907  cntr=cntr+1
908

```

```

909 FVX(I,J,K) = FVX(I,J,K) + ((pendat(II,13))*((UU(I,J,K)-pendat(II,14+cntr)))/pendat(II,1)
910 FVY(I,J,K) = FVY(I,J,K) + ((pendat(II,13))*((VV(I,J,K)-pendat(II,14+size(penU0)+cntr)))/pendat(II,1)
911 FVZ(I,J,K) = FVZ(I,J,K) + ((pendat(II,13))*((WW(I,J,K)-pendat(II,14+size(penU0)+size(penV0)+cntr)))/pendat(II,1)
912
913 endif
914 endif
915 endif
916 else
917 !blending region start
918 if ((Z(K).le.pendat(II,9)).and.(Z(K).ge.pendat(II,8))) then
919 if ((Y(J).le.pendat(II,7)).and.(Y(J).ge.pendat(II,6))) then
920 if ((X(I).le.pendat(II,5)).and.(X(I).ge.pendat(II,4))) then
921
922 cntr=cntr+1
923
924 FVX(I,J,K) = FVX(I,J,K) + (pendat(II,13) + pendat(II,10)*X(I)-((pendat(II,10)*(1-pendat(II,10))*pendat(II,5))/2d0
) &
925 -(pendat(II,10)*(1+pendat(II,10))*pendat(II,4)/2d0))*((UU(I,J,K)-pendat(II,14+cntr)))/pendat(II,2) &
926 + (pendat(II,13) + pendat(II,11)*Y(J)-((pendat(II,11)*(1-pendat(II,11))*pendat(II,7))/2d0) &
927 -(pendat(II,11)*(1+pendat(II,11))*pendat(II,6)/2d0))*((UU(I,J,K)-pendat(II,14+cntr)))/pendat(II,2) &
928 + (pendat(II,13) + pendat(II,12)*Z(K)-((pendat(II,12)*(1-pendat(II,12))*pendat(II,9))/2d0) &
929 -(pendat(II,12)*(1+pendat(II,12))*pendat(II,8)/2d0))*((UU(I,J,K)-pendat(II,14+cntr)))/pendat(II,2)
930
931 FVY(I,J,K) = FVY(I,J,K) + (pendat(II,13) + pendat(II,10)*X(I)-((pendat(II,10)*(1-pendat(II,10))*pendat(II,5))/2d0)
&
932 -(pendat(II,10)*(1+pendat(II,10))*pendat(II,4)/2d0))*((VV(I,J,K)-pendat(II,14+size(penU0)+cntr)))/pendat(II,2) &
933 + (pendat(II,13) + pendat(II,11)*Y(J)-((pendat(II,11)*(1-pendat(II,11))*pendat(II,7))/2d0) &
934 -(pendat(II,11)*(1+pendat(II,11))*pendat(II,6)/2d0))*((VV(I,J,K)-pendat(II,14+size(penU0)+cntr)))/pendat(II,2) &
935 + (pendat(II,13) + pendat(II,12)*Z(K)-((pendat(II,12)*(1-pendat(II,12))*pendat(II,9))/2d0) &
936 -(pendat(II,12)*(1+pendat(II,12))*pendat(II,8)/2d0))*((VV(I,J,K)-pendat(II,14+size(penU0)+cntr)))/pendat(II,2)
937
938 FVZ(I,J,K) = FVZ(I,J,K) + (pendat(II,13) + pendat(II,10)*X(I)-((pendat(II,10)*(1-pendat(II,10))*pendat(II,5))/2d0)
&
939 -(pendat(II,10)*(1+pendat(II,10))*pendat(II,4)/2d0))*((WW(I,J,K)-pendat(II,14+size(penU0)+size(penV0)+cntr))/
pendat(II,2) &
940 + (pendat(II,13) + pendat(II,11)*Y(J)-((pendat(II,11)*(1-pendat(II,11))*pendat(II,7))/2d0) &
941 -(pendat(II,11)*(1+pendat(II,11))*pendat(II,6)/2d0))*((WW(I,J,K)-pendat(II,14+size(penU0)+size(penV0)+cntr))/
pendat(II,2) &
942 + (pendat(II,13) + pendat(II,12)*Z(K)-((pendat(II,12)*(1-pendat(II,12))*pendat(II,9))/2d0) &
943 -(pendat(II,12)*(1+pendat(II,12))*pendat(II,8)/2d0))*((WW(I,J,K)-pendat(II,14+size(penU0)+size(penV0)+cntr))/
pendat(II,2)
944
945 endif
946 endif
947 endif
948 endif
949 ENDDO
950 ENDDO
951 ENDDO
952
953 endif
954
955 if((II.gt.4).AND.pendat(II,14).le.floor(T)) then
956
957 if(mod(II,4).eq.1) then
958 pendat(II-1,14)=-1
959 pendat(II-2,14)=-1
960 pendat(II-3,14)=-1
961 pendat(II-4,14)=-1
962 endif
963
964 DO K=0,KBAR
965 DO J=0,JBAR
966 DO I=0,IBAR
967
968 !penalisation region starts
969 if (pendat(II,13).eq.1) then
970
971 if ((Z(K).le.pendat(II,9)).and.(Z(K).ge.pendat(II,8))) then
972 if ((Y(J).le.pendat(II,7)).and.(Y(J).ge.pendat(II,6))) then
973 if ((X(I).le.pendat(II,5)).and.(X(I).ge.pendat(II,4))) then
974
975 cntr=cntr+1
976
977 FVX(I,J,K) = FVX(I,J,K) + ((pendat(II,13))*((UU(I,J,K)-pendat(II,14+cntr)))/pendat(II,1)
978 FVY(I,J,K) = FVY(I,J,K) + ((pendat(II,13))*((VV(I,J,K)-pendat(II,14+size(penU0)+cntr)))/pendat(II,1)
979 FVZ(I,J,K) = FVZ(I,J,K) + ((pendat(II,13))*((WW(I,J,K)-pendat(II,14+size(penU0)+size(penV0)+cntr)))/pendat(II,1)
980
981 endif
982 endif
983 endif
984 else
985 !blending region starts
986 if ((Z(K).le.pendat(II,9)).and.(Z(K).ge.pendat(II,8))) then
987 if ((Y(J).le.pendat(II,7)).and.(Y(J).ge.pendat(II,6))) then
988 if ((X(I).le.pendat(II,5)).and.(X(I).ge.pendat(II,4))) then
989
990 cntr=cntr+1

```



```

991
992 FVX(I,J,K) = FVX(I,J,K) + (pendat(II,13) + pendat(II,10)*X(I) - ((pendat(II,10)*(1-pendat(II,10))*pendat(II,5))/2d0
    ) &
993 -(pendat(II,10)*(1+pendat(II,10))*pendat(II,4)/2d0)*(UU(I,J,K)-pendat(II,14+cntr))/pendat(II,2) &
994 + (pendat(II,13) + pendat(II,11)*Y(J) - ((pendat(II,11)*(1-pendat(II,11))*pendat(II,6))/2d0) &
995 -(pendat(II,11)*(1+pendat(II,11))*pendat(II,7)/2d0)*(UU(I,J,K)-pendat(II,14+cntr))/pendat(II,2) &
996 + (pendat(II,13) + pendat(II,12)*Z(K) - ((pendat(II,12)*(1-pendat(II,12))*pendat(II,8))/2d0) &
997 -(pendat(II,12)*(1+pendat(II,12))*pendat(II,9)/2d0)*(UU(I,J,K)-pendat(II,14+cntr))/pendat(II,2)
998
999 FVY(I,J,K) = FVY(I,J,K) + (pendat(II,13) + pendat(II,10)*X(I) - ((pendat(II,10)*(1-pendat(II,10))*pendat(II,5))/2d0)
    ) &
1000 -(pendat(II,10)*(1+pendat(II,10))*pendat(II,4)/2d0)*(VV(I,J,K)-pendat(II,14+size(penU0)+cntr))/pendat(II,2) &
1001 + (pendat(II,13) + pendat(II,11)*Y(J) - ((pendat(II,11)*(1-pendat(II,11))*pendat(II,6))/2d0) &
1002 -(pendat(II,11)*(1+pendat(II,11))*pendat(II,7)/2d0)*(VV(I,J,K)-pendat(II,14+size(penU0)+cntr))/pendat(II,2) &
1003 + (pendat(II,13) + pendat(II,12)*Z(K) - ((pendat(II,12)*(1-pendat(II,12))*pendat(II,8))/2d0) &
1004 -(pendat(II,12)*(1+pendat(II,12))*pendat(II,9)/2d0)*(VV(I,J,K)-pendat(II,14+size(penU0)+cntr))/pendat(II,2)
1005
1006 FVZ(I,J,K) = FVZ(I,J,K) + (pendat(II,13) + pendat(II,10)*X(I) - ((pendat(II,10)*(1-pendat(II,10))*pendat(II,5))/2d0)
    ) &
1007 -(pendat(II,10)*(1+pendat(II,10))*pendat(II,4)/2d0)*(WW(I,J,K)-pendat(II,14+size(penU0)+size(penV0)+cntr))/
    pendat(II,2) &
1008 + (pendat(II,13) + pendat(II,11)*Y(J) - ((pendat(II,11)*(1-pendat(II,11))*pendat(II,6))/2d0) &
1009 -(pendat(II,11)*(1+pendat(II,11))*pendat(II,7)/2d0)*(WW(I,J,K)-pendat(II,14+size(penU0)+size(penV0)+cntr))/
    pendat(II,2) &
1010 + (pendat(II,13) + pendat(II,12)*Z(K) - ((pendat(II,12)*(1-pendat(II,12))*pendat(II,8))/2d0) &
1011 -(pendat(II,12)*(1+pendat(II,12))*pendat(II,9)/2d0)*(WW(I,J,K)-pendat(II,14+size(penU0)+size(penV0)+cntr))/
    pendat(II,2)
1012
1013 endif
1014 endif
1015 endif
1016
1017 endif
1018 ENDDO
1019 ENDDO
1020 ENDDO
1021 endif
1022
1023 endif
1024 endif
1025 enddo
1026
1027 TRUNK_IF: IF (trunks) THEN
1028 do II=1,ntrunks
1029 DO I=0,IBAR
1030 DO J=0,JBAR
1031 xloc=trnk_loc(1,II)
1032 yloc=trnk_loc(2,II)
1033 if ((X(I).eq.xloc) then
1034 if ((Y(J).eq.yloc) then
1035
1036 do k=0,KBAR
1037 if ((Z(K).gt.trnk_min).and.(Z(K).lt.trnk_max)) then
1038 FVX(I,J,K)=FVX(I,J,K)+UU(I,J,K)/eta
1039 FVY(I,J,K)=FVY(I,J,K)+VV(I,J,K)/eta
1040 FVZ(I,J,K)=FVZ(I,J,K)+WW(I,J,K)/eta
1041 endif
1042 enddo
1043
1044 endif
1045 endif
1046 ENDDO
1047 ENDDO
1048 enddo
1049 ENDIF TRUNK_IF
1050
1051
1052 IF (WFDS.BNDRYFUEL) CALL VEGETATION_DRAG ! Surface vegetation drag
1053 IF (PATCH.VELOCITY) CALL PATCH.VELOCITY_FLUX(DT,NM) ! Specified patch velocity
1054 IF (CC.IBM) CALL CCIBM.VELOCITY_FLUX(DT,NM) ! Direct-forcing Immersed Boundary Method
1055 IF (PERIODIC.TEST==7) CALL MMS.VELOCITY_FLUX(NM,T) ! Source term in manufactured solution
1056
1057 CONTAINS
1058
1059 SUBROUTINE MOMENTUMNUDGING
1060
1061 ! Add a force vector to the momentum equation that moves the flow field towards the direction of the mean flow.
1062
1063 REAL(EB) :: UBAR,VBAR,WBAR,INTEGRAL,SUM,VOLUME,VC,UMEAN,VMEAN,WMEAN,DU_FORCING,DV_FORCING,DW_FORCING,DT_LOC
1064 INTEGER :: NSC,I_LO,J_LO,I_HI,J_HI
1065
1066 DT_LOC = MAX(DT,DT_MEAN_FORCING)
1067 NSC = SPONGE.CELLS
1068
1069 MEAN_FORCING_X: IF (MEAN_FORCING(1)) THEN
1070 SELECT_RAMP_U: SELECT CASE(LRAMP_U0_Z)
1071 CASE(0) SELECT_RAMP_U
1072 INTEGRAL = 0._EB

```

Source Code files for edited portions of FDS

```

1073 SUM.VOLUME = 0. _EB
1074 DO K=1,KBAR
1075 DO J=1,JBAR
1076 DO I=0,IBAR
1077 IC1 = CELL.INDEX(I,J,K)
1078 IC2 = CELL.INDEX(I+1,J,K)
1079 IF (SOLID(IC1)) CYCLE
1080 IF (SOLID(IC2)) CYCLE
1081 IF (.NOT.MEAN.FORCING.CELL(I,J,K) ) CYCLE
1082 IF (.NOT.MEAN.FORCING.CELL(I+1,J,K) ) CYCLE
1083 VC = DXN(I)*DY(J)*DZ(K)
1084 INTEGRAL = INTEGRAL + UU(I,J,K)*VC
1085 SUM.VOLUME = SUM.VOLUME + VC
1086 ENDDO
1087 ENDDO
1088 ENDDO
1089 IF (SUM.VOLUME>TWO.EPSILON.EB) THEN
1090 UMEAN = INTEGRAL/SUM.VOLUME
1091 ELSE
1092 UMEAN = 0. _EB
1093 ENDIF
1094 UBAR = U0*EVALUATE.RAMP(T,DUMMY,I.RAMP.U0.T)
1095 DU.FORCING = (UBAR-UMEAN)/DT.LOC
1096 DO K=1,KBAR
1097 DO J=1,JBAR
1098 DO I=0,IBAR
1099 IF (.NOT.MEAN.FORCING.CELL(I,J,K) ) CYCLE
1100 IF (.NOT.MEAN.FORCING.CELL(I+1,J,K) ) CYCLE
1101 FVX(I,J,K) = FVX(I,J,K) - DU.FORCING
1102 ENDDO
1103 ENDDO
1104 ENDDO
1105 CASE(1:) SELECT_RAMP.U
1106 K.LOOP.U: DO K=1,KBAR
1107 INTEGRAL = 0. _EB
1108 SUM.VOLUME = 0. _EB
1109 DO J=1,JBAR
1110 DO I=0,IBAR
1111 IC1 = CELL.INDEX(I,J,K)
1112 IC2 = CELL.INDEX(I+1,J,K)
1113 IF (SOLID(IC1)) CYCLE
1114 IF (SOLID(IC2)) CYCLE
1115 VC = DXN(I)*DY(J)*DZ(K)
1116 INTEGRAL = INTEGRAL + UU(I,J,K)*VC
1117 SUM.VOLUME = SUM.VOLUME + VC
1118 ENDDO
1119 ENDDO
1120 IF (SUM.VOLUME>TWO.EPSILON.EB) THEN
1121 UMEAN = INTEGRAL/SUM.VOLUME
1122 ELSE
1123 ! this can happen if all cells in a given row, k, are solid
1124 UMEAN = 0. _EB
1125 ENDF
1126 UBAR = U0*EVALUATE.RAMP(T,DUMMY,I.RAMP.U0.T)*EVALUATE.RAMP(ZC(K),DUMMY,I.RAMP.U0.Z)
1127 DU.FORCING = (UBAR-UMEAN)/DT.LOC
1128 ! Apply the average force term to bulk of domain, and apply more aggressive forcing at boundary
1129 I.LO = 0
1130 I.HI = IBAR
1131 IF (APPLY.SPONGE.LAYER(1)) THEN
1132 FVX(0:NSC-1,;,K) = FVX(0:NSC-1,;,K) - (UBAR-UU(0:NSC-1,;,K))/DT.LOC
1133 I.LO = NSC
1134 ENDF
1135 IF (APPLY.SPONGE.LAYER(-1)) THEN
1136 FVX(IBAR-NSC+1:IBAR,;,K) = FVX(IBAR-NSC+1:IBAR,;,K) - (UBAR-UU(IBAR-NSC+1:IBAR,;,K))/DT.LOC
1137 I.HI = IBAR-NSC
1138 ENDF
1139 FVX(I.LO:I.HI,;,K) = FVX(I.LO:I.HI,;,K) - DU.FORCING
1140 ENDDO K.LOOP.U
1141 END SELECT SELECT_RAMP.U
1142 ENDF MEAN.FORCING.X
1143
1144 MEAN.FORCING.Y: IF (MEAN.FORCING(2)) THEN
1145 SELECT_RAMP.V: SELECT CASE(I.RAMP.V0.Z)
1146 CASE(0) SELECT_RAMP.V
1147 INTEGRAL = 0. _EB
1148 SUM.VOLUME = 0. _EB
1149 DO K=1,KBAR
1150 DO J=0,JBAR
1151 DO I=1,IBAR
1152 IC1 = CELL.INDEX(I,J,K)
1153 IC2 = CELL.INDEX(I,J+1,K)
1154 IF (SOLID(IC1)) CYCLE
1155 IF (SOLID(IC2)) CYCLE
1156 IF (.NOT.MEAN.FORCING.CELL(I,J,K) ) CYCLE
1157 IF (.NOT.MEAN.FORCING.CELL(I,J+1,K) ) CYCLE
1158 VC = DX(I)*DYN(J)*DZ(K)
1159 INTEGRAL = INTEGRAL + VV(I,J,K)*VC
1160 SUM.VOLUME = SUM.VOLUME + VC

```

```

1161 ENDDO
1162 ENDDO
1163 ENDDO
1164 IF (SUM.VOLUME>TWO.EPSILON.EB) THEN
1165 VMEAN = INTEGRAL/SUM.VOLUME
1166 ELSE
1167 VMEAN = 0. _EB
1168 ENDF
1169 VBAR = V0*EVALUATE.RAMP(T,DUMMY,L.RAMP.V0.T)
1170 DV.FORCING = (VBAR-VMEAN)/DT.LOC
1171 DO K=1,KBAR
1172 DO J=0,JBAR
1173 DO I=1,IBAR
1174 IF (.NOT.MEAN.FORCING.CELL(I,J,K) ) CYCLE
1175 IF (.NOT.MEAN.FORCING.CELL(I,J+1,K)) CYCLE
1176 FVY(I,J,K) = FVY(I,J,K) - DV.FORCING
1177 ENDDO
1178 ENDDO
1179 ENDDO
1180 CASE(1:) SELECT.RAMP.V
1181 K.LOOP.V: DO K=1,KBAR
1182 INTEGRAL = 0. _EB
1183 SUM.VOLUME = 0. _EB
1184 DO J=0,JBAR
1185 DO I=1,IBAR
1186 IC1 = CELL.INDEX(I,J,K)
1187 IC2 = CELL.INDEX(I,J+1,K)
1188 IF (SOLID(IC1)) CYCLE
1189 IF (SOLID(IC2)) CYCLE
1190 VC = DX(I)*DYN(J)*DZ(K)
1191 INTEGRAL = INTEGRAL + VV(I,J,K)*VC
1192 SUM.VOLUME = SUM.VOLUME + VC
1193 ENDDO
1194 ENDDO
1195 IF (SUM.VOLUME>TWO.EPSILON.EB) THEN
1196 VMEAN = INTEGRAL/SUM.VOLUME
1197 ELSE
1198 VMEAN = 0. _EB
1199 ENDF
1200 VBAR = V0*EVALUATE.RAMP(T,DUMMY,L.RAMP.V0.T)*EVALUATE.RAMP(ZC(K),DUMMY,L.RAMP.V0.Z)
1201 DV.FORCING = (VBAR-VMEAN)/DT.LOC
1202 ! Apply the average force term to bulk of domain, and apply more aggressive forcing at boundary
1203 J.LO = 0
1204 J.HI = JBAR
1205 IF (APPLY.SPONGE.LAYER(2)) THEN
1206 FVY(:,0:NSC-1,K) = FVY(:,0:NSC-1,K) - (VBAR-VV(:,0:NSC-1,K))/DT.LOC
1207 J.LO = NSC
1208 ENDF
1209 IF (APPLY.SPONGE.LAYER(-2)) THEN
1210 FVY(:,JBAR-NSC+1:JBAR,K) = FVY(:,JBAR-NSC+1:JBAR,K) - (VBAR-VV(:,JBAR-NSC+1:JBAR,K))/DT.LOC
1211 J.HI = JBAR-NSC
1212 ENDF
1213 FVY(:,J.LO:J.HI,K) = FVY(:,J.LO:J.HI,K) - DV.FORCING
1214 ENDDO K.LOOP.V
1215 END SELECT SELECT.RAMP.V
1216 ENDF MEAN.FORCING.Y
1217
1218 MEAN.FORCING.Z: IF (MEAN.FORCING(3)) THEN
1219 SELECT.RAMP.W: SELECT CASE(L.RAMP.W0.Z)
1220 CASE(0) SELECT.RAMP.W
1221 INTEGRAL = 0. _EB
1222 SUM.VOLUME = 0. _EB
1223 DO K=0,KBAR
1224 DO J=1,JBAR
1225 DO I=1,IBAR
1226 IC1 = CELL.INDEX(I,J,K)
1227 IC2 = CELL.INDEX(I,J,K+1)
1228 IF (SOLID(IC1)) CYCLE
1229 IF (SOLID(IC2)) CYCLE
1230 IF (.NOT.MEAN.FORCING.CELL(I,J,K) ) CYCLE
1231 IF (.NOT.MEAN.FORCING.CELL(I,J,K+1)) CYCLE
1232 VC = DX(I)*DY(J)*DZN(K)
1233 INTEGRAL = INTEGRAL + WW(I,J,K)*VC
1234 SUM.VOLUME = SUM.VOLUME + VC
1235 ENDDO
1236 ENDDO
1237 ENDDO
1238 IF (SUM.VOLUME>TWO.EPSILON.EB) THEN
1239 WMEAN = INTEGRAL/SUM.VOLUME
1240 ELSE
1241 WMEAN = 0. _EB
1242 ENDF
1243 WBAR = W0*EVALUATE.RAMP(T,DUMMY,L.RAMP.W0.T)
1244 DW.FORCING = (WBAR-WMEAN)/DT.LOC
1245 DO K=0,KBAR
1246 DO J=1,JBAR
1247 DO I=1,IBAR
1248 IF (.NOT.MEAN.FORCING.CELL(I,J,K) ) CYCLE

```

```

1249 IF (.NOT.MEAN.FORCING.CELL(I,J,K+1)) CYCLE
1250 FVZ(I,J,K) = FVZ(I,J,K) - DW.FORCING
1251 ENDDO
1252 ENDDO
1253 ENDDO
1254 CASE(1:)
1255 K.LOOP.W: DO K=0,KBAR
1256 INTEGRAL = 0._EB
1257 SUM.VOLUME = 0._EB
1258 DO J=1,JBAR
1259 DO I=1,Iبار
1260 IC1 = CELL.INDEX(I,J,K)
1261 IC2 = CELL.INDEX(I,J,K+1)
1262 IF (SOLID(IC1)) CYCLE
1263 IF (SOLID(IC2)) CYCLE
1264 VC = DX(I)*DY(J)*DZN(K)
1265 INTEGRAL = INTEGRAL + WW(I,J,K)*VC
1266 SUM.VOLUME = SUM.VOLUME + VC
1267 ENDDO
1268 ENDDO
1269 IF (SUM.VOLUME>TWO.EPSILON.EB) THEN
1270 WMEAN = INTEGRAL/SUM.VOLUME
1271 ELSE
1272 WMEAN = 0._EB
1273 ENDF
1274 WBAR = W0*EVALUATE.RAMP(T,DUMMY,I,RAMP.W0.T)*EVALUATE.RAMP(Z(K),DUMMY,I,RAMP.W0.Z)
1275 DW.FORCING = (WBAR-WMEAN)/DT.LOC
1276 ! Apply the average force term to bulk of domain, and apply more aggressive forcing at boundary
1277 IF (APPLY.SPONGE.LAYER(-3).AND. K==Kبار) THEN
1278 DO J=1,Jبار
1279 DO I=1,Iبار
1280 FVZ(I,J,K) = FVZ(I,J,K) - (WBAR-WW(I,J,K))/DT.LOC
1281 ENDDO
1282 ENDDO
1283 ELSE
1284 FVZ(:, :, K) = FVZ(:, :, K) - DW.FORCING
1285 ENDF
1286 ENDDO K.LOOP.W
1287 END SELECT SELECT.RAMP.W
1288 ENDF MEAN.FORCING.Z
1289
1290 END SUBROUTINE MOMENTUM.NUDGING
1291
1292
1293 SUBROUTINE DIRECT.FORCE()
1294 REAL(EB) :: TIME.RAMP.FACTOR
1295
1296 TIME.RAMP.FACTOR = EVALUATE.RAMP(T,DUMMY,I,RAMP.FVX.T)
1297 !$OMP PARALLEL DO PRIVATE(RRHO) SCHEDULE(STATIC)
1298 DO K=1,Kبار
1299 DO J=1,Jبار
1300 DO I=0,Iبار
1301 RRHO = 2._EB/(RHOP(I,J,K)+RHOP(I+1,J,K))
1302 FVX(I,J,K) = FVX(I,J,K) - RRHO*FVEC(1)*TIME.RAMP.FACTOR
1303 ENDDO
1304 ENDDO
1305 ENDDO
1306 !$OMP END PARALLEL DO
1307
1308 TIME.RAMP.FACTOR = EVALUATE.RAMP(T,DUMMY,I,RAMP.FVY.T)
1309 !$OMP PARALLEL DO PRIVATE(RRHO) SCHEDULE(STATIC)
1310 DO K=1,Kبار
1311 DO J=0,Jبار
1312 DO I=1,Iبار
1313 RRHO = 2._EB/(RHOP(I,J,K)+RHOP(I,J+1,K))
1314 FVY(I,J,K) = FVY(I,J,K) - RRHO*FVEC(2)*TIME.RAMP.FACTOR
1315 ENDDO
1316 ENDDO
1317 ENDDO
1318 !$OMP END PARALLEL DO
1319
1320 TIME.RAMP.FACTOR = EVALUATE.RAMP(T,DUMMY,I,RAMP.FVZ.T)
1321 !$OMP PARALLEL DO PRIVATE(RRHO) SCHEDULE(STATIC)
1322 DO K=0,Kبار
1323 DO J=1,Jبار
1324 DO I=1,Iبار
1325 RRHO = 2._EB/(RHOP(I,J,K)+RHOP(I,J,K+1))
1326 FVZ(I,J,K) = FVZ(I,J,K) - RRHO*FVEC(3)*TIME.RAMP.FACTOR
1327 ENDDO
1328 ENDDO
1329 ENDDO
1330 !$OMP END PARALLEL DO
1331
1332 END SUBROUTINE DIRECT.FORCE
1333
1334 SUBROUTINE CORIOLIS.FORCE()
1335
1336

```

Source Code files for edited portions of FDS

