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

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 PenaBlending method can apply any general driving wind, and the effect of atmospheric stability, could be included, as part of future studies. The PenaBlending 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.

[redacted]................

Sesa Singha Roy 29 May, 2019

ACKNOWLEDGEMENT

Prima facea, 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

List of Figures

List of Tables

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
FORTRAN	FORmula TRANslator
CFD	Computational Fluid Dynamics
RANS	Reynolds Averaged Navier-Stokes
SGS	Sub-Grid Scale
SEM	Synthetic Eddy Method
$pSim$	precursor simulation
FS	Fire-spread Simulation
HRR	Heat Release Rate
HRRPUV	Heat Release Rate per Unit Volume

ρ	fluid density
U	instantaneous velocity
p	pressure
g	gravity
τ_{ij}	viscous stress tensor
$\bar{U}$	filtered velocity
ν	molecular viscosity
S_{ij}	strain-rate tensor
τ_{turb}	sub-grid scale Reynolds stress
ν_T	turbulent viscosity
δ_{ij}	Kronecker delta
C_v	Deardorff model constant
k_{sgs}	sub-grid scale kinetic energy
τ_w	wall shear stress
u^+	non-dimensional stream-wise velocity
y^+	non-dimensional wall normal distance
u_τ	friction velocity
κ	*von Kármán* constant
s^+	roughness length in viscous units
δ_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
$\triangle h_{vap}$	latent heat of evaporation
$\dot{m_{pyr}}$	rate of pyrolysis
$\triangle h_{pyr}$	heat of pyrolysis/ heat of reaction

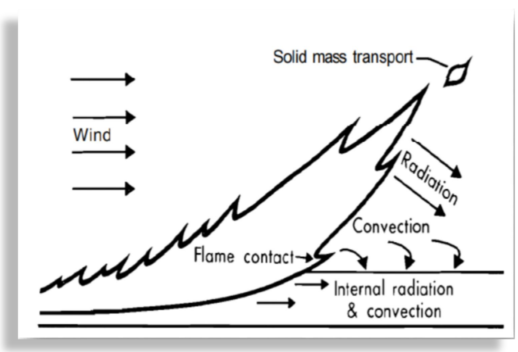
$a_{CO_2}, a_{H_2O}, a_{CO}, a_{N_2}$	stoichiometric coefficients
$\zeta(t)$	unmixed fraction
τ_{mix}	mixing time scale
u_0	reference velocity
z	domain height
$z0$	specified height of the domain
$p = 1/7$	empirically derived value of power in power-law
N	number of eddies
σ	size od eddies
V_B	cross-section 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
χ_{blend}	blending factor
η_{blend}	blending parameter
F_{penal}	penalisation forcing term
χ_{penal}	penalisation factor
η_{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
$\triangle H_{f,\alpha}$	heat of formation

$\bar{}$	average
$\widehat{}$	weighted average
CO_2	carbon-di-oxide
H_2O	water
N_2	nitrogen
CO	carbon-monoxide

Chapter 1

Introduction

WIND is an eminent environmental factor that plays a major role in affecting wildland 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)*

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 surface 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.

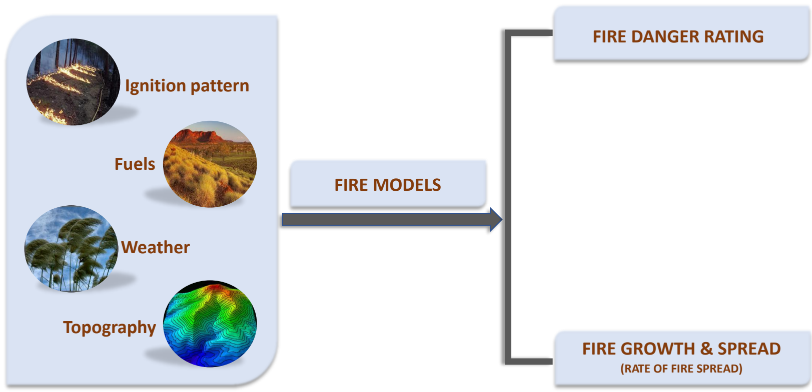


***Figure 1.3:** An overview of wildland fire behaviour and their relation to its fire modelling*

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 Section1.2.1.

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 operates 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 wildfire 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	McArthur (1967)	(a)
CSIRO grassland model	Cheney et al. (1998)	
CSIRO forest model	Gould et al. (2007)	
Rothermal Model	Rothermel (1972)	
Simulators		
BEHAVE	Andrews (1986)	(b)
PHOENIX	Tolhurst et al. (2008)	
SPARK	Miller et al. (2015)	
FARSITE	Finney (1998)	

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 models 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/open?id=18uQEmprdpmNDgBGIDVswu9ER_mHpN9Ho). 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).

$$\frac{\partial \rho}{\partial t} + \bigtriangledown.\rho U = 0, \tag{2.1}$$

where $\bigtriangledown.\rho U$ 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.

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

where pressure p, gravity g, viscous stress tensor τ_{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 boyuancy.The stress tensor can be defined as:

$$\tau_{ij} = \mu(2\overline{S_{ij}} - \frac{2}{3}(\bigtriangledown.\bar{U})\delta_{ij}), \tag{2.3}$$

where μ represents the molecular viscosity and $\overline{S_{ij}}$ represents the strain-rate tensor given by :

$$\overline{S_{ij}} = \frac{1}{2}(\frac{\partial \overline{u_i}}{\partial x_j} + \frac{\partial \overline{u_j}}{\partial x_i}) \tag{2.4}$$

The τ_{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).

$$\tau_{turb} = \rho\nu_T\overline{(2S_{ij}} - \frac{2}{3}(\bigtriangledown.\bar{U})\delta_{ij}) \tag{2.5}$$

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

$$\nu_T = \rho C_v \bigtriangleup \sqrt{k_{sgs}} \tag{2.6}$$

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

$$k_{sgs} = \frac{1}{2}((\overline{u} - \widehat{u})^2 + (\overline{v} - \widehat{v})^2 + (\overline{w} - \widehat{w})^2) \tag{2.7}$$

where $\overline{u}$ represents the average value of u at the centre of the grid cell and $\widehat{u}$ 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, τ_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).

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

where u^+ represents the non-dimensional stream-wise velocity, ($u^+ = u/u_\tau$), u_τ 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).

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

where $s^+ = s/\partial_y$ represents roughness length in viscous units, ∂_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:

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

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).

$$\dot{m}_{vap} = \frac{\dot{Q}_{net}}{\triangle h_{vap}}\ , \tag{2.11}$$

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 $\triangle h_{vap}$ 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.

$$If \ \ 400\ K \leq T_s \leq 500\ K, \quad \dot{m}_{pyr} = \frac{\dot{Q}_{net}}{\triangle h_{pyr}} \chi \frac{\dot{T}_s - 400}{500 - 400}\ , \tag{2.12}$$

where $\dot{m}_{pyr}$ represents the rate of pyrolysis and $\triangle h_{pyr}$ 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).

$$C_xH_yO_zN_v + a_{O_2} \rightarrow a_{CO_2}CO_2 + a_{H_20}H_2O + a_{CO}CO + a_S Soot + a_{N_2}N_2 \tag{2.13}$$

where $a_{CO_2}, a_{H_20}, a_{CO}, a_S, a_{N_2}$ 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).

$$\frac{d\zeta}{dt} = -\frac{\zeta}{\tau_{mix}} \tag{2.14}$$

where τ_{mix} denotes the mixing time scale. ζ 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, ζ 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) stablise over the fire ground. In FDS, the wall of wind can be specified as follows:

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

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).

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

where σ (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° 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.

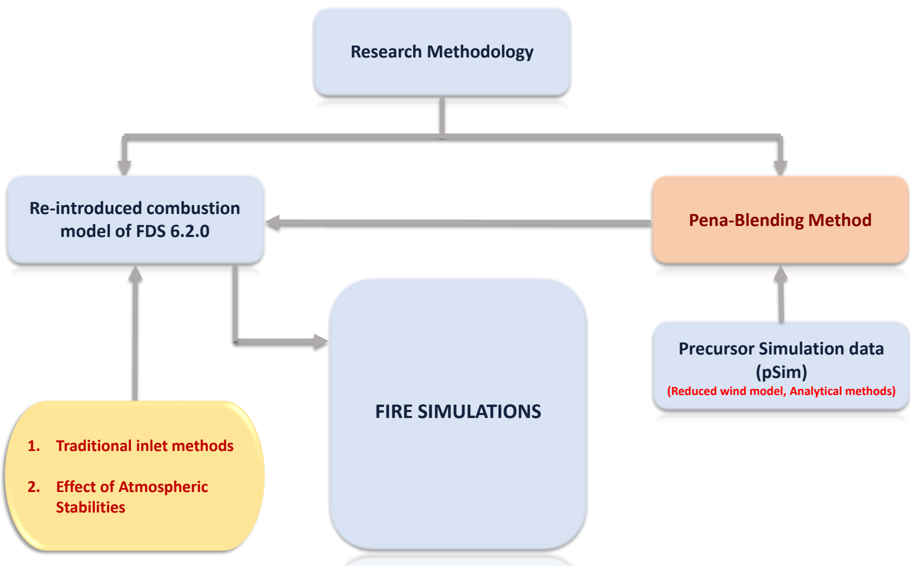


Figure 3.1: *Schematic representation of research methodology*

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 products 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 modern 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 transition 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 *PenaBlending 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.

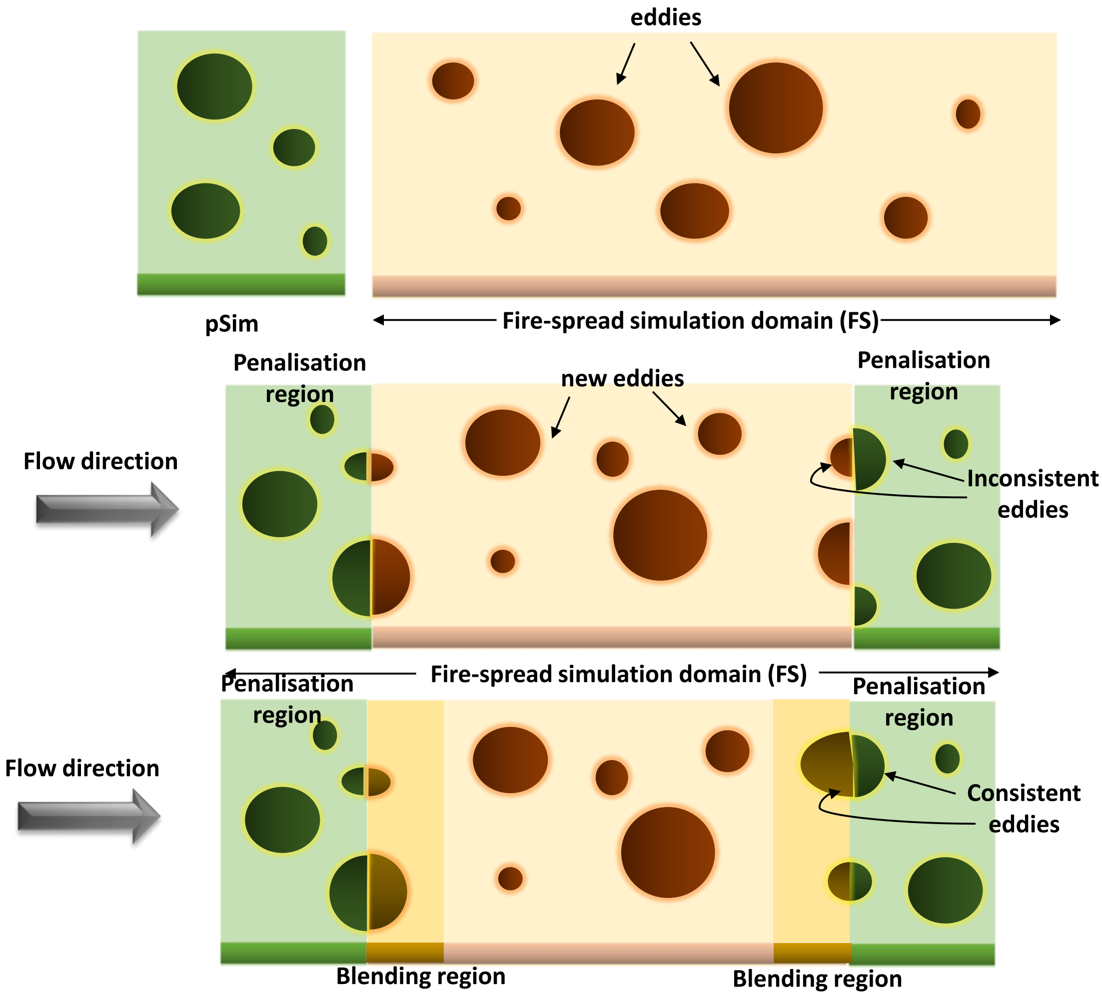


Figure 3.2: *Schematic representation of turbulent eddies in FS domain using the PenaBlending method*

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.

$$\frac{\partial \overline{u_i}}{\partial t} + \overline{u_j}\left(\frac{\partial \overline{u_i}}{\partial y_j} - \frac{\partial \overline{u_j}}{\partial x_i}\right) = \frac{1}{\rho}\frac{\partial \overline{p}}{\partial x_j} + \frac{\partial \tau_{ij}}{\partial x_j} + F\,, \tag{3.1}$$

$$\frac{\partial u_i}{\partial x_i} = 0\,, \tag{3.2}$$

where $\overline{u_i}$ is the resolved part of velocities, $i, j \in (x, y, z)$ are the three spatial coordinates, ρ is the fluid density, $\overline{p}$ is the resolved pressure, τ_{ij} is the deviatoric part of the stress tensor, given by :

$$\tau_{ij} = \overline{u_i u_j} - \overline{u_i}\ \overline{u_j} \tag{3.3}$$

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:

$$F = F_{blend} + F_{penal} + F_{damp} + F_{drag} + ...\ , \tag{3.4}$$

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

$$\begin{aligned} F_{penal} &= \frac{\chi_{penal}}{\eta_{penal}}(\overline{u_i}^{pSim} - \overline{u_i}) \\ where, \quad \chi_{penal}(x,y,z) &= \begin{cases} 1 & X_{min} \ \le \ x \ \le \ X_{max}, \\ & Y_{min} \ \le \ y \ \le \ Y_{max}, \\ & Z_{min} \ \le \ z \ \le \ Z_{max}, \\ 0 & otherwise, \end{cases} \end{aligned} \tag{3.5}$$

The term χ_{penal} represents that the 'penalisation factor' is applied uniformly in space. The *penalisation parameter* (η_{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 η_{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* [4]. The part of the source code showing this region can be given in the following lines of code:

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

!checking if inside penalisation boundaries
if ((Z(K).le.pendat(ll,9)).and.(Z(K).ge.pendat(ll,8))) then
if ((Y(J).le.pendat(ll,7)).and.(Y(J).ge.pendat(ll,6))) then
if ((X(I).le.pendat(ll,5)).and.(X(I).ge.pendat(ll,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(ll,13))*(UU(I,J,K)- pendat(ll,14+cntr)))
    /pendat(ll,1)
FVY(I,J,K) = FVY(I,J,K) + ((pendat(ll,13))*(VV(I,J,K)- pendat(ll,14+size(
    penU0)+cntr)))/pendat(ll,1)
FVZ(I,J,K) = FVZ(I,J,K) + ((pendat(ll,13))*(WW(I,J,K)- pendat(ll,14+size(
    penU0)+size(penV0)+cntr)))/pendat(ll,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:

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

[4]The complete *velo.f90* source code is in Appendix I

The *blending factor* χ_{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).

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

The relaxation scale(η_{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 η_{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, η_{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.

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

The complete code including the implementation of a blending region has been included in the *velo.f90* source file [3] 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(ll,9)).and.(Z(K).ge.pendat(ll,8))) then
if ((Y(J).le.pendat(ll,7)).and.(Y(J).ge.pendat(ll,6))) then
if ((X(I).le.pendat(ll,5)).and.(X(I).ge.pendat(ll,4))) then

!assigning the counter to traverse the array(pendat) where the penalisation
```

[3] 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(ll,13) + pendat(ll,10)*X(I)-((pendat(ll
    ,10)*(1-pendat(ll,10))*pendat(ll,5))/2d0) &
-(pendat(ll,10)*(1+pendat(ll,10))*pendat(ll,4)/2d0))*(UU(I,J,K)-pendat(ll
    ,14+cntr))/pendat(ll,2) &
+ (pendat(ll,13) + pendat(ll,11)*Y(J)-((pendat(ll,11)*(1-pendat(ll,11))*
    pendat(ll,7))/2d0) &
-(pendat(ll,11)*(1+pendat(ll,11))*pendat(ll,6)/2d0 ))*(UU(I,J,K)-pendat(ll
    ,14+cntr))/pendat(ll,2) &
+ (pendat(ll,13) + pendat(ll,12)*Z(K)-((pendat(ll,12)*(1-pendat(ll,12))*
    pendat(ll,9))/2d0) &
-(pendat(ll,12)*(1+pendat(ll,12))*pendat(ll,8)/2d0 ))*(UU(I,J,K)-pendat(ll
    ,14+cntr))/pendat(ll,2)

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

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

```
-(pendat(ll,11)*(1+pendat(ll,11))*pendat(ll,6)/2d0 ))*(WW(I,J,K)-pendat(ll
    ,14+size(penU0)+size(penV0)+cntr))/pendat(ll,2) &
+ (pendat(ll,13) + pendat(ll,12)*Z(K)-((pendat(ll,12)*(1-pendat(ll,12))*
    pendat(ll,9))/2d0) &
-(pendat(ll,12)*(1+pendat(ll,12))*pendat(ll,8)/2d0 ))*(WW(I,J,K)-pendat(ll
    ,14+size(penU0)+size(penV0)+cntr))/pendat(ll,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) [4]):

```
pendat(ll,:)=(/ penalizationParameter, blendingParameter, dampingParameter,
     &
penXmin, penXmax, penYmin, penYmax, penZmin, penZmax, &
mX, mY, mZ, b, timesttep, &
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

[4]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*. $timesttep$ 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,
```

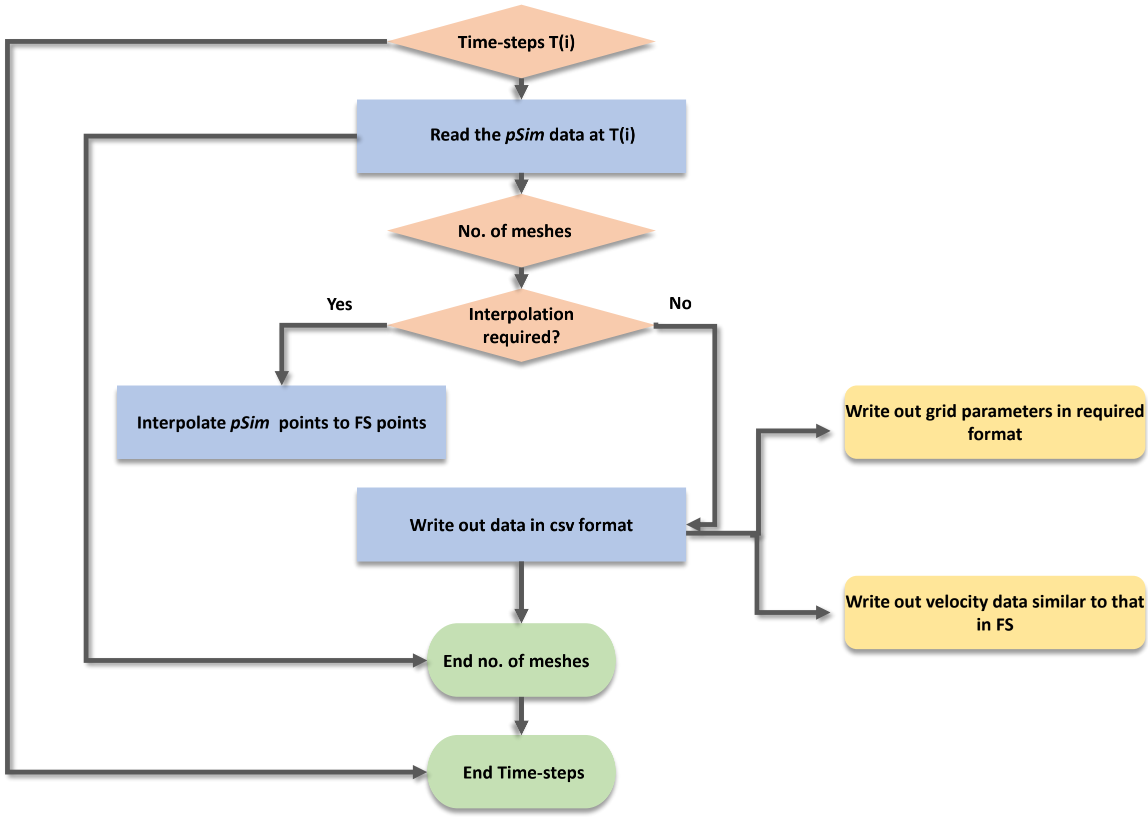


Figure 3.3: Flowchart representation of the pre-processing algorithm for pSim data

```
pena_K=80,
dataFileName='sample_read_blend_in_1.csv'
```

Here the *penalizationParameter* and *blendingParameter* represents the η_{penal} and η_{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.

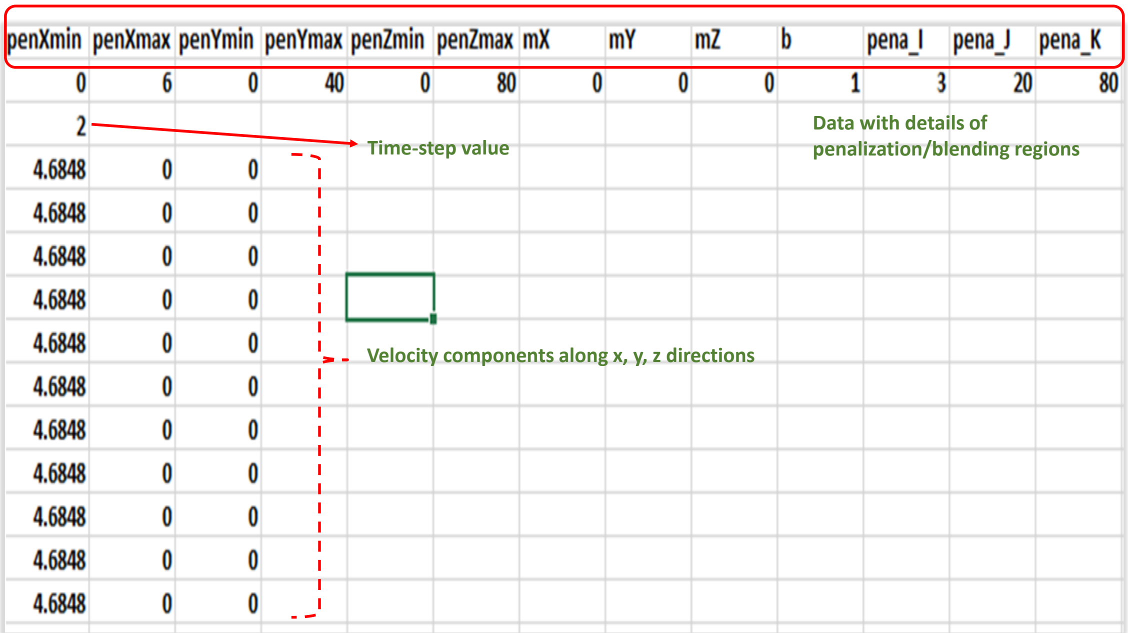


Figure 3.4: *Format for writing out pSim data*

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* [5]:

```
!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-

[5] The complete *read.f90* source code is in Appendix I

segment from *read.f90*: [5]

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

(a)

Boundary conditions	*wind simulations*
Lateral	Periodic (*wind1*, *wind2*)
	Open (*wind3*)
	Free-slip, no normal velocity (*wind4*)
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 wind2)
N_{eddy}	40 (for wind2)
σ_{eddy}(*large domain*)	1.0 ms $^{-1}$ if $z < 20$ m
	0.5 ms $^{-1}$ if $20 \leq z < 40$ m
	0 ms $^{-1}$ if $z \geq 40$ m
σ_{eddy}(*small domain*)	0.185 ms $^{-1}$
Outlet	Open (*wind1*, *wind2*, *wind3*, *wind4*)
Temperature BCs	zero fluxes

Boundary conditions	*fire simulations*
Lateral	Periodic (*fire1*, *fire2*, *fire5*)
	Open (*fire3*)
	Free-slip, no normal velocity (*fire4*)
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 fire2)
N_{eddy}	40 (for fire2)
σ_{eddy}(*large domain*)	1.0 ms $^{-1}$ if $z < 20$ m
	0.5 ms $^{-1}$ if $20 \leq z < 40$ m
	0 ms $^{-1}$ if $z \geq 40$ m
σ_{eddy}(*small domain*)	0.185 ms $^{-1}$
Outlet	Open
Temperature BCs	zero fluxes

(b)

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

(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).

$$\dot{q}''' = -\sum_{\alpha} \dot{m}'''_{\alpha} \triangle H_{f,\alpha} \tag{3.9}$$

where $\dot{m}'''_{\alpha}$ represents the lumped species mass production rates and $\triangle H_{f,\alpha}$ 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 [6] by Orloff and De Ris (1982), FDS imposes an upper bound using Equation(3.10).

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

where ∂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

[6] 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)

$$q''_{max} = HRRPUA\_SHEET/\partial x + HRRPUV\_AVERAGE \tag{3.11}$$

similar to FDS 6.2.0 McGrattan et al. (2015). This has been included in the *fire.f90* [1] 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^2 and that of *HRRPUV_AVERAGE* is 2500 KW/m^3. 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* [2] 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.

[1]The complete *fire.90* source code is in Appendix A (A.1)

[2]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 X 0.1m X 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-

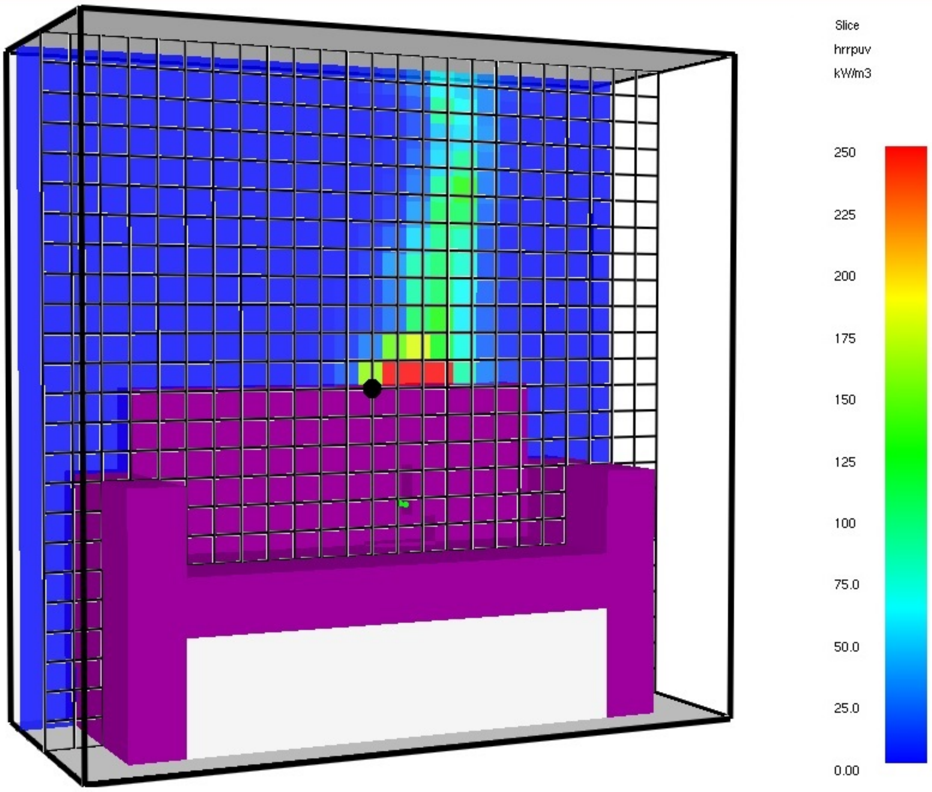


Figure 3.5: *Domain setup for the couch case*

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

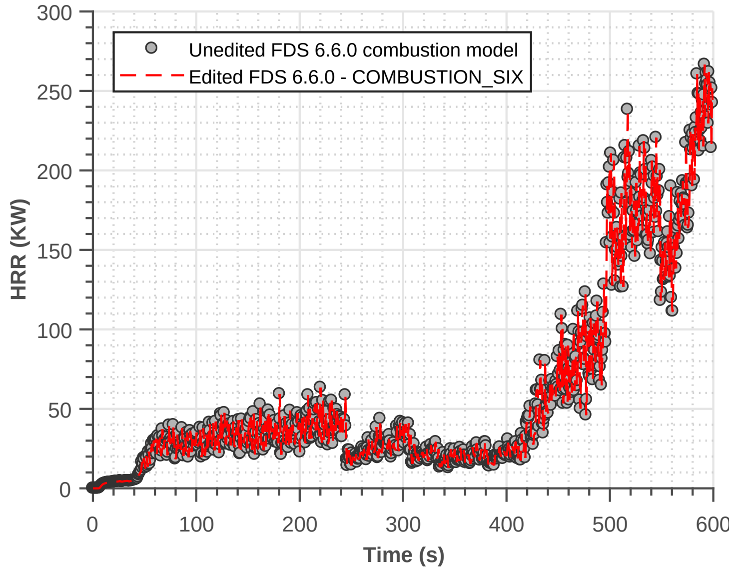


Figure 3.6: *Heat Release Rates comparison of in-built combustion model for edited and unedited versions of FDS 6.6.0 for couch case*

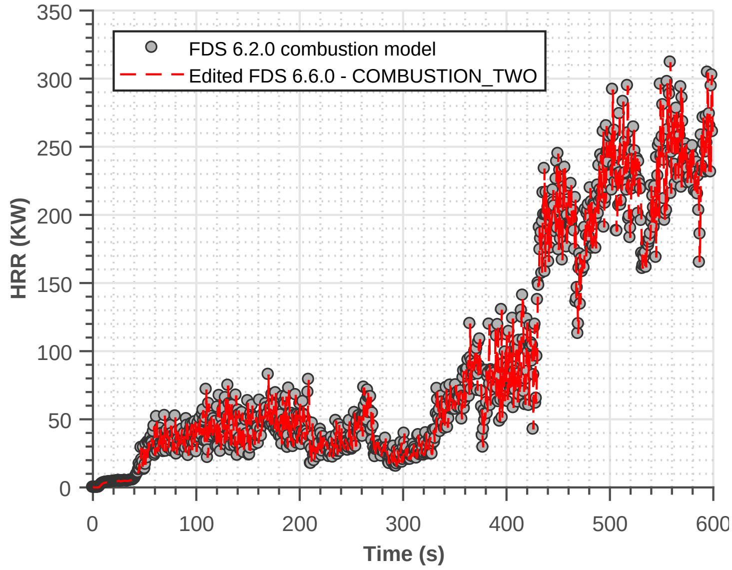


Figure 3.7: *Heat Release Rates comparison of FDS 6.2.0 combustion model and identical model introduced in FDS 6.6.0 for couch case*

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.

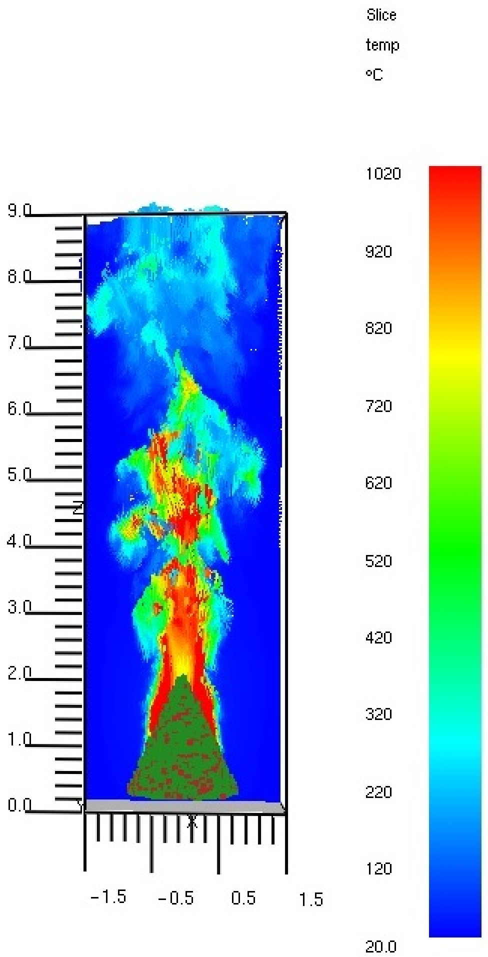


Figure 3.8: *Tree burning case: This figure shows the temperature contours of burning the tree represented by the green triangle. The red colour represents the maximum temperature (1020^{o}C),the fire plume region and the blue colour represents minimum temperature, the smoke region*

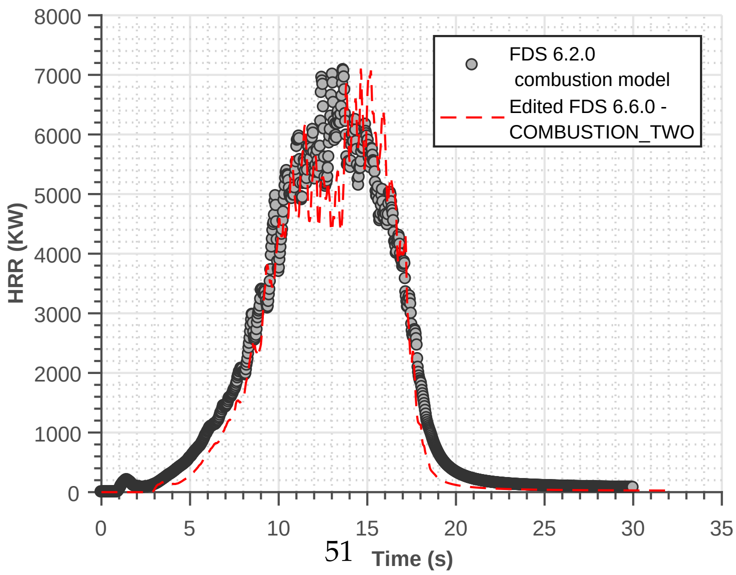


Figure 3.10: *Heat Release Rates comparison of FDS 6.2.0 combustion model and identical model introduced in FDS 6.6.0 for the tree burning case*

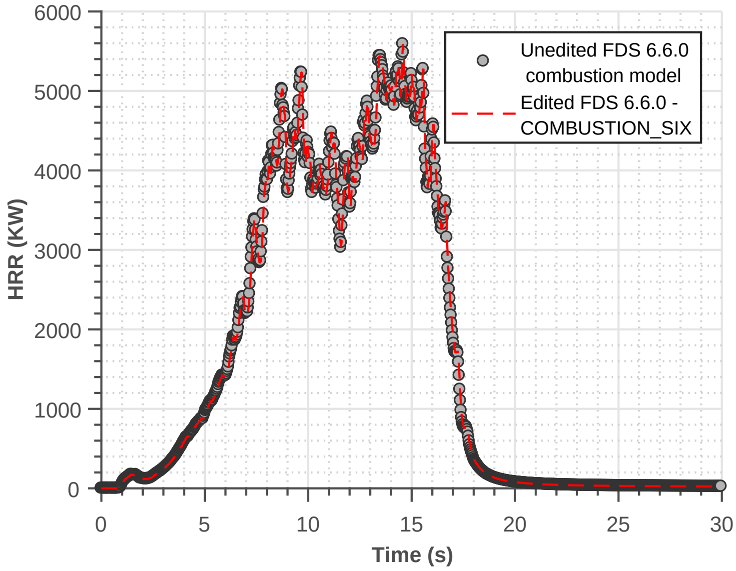


Figure 3.9: *Heat Release Rates comparison of in-built combustion model for edited and unedited versions of FDS 6.6.0 for the tree burning case*

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.

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 m $\times$ 40 m $\times$ 80 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 $z = 6$ 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 m $\times$ 40 m) is located at x=40 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*.

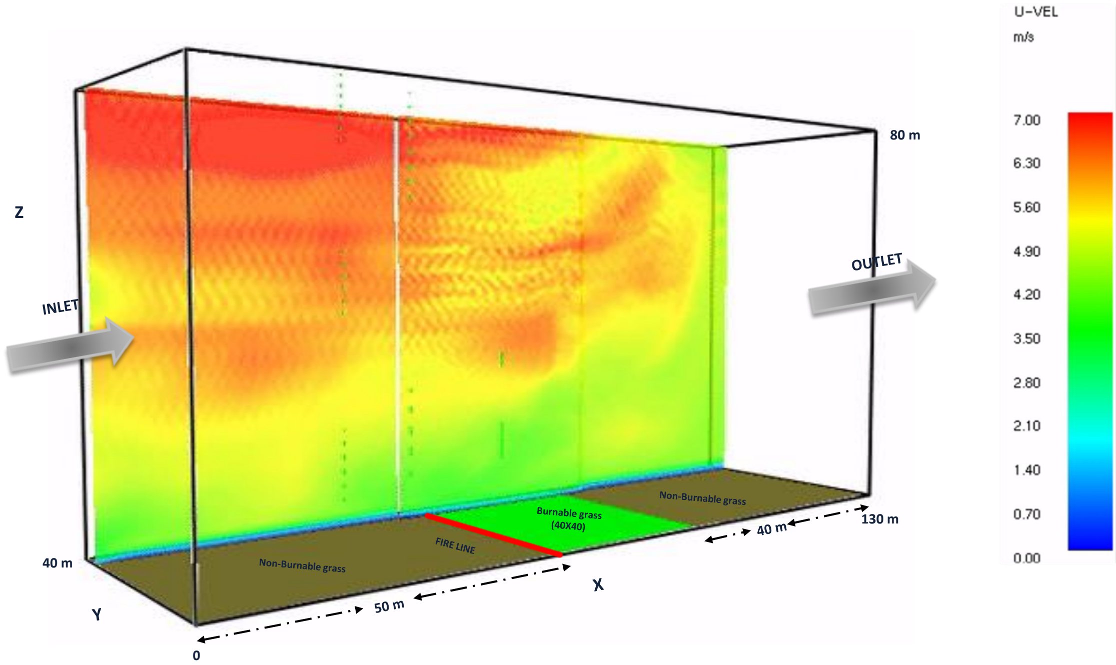


Figure 4.1: *A generalised schematic representation of Small Domain for simulation, representing the external dimensions, fire-line, fire plot and a slice of establishing ABL*

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.

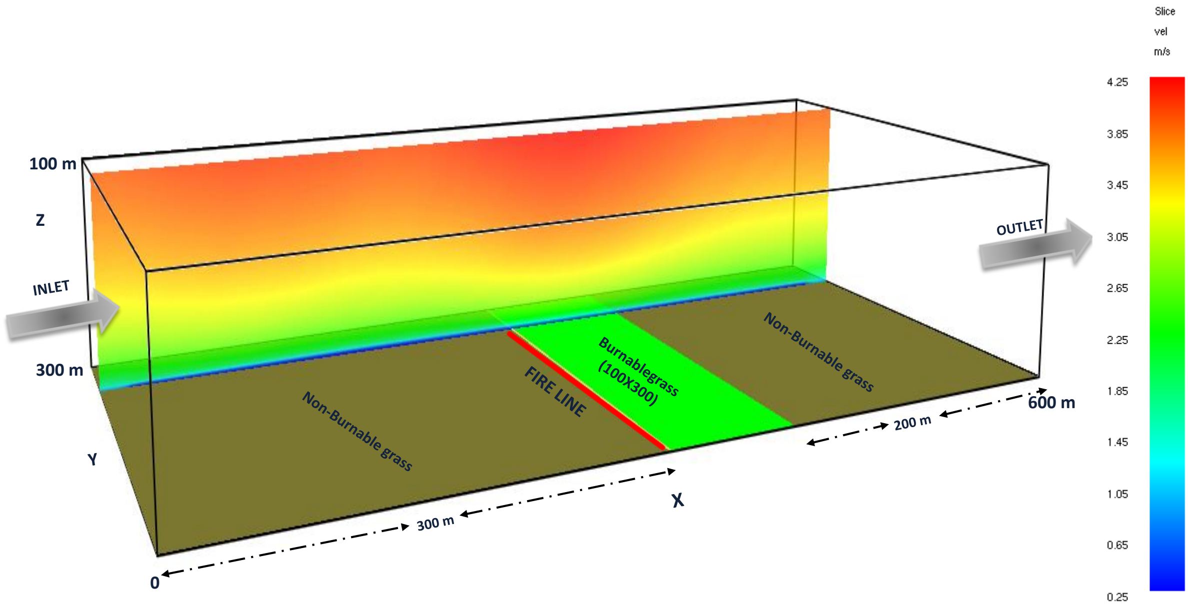


Figure 4.2: *A generalised schematic representation of the large domain for a simulation representing the external dimensions, fire-line, fire plot and a slice of establishing ABL*

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*.

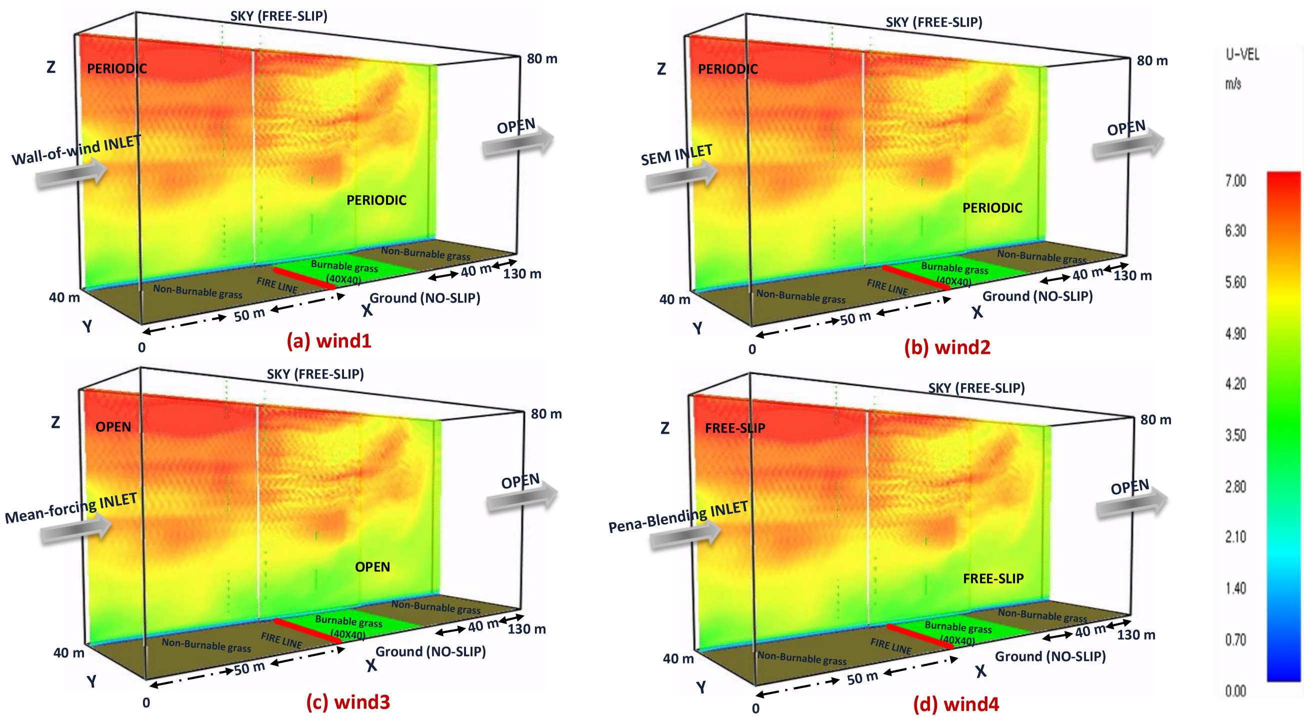


Figure 4.3: *A generalised schematic representation of the domain set-up with boundary conditions of small domain cases corresponding to Table-3.3-b for (a) wind1; (b) wind2; (c) wind3; (d) wind4, including the external dimensions, fire-line, fire plot and a slice of establishing ABL. The fire simulations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

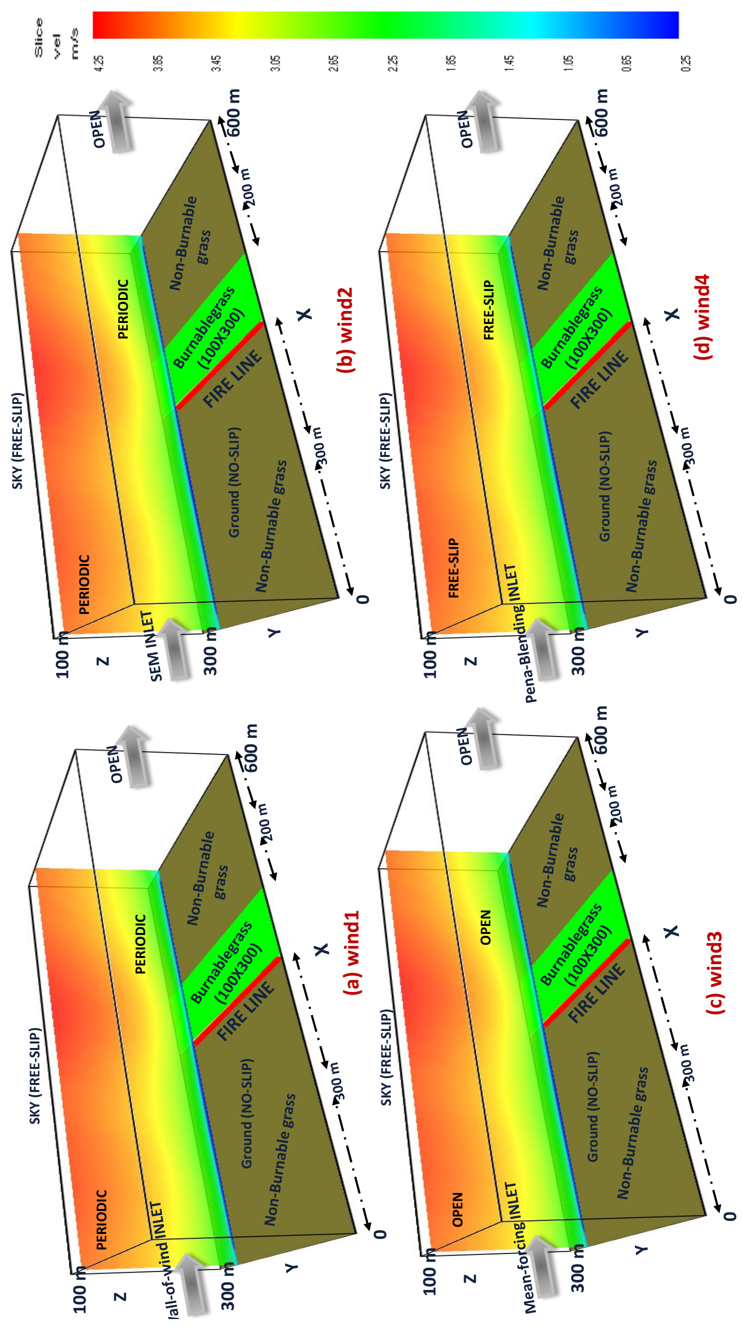


Figure 4.4: *A generalised schematic representation of the domain set-up with boundary conditions of large domain cases corresponding to Table-3.3-b for (a) wind1; (b) wind2; (c) wind3; (d) wind4, including the external dimensions, fire-line, fire plot and a slice of establishing ABL. The fire simulations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

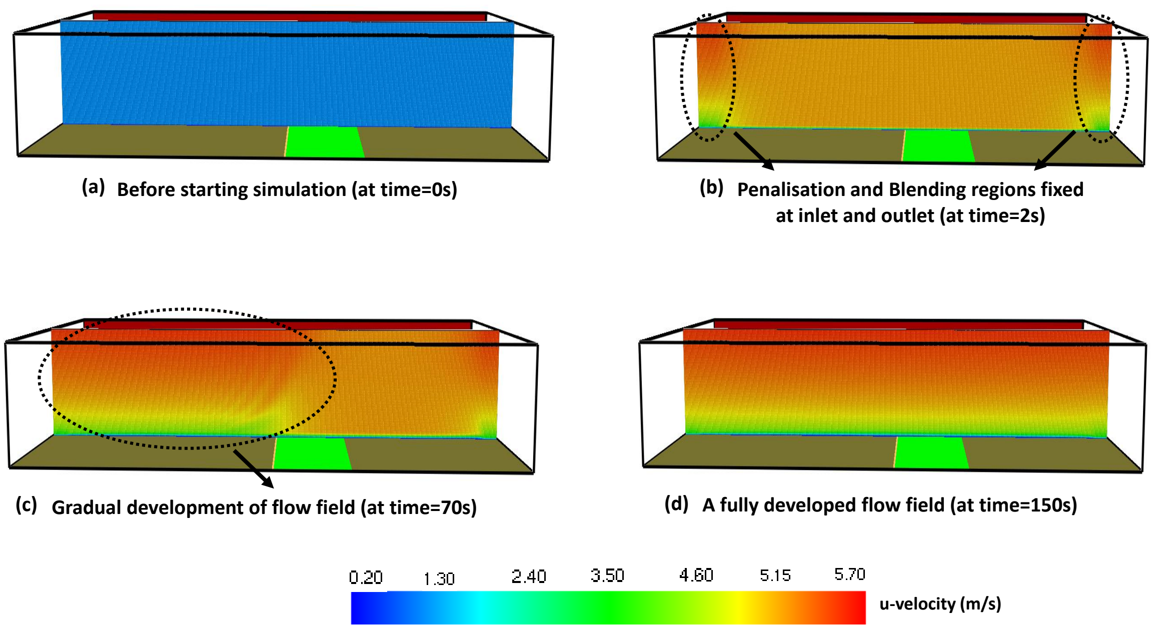


Figure 4.5: *Development of wind-field using the PenaBlending method for large domain*

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
Spin-up time :	$\sim 600 - 800$ s(wind1)	~ 1000 s(wind1)
	$\sim 400 - 500$ s(wind2)	~ 800 s(wind2)
	> 100 s(wind3)	> 150 s(wind3)
	$\sim 80 - 100$ s(wind4)	~ 200 s(wind4)
Total Simulation time :	3000 s	5000 s
Measurement time :	~ 1 s (varying)	~ 1 s (varying)
Time (improvement):	wind4 is $\sim$ 85% of wind1	wind4 is $\sim$ 80% of wind1
	wind4 is $\sim$ 80% of wind2	wind4 is $\sim$ 75% of wind2
	wind4 is similar to wind3	wind4 is similar to wind3
Accuracy(u_{10}) :	wind4 is $\sim$ 97% of wind1	wind4 is $\sim$ 96% of wind1
	wind4 is $\sim$ 100% of wind2	wind4 is $\sim$ 97% of wind2
	wind4 is $\sim$ 96.6% of wind3	wind4 is $\sim$ 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 [3] for *wind1* is

[3] 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 ~ 1000 s, *wind2* is ~ 800 s and *wind3* is less than150 s. In this case, the spin-up time for the *PenaBlending method* is ~ 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
Domain Dimension: 130m X 40m X 80m	**Domain Dimension:** 600m X 300m X 100m
penalisationParameter : 0.1	**penalisationParameter :** 0.1
blendingParameter : 1	**blendingParameter :** 1
penXmin : 0 (penalisation-inlet)	**penXmin :** 0 (penalisation-inlet)
6 (blending inlet)	10 (blending inlet)
113 (blending outlet)	580 (blending outlet)
119 (penalisation outlet)	590 (penalisation outlet)
penXmax : 6 (penalisation-inlet)	**penXmax :** 10 (penalisation-inlet)
12 (blending inlet)	20 (blending inlet)
119 (blending outlet)	590 (blending outlet)
125 (penalisation outlet)	600 (penalisation outlet)
penYmin : 0	**penYmin :** 0
penYmax : 40	**penYmax :** 300
penZmin : 0	**penZmin :** 0
penZmax : 80	**penZmax :** 100
mX : 0 (penalisation inlet and outlet)	**mX :** 0 (penalisation inlet and outlet)
-1 (blending inlet)	-1 (blending inlet)
1 (blending outlet)	1 (blending outlet)
mY : 0	**mY :** 0
mZ : 0	**mZ :** 0
b : 1 (penalisation inlet and outlet)	**b :** 1 (penalisation inlet and outlet)
0 (blending inlet and outlet)	0 (blending inlet and outlet)
pena_I : 6	**pena_I :** 5
pena_J : 40	**pena_J :** 150
pens_K : 80	**pens_K :** 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 m $\times$ 40 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 m $\times$ 300 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.

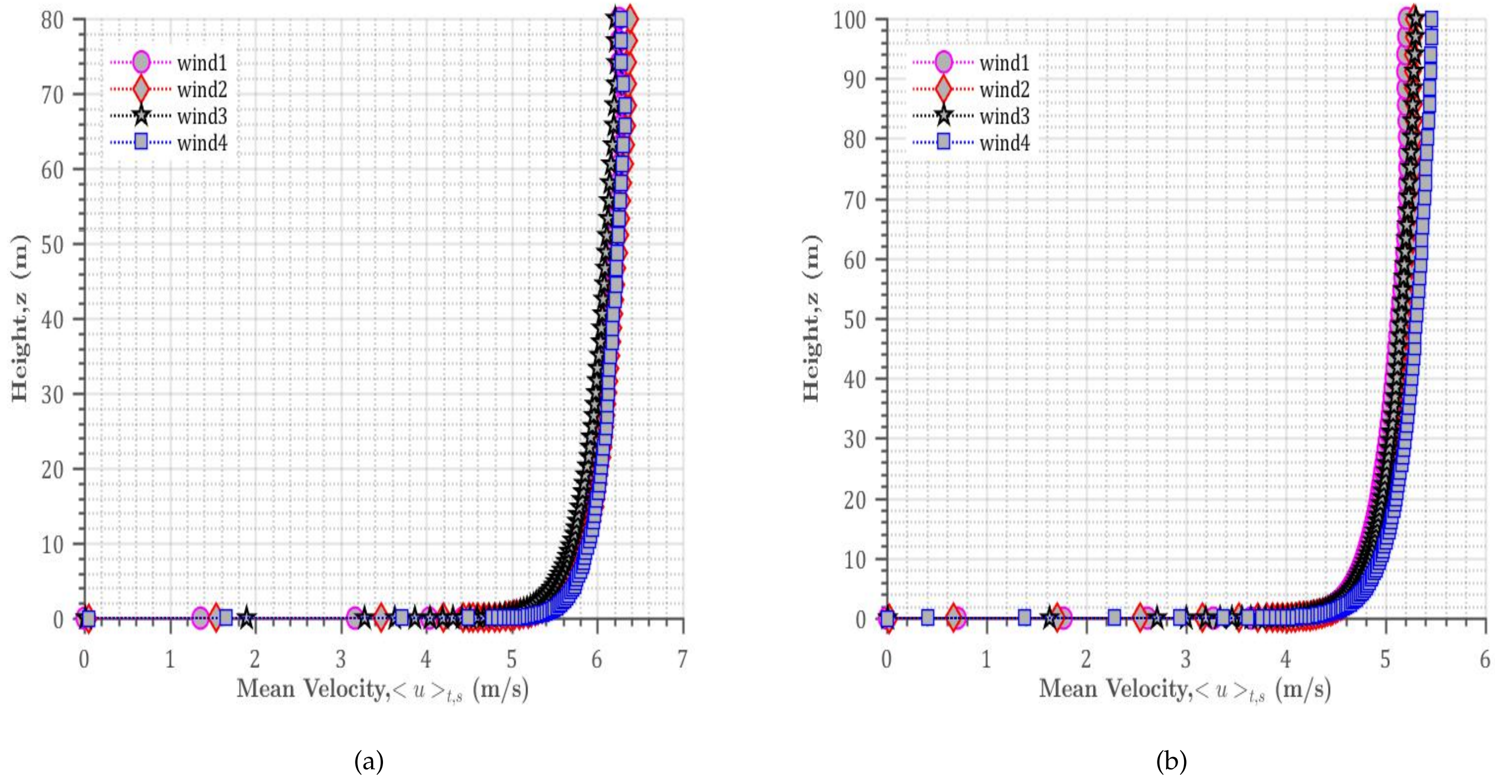
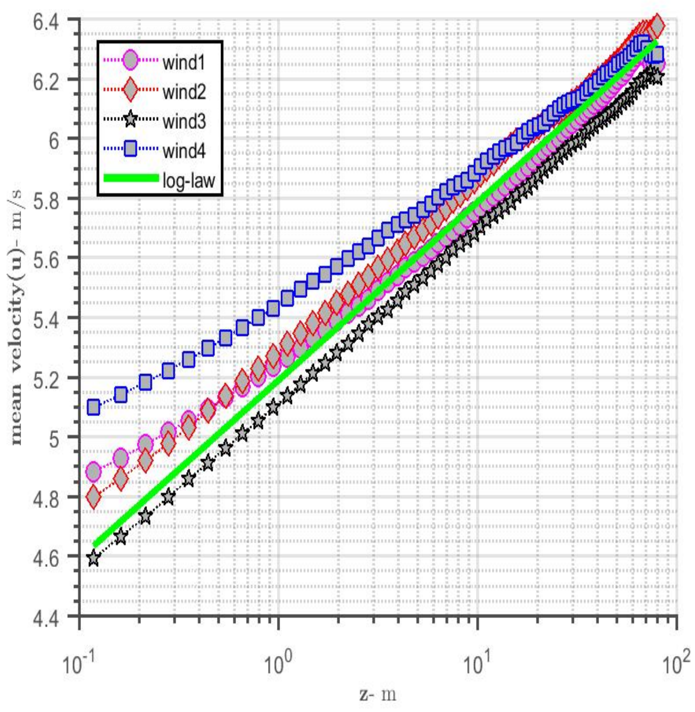
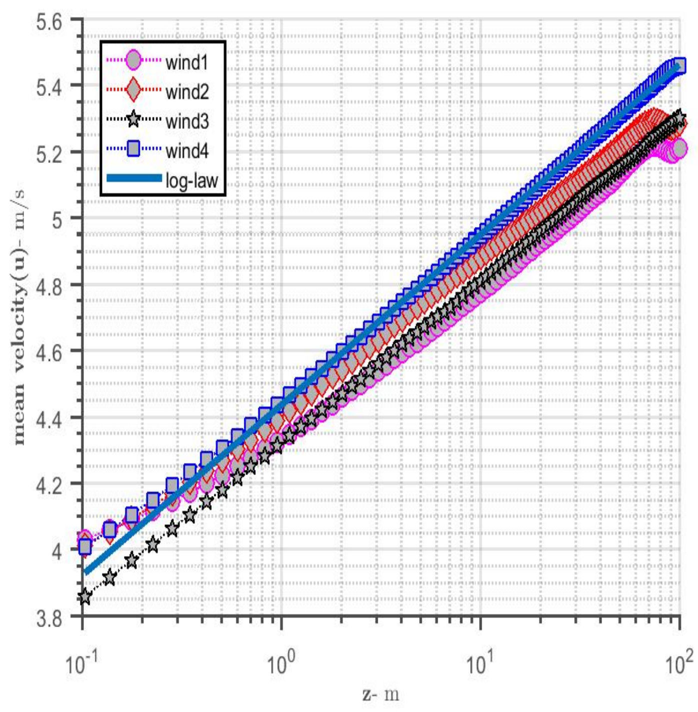