```

1337 REAL(EB), POINTER, DIMENSION(:, :, :) :: UP=>NULL(), VP=>NULL(), WP=>NULL()
1338 REAL(EB) :: UBAR, VBAR, WBAR
1339 INTEGER :: II, JJ, KK, IW
1340 TYPE(WALLTYPE), POINTER :: WC=>NULL()
1341
1342 ! Velocities relative to the p-cell center (same work done in Deardorff eddy viscosity)
1343
1344 UP => WORK7
1345 VP => WORK8
1346 WP => WORK9
1347 UP=0._EB
1348 VP=0._EB
1349 WP=0._EB
1350
1351 !$OMP PARALLEL DO SCHEDULE(static)
1352 DO K=1,KBAR
1353 DO J=1,JBAR
1354 DO I=1,IBAR
1355 IF (SOLID(CELL_INDEX(I, J, K))) CYCLE
1356 UP(I, J, K) = 0.5_EB*(UU(I, J, K) + UU(I-1, J, K))
1357 VP(I, J, K) = 0.5_EB*(VV(I, J, K) + VV(I, J-1, K))
1358 WP(I, J, K) = 0.5_EB*(WW(I, J, K) + WW(I, J, K-1))
1359 ENDDO
1360 ENDDO
1361 ENDDO
1362 !$OMP END PARALLEL DO
1363
1364 DO IW=1, N_EXTERNAL_WALL_CELLS
1365 WC=>WALL(IW)
1366 II = WC%ONE_D%II
1367 JJ = WC%ONE_D%JJ
1368 KK = WC%ONE_D%KK
1369 UP(II, JJ, KK) = U_GHOST(IW)
1370 VP(II, JJ, KK) = V_GHOST(IW)
1371 WP(II, JJ, KK) = W_GHOST(IW)
1372 ENDDO
1373
1374 ! x momentum
1375
1376 !$OMP PARALLEL DO PRIVATE(VBAR, WBAR) SCHEDULE(STATIC)
1377 DO K=1,KBAR
1378 DO J=1,JBAR
1379 DO I=0,IBAR
1380 VBAR = 0.5_EB*(VP(I, J, K)+VP(I+1, J, K))
1381 WBAR = 0.5_EB*(WP(I, J, K)+WP(I+1, J, K))
1382 FVX(I, J, K) = FVX(I, J, K) + 2._EB*(OVEC(2)*WBAR-OVEC(3)*VBAR)
1383 ENDDO
1384 ENDDO
1385 ENDDO
1386 !$OMP END PARALLEL DO
1387
1388 ! y momentum
1389
1390 !$OMP PARALLEL DO PRIVATE(UBAR, WBAR) SCHEDULE(STATIC)
1391 DO K=1,KBAR
1392 DO J=0,JBAR
1393 DO I=1,IBAR
1394 UBAR = 0.5_EB*(UP(I, J, K)+UP(I, J+1, K))
1395 WBAR = 0.5_EB*(WP(I, J, K)+WP(I, J+1, K))
1396 FVY(I, J, K) = FVY(I, J, K) + 2._EB*(OVEC(3)*UBAR - OVEC(1)*WBAR)
1397 ENDDO
1398 ENDDO
1399 ENDDO
1400 !$OMP END PARALLEL DO
1401
1402 ! z momentum
1403
1404 !$OMP PARALLEL DO PRIVATE(UBAR, VBAR) SCHEDULE(STATIC)
1405 DO K=0,KBAR
1406 DO J=1,JBAR
1407 DO I=1,IBAR
1408 UBAR = 0.5_EB*(UP(I, J, K)+UP(I, J, K+1))
1409 VBAR = 0.5_EB*(VP(I, J, K)+VP(I, J, K+1))
1410 FVZ(I, J, K) = FVZ(I, J, K) + 2._EB*(OVEC(1)*VBAR - OVEC(2)*UBAR)
1411 ENDDO
1412 ENDDO
1413 ENDDO
1414 !$OMP END PARALLEL DO
1415
1416 END SUBROUTINE CORIOLIS_FORCE
1417
1418
1419 SUBROUTINE VEGETATION_DRAG()
1420
1421 VEG_DRAG(0, :) = VEG_DRAG(1, :)
1422 K=1
1423 DO J=1,JBAR
1424 DO I=0,IBAR

```

Source Code files for edited portions of FDS

```

1425 VEGLMAG = SQRT(UU(I,J,K)**2 + VV(I,J,K)**2 + WW(I,J,K)**2) ! VEGLMAG=2.*EB*KRES(I,J,K)
1426 FVX(I,J,K) = FVX(I,J,K) + VEG.DRAG(I,J)*VEGLMAG*UU(I,J,K)
1427 ENDDO
1428 ENDDO
1429
1430 VEG.DRAG(:,0) = VEG.DRAG(:,1)
1431 DO J=0,JBAR
1432 DO I=1,IBAR
1433 VEGLMAG = SQRT(UU(I,J,K)**2 + VV(I,J,K)**2 + WW(I,J,K)**2)
1434 FVY(I,J,K) = FVY(I,J,K) + VEG.DRAG(I,J)*VEGLMAG*VV(I,J,K)
1435 ENDDO
1436 ENDDO
1437
1438 DO J=1,JBAR
1439 DO I=1,IBAR
1440 VEGLMAG = SQRT(UU(I,J,K)**2 + VV(I,J,K)**2 + WW(I,J,K)**2)
1441 FVZ(I,J,K) = FVZ(I,J,K) + VEG.DRAG(I,J)*VEGLMAG*WV(I,J,K)
1442 ENDDO
1443 ENDDO
1444
1445 END SUBROUTINE VEGETATION_DRAG
1446
1447 END SUBROUTINE VELOCITY_FLUX
1448
1449
1450 SUBROUTINE MMS_VELOCITY_FLUX(NM,T)
1451
1452 ! Shunn et al., JCP (2012) prob 3
1453
1454 USE MANUFACTURED_SOLUTIONS, ONLY: VD2D.MMS.U.SRC.3, VD2D.MMS.V.SRC.3
1455 INTEGER, INTENT(IN) :: NM
1456 REAL(EB), INTENT(IN) :: T
1457 INTEGER :: I,J,K
1458 REAL(EB), POINTER, DIMENSION(:,:,:) :: UU=>NULL(),VV=>NULL()
1459
1460 CALL POINT_TO_MESH(NM)
1461
1462 IF (PREDICTOR) THEN
1463 UU=>U
1464 VV=>V
1465 ELSE
1466 UU=>US
1467 VV=>VS
1468 ENDF
1469
1470 DO K=1,KBAR
1471 DO J=1,JBAR
1472 DO I=0,IBAR
1473 FVX(I,J,K) = FVX(I,J,K) - VD2D.MMS.U.SRC.3(X(I),ZC(K),T)
1474 ENDDO
1475 ENDDO
1476 ENDDO
1477
1478 DO K=0,KBAR
1479 DO J=1,JBAR
1480 DO I=1,IBAR
1481 FVZ(I,J,K) = FVZ(I,J,K) - VD2D.MMS.V.SRC.3(XC(I),ZC(K),T)
1482 ENDDO
1483 ENDDO
1484 ENDDO
1485
1486 END SUBROUTINE MMS_VELOCITY_FLUX
1487
1488
1489 SUBROUTINE VELOCITY_FLUX_CYLINDRICAL(T,NM)
1490
1491 ! Compute convective and diffusive terms for 2D axisymmetric
1492
1493 USE MATH_FUNCTIONS, ONLY: EVALUATE_RAMP
1494 REAL(EB) :: T,DMUDX
1495 INTEGER :: IO
1496 INTEGER, INTENT(IN) :: NM
1497 REAL(EB) :: MU,Y,UP,UM,WP,WM,VIRM,DTXZDZ,DTXZDX,DUDX,DWDZ,DUDZ,DWDZ,WOMY,UOMY,OMYP,OMM,TXZP,TXZM, &
1498 AH,RRHO,GX,GZ,TXXP,TXXM,TZZP,TZZM,DTXXDX,DTZZDZ,DUMMY=.EB
1499 INTEGER :: I,J,K,IEYP,IEYM,IC
1500 REAL(EB), POINTER, DIMENSION(:,:,:) :: TXZ=>NULL(),OMY=>NULL(),UU=>NULL(),VV=>NULL(),RHOP=>NULL(),DP=>NULL()
1501
1502 CALL POINT_TO_MESH(NM)
1503
1504 IF (PREDICTOR) THEN
1505 UU => U
1506 VV => V
1507 DP => D
1508 RHOP => RHOP
1509 ELSE
1510 UU => US
1511 VV => VS
1512 DP => DS

```

```

1513 RHOP => RHOS
1514 ENDIF
1515
1516 TXZ => WORK2
1517 OMY => WORK5
1518
1519 ! Compute vorticity and stress tensor components
1520
1521 DO K=0,KBAR
1522 DO J=0,IBAR
1523 DO I=0,IBAR
1524 DUDZ = RDZN(K)*(UU(I,J,K+1)-UU(I,J,K))
1525 DWDX = RDZN(I)*(WW(I+1,J,K)-WW(I,J,K))
1526 OMY(I,J,K) = DUDZ - DWDX
1527 MLY = 0.25*EB*(MU(I+1,J,K)+MU(I,J,K)+MU(I,J,K+1)+MU(I+1,J,K+1))
1528 TXZ(I,J,K) = MLY*(DUDZ + DWDX)
1529 ENDDO
1530 ENDDO
1531 ENDDO
1532
1533 ! Compute gravity components
1534
1535 GX = 0.*EB
1536 GZ = EVALUATERAMP(T,DUMMY,I,RAMP,GZ)*GVEC(3)
1537
1538 ! Compute r-direction flux term FVX
1539
1540 IF (ABS(XS)<=TWO*EPSILON*EB) THEN
1541 I0 = 1
1542 ELSE
1543 I0 = 0
1544 ENDIF
1545
1546 J = 1
1547
1548 DO K= 1,KBAR
1549 DO I=I0,IBAR
1550 WP = WW(I,J,K) + WW(I+1,J,K)
1551 WM = WW(I,J,K-1) + WW(I+1,J,K-1)
1552 OMY = OMY(I,J,K)
1553 OMM = OMY(I,J,K-1)
1554 TXZP = TXZ(I,J,K)
1555 TXZM = TXZ(I,J,K-1)
1556 IC = CELLINDEX(I,J,K)
1557 IEYP = EDGEINDEX(8,IC)
1558 IEYM = EDGEINDEX(6,IC)
1559 IF (OME.E(-1,IEYP)>-1.E5*EB) THEN
1560 OMP = OME.E(-1,IEYP)
1561 TXZP = TAU.E(-1,IEYP)
1562 ENDF
1563 IF (OME.E( 1,IEYM)>-1.E5*EB) THEN
1564 OMM = OME.E( 1,IEYM)
1565 TXZM = TAU.E( 1,IEYM)
1566 ENDF
1567 WCMY = WP*OMY + WM*OMM
1568 RRHO = 2.*EB/(RHOP(I,J,K)+RHOP(I+1,J,K))
1569 AH = RHO.0(K)*RRHO - 1.*EB
1570 DWDZ = (WW(I+1,J,K)-WW(I+1,J,K-1))*RDZ(K)
1571 TXXP = MU(I+1,J,K)*(FOIH*DP(I+1,J,K) - 2.*EB*DWDZ)
1572 DWDZ = (WW(I,J,K)-WW(I,J,K-1))*RDZ(K)
1573 TXXM = MU(I,J,K)*(FOIH*DP(I,J,K) - 2.*EB*DWDZ)
1574 DTXXDX = RDZN(I)*(TXXP-TXXM)
1575 DTXZDZ = RDZ(K)*(TXZP-TXZM)
1576 DMUDX = (MU(I+1,J,K)-MU(I,J,K))*RDZN(I)
1577 VIRM = RRHO*(DTXXDX + DTXZDZ - 2.*EB*UU(I,J,K)*DMUDX/R(I))
1578 FVX(I,J,K) = 0.25*EB*WCMY + GX*AH - VIRM
1579 ENDDO
1580 ENDDO
1581
1582 ! Compute z-direction flux term FVZ
1583
1584 DO K=0,KBAR
1585 DO I=1,IBAR
1586 UP = UU(I,J,K) + UU(I,J,K+1)
1587 UM = UU(I-1,J,K) + UU(I-1,J,K+1)
1588 OMY = OMY(I,J,K)
1589 OMM = OMY(I-1,J,K)
1590 TXZP = TXZ(I,J,K)
1591 TXZM = TXZ(I-1,J,K)
1592 IC = CELLINDEX(I,J,K)
1593 IEYP = EDGEINDEX(8,IC)
1594 IEYM = EDGEINDEX(7,IC)
1595 IF (OME.E(-2,IEYP)>-1.E5*EB) THEN
1596 OMP = OME.E(-2,IEYP)
1597 TXZP = TAU.E(-2,IEYP)
1598 ENDF
1599 IF (OME.E( 2,IEYM)>-1.E5*EB) THEN
1600 OMM = OME.E( 2,IEYM)

```

```

1601 TXZM = TAU.E( 2,IEYM)
1602 ENDIF
1603 UOMY = UP*OMYP + UM*CMM
1604 RRHO = 2.*EB/(RHOP(I,J,K)+RHOP(I,J,K+1))
1605 AH = 0.5*EB*(RHO.0(K)+RHO.0(K+1))*RRHO - 1.*EB
1606 DUDX = (R(I)*UU(I,J,K+1)-R(I-1)*UU(I-1,J,K+1))*RDX(1)*RRN(1)
1607 TZZP = MU(I,J,K+1)*( FOTH*DP(I,J,K+1) - 2.*EB*DUDX )
1608 DUDX = (R(I)*UU(I,J,K)-R(I-1)*UU(I-1,J,K))*RDX(1)*RRN(1)
1609 TZZM = MU(I,J,K) *( FOTH*DP(I,J,K) - 2.*EB*DUDX )
1610 DTXZDX= RDX(1) *(R(I)*TXZP-R(I-1)*TXZM)*RRN(1)
1611 DTZZDZ= RDZN(K)*( TZZP -TZZM)
1612 VIRM = RRHO*(DTXZDX + DTZZDZ)
1613 FVZ(I,J,K) = -0.25*EB*UOMY + CZ*AH - VIRM
1614 ENDDO
1615 ENDDO
1616
1617 ! Adjust FVX and FVZ at solid , internal obstructions for no flux
1618
1619 END SUBROUTINE VELOCITY_FLUX_CYLINDRICAL
1620
1621
1622 SUBROUTINE NO_FLUX(DT,NM)
1623
1624 ! Set FVX,FVY,FVZ inside and on the surface of solid obstructions to maintain no flux
1625
1626 USE MATH_FUNCTIONS, ONLY: EVALUATE_RAMP
1627 INTEGER, INTENT(IN) :: NM
1628 REAL(EB), INTENT(IN) :: DT
1629 REAL(EB), POINTER, DIMENSION(:,:,:), :: HP=>NULL(), OMLHP=>NULL()
1630 REAL(EB) :: RFODT,H_OTHER,DUUDT,DVVDT,DWWDT,UN,TNOW,DHFC
1631 INTEGER :: IC2,IC1,N,I,J,K,IW,II,JJ,KK,IOR,N.INT.CELLS,IIO,JJO,KKO,NOM
1632 TYPE (OBSTRUCTION_TYPE), POINTER :: OB=>NULL()
1633 TYPE (WALL_TYPE), POINTER :: WC=>NULL()
1634 TYPE (EXTERNAL_WALL_TYPE), POINTER :: EWC=>NULL()
1635
1636 TNOW=SECOND()
1637 CALL POINT_TO_MESH(NM)
1638
1639 RFODT = RELAXATION_FACTOR/DT
1640
1641 IF (PREDICTOR) HP => H
1642 IF (CORRECTOR) HP => HS
1643
1644 ! Exchange H at interpolated boundaries
1645
1646 NO_SCARC_IF: IF (PRES.METHOD /= 'SCARC') THEN
1647
1648 DO IW=1,N,EXTERNAL_WALL.CELLS
1649 WC=>WALL(IW)
1650 EWC=>EXTERNAL_WALL(IW)
1651 NOM =EWC%NOM
1652 IF (NOM/=0) CYCLE
1653 IF (PREDICTOR) THEN
1654 OMLHP=>OMESH(NOM)%H
1655 ELSE
1656 OMLHP=>OMESH(NOM)%HS
1657 ENDIF
1658 II = WC%ONE.D%II
1659 JJ = WC%ONE.D%JJ
1660 KK = WC%ONE.D%KK
1661 H_OTHER = 0.*EB
1662 DO KKO=EWC%KKO.MIN,EWC%KKO.MAX
1663 DO JJO=EWC%JJO.MIN,EWC%JJO.MAX
1664 DO IIO=EWC%IIO.MIN,EWC%IIO.MAX
1665 H_OTHER = H_OTHER + OMLHP(IIO,JJO,KKO)
1666 ENDDO
1667 ENDDO
1668 ENDDO
1669 N.INT.CELLS = (EWC%IIO.MAX-EWC%IIO.MIN+1) * (EWC%JJO.MAX-EWC%JJO.MIN+1) * (EWC%KKO.MAX-EWC%KKO.MIN+1)
1670 HP(II,JJ,KK) = H_OTHER/REAL(N.INT.CELLS,EB)
1671 ENDDO
1672
1673 ENDIF NO_SCARC_IF
1674
1675 ! Set FVX, FVY and FVZ to drive velocity components at solid boundaries within obstructions towards zero
1676
1677 OBST_LOOP: DO N=1,N.OBST
1678
1679 OB=>OBSTRUCTION(N)
1680
1681 DO K=OB%K1+1,OB%K2
1682 DO J=OB%J1+1,OB%J2
1683 DO I=OB%I1 ,OB%I2
1684 IC1 = CELL_INDEX(I,J,K)
1685 IC2 = CELL_INDEX(I+1,J,K)
1686 IF (SOLID(IC1) .AND. SOLID(IC2)) THEN
1687 IF (PREDICTOR) THEN
1688 DUUDT = -RFODT*U(I,J,K)

```



```

1689 ELSE
1690 DUUDT = -RFODT*(U(I,J,K)+US(I,J,K))
1691 ENDIF
1692 FVX(I,J,K) = -RDXN(1)*(HP(I+1,J,K)-HP(I,J,K)) - DUUDT
1693 ENDIF
1694 ENDDO
1695 ENDDO
1696 ENDDO
1697
1698 DO K=OB%K1+1,OB%K2
1699 DO J=OB%J1 ,OB%J2
1700 DO I=OB%I1+1,OB%I2
1701 IC1 = CELL_INDEX(I,J,K)
1702 IC2 = CELL_INDEX(I,J+1,K)
1703 IF (SOLID(IC1) .AND. SOLID(IC2)) THEN
1704 IF (PREDICTOR) THEN
1705 DVVDT = -RFODT*V(I,J,K)
1706 ELSE
1707 DVVDT = -RFODT*(V(I,J,K)+VS(I,J,K))
1708 ENDIF
1709 FVY(I,J,K) = -RDYN(J)*(HP(I,J+1,K)-HP(I,J,K)) - DVVDT
1710 ENDIF
1711 ENDDO
1712 ENDDO
1713 ENDDO
1714
1715 DO K=OB%K1 ,OB%K2
1716 DO J=OB%J1+1,OB%J2
1717 DO I=OB%I1+1,OB%I2
1718 IC1 = CELL_INDEX(I,J,K)
1719 IC2 = CELL_INDEX(I,J,K+1)
1720 IF (SOLID(IC1) .AND. SOLID(IC2)) THEN
1721 IF (PREDICTOR) THEN
1722 DWWDT = -RFODT*W(I,J,K)
1723 ELSE
1724 DWWDT = -RFODT*(W(I,J,K)+WS(I,J,K))
1725 ENDIF
1726 FVZ(I,J,K) = -RDZN(K)*(HP(I,J,K+1)-HP(I,J,K)) - DWWDT
1727 ENDIF
1728 ENDDO
1729 ENDDO
1730 ENDDO
1731
1732 ENDDO OBST_LOOP
1733
1734 ! Set FVX, FVY and FVZ to drive the normal velocity at solid boundaries towards the specified value (LW or UWS)
1735 DHFCT=1._EB
1736 IF (.NOT. PRES.ON.WHOLE.DOMAIN) DHFCT=0._EB
1737
1738 WALL_LOOP: DO IW=1,N.EXTERNAL.WALL.CELLS+N.INTERNAL.WALL.CELLS
1739
1740 WC => WALL(IW)
1741
1742 IF (WC%BOUNDARY.TYPE==INTERPOLATED.BOUNDARY .OR. WC%BOUNDARY.TYPE==OPEN.BOUNDARY) CYCLE WALL_LOOP
1743
1744 IF (IW<=N.EXTERNAL.WALL.CELLS) THEN
1745 NCM = EXTERNAL.WALL(IW)%NCM
1746 ELSE
1747 NCM = 0
1748 ENDIF
1749
1750 IF (IW>N.EXTERNAL.WALL.CELLS .AND. WC%BOUNDARY.TYPE==NULL.BOUNDARY .AND. NCM==0) CYCLE WALL_LOOP
1751
1752 II = WC%ONE.D%II
1753 JJ = WC%ONE.D%JJ
1754 KK = WC%ONE.D%KK
1755 IOR = WC%ONE.D%IOR
1756
1757 IF (NCM/=0 .OR. WC%BOUNDARY.TYPE==SOLID.BOUNDARY .OR. WC%BOUNDARY.TYPE==NULL.BOUNDARY) THEN
1758 IF (PREDICTOR) THEN
1759 UN = -SIGN(1._EB,REAL(IOR,EB))*WC%ONE.D%UWS
1760 ELSE
1761 UN = -SIGN(1._EB,REAL(IOR,EB))*WC%ONE.D%LW
1762 ENDIF
1763 SELECT CASE(IOR)
1764 CASE( 1)
1765 IF (PREDICTOR) THEN
1766 DUUDT = RFODT*(UN-U(II,JJ,KK))
1767 ELSE
1768 DUUDT = 2._EB*RFODT*(UN-0.5._EB*(U(II,JJ,KK)+US(II,JJ,KK)))
1769 ENDIF
1770 FVX(II,JJ,KK) = -RDXN(II)*(HP(II+1,JJ,KK)-HP(II,JJ,KK))*DHFCT - DUUDT
1771 CASE(-1)
1772 IF (PREDICTOR) THEN
1773 DUUDT = RFODT*(UN-U(II-1,JJ,KK))
1774 ELSE
1775 DUUDT = 2._EB*RFODT*(UN-0.5._EB*(U(II-1,JJ,KK)+US(II-1,JJ,KK)))
1776 ENDIF
1777

```

Source Code files for edited portions of FDS