Figure 4.6: *The mean velocity profiles are plotted over the fire-ground for wall-of-wind method (wind1), SEM method (wind2), mean-forcing method (wind3) and the PenaBlending method (wind4) for: (a)Small domain of 130m X 40m X 80m ; and (b) Large Domain of 600m X 300m X 100m.*



(a)



(b)

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 m $\times$ 0.25 m is selected based on the grid convergence study of Moinuddin et al. (2018), upto a height of $z =$ 6 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 40m $\times$ 40m 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 $t = 0$ s, $t = 14$ s, $t = 20$ s and $t =$time at which fire reaches the end of the plot. For the *large domain*, the fire locations have been taken at $t = 0$ s, $t = 15$ s, $t = 30$ s and $t =$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 s and $\sim$ 38 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 $\sim$ 45s to reach the end of fire-plot. For the rest of the cases, the fire requires $\sim$ 24 s and $\sim$ 50 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.

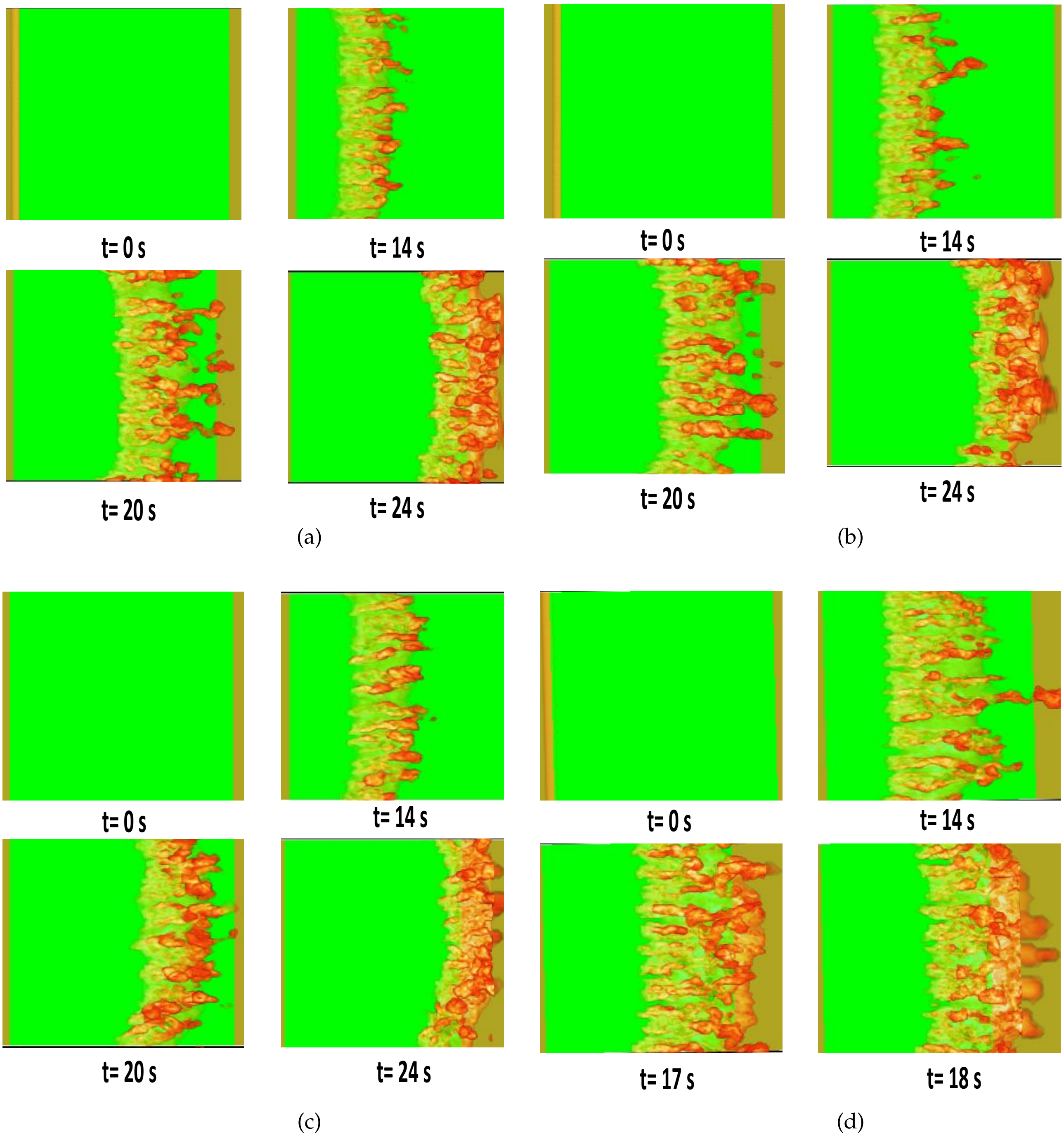


Figure 4.8: *The fire propagation contour for a small domain with a fire-plot of (40m X 40m) where the green area represents the 'burnable grass plot' and the non green area represents the 'non-burnable grassplot'. The propagation of fire is represented at various time-steps for (a) fire1; (b) fire2;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

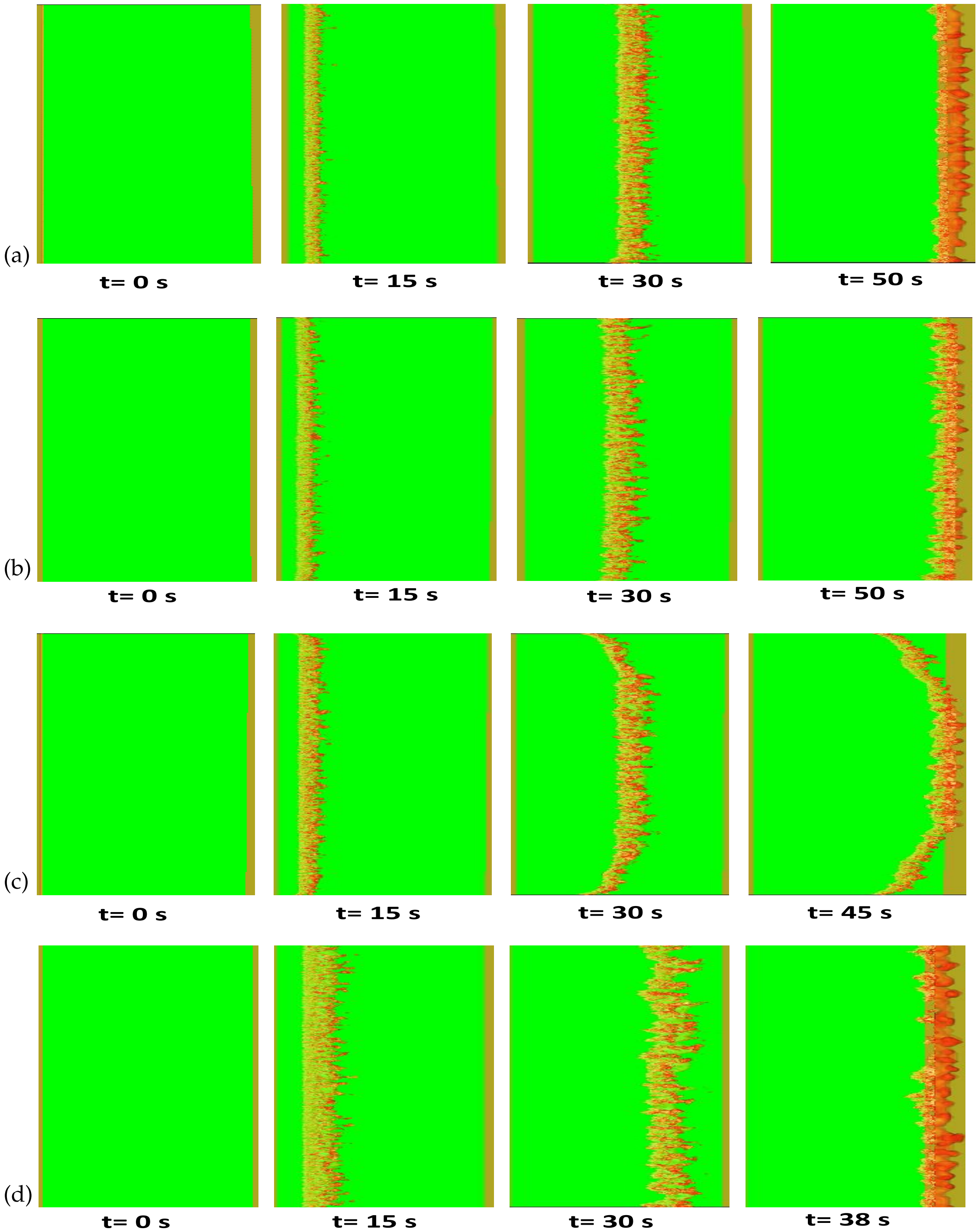


Figure 4.9: *The fire propagation contour for a large domain with fire plot of (100m X 300m) where the green area represents the 'burnable grass plot' and the non green area represents the 'non-burnable grassplot'. The propagation of fire is represented at various time-steps for (a)fire1; (b)fire2; (c)fire3; (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.:

$$RoS = \frac{dx_*}{dt} \tag{4.1}$$

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.

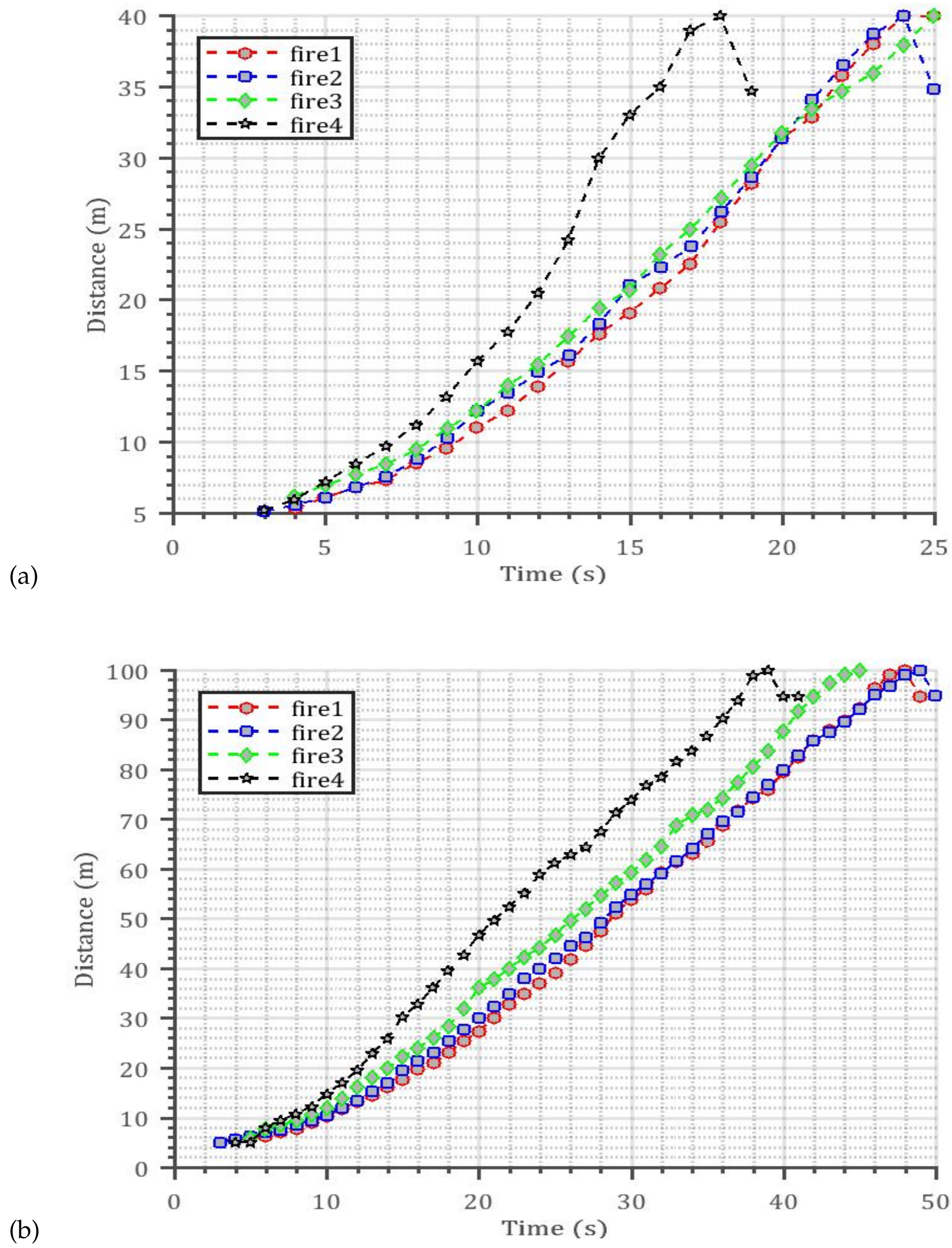


Figure 4.10: *The fire front (x_*) location in four fire cases (fire1; fire2; fire3; fire4 as a function of time for (a) small domain and (b) large domain, over the fire-plot.*

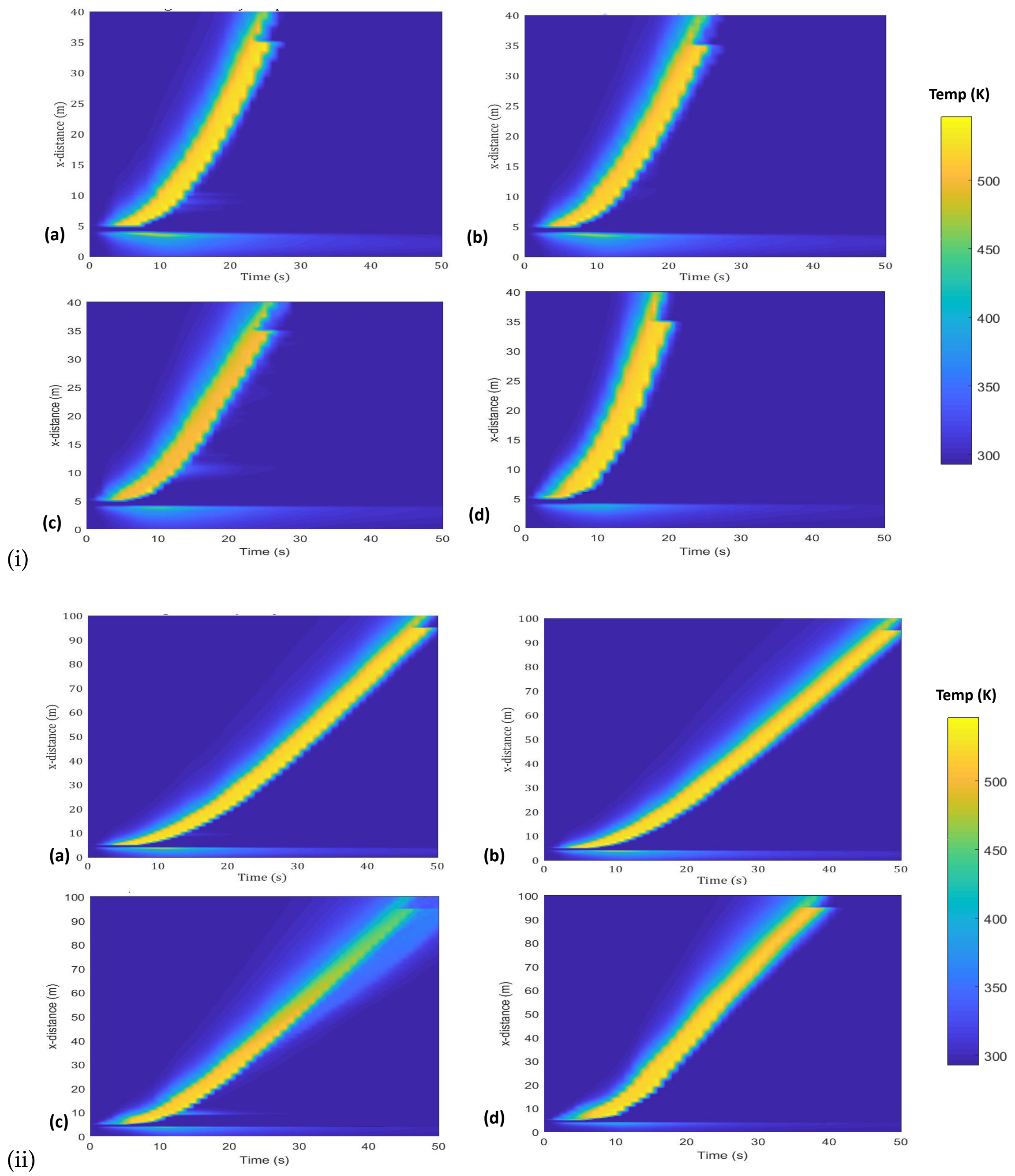
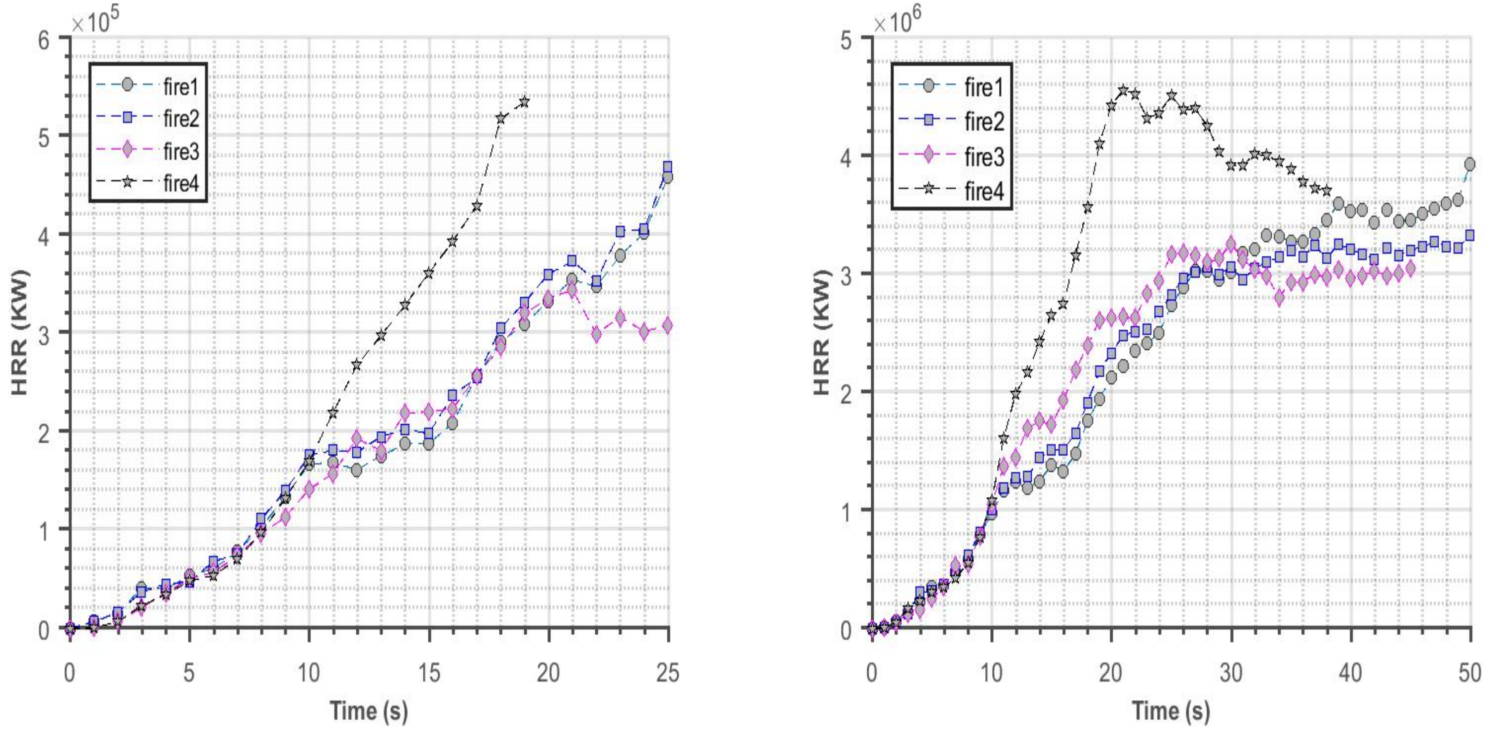


Figure 4.11: *The boundary temperature contours showing the fire-front propagation over the fire plot for all the fire simulation cases:(a) fire1; (b) fire2; (c) fire3; (d) fire4 for (i) small domain and (ii)large domain. The legend shows the temperature variation in K. The pyrolysis region is obtained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



(a)

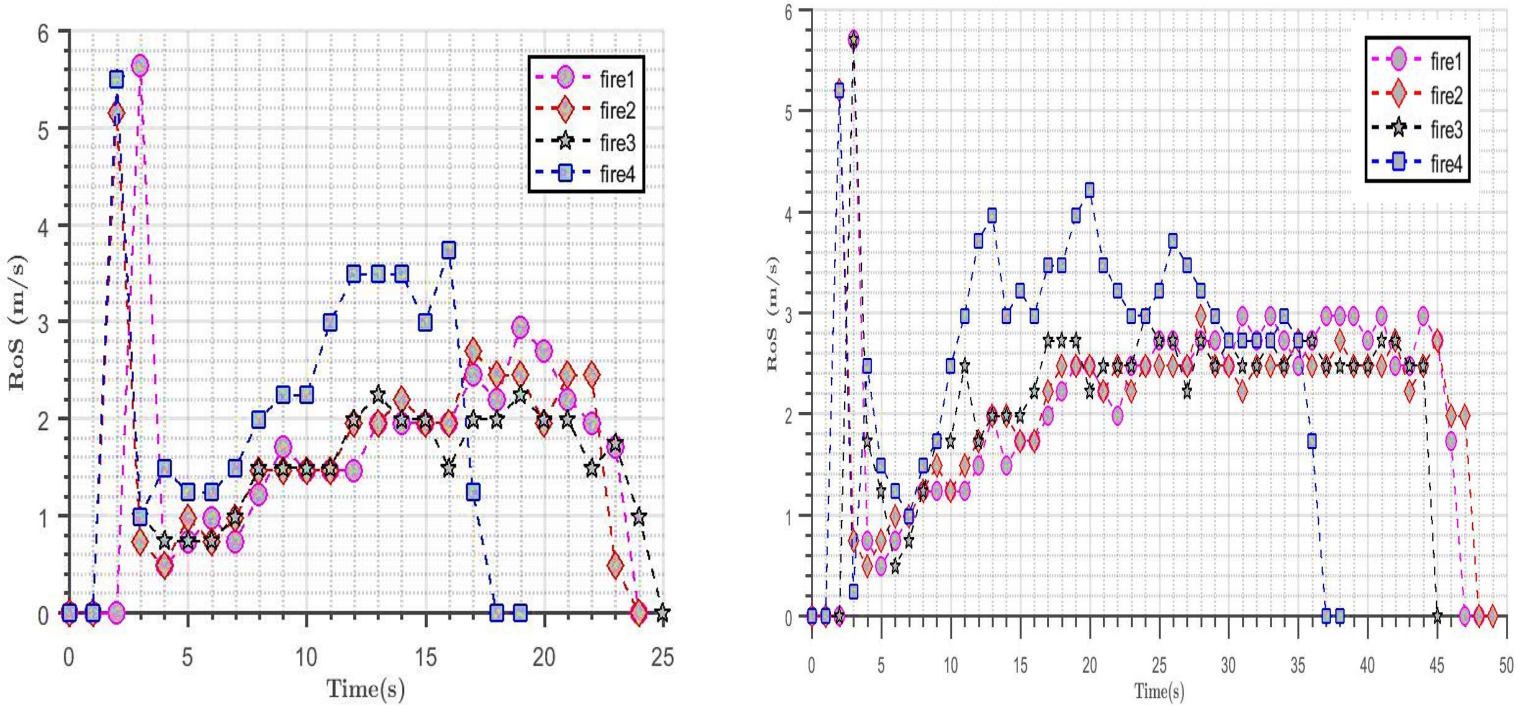
(b)

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 ~ 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 ~ 18 *seconds*. Similarly, considering the *large domain*, it is observed that *fire1* and *fire2* require ~ 50 *seconds* to burn all the fuel, whereas *fire3* requires ~ 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**
fire1 : $\sim$ 24 s	**fire1 :** $\sim$ 50 s
fire2 : $\sim$ 24 s	**fire2 :** $\sim$ 50 s
fire3 :$\sim$ 24 s	**fire3 :** $\sim$ 45 s
fire4 :$\sim$ 18 s	**fire4 :** $\sim$ 38 s

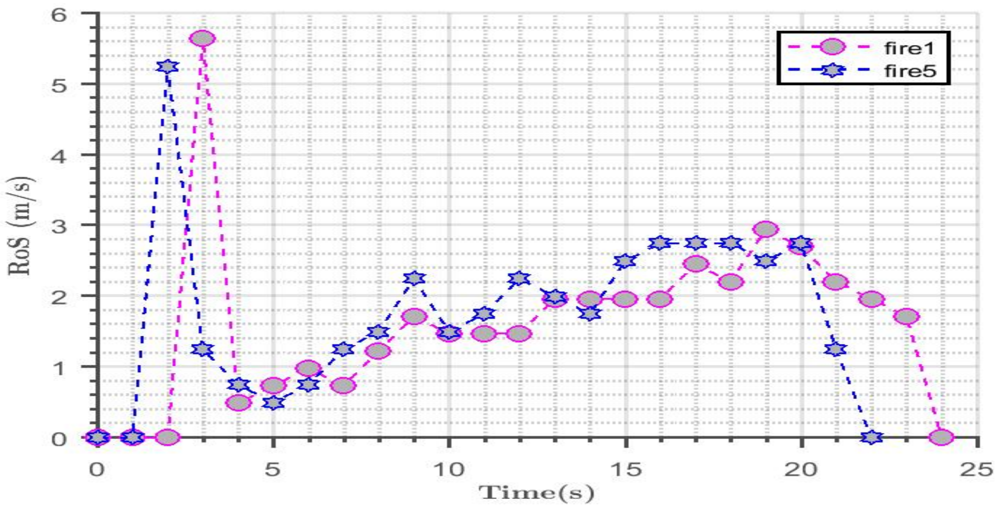


(a) (b)

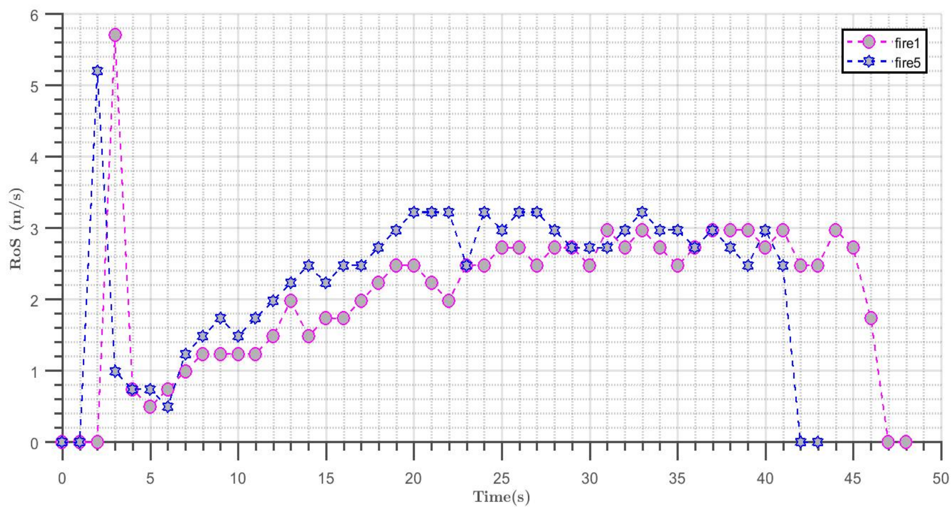
Figure 4.13: *The rate-of-spread of fire as a function of time using wall-of-wind method (fire1), SEM method (fire2), mean-forcing method (fire3) and the PenaBlending method (fire4) for: (a)Small domain of 130m X 40m X 80m ; and (b) Large Domain of 600m X 300m X 100m.*

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)

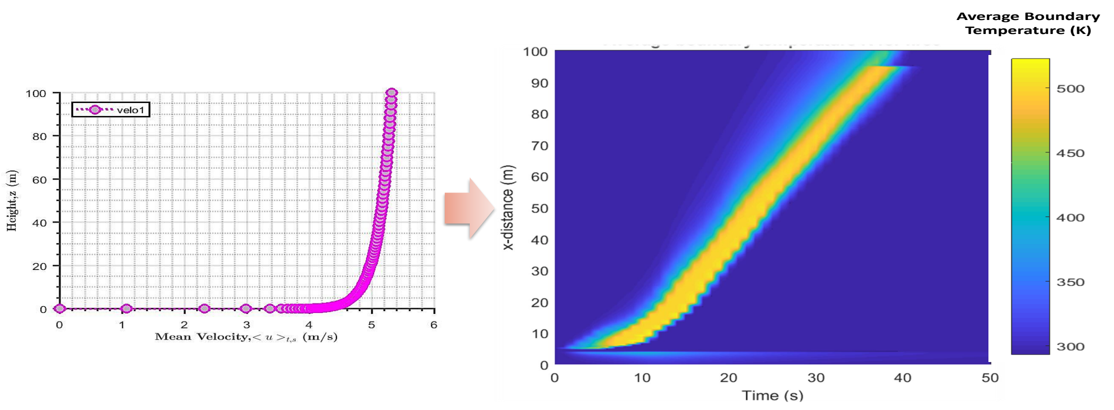


(b)

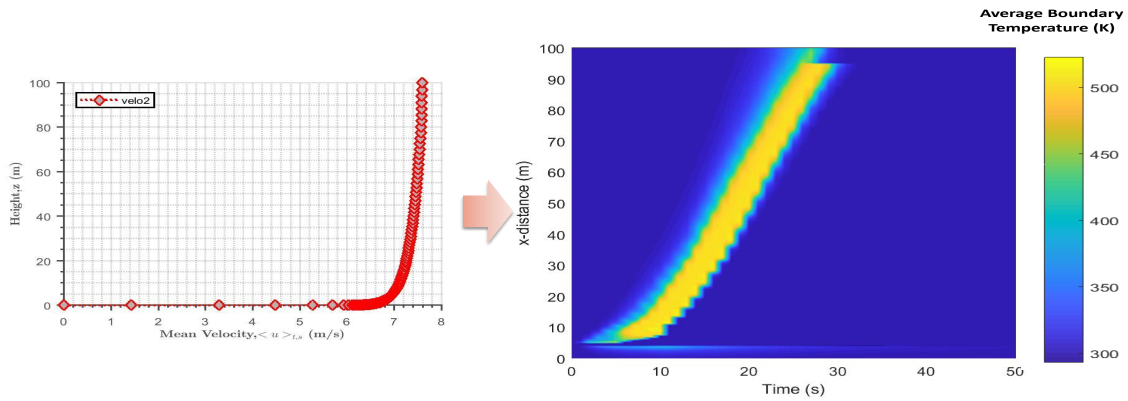
Figure 4.14: *The RoS comparison for underdeveloped-wind field for fire simulations (fire5) compared with the wall-of-wind method (fire1) for: (a) small domain and (b) large domain. In both the cases, all the conditions parameters used, boundary conditions and inlet method are same*

4.6 Gusting effect of wind on fire

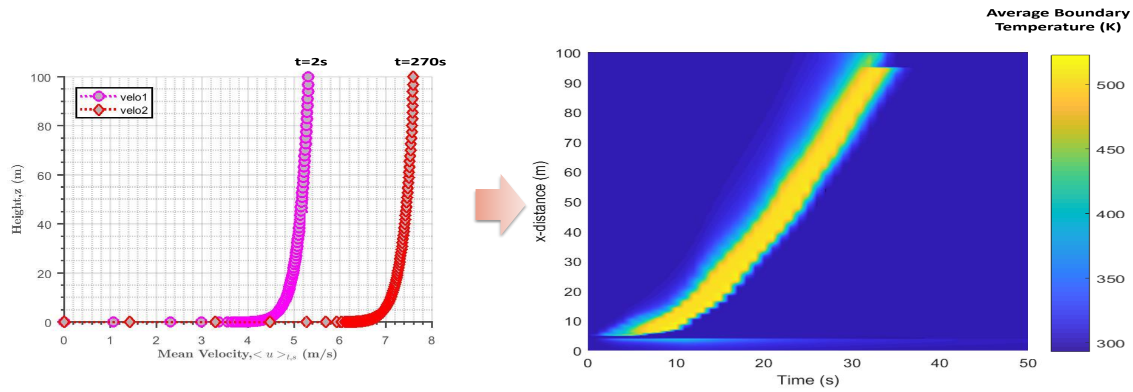
The *PenaBlending Method* provides the ability for FDS to read different velocity fields at various time-steps from any external model or data (called *pSim* data). This section validates this ability with three sets of simulations. To observe the effect of gusting winds on a fire simulation properly, 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.



(a)



(b)



(c)

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^{th} *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.

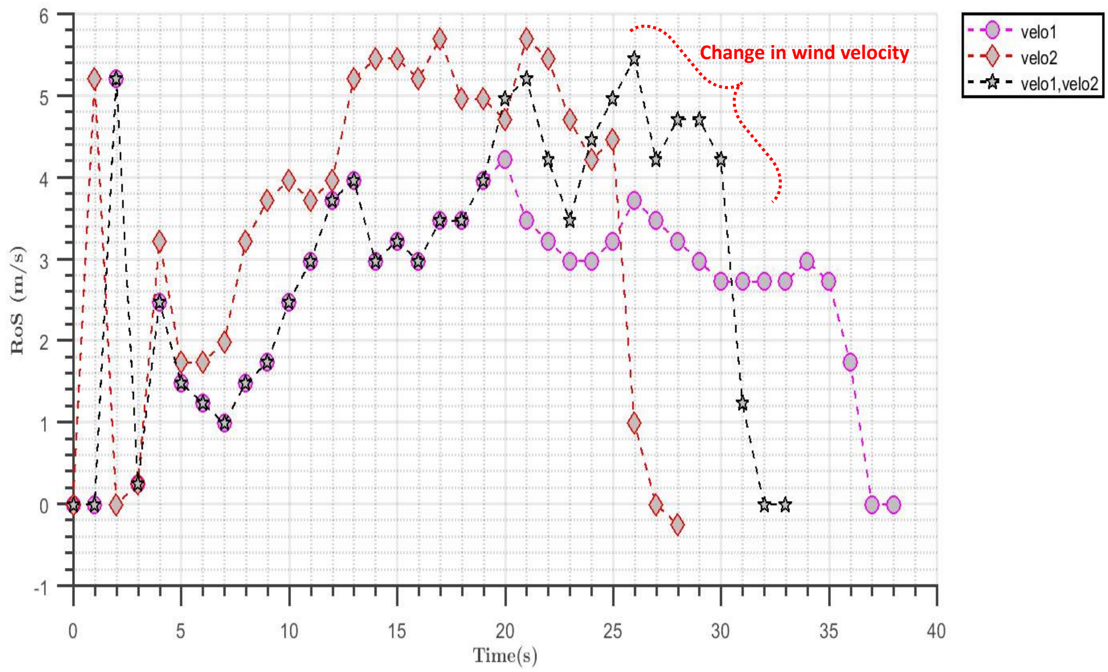


Figure 4.16: *Rate-of-spread comparison of fire simulations with velocity1(velo1), velocity2(velo2) and gusting effect of velo1 and velo2. The area marked with a red semi-circle shows how an increase in velocity during a fire propagation changes the RoS and hence blows the plume out of the domain faster.*

4.7 Modelling wind using reduced wind model to use with *PenaBlending 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).

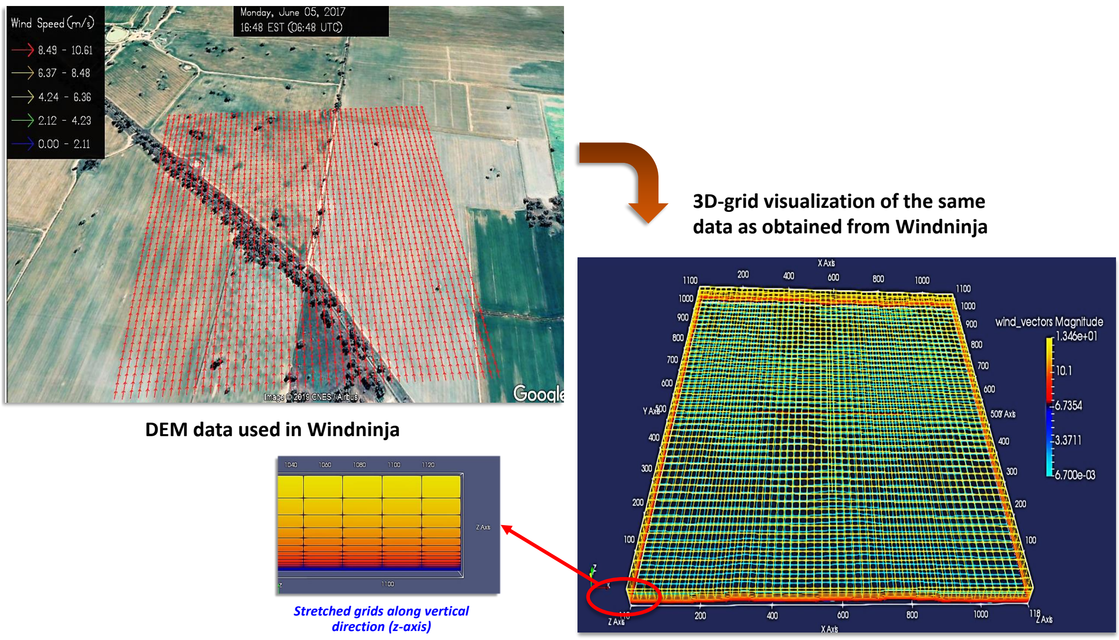


Figure 4.17: *The 3D grid representation of the simulated wind field in Windninja. The grid generated is parallel to the underlying terrain.*

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 reduced wind for *pSim* data. The modelling domain considered is a DEM (Digital Elevation Model) of an area of 1.17 km $\times$ 1.17 km with latitude and longitude of 35° 45′ South and 146°6′ East, near the northern boundary of Melbourne, Australia. An average speed of 10m/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 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 (1.17km $\times$ 1.17km) 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.

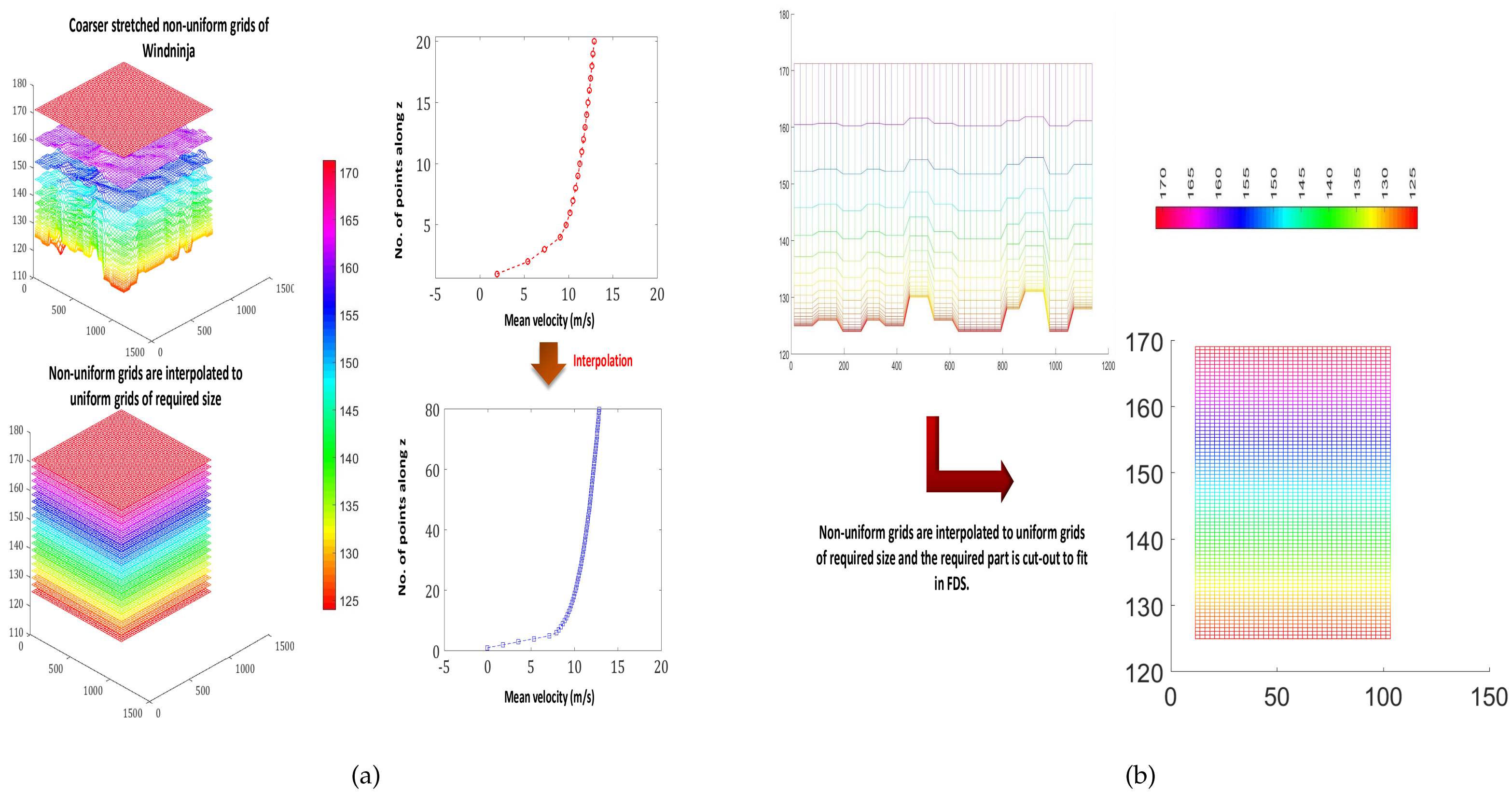


Figure 4.18: *(a) The original and interpolated uniform grids with required resolution similar to FDS domain. The mean velocity profiles for the original wind data and the interpolated data. The y-axis in the velocity profiles represents the grid points along z-axis; 20 grid points along z-axis have 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 Windninja with coarser non-uniform grid resolution and the cut-out uniform grid Windninja data with fine resolution of* $1\boldsymbol{m} \times 1\boldsymbol{m}$ *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.

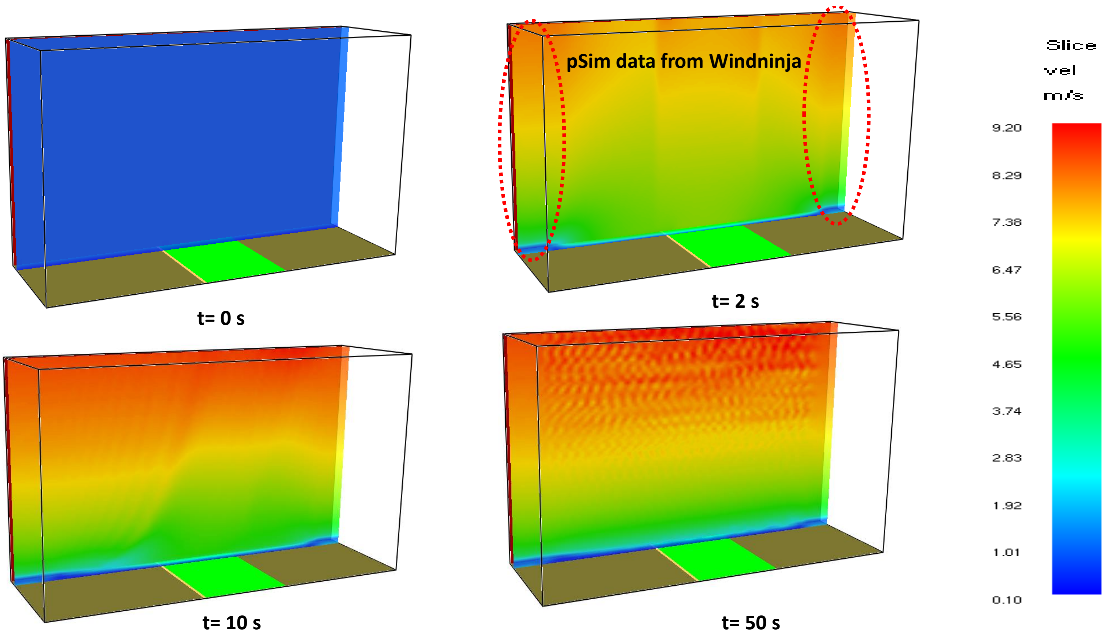


Figure 4.19: *Development of wind profile over the domain using Windninja data over various time-steps: (a) at time=0s; (b) at time= 2s, when the Windninja data is read at inlet/outlet; (c) at time=10s, depicting the wind developing from the inlet; (d) at time=50s, depicting a fully developed wind profile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

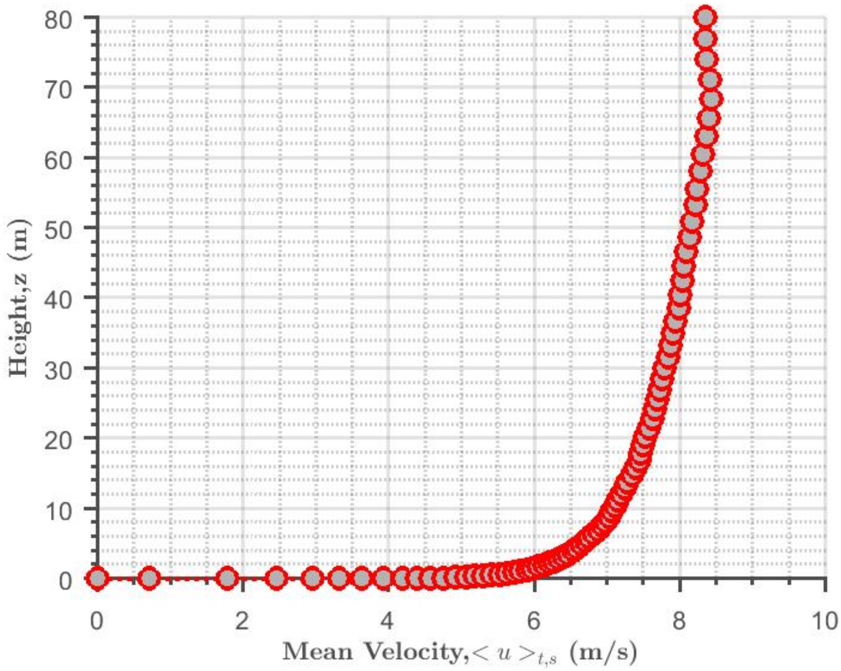


Figure 4.20: *Mean velocity profile over the fire-plot using Windninja data as pSim data for PenaBlending method.*

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 $\sim$ 18s. 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 $\sim$ 18s.

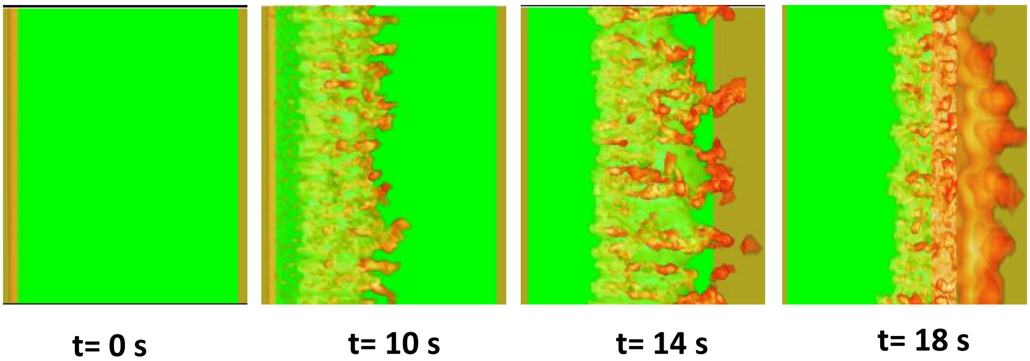


Figure 4.21: *The fire propagation over the fire plot using Windninja data at: time=0s; time=10s; time=14s; time=18s.*

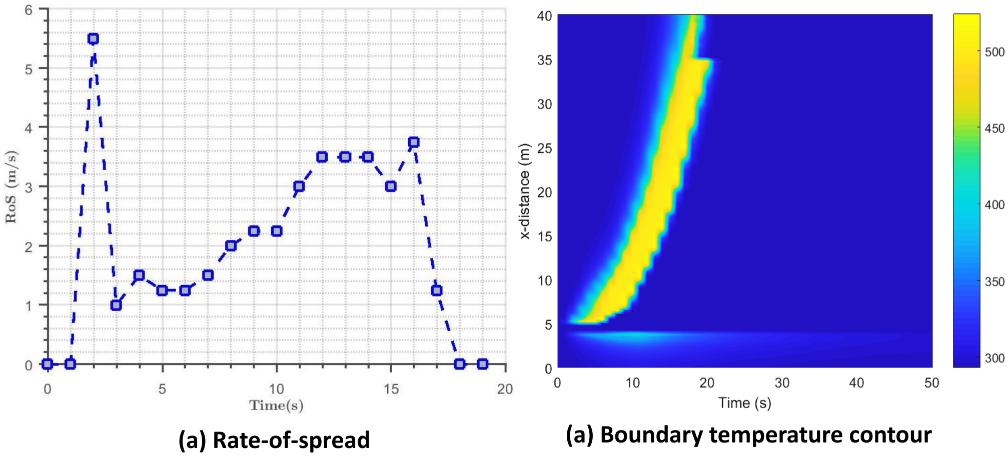


Figure 4.22: *(a) The RoS profile for fire over the fire plot using Windninja data; (b) The boundary temperature contour showing the fire front propagation over the fire plot*

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 0.25m $\times$ 0.25m $\times$ 0.25m 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 1.17km $\times$ 1.17km 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-quaternary 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., Wadhwani, 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.

Whittaker, J., McLennan, J., Elliott, G., Gilbert, J., Handmer, J., Haynes, K., and Cowlishaw, 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=18uQEmprdpmNDgBGIDVswu9ER_mHpN9Ho).

A.1 *fire.f90*