```

1777 FVX(II-1, JJ, KK) = -RDYN(II-1)*(HP(II, JJ, KK)-HP(II-1, JJ, KK))*DHFCT - DUVDI
1778 CASE( 2)
1779 IF (PREDICTOR) THEN
1780 DVVDI = RFODT*(UN-V(II, JJ, KK))
1781 ELSE
1782 DVVDI = 2. _EB*RFODT*(UN-0.5 _EB*(V(II, JJ, KK)+VS(II, JJ, KK)) )
1783 ENDIF
1784 FVY(II, JJ, KK) = -RDYN(JJ)*(HP(II, JJ+1, KK)-HP(II, JJ, KK))*DHFCT - DVVDI
1785 CASE(-2)
1786 IF (PREDICTOR) THEN
1787 DVVDI = RFODT*(UN-V(II, JJ-1, KK))
1788 ELSE
1789 DVVDI = 2. _EB*RFODT*(UN-0.5 _EB*(V(II, JJ-1, KK)+VS(II, JJ-1, KK)) )
1790 ENDIF
1791 FVZ(II, JJ-1, KK) = -RDYN(JJ-1)*(HP(II, JJ, KK)-HP(II, JJ-1, KK))*DHFCT - DVVDI
1792 CASE( 3)
1793 IF (PREDICTOR) THEN
1794 DWWDI = RFODT*(UN-W(II, JJ, KK))
1795 ELSE
1796 DWWDI = 2. _EB*RFODT*(UN-0.5 _EB*(W(II, JJ, KK)+WS(II, JJ, KK)) )
1797 ENDIF
1798 FVZ(II, JJ, KK) = -RDZN(KK)*(HP(II, JJ, KK+1)-HP(II, JJ, KK))*DHFCT - DWWDI
1799 CASE(-3)
1800 IF (PREDICTOR) THEN
1801 DWWDI = RFODT*(UN-W(II, JJ, KK-1))
1802 ELSE
1803 DWWDI = 2. _EB*RFODT*(UN-0.5 _EB*(W(II, JJ, KK-1)+WS(II, JJ, KK-1)) )
1804 ENDIF
1805 FVZ(II, JJ, KK-1) = -RDZN(KK-1)*(HP(II, JJ, KK)-HP(II, JJ, KK-1))*DHFCT - DWWDI
1806 END SELECT
1807 ENDIF
1808
1809 IF (WC@BOUNDARY_TYPE==MIRROR_BOUNDARY) THEN
1810 SELECT CASE(IOR)
1811 CASE( 1)
1812 FVX(II, JJ, KK) = 0. _EB
1813 CASE(-1)
1814 FVX(II-1, JJ, KK) = 0. _EB
1815 CASE( 2)
1816 FVY(II, JJ, KK) = 0. _EB
1817 CASE(-2)
1818 FVY(II, JJ-1, KK) = 0. _EB
1819 CASE( 3)
1820 FVZ(II, JJ, KK) = 0. _EB
1821 CASE(-3)
1822 FVZ(II, JJ, KK-1) = 0. _EB
1823 END SELECT
1824 ENDIF
1825
1826 ENDDO WALL_LOOP
1827
1828 T_USED(4)=T_USED(4)+SECOND()-TNOW
1829 END SUBROUTINE NO_FLUX
1830
1831
1832 SUBROUTINE VELOCITY_PREDICTOR(T, DT, DT_NEW, NM)
1833
1834 USE TURBULENCE, ONLY: COMPRESSION_WAVE
1835 USE MANUFACTURED_SOLUTIONS, ONLY: UF_MMS, WF_MMS, VD2D_MMS_U, VD2D_MMS_V
1836 USE COMPLEX_GEOMETRY, ONLY: CCIBM_VELOCITY_NO_GRADH
1837
1838 ! Estimates the velocity components at the next time step
1839
1840 REAL(EB) :: TNOW, XHAT, ZHAT
1841 INTEGER :: I, J, K
1842 INTEGER, INTENT(IN) :: NM
1843 REAL(EB), INTENT(IN) :: T, DT
1844 REAL(EB) :: DT_NEW(NMESHES)
1845
1846 IF (SOLID_PHASE_ONLY) RETURN
1847 IF (PERIODIC_TEST==4) THEN
1848 CALL COMPRESSION_WAVE(NM, T, 4)
1849 CALL CHECK_STABILITY(DT, DT_NEW, NM)
1850 RETURN
1851 ENDIF
1852
1853 TNOW=SECOND()
1854 CALL POINT_TO_MESH(NM)
1855
1856 FREEZE_VELOCITY_IF: IF (FREEZE_VELOCITY) THEN
1857 US = U
1858 VS = V
1859 WS = W
1860 ELSE FREEZE_VELOCITY_IF
1861
1862 DO K=1, KBAR
1863 DO J=1, JBAR
1864 DO I=0, IBAR

```

Source Code files for edited portions of FDS

```

1865 US(I,J,K) = U(I,J,K) - DT*( FVX(I,J,K) + RDXN(I)*(H(I+1,J,K)-H(I,J,K)) )
1866 ENDDO
1867 ENDDO
1868 ENDDO
1869
1870 DO K=1,KBAR
1871 DO J=0,JBAR
1872 DO I=1,IBAR
1873 VS(I,J,K) = V(I,J,K) - DT*( FVY(I,J,K) + RDYN(J)*(H(I,J+1,K)-H(I,J,K)) )
1874 ENDDO
1875 ENDDO
1876 ENDDO
1877
1878 DO K=0,KBAR
1879 DO J=1,JBAR
1880 DO I=1,IBAR
1881 WS(I,J,K) = W(I,J,K) - DT*( FVZ(I,J,K) + RDZN(K)*(H(I,J,K+1)-H(I,J,K)) )
1882 ENDDO
1883 ENDDO
1884 ENDDO
1885
1886 IF (PRES.METHOD == 'GLMAT') CALL WALL_VELOCITY_NO_GRADH(DT, .FALSE.)
1887 IF (CC_IBM) CALL CCIBM_VELOCITY_NO_GRADH(DT)
1888
1889 ENDIF FREEZE_VELOCITY_IF
1890
1891 ! Manufactured solution (debug)
1892
1893 IF (PERIODIC_TEST==7 .AND. .FALSE.) THEN
1894 DO K=1,KBAR
1895 DO J=1,JBAR
1896 DO I=0,IBAR
1897 XHAT = X(I) - UF.MMS*(T)
1898 ZHAT = ZC(K) - WF.MMS*(T)
1899 US(I,J,K) = VD2D.MMS.U(XHAT,ZHAT,T)
1900 ENDDO
1901 ENDDO
1902 ENDDO
1903 DO K=0,KBAR
1904 DO J=1,JBAR
1905 DO I=1,IBAR
1906 XHAT = XC(I) - UF.MMS*(T)
1907 ZHAT = Z(K) - WF.MMS*(T)
1908 WS(I,J,K) = VD2D.MMS.V(XHAT,ZHAT,T)
1909 ENDDO
1910 ENDDO
1911 ENDDO
1912 ENDIF
1913
1914 ! No vertical velocity in Evacuation meshes
1915
1916 IF (EVACUATION_ONLY(NM)) WS = 0..EB
1917
1918 ! Check the stability criteria , and if the time step is too small , send back a signal to kill the job
1919
1920 CALL CHECK_STABILITY(DT,DT_NEW,NM)
1921
1922 IF (DT_NEW(NM)<DT_INITIAL*LIMITING_DT_RATIO .AND. (T+DT_NEW(NM)<(T_END-TWO_EPSILON_EB))) STOP_STATUS =
INSTABILITY_STOP
1923
1924 T_USED(4)=T_USED(4)+SECOND()-TNOW
1925 END SUBROUTINE VELOCITY_PREDICTOR
1926
1927
1928 SUBROUTINE VELOCITY_CORRECTOR(T,DT,NM)
1929
1930 USE TURBULENCE, ONLY: COMPRESSION_WAVE
1931 USE MANUFACTURED_SOLUTIONS, ONLY: UF.MMS,WF.MMS,VD2D.MMS.U,VD2D.MMS.V
1932 USE COMPLEX_GEOMETRY, ONLY : CCIBM_VELOCITY_NO_GRADH
1933
1934 ! Correct the velocity components
1935
1936 REAL(EB) :: TNOW,XHAT,ZHAT
1937 INTEGER :: I,J,K
1938 INTEGER, INTENT(IN) :: NM
1939 REAL(EB), INTENT(IN) :: T,DT
1940
1941 IF (SOLID_PHASE_ONLY) RETURN
1942 IF (PERIODIC_TEST==4) THEN
1943 CALL COMPRESSION_WAVE(NM,T,4)
1944 RETURN
1945 ENDIF
1946
1947 TNOW=SECOND()
1948 CALL POINT_TO_MESH(NM)
1949
1950 FREEZE_VELOCITY_IF: IF (FREEZE_VELOCITY) THEN
1951 U = US

```

Source Code files for edited portions of FDS

```

1952 V = VS
1953 W = WS
1954 ELSE FREEZE_VELOCITY_IF
1955
1956 IF (STORE_OLD_VELOCITY) THEN
1957 U_OLD = U
1958 V_OLD = V
1959 W_OLD = W
1960 ENDIF
1961
1962 IF (PRES_METHOD == 'GLMAT') CALL WALL_VELOCITY_NO_GRADH(DT, .TRUE.) ! Store U velocities on OBST surfaces.
1963
1964 DO K=1,KBAR
1965 DO J=1,JBAR
1966 DO I=0,IBAR
1967 U(I,J,K) = 0.5*_EB*( U(I,J,K) + US(I,J,K) - DT*(FVX(I,J,K) + RDXN(I)*(HS(I+1,J,K)-HS(I,J,K))) )
1968 ENDDO
1969 ENDDO
1970 ENDDO
1971
1972 DO K=1,KBAR
1973 DO J=0,JBAR
1974 DO I=1,IBAR
1975 V(I,J,K) = 0.5*_EB*( V(I,J,K) + VS(I,J,K) - DT*(FVY(I,J,K) + RDXN(J)*(HS(I,J+1,K)-HS(I,J,K))) )
1976 ENDDO
1977 ENDDO
1978 ENDDO
1979
1980 DO K=0,KBAR
1981 DO J=1,JBAR
1982 DO I=1,IBAR
1983 W(I,J,K) = 0.5*_EB*( W(I,J,K) + WS(I,J,K) - DT*(FVZ(I,J,K) + RDXN(K)*(HS(I,J,K+1)-HS(I,J,K))) )
1984 ENDDO
1985 ENDDO
1986 ENDDO
1987
1988 IF (PRES_METHOD == 'GLMAT') CALL WALL_VELOCITY_NO_GRADH(DT, .FALSE.)
1989 IF (CC_IBM) CALL CCIBM_VELOCITY_NO_GRADH(DT)
1990
1991 ENDIF FREEZE_VELOCITY_IF
1992
1993 ! Manufactured solution (debug)
1994
1995 IF (PERIODIC_TEST==7 .AND. .FALSE.) THEN
1996 DO K=1,KBAR
1997 DO J=1,JBAR
1998 DO I=0,IBAR
1999 XHAT = X(I) - UF.MMS*T
2000 ZHAT = ZC(K) - WF.MMS*T
2001 U(I,J,K) = VD2D.MMS.U(XHAT,ZHAT,T)
2002 ENDDO
2003 ENDDO
2004 ENDDO
2005 DO K=0,KBAR
2006 DO J=1,JBAR
2007 DO I=1,IBAR
2008 XHAT = XC(I) - UF.MMS*T
2009 ZHAT = Z(K) - WF.MMS*T
2010 W(I,J,K) = VD2D.MMS.V(XHAT,ZHAT,T)
2011 ENDDO
2012 ENDDO
2013 ENDDO
2014 ENDIF
2015
2016 ! No vertical velocity in Evacuation meshes
2017
2018 IF (EVACUATION_ONLY(NM)) W = 0._EB
2019
2020 T_USED(4)=T_USED(4)+SECOND()-TNOW
2021 END SUBROUTINE VELOCITY_CORRECTOR
2022
2023
2024 SUBROUTINE VELOCITY_BC(T,NM)
2025
2026 ! Assert tangential velocity boundary conditions
2027
2028 USE MATH_FUNCTIONS, ONLY: EVALUATE_RAMP
2029 USE TURBULENCE, ONLY: WALL_MODEL,WANNIER_FLOW
2030 REAL(EB), INTENT(IN) :: T
2031 REAL(EB) :: MUA,TSI,WGT,TNOW,RAMP,T,OMW,MU_WALL,RHO_WALL,SLIP_COEF,VEL_T,UBAR,VBAR,WBAR, &
2032 UUP(2),UUM(2),DX(2),MU_DUIDX(-2:2),DUIDX(-2:2),PROFILE_FACTOR,VEL_GAS,VEL_GHOST, &
2033 MU_DUIDX.USE(2),DUIDX.USE(2),VEL_EDDY,U_TAU,Y_PLUS,W1,WT2,DUMMY
2034 INTEGER :: I,J,K,NOM(2),HIO(2),JJO(2),KKO(2),IE,II,JI,KK,IEC,IOR,IWM,IWP,ICMM,ICMP,ICPP,IC,ICD,ICDO,IVL,
2035 LSGN,IS, &
2036 VELOCITY_BC_INDEX,IIGM,JJGM,KKGM,IIGP,JJGP,KKGP,SURF_INDEXM,SURF_INDEXP,ITMP,ICD_SGN,ICDO_SGN, &
2037 BOUNDARY_TYPE_M,BOUNDARY_TYPE_P,IS2,IWPI,IWMI,VENT_INDEX
2038 LOGICAL :: ALTERED_GRADIENT(-2:2),PROCESS_EDGE,SYNTHETIC_EDDY_METHOD,HVAC_TANGENTIAL,INTERPOLATED_EDGE
2039 INTEGER, INTENT(IN) :: NM

```

Source Code files for edited portions of FDS

```

2039 REAL(EB), POINTER, DIMENSION(:, :, :) :: UU=>NULL(), VV=>NULL(), WW=>NULL(), U.Y=>NULL(), U.Z=>NULL(), &
2040 V.X=>NULL(), V.Z=>NULL(), W.X=>NULL(), W.Y=>NULL(), RHOP=>NULL(), VELO.OTHER=>NULL()
2041 TYPE (SURFACE.TYPE), POINTER :: SF=>NULL()
2042 TYPE (OMESH.TYPE), POINTER :: CM=>NULL()
2043 TYPE (VENTS.TYPE), POINTER :: VT
2044 TYPE (WALL.TYPE), POINTER :: WCM/WCP
2045
2046 IF (SOLID.PHASE.ONLY) RETURN
2047
2048 TNCW = SECOND()
2049
2050 ! Assign local names to variables
2051
2052 CALL POINT.TO.MESH(NM)
2053
2054 ! Point to the appropriate velocity field
2055
2056 IF (PREDICTOR) THEN
2057   UU => US
2058   VV => VS
2059   WW => WS
2060   RHOP => RHOS
2061 ELSE
2062   UU => U
2063   VV => V
2064   WW => W
2065   RHOP => RHO
2066 ENDIF
2067
2068 ! Set the boundary velocity place holder to some large negative number
2069
2070 IF (CORRECTOR) THEN
2071   U.Y => WORK1
2072   U.Z => WORK2
2073   V.X => WORK3
2074   V.Z => WORK4
2075   W.X => WORK5
2076   W.Y => WORK6
2077   U.Y = -1.E6.EB
2078   U.Z = -1.E6.EB
2079   V.X = -1.E6.EB
2080   V.Z = -1.E6.EB
2081   W.X = -1.E6.EB
2082   W.Y = -1.E6.EB
2083   UVW.GHOST = -1.E6.EB
2084 ENDIF
2085
2086 ! Set OME.E and TAUE to very negative number
2087
2088 TAUE = -1.E6.EB
2089 OME.E = -1.E6.EB
2090
2091 ! Loop over all cell edges and determine the appropriate velocity BCs
2092
2093 EDGELoop: DO IE=1,N.EDGES
2094
2095   INTERPOLATED.EDGE = .FALSE.
2096
2097   ! Throw out edges that are completely surrounded by blockages or the exterior of the domain
2098
2099   PROCESS.EDGE = .FALSE.
2100   DO IS=5,8
2101     IF (.NOT.EXTERIOR(IJKE(IS,IE)) .AND. .NOT.SOLID(IJKE(IS,IE))) THEN
2102       PROCESS.EDGE = .TRUE.
2103     EXIT
2104     ENDIF
2105   ENDDO
2106   IF (.NOT.PROCESS.EDGE) CYCLE EDGELoop
2107
2108   ! If the edge is to be "smoothed," set tau and omega to zero and cycle
2109
2110   IF (EVACUATION.ONLY(NM)) THEN
2111     OME.E(:,IE) = 0..EB
2112     TAUE(:,IE) = 0..EB
2113     CYCLE EDGELoop
2114   ENDIF
2115
2116   ! Unpack indices for the edge
2117
2118   II = IJKE( 1,IE)
2119   JJ = IJKE( 2,IE)
2120   KK = IJKE( 3,IE)
2121   IEC = IJKE( 4,IE)
2122   ICMM = IJKE( 5,IE)
2123   ICPM = IJKE( 6,IE)
2124   ICMP = IJKE( 7,IE)
2125   ICPP = IJKE( 8,IE)
2126   NOM(1) = IJKE( 9,IE)

```

```

2127 IIO(1) = IJKE(10,IE)
2128 JJO(1) = IJKE(11,IE)
2129 KKO(1) = IJKE(12,IE)
2130 NCM(2) = IJKE(13,IE)
2131 IIO(2) = IJKE(14,IE)
2132 JJO(2) = IJKE(15,IE)
2133 KKO(2) = IJKE(16,IE)
2134
2135 ! Get the velocity components at the appropriate cell faces
2136
2137 COMPONENT: SELECT CASE(IEC)
2138 CASE(1) COMPONENT
2139 UUP(1) = VV(II, JJ, KK+1)
2140 UUM(1) = VV(II, JJ, KK)
2141 UUP(2) = WW(II, JJ+1, KK)
2142 UUM(2) = WW(II, JJ, KK)
2143 DXX(1) = DY(JJ)
2144 DXX(2) = DZ(KK)
2145 MUA = 0.25*EB*(MU(II, JJ, KK) + MU(II, JJ+1, KK) + MU(II, JJ+1, KK+1) + MU(II, JJ, KK+1) )
2146 CASE(2) COMPONENT
2147 UUP(1) = WW(II+1, JJ, KK)
2148 UUM(1) = WW(II, JJ, KK)
2149 UUP(2) = UU(II, JJ, KK+1)
2150 UUM(2) = UU(II, JJ, KK)
2151 DXX(1) = DZ(KK)
2152 DXX(2) = DX(II)
2153 MUA = 0.25*EB*(MU(II, JJ, KK) + MU(II+1, JJ, KK) + MU(II+1, JJ, KK+1) + MU(II, JJ, KK+1) )
2154 CASE(3) COMPONENT
2155 UUP(1) = UU(II, JJ+1, KK)
2156 UUM(1) = UU(II, JJ, KK)
2157 UUP(2) = VV(II+1, JJ, KK)
2158 UUM(2) = VV(II, JJ, KK)
2159 DXX(1) = DX(II)
2160 DXX(2) = DY(JJ)
2161 MUA = 0.25*EB*(MU(II, JJ, KK) + MU(II+1, JJ, KK) + MU(II+1, JJ+1, KK) + MU(II, JJ+1, KK) )
2162 END SELECT COMPONENT
2163
2164 ! Indicate that the velocity gradients in the two orthogonal directions have not been changed yet
2165
2166 ALTERED_GRADIENT = .FALSE.
2167
2168 ! Loop over all possible orientations of edge and reassign velocity gradients if appropriate
2169
2170 SIGN_LOOP: DO LSGN=-1,1,2
2171 ORIENTATION_LOOP: DO IS=1,3
2172
2173 IF (IS==IEC) CYCLE ORIENTATION_LOOP
2174
2175 ! IOR is the orientation of the wall cells adjacent to the edge
2176
2177 IOR = LSGN*IS
2178
2179 ! IS2 is the other coordinate direction besides IOR.
2180
2181 SELECT CASE(IEC)
2182 CASE(1)
2183 IF (IS==2) IS2 = 3
2184 IF (IS==3) IS2 = 2
2185 CASE(2)
2186 IF (IS==1) IS2 = 3
2187 IF (IS==3) IS2 = 1
2188 CASE(3)
2189 IF (IS==1) IS2 = 2
2190 IF (IS==2) IS2 = 1
2191 END SELECT
2192
2193 ! Determine Index_Coordinate_Direction
2194 ! IEC=1, ICD=1 refers to DWDY; ICD=2 refers to DVDZ
2195 ! IEC=2, ICD=1 refers to DUDZ; ICD=2 refers to DWDX
2196 ! IEC=3, ICD=1 refers to DVDX; ICD=2 refers to DUDY
2197
2198 IF (IS>IEC) ICD = IS-IEC
2199 IF (IS<IEC) ICD = IS-IEC+3
2200 ICD_SGN = LSGN * ICD
2201
2202 ! IWM and IWP are the wall cell indices of the boundary on either side of the edge.
2203
2204 IF (IOR<0) THEN
2205 IWM = WALL_INDEX(ICMM, -IOR)
2206 IWM_I = WALL_INDEX(ICMM, IS2)
2207 IF (ICD==1) THEN
2208 IWP = WALL_INDEX(ICMP, -IOR)
2209 IWP_I = WALL_INDEX(ICMP, -IS2)
2210 ELSE ! ICD==2
2211 IWP = WALL_INDEX(ICPM, -IOR)
2212 IWP_I = WALL_INDEX(ICPM, -IS2)
2213 ENDF
2214 ELSE

```