```
MODULE FIRE

! Compute combustion

USE PRECISION_PARAMETERS
USE GLOBAL_CONSTANTS
USE MESH_POINTERS
USE COMP_FUNCTIONS, ONLY: SECOND

IMPLICIT NONE
PRIVATE

TYPE(REACTION_TYPE), POINTER :: RN=>NULL()
REAL(EB) :: Q_UPPER
LOGICAL :: EXTINCT1 = .FALSE.        !changed name to EXTINCT1 for 6.2.0

INTEGER :: NM
REAL(EB) :: T,DT

PUBLIC GET_REV_fire
PUBLIC COMBUSTION

CONTAINS

SUBROUTINE COMBUSTION(T,DT,NM)

INTEGER, INTENT(IN) :: NM
REAL(EB), INTENT(IN) :: T,DT
REAL(EB) :: TNOW

IF (EVACUATION_ONLY(NM)) RETURN

TNOW=SECOND()

IF (INIT_HRRPUV) RETURN

CALL POINT_TO_MESH(NM)

SELECT_COMB: SELECT CASE (COMB_MODEL)

CASE (COMBUSTIONSIX) SELECT_COMB

CALL COMBUSTION_GENERAL(T,DT)  !combustion model for 6.6.0

CASE (COMBUSTIONTWO) SELECT_COMB

```

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

CALL COMBUSTION_GENERAL_1(T,DT)  !combustion model for 6.2.0

END SELECT SELECT_COMB

IF (CC_IBM) CALL CCREGION_COMBUSTION(T,DT,NM)

T_USED(10)=T_USED(10)+SECOND()-TNOW

END SUBROUTINE COMBUSTION

SUBROUTINE COMBUSTION_GENERAL_1(T,DT)

! Generic combustion routine for multi-step reactions

USE PHYSICAL_FUNCTIONS, ONLY: GET_SPECIFIC_GAS_CONSTANT,GET_MASS_FRACTION_ALL,GET_SPECIFIC_HEAT,
      GET_MOLECULAR_WEIGHT, &
GET_SENSIBLE_ENTHALPY_Z,IS_REALIZABLE,LES_FILTER_WIDTH_FUNCTION
USE COMPLEX_GEOMETRY, ONLY : IBM_CGSC, IBM_GASPHASE
INTEGER :: I,J,K,NS,NR,II,JJ,KK,IIG,JJG,KKG,IW,N,CHEM_SUBIT_TMP
REAL(EB), INTENT(IN) :: T,DT
REAL(EB) :: ZZ_GET(1:N_TRACKED_SPECIES),DZZ(1:N_TRACKED_SPECIES),CP,H_S_N,&
REAC_SOURCE_TERM_TMP(N_TRACKED_SPECIES),Q_REAC_TMP(N_REACTIONS)
LOGICAL :: Q_EXISTS
REAL(EB), POINTER, DIMENSION(:,:,:) :: ZETA_P=>NULL(),AIT_P=>NULL()
TYPE (REACTION_TYPE), POINTER :: RN
TYPE (SPECIES_MIXTURE_TYPE), POINTER :: SM
LOGICAL :: DO_REACTION,REALIZABLE

Q          = 0._EB
Q_EXISTS   = .FALSE.

CHI_R = 0._EB
IF (REAC_SOURCE_CHECK) Q_REAC=0._EB

IF (TRANSPORT_UNMIXED_FRACTION  .AND. &
COMPUTE_ZETA_SOURCE_TERM    .AND. &
TRANSPORT_ZETA_SCHEME==1         ) CALL ZETA_PRODUCTION(DT) ! scheme 1: zeta production before mixing

ZETA_P => WORK1
ZETA_P = 0._EB
IF (TRANSPORT_UNMIXED_FRACTION) ZETA_P = ZZ(:,:,:,ZETA_INDEX)

AIT_P => WORK2
AIT_P = 0._EB
IF (REIGNITION_MODEL) AIT_P = AIT

DO K=1,KBAR
DO J=1,JBAR
ILOOP: DO I=1,IBAR
! Check to see if a reaction is possible
IF (SOLID(CELL_INDEX(I,J,K))) CYCLE ILOOP
IF (CC_IBM) THEN
IF (CCVAR(I,J,K,IBM_CGSC) /= IBM_GASPHASE) CYCLE ILOOP
ENDIF
ZZ_GET = ZZ(I,J,K,1:N_TRACKED_SPECIES)
IF (CHECK_REALIZABILITY) THEN
REALIZABLE=IS_REALIZABLE(ZZ_GET)
IF (.NOT.REALIZABLE) THEN
WRITE(LU_ERR,*) I,J,K
WRITE(LU_ERR,*) ZZ_GET
WRITE(LU_ERR,*) SUM(ZZ_GET)
WRITE(LU_ERR,*) 'ERROR: Unrealizable mass fractions input to COMBUSTION_MODEL'
STOP_STATUS=REALIZABILITY_STOP
ENDIF
ENDIF
CALL CHECK_REACTION_1
IF (.NOT.DO_REACTION) CYCLE ILOOP ! Check whether any reactions are possible.
DZZ = ZZ_GET ! store old ZZ for divergence term
!***************************************************************************************
! Call combustion integration routine for Cartesian cell (I,J,K)
CALL COMBUSTION_MODEL_1(T,DT,ZZ_GET,Q(I,J,K),MIX_TIME(I,J,K),CHI_R(I,J,K),&
CHEM_SUBIT_TMP,REAC_SOURCE_TERM_TMP,Q_REAC_TMP,&
TMP(I,J,K),RHO(I,J,K),MU(I,J,K),KRES(I,J,K),&
ZETA_P(I,J,K),AIT_P(I,J,K),PBAR(K,PRESSURE_ZONE(I,J,K)),&
LES_FILTER_WIDTH_FUNCTION(DX(I),DY(J),DZ(K)),DX(I)*DY(J)*DZ(K) )
!***************************************************************************************
IF (OUTPUT_CHEM_IT) CHEM_SUBIT(I,J,K) = CHEM_SUBIT_TMP
IF (REAC_SOURCE_CHECK) THEN ! Store special diagnostic quantities
REAC_SOURCE_TERM(I,J,K,:) = REAC_SOURCE_TERM_TMP
Q_REAC(I,J,K,:) = Q_REAC_TMP
ENDIF
IF (CHECK_REALIZABILITY) THEN
REALIZABLE=IS_REALIZABLE(ZZ_GET)
IF (.NOT.REALIZABLE) THEN
WRITE(LU_ERR,*) ZZ_GET,SUM(ZZ_GET)
```

```
WRITE(LU_ERR,*) 'ERROR: Unrealizable mass fractions after COMBUSTION_MODEL'
STOP_STATUS=REALIZABILITY_STOP
ENDIF
ENDIF
DZZ = ZZ_GET - DZZ
! Update RSUM and ZZ
DZZ_IF: IF ( ANY(ABS(DZZ) > TWO_EPSILON_EB) ) THEN
IF (ABS(Q(I,J,K)) > TWO_EPSILON_EB) Q_EXISTS = .TRUE.
! Divergence term
CALL GET_SPECIFIC_HEAT(ZZ_GET,CP,TMP(I,J,K))
CALL GET_SPECIFIC_GAS_CONSTANT(ZZ_GET,RSUM(I,J,K))
DO N=1,N_TRACKED_SPECIES
SM => SPECIES_MIXTURE(N)
CALL GET_SENSIBLE_ENTHALPY_Z(N,TMP(I,J,K),H_S_N)
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
M_DOT_PPP(I,J,K,N) = M_DOT_PPP(I,J,K,N) + RHO(I,J,K)*DZZ(N)/DT
ENDDO
ENDIF DZZ_IF
ENDDO ILOOP
ENDDO
ENDDO

IF (TRANSPORT_UNMIXED_FRACTION) ZZ(:,:,:,ZETA_INDEX) = ZETA_P

IF (TRANSPORT_UNMIXED_FRACTION .AND. &
COMPUTE_ZETA_SOURCE_TERM     .AND. &
TRANSPORT_ZETA_SCHEME==2          ) CALL ZETA_PRODUCTION(DT) ! scheme 2: zeta production after mixing

IF (.NOT.Q_EXISTS) RETURN

! Set Q in the ghost cell, just for better visualization.

DO IW=1,N_EXTERNAL_WALL_CELLS
IF (WALL(IW)%BOUNDARY_TYPE/=INTERPOLATED_BOUNDARY .AND. WALL(IW)%BOUNDARY_TYPE/=OPEN_BOUNDARY) CYCLE
II  = WALL(IW)%ONE_D%II
JJ  = WALL(IW)%ONE_D%JJ
KK  = WALL(IW)%ONE_D%KK
IIG = WALL(IW)%ONE_D%IIG
JJG = WALL(IW)%ONE_D%JJG
KKG = WALL(IW)%ONE_D%KKG
Q(II,JJ,KK) = Q(IIG,JJG,KKG)
ENDDO

CONTAINS

SUBROUTINE CHECK_REACTION_1

! Check whether any reactions are possible.

LOGICAL :: REACTANTS_PRESENT

DO_REACTION = .FALSE.
REACTION_LOOP: DO NR=1,N_REACTIONS
RN=>REACTION(NR)
REACTANTS_PRESENT = .TRUE.
DO NS=1,N_TRACKED_SPECIES
IF ( RN%NU(NS) < -TWO_EPSILON_EB .AND. ZZ_GET(NS) < ZZ_MIN_GLOBAL ) THEN
REACTANTS_PRESENT = .FALSE.
EXIT
ENDIF
ENDDO
DO_REACTION = REACTANTS_PRESENT
IF (DO_REACTION) EXIT REACTION_LOOP
ENDDO REACTION_LOOP

END SUBROUTINE CHECK_REACTION_1

END SUBROUTINE COMBUSTION_GENERAL_1

SUBROUTINE COMBUSTION_GENERAL(T,DT)

! Generic combustion routine for multi-step reactions

USE PHYSICAL_FUNCTIONS, ONLY: GET_SPECIFIC_GAS_CONSTANT,GET_MASS_FRACTION_ALL,GET_SPECIFIC_HEAT,
       GET_MOLECULAR_WEIGHT, &
GET_SENSIBLE_ENTHALPY_Z,IS_REALIZABLE,LES_FILTER_WIDTH_FUNCTION
USE COMPLEX_GEOMETRY, ONLY : IBM_CGSC, IBM_GASPHASE
INTEGER :: I,J,K,NS,NR,II,JJ,KK,IIG,JJG,KKG,IW,N,CHEM_SUBIT_TMP
REAL(EB), INTENT(IN) :: T,DT
REAL(EB) :: ZZ_GET(1:N_TRACKED_SPECIES),DZZ(1:N_TRACKED_SPECIES),CP,H_S_N,&
REAC_SOURCE_TERM_TMP(N_TRACKED_SPECIES),Q_REAC_TMP(N_REACTIONS)
LOGICAL :: Q_EXISTS
REAL(EB), POINTER, DIMENSION(:,:,:) :: ZETA_P=>NULL(),AIT_P=>NULL()
TYPE (REACTION_TYPE), POINTER :: RN
TYPE (SPECIES_MIXTURE_TYPE), POINTER :: SM
LOGICAL :: DO_REACTION,REALIZABLE

Q          = 0._EB
```

```
Q_EXISTS    = .FALSE.

CHI_R = 0._EB
IF (REAC_SOURCE_CHECK) Q_REAC=0._EB

IF (TRANSPORT_UNMIXED_FRACTION .AND. &
COMPUTE_ZETA_SOURCE_TERM     .AND. &
TRANSPORT_ZETA_SCHEME==1            ) CALL ZETA_PRODUCTION(DT) ! scheme 1: zeta production before mixing

ZETA_P => WORK1
ZETA_P = 0._EB
IF (TRANSPORT_UNMIXED_FRACTION) ZETA_P = ZZ(:,:,:,ZETA_INDEX)

AIT_P => WORK2
AIT_P = 0._EB
IF (REIGNITION_MODEL) AIT_P = AIT

DO K=1,KBAR
DO J=1,JBAR
ILOOP: DO I=1,IBAR
! Check to see if a reaction is possible
IF (SOLID(CELL_INDEX(I,J,K))) CYCLE ILOOP
IF (CC_IBM) THEN
IF (CCVAR(I,J,K,IBM_CGSC) /= IBM_GASPHASE) CYCLE ILOOP
ENDIF
ZZ_GET = ZZ(I,J,K,1:N_TRACKED_SPECIES)
IF (CHECK_REALIZABILITY) THEN
REALIZABLE=IS_REALIZABLE(ZZ_GET)
IF (.NOT.REALIZABLE) THEN
WRITE(LU_ERR,*) I,J,K
WRITE(LU_ERR,*) ZZ_GET
WRITE(LU_ERR,*) SUM(ZZ_GET)
WRITE(LU_ERR,*) 'ERROR: Unrealizable mass fractions input to COMBUSTION_MODEL'
STOP_STATUS=REALIZABILITY_STOP
ENDIF
ENDIF
CALL CHECK_REACTION
IF (.NOT.DO_REACTION) CYCLE ILOOP ! Check whether any reactions are possible.
DZZ = ZZ_GET ! store old ZZ for divergence term
!***************************************************************************************
! Call combustion integration routine for Cartesian cell (I,J,K)
CALL COMBUSTION_MODEL( T,DT,ZZ_GET,Q(I,J,K),MIX_TIME(I,J,K),CHI_R(I,J,K),&
CHEM_SUBIT_TMP,REAC_SOURCE_TERM_TMP,Q_REAC_TMP,&
TMP(I,J,K),RHO(I,J,K),MU(I,J,K),KRES(I,J,K),&
ZETA_P(I,J,K),AIT_P(I,J,K),PBAR(K,PRESSURE_ZONE(I,J,K)),&
LES_FILTER_WIDTH_FUNCTION(DX(I),DY(J),DZ(K)),DX(I)*DY(J)*DZ(K) )
!***************************************************************************************
IF (OUTPUT_CHEM_IT) CHEM_SUBIT(I,J,K) = CHEM_SUBIT_TMP
IF (REAC_SOURCE_CHECK) THEN ! Store special diagnostic quantities
REAC_SOURCE_TERM(I,J,K,:) = REAC_SOURCE_TERM_TMP
Q_REAC(I,J,K,:) = Q_REAC_TMP
ENDIF
IF (CHECK_REALIZABILITY) THEN
REALIZABLE=IS_REALIZABLE(ZZ_GET)
IF (.NOT.REALIZABLE) THEN
WRITE(LU_ERR,*) ZZ_GET,SUM(ZZ_GET)
WRITE(LU_ERR,*) 'ERROR: Unrealizable mass fractions after COMBUSTION_MODEL'
STOP_STATUS=REALIZABILITY_STOP
ENDIF
ENDIF
DZZ = ZZ_GET - DZZ
! Update RSUM and ZZ
DZZ_IF: IF ( ANY(ABS(DZZ) > TWO_EPSILON_EB) ) THEN
IF (ABS(Q(I,J,K)) > TWO_EPSILON_EB) Q_EXISTS = .TRUE.
! Divergence term
CALL GET_SPECIFIC_HEAT(ZZ_GET,CP,TMP(I,J,K))
CALL GET_SPECIFIC_GAS_CONSTANT(ZZ_GET,RSUM(I,J,K))
DO N=1,N_TRACKED_SPECIES
SM => SPECIES_MIXTURE(N)
CALL GET_SENSIBLE_ENTHALPY_Z(N,TMP(I,J,K),H_S_N)
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
M_DOT_PPP(I,J,K,N) = M_DOT_PPP(I,J,K,N) + RHO(I,J,K)*DZZ(N)/DT
ENDDO
ENDIF DZZ_IF
ENDDO ILOOP
ENDDO
ENDDO

IF (TRANSPORT_UNMIXED_FRACTION) ZZ(:,:,:,ZETA_INDEX) = ZETA_P

IF (TRANSPORT_UNMIXED_FRACTION .AND. &
COMPUTE_ZETA_SOURCE_TERM     .AND. &
TRANSPORT_ZETA_SCHEME==2            ) CALL ZETA_PRODUCTION(DT) ! scheme 2: zeta production after mixing

IF (.NOT.Q_EXISTS) RETURN

! Set Q in the ghost cell, just for better visualization.