```

2215 IF (ICD==1) THEN
2216 IWM = WALL_INDEX(ICPM,-IOR)
2217 IWM = WALL_INDEX(ICPM,IS2)
2218 ELSE ! ICD==2
2219 IWM = WALL_INDEX(ICMP,-IOR)
2220 IWM = WALL_INDEX(ICMP,IS2)
2221 ENDIF
2222 IWP = WALL_INDEX(ICPP,-IOR)
2223 IWP = WALL_INDEX(ICPP,-IS2)
2224 ENDIF
2225
2226 ! If both adjacent wall cells are undefined, cycle out of the loop.
2227
2228 IF (IWM==0 .AND. IWP==0) CYCLE ORIENTATION_LOOP
2229
2230 ! If there is a solid wall separating the two adjacent wall cells, cycle out of the loop.
2231
2232 IF (WALL(IWM)%BOUNDARY_TYPE==SOLID_BOUNDARY .OR. WALL(IWP)%BOUNDARY_TYPE==SOLID_BOUNDARY) CYCLE
    ORIENTATION_LOOP
2233
2234 ! If only one adjacent wall cell is defined, use its properties.
2235
2236 IF (IWM>0) THEN
2237 WCM => WALL(IWM)
2238 ELSE
2239 WCM => WALL(IWP)
2240 ENDIF
2241
2242 IF (IWP>0) THEN
2243 WCP => WALL(IWP)
2244 ELSE
2245 WCP => WALL(IWM)
2246 ENDIF
2247
2248 ! If both adjacent wall cells are NULL, cycle out.
2249
2250 BOUNDARY_TYPE_M = WCM%BOUNDARY_TYPE
2251 BOUNDARY_TYPE_P = WCP%BOUNDARY_TYPE
2252
2253 IF (BOUNDARY_TYPE_M==NULL_BOUNDARY .AND. BOUNDARY_TYPE_P==NULL_BOUNDARY) CYCLE ORIENTATION_LOOP
2254
2255 ! OPEN boundary conditions, both varieties, with and without a wind
2256
2257 OPEN_AND_WIND_BC: IF ((IWM==0.OR.WALL(IWM)%BOUNDARY_TYPE==OPEN_BOUNDARY) .AND. &
2258 (IWP==0.OR.WALL(IWP)%BOUNDARY_TYPE==OPEN_BOUNDARY)) THEN
2259
2260 VENT_INDEX = MAX(WCM%VENT_INDEX,WCP%VENT_INDEX)
2261 VT => VENTS(VENT_INDEX)
2262
2263 WIND_NO_WIND_IF: IF (.NOT.ANY(MEAN_FORCING)) THEN ! For regular OPEN boundary, (free-slip) BCs
2264
2265 SELECT CASE(IEC)
2266 CASE(1)
2267 IF (JJ==0 .AND. IOR== 2) WW(I,0,KK) = WW(I,1,KK)
2268 IF (JJ==JBAR .AND. IOR== -2) WW(I,JBP1,KK) = WW(I,JBAR,KK)
2269 IF (KK==0 .AND. IOR== 3) VV(I,JJ,0) = VV(I,JJ,1)
2270 IF (KK==KBAR .AND. IOR== -3) VV(I,JJ,KBP1) = VV(I,JJ,KBAR)
2271 CASE(2)
2272 IF (II==0 .AND. IOR== 1) WW(0,JJ,KK) = WW(1,JJ,KK)
2273 IF (II==IBAR .AND. IOR== -1) WW(IBP1,JJ,KK) = WW(IBAR,JJ,KK)
2274 IF (KK==0 .AND. IOR== 3) UU(I,JJ,0) = UU(I,JJ,1)
2275 IF (KK==KBAR .AND. IOR== -3) UU(I,JJ,KBP1) = UU(I,JJ,KBAR)
2276 CASE(3)
2277 IF (II==0 .AND. IOR== 1) VV(0,JJ,KK) = VV(1,JJ,KK)
2278 IF (II==IBAR .AND. IOR== -1) VV(IBP1,JJ,KK) = VV(IBAR,JJ,KK)
2279 IF (JJ==0 .AND. IOR== 2) UU(I,0,KK) = UU(I,1,KK)
2280 IF (JJ==JBAR .AND. IOR== -2) UU(I,JBP1,KK) = UU(I,JBAR,KK)
2281 END SELECT
2282
2283 ELSE WIND_NO_WIND_IF ! For wind, use prescribed far-field velocity all around
2284
2285 UBAR = U0*EVALUATE_RAMP(T,DUMMY,LRAMP,U0.T)*EVALUATE_RAMP(ZC(KK),DUMMY,LRAMP,U0.Z)
2286 VBAR = V0*EVALUATE_RAMP(T,DUMMY,LRAMP,V0.T)*EVALUATE_RAMP(ZC(KK),DUMMY,LRAMP,V0.Z)
2287 WBAR = W0*EVALUATE_RAMP(T,DUMMY,LRAMP,W0.T)*EVALUATE_RAMP(ZC(KK),DUMMY,LRAMP,W0.Z)
2288
2289 SELECT CASE(IEC)
2290 CASE(1)
2291 IF (JJ==0 .AND. IOR== 2) WW(I,0,KK) = WBAR
2292 IF (JJ==JBAR .AND. IOR== -2) WW(I,JBP1,KK) = WBAR
2293 IF (KK==0 .AND. IOR== 3) VV(I,JJ,0) = VBAR
2294 IF (KK==KBAR .AND. IOR== -3) VV(I,JJ,KBP1) = VBAR
2295 CASE(2)
2296 IF (II==0 .AND. IOR== 1) WW(0,JJ,KK) = WBAR
2297 IF (II==IBAR .AND. IOR== -1) WW(IBP1,JJ,KK) = WBAR
2298 IF (KK==0 .AND. IOR== 3) UU(I,JJ,0) = UBAR
2299 IF (KK==KBAR .AND. IOR== -3) UU(I,JJ,KBP1) = UBAR
2300 CASE(3)
2301 IF (II==0 .AND. IOR== 1) VV(0,JJ,KK) = VBAR

```

Source Code files for edited portions of FDS

```

2302 IF (I1==IBAR .AND. IOR==-1) VV(IBP1, JJ, KK) = VBAR
2303 IF (JJ==0 .AND. IOR== 2) UU(I1, 0, KK) = UBAR
2304 IF (JJ==JBAR .AND. IOR== -2) UU(I1, JBP1, KK) = UBAR
2305 END SELECT
2306
2307 ENDDIF WIND.NO.WIND.IF
2308
2309 IF (IWM/=0 .AND. IWP/=0) THEN
2310 CYCLE EDGELOOP ! Do no further processing of this edge if both cell faces are OPEN
2311 ELSE
2312 CYCLE ORIENTATION_LOOP
2313 ENDDIF
2314
2315 ENDDIF OPEN.AND.WIND.BC
2316
2317 ! Define the appropriate gas and ghost velocity
2318
2319 IF (ICD==1) THEN ! Used to pick the appropriate velocity component
2320 IVL=2
2321 ELSE !ICD==2
2322 IVL=1
2323 ENDDIF
2324
2325 IF (IOR<0) THEN
2326 VEL_GAS = UUM(IVL)
2327 VEL_GHOST = UUP(IVL)
2328 IIGM = I_CELL(ICMM)
2329 JJGM = J_CELL(ICMM)
2330 KKGM = K_CELL(ICMM)
2331 IF (ICD==1) THEN
2332 IIGP = I_CELL(ICMP)
2333 JJGP = J_CELL(ICMP)
2334 KKGP = K_CELL(ICMP)
2335 ELSE ! ICD==2
2336 IIGP = I_CELL(ICPM)
2337 JJGP = J_CELL(ICPM)
2338 KKGP = K_CELL(ICPM)
2339 ENDDIF
2340 ELSE
2341 VEL_GAS = UUP(IVL)
2342 VEL_GHOST = UUM(IVL)
2343 IF (ICD==1) THEN
2344 IIGM = I_CELL(ICPM)
2345 JJGM = J_CELL(ICPM)
2346 KKGM = K_CELL(ICPM)
2347 ELSE ! ICD==2
2348 IIGM = I_CELL(ICMP)
2349 JJGM = J_CELL(ICMP)
2350 KKGM = K_CELL(ICMP)
2351 ENDDIF
2352 IIGP = I_CELL(ICPP)
2353 JJGP = J_CELL(ICPP)
2354 KKGP = K_CELL(ICPP)
2355 ENDDIF
2356
2357 ! Decide whether or not to process edge using data interpolated from another mesh
2358
2359 INTERPOLATION_IF: IF (NCM(ICD)==0 .OR. &
2360 (BOUNDARY.TYPE.M==SOLID.BOUNDARY .OR. BOUNDARY.TYPE.P==SOLID.BOUNDARY) .OR. &
2361 (BOUNDARY.TYPE.M/=INTERPOLATED.BOUNDARY .AND. BOUNDARY.TYPE.P/=INTERPOLATED.BOUNDARY)) THEN
2362
2363 ! Determine appropriate velocity BC by assessing each adjacent wall cell. If the BCs are different on each
2364 ! side of the edge, choose the one with the specified velocity or velocity gradient, if there is one.
2365 ! If not, choose the max value of boundary condition index, simply for consistency.
2366
2367 SURF.INDEXM = WCM%SURF.INDEX
2368 SURF.INDEXP = WCP%SURF.INDEX
2369 IF (SURFACE(SURF.INDEXM)%SPECIFIED.NORMAL.VELOCITY .OR. SURFACE(SURF.INDEXM)%SPECIFIED.NORMAL.GRAIDENT) THEN
2370 SF=>SURFACE(SURF.INDEXM)
2371 ELSEIF (SURFACE(SURF.INDEXP)%SPECIFIED.NORMAL.VELOCITY .OR. SURFACE(SURF.INDEXP)%SPECIFIED.NORMAL.GRAIDENT) THEN
2372 SF=>SURFACE(SURF.INDEXP)
2373 ELSE
2374 SF=>SURFACE(MAX(SURF.INDEXM, SURF.INDEXP))
2375 ENDDIF
2376 VELOCITY_BC_INDEX = SF%VELOCITY_BC_INDEX
2377 IF (WCM%VENT_INDEX==WCP%VENT_INDEX .AND. WCP%VENT_INDEX > 0) THEN
2378 IF (VENTS(WCM%VENT_INDEX)%NODE_INDEX>0 .AND. WCM%ONE.D%UW >= 0. .EB) VELOCITY_BC_INDEX=FREE_SLIP_BC
2379 ENDDIF
2380
2381 ! Compute the viscosity in the two adjacent gas cells
2382
2383 MUA = 0.5_EB*(MU(IIGM, JJGM, KKGM) + MU(IIGP, JJGP, KKGP))
2384
2385 ! Set up synthetic eddy method (experimental)
2386
2387 SYNTHETIC.EDDY.METHOD = .FALSE.
2388 HVACTANGENTIAL = .FALSE.
2389 IF (IWM>0 .AND. IWP>0) THEN

```



Source Code files for edited portions of FDS

```

2390 IF (WCM%VENT_INDEX==WCP%VENT_INDEX) THEN
2391 IF (WCM%VENT_INDEX>0) THEN
2392 VT=>VENTS(WCM%VENT_INDEX)
2393 IF (VT%LEDDY>0) SYNTHETIC_EDDY_METHOD=.TRUE.
2394 IF (ALL(VT%UW> -1.E12.EB) .AND. VT%NODE_INDEX > 0) HVAC.TANGENTIAL = .TRUE.
2395 ENDIF
2396 ENDIF
2397 ENDIF
2398
2399 ! Determine if there is a tangential velocity component
2400
2401 VEL_T_IF: IF (.NOT.SP%SPECIFIED.TANGENTIAL.VELOCITY .AND. .NOT.SYNTHETIC_EDDY_METHOD .AND. .NOT. HVAC.TANGENTIAL)
      THEN
2402 VEL_T = 0. _EB
2403 ELSE VEL_T_IF
2404 VELEDDY = 0. _EB
2405 SYNTHETIC_EDDY_IF: IF (SYNTHETIC_EDDY_METHOD) THEN
2406 IS_SELECT: SELECT CASE(15) ! unsigned vent orientation
2407 CASE(1) ! yz plane
2408 SELECT CASE(IEC) ! edge orientation
2409 CASE(2)
2410 IF (ICD==1) VELEDDY = 0.5 _EB*(VT%U_EDDY(JJ ,KK)+VT%U_EDDY(JJ ,KK+1))
2411 IF (ICD==2) VELEDDY = 0.5 _EB*(VT%W_EDDY(JJ ,KK)+VT%W_EDDY(JJ ,KK+1))
2412 CASE(3)
2413 IF (ICD==1) VELEDDY = 0.5 _EB*(VT%W_EDDY(JJ ,KK)+VT%W_EDDY(JJ +1 ,KK))
2414 IF (ICD==2) VELEDDY = 0.5 _EB*(VT%U_EDDY(JJ ,KK)+VT%U_EDDY(JJ +1 ,KK))
2415 END SELECT
2416 CASE(2) ! zx plane
2417 SELECT CASE(IEC)
2418 CASE(3)
2419 IF (ICD==1) VELEDDY = 0.5 _EB*(VT%W_EDDY(II ,KK)+VT%W_EDDY(II +1 ,KK))
2420 IF (ICD==2) VELEDDY = 0.5 _EB*(VT%U_EDDY(II ,KK)+VT%U_EDDY(II +1 ,KK))
2421 CASE(1)
2422 IF (ICD==1) VELEDDY = 0.5 _EB*(VT%W_EDDY(II ,KK)+VT%W_EDDY(II ,KK+1))
2423 IF (ICD==2) VELEDDY = 0.5 _EB*(VT%W_EDDY(II ,KK)+VT%W_EDDY(II ,KK+1))
2424 END SELECT
2425 CASE(3) ! xy plane
2426 SELECT CASE(IEC)
2427 CASE(1)
2428 IF (ICD==1) VELEDDY = 0.5 _EB*(VT%W_EDDY(II ,JJ)+VT%W_EDDY(II ,JJ +1))
2429 IF (ICD==2) VELEDDY = 0.5 _EB*(VT%W_EDDY(II ,JJ)+VT%W_EDDY(II ,JJ +1))
2430 CASE(2)
2431 IF (ICD==1) VELEDDY = 0.5 _EB*(VT%U_EDDY(II ,JJ)+VT%U_EDDY(II +1 ,JJ))
2432 IF (ICD==2) VELEDDY = 0.5 _EB*(VT%W_EDDY(II ,JJ)+VT%W_EDDY(II +1 ,JJ))
2433 END SELECT
2434 END SELECT IS_SELECT
2435 ENDIF SYNTHETIC_EDDY_IF
2436 IF (ABS(SP%T_IGN-T.BEGIN)<=SPACING(SP%T_IGN) .AND. SP%RAMP_INDEX(TIME.VELO)>=1) THEN
2437 TSI = T
2438 ELSE
2439 TSI=T-SP%T_IGN
2440 ENDIF
2441 PROFILE_FACTOR = 1. _EB
2442 IF (HVAC.TANGENTIAL .AND. 0.5 _EB*(WCM%ONE.D%UWS+WCP%ONE.D%UWS) > 0. _EB) HVAC.TANGENTIAL = .FALSE.
2443 IF (HVAC.TANGENTIAL) THEN
2444 VEL_T = 0. _EB
2445 IEC_SELECT: SELECT CASE(IEC) ! edge orientation
2446 CASE (1)
2447 IF (ICD==1) VEL_T = 0.5 _EB*ABS((WCM%ONE.D%UWS+WCP%ONE.D%UWS)/VT%UW(ABS(VT%UOR)))*VT%UW(3)
2448 IF (ICD==2) VEL_T = 0.5 _EB*ABS((WCM%ONE.D%UWS+WCP%ONE.D%UWS)/VT%UW(ABS(VT%UOR)))*VT%UW(2)
2449 CASE (2)
2450 IF (ICD==1) VEL_T = 0.5 _EB*ABS((WCM%ONE.D%UWS+WCP%ONE.D%UWS)/VT%UW(ABS(VT%UOR)))*VT%UW(1)
2451 IF (ICD==2) VEL_T = 0.5 _EB*ABS((WCM%ONE.D%UWS+WCP%ONE.D%UWS)/VT%UW(ABS(VT%UOR)))*VT%UW(3)
2452 CASE (3)
2453 IF (ICD==1) VEL_T = 0.5 _EB*ABS((WCM%ONE.D%UWS+WCP%ONE.D%UWS)/VT%UW(ABS(VT%UOR)))*VT%UW(2)
2454 IF (ICD==2) VEL_T = 0.5 _EB*ABS((WCM%ONE.D%UWS+WCP%ONE.D%UWS)/VT%UW(ABS(VT%UOR)))*VT%UW(1)
2455 END SELECT IEC_SELECT
2456 ELSE
2457 IF (SP%PROFILE/=0 .AND. SP%VEL>TWO_EPSILON.EB) &
2458 PROFILE_FACTOR = ABS(0.5 _EB*(WCM%UW+WCP%UW)/SP%VEL)
2459 RAMP_T = EVALUATE_RAMP(TSI , SP%TAU(TIME.VELO) , SP%RAMP_INDEX(TIME.VELO))
2460 IF (IEC==1 .OR. (IEC==2 .AND. ICD==2)) VEL_T = RAMP_T*(PROFILE_FACTOR*(SP%VEL_T(2) + VELEDDY))
2461 IF (IEC==3 .OR. (IEC==2 .AND. ICD==1)) VEL_T = RAMP_T*(PROFILE_FACTOR*(SP%VEL_T(1) + VELEDDY))
2462 ENDIF
2463 ENDIF VEL_T_IF
2464
2465 ! Choose the appropriate boundary condition to apply
2466
2467 HVAC_IF: IF (HVAC.TANGENTIAL) THEN
2468
2469 VEL_GHOST = 2. _EB*VEL_T - VEL_GAS
2470 DUIDXJ(ICD.SGN) = I.SGN*(VEL_GAS-VEL_GHOST)/DXX(ICD)
2471 MU_DUIDXJ(ICD.SGN) = MUA*DUIDXJ(ICD.SGN)
2472 ALTERED_GRADIENT(ICD.SGN) = .TRUE.
2473
2474 ELSE HVAC_IF
2475
2476 BOUNDARY_CONDITION: SELECT CASE(VELOCITY_BC_INDEX)

```

```

2477
2478 CASE (FREE.SLIP_BC) BOUNDARY.CONDITION
2479
2480 VEL_GHOST = VEL_GAS
2481 DUIDXJ(ICD.SGN) = I.SGN*(VEL_GAS-VEL_GHOST)/DXX(ICD)
2482 MU_DUIDXJ(ICD.SGN) = MUA*DUIDXJ(ICD.SGN)
2483 ALTERED.GRAIENT(ICD.SGN) = .TRUE.
2484
2485 CASE (NO.SLIP_BC) BOUNDARY.CONDITION
2486
2487 WANNIER_BC: IF (PERIODIC.TEST==5) THEN
2488 SELECT CASE(IOR)
2489 CASE( 1)
2490 VEL.T = WANNIER_FLOW(X(II),Z(KK),2)
2491 CASE(-1)
2492 VEL.T = WANNIER_FLOW(X(II),Z(KK),2)
2493 CASE( 3)
2494 VEL.T = WANNIER_FLOW(X(II),Z(KK),1)
2495 CASE(-3)
2496 VEL.T = WANNIER_FLOW(X(II),Z(KK),1)
2497 END SELECT
2498 ENDIF WANNIER_BC
2499
2500 VEL_GHOST = 2._EB*VEL.T - VEL_GAS
2501 DUIDXJ(ICD.SGN) = I.SGN*(VEL_GAS-VEL_GHOST)/DXX(ICD)
2502 MU_DUIDXJ(ICD.SGN) = MUA*DUIDXJ(ICD.SGN)
2503 ALTERED.GRAIENT(ICD.SGN) = .TRUE.
2504
2505 CASE (WALL.MODEL_BC) BOUNDARY.CONDITION
2506
2507 ITMP = MIN(5000,NINT(0.5._EB*(TMP(IIGM,JJGM,KKGM)+TMP(IIGP,JJGP,KKGP))))
2508 MU_WALL = MU_RSQMWZ(ITMP,1)/RSQ.MW.Z(1)
2509 RHO_WALL = 0.5._EB*(RHO_P(IIGM,JJGM,KKGM) + RHO_P(IIGP,JJGP,KKGP))
2510 CALL WALL_MODEL(SLIP_COEF,U.TAU,Y.PLUS,VEL_GAS-VEL.T,MU_WALL/RHO_WALL,DXX(ICD),SP%ROUGHNESS)
2511 SELECT CASE(SLIP.CONDITION)
2512 CASE(0)
2513 SLIP_COEF = -1._EB
2514 CASE(1)
2515 SLIP_COEF = SLIP_COEF
2516 CASE(2)
2517 SLIP_COEF = 0.5._EB*(SLIP_COEF-1._EB)
2518 CASE(3)
2519 WT1 = MAX(0._EB,MIN(1._EB,(Y.PLUS-Y.WERNER.WENGLE)/(Y.PLUS+TWO.EPSILON.EB)))
2520 WT2 = 1._EB-WT1
2521 SLIP_COEF = WT1*SLIP_COEF-WT2
2522 CASE(4)
2523 IF (ABS(0.5._EB*(WCM%ONED%UWS+WCP%ONED%UWS))>ABS(VEL_GAS-VEL.T)) THEN
2524 SLIP_COEF = -1._EB
2525 ELSE
2526 SLIP_COEF = 0.5._EB*(SLIP_COEF-1._EB)
2527 ENDIF
2528 END SELECT
2529 VEL_GHOST = VEL.T + SLIP_COEF*(VEL_GAS-VEL.T)
2530 DUIDXJ(ICD.SGN) = I.SGN*(VEL_GAS-VEL_GHOST)/DXX(ICD)
2531 MU_DUIDXJ(ICD.SGN) = RHO_WALL*U.TAU**2 * SIGN(1._EB,I.SGN*(VEL_GAS-VEL.T))
2532 ALTERED.GRAIENT(ICD.SGN) = .TRUE.
2533
2534 END SELECT BOUNDARY.CONDITION
2535
2536 ENDIF HVAC_IF
2537
2538 ELSE INTERPOLATION_IF ! Use data from another mesh
2539
2540 INTERPOLATED.EDGE = .TRUE.
2541 OM => O_MESH(ABS(NCM(ICD)))
2542
2543 IF (PREDICTOR) THEN
2544 SELECT CASE(IEC)
2545 CASE(1)
2546 IF (ICD==1) THEN
2547 VEL_OTHER => OM%WS
2548 ELSE ! ICD=2
2549 VEL_OTHER => OM%VS
2550 ENDIF
2551 CASE(2)
2552 IF (ICD==1) THEN
2553 VEL_OTHER => OM%US
2554 ELSE ! ICD=2
2555 VEL_OTHER => OM%WS
2556 ENDIF
2557 CASE(3)
2558 IF (ICD==1) THEN
2559 VEL_OTHER => OM%VS
2560 ELSE ! ICD=2
2561 VEL_OTHER => OM%US
2562 ENDIF
2563 END SELECT
2564 ELSE

```

```

2565 SELECT CASE(IEC)
2566 CASE(1)
2567 IF (ICD==1) THEN
2568 VEL_OTHER => CM%W
2569 ELSE ! ICD=2
2570 VEL_OTHER => CM%V
2571 ENDIF
2572 CASE(2)
2573 IF (ICD==1) THEN
2574 VEL_OTHER => CM%U
2575 ELSE ! ICD=2
2576 VEL_OTHER => CM%W
2577 ENDIF
2578 CASE(3)
2579 IF (ICD==1) THEN
2580 VEL_OTHER => CM%V
2581 ELSE ! ICD=2
2582 VEL_OTHER => CM%U
2583 ENDIF
2584 END SELECT
2585 ENDIF
2586
2587 WGT = EDGE.INTERPOLATION.FACTOR(IE,ICD)
2588 CMV = 1. -EB-WGT
2589
2590 SELECT CASE(IEC)
2591 CASE(1)
2592 IF (ICD==1) THEN
2593 VEL_GHOST = WGT*VEL_OTHER(IIO(ICD),JJO(ICD),KKO(ICD)) + CMV*VEL_OTHER(IIO(ICD),JJO(ICD),KKO(ICD)-1)
2594 ELSE ! ICD=2
2595 VEL_GHOST = WGT*VEL_OTHER(IIO(ICD),JJO(ICD),KKO(ICD)) + CMV*VEL_OTHER(IIO(ICD),JJO(ICD)-1,KKO(ICD))
2596 ENDIF
2597 CASE(2)
2598 IF (ICD==1) THEN
2599 VEL_GHOST = WGT*VEL_OTHER(IIO(ICD),JJO(ICD),KKO(ICD)) + CMV*VEL_OTHER(IIO(ICD)-1,JJO(ICD),KKO(ICD))
2600 ELSE ! ICD=2
2601 VEL_GHOST = WGT*VEL_OTHER(IIO(ICD),JJO(ICD),KKO(ICD)) + CMV*VEL_OTHER(IIO(ICD),JJO(ICD),KKO(ICD)-1)
2602 ENDIF
2603 CASE(3)
2604 IF (ICD==1) THEN
2605 VEL_GHOST = WGT*VEL_OTHER(IIO(ICD),JJO(ICD),KKO(ICD)) + CMV*VEL_OTHER(IIO(ICD),JJO(ICD)-1,KKO(ICD))
2606 ELSE ! ICD=2
2607 VEL_GHOST = WGT*VEL_OTHER(IIO(ICD),JJO(ICD),KKO(ICD)) + CMV*VEL_OTHER(IIO(ICD)-1,JJO(ICD),KKO(ICD))
2608 ENDIF
2609 END SELECT
2610
2611 ENDIF INTERPOLATION_IF
2612
2613 ! Set ghost cell values at edge of computational domain
2614
2615 SELECT CASE(IEC)
2616 CASE(1)
2617 IF (JJ==0 .AND. IOR== 2) WW(I1,JJ,KK) = VEL_GHOST
2618 IF (JJ==JBAR .AND. IOR== -2) WW(I1,JJ+1,KK) = VEL_GHOST
2619 IF (KK==0 .AND. IOR== 3) VV(I1,JJ,KK) = VEL_GHOST
2620 IF (KK==KBAR .AND. IOR== -3) VV(I1,JJ,KK+1) = VEL_GHOST
2621 IF (CORRECTOR .AND. JJ>0 .AND. JJ<JBAR .AND. KK>0 .AND. KK<KBAR) THEN
2622 IF (ICD==1) THEN
2623 W.Y(I1,JJ,KK) = 0.5*EB*(VEL_GHOST+VEL_GAS)
2624 ELSE ! ICD=2
2625 V.Z(I1,JJ,KK) = 0.5*EB*(VEL_GHOST+VEL_GAS)
2626 ENDIF
2627 ENDIF
2628 CASE(2)
2629 IF (I1==0 .AND. IOR== 1) WW(I1,JJ,KK) = VEL_GHOST
2630 IF (I1==IBAR .AND. IOR== -1) WW(I1+1,JJ,KK) = VEL_GHOST
2631 IF (KK==0 .AND. IOR== 3) UU(I1,JJ,KK) = VEL_GHOST
2632 IF (KK==KBAR .AND. IOR== -3) UU(I1,JJ,KK+1) = VEL_GHOST
2633 IF (CORRECTOR .AND. I1>0 .AND. I1<IBAR .AND. KK>0 .AND. KK<KBAR) THEN
2634 IF (ICD==1) THEN
2635 U.Z(I1,JJ,KK) = 0.5*EB*(VEL_GHOST+VEL_GAS)
2636 ELSE ! ICD=2
2637 W.X(I1,JJ,KK) = 0.5*EB*(VEL_GHOST+VEL_GAS)
2638 ENDIF
2639 ENDIF
2640 CASE(3)
2641 IF (I1==0 .AND. IOR== 1) VV(I1,JJ,KK) = VEL_GHOST
2642 IF (I1==IBAR .AND. IOR== -1) VV(I1+1,JJ,KK) = VEL_GHOST
2643 IF (JJ==0 .AND. IOR== 2) UU(I1,JJ,KK) = VEL_GHOST
2644 IF (JJ==JBAR .AND. IOR== -2) UU(I1,JJ+1,KK) = VEL_GHOST
2645 IF (CORRECTOR .AND. I1>0 .AND. I1<IBAR .AND. JJ>0 .AND. JJ<JBAR) THEN
2646 IF (ICD==1) THEN
2647 V.X(I1,JJ,KK) = 0.5*EB*(VEL_GHOST+VEL_GAS)
2648 ELSE ! ICD=2
2649 U.Y(I1,JJ,KK) = 0.5*EB*(VEL_GHOST+VEL_GAS)
2650 ENDIF
2651 ENDIF
2652 END SELECT

```