```

```fortran
DO IW=1,N_EXTERNAL_WALL_CELLS
IF (WALL(IW)%BOUNDARY_TYPE/=INTERPOLATED_BOUNDARY .AND. WALL(IW)%BOUNDARY_TYPE/=OPEN_BOUNDARY) CYCLE
II  = WALL(IW)%ONE_D%II
JJ  = WALL(IW)%ONE_D%JJ
KK  = WALL(IW)%ONE_D%KK
IIG = WALL(IW)%ONE_D%IIG
JJG = WALL(IW)%ONE_D%JJG
KKG = WALL(IW)%ONE_D%KKG
Q(II,JJ,KK) = Q(IIG,JJG,KKG)
ENDDO

CONTAINS

SUBROUTINE CHECK_REACTION

! Check whether any reactions are possible.

LOGICAL :: REACTANTS_PRESENT

DO_REACTION = .FALSE.
REACTION_LOOP: DO NR=1,N_REACTIONS
RN=>REACTION(NR)
REACTANTS_PRESENT = .TRUE.
DO NS=1,N_TRACKED_SPECIES
IF ( RN%NU(NS) < -TWO_EPSILON_EB .AND. ZZ_GET(NS) < ZZ_MIN_GLOBAL ) THEN
REACTANTS_PRESENT = .FALSE.
EXIT
ENDIF
ENDDO
DO_REACTION = REACTANTS_PRESENT
IF (DO_REACTION) EXIT REACTION_LOOP
ENDDO REACTION_LOOP

END SUBROUTINE CHECK_REACTION

END SUBROUTINE COMBUSTION_GENERAL

SUBROUTINE COMBUSTION_MODEL_1(T,DT,ZZ_GET,Q_OUT,MIX_TIME_OUT,CHI_R_OUT,CHEM_SUBIT_OUT,REAC_SOURCE_TERM_OUT,
    Q_REAC_OUT,&
TMP_IN,RHO_IN,MU_IN,KRES_IN,ZETA_INOUT,AIT_IN,PBAR_IN,DELTA,CELL_VOLUME)
USE COMP_FUNCTIONS, ONLY: SHUTDOWN
USE MATH_FUNCTIONS, ONLY: EVALUATE_RAMP
USE PHYSICAL_FUNCTIONS, ONLY: GET_AVERAGE_SPECIFIC_HEAT,GET_SPECIFIC_GAS_CONSTANT,GET_ENTHALPY
!INTEGER,INTENT(IN) :: I,J,K  !added asin 6.2.0
REAL(EB), INTENT(IN) :: T,DT,TMP_IN,RHO_IN,MU_IN,KRES_IN,PBAR_IN,AIT_IN,DELTA,CELL_VOLUME
REAL(EB), INTENT(OUT) :: Q_OUT,MIX_TIME_OUT,CHI_R_OUT,REAC_SOURCE_TERM_OUT(N_TRACKED_SPECIES),Q_REAC_OUT(
    N_REACTIONS)
INTEGER, INTENT(OUT) :: CHEM_SUBIT_OUT
REAL(EB), INTENT(INOUT) :: ZZ_GET(1:N_TRACKED_SPECIES),ZETA_INOUT
REAL(EB) :: ERR_EST,ERR_TOL,A1(1:N_TRACKED_SPECIES),A2(1:N_TRACKED_SPECIES),A4(1:N_TRACKED_SPECIES),ZETA,ZETA_0,&
DT_SUB,DT_SUB_NEW,DT_ITER,ZZ_STORE(1:N_TRACKED_SPECIES,1:4),TV(1:3,1:N_TRACKED_SPECIES),CELL_MASS,&
ZZ_DIFF(1:3,1:N_TRACKED_SPECIES),ZZ_MIXED(1:N_TRACKED_SPECIES),ZZ_UNMIXED(1:N_TRACKED_SPECIES),&
ZZ_MIXED_NEW(1:N_TRACKED_SPECIES),TAU_D,TAU_G,TAU_U,TAU_MIX,TMP_MIXED,TMP_UNMIXED,DT_SUB_MIN,RHO_HAT,&
PBAR_0,VEL_RMS,TAU_RES,ZZ_0(1:N_TRACKED_SPECIES),&
Q_REAC_SUB(1:N_REACTIONS),Q_REAC_1(1:N_REACTIONS),Q_REAC_2(1:N_REACTIONS),Q_REAC_4(1:N_REACTIONS),&
Q_REAC_SUM(1:N_REACTIONS),CHI_R_SUM,TIME_RAMP_FACTOR,&
TOTAL_MIXED_MASS_1,TOTAL_MIXED_MASS_2,TOTAL_MIXED_MASS_4,TOTAL_MIXED_MASS,&
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)
INTEGER :: NR,NS,ITER,TVI,RICH_ITER,TIME_ITER,RICH_ITER_MAX,CO_PASS,N_CO_PASS,SR  !added SR
INTEGER, PARAMETER :: TV_ITER_MIN=5
LOGICAL :: TV_FLUCT(1:N_TRACKED_SPECIES),EXTINCT
TYPE(REACTION_TYPE), POINTER :: RN=>NULL()
REAL(EB), PARAMETER :: C_U = 0.4_EB*0.1_EB*SQRT(1.5_EB) ! C_U*C_DEARDORFF/SQRT(2/3)

ZZ_TEMP = ZZ_GET  !added as in 6.2.0
ZZ_0 = ZZ_GET
EXTINCT = .FALSE.

VEL_RMS = 0._EB
IF (FIXED_MIX_TIME>0._EB) THEN
MIX_TIME_OUT=FIXED_MIX_TIME
ELSE
TAU_D=0._EB
DO NR =1,N_REACTIONS
RN => REACTION(NR)
TAU_D = MAX(TAU_D,D_Z(MIN(4999,NINT(TMP_IN)),RN%FUEL_SMIX_INDEX))
ENDDO
TAU_D = DELTA**2/MAX(TAU_D,TWO_EPSILON_EB)                    ! FDS Tech Guide (5.20)
IF (LES) THEN
TAU_U = C_U*RHO_IN*DELTA**2/MAX(MU_IN,TWO_EPSILON_EB)          ! FDS Tech Guide (5.24)
TAU_G = SQRT(2._EB*DELTA/(GRAV+1.E-10_EB))                     ! FDS Tech Guide (5.2)
MIX_TIME_OUT= MAX(TAU_CHEM,MIN(TAU_D,TAU_U,TAU_G,TAU_FLAME))   ! FDS Tech Guide (5.22)
VEL_RMS = SQRT(TWTH)*MU_IN/(RHO_IN*C_DEARDORFF*DELTA)
ELSE
MIX_TIME_OUT= MAX(TAU_CHEM,TAU_D)
ENDIF
ENDIF
```

```
IF (TRANSPORT_UNMIXED_FRACTION) THEN
ZETA_0 = ZETA_INOUT
ELSE
ZETA_0 = INITIAL_UNMIXED_FRACTION
ENDIF
CELL_MASS = RHO_IN*CELL_VOLUME
TAU_RES = MU_IN/(RHO_IN*SC)/MAX(2._EB*KRES_IN,TWO_EPSILON_EB)

IF (REIGNITION_MODEL) THEN
AIT_LOC = AIT_IN
ELSE
AIT_LOC = 1.E20_EB
ENDIF

DT_SUB_MIN = DT/REAL(MAX_CHEMISTRY_ITERATIONS,EB)

N_CO_PASS = 1
IF (EXTINCT_MOD==EXTINCTION_3) N_CO_PASS = 2

CO_EXTINCT_LOOP: DO CO_PASS = 1,N_CO_PASS

ZZ_STORE(:,:) = 0._EB
Q_OUT = 0._EB
Q_CUM = 0._EB    !added
Q_SUM = 0._EB    !added
ITER= 0
DT_ITER = 0._EB
CHI_R_OUT = 0._EB
CHEM_SUBIT_OUT = 0
REAC_SOURCE_TERM_OUT(:) = 0._EB
Q_REAC_OUT(:) = 0._EB
Q_REAC_SUM(:) = 0._EB
IF (N_FIXED_CHEMISTRY_SUBSTEPS>0) THEN
DT_SUB = DT/REAL(N_FIXED_CHEMISTRY_SUBSTEPS,EB)
DT_SUB_NEW = DT_SUB
RICH_ITER_MAX = 1
ELSE
DT_SUB = DT
DT_SUB_NEW = DT
RICH_ITER_MAX = 5
ENDIF
ZZ_UNMIXED = ZZ_GET
ZZ_MIXED = ZZ_GET
A1 = ZZ_GET
A2 = ZZ_GET
A4 = ZZ_GET

ZETA = ZETA_0
RHO_HAT = RHO_IN
TMP_MIXED = TMP_IN
TMP_UNMIXED = TMP_IN
TAU_MIX = MIX_TIME_OUT
PBAR_0 = PBAR_IN
added upto as in 6.6.0

INTEGRATION_LOOP: DO TIME_ITER = 1,MAX_CHEMISTRY_ITERATIONS

IF (SUPPRESSION) CALL CHECK_AUTO_IGNITION(EXTINCT,TMP_MIXED,AIT_LOC,CO_PASS)
IF (EXTINCT) EXIT INTEGRATION_LOOP

INTEGRATOR_SELECT: SELECT CASE (COMBUSTION_ODE_SOLVER)

CASE (EXPLICIT_EULER) ! Simple chemistry

! May be used with N_FIXED_CHEMISTRY_SUBSTEPS, but default mode is DT_SUB=DT for fast chemistry

CALL FIRE_FORWARD_EULER(ZZ_MIXED_NEW,ZZ_MIXED,ZZ_UNMIXED,ZETA,ZETA_0,DT_SUB,TMP_MIXED,TMP_UNMIXED,RHO_HAT,&
CELL_MASS,TAU_MIX,PBAR_0,DELTA,VEL_RMS,Q_REAC_SUB,TIME_ITER,TOTAL_MIXED_MASS,CO_PASS)
ZETA_0 = ZETA
ZZ_MIXED = ZZ_MIXED_NEW
IF (SIMPLE_CHEMISTRY .AND. N_FIXED_CHEMISTRY_SUBSTEPS<0 .AND. TIME_ITER>1) THEN
CALL SHUTDOWN('ERROR: Error in Simple Chemistry')
ENDIF

CASE (RK2) ! Runge-Kutta 2 stage (use in combination with N_FIXED_CHEMISTRY_SUBSTEPS)

CALL FIRE_RK2(ZZ_MIXED_NEW,ZZ_MIXED,ZZ_UNMIXED,ZETA,ZETA_0,DT_SUB,1,TMP_MIXED,TMP_UNMIXED,RHO_HAT,&
CELL_MASS,TAU_MIX,PBAR_0,DELTA,VEL_RMS,Q_REAC_SUB,TIME_ITER,TOTAL_MIXED_MASS,CO_PASS)
ZETA_0 = ZETA
ZZ_MIXED = ZZ_MIXED_NEW

CASE (RK3) ! Runge-Kutta 3 stage (use in combination with N_FIXED_CHEMISTRY_SUBSTEPS)

CALL FIRE_RK3(ZZ_MIXED_NEW,ZZ_MIXED,ZZ_UNMIXED,ZETA,ZETA_0,DT_SUB,1,TMP_MIXED,TMP_UNMIXED,RHO_HAT,&
CELL_MASS,TAU_MIX,PBAR_0,DELTA,VEL_RMS,Q_REAC_SUB,TIME_ITER,TOTAL_MIXED_MASS,CO_PASS)
ZETA_0 = ZETA
ZZ_MIXED = ZZ_MIXED_NEW
```

```

CASE (RK2_RICHARDSON) ! Finite-rate (or mixed finite-rate/fast) chemistry

! May be used with N_FIXED_CHEMISTRY_SUBSTEPS, but default mode is to use error estimator and variable DT_SUB

ERR_TOL = RICHARDSON_ERROR_TOLERANCE
RICH_EX_LOOP: DO RICH_ITER = 1,RICH_ITER_MAX

DT_SUB = MIN(DT_SUB_NEW,DT-DT_ITER)

! FDS Tech Guide (E.3), (E.4), (E.5)
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,
        PBAR_0,&
DELTA,VEL_RMS,Q_REAC_1,TIME_ITER,TOTAL_MIXED_MASS_1,CO_PASS)
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,
        PBAR_0,&
DELTA,VEL_RMS,Q_REAC_2,TIME_ITER,TOTAL_MIXED_MASS_2,CO_PASS)
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,
        PBAR_0,&
DELTA,VEL_RMS,Q_REAC_4,TIME_ITER,TOTAL_MIXED_MASS_4,CO_PASS)

! Species Error Analysis
ERR_EST = MAXVAL(ABS((4._EB*A4-5._EB*A2+A1)))/45._EB ! FDS Tech Guide (E.7)

IF (N_FIXED_CHEMISTRY_SUBSTEPS<0) THEN
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)
IF (ERR_EST<ERR_TOL) EXIT RICH_EX_LOOP
ENDIF

ENDDO RICH_EX_LOOP

ZZ_MIXED   = (4._EB*A4-A2)*ONTH ! FDS Tech Guide (E.6)
Q_REAC_SUB = (4._EB*Q_REAC_4-Q_REAC_2)*ONTH
ZETA       = (4._EB*ZETA_4-ZETA_2)*ONTH
ZETA_0     = ZETA


END SELECT INTEGRATOR_SELECT

ZZ_GET = ZETA*ZZ_UNMIXED + (1._EB-ZETA)*ZZ_MIXED ! FDS Tech Guide (5.29)
ZETA_INOUT = ZETA

DT_ITER = DT_ITER + DT_SUB
ITER = ITER + 1
IF (OUTPUT_CHEM_IT) CHEM_SUBIT_OUT = ITER

Q_REAC_SUM = Q_REAC_SUM + Q_REAC_SUB


! Compute heat release rate
print *, 'Calculating heat release rate according to FDS-6.2.0'
Q_SUM = 0._EB
IF (MAXVAL(ABS(ZZ_GET-ZZ_TEMP)) > TWO_EPSILON_EB) THEN
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
        RHO_IN as in 6.6.0
!print *, 'first if print Q_SUM,RHO_IN', Q_SUM, RHO_IN
ENDIF
IF (Q_CUM + Q_SUM > Q_UPPER*DT) THEN
Q_OUT = Q_UPPER
ZZ_GET = ZZ_TEMP + (Q_UPPER*DT/(Q_CUM + Q_SUM))*(ZZ_GET-ZZ_TEMP)
! print *, 'second print Q_OUT, ZZ_GET', Q_OUT, ZZ_GET
EXIT INTEGRATION_LOOP
ELSE
Q_CUM = Q_CUM+Q_SUM
Q_OUT = Q_CUM/DT
! print *, 'last else Q_CUM, Q_OUT' , Q_CUM , Q_OUT
ENDIF


! Total Variation (TV) scheme (accelerates integration for finite-rate equilibrium calculations)
! See FDS Tech Guide Appendix E

IF (COMBUSTION_ODE_SOLVER==RK2_RICHARDSON .AND. N_REACTIONS>1) THEN
DO NS = 1,N_TRACKED_SPECIES
DO TVI = 1,3
ZZ_STORE(NS,TVI)=ZZ_STORE(NS,TVI+1)
ENDDO
ZZ_STORE(NS,4) = ZZ_GET(NS)
ENDDO
TV_FLUCT(:) = .FALSE.
IF (ITER >= TV_ITER_MIN) THEN
SPECIES_LOOP_TV: DO NS = 1,N_TRACKED_SPECIES
DO TVI = 1,3
TV(TVI,NS) = ABS(ZZ_STORE(NS,TVI+1)-ZZ_STORE(NS,TVI))
ZZ_DIFF(TVI,NS) = ZZ_STORE(NS,TVI+1)-ZZ_STORE(NS,TVI)
ENDDO
IF (SUM(TV(:,NS)) < ERR_TOL .OR. SUM(TV(:,NS)) >= ABS(2.9_EB*SUM(ZZ_DIFF(:,NS)))) THEN ! FDS Tech Guide (E.10)
TV_FLUCT(NS) = .TRUE.
```

```fortran
ENDIF
IF (ALL(TV_FLUCT)) EXIT INTEGRATION_LOOP
ENDDO SPECIES_LOOP_TV
ENDIF
ENDIF

IF ( DT_ITER > (DT+TWO_EPSILON_EB) ) CALL SHUTDOWN('ERROR: DT_ITER > DT in COMBUSTION_MODEL')
IF ( DT_ITER > (DT-TWO_EPSILON_EB) ) EXIT INTEGRATION_LOOP

ENDDO INTEGRATION_LOOP

!added as in 6.6.0
! Extinction model

IF (SUPPRESSION) THEN
SELECT CASE(EXTINCT_MOD)
CASE(EXTINCTION_1); CALL EXTINCT_1(EXTINCT,ZZ_0,TMP_IN)
CASE(EXTINCTION_2); CALL EXTINCT_2(EXTINCT,ZZ_0,ZZ_MIXED,TMP_IN)
CASE(EXTINCTION_3); CALL EXTINCT_3(EXTINCT,ZZ_0,ZZ_MIXED,TMP_IN,CO_PASS)
END SELECT
ENDIF

IF (.NOT.EXTINCT) THEN
EXIT CO_EXTINCT_LOOP
ELSE
ZZ_GET = ZZ_0
ZZ_STORE(:,:) = 0._EB
Q_OUT = 0._EB
CHI_R_OUT = 0._EB
CHEM_SUBIT_OUT = 0
REAC_SOURCE_TERM_OUT(:) = 0._EB
Q_REAC_OUT(:) = 0._EB
Q_REAC_SUM(:) = 0._EB
ENDIF

ENDDO CO_EXTINCT_LOOP

!IF (REAC_SOURCE_CHECK) REAC_SOURCE_TERM(I,J,K,:) = (ZZ_UNMIXED-ZZ_GET)*CELL_MASS/DT ! store special output
        quantity
! Reaction rate-weighted radiative fraction

IF (SUM(Q_REAC_SUM)>TWO_EPSILON_EB) THEN
CHI_R_SUM=0._EB
DO NR=1,N_REACTIONS
RN=>REACTION(NR)
TIME_RAMP_FACTOR = EVALUATE_RAMP(T,0._EB,RN%RAMP_CHI_R_INDEX)
CHI_R_SUM = CHI_R_SUM + Q_REAC_SUM(NR)*RN%CHI_R*TIME_RAMP_FACTOR
ENDDO
CHI_R_OUT = CHI_R_SUM/(SUM(Q_REAC_SUM))
ENDIF
CHI_R_OUT = MAX(CHI_R_MIN,MIN(CHI_R_MAX,CHI_R_OUT))

! Store special diagnostic quantities

IF (REAC_SOURCE_CHECK) THEN
REAC_SOURCE_TERM_OUT = RHO_IN*(ZZ_GET-ZZ_0)/DT
Q_REAC_OUT = Q_REAC_SUM/CELL_VOLUME/DT
ENDIF
add complete as in 6.6.0
END SUBROUTINE COMBUSTION_MODEL_1    !changing names

SUBROUTINE COMBUSTION_MODEL(T,DT,ZZ_GET,Q_OUT,MIX_TIME_OUT,CHI_R_OUT,CHEM_SUBIT_OUT,REAC_SOURCE_TERM_OUT,
        Q_REAC_OUT,&
TMP_IN,RHO_IN,MU_IN,KRES_IN,ZETA_INOUT,AIT_IN,PBAR_IN,DELTA,CELL_VOLUME)
USE COMP_FUNCTIONS, ONLY: SHUTDOWN
USE MATH_FUNCTIONS, ONLY: EVALUATE_RAMP
USE PHYSICAL_FUNCTIONS, ONLY: GET_AVERAGE_SPECIFIC_HEAT,GET_SPECIFIC_GAS_CONSTANT,GET_ENTHALPY
REAL(EB), INTENT(IN) :: T,DT,TMP_IN,RHO_IN,MU_IN,KRES_IN,PBAR_IN,AIT_IN,DELTA,CELL_VOLUME
REAL(EB), INTENT(OUT) :: Q_OUT,MIX_TIME_OUT,CHI_R_OUT,REAC_SOURCE_TERM_OUT(N_TRACKED_SPECIES),Q_REAC_OUT(
        N_REACTIONS)
INTEGER, INTENT(OUT) :: CHEM_SUBIT_OUT
REAL(EB), INTENT(INOUT) :: ZZ_GET(1:N_TRACKED_SPECIES),ZETA_INOUT
REAL(EB) :: ERR_EST,ERR_TOL,A1(1:N_TRACKED_SPECIES),A2(1:N_TRACKED_SPECIES),A4(1:N_TRACKED_SPECIES),ZETA,ZETA_0,&
DT_SUB,DT_SUB_NEW,DT_ITER,ZZ_STORE(1:N_TRACKED_SPECIES,1:4),TV(1:3,1:N_TRACKED_SPECIES),CELL_MASS,&
ZZ_DIFF(1:3,1:N_TRACKED_SPECIES),ZZ_MIXED(1:N_TRACKED_SPECIES),ZZ_UNMIXED(1:N_TRACKED_SPECIES),&
ZZ_MIXED_NEW(1:N_TRACKED_SPECIES),TAU_D,TAU_G,TAU_U,TAU_MIX,TMP_MIXED,TMP_UNMIXED,DT_SUB_MIN,RHO_HAT,&
PBAR_0,VEL_RMS,TAU_RES,ZZ_0(1:N_TRACKED_SPECIES),&
Q_REAC_SUB(1:N_REACTIONS),Q_REAC_1(1:N_REACTIONS),Q_REAC_2(1:N_REACTIONS),Q_REAC_4(1:N_REACTIONS),&
Q_REAC_SUM(1:N_REACTIONS),CHI_R_SUM,TIME_RAMP_FACTOR,&
TOTAL_MIXED_MASS_1,TOTAL_MIXED_MASS_2,TOTAL_MIXED_MASS_4,TOTAL_MIXED_MASS,&
ZETA_1,ZETA_2,ZETA_4,AIT_LOC
INTEGER :: NR,NS,ITER,TVI,RICH_ITER,TIME_ITER,RICH_ITER_MAX,CO_PASS,N_CO_PASS
INTEGER, PARAMETER :: TV_ITER_MIN=5
LOGICAL :: TV_FLUCT(1:N_TRACKED_SPECIES),EXTINCT
TYPE(REACTION_TYPE), POINTER :: RN=>NULL()
REAL(EB), PARAMETER :: C_U = 0.4_EB*0.1_EB*SQRT(1.5_EB) ! C_U*C_DEARDORFF/SQRT(2/3)

ZZ_0 = ZZ_GET
```

```
EXTINCT = .FALSE.

VEL_RMS = 0._EB
IF (FIXED_MIX_TIME>0._EB) THEN
MIX_TIME_OUT=FIXED_MIX_TIME
ELSE
TAU_D=0._EB
DO NR =1,N_REACTIONS
RN => REACTION(NR)
TAU_D = MAX(TAU_D,D_Z(MIN(4999,NINT(TMP_IN)),RN%FUEL_SMIX_INDEX))
ENDDO
TAU_D = DELTA**2/MAX(TAU_D,TWO_EPSILON_EB)                        ! FDS Tech Guide (5.20)
IF (LES) THEN
TAU_U = C_U*RHO_IN*DELTA**2/MAX(MU_IN,TWO_EPSILON_EB)            ! FDS Tech Guide (5.24)
TAU_G = SQRT(2._EB*DELTA/(GRAV+1.E-10_EB))                       ! FDS Tech Guide (5.2)
MIX_TIME_OUT= MAX(TAU_CHEM,MIN(TAU_D,TAU_U,TAU_G,TAU_FLAME))     ! FDS Tech Guide (5.22)
VEL_RMS = SQRT(TWTH)*MU_IN/(RHO_IN*C_DEARDORFF*DELTA)
ELSE
MIX_TIME_OUT= MAX(TAU_CHEM,TAU_D)
ENDIF
ENDIF

IF (TRANSPORT_UNMIXED_FRACTION) THEN
ZETA_0 = ZETA_INOUT
ELSE
ZETA_0 = INITIAL_UNMIXED_FRACTION
ENDIF
CELL_MASS = RHO_IN*CELL_VOLUME
TAU_RES = MU_IN/(RHO_IN*SC)/MAX(2._EB*KRES_IN,TWO_EPSILON_EB)

IF (REIGNITION_MODEL) THEN
AIT_LOC = AIT_IN
ELSE
AIT_LOC = 1.E20_EB
ENDIF

DT_SUB_MIN = DT/REAL(MAX_CHEMISTRY_ITERATIONS,EB)

N_CO_PASS = 1
IF (EXTINCT_MOD==EXTINCTION_3) N_CO_PASS = 2

CO_EXTINCT_LOOP: DO CO_PASS = 1,N_CO_PASS

ZZ_STORE(:,:) = 0._EB
Q_OUT = 0._EB
ITER= 0
DT_ITER = 0._EB
CHI_R_OUT = 0._EB
CHEM_SUBIT_OUT = 0
REAC_SOURCE_TERM_OUT(:) = 0._EB
Q_REAC_OUT(:) = 0._EB
Q_REAC_SUM(:) = 0._EB
IF (N_FIXED_CHEMISTRY_SUBSTEPS>0) THEN
DT_SUB = DT/REAL(N_FIXED_CHEMISTRY_SUBSTEPS,EB)
DT_SUB_NEW = DT_SUB
RICH_ITER_MAX = 1
ELSE
DT_SUB = DT
DT_SUB_NEW = DT
RICH_ITER_MAX = 5
ENDIF
ZZ_UNMIXED = ZZ_GET
ZZ_MIXED = ZZ_GET
A1 = ZZ_GET
A2 = ZZ_GET
A4 = ZZ_GET

ZETA = ZETA_0
RHO_HAT = RHO_IN
TMP_MIXED = TMP_IN
TMP_UNMIXED = TMP_IN
TAU_MIX = MIX_TIME_OUT
PBAR_0 = PBAR_IN

INTEGRATION_LOOP: DO TIME_ITER = 1,MAX_CHEMISTRY_ITERATIONS

IF (SUPPRESSION) CALL CHECK_AUTO_IGNITION(EXTINCT,TMP_MIXED,AIT_LOC,CO_PASS)
IF (EXTINCT) EXIT INTEGRATION_LOOP
INTEGRATOR_SELECT: SELECT CASE (COMBUSTION_ODE_SOLVER)

CASE (EXPLICIT_EULER) ! Simple chemistry

! May be used with N_FIXED_CHEMISTRY_SUBSTEPS, but default mode is DT_SUB=DT for fast chemistry

CALL FIRE_FORWARD_EULER(ZZ_MIXED_NEW,ZZ_MIXED,ZZ_UNMIXED,ZETA,ZETA_0,DT_SUB,TMP_MIXED,TMP_UNMIXED,RHO_HAT,&
CELL_MASS,TAU_MIX,PBAR_0,DELTA,VEL_RMS,Q_REAC_SUB,TIME_ITER,TOTAL_MIXED_MASS,CO_PASS)
ZETA_0 = ZETA
ZZ_MIXED = ZZ_MIXED_NEW
```

```fortran
IF (SIMPLE_CHEMISTRY .AND. N_FIXED_CHEMISTRY_SUBSTEPS<0 .AND. TIME_ITER>1) THEN
CALL SHUTDOWN('ERROR: Error in Simple Chemistry')
ENDIF

CASE (RK2) ! Runge-Kutta 2 stage (use in combination with N_FIXED_CHEMISTRY_SUBSTEPS)

CALL FIRE_RK2(ZZ_MIXED_NEW,ZZ_MIXED,ZZ_UNMIXED,ZETA,ZETA_0,DT_SUB,1,TMP_MIXED,TMP_UNMIXED,RHO_HAT,&
CELL_MASS,TAU_MIX,PBAR_0,DELTA,VEL_RMS,Q_REAC_SUB,TIME_ITER,TOTAL_MIXED_MASS,CO_PASS)
ZETA_0 = ZETA
ZZ_MIXED = ZZ_MIXED_NEW

CASE (RK3) ! Runge-Kutta 3 stage (use in combination with N_FIXED_CHEMISTRY_SUBSTEPS)

CALL FIRE_RK3(ZZ_MIXED_NEW,ZZ_MIXED,ZZ_UNMIXED,ZETA,ZETA_0,DT_SUB,1,TMP_MIXED,TMP_UNMIXED,RHO_HAT,&
CELL_MASS,TAU_MIX,PBAR_0,DELTA,VEL_RMS,Q_REAC_SUB,TIME_ITER,TOTAL_MIXED_MASS,CO_PASS)
ZETA_0 = ZETA
ZZ_MIXED = ZZ_MIXED_NEW

CASE (RK2_RICHARDSON) ! Finite-rate (or mixed finite-rate/fast) chemistry

! May be used with N_FIXED_CHEMISTRY_SUBSTEPS, but default mode is to use error estimator and variable DT_SUB

ERR_TOL = RICHARDSON_ERROR_TOLERANCE
RICH_EX_LOOP: DO RICH_ITER = 1,RICH_ITER_MAX

DT_SUB = MIN(DT_SUB_NEW,DT-DT_ITER)

! FDS Tech Guide (E.3), (E.4), (E.5)
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,
      PBAR_0,&
DELTA,VEL_RMS,Q_REAC_1,TIME_ITER,TOTAL_MIXED_MASS_1,CO_PASS)
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,
      PBAR_0,&
DELTA,VEL_RMS,Q_REAC_2,TIME_ITER,TOTAL_MIXED_MASS_2,CO_PASS)
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,
      PBAR_0,&
DELTA,VEL_RMS,Q_REAC_4,TIME_ITER,TOTAL_MIXED_MASS_4,CO_PASS)

! Species Error Analysis
ERR_EST = MAXVAL(ABS((4._EB*A4-5._EB*A2+A1)))/45._EB ! FDS Tech Guide (E.7)

IF (N_FIXED_CHEMISTRY_SUBSTEPS<0) THEN
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)
IF (ERR_EST<ERR_TOL) EXIT RICH_EX_LOOP
ENDIF

ENDDO RICH_EX_LOOP

ZZ_MIXED   = (4._EB*A4-A2)*ONTH ! FDS Tech Guide (E.6)
Q_REAC_SUB = (4._EB*Q_REAC_4-Q_REAC_2)*ONTH
ZETA       = (4._EB*ZETA_4-ZETA_2)*ONTH
ZETA_0     = ZETA

!! debug
!ZETA_0 = ZETA
!ZZ_MIXED = A4
!Q_REAC_SUB = Q_REAC_4

END SELECT INTEGRATOR_SELECT

ZZ_GET = ZETA*ZZ_UNMIXED + (1._EB-ZETA)*ZZ_MIXED ! FDS Tech Guide (5.29)
ZETA_INOUT = ZETA

DT_ITER = DT_ITER + DT_SUB
ITER = ITER + 1
IF (OUTPUT_CHEM_IT) CHEM_SUBIT_OUT = ITER

Q_REAC_SUM = Q_REAC_SUM + Q_REAC_SUB

! Total Variation (TV) scheme (accelerates integration for finite-rate equilibrium calculations)
! See FDS Tech Guide Appendix E

IF (COMBUSTION_ODE_SOLVER==RK2_RICHARDSON .AND. N_REACTIONS>1) THEN
DO NS = 1,N_TRACKED_SPECIES
DO TVI = 1,3
ZZ_STORE(NS,TVI)=ZZ_STORE(NS,TVI+1)
ENDDO
ZZ_STORE(NS,4) = ZZ_GET(NS)
ENDDO
TV_FLUCT(:) = .FALSE.
IF (ITER >= TV_ITER_MIN) THEN
SPECIES_LOOP_TV: DO NS = 1,N_TRACKED_SPECIES
DO TVI = 1,3
TV(TVI,NS) = ABS(ZZ_STORE(NS,TVI+1)-ZZ_STORE(NS,TVI))
ZZ_DIFF(TVI,NS) = ZZ_STORE(NS,TVI+1)-ZZ_STORE(NS,TVI)
ENDDO
IF (SUM(TV(:,NS)) < ERR_TOL .OR. SUM(TV(:,NS)) >= ABS(2.9_EB*SUM(ZZ_DIFF(:,NS)))) THEN ! FDS Tech Guide (E.10)
TV_FLUCT(NS) = .TRUE.
```

```
ENDIF
IF (ALL(TV_FLUCT)) EXIT INTEGRATION_LOOP
ENDDO SPECIES_LOOP_TV
ENDIF
ENDIF

IF ( DT_ITER > (DT+TWO_EPSILON_EB) ) CALL SHUTDOWN('ERROR: DT_ITER > DT in COMBUSTION_MODEL')
IF ( DT_ITER > (DT-TWO_EPSILON_EB) ) EXIT INTEGRATION_LOOP

ENDDO INTEGRATION_LOOP

! Compute heat release rate

Q_OUT = -RHO_IN*SUM(SPECIES_MIXTURE%H_F*(ZZ_GET-ZZ_0))/DT ! FDS Tech Guide (5.14)
!print *, 'first if print Q_OUT', Q_OUT

! Extinction model

IF (SUPPRESSION) THEN
SELECT CASE(EXTINCT_MOD)
CASE(EXTINCTION_1); CALL EXTINCT_1(EXTINCT,ZZ_0,TMP_IN)
CASE(EXTINCTION_2); CALL EXTINCT_2(EXTINCT,ZZ_0,ZZ_MIXED,TMP_IN)
CASE(EXTINCTION_3); CALL EXTINCT_3(EXTINCT,ZZ_0,ZZ_MIXED,TMP_IN,CO_PASS)
END SELECT
ENDIF

IF (.NOT.EXTINCT) THEN
EXIT CO_EXTINCT_LOOP
ELSE
ZZ_GET = ZZ_0
ZZ_STORE(:,:) = 0._EB
Q_OUT = 0._EB
CHI_R_OUT = 0._EB
CHEM_SUBIT_OUT = 0
REAC_SOURCE_TERM_OUT(:) = 0._EB
Q_REAC_OUT(:) = 0._EB
Q_REAC_SUM(:) = 0._EB
ENDIF

ENDDO CO_EXTINCT_LOOP

! Reaction rate-weighted radiative fraction

IF (SUM(Q_REAC_SUM)>TWO_EPSILON_EB) THEN
CHI_R_SUM=0._EB
DO NR=1,N_REACTIONS
RN=>REACTION(NR)
TIME_RAMP_FACTOR = EVALUATE_RAMP(T,0._EB,RN%RAMP_CHI_R_INDEX)
CHI_R_SUM = CHI_R_SUM + Q_REAC_SUM(NR)*RN%CHI_R*TIME_RAMP_FACTOR
ENDDO
CHI_R_OUT = CHI_R_SUM/(SUM(Q_REAC_SUM))
ENDIF
CHI_R_OUT = MAX(CHI_R_MIN,MIN(CHI_R_MAX,CHI_R_OUT))

! Store special diagnostic quantities

IF (REAC_SOURCE_CHECK) THEN
REAC_SOURCE_TERM_OUT = RHO_IN*(ZZ_GET-ZZ_0)/DT
Q_REAC_OUT = Q_REAC_SUM/CELL_VOLUME/DT
ENDIF

END SUBROUTINE COMBUSTION_MODEL


SUBROUTINE CHECK_AUTO_IGNITION(EXTINCT,TMP_MIXED,AIT_IN,CO_PASS)
LOGICAL, INTENT(INOUT) :: EXTINCT
INTEGER, INTENT(IN) :: CO_PASS
REAL(EB), INTENT(IN) :: TMP_MIXED,AIT_IN
INTEGER :: NR
REAL(EB):: AIT_LOC
TYPE(REACTION_TYPE),POINTER :: RN=>NULL()

EXTINCT = .TRUE.

SELECT CASE (EXTINCT_MOD)
CASE DEFAULT
! if ANY reaction exceeds AIT, allow all reactions and proceed to EXTINCTION MODEL
! note: here we include finite-rate reactions, else combustion model will exit
! integration loop (as presently coded)
REACTION_LOOP: DO NR=1,N_REACTIONS
RN => REACTION(NR)
IF (AIT_IN < 1.E10_EB) THEN
AIT_LOC = AIT_IN
ELSE
AIT_LOC = RN%AUTO_IGNITION_TEMPERATURE
ENDIF
IF ( TMP_MIXED > AIT_LOC ) THEN
EXTINCT = .FALSE.
```

```
EXIT REACTION_LOOP
ENDIF
ENDDO REACTION_LOOP

CASE(EXTINCTION_3)
! special case: CO production with 2 fast reactions
! CO_PASS==1 --> check hydrocarbon AIT
! CO_PASS==2 --> check CO AIT (not subject to pilot zone ignition [AIT_IN < 1.E10_EB])
RN => REACTION(CO_PASS)
IF (CO_PASS==1 .AND. AIT_IN < 1.E10_EB) THEN
AIT_LOC = AIT_IN
ELSE
AIT_LOC = RN%AUTO_IGNITION_TEMPERATURE
ENDIF
IF ( TMP_MIXED > AIT_LOC ) EXTINCT = .FALSE.

END SELECT

END SUBROUTINE CHECK_AUTO_IGNITION


SUBROUTINE EXTINCT_1(EXTINCT,ZZ_IN,TMP_IN)
USE PHYSICAL_FUNCTIONS,ONLY:GET_AVERAGE_SPECIFIC_HEAT
REAL(EB),INTENT(IN)::TMP_IN,ZZ_IN(1:N_TRACKED_SPECIES)
LOGICAL, INTENT(INOUT):: EXTINCT
REAL(EB):: Y_O2,Y_O2_CRIT,CPBAR
INTEGER :: NR
TYPE(REACTION_TYPE),POINTER :: RN=>NULL()

EXTINCT = .FALSE.
REACTION_LOOP: DO NR=1,N_REACTIONS
RN => REACTION(NR)
IF (.NOT.RN%FAST_CHEMISTRY) CYCLE REACTION_LOOP
CALL GET_AVERAGE_SPECIFIC_HEAT(ZZ_IN,CPBAR,TMP_IN)
Y_O2 = ZZ_IN(RN%AIR_SMIX_INDEX)
Y_O2_CRIT = CPBAR*(RN%CRIT_FLAME_TMP-TMP_IN)/RN%EPUMO2
IF (Y_O2 < Y_O2_CRIT) EXTINCT = .TRUE.
ENDDO REACTION_LOOP

END SUBROUTINE EXTINCT_1


SUBROUTINE EXTINCT_2(EXTINCT,ZZ_0_IN,ZZ_IN,TMP_IN)
USE PHYSICAL_FUNCTIONS,ONLY:GET_ENTHALPY
REAL(EB),INTENT(IN) ::TMP_IN,ZZ_0_IN(1:N_TRACKED_SPECIES),ZZ_IN(1:N_TRACKED_SPECIES)
LOGICAL, INTENT(INOUT) :: EXTINCT
REAL(EB) :: ZZ_HAT_0(1:N_TRACKED_SPECIES),ZZ_HAT(1:N_TRACKED_SPECIES),H_0,H_CRIT
INTEGER :: NS
TYPE(REACTION_TYPE), POINTER :: R1=>NULL()

IF (N_REACTIONS /= 1 .OR. .NOT.REACTION(1)%FAST_CHEMISTRY) RETURN
R1 => REACTION(1)

DO NS = 1,N_TRACKED_SPECIES
IF (NS==R1%FUEL_SMIX_INDEX) THEN
ZZ_HAT_0(NS) = ZZ_0_IN(NS)            ! FDS Tech Guide (5.15)
ZZ_HAT(NS)   = ZZ_IN(NS)              ! FDS Tech Guide (5.18)
ELSEIF (NS==R1%AIR_SMIX_INDEX) THEN
ZZ_HAT_0(NS) = ZZ_0_IN(NS) - ZZ_IN(NS) ! FDS Tech Guide (5.16)
ZZ_HAT(NS)   = 0._EB                  ! FDS Tech Guide (5.19)
ELSE
! FDS Tech Guide (5.17)
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)
ZZ_HAT(NS)   = ZZ_HAT_0(NS) + ZZ_IN(NS) - ZZ_0_IN(NS) ! FDS Tech Guide (5.20)
ENDIF
ENDDO

ZZ_HAT_0 = ZZ_HAT_0/SUM(ZZ_HAT_0)
ZZ_HAT = ZZ_HAT/SUM(ZZ_HAT)

! See if enough energy is released to raise the fuel and required "air" temperatures above the critical flame temp.

CALL GET_ENTHALPY(ZZ_HAT_0,H_0,TMP_IN) ! H of reactants participating in reaction (includes chemical enthalpy)
CALL GET_ENTHALPY(ZZ_HAT,H_CRIT,REACTION(1)%CRIT_FLAME_TMP) ! H of products at the critical flame temperature
IF (H_0 < H_CRIT) EXTINCT = .TRUE. ! FDS Tech Guide (5.21)

END SUBROUTINE EXTINCT_2


SUBROUTINE EXTINCT_3(EXTINCT,ZZ_0_IN,ZZ_IN,TMP_IN,CO_PASS)
USE PHYSICAL_FUNCTIONS,ONLY:GET_ENTHALPY
REAL(EB),INTENT(IN) ::TMP_IN,ZZ_0_IN(1:N_TRACKED_SPECIES),ZZ_IN(1:N_TRACKED_SPECIES)
INTEGER, INTENT(IN) :: CO_PASS
LOGICAL, INTENT(INOUT) :: EXTINCT
REAL(EB) :: ZZ_HAT_0(1:N_TRACKED_SPECIES),ZZ_HAT(1:N_TRACKED_SPECIES),H_0,H_CRIT
INTEGER :: NS
```

```
TYPE(REACTION_TYPE), POINTER :: R1=>NULL(),R2=>NULL()

! R1: Fuel + O2 => CO + Products
! R2: CO + (1/2)O2 => CO2
! extinction model:
!   1. (first pass) Evaluate EXTINCT based on R1 + R2
!   2. (second pass, EXTINCT=T on first pass) Evaluate EXTINCT based on R2

R1 => REACTION(1)
R2 => REACTION(2)

DO NS = 1,N_TRACKED_SPECIES
IF (NS==R1%FUEL_SMIX_INDEX .OR. NS==R2%FUEL_SMIX_INDEX) THEN
ZZ_HAT_0(NS) = ZZ_0_IN(NS)
ZZ_HAT(NS)   = ZZ_IN(NS)
ELSEIF (NS==R1%AIR_SMIX_INDEX) THEN
ZZ_HAT_0(NS) = ZZ_0_IN(NS) - ZZ_IN(NS)
ZZ_HAT(NS)   = 0._EB
ELSE
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)
ZZ_HAT(NS)   = ZZ_IN(NS) - ZZ_0_IN(NS) + ZZ_HAT_0(NS)
ENDIF
ENDDO

ZZ_HAT_0 = ZZ_HAT_0/SUM(ZZ_HAT_0)
ZZ_HAT = ZZ_HAT/SUM(ZZ_HAT)

! See if enough energy is released to raise the fuel and required "air" temperatures above the critical flame temp.

CALL GET_ENTHALPY(ZZ_HAT_0,H_0,TMP_IN) ! H of reactants participating in reaction(includes chemical enthalpy)
CALL GET_ENTHALPY(ZZ_HAT,H_CRIT,REACTION(CO_PASS)%CRIT_FLAME_TMP) ! H of products at the critical flame temperature
IF (H_0 < H_CRIT) EXTINCT = .TRUE.

END SUBROUTINE EXTINCT_3

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
USE COMP_FUNCTIONS, ONLY:SHUTDOWN
USE PHYSICAL_FUNCTIONS, ONLY: GET_REALIZABLE_MF,GET_AVERAGE_SPECIFIC_HEAT
!USE RADCONS, ONLY: RADIATIVE_FRACTION_1    !changed name RADIATIVE_FRACTION_1
REAL(EB) :: RADIATIVE_FRACTION      !added this as required
REAL(EB), INTENT(IN) :: ZZ_IN(1:N_TRACKED_SPECIES),ZETA_IN,DT_LOC,RHO_HAT,ZZ_UNMIXED(1:N_TRACKED_SPECIES),CELL_MASS,&
TAU_MIX
REAL(EB), INTENT(OUT) :: ZZ_OUT(1:N_TRACKED_SPECIES),ZETA_OUT
REAL(EB), INTENT(INOUT) :: TMP_MIXED
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),&
BOUNDEDNESS_CORRECTION,MIXED_MASS(1:N_TRACKED_SPECIES),MIXED_MASS_0(1:N_TRACKED_SPECIES),TOTAL_MIXED_MASS,Q_TEMP,CPBAR
INTEGER :: SR
INTEGER, PARAMETER :: INFINITELY_FAST=1,FINITE_RATE=2
LOGICAL :: TEMPERATURE_DEPENDENT_REACTION=.FALSE.

ZETA_OUT = ZETA_IN*EXP(-DT_LOC/TAU_MIX) ! FDS Tech Guide (5.29)
IF (ZETA_OUT < TWO_EPSILON_EB) ZETA_OUT = 0._EB
MIXED_MASS_0 = CELL_MASS*ZZ_IN
UNMIXED_MASS_0 = CELL_MASS*ZZ_UNMIXED
MIXED_MASS = MAX(0._EB,MIXED_MASS_0 - (ZETA_OUT - ZETA_IN)*UNMIXED_MASS_0) ! FDS Tech Guide (5.37)
TOTAL_MIXED_MASS = SUM(MIXED_MASS)
ZZ_0 = MIXED_MASS/MAX(TOTAL_MIXED_MASS,TWO_EPSILON_EB) ! FDS Tech Guide (5.35)

IF (ANY(REACTION(:)%FAST_CHEMISTRY)) THEN
DO SR = 0,N_SERIES_REACTIONS
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
ZZ_NEW = ZZ_0 + DZZ ! test Forward Euler step (5.53)
BOUNDEDNESS_CORRECTION = FUNC_BCOR(ZZ_0,ZZ_NEW) ! Reaction rate boundedness correction