```

2653
2654 ENDDO ORIENTATION_LOOP
2655 ENDDO SIGN_LOOP
2656
2657 ! Cycle out of the EDGE_LOOP if no tangential gradients have been altered.
2658
2659 IF (.NOT.ANY(ALTERED_GRADIENT)) CYCLE EDGE_LOOP
2660
2661 ! If the edge is on an interpolated boundary, and all cells around it are not solid, cycle
2662
2663 IF (INTERPOLATED_EDGE) THEN
2664   PROCESS_EDGE = .FALSE.
2665   DO IS=5,8
2666     IF (SOLID(IJKE(IS,IE))) PROCESS_EDGE = .TRUE.
2667   ENDDO
2668   IF (.NOT.PROCESS_EDGE) CYCLE EDGE_LOOP
2669   ENDF
2670
2671 ! Loop over all 4 normal directions and compute vorticity and stress tensor components for each
2672
2673 SIGN_LOOP_2: DO I_SGN=-1,1,2
2674   ORIENTATION_LOOP_2: DO ICD=1,2
2675     IF (ICD==1) THEN
2676       ICDO=2
2677       ELSE ! ICD=2
2678       ICDO=1
2679     ENDF
2680     ICD_SGN = I_SGN*ICD
2681     IF (ALTERED_GRADIENT(ICD_SGN)) THEN
2682       DUIDXJ_USE(ICD) = DUIDXJ(ICD_SGN)
2683       MU_DUIDXJ_USE(ICD) = MU_DUIDXJ(ICD_SGN)
2684     ELSEIF (ALTERED_GRADIENT(-ICD_SGN)) THEN
2685       DUIDXJ_USE(ICD) = DUIDXJ(-ICD_SGN)
2686       MU_DUIDXJ_USE(ICD) = MU_DUIDXJ(-ICD_SGN)
2687     ELSE
2688       CYCLE ORIENTATION_LOOP_2
2689     ENDF
2690     ICDO_SGN = I_SGN*ICDO
2691     IF (ALTERED_GRADIENT(ICDO_SGN)) THEN
2692       DUIDXJ_USE(ICDO) = DUIDXJ(ICDO_SGN)
2693       MU_DUIDXJ_USE(ICDO) = MU_DUIDXJ(ICDO_SGN)
2694     ELSEIF (ALTERED_GRADIENT(-ICDO_SGN)) THEN
2695       DUIDXJ_USE(ICDO) = DUIDXJ(-ICDO_SGN)
2696       MU_DUIDXJ_USE(ICDO) = MU_DUIDXJ(-ICDO_SGN)
2697     ELSE
2698       DUIDXJ_USE(ICDO) = 0._EB
2699       MU_DUIDXJ_USE(ICDO) = 0._EB
2700     ENDF
2701     OME_E(ICD_SGN,IE) = DUIDXJ_USE(1) - DUIDXJ_USE(2)
2702     TAU_E(ICD_SGN,IE) = MU_DUIDXJ_USE(1) + MU_DUIDXJ_USE(2)
2703   ENDDO ORIENTATION_LOOP_2
2704   ENDDO SIGN_LOOP_2
2705
2706 ENDDO EDGE_LOOP
2707
2708 ! Store cell node averages of the velocity components in U,V,W,GHOST for use in Smokeview only
2709
2710 IF (CORRECTOR) THEN
2711   DO K=0,KBAR
2712     DO J=0,JBAR
2713       DO I=0,IBAR
2714         IC = CELL_INDEX(I,J,K)
2715         IF (IC==0) CYCLE
2716         IF (U_Y(I,J,K) > -1.E5._EB) U,V,W,GHOST(IC,1) = U_Y(I,J,K)
2717         IF (U_Z(I,J,K) > -1.E5._EB) U,V,W,GHOST(IC,1) = U_Z(I,J,K)
2718         IF (V_X(I,J,K) > -1.E5._EB) U,V,W,GHOST(IC,2) = V_X(I,J,K)
2719         IF (V_Z(I,J,K) > -1.E5._EB) U,V,W,GHOST(IC,2) = V_Z(I,J,K)
2720         IF (W_X(I,J,K) > -1.E5._EB) U,V,W,GHOST(IC,3) = W_X(I,J,K)
2721         IF (W_Y(I,J,K) > -1.E5._EB) U,V,W,GHOST(IC,3) = W_Y(I,J,K)
2722       ENDDO
2723     ENDDO
2724   ENDDO
2725   ENDF
2726
2727   T_USED(4) = T_USED(4) + SECOND() - TNOW
2728   END SUBROUTINE VELOCITY_BC
2729
2730
2731 SUBROUTINE MATCH_VELOCITY(NM)
2732
2733 ! Force normal component of velocity to match at interpolated boundaries
2734
2735 INTEGER :: NCM, II, JJ, KK, IOR, IW, IIO, JJO, KKO
2736 INTEGER, INTENT(IN) :: NM
2737 REAL(EB) :: UU_AVG, VV_AVG, WW_AVG, TNOW, DA_OTHER, UU_OTHER, VV_OTHER, WW_OTHER, NOM_CELLS
2738 REAL(EB), POINTER, DIMENSION(:,:,:) :: U=>NULL(), V=>NULL(), W=>NULL(), OMLU=>NULL(), OMLV=>NULL(), OMLW=>NULL()
2739 TYPE (OMESH_TYPE), POINTER :: OM=>NULL()
2740 TYPE (MESH_TYPE), POINTER :: M=>NULL()

```

Source Code files for edited portions of FDS

```

2741 TYPE (WALL_TYPE), POINTER :: WC=>NULL()
2742 TYPE (EXTERNAL_WALL_TYPE), POINTER :: EWC=>NULL()
2743 IF (SOLID.PHASE.ONLY) RETURN
2744 IF (EVACUATION.ONLY(NM)) RETURN
2745
2746 TNCW = SECOND()
2747
2748 ! Assign local variable names
2749
2750 CALL POINT.TO.MESH(NM)
2751
2752 ! Point to the appropriate velocity field
2753
2754 IF (PREDICTOR) THEN
2755   UU => US
2756   VV => VS
2757   WW => WS
2758   D.CORR = 0..EB
2759 ELSE
2760   UU => U
2761   VV => V
2762   WW => W
2763   DS.CORR = 0..EB
2764 ENDIF
2765
2766 ! Loop over all external wall cells and force adjacent normal components of velocity at interpolated boundaries to
      match.
2767
2768 EXTERNAL_WALL_LOOP: DO IW=1,N.EXTERNAL_WALL.CELLS
2769
2770   WC=>WALL(IW)
2771
2772   IF (WC%BOUNDARY_TYPE/=INTERPOLATED.BOUNDARY) CYCLE EXTERNAL_WALL_LOOP
2773
2774   EWC=>EXTERNAL_WALL(IW)
2775
2776   II = WC%ONE.D%II
2777   JJ = WC%ONE.D%JJ
2778   KK = WC%ONE.D%KK
2779   IOR = WC%ONE.D%IOR
2780   NCM = EWC%NCM
2781   OM => O.MESH(NCM)
2782   M2 => MESHES(NCM)
2783
2784   ! Determine the area of the interpolated cell face
2785
2786   DA.OTHER = 0..EB
2787
2788   SELECT CASE(ABS(IOR))
2789   CASE(1)
2790     IF (PREDICTOR) OMLUU => OM%US
2791     IF (CORRECTOR) OMLUU => OM%U
2792     DO KKO=EWC%KKO.MIN,EWC%KKO.MAX
2793     DO JJO=EWC%JJO.MIN,EWC%JJO.MAX
2794     DO IIO=EWC%IIO.MIN,EWC%IIO.MAX
2795     DA.OTHER = DA.OTHER + M2%DY(JJO)*M2%DZ(KKO)
2796     ENDDO
2797     ENDDO
2798     ENDDO
2799   CASE(2)
2800     IF (PREDICTOR) OMLVV => OM%VS
2801     IF (CORRECTOR) OMLVV => OM%V
2802     DO KKO=EWC%KKO.MIN,EWC%KKO.MAX
2803     DO JJO=EWC%JJO.MIN,EWC%JJO.MAX
2804     DO IIO=EWC%IIO.MIN,EWC%IIO.MAX
2805     DA.OTHER = DA.OTHER + M2%DX(IIO)*M2%DZ(KKO)
2806     ENDDO
2807     ENDDO
2808     ENDDO
2809   CASE(3)
2810     IF (PREDICTOR) OMBWW => OM%WS
2811     IF (CORRECTOR) OMBWW => OM%W
2812     DO KKO=EWC%KKO.MIN,EWC%KKO.MAX
2813     DO JJO=EWC%JJO.MIN,EWC%JJO.MAX
2814     DO IIO=EWC%IIO.MIN,EWC%IIO.MAX
2815     DA.OTHER = DA.OTHER + M2%DX(IIO)*M2%DY(JJO)
2816     ENDDO
2817     ENDDO
2818     ENDDO
2819   END SELECT
2820
2821   ! Determine the normal component of velocity from the other mesh and use it for average
2822
2823   SELECT CASE(IOR)
2824
2825   CASE( 1)
2826
2827     UU.OTHER = 0..EB

```

Source Code files for edited portions of FDS

```

2828 DO KKO=EWC%KKO_MIN,EWC%KKO_MAX
2829 DO JJO=EWC%JJO_MIN,EWC%JJO_MAX
2830 DO IIO=EWC%IIO_MIN,EWC%IIO_MAX
2831 UU_OTHER = UU_OTHER + OMLUU(IIO,JJO,KKO)*M2%DX(JJO)*M2%DZ(KKO)/DA_OTHER
2832 IF (EWC%AREA_RATIO>0.9_EB) OMLUU(IIO,JJO,KKO) = 0.5_EB*(OMLUU(IIO,JJO,KKO)+UU(0,JJ,KK))
2833 ENDDO
2834 ENDDO
2835 ENDDO
2836 UU_AVG = 0.5_EB*(UU(0,JJ,KK) + UU_OTHER)
2837 IF (PREDICTOR) D_CORR(IW) = DS_CORR(IW) + 0.5*(UU_AVG-UU(0,JJ,KK))*R(0)*RDX(1)*RRN(1)
2838 IF (CORRECTOR) DS_CORR(IW) = (UU_AVG-UU(0,JJ,KK))*R(0)*RDX(1)*RRN(1)
2839 UVW_SAVE(IW) = UU(0,JJ,KK)
2840 UU(0,JJ,KK) = UU_AVG
2841
2842 CASE(-1)
2843
2844 UU_OTHER = 0._EB
2845 DO KKO=EWC%KKO_MIN,EWC%KKO_MAX
2846 DO JJO=EWC%JJO_MIN,EWC%JJO_MAX
2847 DO IIO=EWC%IIO_MIN,EWC%IIO_MAX
2848 UU_OTHER = UU_OTHER + OMLUU(IIO-1,JJO,KKO)*M2%DX(JJO)*M2%DZ(KKO)/DA_OTHER
2849 IF (EWC%AREA_RATIO>0.9_EB) OMLUU(IIO-1,JJO,KKO) = 0.5_EB*(OMLUU(IIO-1,JJO,KKO)+UU(IBAR,JJ,KK))
2850 ENDDO
2851 ENDDO
2852 ENDDO
2853 UU_AVG = 0.5_EB*(UU(IBAR,JJ,KK) + UU_OTHER)
2854 IF (PREDICTOR) D_CORR(IW) = DS_CORR(IW) - 0.5*(UU_AVG-UU(IBAR,JJ,KK))*R(IBAR)*RDX(IBAR)*RRN(IBAR)
2855 IF (CORRECTOR) DS_CORR(IW) = -(UU_AVG-UU(IBAR,JJ,KK))*R(IBAR)*RDX(IBAR)*RRN(IBAR)
2856 UVW_SAVE(IW) = UU(IBAR,JJ,KK)
2857 UU(IBAR,JJ,KK) = UU_AVG
2858
2859 CASE( 2)
2860
2861 VV_OTHER = 0._EB
2862 DO KKO=EWC%KKO_MIN,EWC%KKO_MAX
2863 DO JJO=EWC%JJO_MIN,EWC%JJO_MAX
2864 DO IIO=EWC%IIO_MIN,EWC%IIO_MAX
2865 VV_OTHER = VV_OTHER + OMLVV(IIO,JJO,KKO)*M2%DX(IIO)*M2%DZ(KKO)/DA_OTHER
2866 IF (EWC%AREA_RATIO>0.9_EB) OMLVV(IIO,JJO,KKO) = 0.5_EB*(OMLVV(IIO,JJO,KKO)+VV(II,0,KK))
2867 ENDDO
2868 ENDDO
2869 ENDDO
2870 VV_AVG = 0.5_EB*(VV(II,0,KK) + VV_OTHER)
2871 IF (PREDICTOR) D_CORR(IW) = DS_CORR(IW) + 0.5*(VV_AVG-VV(II,0,KK))*RDY(1)
2872 IF (CORRECTOR) DS_CORR(IW) = (VV_AVG-VV(II,0,KK))*RDY(1)
2873 UVW_SAVE(IW) = VV(II,0,KK)
2874 VV(II,0,KK) = VV_AVG
2875
2876 CASE(-2)
2877
2878 VV_OTHER = 0._EB
2879 DO KKO=EWC%KKO_MIN,EWC%KKO_MAX
2880 DO JJO=EWC%JJO_MIN,EWC%JJO_MAX
2881 DO IIO=EWC%IIO_MIN,EWC%IIO_MAX
2882 VV_OTHER = VV_OTHER + OMLVV(IIO,JJO-1,KKO)*M2%DX(IIO)*M2%DZ(KKO)/DA_OTHER
2883 IF (EWC%AREA_RATIO>0.9_EB) OMLVV(IIO,JJO-1,KKO) = 0.5_EB*(OMLVV(IIO,JJO-1,KKO)+VV(II,JBAR,KK))
2884 ENDDO
2885 ENDDO
2886 ENDDO
2887 VV_AVG = 0.5_EB*(VV(II,JBAR,KK) + VV_OTHER)
2888 IF (PREDICTOR) D_CORR(IW) = DS_CORR(IW) - 0.5*(VV_AVG-VV(II,JBAR,KK))*RDY(JBAR)
2889 IF (CORRECTOR) DS_CORR(IW) = -(VV_AVG-VV(II,JBAR,KK))*RDY(JBAR)
2890 UVW_SAVE(IW) = VV(II,JBAR,KK)
2891 VV(II,JBAR,KK) = VV_AVG
2892
2893 CASE( 3)
2894
2895 WW_OTHER = 0._EB
2896 DO KKO=EWC%KKO_MIN,EWC%KKO_MAX
2897 DO JJO=EWC%JJO_MIN,EWC%JJO_MAX
2898 DO IIO=EWC%IIO_MIN,EWC%IIO_MAX
2899 WW_OTHER = WW_OTHER + OMLWW(IIO,JJO,KKO)*M2%DX(IIO)*M2%DY(JJO)/DA_OTHER
2900 IF (EWC%AREA_RATIO>0.9_EB) OMLWW(IIO,JJO,KKO) = 0.5_EB*(OMLWW(IIO,JJO,KKO)+WW(II,JJ,0))
2901 ENDDO
2902 ENDDO
2903 ENDDO
2904 WW_AVG = 0.5_EB*(WW(II,JJ,0) + WW_OTHER)
2905 IF (PREDICTOR) D_CORR(IW) = DS_CORR(IW) + 0.5*(WW_AVG-WW(II,JJ,0))*RDZ(1)
2906 IF (CORRECTOR) DS_CORR(IW) = (WW_AVG-WW(II,JJ,0))*RDZ(1)
2907 UVW_SAVE(IW) = WW(II,JJ,0)
2908 WW(II,JJ,0) = WW_AVG
2909
2910 CASE(-3)
2911
2912 WW_OTHER = 0._EB
2913 DO KKO=EWC%KKO_MIN,EWC%KKO_MAX
2914 DO JJO=EWC%JJO_MIN,EWC%JJO_MAX
2915 DO IIO=EWC%IIO_MIN,EWC%IIO_MAX

```

Source Code files for edited portions of FDS

```

2916 | WW.OTHER = WW.OTHER + CM$WW(IIO, JJO, KKO-1)*M2$DX(IIO)*M2$DY(JJO)/DA.OTHER
2917 | IF (EWC$AREA_RATIO>0.9_EB) CM$WW(IIO, JJO, KKO-1) = 0.5_EB*(CM$WW(IIO, JJO, KKO-1)+WW(I, JJ, KBAR))
2918 | ENDDO
2919 | ENDDO
2920 | ENDDO
2921 | WW.AVG = 0.5_EB*(WW(I, JJ, KBAR) + WW.OTHER)
2922 | IF (PREDICTOR) D.CORR(IW) = DS.CORR(IW) - 0.5*(WW.AVG+WW(I, JJ, KBAR))*RDZ(KBAR)
2923 | IF (CORRECTOR) DS.CORR(IW) = -(WW.AVG+WW(I, JJ, KBAR))*RDZ(KBAR)
2924 | UVW.SAVE(IW) = WW(I, JJ, KBAR)
2925 | WW(I, JJ, KBAR) = WW.AVG
2926 |
2927 | END SELECT
2928 |
2929 | ! Save velocity components at the ghost cell midpoint
2930 |
2931 | U.GHOST(IW) = 0._EB
2932 | V.GHOST(IW) = 0._EB
2933 | W.GHOST(IW) = 0._EB
2934 |
2935 | IF (PREDICTOR) OMLU => CM$US
2936 | IF (CORRECTOR) OMLU => CM$U
2937 | IF (PREDICTOR) OMLV => CM$VS
2938 | IF (CORRECTOR) OMLV => CM$V
2939 | IF (PREDICTOR) CM$W => CM$WS
2940 | IF (CORRECTOR) CM$W => CM$W
2941 |
2942 | DO KKO=EWC$KKO_MIN, EWC$KKO_MAX
2943 | DO JJO=EWC$JJO_MIN, EWC$JJO_MAX
2944 | DO IIO=EWC$IIO_MIN, EWC$IIO_MAX
2945 | U.GHOST(IW) = U.GHOST(IW) + 0.5_EB*(OMLU(IIO, JJO, KKO)+OMLU(IIO-1, JJO, KKO))
2946 | V.GHOST(IW) = V.GHOST(IW) + 0.5_EB*(OMLV(IIO, JJO, KKO)+OMLV(IIO, JJO-1, KKO))
2947 | W.GHOST(IW) = W.GHOST(IW) + 0.5_EB*(CM$W(IIO, JJO, KKO)+CM$W(IIO, JJO, KKO-1))
2948 | ENDDO
2949 | ENDDO
2950 | ENDDO
2951 | NOM.CELLS = REAL((EWC$IIO_MAX-EWC$IIO_MIN+1)*(EWC$JJO_MAX-EWC$JJO_MIN+1)*(EWC$KKO_MAX-EWC$KKO_MIN+1), EB)
2952 | U.GHOST(IW) = U.GHOST(IW)/NOM.CELLS
2953 | V.GHOST(IW) = V.GHOST(IW)/NOM.CELLS
2954 | W.GHOST(IW) = W.GHOST(IW)/NOM.CELLS
2955 |
2956 | ENDDO EXTERNAL_WALL_LOOP
2957 |
2958 | T.USED(4)=T.USED(4)+SECOND()-TNOW
2959 | END SUBROUTINE MATCH_VELOCITY
2960 |
2961 |
2962 | SUBROUTINE MATCH_VELOCITY_FLUX(NM)
2963 |
2964 | ! Force normal component of velocity flux to match at interpolated boundaries
2965 |
2966 | INTEGER :: NOM, I, JJ, KK, IOR, IW, IIO, JJO, KKO
2967 | INTEGER, INTENT(IN) :: NM
2968 | REAL(EB) :: TNOW, DA.OTHER, FVX.OTHER, FVY.OTHER, FVZ.OTHER
2969 | TYPE (OMESH_TYPE), POINTER :: CM
2970 | TYPE (MESH_TYPE), POINTER :: M2
2971 | TYPE (WALL_TYPE), POINTER :: WC
2972 | TYPE (EXTERNAL_WALL_TYPE), POINTER :: EWC
2973 |
2974 | IF (NMESHES==1) RETURN
2975 | IF (SOLID_PHASE_ONLY) RETURN
2976 | IF (EVACUATION_ONLY(NM)) RETURN
2977 |
2978 | TNOW = SECOND()
2979 |
2980 | ! Assign local variable names
2981 |
2982 | CALL POINT_TO_MESH(NM)
2983 |
2984 | ! Loop over all cell edges and determine the appropriate velocity BCs
2985 |
2986 | EXTERNAL_WALL_LOOP: DO IW=1, N.EXTERNAL_WALL_CELLS
2987 |
2988 | WC=>WALL(IW)
2989 | EWC=>EXTERNAL_WALL(IW)
2990 | IF (WC$BOUNDARY_TYPE/=INTERPOLATED_BOUNDARY) CYCLE EXTERNAL_WALL_LOOP
2991 |
2992 | I = WC$ONE_D$I
2993 | JJ = WC$ONE_D$J
2994 | KK = WC$ONE_D$K
2995 | IOR = WC$ONE_D$IOR
2996 | NOM = EWC$NOM
2997 | CM => OMESH(NOM)
2998 | M2 => MESHES(NOM)
2999 |
3000 | ! Determine the area of the interpolated cell face
3001 |
3002 | DA.OTHER = 0._EB
3003 |

```