ZZ_NEW = ZZ_0 + DZZ*BOUNDEDNESS_CORRECTION ! corrected FE step for all species (5.54)
ZZ_0 = ZZ_NEW
ENDDO
ENDIF

IF (.NOT.ALL(REACTION(:)%FAST_CHEMISTRY)) THEN
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
ZZ_NEW = ZZ_0 + DZZ
BOUNDEDNESS_CORRECTION = FUNC_BCOR(ZZ_0,ZZ_NEW)
ZZ_NEW = ZZ_0 + DZZ*BOUNDEDNESS_CORRECTION
IF (TEMPERATURE_DEPENDENT_REACTION) THEN
Q_TEMP = SUM(SPECIES_MIXTURE%H_F*DZZ*BOUNDEDNESS_CORRECTION)
CALL GET_AVERAGE_SPECIFIC_HEAT(ZZ_NEW,CPBAR,TMP_MIXED)
TMP_MIXED = TMP_MIXED + DT_LOC*(1._EB-RADIATIVE_FRACTION)*Q_TEMP/CPBAR
ENDIF
ENDIF
```

```fortran
! Enforce realizability on mass fractions

CALL GET_REALIZABLE_MF(ZZ_NEW)

ZZ_OUT = ZZ_NEW

END SUBROUTINE FIRE_FORWARD_EULER_1     !changing names

SUBROUTINE FIRE_FORWARD_EULER(ZZ_OUT,ZZ_IN,ZZ_UNMIXED,ZETA_OUT,ZETA_IN,DT_LOC,TMP_MIXED,TMP_UNMIXED,RHO_HAT,
        CELL_MASS,TAU_MIX,&
PBAR_0,DELTA,VEL_RMS,Q_REAC_LOC,SUB_IT,TOTAL_MIXED_MASS,CO_PASS)
USE COMP_FUNCTIONS, ONLY:SHUTDOWN
USE PHYSICAL_FUNCTIONS, ONLY: GET_REALIZABLE_MF,GET_AVERAGE_SPECIFIC_HEAT
REAL(EB), INTENT(IN) :: ZZ_IN(1:N_TRACKED_SPECIES),ZETA_IN,DT_LOC,RHO_HAT,ZZ_UNMIXED(1:N_TRACKED_SPECIES),
        CELL_MASS,TAU_MIX,&
PBAR_0,DELTA,VEL_RMS,TMP_UNMIXED
INTEGER, INTENT(IN) :: SUB_IT,CO_PASS
REAL(EB), INTENT(OUT) :: ZZ_OUT(1:N_TRACKED_SPECIES),ZETA_OUT,Q_REAC_LOC(1:N_REACTIONS),TOTAL_MIXED_MASS
REAL(EB), INTENT(INOUT) :: TMP_MIXED
REAL(EB) :: ZZ_0(1:N_TRACKED_SPECIES),ZZ_NEW(1:N_TRACKED_SPECIES),DZZ(1:N_TRACKED_SPECIES),&
MIXED_MASS(1:N_TRACKED_SPECIES),MIXED_MASS_0(1:N_TRACKED_SPECIES),&
Q_REAC_OUT(1:N_REACTIONS),TOTAL_MIXED_MASS_0
INTEGER, PARAMETER :: INFINITELY_FAST=1,FINITE_RATE=2

! Determine initial state of mixed reactor zone

TOTAL_MIXED_MASS_0  = (1._EB-ZETA_IN)*CELL_MASS
MIXED_MASS_0  = ZZ_IN*TOTAL_MIXED_MASS_0

! Mixing step

ZETA_OUT = MAX(0._EB,ZETA_IN*EXP(-DT_LOC/TAU_MIX)) ! FDS Tech Guide (5.28)
TOTAL_MIXED_MASS = (1._EB-ZETA_OUT)*CELL_MASS
MIXED_MASS = MAX(0._EB,MIXED_MASS_0 - (ZETA_OUT - ZETA_IN)*ZZ_UNMIXED*CELL_MASS) ! after mixing step, FDS Tech
        Guide (5.36)
ZZ_0 = MIXED_MASS/MAX(TOTAL_MIXED_MASS,TWO_EPSILON_EB) ! FDS Tech Guide (5.37)

! Enforce realizability on mass fractions

CALL GET_REALIZABLE_MF(ZZ_0)

Q_REAC_LOC(:) = 0._EB

! Removed TEMPERATURE_DEPENDENT_REACTION until other bugs are sorted out
TMP_MIXED = TMP_UNMIXED
IF (ANY(REACTION(:)%FAST_CHEMISTRY)) THEN
CALL REACTION_RATE(DZZ,ZZ_0,DT_LOC,RHO_HAT,TMP_MIXED,INFINITELY_FAST,PBAR_0,DELTA,VEL_RMS,Q_REAC_OUT,SUB_IT,
        CO_PASS)
ZZ_NEW = ZZ_0 + DZZ
ZZ_0 = ZZ_NEW
Q_REAC_LOC = Q_REAC_LOC + Q_REAC_OUT*TOTAL_MIXED_MASS
ENDIF
IF (.NOT.ALL(REACTION(:)%FAST_CHEMISTRY)) THEN
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)
ZZ_NEW = ZZ_0 + DZZ
Q_REAC_LOC = Q_REAC_LOC + Q_REAC_OUT*TOTAL_MIXED_MASS
ENDIF

! Enforce realizability on mass fractions

CALL GET_REALIZABLE_MF(ZZ_NEW)

ZZ_OUT = ZZ_NEW

END SUBROUTINE FIRE_FORWARD_EULER



REAL(EB) FUNCTION FUNC_BCOR(ZZ_0,ZZ_NEW)
! This function finds a correction for reaction rates such that all species remain bounded.

REAL(EB), INTENT(IN) :: ZZ_0(1:N_TRACKED_SPECIES),ZZ_NEW(1:N_TRACKED_SPECIES)
REAL(EB) :: BCOR,DZ_IB,DZ_OB
INTEGER :: NS

!print *, 'BCOR function'

BCOR = 1._EB
DO NS=1,N_TRACKED_SPECIES
IF (ZZ_NEW(NS)<0._EB) THEN ! FDS Tech Guide (5.55)
DZ_IB=ZZ_0(NS)           ! DZ "in bounds"
DZ_OB=ABS(ZZ_NEW(NS)) ! DZ "out of bounds"
BCOR = MIN( BCOR, DZ_IB/MAX(DZ_IB+DZ_OB,TWO_EPSILON_EB) )
ENDIF
IF (ZZ_NEW(NS)>1._EB) THEN ! FDS Tech Guide (5.55)
DZ_IB=1._EB-ZZ_0(NS)
DZ_OB=ZZ_NEW(NS)-1._EB
BCOR = MIN( BCOR, DZ_IB/MAX(DZ_IB+DZ_OB,TWO_EPSILON_EB) )
```

```fortran
ENDIF
ENDDO
FUNC_BCOR = BCOR

END FUNCTION FUNC_BCOR

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) !changed name to FIRE_RK2_1 for 6.2.0
! This function uses RK2 to integrate ZZ_O from t=0 to t=DT_SUB in increments of DT_LOC=DT_SUB/N_INC

REAL(EB), INTENT(IN) :: ZZ_IN(1:N_TRACKED_SPECIES),DT_SUB,ZETA_IN,RHO_HAT,ZZ_UNMIXED(1:N_TRACKED_SPECIES),CELL_MASS,&
TAU_MIX
REAL(EB), INTENT(OUT) :: ZZ_OUT(1:N_TRACKED_SPECIES),ZETA_OUT
REAL(EB), INTENT(INOUT) :: TMP_MIXED
INTEGER, INTENT(IN) :: N_INC
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,ZETA_2
INTEGER :: N

DT_LOC = DT_SUB/REAL(N_INC,EB)
ZZ_0 = ZZ_IN
ZETA_0 = ZETA_IN
DO N=1,N_INC
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 name
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 name
ZZ_OUT = 0.5_EB*(ZZ_0 + ZZ_2)
ZZ_0 = ZZ_OUT
ZETA_OUT = ZETA_1
ZETA_0 = ZETA_OUT
ENDDO

END SUBROUTINE FIRE_RK2_1      !changed name

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,TAU_MIX,&
PBAR_0,DELTA,VEL_RMS,Q_REAC_OUT,SUB_IT,TOTAL_MIXED_MASS_OUT,CO_PASS)

! This function uses RK2 to integrate ZZ_O from t=0 to t=DT_SUB in increments of DT_LOC=DT_SUB/N_INC

REAL(EB), INTENT(IN) :: ZZ_IN(1:N_TRACKED_SPECIES),DT_SUB,ZETA_IN,RHO_HAT,ZZ_UNMIXED(1:N_TRACKED_SPECIES),CELL_MASS,&
TAU_MIX,PBAR_0,DELTA,VEL_RMS,TMP_UNMIXED
REAL(EB), INTENT(OUT) :: ZZ_OUT(1:N_TRACKED_SPECIES),ZETA_OUT,Q_REAC_OUT(1:N_REACTIONS),TOTAL_MIXED_MASS_OUT
REAL(EB), INTENT(INOUT) :: TMP_MIXED
INTEGER, INTENT(IN) :: N_INC,SUB_IT,CO_PASS
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,ZETA_2,&
Q_REAC_1(1:N_REACTIONS),Q_REAC_2(1:N_REACTIONS),TOTAL_MIXED_MASS_0,TOTAL_MIXED_MASS_1,TOTAL_MIXED_MASS_2
INTEGER :: N

DT_LOC = DT_SUB/REAL(N_INC,EB)
ZZ_0 = ZZ_IN
ZETA_0 = ZETA_IN
Q_REAC_OUT(:) = 0._EB
TOTAL_MIXED_MASS_0 = (1._EB-ZETA_0)*CELL_MASS

DO N=1,N_INC

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,&
PBAR_0,DELTA,VEL_RMS,Q_REAC_1,SUB_IT,TOTAL_MIXED_MASS_1,CO_PASS)

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,&
PBAR_0,DELTA,VEL_RMS,Q_REAC_2,SUB_IT,TOTAL_MIXED_MASS_2,CO_PASS)

IF (TOTAL_MIXED_MASS_2>TWO_EPSILON_EB) THEN
ZZ_OUT = 0.5_EB*(ZZ_0*TOTAL_MIXED_MASS_0 + ZZ_2*TOTAL_MIXED_MASS_2)
TOTAL_MIXED_MASS_OUT = SUM(ZZ_OUT)
ZZ_OUT = ZZ_OUT/TOTAL_MIXED_MASS_OUT
ELSE
ZZ_OUT = ZZ_0
ENDIF

ZETA_OUT = MAX(0._EB,1._EB-TOTAL_MIXED_MASS_OUT/CELL_MASS)

Q_REAC_OUT = Q_REAC_OUT + 0.5_EB*(Q_REAC_1+Q_REAC_2)

ZZ_0 = ZZ_OUT
ZETA_0 = ZETA_OUT
TOTAL_MIXED_MASS_0 = TOTAL_MIXED_MASS_OUT
ENDDO

END SUBROUTINE FIRE_RK2
```

```
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,&
PBAR_0,DELTA,VEL_RMS,Q_REAC_OUT,SUB_IT,TOTAL_MIXED_MASS_OUT,CO_PASS)

! This function uses SSP RK3.  See Gottlieb, Shu, Tadmor, SIAM Review, 2001.

REAL(EB), INTENT(IN) :: ZZ_IN(1:N_TRACKED_SPECIES),DT_SUB,ZETA_IN,RHO_HAT,ZZ_UNMIXED(1:N_TRACKED_SPECIES),CELL_MASS,&
TAU_MIX,PBAR_0,DELTA,VEL_RMS,TMP_UNMIXED
REAL(EB), INTENT(OUT) :: ZZ_OUT(1:N_TRACKED_SPECIES),ZETA_OUT,Q_REAC_OUT(1:N_REACTIONS),TOTAL_MIXED_MASS_OUT
REAL(EB), INTENT(INOUT) :: TMP_MIXED
INTEGER, INTENT(IN) :: N_INC,SUB_IT,CO_PASS
REAL(EB) :: DT_LOC,TOTAL_MIXED_MASS_0,TOTAL_MIXED_MASS_1,TOTAL_MIXED_MASS_2,TOTAL_MIXED_MASS_3,&
ZZ_0(1:N_TRACKED_SPECIES),ZZ_1(1:N_TRACKED_SPECIES),ZZ_2(1:N_TRACKED_SPECIES),ZZ_3(1:N_TRACKED_SPECIES),&
ZETA_0,ZETA_1,ZETA_2,ZETA_3,&
Q_REAC_1(1:N_REACTIONS),Q_REAC_2(1:N_REACTIONS),Q_REAC_3(1:N_REACTIONS)
INTEGER :: N

DT_LOC = DT_SUB/REAL(N_INC,EB) ! in principle, multiple increments could be used for Richardson extrapolation
ZZ_0 = ZZ_IN
ZETA_0 = ZETA_IN
Q_REAC_OUT(:) = 0._EB
TOTAL_MIXED_MASS_0 = (1._EB-ZETA_0)*CELL_MASS

INC_LOOP: DO N=1,N_INC

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,&
PBAR_0,DELTA,VEL_RMS,Q_REAC_1,SUB_IT,TOTAL_MIXED_MASS_1,CO_PASS)

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,&
PBAR_0,DELTA,VEL_RMS,Q_REAC_2,SUB_IT,TOTAL_MIXED_MASS_2,CO_PASS)

IF (TOTAL_MIXED_MASS_2>TWO_EPSILON_EB) THEN
ZZ_2 = 0.75_EB*ZZ_0*TOTAL_MIXED_MASS_0 + 0.25_EB*ZZ_2*TOTAL_MIXED_MASS_2
ZZ_2 = ZZ_2/SUM(ZZ_2)
ELSE
ZZ_2 = ZZ_0
ENDIF

Q_REAC_2 = 0.25_EB * (Q_REAC_1 + Q_REAC_2)

ZETA_2 = 0.75_EB*ZETA_0 + 0.25_EB*ZETA_2

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,&
PBAR_0,DELTA,VEL_RMS,Q_REAC_3,SUB_IT,TOTAL_MIXED_MASS_3,CO_PASS)

IF (TOTAL_MIXED_MASS_3>TWO_EPSILON_EB) THEN
ZZ_OUT = ONTH*ZZ_0*TOTAL_MIXED_MASS_0 + TWTH*ZZ_3*TOTAL_MIXED_MASS_3
TOTAL_MIXED_MASS_OUT = SUM(ZZ_OUT)
ZZ_OUT = ZZ_OUT/TOTAL_MIXED_MASS_OUT
ELSE
ZZ_OUT = ZZ_0
ENDIF

Q_REAC_OUT = Q_REAC_OUT + TWTH * (Q_REAC_2 + Q_REAC_3)

ZETA_OUT = MAX(0._EB,1._EB-TOTAL_MIXED_MASS_OUT/CELL_MASS)

ZZ_0 = ZZ_OUT
ZETA_0 = ZETA_OUT
TOTAL_MIXED_MASS_0 = TOTAL_MIXED_MASS_OUT

ENDDO INC_LOOP

END SUBROUTINE FIRE_RK3

SUBROUTINE REACTION_RATE_1(DZZ,ZZ_0,DT_LOC,RHO_0,TMP_0,KINETICS)        !changed name to REACTION_RATE_1 for 6.2.0
USE PHYSICAL_FUNCTIONS, ONLY : GET_MASS_FRACTION_ALL,GET_SPECIFIC_GAS_CONSTANT,GET_GIBBS_FREE_ENERGY,GET_MOLECULAR_WEIGHT
REAL(EB), INTENT(OUT) :: DZZ(1:N_TRACKED_SPECIES)
REAL(EB), INTENT(IN) :: ZZ_0(1:N_TRACKED_SPECIES),DT_LOC,RHO_0,TMP_0
INTEGER, INTENT(IN) :: KINETICS
REAL(EB) :: DZ_F(1:N_REACTIONS),YY_PRIMITIVE(1:N_SPECIES),DG_RXN,MW,MOLPCM3
INTEGER :: I,NS
INTEGER, PARAMETER :: INFINITELY_FAST=1,FINITE_RATE=2
TYPE(REACTION_TYPE),POINTER :: RN=>NULL()

DZ_F = 0._EB
DZZ = 0._EB

KINETICS_SELECT: SELECT CASE(KINETICS)

CASE(INFINITELY_FAST)
IF (EXTINCT1) RETURN        !changed name for 6.2.0
```

```
REACTION_LOOP_1: DO I=1,N_REACTIONS
RN => REACTION(I)
IF (.NOT.RN%FAST_CHEMISTRY) CYCLE REACTION_LOOP_1
DZ_F(I) = ZZ_0(RN%FUEL_SMIX_INDEX)
DZZ = DZZ + RN%NU_MW_O_MW_F*DZ_F(I)
ENDDO REACTION_LOOP_1

CASE(FINITE_RATE)
REACTION_LOOP_2: DO I=1,N_REACTIONS
RN => REACTION(I)
IF (RN%FAST_CHEMISTRY .OR. ZZ_0(RN%FUEL_SMIX_INDEX) < ZZ_MIN_GLOBAL) CYCLE REACTION_LOOP_2
IF (RN%AIR_SMIX_INDEX > -1) THEN
IF (ZZ_0(RN%AIR_SMIX_INDEX) < ZZ_MIN_GLOBAL) CYCLE REACTION_LOOP_2 ! no expected air
ENDIF
CALL GET_MASS_FRACTION_ALL(ZZ_0,YY_PRIMITIVE)
DO NS=1,N_SPECIES
IF(RN%N_S(NS)>= -998._EB .AND. YY_PRIMITIVE(NS) < ZZ_MIN_GLOBAL) CYCLE REACTION_LOOP_2
ENDDO
DZ_F(I) = 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)
DO NS=1,N_SPECIES
IF(RN%N_S(NS)>= -998._EB) DZ_F(I) = YY_PRIMITIVE(NS)**RN%N_S(NS)*DZ_F(I)
ENDDO
IF (RN%THIRD_BODY) THEN
CALL GET_MOLECULAR_WEIGHT(ZZ_0,MW)
MOLPCM3 = RHO_0/MW*0.001_EB ! mol/cm^3
DZ_F(I) = DZ_F(I) * MOLPCM3
ENDIF
IF(RN%REVERSE) THEN ! compute equilibrium constant
CALL GET_GIBBS_FREE_ENERGY(DG_RXN,RN%NU,TMP_0)
RN%K = EXP(-DG_RXN/(R0*TMP_0))
ENDIF
DZZ = DZZ + RN%NU_MW_O_MW_F*DZ_F(I)*DT_LOC/RN%K
ENDDO REACTION_LOOP_2

END SELECT KINETICS_SELECT

END SUBROUTINE REACTION_RATE_1      !changed name

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)
USE COMP_FUNCTIONS, ONLY: SHUTDOWN
USE PHYSICAL_FUNCTIONS, ONLY : GET_MASS_FRACTION_ALL,GET_SPECIFIC_GAS_CONSTANT,GET_GIBBS_FREE_ENERGY,
    GET_MOLECULAR_WEIGHT
REAL(EB), INTENT(OUT) :: DZZ(1:N_TRACKED_SPECIES),Q_REAC_OUT(1:N_REACTIONS)
REAL(EB), INTENT(IN) :: ZZ_0(1:N_TRACKED_SPECIES),DT_SUB,RHO_0,TMP_0,PBAR_0,DELTA,VEL_RMS
INTEGER, INTENT(IN) :: KINETICS,SUB_IT,CO_PASS
REAL(EB) :: DZ_F,YY_PRIMITIVE(1:N_SPECIES),DG_RXN,MW,MOLPCM3,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
INTEGER :: I,NS,SUB_IT_USE,OUTER_IT
LOGICAL :: REACTANTS_PRESENT
INTEGER, PARAMETER :: INFINITELY_FAST=1,FINITE_RATE=2
TYPE(REACTION_TYPE), POINTER :: RN=>NULL()

ZZ_NEW = ZZ_0
Q_REAC_OUT = 0._EB
Q_REAC_TMP = 0._EB
SUB_IT_USE = SUB_IT ! keep this for debug

KINETICS_SELECT: SELECT CASE(KINETICS)

CASE(INFINITELY_FAST)

FAST_REAC_LOOP: DO OUTER_IT=1,N_REACTIONS
IF (CO_PASS==2 .AND. OUTER_IT/=2) CYCLE FAST_REAC_LOOP
ZZ_TMP = ZZ_NEW
DZZ = 0._EB
REACTANTS_PRESENT = .FALSE.
REACTION_LOOP_1: DO I=1,N_REACTIONS
RN => REACTION(I)
IF (.NOT.RN%FAST_CHEMISTRY) CYCLE REACTION_LOOP_1
IF (RN%AIR_SMIX_INDEX > -1) THEN
DZ_F = ZZ_TMP(RN%FUEL_SMIX_INDEX)*ZZ_TMP(RN%AIR_SMIX_INDEX) ! 2nd-order reaction
ELSE
DZ_F = ZZ_TMP(RN%FUEL_SMIX_INDEX) ! 1st-order
ENDIF
IF (DZ_F > TWO_EPSILON_EB) REACTANTS_PRESENT = .TRUE.
DTHETA = FLAME_SPEED_FACTOR(ZZ_TMP,DT_LOC,RHO_0,TMP_0,PBAR_0,I,DELTA,VEL_RMS)
AA = RN%A_PRIME_FAST * RHO_0**RN%RHO_EXPONENT_FAST * DTHETA
DZZ = DZZ + AA * RN%NU_MW_O_MW_F * DZ_F
Q_REAC_TMP(I) = RN%HEAT_OF_COMBUSTION * AA * DZ_F
ENDDO REACTION_LOOP_1
IF (REACTANTS_PRESENT) THEN
DT_TMP = HUGE_EB
DO NS = 1,N_TRACKED_SPECIES
IF (DZZ(NS) < 0._EB) DT_TMP(NS) = -ZZ_TMP(NS)/DZZ(NS)
ENDDO
DT_MIN = MINVAL(DT_TMP)
ZZ_NEW = ZZ_TMP + DZZ*DT_MIN
Q_REAC_OUT = Q_REAC_OUT + Q_REAC_TMP*DT_MIN
```

```
ELSE
EXIT FAST_REAC_LOOP
ENDIF
ENDDO FAST_REAC_LOOP
DZZ = ZZ_NEW - ZZ_0

CASE(FINITE_RATE)

DT_LOC = DT_SUB
SLOW_REAC_LOOP: DO OUTER_IT=1,N_REACTIONS
ZZ_TMP = ZZ_NEW
DZZ = 0._EB
REACTANTS_PRESENT = .FALSE.
REACTION_LOOP_2: DO I=1,N_REACTIONS
RN => REACTION(I)
IF (RN%FAST_CHEMISTRY) CYCLE REACTION_LOOP_2
IF (ZZ_TMP(RN%FUEL_SMIX_INDEX) < ZZ_MIN_GLOBAL) CYCLE REACTION_LOOP_2
IF (RN%AIR_SMIX_INDEX > -1) THEN
IF (ZZ_TMP(RN%AIR_SMIX_INDEX) < ZZ_MIN_GLOBAL) CYCLE REACTION_LOOP_2 ! no expected air
ENDIF
CALL GET_MASS_FRACTION_ALL(ZZ_TMP,YY_PRIMITIVE)
DO NS=1,N_SPECIES
IF(RN%N_S(NS) > -998._EB .AND. YY_PRIMITIVE(NS) < ZZ_MIN_GLOBAL) CYCLE REACTION_LOOP_2
ENDDO
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)
DO NS=1,N_SPECIES
IF(RN%N_S(NS) > -998._EB) DZ_F = YY_PRIMITIVE(NS)**RN%N_S(NS)*DZ_F
ENDDO
IF (RN%THIRD_BODY) THEN
CALL GET_MOLECULAR_WEIGHT(ZZ_TMP,MW)
MOLPCM3 = RHO_0/MW*0.001_EB ! mol/cm^3
DZ_F = DZ_F * MOLPCM3
ENDIF
IF(RN%REVERSE) THEN ! compute equilibrium constant
CALL GET_GIBBS_FREE_ENERGY(DG_RXN,RN%NU,TMP_0)
RN%K = EXP(-DG_RXN/(R0*TMP_0))
DZ_F = DZ_F/RN%K
ENDIF
IF (DZ_F > TWO_EPSILON_EB) REACTANTS_PRESENT = .TRUE.
Q_REAC_TMP(I) = RN%HEAT_OF_COMBUSTION * DZ_F * DT_LOC ! Note: here DZ_F=dZ/dt, hence need DT_LOC
DZZ = DZZ + RN%NU_MW_O_MW_F*DZ_F*DT_LOC
ENDDO REACTION_LOOP_2
IF (REACTANTS_PRESENT) THEN
DT_TMP = HUGE_EB
DO NS = 1,N_TRACKED_SPECIES
IF (DZZ(NS) < 0._EB) DT_TMP(NS) = -ZZ_TMP(NS)/DZZ(NS)
ENDDO
! Think of DT_MIN as the fraction of DT_LOC we can take and remain bounded.
DT_MIN = MIN(1._EB,MINVAL(DT_TMP))
DT_LOC = DT_LOC*(1._EB-DT_MIN)
ZZ_NEW = ZZ_TMP + DZZ*DT_MIN
Q_REAC_OUT = Q_REAC_OUT + Q_REAC_TMP*DT_MIN
IF (DT_LOC<TWO_EPSILON_EB) EXIT SLOW_REAC_LOOP
ELSE
EXIT SLOW_REAC_LOOP
ENDIF
ENDDO SLOW_REAC_LOOP
DZZ = ZZ_NEW - ZZ_0

END SELECT KINETICS_SELECT

END SUBROUTINE REACTION_RATE


REAL(EB) FUNCTION FLAME_SPEED_FACTOR(ZZ_0,DT_LOC,RHO_0,TMP_0,PBAR_0,NR,DELTA,VEL_RMS)
USE PHYSICAL_FUNCTIONS, ONLY : GET_SENSIBLE_ENTHALPY,GET_SPECIFIC_GAS_CONSTANT,GET_SPECIFIC_HEAT
REAL(EB), INTENT(IN) :: ZZ_0(1:N_TRACKED_SPECIES),RHO_0,TMP_0,PBAR_0,DT_LOC,DELTA,VEL_RMS
INTEGER, INTENT(IN) :: NR
TYPE(REACTION_TYPE),POINTER :: RN=>NULL()
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,H_NEW,TMP_2,CP_B
INTEGER :: IT
! REAL(EB) :: DPHI ! debug

FLAME_SPEED_FACTOR = 1._EB

RN=>REACTION(NR)
IF (RN%FLAME_SPEED<0._EB) RETURN

! equivalence ratio of unburnt mixture
PHI = RN%S*ZZ_0(RN%FUEL_SMIX_INDEX)/ZZ_0(RN%AIR_SMIX_INDEX)

! burnt composition
DZ_F = MIN(ZZ_0(RN%FUEL_SMIX_INDEX),ZZ_0(RN%AIR_SMIX_INDEX)/RN%S)
ZZ_B = ZZ_0 + RN%NU_MW_O_MW_F*DZ_F
ZZ_B = MIN(1._EB,MAX(0._EB,ZZ_B))

! find burnt zone temperature
CALL GET_SENSIBLE_ENTHALPY(ZZ_0,H_S_0,TMP_0)
```

```
H_NEW = H_S_0 + (1._EB-RN%CHI_R)*DZ_F*RN%HEAT_OF_COMBUSTION
TMP_B = TMP_0
TMP_2 = TMP_B

DO IT=1,10
CALL GET_SENSIBLE_ENTHALPY(ZZ_B,H_S_B,TMP_B)
CALL GET_SPECIFIC_HEAT(ZZ_B,CP_B,TMP_B)
TMP_B = TMP_B+(H_NEW - H_S_B)/CP_B
! < 10 K error for determining flame speed is sufficient
IF(ABS(TMP_2-TMP_B)<10._EB) EXIT
TMP_2 = TMP_B
ENDDO

! compute burnt zone density
CALL GET_SPECIFIC_GAS_CONSTANT(ZZ_B,RSUM_B)
RHO_B = PBAR_0/(RSUM_B*TMP_B)

! get turbulent flame speed

! ! (debug) check laminar flame speed ramp
! PHI = 0._EB
! DPHI = .1_EB
! DO IT=1,20
!    PHI = PHI+DPHI
!    S_L = LAMINAR_FLAME_SPEED(TMP_0,PHI,NR)
!    print *,PHI,S_L
! ENDDO
! stop

S_L = LAMINAR_FLAME_SPEED(TMP_0,PHI,NR)

IF (S_L<TWO_EPSILON_EB) THEN
FLAME_SPEED_FACTOR = 0._EB
ELSE
S_T = MAX( S_L, S_L*( 1._EB + RN%TURBULENT_FLAME_SPEED_ALPHA*(VEL_RMS/S_L)**RN%TURBULENT_FLAME_SPEED_EXPONENT ) )
FLAME_SPEED_FACTOR = RHO_B/RHO_0 * S_T * DT_LOC/DELTA
ENDIF

END FUNCTION FLAME_SPEED_FACTOR


REAL(EB) FUNCTION LAMINAR_FLAME_SPEED(TMP,EQ,NR)
USE MATH_FUNCTIONS, ONLY: EVALUATE_RAMP, INTERPOLATE2D
REAL(EB), INTENT(IN) :: TMP,EQ
INTEGER, INTENT(IN) :: NR
TYPE(REACTION_TYPE),POINTER :: RN=>NULL()

RN=>REACTION(NR)

IF (RN%TABLE_FS_INDEX>0) THEN
CALL INTERPOLATE2D(RN%TABLE_FS_INDEX,EQ,TMP,LAMINAR_FLAME_SPEED)
ELSE
LAMINAR_FLAME_SPEED = RN%FLAME_SPEED*(TMP/RN%FLAME_SPEED_TEMPERATURE)**RN%FLAME_SPEED_EXPONENT &
*EVALUATE_RAMP(EQ,0._EB,RN%RAMP_FS_INDEX)
ENDIF

END FUNCTION LAMINAR_FLAME_SPEED


SUBROUTINE ZETA_PRODUCTION(DT)
USE MASS, ONLY: SCALAR_FACE_VALUE

REAL(EB), INTENT(IN) :: DT
INTEGER :: I,J,K,IIG,JJG,KKG,IOR,IW,II,JJ,KK
REAL(EB) :: Z_F,DENOM,ZZZ(1:4),DZDX,DZDY,DZDZ
REAL(EB), POINTER, DIMENSION(:,:,:) :: ZFX=>NULL(),ZFY=>NULL(),ZFZ=>NULL(),ZZP=>NULL(),UU=>NULL(),VV=>NULL(),WW=>
      NULL()
TYPE(WALL_TYPE), POINTER :: WC=>NULL()

ZFX =>WORK1
ZFY =>WORK2
ZFZ =>WORK3
ZZP =>WORK4

UU=>U
VV=>V
WW=>W

!$OMP PARALLEL PRIVATE(ZZZ)
!$OMP DO SCHEDULE(STATIC)
DO K=0,KBP1
DO J=0,JBP1
DO I=0,IBP1
ZZP(I,J,K) = ZZ(I,J,K,REACTION(1)%FUEL_SMIX_INDEX)
ENDDO
ENDDO
ENDDO
!$OMP END DO
```

```
! Compute scalar face values

!$OMP DO SCHEDULE(STATIC)
DO K=1,KBAR
DO J=1,JBAR
DO I=1,IBM1
ZZZ(1:4) = ZZP(I-1:I+2,J,K)
ZFX(I,J,K) = SCALAR_FACE_VALUE(UU(I,J,K),ZZZ,FLUX_LIMITER)
ENDDO
ENDDO
ENDDO
!$OMP END DO NOWAIT

!$OMP DO SCHEDULE(STATIC)
DO K=1,KBAR
DO J=1,JBM1
DO I=1,IBAR
ZZZ(1:4) = ZZP(I,J-1:J+2,K)
ZFY(I,J,K) = SCALAR_FACE_VALUE(VV(I,J,K),ZZZ,FLUX_LIMITER)
ENDDO
ENDDO
ENDDO
!$OMP END DO NOWAIT

!$OMP DO SCHEDULE(STATIC)
DO K=1,KBM1
DO J=1,JBAR
DO I=1,IBAR
ZZZ(1:4) = ZZP(I,J,K-1:K+2)
ZFZ(I,J,K) = SCALAR_FACE_VALUE(WW(I,J,K),ZZZ,FLUX_LIMITER)
ENDDO
ENDDO
ENDDO
!$OMP END DO
!$OMP END PARALLEL

WALL_LOOP_2: DO IW=1,N_EXTERNAL_WALL_CELLS+N_INTERNAL_WALL_CELLS
WC=>WALL(IW)
IF (WC%BOUNDARY_TYPE==NULL_BOUNDARY) CYCLE WALL_LOOP_2

II  = WC%ONE_D%II
JJ  = WC%ONE_D%JJ
KK  = WC%ONE_D%KK
IIG = WC%ONE_D%IIG
JJG = WC%ONE_D%JJG
KKG = WC%ONE_D%KKG
IOR = WC%ONE_D%IOR

Z_F = WC%ONE_D%ZZ_F(REACTION(1)%FUEL_SMIX_INDEX)

SELECT CASE(IOR)
CASE( 1); ZFX(IIG-1,JJG,KKG) = Z_F
CASE(-1); ZFX(IIG,JJG,KKG)   = Z_F
CASE( 2); ZFY(IIG,JJG-1,KKG) = Z_F
CASE(-2); ZFY(IIG,JJG,KKG)   = Z_F
CASE( 3); ZFZ(IIG,JJG,KKG-1) = Z_F
CASE(-3); ZFZ(IIG,JJG,KKG)   = Z_F
END SELECT

! Overwrite first off-wall advective flux if flow is away from the wall and if the face is not also a wall cell

OFF_WALL_IF_2: IF (WC%BOUNDARY_TYPE/=INTERPOLATED_BOUNDARY .AND. WC%BOUNDARY_TYPE/=OPEN_BOUNDARY) THEN

OFF_WALL_SELECT_2: SELECT CASE(IOR)
CASE( 1) OFF_WALL_SELECT_2
!        ghost             FX/UU(II+1)
! ///     II     ///    II+1    |    II+2    |  ...
!                             ^ WALL_INDEX(II+1,+1)
IF ((UU(II+1,JJ,KK)>0._EB) .AND. .NOT.(WALL_INDEX(CELL_INDEX(II+1,JJ,KK),+1)>0)) THEN
ZZZ(1:3) = (/Z_F,ZZP(II+1:II+2,JJ,KK)/)
ZFX(II+1,JJ,KK) = SCALAR_FACE_VALUE(UU(II+1,JJ,KK),ZZZ,FLUX_LIMITER)
ENDIF
CASE(-1) OFF_WALL_SELECT_2
!            FX/UU(II-2)       ghost
! ...  |   II-2   |   II-1   ///    II    ///
!                ^ WALL_INDEX(II-1,-1)
IF ((UU(II-2,JJ,KK)<0._EB) .AND. .NOT.(WALL_INDEX(CELL_INDEX(II-1,JJ,KK),-1)>0)) THEN
ZZZ(2:4) = (/ZZP(II-2:II-1,JJ,KK),Z_F/)
ZFX(II-2,JJ,KK) = SCALAR_FACE_VALUE(UU(II-2,JJ,KK),ZZZ,FLUX_LIMITER)
ENDIF
CASE( 2) OFF_WALL_SELECT_2
IF ((VV(II,JJ+1,KK)>0._EB) .AND. .NOT.(WALL_INDEX(CELL_INDEX(II,JJ+1,KK),+2)>0)) THEN
ZZZ(1:3) = (/Z_F,ZZP(II,JJ+1:JJ+2,KK)/)
ZFY(II,JJ+1,KK) = SCALAR_FACE_VALUE(VV(II,JJ+1,KK),ZZZ,FLUX_LIMITER)
ENDIF
CASE(-2) OFF_WALL_SELECT_2
IF ((VV(II,JJ-2,KK)<0._EB) .AND. .NOT.(WALL_INDEX(CELL_INDEX(II,JJ-1,KK),-2)>0)) THEN
```

```
ZZZ(2:4) = (/ZZP(II,JJ-2:JJ-1,KK),Z_F/)
ZFY(II,JJ-2,KK) = SCALAR_FACE_VALUE(VV(II,JJ-2,KK),ZZZ,FLUX_LIMITER)
ENDIF
CASE( 3) OFF_WALL_SELECT_2
IF ((WW(II,JJ,KK+1)>0._EB) .AND. .NOT.(WALL_INDEX(CELL_INDEX(II,JJ,KK+1),+3)>0)) THEN
ZZZ(1:3) = (/Z_F,ZZP(II,JJ,KK+1:KK+2)/)
ZFZ(II,JJ,KK+1) = SCALAR_FACE_VALUE(WW(II,JJ,KK+1),ZZZ,FLUX_LIMITER)
ENDIF
CASE(-3) OFF_WALL_SELECT_2
IF ((WW(II,JJ,KK-2)<0._EB) .AND. .NOT.(WALL_INDEX(CELL_INDEX(II,JJ,KK-1),-3)>0)) THEN
ZZZ(2:4) = (/ZZP(II,JJ,KK-2:KK-1),Z_F/)
ZFZ(II,JJ,KK-2) = SCALAR_FACE_VALUE(WW(II,JJ,KK-2),ZZZ,FLUX_LIMITER)
ENDIF
END SELECT OFF_WALL_SELECT_2

ENDIF OFF_WALL_IF_2

ENDDO WALL_LOOP_2

! Production term

DO K=1,KBAR
DO J=1,JBAR
DO I=1,IBAR
IF (SOLID(CELL_INDEX(I,J,K))) CYCLE

DZDX = (ZFX(I,J,K)-ZFX(I-1,J,K))*RDX(I)
DZDY = (ZFY(I,J,K)-ZFY(I,J-1,K))*RDY(J)
DZDZ = (ZFZ(I,J,K)-ZFZ(I,J,K-1))*RDZ(K)

DENOM = RHO(I,J,K)*( ZZP(I,J,K) - ZZP(I,J,K)**2 )

IF (DENOM>TWO_EPSILON_EB) THEN
! scale sgs variance production
ZETA_SOURCE_TERM(I,J,K) = 2._EB*MU(I,J,K)/SC*( DZDX**2 + DZDY**2 + DZDZ**2 ) / DENOM
ELSE
! cell is pure, unmix
ZETA_SOURCE_TERM(I,J,K) = (1._EB - ZZ(I,J,K,ZETA_INDEX))/DT
ENDIF

ZZ(I,J,K,ZETA_INDEX) = MIN( 1._EB, ZZ(I,J,K,ZETA_INDEX) + DT*ZETA_SOURCE_TERM(I,J,K) )
ENDDO
ENDDO
ENDDO

END SUBROUTINE ZETA_PRODUCTION


! ------------------------------ CCREGION_COMBUSTION ------------------------------

SUBROUTINE CCREGION_COMBUSTION(T,DT,NM)

USE PHYSICAL_FUNCTIONS, ONLY: GET_SPECIFIC_GAS_CONSTANT,GET_MASS_FRACTION_ALL,GET_SPECIFIC_HEAT,
      GET_MOLECULAR_WEIGHT, &
GET_SENSIBLE_ENTHALPY_Z,IS_REALIZABLE,LES_FILTER_WIDTH_FUNCTION
USE COMPLEX_GEOMETRY, ONLY : IBM_CGSC,IBM_GASPHASE

REAL(EB), INTENT(IN) :: T, DT
INTEGER, INTENT(IN) :: NM

! Local Variables:
INTEGER   :: I,J,K,ICC,JCC,NCELL,NS,NR,N,CHEM_SUBIT_TMP
REAL(EB)  :: ZZ_GET(1:N_TRACKED_SPECIES),DZZ(1:N_TRACKED_SPECIES),CP,H_S_N,&
REAC_SOURCE_TERM_TMP(N_TRACKED_SPECIES),Q_REAC_TMP(N_REACTIONS),VCELL
REAL(EB)  :: AIT_P,ZETA_P
LOGICAL   :: Q_EXISTS_CC
TYPE (REACTION_TYPE), POINTER :: RN
TYPE (SPECIES_MIXTURE_TYPE), POINTER :: SM
LOGICAL   :: DO_REACTION,REALIZABLE,DEBUG
LOGICAL :: Q_EXISTS

! Set to zero Reaction, Radiation sources of heat and thermodynamic div:
DO K=1,KBAR
DO J=1,JBAR
DO I=1,IBAR
IF (CCVAR(I,J,K,IBM_CGSC) == IBM_GASPHASE) CYCLE
Q(I,J,K) = 0._EB
QR(I,J,K)= 0._EB
D_SOURCE(I,J,K)= 0._EB
ENDDO
ENDDO
ENDDO

! Now do COMBUSTION_GENERAL for cut-cells.
Q_EXISTS_CC    = .FALSE.

IF (REAC_SOURCE_CHECK) THEN
DO ICC=1,MESHES(NM)%N_CUTCELL_MESH
```

```
DO JCC=1,CUT_CELL(ICC)%NCELL
CUT_CELL(ICC)%Q_REAC(:,JCC) = 0._EB
ENDDO
ENDDO
ENDIF

ZETA_P = 0._EB
DEBUG  = .FALSE.

ICC_LOOP : DO ICC=1,MESHES(NM)%N_CUTCELL_MESH
I      = CUT_CELL(ICC)%IJK(IAXIS)
J      = CUT_CELL(ICC)%IJK(JAXIS)
K      = CUT_CELL(ICC)%IJK(KAXIS)

VCELL = DX(I)*DY(J)*DZ(K)

IF (SOLID(CELL_INDEX(I,J,K))) CYCLE ICC_LOOP ! Cycle in case Cartesian cell inside OBSTS.

NCELL = CUT_CELL(ICC)%NCELL
JCC_LOOP : DO JCC=1,NCELL

! Drop if cut-cell is very small compared to Cartesian cells:
IF ( ABS(CUT_CELL(ICC)%VOLUME(JCC)/VCELL) <  1.E-12_EB ) CYCLE JCC_LOOP

CUT_CELL(ICC)%CHI_R(JCC)     = 0._EB
ZZ_GET = CUT_CELL(ICC)%ZZ(1:N_TRACKED_SPECIES,JCC)

AIT_P = 0._EB
IF (REIGNITION_MODEL) AIT_P = CUT_CELL(ICC)%AIT(JCC)

IF (CHECK_REALIZABILITY) THEN
REALIZABLE=IS_REALIZABLE(ZZ_GET)
IF (.NOT.REALIZABLE) THEN
WRITE(LU_ERR,*) I,J,K
WRITE(LU_ERR,*) ZZ_GET
WRITE(LU_ERR,*) SUM(ZZ_GET)
WRITE(LU_ERR,*) 'ERROR: Unrealizable mass fractions input to COMBUSTION_MODEL'
STOP_STATUS=REALIZABILITY_STOP
ENDIF
ENDIF
CALL CCCHECK_REACTION
IF (.NOT.DO_REACTION) CYCLE ICC_LOOP ! Check whether any reactions are possible.

DZZ = ZZ_GET ! store old ZZ for divergence term
!***************************************************************************************
! Call combustion integration routine for CUT_CELL(ICC)%XX(JCC)
CALL COMBUSTION_MODEL( T,DT,ZZ_GET,CUT_CELL(ICC)%Q(JCC),CUT_CELL(ICC)%MIX_TIME(JCC),&
CUT_CELL(ICC)%CHI_R(JCC),&
CHEM_SUBIT_TMP,REAC_SOURCE_TERM_TMP,Q_REAC_TMP,&
CUT_CELL(ICC)%TMP(JCC),CUT_CELL(ICC)%RHO(JCC),MU(I,J,K),KRES(I,J,K),&
ZETA_P,AIT_P,PBAR(K,PRESSURE_ZONE(I,J,K)),&
LES_FILTER_WIDTH_FUNCTION(DX(I),DY(J),DZ(K)),&
CUT_CELL(ICC)%VOLUME(JCC))
!***************************************************************************************
IF (REAC_SOURCE_CHECK) THEN ! Store special diagnostic quantities
CUT_CELL(ICC)%REAC_SOURCE_TERM(1:N_TRACKED_SPECIES,JCC)=REAC_SOURCE_TERM_TMP(1:N_TRACKED_SPECIES)
CUT_CELL(ICC)%Q_REAC(1:N_REACTIONS,JCC)=Q_REAC_TMP(1:N_REACTIONS)
ENDIF

IF (CHECK_REALIZABILITY) THEN
REALIZABLE=IS_REALIZABLE(ZZ_GET)
IF (.NOT.REALIZABLE) THEN
WRITE(LU_ERR,*) ZZ_GET,SUM(ZZ_GET)
WRITE(LU_ERR,*) 'ERROR: Unrealizable mass fractions after COMBUSTION_MODEL'
STOP_STATUS=REALIZABILITY_STOP
ENDIF
ENDIF

DZZ = ZZ_GET - DZZ

! Update RSUM and ZZ
DZZ_IF: IF ( ANY(ABS(DZZ) > TWO_EPSILON_EB) ) THEN
IF (ABS(CUT_CELL(ICC)%Q(JCC)) > TWO_EPSILON_EB) Q_EXISTS = .TRUE.
! Divergence term
CALL GET_SPECIFIC_HEAT(ZZ_GET,CP,CUT_CELL(ICC)%TMP(JCC))
CALL GET_SPECIFIC_GAS_CONSTANT(ZZ_GET,CUT_CELL(ICC)%RSUM(JCC))
DO N=1,N_TRACKED_SPECIES
SM => SPECIES_MIXTURE(N)
CALL GET_SENSIBLE_ENTHALPY_Z(N,CUT_CELL(ICC)%TMP(JCC),H_S_N)
CUT_CELL(ICC)%D_SOURCE(JCC) = CUT_CELL(ICC)%D_SOURCE(JCC) + &
( SM%RCON/CUT_CELL(ICC)%RSUM(JCC) - H_S_N/(CP*CUT_CELL(ICC)%TMP(JCC)) )*DZZ(N)/DT
CUT_CELL(ICC)%M_DOT_PPP(N,JCC) = CUT_CELL(ICC)%M_DOT_PPP(N,JCC) + &
CUT_CELL(ICC)%RHO(JCC)*DZZ(N)/DT
ENDDO
ENDIF DZZ_IF
ENDDO JCC_LOOP
ENDDO ICC_LOOP