```

3004 SELECT CASE(ABS(IOR))
3005 CASE(1)
3006 DO KKO=EWC%KKO.MIN,EWC%KKO.MAX
3007 DO JJO=EWC%JJO.MIN,EWC%JJO.MAX
3008 DO IIO=EWC%IIO.MIN,EWC%IIO.MAX
3009 DA.OTHER = DA.OTHER + M2%DY(JJO)*M2%DZ(KKO)
3010 ENDDO
3011 ENDDO
3012 ENDDO
3013 CASE(2)
3014 DO KKO=EWC%KKO.MIN,EWC%KKO.MAX
3015 DO JJO=EWC%JJO.MIN,EWC%JJO.MAX
3016 DO IIO=EWC%IIO.MIN,EWC%IIO.MAX
3017 DA.OTHER = DA.OTHER + M2%DX(IIO)*M2%DZ(KKO)
3018 ENDDO
3019 ENDDO
3020 ENDDO
3021 CASE(3)
3022 DO KKO=EWC%KKO.MIN,EWC%KKO.MAX
3023 DO JJO=EWC%JJO.MIN,EWC%JJO.MAX
3024 DO IIO=EWC%IIO.MIN,EWC%IIO.MAX
3025 DA.OTHER = DA.OTHER + M2%DX(IIO)*M2%DY(JJO)
3026 ENDDO
3027 ENDDO
3028 ENDDO
3029 END SELECT
3030
3031 ! Determine the normal component of velocity from the other mesh and use it for average
3032
3033 SELECT CASE(IOR)
3034
3035 CASE( 1)
3036
3037 FVX.OTHER = 0._EB
3038 DO KKO=EWC%KKO.MIN,EWC%KKO.MAX
3039 DO JJO=EWC%JJO.MIN,EWC%JJO.MAX
3040 DO IIO=EWC%IIO.MIN,EWC%IIO.MAX
3041 FVX.OTHER = FVX.OTHER + OM%FVX(IIO,JJO,KKO)*M2%DY(JJO)*M2%DZ(KKO)/DA.OTHER
3042 ENDDO
3043 ENDDO
3044 ENDDO
3045 FVX(0,JJ,KK) = 0.5._EB*(FVX(0,JJ,KK) + FVX.OTHER)
3046
3047 CASE(-1)
3048
3049 FVX.OTHER = 0._EB
3050 DO KKO=EWC%KKO.MIN,EWC%KKO.MAX
3051 DO JJO=EWC%JJO.MIN,EWC%JJO.MAX
3052 DO IIO=EWC%IIO.MIN,EWC%IIO.MAX
3053 FVX.OTHER = FVX.OTHER + OM%FVX(IIO-1,JJO,KKO)*M2%DY(JJO)*M2%DZ(KKO)/DA.OTHER
3054 ENDDO
3055 ENDDO
3056 ENDDO
3057 FVX(IBAR,JJ,KK) = 0.5._EB*(FVX(IBAR,JJ,KK) + FVX.OTHER)
3058
3059 CASE( 2)
3060
3061 FVY.OTHER = 0._EB
3062 DO KKO=EWC%KKO.MIN,EWC%KKO.MAX
3063 DO JJO=EWC%JJO.MIN,EWC%JJO.MAX
3064 DO IIO=EWC%IIO.MIN,EWC%IIO.MAX
3065 FVY.OTHER = FVY.OTHER + OM%FVY(IIO,JJO,KKO)*M2%DX(IIO)*M2%DZ(KKO)/DA.OTHER
3066 ENDDO
3067 ENDDO
3068 ENDDO
3069 FVY(I1,0,KK) = 0.5._EB*(FVY(I1,0,KK) + FVY.OTHER)
3070
3071 CASE(-2)
3072
3073 FVY.OTHER = 0._EB
3074 DO KKO=EWC%KKO.MIN,EWC%KKO.MAX
3075 DO JJO=EWC%JJO.MIN,EWC%JJO.MAX
3076 DO IIO=EWC%IIO.MIN,EWC%IIO.MAX
3077 FVY.OTHER = FVY.OTHER + OM%FVY(IIO,JJO-1,KKO)*M2%DX(IIO)*M2%DZ(KKO)/DA.OTHER
3078 ENDDO
3079 ENDDO
3080 ENDDO
3081 FVY(I1,JBAR,KK) = 0.5._EB*(FVY(I1,JBAR,KK) + FVY.OTHER)
3082
3083 CASE( 3)
3084
3085 FVZ.OTHER = 0._EB
3086 DO KKO=EWC%KKO.MIN,EWC%KKO.MAX
3087 DO JJO=EWC%JJO.MIN,EWC%JJO.MAX
3088 DO IIO=EWC%IIO.MIN,EWC%IIO.MAX
3089 FVZ.OTHER = FVZ.OTHER + OM%FVZ(IIO,JJO,KKO)*M2%DX(IIO)*M2%DY(JJO)/DA.OTHER
3090 ENDDO
3091 ENDDO

```



```

3092 ENDDO
3093 FVZ(I1, JJ, 0) = 0.5_EB*(FVZ(I1, JJ, 0) + FVZ.OTHER)
3094
3095 CASE(-3)
3096
3097 FVZ.OTHER = 0._EB
3098 DO KKO=EW%KKO_MIN,EW%KKO_MAX
3099 DO JJO=EW%JJO_MIN,EW%JJO_MAX
3100 DO IIO=EW%IIO_MIN,EW%IIO_MAX
3101 FVZ.OTHER = FVZ.OTHER + OM%FVZ(IIO, JJO, KKO-1)*M2%DX(IIO)*M2%DY(JJO)/DA.OTHER
3102 ENDDO
3103 ENDDO
3104 ENDDO
3105 FVZ(I1, JJ, KBAR) = 0.5_EB*(FVZ(I1, JJ, KBAR) + FVZ.OTHER)
3106
3107 END SELECT
3108
3109 ENDDO EXTERNAL_WALL_LOOP
3110
3111 T_USED(4)=T_USED(4)+SECOND()-TNOW
3112 END SUBROUTINE MATCH_VELOCITY_FLUX
3113
3114
3115 SUBROUTINE CHECK_STABILITY(DT,DT_NEW,NM)
3116
3117 ! Checks the Courant and Von Neumann stability criteria, and if necessary, reduces the time step accordingly
3118
3119 USE PHYSICAL_FUNCTIONS, ONLY: GET_SPECIFIC_HEAT
3120 INTEGER, INTENT(IN) :: NM
3121 REAL(EB), INTENT(IN) :: DT
3122 REAL(EB) :: UODX, VODY, WCDZ, UWW, UWWMAX, R_DX2, MU_MAX, MUTRM, CP, ZZ_GET(1:N_TRACKED_SPECIES), PART_CFL, MU_TMP
3123 REAL(EB) :: DT_NEW(NMESHES)
3124 INTEGER :: I, J, K, IW, IIG, JJG, KKG
3125 TYPE(WALL_TYPE), POINTER :: WC=>NULL()
3126 REAL(EB), PARAMETER :: DT_EPS = 1.E-10_EB
3127
3128 IF (EVACUATION_ONLY(NM)) RETURN
3129
3130 UWWMAX = 0._EB
3131 VN = 0._EB
3132 MUTRM = 1.E-9_EB
3133 R_DX2 = 1.E-9_EB
3134
3135 ! Determine max CFL number from all grid cells
3136
3137 DO K=1, KBAR
3138 DO J=1, JBAR
3139 DO I=1, IBAR
3140 IF (SOLID(CELL_INDEX(I, J, K))) CYCLE
3141 UODX = MAXVAL(ABS(US(I-1:I, J, K))) * RDX(I)
3142 VODY = MAXVAL(ABS(VS(I, J-1:J, K))) * RDY(J)
3143 WCDZ = MAXVAL(ABS(WS(I, J, K-1:K))) * RDZ(K)
3144 SELECT CASE (CFL_VELOCITY_NORM)
3145 CASE(0) ; UWW = MAX(UODX, VODY, WCDZ) + ABS(DS(I, J, K))
3146 CASE(1) ; UWW = UODX + VODY + WCDZ + ABS(DS(I, J, K))
3147 CASE(2) ; UWW = SQRT(UODX**2 + VODY**2 + WCDZ**2) + ABS(DS(I, J, K))
3148 CASE(3) ; UWW = MAX(UODX, VODY, WCDZ)
3149 END SELECT
3150 IF (UWW >= UWWMAX) THEN
3151 UWWMAX = UWW
3152 ICFL = I
3153 JCFL = J
3154 KCFL = K
3155 ENDIF
3156 ENDDO
3157 ENDDO
3158 ENDDO
3159
3160 HEAT_TRANSFER_IF: IF (CHECK_HT) THEN
3161 WALL_LOOP: DO IW=1, N_EXTERNAL_WALL_CELLS + N_INTERNAL_WALL_CELLS
3162 WC=>WALL(IW)
3163 IF (WC%BOUNDARY_TYPE/=SOLID_BOUNDARY) CYCLE WALL_LOOP
3164 IIG = WC%ONE_D%IIG
3165 JJG = WC%ONE_D%JJG
3166 KKG = WC%ONE_D%KKG
3167 ZZ_GET(1:N_TRACKED_SPECIES) = ZZS(IIG, JJG, KKG, 1:N_TRACKED_SPECIES)
3168 CALL GET_SPECIFIC_HEAT(ZZ_GET, CP, TMP(IIG, JJG, KKG))
3169 UWW = WC%ONE_D%HEAT_TRANS_COEF / (WC%ONE_D%RHO_F * CP) * WC%ONE_D%RDN
3170 IF (UWW >= UWWMAX) THEN
3171 UWWMAX = UWW
3172 ICFL = IIG
3173 JCFL = JJG
3174 KCFL = KKG
3175 ENDIF
3176 ENDDO WALL_LOOP
3177 ENDIF HEAT_TRANSFER_IF
3178
3179 CFL = DT * UWWMAX

```

```

3180 PART_CFL = DT*PART_UVW_MAX
3181
3182 ! Determine max Von Neumann Number for fine grid calcs
3183
3184 PARABOLIC_IF: IF (CHECK_VN) THEN
3185
3186 MUMAX = 0._EB
3187 DO K=1,KBAR
3188 DO J=1,JBAR
3189 LLOOP: DO I=1,IBAR
3190 IF (SOLID(CELL_INDEX(I,J,K))) CYCLE LLOOP
3191 MULTMP = MAX(DZ_MAX(I,J,K),MU(I,J,K)/RHOS(I,J,K))
3192 IF (MULTMP >= MUMAX) THEN
3193 MUMAX = MULTMP
3194 I_VN=I
3195 J_VN=J
3196 K_VN=K
3197 ENDF
3198 ENDDO LLOOP
3199 ENDDO
3200 ENDDO
3201
3202 IF (TWO_D) THEN
3203 R_DX2 = RDX(I_VN)**2 + RDZ(K_VN)**2
3204 ELSE
3205 R_DX2 = RDX(I_VN)**2 + RDY(J_VN)**2 + RDZ(K_VN)**2
3206 ENDF
3207
3208 MUTRM = MUMAX
3209 VN = DT*2._EB*R_DX2*MUTRM
3210
3211 ENDF PARABOLIC_IF
3212
3213 ! Adjust time step size if necessary
3214
3215 IF ((CFL < CFL_MAX .AND. VN < VN_MAX .AND. PART_CFL < PARTICLE_CFL_MAX .AND. DRAG_CFL < DRAG_CFL_MAX) .OR.
    LOCK_TIME_STEP) THEN
3216 DT_NEW(NM) = DT
3217 IF (CFL <= CFL_MIN .AND. VN < VN_MIN .AND. PART_CFL < PARTICLE_CFL_MIN .AND. .NOT. LOCK_TIME_STEP) THEN
3218 SELECT CASE (RESTRICT_TIME_STEP)
3219 CASE (.TRUE.); DT_NEW(NM) = MIN(1.1._EB*DT,DT_INITIAL)
3220 CASE (.FALSE.); DT_NEW(NM) = 1.1._EB*DT
3221 END SELECT
3222 CHANGE_TIME_STEP_INDEX(NM) = 1
3223 ENDF
3224 ELSE
3225 DT_NEW(NM) = 0.9._EB*MIN( CFL_MAX/MAX(UVW_MAX,DT_EPS) , &
3226 VN_MAX/(2._EB*R_DX2*MAX(MUTRM,DT_EPS)) , &
3227 PARTICLE_CFL_MAX/MAX(PART_UVW_MAX,DT_EPS) ) , &
3228 DT*DRAG_CFL_MAX/MAX(DRAG_CFL,DT_EPS) )
3229 DRAG_CFL = 0._EB
3230 CHANGE_TIME_STEP_INDEX(NM) = -1
3231 ENDF
3232
3233 END SUBROUTINE CHECK_STABILITY
3234
3235
3236 SUBROUTINE BAROCLINIC_CORRECTION(T,NM)
3237
3238 ! Add baroclinic term to the momentum equation
3239
3240 USE MATH_FUNCTIONS, ONLY: EVALUATE_RAMP
3241 REAL(EB), INTENT(IN) :: T
3242 INTEGER, INTENT(IN) :: NM
3243 REAL(EB), POINTER, DIMENSION(:,:,:) :: UU=>NULL(),VV=>NULL(),WW=>NULL(),RHOP=>NULL(),HP=>NULL(),RHM=>NULL(),RRHO
=>NULL()
3244 INTEGER :: I,J,K,II,JJ,KK,IIG,JJG,KKG,IOR,IW
3245 REAL(EB) :: P_EXTERNAL,TST,TIME_RAMP_FACTOR,DUMMY,UN,TNW
3246 LOGICAL :: INFLOW
3247 TYPE(VENTS_TYPE), POINTER :: VT=>NULL()
3248 TYPE(WALL_TYPE), POINTER :: WC=>NULL()
3249
3250 TNW=SECOND()
3251 CALL POINT_TO_MESH(NM)
3252
3253 ! If the baroclinic torque term has been added to the momentum equation RHS, subtract it off.
3254
3255 IF (BAROCLINIC_TERMS_ATTACHED) THEN
3256 FVX = FVX - FVX_B
3257 FVY = FVY - FVY_B
3258 FVZ = FVZ - FVZ_B
3259 ENDF
3260
3261 BAROCLINIC_TERMS_ATTACHED = .TRUE.
3262
3263 RHM => WORK1 ! p=rho*(H-K)
3264 RRHO => WORK2 ! reciprocal of rho
3265

```

Source Code files for edited portions of FDS

```

3266 IF (PREDICTOR) THEN
3267   UU => U
3268   VV => V
3269   WW => W
3270   RHOP=>RHO
3271   IF (PRESSURE_ITERATIONS>1) THEN
3272     HP => H
3273   ELSE
3274     HP => HS
3275   ENDIF
3276   ! Note: this ordering of HP=HS in PREDICTOR is required to achieve 2nd order temporal convergence.
3277   ! We should rethink our notation and re-examine whether both H and HS are required.
3278   ELSE
3279     UU => US
3280     VV => VS
3281     WW => WS
3282     RHOP=>RHOS
3283     IF (PRESSURE_ITERATIONS>1) THEN
3284       HP => HS
3285     ELSE
3286       HP => H
3287     ENDIF
3288   ENDIF
3289
3290   ! Compute pressure and 1/rho in each grid cell
3291
3292   !$OMP PARALLEL PRIVATE(WC, VT, TSI, TIME_RAMP_FACTOR, P_EXTERNAL, &
3293   !$OMP& II, JJ, KK, IOR, IIG, JIG, KKG, UN, INFLOW)
3294   !$OMP DO SCHEDULE(static)
3295   DO K=0,KBP1
3296     DO J=0,JBP1
3297       DO I=0,IBP1
3298         RHMK(I,J,K) = RHOP(I,J,K)*(HP(I,J,K)-KRES(I,J,K))
3299         RRHO(I,J,K) = 1._EB/RHOP(I,J,K)
3300       ENDDO
3301     ENDDO
3302   ENDDO
3303   !$OMP END DO
3304
3305   ! Set baroclinic term to zero at outflow boundaries and P_EXTERNAL at inflow boundaries
3306
3307   !$OMP MASTER
3308   EXTERNAL_WALL_LOOP: DO IW=1,N_EXTERNAL_WALL_CELLS
3309     WC=>WALL(IW)
3310     IF (WC%BOUNDARY_TYPE/=OPEN_BOUNDARY) CYCLE EXTERNAL_WALL_LOOP
3311     IF (WC%VENT_INDEX>0) THEN
3312       VT => VENTS(WC%VENT_INDEX)
3313       IF (ABS(WC%ONED%T_IGN-T_BEGIN)<=SPACING(WC%ONED%T_IGN) .AND. VT%PRESSURE_RAMP_INDEX>=1) THEN
3314         TSI = T
3315       ELSE
3316         TSI = T - T_BEGIN
3317       ENDIF
3318       TIME_RAMP_FACTOR = EVALUATE_RAMP(TSI,DUMMY,VT%PRESSURE_RAMP_INDEX)
3319       P_EXTERNAL = TIME_RAMP_FACTOR*VT%DYNAMIC_PRESSURE
3320     ENDIF
3321     II = WC%ONED%II
3322     JJ = WC%ONED%JJ
3323     KK = WC%ONED%KK
3324     IOR = WC%ONED%IOR
3325     IIG = WC%ONED%IIG
3326     JIG = WC%ONED%JIG
3327     KKG = WC%ONED%KKG
3328     INFLOW = .FALSE.
3329     IOR_SELECT: SELECT CASE(IOR)
3330     CASE( 1); UN = UU(II, JJ, KK)
3331     CASE(-1); UN = UU(II -1, JJ, KK)
3332     CASE( 2); UN = VV(II, JJ, KK)
3333     CASE(-2); UN = VV(II, JJ -1, KK)
3334     CASE( 3); UN = WW(II, JJ, KK)
3335     CASE(-3); UN = WW(II, JJ, KK-1)
3336     END SELECT IOR_SELECT
3337     IF (UN*SIGN(1._EB,REAL(IOR,EB))>TWO_EPSILON_EB) INFLOW=.TRUE.
3338     IF (INFLOW) THEN
3339       RHMK(II, JJ, KK) = 2._EB*P_EXTERNAL - RHMK(IIG, JIG, KKG) ! Pressure at inflow boundary is P_EXTERNAL
3340     ELSE
3341       RHMK(II, JJ, KK) = -RHMK(IIG, JIG, KKG) ! No baroclinic correction for outflow boundary
3342     ENDIF
3343   ENDDO EXTERNAL_WALL_LOOP
3344   !$OMP END MASTER
3345   !$OMP BARRIER
3346
3347   ! Compute baroclinic term in the x momentum equation, p*d/dx(1/rho)
3348
3349   !$OMP DO SCHEDULE(static)
3350   DO K=1,KBAR
3351     DO J=1,JBAR
3352     DO I=0,IBAR
3353     FVX_B(I,J,K) = -(RHMK(I,J,K)*RHOP(I+1,J,K)+RHMK(I+1,J,K)*RHOP(I,J,K))*(RRHO(I+1,J,K)-RRHO(I,J,K))*RDXN(I)/ &

```

Source Code files for edited portions of FDS

```

3354 (RHOP(I+1,J,K)+RHOP(I,J,K))
3355 FVX(I,J,K) = FVX(I,J,K) + FVX.B(I,J,K)
3356 ENDDO
3357 ENDDO
3358 ENDDO
3359 !$OMP END DO nowait
3360
3361 ! Compute baroclinic term in the y momentum equation , p*d/dy(1/rho)
3362
3363 IF (.NOT.TWOD) THEN
3364 !$OMP DO SCHEDULE(static)
3365 DO K=1,KBAR
3366 DO J=0,JBAR
3367 DO I=1,IBAR
3368 FVY.B(I,J,K) = -(RHMK(I,J,K)*RHOP(I,J+1,K)+RHMK(I,J+1,K)*RHOP(I,J,K))*(RRHO(I,J+1,K)-RRHO(I,J,K))*RDYN(J)/ &
3369 (RHOP(I,J+1,K)+RHOP(I,J,K))
3370 FVY(I,J,K) = FVY(I,J,K) + FVY.B(I,J,K)
3371 ENDDO
3372 ENDDO
3373 ENDDO
3374 !$OMP END DO nowait
3375 ENDIF
3376
3377 ! Compute baroclinic term in the z momentum equation , p*d/dz(1/rho)
3378
3379 !$OMP DO SCHEDULE(static)
3380 DO K=0,KBAR
3381 DO J=1,JBAR
3382 DO I=1,IBAR
3383 FVZ.B(I,J,K) = -(RHMK(I,J,K)*RHOP(I,J,K+1)+RHMK(I,J,K+1)*RHOP(I,J,K))*(RRHO(I,J,K+1)-RRHO(I,J,K))*RDZN(K)/ &
3384 (RHOP(I,J,K+1)+RHOP(I,J,K))
3385 FVZ(I,J,K) = FVZ(I,J,K) + FVZ.B(I,J,K)
3386 ENDDO
3387 ENDDO
3388 ENDDO
3389 !$OMP END DO nowait
3390 !$OMP END PARALLEL
3391
3392 T_USED(4) = T_USED(4) + SECOND() - TNOW
3393 END SUBROUTINE BAROCLINIC.CORRECTION
3394
3395 ! ----- PATCH.VELOCITY.FLUX -----
3396
3397 SUBROUTINE PATCH.VELOCITY.FLUX(DT,NM)
3398
3399 ! The user may specify a polynomial profile using the PROP and DEVC lines. This routine
3400 ! specifies the source term in the momentum equation to drive the local velocity toward
3401 ! this user-specified value, in much the same way as the immersed boundary method
3402 ! (see IBM.VELOCITY.FLUX).
3403
3404 USE DEVICE_VARIABLES, ONLY: DEVICE_TYPE,PROPERTY_TYPE,NDEV,DEVICE,PROPERTY
3405 USE TRAN, ONLY: GINV
3406 REAL(EB), INTENT(IN) :: DT
3407 TYPE(DEVICE_TYPE), POINTER :: DV=>NULL()
3408 TYPE(PROPERTY_TYPE), POINTER :: PY=>NULL()
3409 INTEGER, INTENT(IN) :: NM
3410 INTEGER :: N,I,J,K,IC1,IC2,I1,I2,J1,J2,K1,K2
3411 REAL(EB), POINTER, DIMENSION(:, :,) :: UU=>NULL(),VV=>NULL(),WW=>NULL(),HH=>NULL()
3412 REAL(EB) :: VELP,DX0,DY0,DZ0
3413
3414 IF (PREDICTOR) THEN
3415 UU => U
3416 VV => V
3417 WW => W
3418 HH => H
3419 ELSE
3420 UU => US
3421 VV => VS
3422 WW => WS
3423 HH => HS
3424 ENDIF
3425
3426 DEVC_LOOP: DO N=1,NDEV
3427 DV=>DEVICE(N)
3428 IF (DV%QUANTITY/= 'VELOCITY PATCH') CYCLE DEVC_LOOP
3429 IF (DV%PROP_INDEX<1) CYCLE DEVC_LOOP
3430 IF (.NOT.DEVICE(DV%DEVC_INDEX(1))%CURRENT_STATE) CYCLE DEVC_LOOP
3431
3432 IF (DV%X1 > XF .OR. DV%X2 < XS .OR. &
3433 DV%Y1 > YF .OR. DV%Y2 < YS .OR. &
3434 DV%Z1 > ZF .OR. DV%Z2 < ZS) CYCLE DEVC_LOOP
3435
3436 PY=>PROPERTY(DV%PROP_INDEX)
3437
3438 I_VEL_SELECT: SELECT CASE(PY%I_VEL)

```

```

3442 CASE(1) I.LEVEL.SELECT
3443
3444 I1 = MAX(0, NINT( GINV(DV%X1-XS,1, NM)*RDXI )-1)
3445 I2 = MIN(IBAR, NINT( GINV(DV%X2-XS,1, NM)*RDXI )+1)
3446 J1 = MAX(0, NINT( GINV(DV%Y1-YS,2, NM)*RDETA )-1)
3447 J2 = MIN(JBAR, NINT( GINV(DV%Y2-YS,2, NM)*RDETA )+1)
3448 K1 = MAX(0, NINT( GINV(DV%Z1-ZS,3, NM)*RDZETA )-1)
3449 K2 = MIN(KBAR, NINT( GINV(DV%Z2-ZS,3, NM)*RDZETA )+1)
3450
3451 DO K=K1, K2
3452 DO J=J1, J2
3453 DO I=I1, I2
3454
3455 IC1 = CELLINDEX(I, J, K)
3456 IC2 = CELLINDEX(I+1, J, K)
3457 IF (SOLID(IC1) .OR. SOLID(IC2)) CYCLE
3458
3459 IF ( X(I)<DV%X1 .OR. X(I)>DV%X2) CYCLE ! Inefficient but simple
3460 IF ( YC(J)<DV%Y1 .OR. YC(J)>DV%Y2) CYCLE
3461 IF ( ZC(K)<DV%Z1 .OR. ZC(K)>DV%Z2) CYCLE
3462
3463 DX0 = X(I)-DV%X
3464 DY0 = YC(J)-DV%Y
3465 DZ0 = ZC(K)-DV%Z
3466 VELP = PY%P0 + DX0*PY%PX(1) + 0.5_EB*(DX0*DX0*PY%PXX(1,1)+DX0*DY0*PY%PXX(1,2)+DX0*DZ0*PY%PXX(1,3)) &
3467 + DY0*PY%PX(2) + 0.5_EB*(DY0*DX0*PY%PXX(2,1)+DY0*DY0*PY%PXX(2,2)+DY0*DZ0*PY%PXX(2,3)) &
3468 + DZ0*PY%PX(3) + 0.5_EB*(DZ0*DX0*PY%PXX(3,1)+DZ0*DY0*PY%PXX(3,2)+DZ0*DZ0*PY%PXX(3,3))
3469
3470 FVX(I, J, K) = -RDXN(I)*(HP(I+1, J, K)-HP(I, J, K)) - (VELP-UU(I, J, K))/DT
3471 ENDDO
3472 ENDDO
3473 ENDDO
3474
3475 CASE(2) I.LEVEL.SELECT
3476
3477 I1 = MAX(0, NINT( GINV(DV%X1-XS,1, NM)*RDXI )-1)
3478 I2 = MIN(IBAR, NINT( GINV(DV%X2-XS,1, NM)*RDXI )+1)
3479 J1 = MAX(0, NINT( GINV(DV%Y1-YS,2, NM)*RDETA )-1)
3480 J2 = MIN(JBAR, NINT( GINV(DV%Y2-YS,2, NM)*RDETA )+1)
3481 K1 = MAX(0, NINT( GINV(DV%Z1-ZS,3, NM)*RDZETA )-1)
3482 K2 = MIN(KBAR, NINT( GINV(DV%Z2-ZS,3, NM)*RDZETA )+1)
3483
3484 DO K=K1, K2
3485 DO J=J1, J2
3486 DO I=I1, I2
3487
3488 IC1 = CELLINDEX(I, J, K)
3489 IC2 = CELLINDEX(I, J+1, K)
3490
3491 IF (SOLID(IC1) .OR. SOLID(IC2)) CYCLE
3492
3493 IF (XC(I)<DV%X1 .OR. XC(I)>DV%X2) CYCLE
3494 IF ( Y(J)<DV%Y1 .OR. Y(J)>DV%Y2) CYCLE
3495 IF ( ZC(K)<DV%Z1 .OR. ZC(K)>DV%Z2) CYCLE
3496
3497 DX0 = XC(I)-DV%X
3498 DY0 = Y(J)-DV%Y
3499 DZ0 = ZC(K)-DV%Z
3500 VELP = PY%P0 + DX0*PY%PX(1) + 0.5_EB*(DX0*DX0*PY%PXX(1,1)+DX0*DY0*PY%PXX(1,2)+DX0*DZ0*PY%PXX(1,3)) &
3501 + DY0*PY%PX(2) + 0.5_EB*(DY0*DX0*PY%PXX(2,1)+DY0*DY0*PY%PXX(2,2)+DY0*DZ0*PY%PXX(2,3)) &
3502 + DZ0*PY%PX(3) + 0.5_EB*(DZ0*DX0*PY%PXX(3,1)+DZ0*DY0*PY%PXX(3,2)+DZ0*DZ0*PY%PXX(3,3))
3503
3504 FVY(I, J, K) = -RDYN(J)*(HP(I, J+1, K)-HP(I, J, K)) - (VELP-VV(I, J, K))/DT
3505 ENDDO
3506 ENDDO
3507 ENDDO
3508
3509 CASE(3) I.LEVEL.SELECT
3510
3511 I1 = MAX(0, NINT( GINV(DV%X1-XS,1, NM)*RDXI )-1)
3512 I2 = MIN(IBAR, NINT( GINV(DV%X2-XS,1, NM)*RDXI )+1)
3513 J1 = MAX(0, NINT( GINV(DV%Y1-YS,2, NM)*RDETA )-1)
3514 J2 = MIN(JBAR, NINT( GINV(DV%Y2-YS,2, NM)*RDETA )+1)
3515 K1 = MAX(0, NINT( GINV(DV%Z1-ZS,3, NM)*RDZETA )-1)
3516 K2 = MIN(KBAR, NINT( GINV(DV%Z2-ZS,3, NM)*RDZETA )+1)
3517
3518 DO K=K1, K2
3519 DO J=J1, J2
3520 DO I=I1, I2
3521
3522 IC1 = CELLINDEX(I, J, K)
3523 IC2 = CELLINDEX(I, J, K+1)
3524 IF (SOLID(IC1) .OR. SOLID(IC2)) CYCLE
3525
3526 IF (XC(I)<DV%X1 .OR. XC(I)>DV%X2) CYCLE
3527 IF ( YC(J)<DV%Y1 .OR. YC(J)>DV%Y2) CYCLE
3528 IF ( Z(K)<DV%Z1 .OR. Z(K)>DV%Z2) CYCLE
3529

```

Source Code files for edited portions of FDS

```

3530 DX0 = XC(1)-DV%X
3531 DY0 = YC(J)-DV%Y
3532 DZ0 = Z(K)-DV%Z
3533 VELP = PY%F0 + DX0*PY%PX(1) + 0.5._EB*(DX0*DX0*PY%PXX(1,1)+DX0*DY0*PY%PXX(1,2)+DX0*DZ0*PY%PXX(1,3)) &
3534 + DY0*PY%PX(2) + 0.5._EB*(DY0*DX0*PY%PXX(2,1)+DY0*DY0*PY%PXX(2,2)+DY0*DZ0*PY%PXX(2,3)) &
3535 + DZ0*PY%PX(3) + 0.5._EB*(DZ0*DX0*PY%PXX(3,1)+DZ0*DY0*PY%PXX(3,2)+DZ0*DZ0*PY%PXX(3,3))
3536
3537 FVZ(I,J,K) = -RDZN(K)*(HP(I,J,K)-HP(I,J,K+1)) - (VELP-WW(I,J,K))/DT
3538 ENDDO
3539 ENDDO
3540 ENDDO
3541
3542 END SELECT L.LEVEL.SELECT
3543
3544 ENDDO DEVC.LOOP
3545
3546 END SUBROUTINE PATCH.VELOCITY.FLUX
3547
3548
3549 ! ----- WALL.VELOCITY.NO.GRADH -----
3550
3551 SUBROUTINE WALL.VELOCITY.NO.GRADH(DT,STORE.UN)
3552
3553 ! This routine recomputes velocities on wall cells, such that the correct
3554 ! normal derivative of H is used on the projection. It is only used when the Poisson equation
3555 ! for the pressure is solved .NOT. PRES.ON.WHOLE.DOMAIN (i.e. using the GLMAT solver).
3556
3557 REAL(EB), INTENT(IN) :: DT
3558 LOGICAL, INTENT(IN) :: STORE.UN
3559
3560 ! Local variables:
3561 INTEGER :: IIG,JJG,KKG,IOR,IW
3562 REAL(EB) :: DHDN,VEL_N
3563 TYPE(WALL.TYPE), POINTER :: WC
3564 REAL(EB), SAVE, ALLOCATABLE, DIMENSION(:) :: UN.WALLS
3565
3566 IF (PRES.ON.WHOLE.DOMAIN) RETURN
3567
3568 STORE.UN.COND : IF ( STORE.UN ) THEN
3569
3570 ! These velocities from the beginning of step are needed for the velocity fix on wall cells at the corrector
3571 ! phase (i.e. the loops in VELOCITY.CORRECTOR will change U,V,W to wrong results using (HP1-HP)/DX gradients,
3572 ! when the pressure solver in the GLMAT solver.
3573 IF (ALLOCATED(UN.WALLS)) DEALLOCATE(UN.WALLS)
3574 ALLOCATE( UN.WALLS(1:N.EXTERNAL.WALL.CELLS+N.INTERNAL.WALL.CELLS) )
3575 UN.WALLS(:) = 0._EB
3576
3577 STORE.LOOP : DO IW=1,N.EXTERNAL.WALL.CELLS+N.INTERNAL.WALL.CELLS
3578
3579 WC => WALL(IW)
3580 IIG = WC%ONE.D%IIG
3581 JJG = WC%ONE.D%JJG
3582 KKG = WC%ONE.D%KKG
3583 IOR = WC%ONE.D%IOR
3584
3585 SELECT CASE(IOR)
3586 CASE( IAXIS)
3587 UN.WALLS(IW) = U(IIG-1,JJG ,KKG )
3588 CASE(-IAXIS)
3589 UN.WALLS(IW) = U(IIG ,JJG ,KKG )
3590 CASE( JAXIS)
3591 UN.WALLS(IW) = V(IIG ,JJG-1,KKG )
3592 CASE(-JAXIS)
3593 UN.WALLS(IW) = V(IIG ,JJG ,KKG )
3594 CASE( KAXIS)
3595 UN.WALLS(IW) = W(IIG ,JJG ,KKG-1)
3596 CASE(-KAXIS)
3597 UN.WALLS(IW) = W(IIG ,JJG ,KKG )
3598 END SELECT
3599
3600 ENDDO STORE.LOOP
3601
3602 RETURN
3603
3604 ENDF STORE.UN.COND
3605
3606 ! Case of not storing, recompute INTERNAL.WALL.CELL velocities, taking into acct that DHDN=0._EB:
3607 PREDICTOR.COND : IF (PREDICTOR) THEN
3608
3609 ! Loop internal wall cells -> on OBST surfaces:
3610 WALL.CELL.LOOP.1: DO IW=1,N.EXTERNAL.WALL.CELLS+N.INTERNAL.WALL.CELLS
3611
3612 WC => WALL(IW)
3613
3614 IF (WC%BOUNDARY.TYPE/=SOLID.BOUNDARY .AND. WC%BOUNDARY.TYPE/=NULL.BOUNDARY) CYCLE
3615
3616 IIG = WC%ONE.D%IIG

```

Source Code files for edited portions of FDS

```

3618 JJG = WC%ONED%JJG
3619 KKG = WC%ONED%KKG
3620 IOR = WC%ONED%IOR
3621
3622 DHDN=0._EB ! Set the normal derivative of H to zero for solids.
3623
3624 SELECT CASE(IOR)
3625 CASE( IAXIS)
3626 US(IIG-1,JJG ,KKG ) = (U(IIG-1,JJG ,KKG ) - DT*( FVX(IIG-1,JJG ,KKG ) + DHDN ))
3627 CASE(-IAxis)
3628 US(IIG ,JJG ,KKG ) = (U(IIG ,JJG ,KKG ) - DT*( FVX(IIG ,JJG ,KKG ) + DHDN ))
3629 CASE( JAXIS)
3630 VS(IIG ,JJG-1,KKG ) = (V(IIG ,JJG-1,KKG ) - DT*( FVY(IIG ,JJG-1,KKG ) + DHDN ))
3631 CASE(-JAXIS)
3632 VS(IIG ,JJG ,KKG ) = (V(IIG ,JJG ,KKG ) - DT*( FVY(IIG ,JJG ,KKG ) + DHDN ))
3633 CASE( KAXIS)
3634 WS(IIG ,JJG ,KKG-1) = (W(IIG ,JJG ,KKG-1) - DT*( FVZ(IIG ,JJG ,KKG-1) + DHDN ))
3635 CASE(-KAXIS)
3636 WS(IIG ,JJG ,KKG ) = (W(IIG ,JJG ,KKG ) - DT*( FVZ(IIG ,JJG ,KKG ) + DHDN ))
3637 END SELECT
3638
3639 ENDDO WALL.CELL.LOOP.1
3640
3641 ELSE ! Corrector
3642
3643 ! Loop internal wall cells -> on OBST surfaces:
3644 WALL.CELL.LOOP.2: DO IW=1,N.EXTERNAL.WALL.CELLS+N.INTERNAL.WALL.CELLS
3645
3646 WC => WALL(IW)
3647
3648 IF (WC%BOUNDARY.TYPE/=SOLID.BOUNDARY .AND. WC%BOUNDARY.TYPE/=NULL.BOUNDARY) CYCLE
3649
3650 IIG = WC%ONED%IIG
3651 JJG = WC%ONED%JJG
3652 KKG = WC%ONED%KKG
3653 IOR = WC%ONED%IOR
3654
3655 DHDN=0._EB ! Set the normal derivative of H to zero for solids.
3656
3657 VEL.N = UN.WALLS(IW)
3658
3659 SELECT CASE(IOR)
3660 CASE( IAXIS) ! | - Problem with this is it was modified in VELOCITY.CORRECTOR,
3661 ! V => Store the untouched U normal on internal WALLS.
3662 U(IIG-1,JJG ,KKG ) = 0.5._EB*( VEL.N + US(IIG-1,JJG ,KKG ) - &
3663 DT*( FVX(IIG-1,JJG ,KKG ) + DHDN ))
3664 CASE(-IAxis)
3665 U(IIG ,JJG ,KKG ) = 0.5._EB*( VEL.N + US(IIG ,JJG ,KKG ) - &
3666 DT*( FVX(IIG ,JJG ,KKG ) + DHDN ))
3667 CASE( JAXIS)
3668 V(IIG ,JJG-1,KKG ) = 0.5._EB*( VEL.N + VS(IIG ,JJG-1,KKG ) - &
3669 DT*( FVY(IIG ,JJG-1,KKG ) + DHDN ))
3670 CASE(-JAXIS)
3671 V(IIG ,JJG ,KKG ) = 0.5._EB*( VEL.N + VS(IIG ,JJG ,KKG ) - &
3672 DT*( FVY(IIG ,JJG ,KKG ) + DHDN ))
3673 CASE( KAXIS)
3674 W(IIG ,JJG ,KKG-1) = 0.5._EB*( VEL.N + WS(IIG ,JJG ,KKG-1) - &
3675 DT*( FVZ(IIG ,JJG ,KKG-1) + DHDN ))
3676 CASE(-KAXIS)
3677 W(IIG ,JJG ,KKG ) = 0.5._EB*( VEL.N + WS(IIG ,JJG ,KKG ) - &
3678 DT*( FVZ(IIG ,JJG ,KKG ) + DHDN ))
3679 END SELECT
3680
3681 ENDDO WALL.CELL.LOOP.2
3682
3683 DEALLOCATE(UN.WALLS)
3684
3685 ENDIF PREDICTOR.COND
3686
3687 RETURN
3688 END SUBROUTINE WALL.VELOCITY.NO.GRADH
3689
3690 END MODULE VELO

```

## A.4 *mod\_canopy.f90*