```

```
! This is for plotting regular slices:
DO ICC=1,MESHES(NM)%N_CUTCELL_MESH
I      = CUT_CELL(ICC)%IJK(IAXIS)
J      = CUT_CELL(ICC)%IJK(JAXIS)
K      = CUT_CELL(ICC)%IJK(KAXIS)

VCELL = DX(I)*DY(J)*DZ(K)

IF (SOLID(CELL_INDEX(I,J,K))) CYCLE ! Cycle in case Cartesian cell inside OBSTS.

NCELL = CUT_CELL(ICC)%NCELL
DO JCC=1,NCELL
Q(I,J,K) = Q(I,J,K)+CUT_CELL(ICC)%Q(JCC)*CUT_CELL(ICC)%VOLUME(JCC)
D_SOURCE(I,J,K)=D_SOURCE(I,J,K)+CUT_CELL(ICC)%D_SOURCE(JCC)*CUT_CELL(ICC)%VOLUME(JCC)
M_DOT_PPP(I,J,K,1:N_TOTAL_SCALARS) = M_DOT_PPP(I,J,K,1:N_TOTAL_SCALARS) + &
CUT_CELL(ICC)%M_DOT_PPP(1:N_TOTAL_SCALARS,JCC)*CUT_CELL(ICC)%VOLUME(JCC)
ENDDO
Q(I,J,K) = Q(I,J,K)/VCELL
D_SOURCE(I,J,K)=D_SOURCE(I,J,K)/VCELL
M_DOT_PPP(I,J,K,1:N_TOTAL_SCALARS) = M_DOT_PPP(I,J,K,1:N_TOTAL_SCALARS)/VCELL
ENDDO

RETURN

CONTAINS

SUBROUTINE CCCHECK_REACTION

! Check whether any reactions are possible.

LOGICAL :: REACTANTS_PRESENT

DO_REACTION = .FALSE.
REACTION_LOOP: DO NR=1,N_REACTIONS
RN=>REACTION(NR)
REACTANTS_PRESENT = .TRUE.
DO NS=1,N_TRACKED_SPECIES
IF ( RN%NU(NS) < -TWO_EPSILON_EB .AND. ZZ_GET(NS) < ZZ_MIN_GLOBAL ) THEN
REACTANTS_PRESENT = .FALSE.
EXIT
ENDIF
ENDDO
DO_REACTION = REACTANTS_PRESENT
IF (DO_REACTION) EXIT REACTION_LOOP
ENDDO REACTION_LOOP

END SUBROUTINE CCCHECK_REACTION


END SUBROUTINE CCREGION_COMBUSTION

!added this function for 6.2.0
LOGICAL FUNCTION FUNC_EXTINCT(ZZ_MIXED_IN,TMP_MIXED)
REAL(EB), INTENT(IN) :: ZZ_MIXED_IN(1:N_TRACKED_SPECIES),TMP_MIXED

FUNC_EXTINCT = .FALSE.
IF (ANY(REACTION(:)%FAST_CHEMISTRY)) THEN
SELECT CASE (EXTINCT_MOD)
CASE(EXTINCTION_1)
FUNC_EXTINCT = EXTINCT_1_1(ZZ_MIXED_IN,TMP_MIXED)    !edited name
CASE(EXTINCTION_2)
FUNC_EXTINCT = EXTINCT_2_1(ZZ_MIXED_IN,TMP_MIXED)    !edited name
CASE(EXTINCTION_3)
FUNC_EXTINCT = .FALSE.
END SELECT
ENDIF

END FUNCTION FUNC_EXTINCT


LOGICAL FUNCTION EXTINCT_1_1(ZZ_IN,TMP_MIXED)     !edited name
USE PHYSICAL_FUNCTIONS,ONLY:GET_AVERAGE_SPECIFIC_HEAT
REAL(EB),INTENT(IN)::ZZ_IN(1:N_TRACKED_SPECIES),TMP_MIXED
REAL(EB):: Y_O2,Y_O2_CRIT,CPBAR
INTEGER :: NR
TYPE(REACTION_TYPE),POINTER :: RN=>NULL()

EXTINCT_1_1 = .FALSE.        !edited name
REACTION_LOOP: DO NR=1,N_REACTIONS
RN => REACTION(NR)
IF (.NOT.RN%FAST_CHEMISTRY) CYCLE REACTION_LOOP
AIT_IF: IF (TMP_MIXED < RN%AUTO_IGNITION_TEMPERATURE) THEN
EXTINCT_1_1 = .TRUE.        !edited name
ELSE AIT_IF
CALL GET_AVERAGE_SPECIFIC_HEAT(ZZ_IN,CPBAR,TMP_MIXED)
Y_O2 = ZZ_IN(RN%AIR_SMIX_INDEX)
Y_O2_CRIT = CPBAR*(RN%CRIT_FLAME_TMP-TMP_MIXED)/RN%EPUMO2
IF (Y_O2 < Y_O2_CRIT) EXTINCT_1_1 = .TRUE.     !edited name
```

```
ENDIF AIT_IF
ENDDO REACTION_LOOP

END FUNCTION EXTINCT_1_1      !edited name


LOGICAL FUNCTION EXTINCT_2_1(ZZ_MIXED_IN,TMP_MIXED)      !edited name
USE PHYSICAL_FUNCTIONS,ONLY:GET_SENSIBLE_ENTHALPY
REAL(EB),INTENT(IN)::ZZ_MIXED_IN(1:N_TRACKED_SPECIES),TMP_MIXED
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,&
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
INTEGER :: NR
TYPE(REACTION_TYPE),POINTER :: RN=>NULL()

EXTINCT_2_1 = .FALSE.   !edited name
REACTION_LOOP: DO NR=1,N_REACTIONS
RN => REACTION(NR)
IF (.NOT.RN%FAST_CHEMISTRY) CYCLE REACTION_LOOP
AIT_IF: IF (TMP_MIXED < RN%AUTO_IGNITION_TEMPERATURE) THEN
EXTINCT_2_1 = .TRUE.   !edited name
ELSE AIT_IF
ZZ_F = ZZ_MIXED_IN(RN%FUEL_SMIX_INDEX)
ZZ_A = ZZ_MIXED_IN(RN%AIR_SMIX_INDEX)
ZZ_P = 1._EB - ZZ_F - ZZ_A

ZZ_HAT_F = MIN(ZZ_F,ZZ_MIXED_IN(RN%AIR_SMIX_INDEX)/RN%S) ! burned fuel, FDS Tech Guide (5.16)
ZZ_HAT_A = ZZ_HAT_F*RN%S ! FDS Tech Guide (5.17)
ZZ_HAT_P = (ZZ_HAT_A/(ZZ_A+TWO_EPSILON_EB))*(ZZ_F - ZZ_HAT_F + ZZ_P) ! reactant diluent concentration, FDS Tech Guide (5.18)

! "GET" indicates a composition vector.  Below we are building up the masses of the constituents in the various
! mixtures.  At this point these composition vectors are not normalized.

ZZ_GET_F = 0._EB
ZZ_GET_A = 0._EB
ZZ_GET_P = ZZ_MIXED_IN

ZZ_GET_F(RN%FUEL_SMIX_INDEX) = ZZ_HAT_F ! fuel in reactant mixture composition
ZZ_GET_A(RN%AIR_SMIX_INDEX)  = ZZ_HAT_A ! air  in reactant mixture composition

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 composition
ZZ_GET_P(RN%AIR_SMIX_INDEX)  = MAX(ZZ_GET_P(RN%AIR_SMIX_INDEX) -ZZ_A,0._EB) ! remove all air from product composition

! Normalize concentrations
ZZ_GET_F = ZZ_GET_F/(SUM(ZZ_GET_F)+TWO_EPSILON_EB)
ZZ_GET_A = ZZ_GET_A/(SUM(ZZ_GET_A)+TWO_EPSILON_EB)
ZZ_GET_P = ZZ_GET_P/(SUM(ZZ_GET_P)+TWO_EPSILON_EB)

! Get the specific heat for the fuel and diluent at the current and critical flame temperatures
CALL GET_SENSIBLE_ENTHALPY(ZZ_GET_F,H_F_0,TMP_MIXED)
CALL GET_SENSIBLE_ENTHALPY(ZZ_GET_A,H_A_0,TMP_MIXED)
CALL GET_SENSIBLE_ENTHALPY(ZZ_GET_P,H_P_0,TMP_MIXED)
CALL GET_SENSIBLE_ENTHALPY(ZZ_GET_F,H_F_N,RN%CRIT_FLAME_TMP)
CALL GET_SENSIBLE_ENTHALPY(ZZ_GET_A,H_A_N,RN%CRIT_FLAME_TMP)
CALL GET_SENSIBLE_ENTHALPY(ZZ_GET_P,H_P_N,RN%CRIT_FLAME_TMP)

! See if enough energy is released to raise the fuel and required "air" temperatures above the critical flame temp.
IF ( ZZ_HAT_F*(H_F_0 + RN%HEAT_OF_COMBUSTION) + ZZ_HAT_A*H_A_0 + ZZ_HAT_P*H_P_0 < &
ZZ_HAT_F*H_F_N  + ZZ_HAT_A*H_A_N + ZZ_HAT_P*H_P_N ) EXTINCT_2_1 = .TRUE. ! FDS Tech Guide (5.19)   !edited name
ENDIF AIT_IF
ENDDO REACTION_LOOP

END FUNCTION EXTINCT_2_1    !edited name


LOGICAL FUNCTION EXTINCT_3_1(ZZ_MIXED_IN,TMP_MIXED)   !edited name
USE PHYSICAL_FUNCTIONS,ONLY:GET_SENSIBLE_ENTHALPY
REAL(EB),INTENT(IN)::ZZ_MIXED_IN(1:N_TRACKED_SPECIES),TMP_MIXED
REAL(EB):: H_F_0,H_A_0,H_P_0,H_P_N,Z_F,Z_A,Z_P,Z_A_STOICH,ZZ_HAT_F,ZZ_HAT_A,ZZ_HAT_P,&
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:N_REACTIONS),&
ZZ_GET_PFP(1:N_TRACKED_SPECIES),DZ_F(1:N_REACTIONS),DZ_FRAC_F(1:N_REACTIONS),DZ_F_SUM,&
HOC_EXTINCT,AIT_EXTINCT,CFT_EXTINCT
INTEGER :: NS,NR
TYPE(REACTION_TYPE),POINTER :: RN=>NULL()

EXTINCT_3_1 = .FALSE.      !edited name
Z_F = 0._EB
Z_A = 0._EB
Z_P = 0._EB
DZ_F = 0._EB
DZ_F_SUM = 0._EB
Z_A_STOICH = 0._EB
ZZ_GET_F = 0._EB
```

```
ZZ_GET_A = 0._EB
ZZ_GET_P = ZZ_MIXED_IN
ZZ_GET_PFP = 0._EB
HOC_EXTINCT = 0._EB
AIT_EXTINCT = 0._EB
CFT_EXTINCT = 0._EB

DO NS=1,N_TRACKED_SPECIES
SUM_FUEL_LOOP: DO NR = 1,N_REACTIONS
RN => REACTION(NR)
IF (RN%FAST_CHEMISTRY .AND. RN%HEAT_OF_COMBUSTION > 0._EB .AND. NS == RN%FUEL_SMIX_INDEX) THEN
Z_F = Z_F + ZZ_MIXED_IN(NS)
EXIT SUM_FUEL_LOOP
ENDIF
ENDDO SUM_FUEL_LOOP
SUM_AIR_LOOP: DO NR = 1,N_REACTIONS
RN => REACTION(NR)
IF (RN%FAST_CHEMISTRY .AND. RN%HEAT_OF_COMBUSTION > 0._EB .AND. RN%NU(NS) < 0._EB .AND. NS /= RN%FUEL_SMIX_INDEX)
    THEN
Z_A = Z_A + ZZ_MIXED_IN(NS)
ZZ_GET_P(NS) = MAX(ZZ_GET_P(NS) - ZZ_MIXED_IN(NS),0._EB)
EXIT SUM_AIR_LOOP
ENDIF
ENDDO SUM_AIR_LOOP
ENDDO
Z_P = 1._EB - Z_F - Z_A
DO NR = 1,N_REACTIONS
RN => REACTION(NR)
IF (RN%FAST_CHEMISTRY .AND. RN%HEAT_OF_COMBUSTION > 0._EB) THEN
DZ_F(NR) = 1.E10_EB
DO NS = 1,N_TRACKED_SPECIES
IF (RN%NU(NS) < 0._EB) THEN
DZ_F(NR) = MIN(DZ_F(NR),-ZZ_MIXED_IN(NS)/RN%NU_MW_O_MW_F(NS))
ENDIF
IF (RN%NU(NS) < 0._EB .AND. NS /= RN%FUEL_SMIX_INDEX) THEN
Z_A_STOICH = Z_A_STOICH + ZZ_MIXED_IN(RN%FUEL_SMIX_INDEX)*RN%S
ENDIF
ENDDO
ENDIF
ENDDO
IF (Z_A_STOICH > Z_A) DZ_F_SUM = SUM(DZ_F)
DO NR = 1,N_REACTIONS
RN => REACTION(NR)
IF (Z_A_STOICH > Z_A .AND. RN%HEAT_OF_COMBUSTION > 0._EB) THEN
DZ_FRAC_F(NR) = DZ_F(NR)/MAX(DZ_F_SUM,TWO_EPSILON_EB)
ZZ_GET_F(RN%FUEL_SMIX_INDEX) = DZ_F(NR)*DZ_FRAC_F(NR)
ZZ_GET_P(RN%FUEL_SMIX_INDEX) = ZZ_GET_P(RN%FUEL_SMIX_INDEX) - ZZ_GET_F(RN%FUEL_SMIX_INDEX)
ZZ_GET_PFP(RN%FUEL_SMIX_INDEX) = ZZ_GET_P(RN%FUEL_SMIX_INDEX)
DO NS = 1,N_TRACKED_SPECIES
IF (RN%NU(NS)< 0._EB .AND. NS/=RN%FUEL_SMIX_INDEX) THEN
ZZ_GET_A(NS) = RN%S*ZZ_GET_F(RN%FUEL_SMIX_INDEX)
!           ZZ_GET_P(NS) = ZZ_GET_P(NS) - ZZ_GET_A(NS)
ZZ_GET_PFP(NS) = ZZ_GET_P(NS)
ELSEIF (RN%NU(NS) >= 0._EB ) THEN
ZZ_GET_PFP(NS) = ZZ_GET_P(NS) + ZZ_GET_F(RN%FUEL_SMIX_INDEX)*RN%NU_MW_O_MW_F(NS)
ENDIF
ENDDO
ELSE
ZZ_GET_F(RN%FUEL_SMIX_INDEX) = DZ_F(NR)
ZZ_GET_P(RN%FUEL_SMIX_INDEX) = ZZ_GET_P(RN%FUEL_SMIX_INDEX) - ZZ_GET_F(RN%FUEL_SMIX_INDEX)
ZZ_GET_PFP(RN%FUEL_SMIX_INDEX) = ZZ_GET_P(RN%FUEL_SMIX_INDEX)
DO NS = 1,N_TRACKED_SPECIES
IF (RN%NU(NS) < 0._EB .AND. NS/=RN%FUEL_SMIX_INDEX) THEN
ZZ_GET_A(NS) = RN%S*ZZ_GET_F(RN%FUEL_SMIX_INDEX)
!           ZZ_GET_P(NS) = ZZ_GET_P(NS) - ZZ_GET_A(NS)
ZZ_GET_PFP(NS) = ZZ_GET_P(NS)
ELSEIF (RN%NU(NS) >= 0._EB ) THEN
ZZ_GET_PFP(NS) = ZZ_GET_P(NS) + ZZ_GET_F(RN%FUEL_SMIX_INDEX)*RN%NU_MW_O_MW_F(NS)
ENDIF
ENDDO
ENDIF
ZZ_GET_F_REAC(NR) = ZZ_GET_F(RN%FUEL_SMIX_INDEX)
ENDDO

ZZ_HAT_F = SUM(ZZ_GET_F)
ZZ_HAT_A = SUM(ZZ_GET_A)
ZZ_HAT_P = (ZZ_HAT_A/(Z_A+TWO_EPSILON_EB))*(Z_F-ZZ_HAT_F+SUM(ZZ_GET_P))
!M_P_ST = SUM(ZZ_GET_P)

! Normalize compositions
ZZ_GET_F = ZZ_GET_F/(SUM(ZZ_GET_F)+TWO_EPSILON_EB)
ZZ_GET_F_REAC = ZZ_GET_F_REAC/(SUM(ZZ_GET_F_REAC)+TWO_EPSILON_EB)
ZZ_GET_A = ZZ_GET_A/(SUM(ZZ_GET_A)+TWO_EPSILON_EB)
ZZ_GET_P = ZZ_GET_P/(SUM(ZZ_GET_P)+TWO_EPSILON_EB)
ZZ_GET_PFP = ZZ_GET_PFP/(SUM(ZZ_GET_PFP)+TWO_EPSILON_EB)

DO NR = 1,N_REACTIONS
RN => REACTION(NR)
```

```fortran
AIT_EXTINCT = AIT_EXTINCT+ZZ_GET_F_REAC(NR)*RN%AUTO_IGNITION_TEMPERATURE
CFT_EXTINCT = CFT_EXTINCT+ZZ_GET_F_REAC(NR)*RN%CRIT_FLAME_TMP
HOC_EXTINCT = HOC_EXTINCT+ZZ_GET_F_REAC(NR)*RN%HEAT_OF_COMBUSTION
ENDDO

IF (TMP_MIXED < AIT_EXTINCT) THEN
EXTINCT_3_1 = .TRUE.      !edited name
ELSE
! Get the specific heat for the fuel and diluent at the current and critical flame temperatures
CALL GET_SENSIBLE_ENTHALPY(ZZ_GET_F,H_F_0,TMP_MIXED)
CALL GET_SENSIBLE_ENTHALPY(ZZ_GET_A,H_A_0,TMP_MIXED)
CALL GET_SENSIBLE_ENTHALPY(ZZ_GET_P,H_P_0,TMP_MIXED)
CALL GET_SENSIBLE_ENTHALPY(ZZ_GET_PFP,H_P_N,CFT_EXTINCT)

! See if enough energy is released to raise the fuel and required "air" temperatures above the critical flame
     temp.
IF (ZZ_HAT_F*(H_F_0+HOC_EXTINCT) + ZZ_HAT_A*H_A_0 + ZZ_HAT_P*H_P_0 < &
(ZZ_HAT_F+ZZ_HAT_A+ZZ_HAT_P)*H_P_N) EXTINCT_3_1 = .TRUE. ! FED Tech Guide (5.19)  !edited name
ENDIF

END FUNCTION EXTINCT_3_1    !edited name

!added for 6.2.0
SUBROUTINE GET_REV_fire(MODULE_REV,MODULE_DATE)
INTEGER,INTENT(INOUT) :: MODULE_REV
CHARACTER(255),INTENT(INOUT) :: MODULE_DATE

WRITE(MODULE_DATE,'(A)') firerev(INDEX(firerev,':')+2:LEN_TRIM(firerev)-2)
READ (MODULE_DATE,'(I5)') MODULE_REV
WRITE(MODULE_DATE,'(A)') firedate

END SUBROUTINE GET_REV_fire

END MODULE FIRE
```