```
1  module penalization
2
3  REAL :: penalizationParameter, blendingParameter, dampingParameter, &
4  penXmin, penXmax, penYmin, penYmax, penZmin, penZmax, &
5  mX, mY, mZ, b, timestep, newtimestep
6
7  INTEGER :: points_Xmin, points_Xmax, points_Ymin, points_Ymax, points_Zmin, points_Zmax, i1, j1, temp,
8  cntr=0, &
9  meshresX, meshresY, meshresZ, pena_I, pena_J, pena_K
10
11  character(len=256), dimension(10) :: dataFileName
12  integer, dimension(1000) :: pendat.size
13
14  logical :: trunks
15  integer :: ntrunks, fileread, npen, penX, penY, penZ, IERROR
16  double precision :: trnk_min, trnk_max, eta
17  double precision, dimension(:, :), allocatable :: trnk_loc, pendat
18  double precision, dimension(:, :), allocatable :: pendat_store
19  real, dimension(:, :), allocatable :: penU0, penV0, penW0
20  double precision, dimension(:, :), allocatable :: arraytime
21
22
23  end module penalization
```



# Appendix B

## Publication

This section contains the works that have been published.

### Peer-reviewed Conference paper

*Title* A comparative study of wind fields generated by different inlet parameters and their effects on fire spread using Fire Dynamics Simulator.

*Conference name* 21<sup>st</sup> Australasian Fluid Mechanics Conference,  
Adelaide, Australia  
10 – 13 December, 2018

(c)

## A comparative study of wind fields generated by different inlet parameters and their effects on fire spread using Fire Dynamics Simulator

S. Singha Roy<sup>1</sup>, D. Sutherland<sup>1,2,3</sup>, N.Khan<sup>1</sup> and K. Moinuddin<sup>1,2</sup>

<sup>1</sup>Institute for Sustainable Industries and Livable Cities  
Victoria University, Melbourne, Victoria 3030, Australia

<sup>2</sup>Bushfire and Natural Hazards Cooperative Research Centre  
Melbourne, Victoria 3002, Australia

<sup>3</sup>School of Physical, Environmental and Mathematical Sciences  
University of New South Wales, Canberra, Australian Capital Territory, Australia

### Abstract

Wind is one of the most important environmental variables that affects the wildland fire spread and intensity. Modelling wind in physics-based models such as Fire Dynamics Simulator (FDS) has been shown to reproduce promising results. There are various methods available to generate wind field in FDS. The current paper deals with finding out a better approach to assign inlet conditions for fire simulations in FDS. Firstly, we explore some basic methods of wind field generation available in FDS. The conventional methods of wind field generation are either an unperturbed inlet profile with a roughness-trip or the by embedding artificial turbulence at the inlet. The wind fields generated by these inlet conditions are compared with each other as well as to the wind field generated using a mean-forcing method for neutral atmospheric conditions. Secondly, we use these inlet conditions to study the effects of fire spread in FDS, since simulating the fire plumes is not compatible with periodic boundary conditions. Finally, we test the effect of an underdeveloped boundary layer on fire spread.

### Introduction

Wildland fires occur very frequently in Australian weather conditions, especially during late spring to mid-autumn and impacts people living in the so-called wildland-urban interface. The frequency of these fires has amplified considerably due to further climatic changes [1]. These wildland fires are a resultant of many environmental factors, among which wind speed is the predominant one [2]. Therefore, accurate prediction of wind is required for accurate fire behavior prediction. Several types of models have been developed for predicting fire behavior, among which physics-based models [3] has been shown to reproduce adequate Atmospheric boundary layer (ABL) flow over flat ground and tree canopies [4]. In the current study, we have used FDS, version 6.6.0, which is a computational fluid dynamics (CFD) model of fire-driven fluid flow and the detailed description of this model can be found in [5]-[6].

The physics-based wildland fire simulations are driven by the inlet and initial boundary conditions which models the ABL. A realistic representation of ABL is required to reproduce a correct manifestation of fire in terms of rate-of-spread, intensity and heat transfer. The inlet and initial conditions prescribed for the simulation preferably leads to a realistic flow over the fire-ground which does not nonphysically develop in space and time. For example, Mell [7] used a 1/7-power-law model at the inlet of their simulations. Due to initial perturbations in the simulation, a fully turbulent flow profile will develop in time and space as the simulation progresses. The spatial and temporal development of wind flow comes with the cost of computational intensiveness to reach a fully developed profile prior to the start of the fire. Development of techniques for imposing

inlet and initial conditions for flow simulations has been a topic of interest in the field of fluid dynamics [8].

Wind can be generated with various initial and inlet conditions with FDS. One way to generate inlet condition is the recycling method of [9]. While this method is an effective way of generating a fully developed inlet condition on a single turbulent inlet such as that required for a channel flow, we aim to eventually develop a one-way nesting method for fire simulation in future, so that complicated wind fields which may change direction during the simulation can be used. The current study can be subdivided into two parts. In the first part, we will deal with the methods of wind generation. The wind can be developed either by introducing an unperturbed log-law or power-law inlet profile with a roughness trip or by superimposing eddies at the inlet with the log-law or power-law wind profile. Wind field can also be generated by using a 'mean-forcing' method following usual log-law profile. This study is limited to neutral atmospheric conditions only. The second part of this study will deal with the fire behavior. Fire simulations will be carried out using these inlet conditions and the rate of fire spread and heat-release-rate will be compared. We will also see the behavior of fire when the fire is set in an undeveloped and non-steady ABL condition. The primary intention of this study is to discover the fastest method for generating a stable wind profile which can give consistent fire spread results.

### Methodology

We tested the effectiveness of our boundary condition implementation through simulations in channel-flow configuration. The reference simulation used in this study is the wind field generation using the 'mean-forcing' method. In this method, FDS adds a mean-forcing term to the momentum equation to 'nudge' [5] the flow in the direction of specified wind velocity. In this case we need to provide any specific inlet conditions, as log-law is used by default for wind generation. The log-law can be given by equation(1)

$$u(z) = \frac{u_*}{\kappa} \left[ \ln \frac{z}{z_0} \right] \quad (1)$$

where  $u(z)$  is the wind velocity at height  $z$ ,  $u_*$  is the friction velocity,  $\kappa$  is the Von Kármán constant which is taken to be 0.41,  $z_0$  is the aerodynamic roughness length and  $z$  is the distance to the bottom wall.

The second wind field generation approach deals with the most commonly used method of wind generation; namely allowing the wind to develop naturally with the application of a roughness trip over the surface with a power-law profile enforced at the inlet. In this case, the wind develops over time and space and acquires turbulence eventually and finally reaches to a fully-developed flow condition. It takes a reasonable amount of time

for the flow to develop a constant and steady ABL. To speed up the process, the Synthetic Eddy Method (SEM), which was originally developed by Jarrin et. al.[8], can be used in FDS, which accelerates the development of a uniform boundary faster than other methods such as physical trip. This comprises our third method of wind field generation. In this method *eddies* are injected into the inlet at random positions and advect with the inlet log-law velocity inflow which subsequently gets rescaled to match the desired turbulent characteristics. FDS uses the log-law as presented by [10]. The length, velocity scales and number of eddies are the parameters that the user supplies. Typically the velocity and the length scales of the eddies should be chosen in a way so that some turbulent statistics, usually Reynolds stresses, are reproduced. [11] says that the total number of eddies can be calculated using Equation(2).

$$N = \max\left(\frac{V_B}{\sigma^3}\right) \quad (2)$$

where ( $\sigma$ ) is the size of eddies,  $V_B$  is the box volume of the inlet where the eddies are embedded. As discussed in [12], the number of eddies  $N$  should be large enough to ensure the Gaussian behaviour of the fluctuating component in each direction. In this study,  $N$  is set to 200.

FDS simulates the fire by solving a system of equations including the Navier-Stokes equations for fluid momentum, Mixing-controlled chemistry for combustion and heat transfer by conduction, convection and radiation. To save the computational cost Large Eddy Simulation (LES) is used in which the filtered Navier-Stokes equations are solved and the effect of the cut-off scales are modelled. FDS uses the Deardorff model of turbulent viscosity by default. A detailed discussion about turbulent models and LES has been given by [10]. For combustion, FDS uses a Mixing-controlled combustion model which involves one gaseous fuel where transport equations for only the lumped species, i.e. fuel and products (such as  $O_2$ ,  $CO_2$ ,  $H_2O$ ,  $N_2$ ,  $CO$  and soot), are solved (the lumped species air is the default background). In the mixture-controlled method, single fuel species that are composed primarily of  $C$ ,  $H$ ,  $O$ , and  $N$  reacts with oxygen in one mixing controlled step to form  $H_2O$ ,  $CO_2$ , soot, and  $CO$ . The reaction of fuel and oxygen is considered infinitely fast. Further details about this model can be found in [5]. Thermal degradation of solid fuel to gaseous fuel is modeled with a linear model following [13]. Radiation is accounted for by solving the radiation transfer equation with a discrete ordinates method. Convective heat transfer is modelled using a series of empirical correlations. Conduction is negligible for grassland fuels. References [6] and [5] gives further details about these models. At some critical points in calculations, like the moment of ignition, the limitations in the models or long time steps can lead to large local reaction rates, which can lead to numerical instabilities. An upper bound on the local heat release rate per unit volume needs to be maintained in order to prevent this. Following the scaling analysis of pool fires by [14], FDS 6.2.0 uses an upper bound following Equation(3):

$$q'''_{upper} = 200/\delta x + 2500kW/m^3 \quad (3)$$

FDS 6.6.0 does not use a reaction rate threshold, instead expecting the computation to be sufficiently resolved to avoid such numerical instabilities. The resolution requirement is prohibitive for large-scale wildfire simulations. However, we introduce the threshold Equation(3) to be consistent with previous fire simulations [15] and to avoid restrictive grid resolution requirements. The fire simulations for the current paper has been conducted using this current edited version of FDS 6.6.0. There are two cases of fire simulations that have been performed for the current study. In the first case, the most widely used log-law inlet condition has been used,

which is similar to the first wind simulation, and the fire is started after the upstream of the fire reaches a steady-state wind profile obtained from the wind simulations. The second fire simulation uses SEM introduced at the inlet, with conditions similar to the SEM wind simulation mentioned previously.

#### Simulation Domain

The size of the external domain is chosen such that it ensures to capture the largest relevant structures. The overall domain size for all the simulations is taken to be 130m X 40m X 80m. Inlet velocity of 4.7 m/s is given at a height of 10 m. The mean velocity of  $\sim 5.5m/s$  at fully developed state is maintained at 2m for all the simulations. 40 m from the inlet in the longitudinal direction, the burnable grass plot (40mX40m) was placed so that there was another 50 m subdomain downstream of the non-burnable grass plot before reaching an open outlet. The spanwise of the flow stream is set to periodic boundary conditions. In case of the fire simulations, a line fire is ignited which covers the width of the domain (along y) as used by [16]. The simulation domain has been divided into multiple meshes with different grid sizes. To avoid any numerical instabilities, the aspect ratio is maintained not more than 2 for any grid cell. The sub-domain with burnable grass plot has 0.25 m grid resolution in all direction throughout the height of the domain. The fuel parameters used in the simulations were replicated as done by Moinuddin et.al.[15]. Figure(1) represents a generalized domain used for all the simulations.

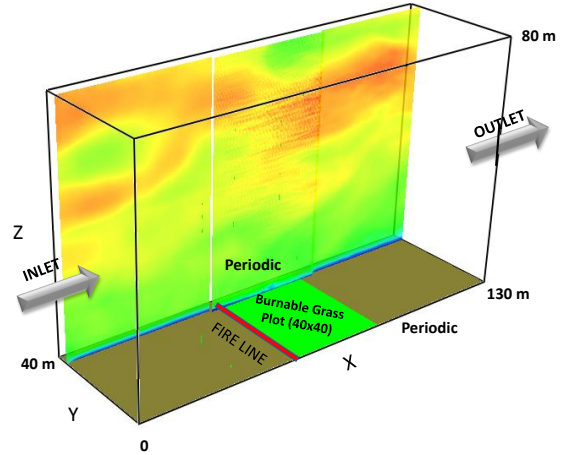


Figure 1: Domain of simulation showing the dimensions, fire plot, fire line and establishment of ABL.

All other relevant information regarding the wind simulations are given in Table 1 and that for fire simulations are given in Table 2. The simulations will be depicted using the case names given in the table hereafter.

Table 1: Wind Simulations

Case name	Generation method	Mean profile	Turbulent profile
wind0	mean-forcing	Log-law	—
wind1	Roughness change-trip	1/7 Power law	—
wind2	Explicit log-law	Log-law	SEM

Table 2: Fire Simulations

Case name	Generation method	Mean profile	Turbulent profile
fire0	Underdeveloped ABL	1/7 Power law	—
fire1	Roughness change-trip	1/7 Power law	—
fire2	Explicit log-law	Log-law	SEM

## Results and Discussions

Several numerical parameters like inlet conditions, domain size, grid resolution and boundary layer development time are considered for a systematic approach. In our study, we are considering a small domain, and our results are strictly according to the parameters that we have used. The results may vary with different domain size, grid size, inlet conditions or wind velocities. The wind simulations *wind0*, *wind1* and *wind2* are run for

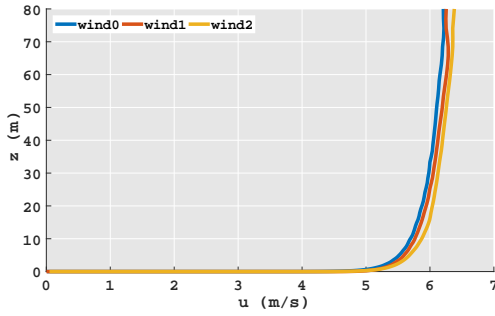


Figure 2: The mean velocity profile comparison for 3 wind cases over the fire ground.

5000 seconds of simulation time to find out time for a stable ABL to get established. We observe that *wind0* acquires a stable ABL in less than 100 seconds. In case of *wind1*, the ABL is established in approximately 1000-1200 seconds, whereas for *wind2*, it takes less than 1000 seconds. Figure(2) depicts the mean wind velocity profile on the fire-plot before the start of the fire. Figure(2) depicts that the three wind simulation cases produces similar mean velocity profiles. In fire simulations the mean u-velocity profile as a function of height is more informative than examining other quantities in wall units. We examined the TKE(Turbulent Kinetic Energy) and confirmed that the flow was well developed for each cases when the TKE was oscillating around a constant value. In case of *wind1*, the flow trips and become turbulent leading to a developing boundary layer. This results in more computational time for wind to get stabilized. On the other hand for *wind2*, since the turbulence is embedded in the form of synthetic eddies along with the inlet log-law profile, the flow develops faster. We observe that the mean profile pattern for *wind0* agrees well with *wind1* and *wind2*.

We have used the stabilized wind-field generated in *wind0* simulation as the initial condition for the fire simulations *fire0*, *fire1* and *fire2* to reduce the time to reach the steady-state ABL over the fire ground and start the fire. We have started the fire for *fire1* and *fire2* after 300 seconds in order to allow a steady-state ABL to develop prior starting the fire. For *fire0* case, we have located the burnable-grass plot near the inlet with minimum upstream of the fire, so that the wind is not allowed to get stabilized over space and started the fire after 100 seconds. The intention here is to not allow the steady-state ABL establishment prior to the start of the fire. We have done some adjustments over the axes so that *fire0* can be plotted against *fire1* and *fire2* for comparison. The fire ignitor was put off after 11 seconds [7]. The fire took about  $\sim 25$  seconds to burn the burnable grass plot completely for all the three cases. The fire propagated in a straight

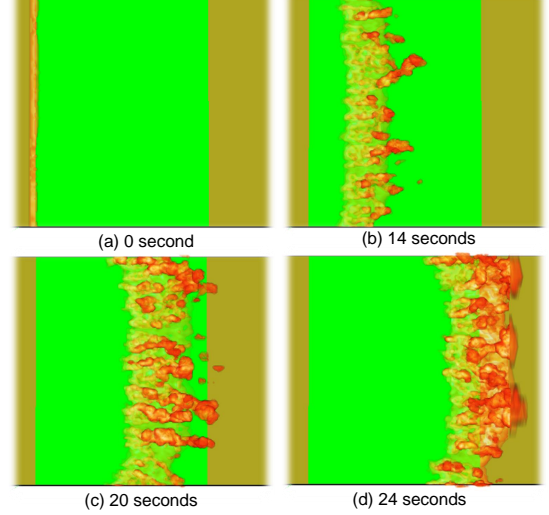


Figure 3: Fire propagation contour for fire1.

line across the domain as shown in Figure(3). Figure(4) depicts the percentage change of wind speeds during the burn at 2.04m over the fire plot. This depicts the percentage change in wind speed varies in a similar pattern for all the three cases.

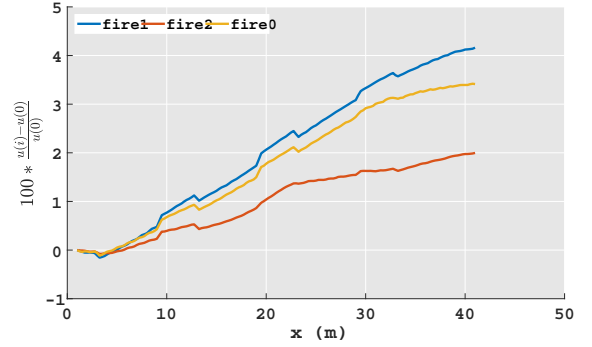


Figure 4: Wind speed variation over the burnable-grass plot at 2.04 m height.

There are various parameters for comparing the simulated fire. In the current study, we have compared the Heat Release Rate (HRR) and the Rate of Spread (ROS) to predict the nature of fire propagation. HRR represents the height or intensity of fire whereas ROS depicts fire spread with respect to time.

Figure(5) depicts the HRR for all the three fire simulations to be similar. we observe that the HRR reaches maximum when the fire has consumed the whole burnable fuel over the fire plot (at about 25 seconds) and then drops down to zero as the plume exists the domain. For the fire simulations, the ROS has been calculated at the maximum value of the fire-front on the boundary where the temperature of the vegetation is above 400K-500K (the pyrolysis temperature). From Figure(6), we observe that towards the start of the fire, the ROS is maximum, then it reaches a quasi-steady of about 2m/s state while burning down the whole fire plot and the reaches zero when whole of the burnable fuel has been consumed.

The fire propagation and its characteristics agree good in both *fire1* and *fire2*. As discussed previously that *fire0* simulation was carried out in an underdeveloped boundary which means that the fire was started in an unsteady ABL condition. However, the fire propagation is not much affected by this. It can be argued that the domain considered in this study is comparatively smaller, and so the steady-state ABL is getting established in as

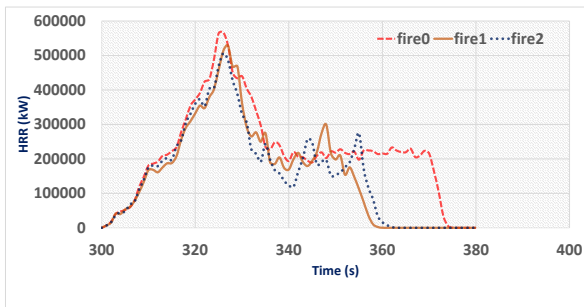


Figure 5: Heat Release Rate (HRR) as functions of time

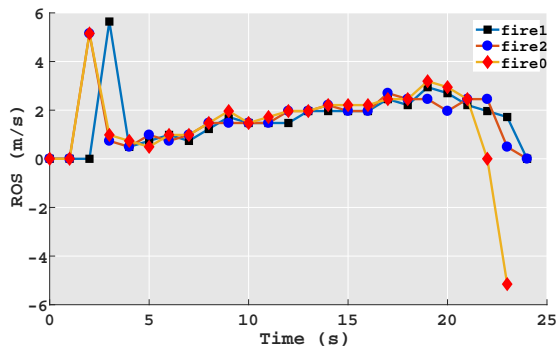


Figure 6: ROS vs Time comparison over fire plot.

short as  $\sim 20m$  in fire upstream. So we see a fire propagation pattern similar to the other cases. The simulation results may vary considerably for larger burnable grass domain.

## Conclusion

The wind simulations performed in this study shows that the SEM and the roughness trip method for wind simulation produce similar steady-state wind profiles to that generated by the mean-forcing method. The mean-forcing method generates a steady-state profile faster than the SEM and roughness trip method and hence uses lesser computational time. The mean-forcing method and roughness-trip method also require fewer input parameters than the SEM. The HRR and ROS profiles shows very little difference between the three fire cases. Therefore, simplicity suggests just taking a 1/7th power-law and a very short upstream distance and spin up time is a simple approach which still recovers the RoS results of more complicated methods. We look forward to developing a method in the future where we can use real-time terrain modified wind data to perform more realistic fire simulations. This method will lead to reduced simulation initialisation time.

## Acknowledgement

This research is supported by VU Strategic Research Scholarship (Bushfires), Victoria University, Melbourne.

## References

- [1] W Matt Jolly, Mark A Cochrane, Patrick H Freeborn, Zachary A Holden, Timothy J Brown, Grant J Williamson, and David MJS Bowman. Climate-induced variations in global wildfire danger from 1979 to 2013. *Nature communications*, 6(7537):1–11.
- [2] Richard C Rothermel. A mathematical model for predicting fire spread in wildland fuels. *Research Paper INT-115. Ogden, UT: US Department of Agriculture, Intermountain Forest and Range Experiment Station.*, 115:1–40, 1972.
- [3] Andrew L Sullivan. Wildland surface fire spread modelling, 1990–2007. 1: Physical and quasi-physical mod-

els. *International Journal of Wildland Fire*, 18(4):349–368, 2009.

- [4] William Mell, Alexander Maranghides, Randall McDermott, and Samuel L. Manzello. Numerical simulation and experiments of burning douglas fir trees. *Combustion and Flame*, 156(10):2023 – 2041, 2009.
- [5] Kevin McGrattan, Simo Hostikka, Randall McDermott, Jason Floyd, and Marcos Vanella. *Fire dynamics simulator: user's guide*. US Department of Commerce, Technology Administration, National Institute of Standards and Technology, 2017.
- [6] Kevin McGrattan, Simo Hostikka, Randall McDermott, Jason Floyd, and Marcos Vanella. *Fire dynamics simulator, technical reference guide Volume 1: Mathematical Model*. US Department of Commerce, Technology Administration, National Institute of Standards and Technology, 2017.
- [7] William Mell, Mary Ann Jenkins, Jim Gould, and Phil Cheney. A physics-based approach to modelling grassland fires. *International Journal of Wildland Fire*, 16(1):1–22, 2007.
- [8] Nicolas Jarrin, Sofiane Benhamadouche, Dominique Laurence, and Robert Prosser. A synthetic-eddy-method for generating inflow conditions for large-eddy simulations. *International Journal of Heat and Fluid Flow*, 27(4):585–593, 2006.
- [9] Thomas S Lund, Xiaohua Wu, and Kyle D Squires. Generation of turbulent inflow data for spatially-developing boundary layer simulations. *Journal of computational physics*, 140(2):233–258, 1998.
- [10] Stephen B Pope. *Turbulent flows*, 2001.
- [11] N Jarrin, R Prosser, J-C Uribe, S Benhamadouche, and D Laurence. Reconstruction of turbulent fluctuations for hybrid rans/les simulations using a synthetic-eddy method. *International Journal of Heat and Fluid Flow*, 30(3):435–442, 2009.
- [12] Dimitrios Pavlidis, Gerard J Gorman, Jefferson LMA Gomes, Christopher C Pain, and Helen ApSimon. Synthetic-eddy method for urban atmospheric flow modelling. *Boundary-layer meteorology*, 136(2):285–299, 2010.
- [13] D Morvan and JL Dupuy. Modeling the propagation of a wildfire through a mediterranean shrub using a multi-phase formulation. *Combustion and flame*, 138(3):199–210, 2004.
- [14] Lawrence Orloff and John De Ris. Froude modeling of pool fires. In *Symposium (International) on Combustion*, volume 19, pages 885–895. Elsevier, 1982.
- [15] Khalid Moinuddin, Duncan Sutherland, and William(Ruddy) Mell. Simulation study of grass fire using a physics-based model:achieving numerical rigour and the effect grass height on the rate-of-spread. *International Journal of Wildland Fire*, accepted for publication, 2018.
- [16] RR Linn, JM Canfield, P Cunningham, C Edminster, J-L Dupuy, and F Pimont. Using periodic line fires to gain a new perspective on multi-dimensional aspects of forward fire spread. *Agricultural and Forest Meteorology*, 157:60–76, 2012.

# Appendix C

## Monin-Obukhov Similarity Theory

The Monin-Obukhov similarity theory (Monin and Obukhov (1954)) states that a horizontally homogeneous atmospheric surface layer is governed by only four parameters:  $z, u_\tau, g/T_0, Q_0$ , where  $z$  is the vertical distance from the ground,  $u_\tau$  is the surface friction velocity,  $g/T_0$  is the buoyancy parameter and  $Q_0$  represents the surface temperature flux. The non-dimensionalised mean temperature and mean wind-flow in the surface layer under non-neutral atmospheric conditions is a function of the dimensionless height parameter  $z/L$ , where  $L$  is known as Monin-Obukhov scale length, given by:

$$L = \frac{-u_\tau^3}{\kappa(g/T_0)Q_0} \quad (\text{C.1})$$

where  $\kappa$  is the *Von Kármán* constant with a value of 0.41. The wind speed profile  $u(z)$  and the potential temperature  $T(z)$  varies with height  $z$ , according to the following equations following (Monin and Obukhov (1954)):

$$u(z) = \frac{u_\tau}{\kappa} \left[ \ln\left(\frac{z}{z_0}\right) - \psi_m\left(\frac{z}{L}\right) \right] \quad (\text{C.2})$$

$$T(z) = T_0 + \frac{T_*}{\kappa} \left[ \ln\left(\frac{z}{z_0}\right) - \psi_h\left(\frac{z}{L}\right) \right] \quad (\text{C.3})$$

for  $z \gg z_0$ , where  $z_0$  is the aerodynamic roughness,  $T_*$  is the scaling potential temperature,  $T_0$  is the ground level potential temperature and  $\psi_m$  and  $\psi_h$  are are similarity functions which are obtained from measurements. The most common similarity func-

tions used are those proposed by Dyer (1974). The velocity is calculated by means of nudging technique as discussed in section(2.2.3). The Obukhov length,  $L$ , characterises the thermal stability of the atmosphere. The atmosphere is said to be *stable* when the atmospheric temperature is more than the surface temperature and the surface acts as a heat sink, usually during the night time. The value of  $L$  becomes positive in stable conditions. The atmosphere is said to be *unstable* when the opposite thing happens and the surface acts as a heat source, especially during the day time. The value of  $L$  is negative at unstable conditions. The *near-stable* or *neutral* atmospheric condition is achieved when the temperature of both the air and surface are same. The value of  $L$  becomes infinity in this case. The atmospheric stability based on the stability parameter  $h/L$  following McGrattan et al. (2017d) can be given in table (C.1) :

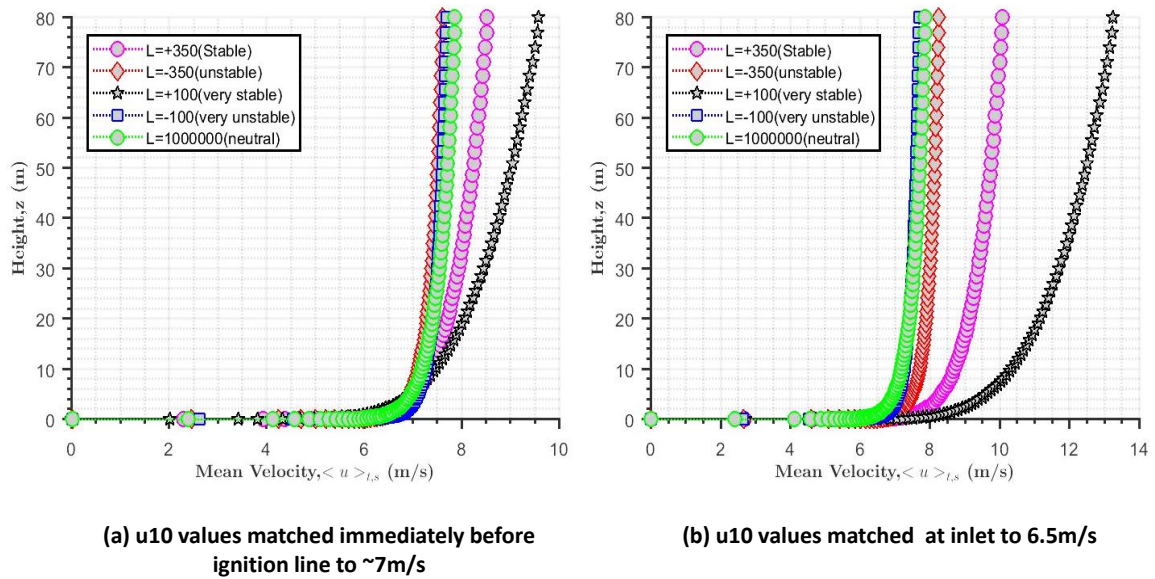
Stability	$h/L$ value range	Suggested Value
<b>Very Unstable</b>	$-200 \leq L < 0$	-100
<b>Unstable</b>	$-500 \leq L < -0.02$	-350
<b>Neutral</b>	$ L  > 500$	1000000
<b>Stable</b>	$200 < L \leq 500$	350
<b>Very stable</b>	$0 < L \leq 200$	100

Table C.1: Different Atmospheric stability parameters

### Wind and Fire simulations at different stabilities

The ABL is modelled using the Monin-Obukhov similarity theory in FDS. It is observed that Obukhov length  $L$  is responsible for characterising the atmospheric stability. The values given in (C.1) have been used to carry out this preliminary study of fire propagation at various atmospheric stabilities. This is a new inclusion in FDS 6.6.0, and hence a preliminary study has been done here. A domain similar to the *small domain* has been used. The dimensions and properties of the domain is same as the *small domain* as given in table(3.3).

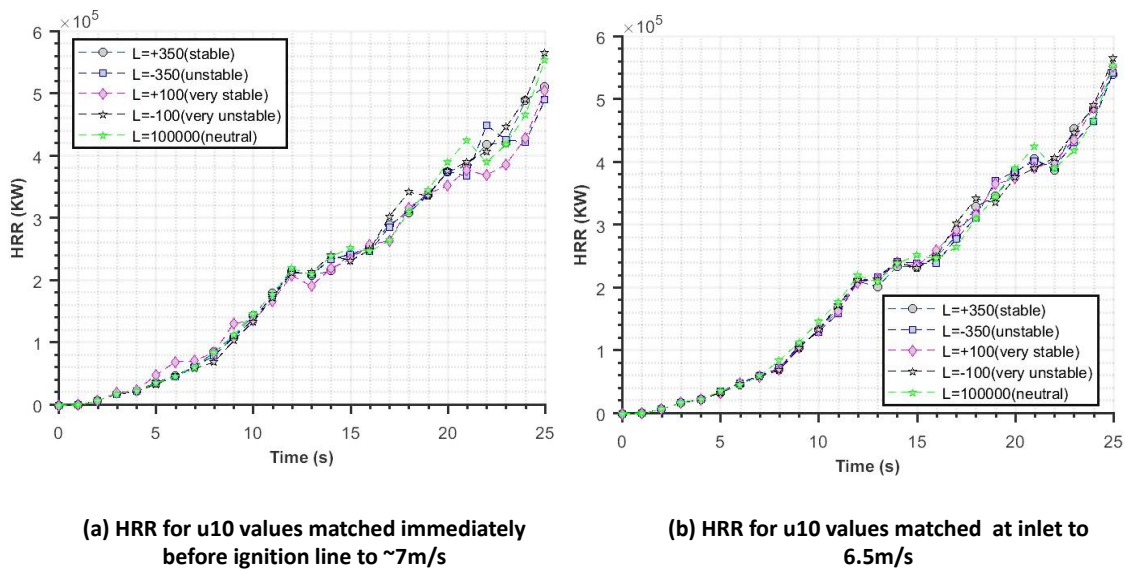




**Figure C.1:** Mean velocity profile over the fire-plot at various atmospheric stabilities: (a) The  $u_{10}$  velocities for different stabilities are matched immediately before the ignition line at  $\sim 7\text{m/s}$ , the inlet velocity may vary ; (b) The  $u_{10}$  velocities matched at the inlet to  $6.5\text{m/s}$  for different stabilities.

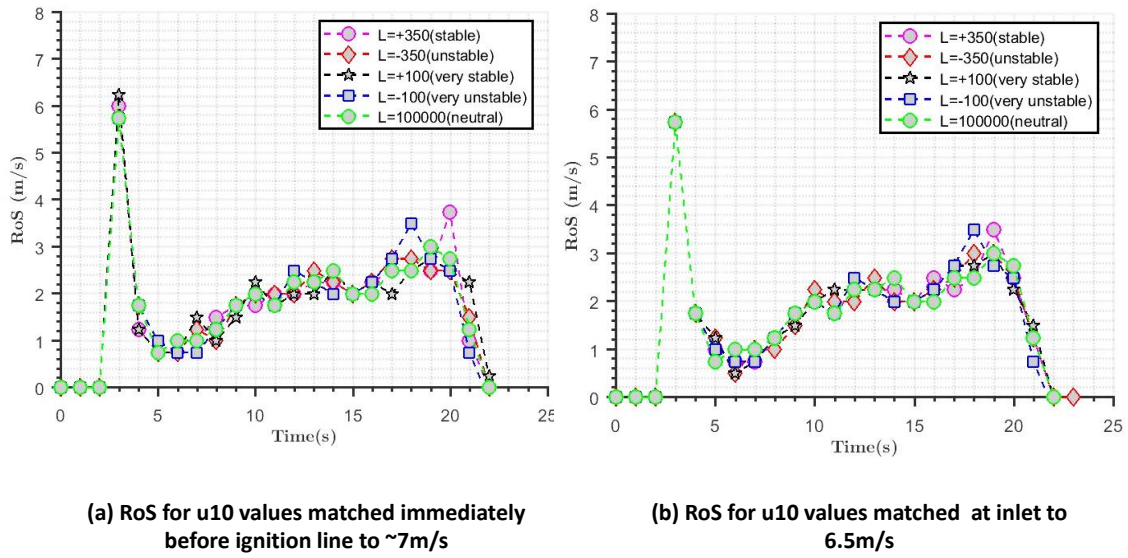
When the atmosphere is unstable, the turbulence is generated from the heat transfer from the heated surface. On the other hand, when the atmosphere is stable, which is generated from cooling of the surface, it suppresses the turbulence. This kind of atmosphere is characterised by strong wind shear and small eddies. Two cases have been considered in this study. For *case1*, the  $u_{10}$  is matched immediately before the ignition line. The inlet velocities are different for different stabilities, in this case. For *case2*,  $u_{10}$  is matched at the inlet, and is taken to be  $6.5\text{ m/s}$  for different stability cases. Due to the effect of stability, the  $u_{10}$  is different immediately before the ignition line, in this case. The wind profiles for these two cases can be shown in figure(C.1).





**Figure C.2:** The HRR plot for the fire simulation at different atmospheric stabilities: (a) The  $u_{10}$  velocity is matched immediately before the ignition line to  $\sim 7\text{m/s}$  for various stabilities; (b) The  $u_{10}$  velocity is matched at the inlet to  $6.5\text{m/s}$  for various stabilities.

On conducting fire simulations at various atmospheric stabilities using Monin-Obukhov similarity theory in FDS, the RoS and HRR profiles are found to be surprising. It can be observed from figures(C.2-b, C.3-b), that the HRR and the RoS profiles for different atmospheric stabilities, with same inlet velocity, appears to follow similar trend and overlap on each other with minimum deviation. Also, the time taken by the fire to reach the end of fire plot is same for all the stability cases. Eventhough the  $u_{10}$  velocities at all the stabilities have been matched immediately before the ignition line, there is not much change in the HRR and RoS plots for these cases, and looks identical with minimum deviation (figures(C.2-b, C.3-b)).



**Figure C.3:** The RoS of the fire over the fire plot at different atmospheric stabilities; (a) The  $u_{10}$  velocity matched immediately before the ignition line to  $\sim 7\text{m/s}$  for various stabilities; (b) The  $u_{10}$  velocity is matched at the inlet to  $6.5\text{m/s}$  for various stabilities.

This shows that the Monin-Obukhov similarity theory as implemented in FDS is still pre-mature to conduct fire simulations and the results currently are not reliable. These needs further investigations, which is out of scope of the current studies. These investigations can be taken up as a part of future research.