A.2 *read.f90*

```fortran
MODULE READ_INPUT

USE PRECISION_PARAMETERS
USE MESH_VARIABLES
USE GLOBAL_CONSTANTS
USE TRAN
USE MESH_POINTERS
USE OUTPUT_DATA
USE COMP_FUNCTIONS, ONLY: SECOND, CHECKREAD, SHUTDOWN, CHECK_XB, SCAN_INPUT_FILE
USE MEMORY_FUNCTIONS, ONLY: ChkMemErr,REALLOCATE2D
USE COMP_FUNCTIONS, ONLY: GET_INPUT_FILE
USE MISC_FUNCTIONS, ONLY: SEARCH_CONTROLLER,WRITE_SUMMARY_INFO
USE EVAC, ONLY: READ_EVAC
USE HVAC_ROUTINES, ONLY: READ_HVAC,PROC_HVAC
USE COMPLEX_GEOMETRY, ONLY: READ_GEOM
USE MPI

!Sesa
USE penalization
USE SCRC

IMPLICIT NONE
PRIVATE

PUBLIC READ_DATA,READ_STOP

CHARACTER(LABEL_LENGTH) :: ID,MB,ODE_SOLVER
CHARACTER(MESSAGE_LENGTH) :: MESSAGE,FYI
CHARACTER(LABEL_LENGTH) :: SURF_DEFAULT='INERT',EVAC_SURF_DEFAULT='INERT',FUEL_RADCAL_ID='METHANE',
     LES_FILTER_WIDTH='null'
LOGICAL :: EX,THICKEN_OBSTRUCTIONS,BAD,IDEAL=.FALSE.,SIMPLE_FUEL_DEFINED=.FALSE.,TARGET_PARTICLES_INCLUDED=.FALSE
     .
REAL(EB) :: XB(6),TEXTURE_ORIGIN(3)
REAL(EB) :: PBX,PBY,PBZ
REAL(EB) :: MW_MIN,MW_MAX
REAL(EB) :: REAC_ATOM_ERROR,REAC_MASS_ERROR,HUMIDITY=-1._EB
INTEGER  :: I,J,K,IZERO,IOS,N_INIT_RESERVED,MAX_LEAK_PATHS,I_DUM(10)
INTEGER :: FUEL_SMIX_INDEX ! Simple chemistry fuel index
TYPE (MESH_TYPE), POINTER :: M=>NULL()
TYPE(OBSTRUCTION_TYPE), POINTER :: OB=>NULL()
TYPE (VENTS_TYPE), POINTER :: VT=>NULL()
TYPE(SURFACE_TYPE), POINTER :: SF=>NULL()
TYPE(MATERIAL_TYPE), POINTER :: ML=>NULL()
TYPE(REACTION_TYPE), POINTER :: RN=>NULL()


CONTAINS
```

```
SUBROUTINE READ_DATA(DT)

REAL(EB) :: DT, T

! Create an array of output QUANTITY names that are included in the various NAMELIST groups

CALL DEFINE_OUTPUT_QUANTITIES

! Get the name of the input file by reading the command line argument

CALL GET_INPUT_FILE

! If no input file is given, just print out the version number and stop

IF (FN_INPUT(1:1)==' ') THEN
IF (MYID==0) THEN
CALL WRITE_SUMMARY_INFO(LU_ERR)
WRITE(LU_ERR,'(/A)') ' Consult FDS Users Guide Chapter, Running FDS, for further instructions.'
WRITE(LU_ERR,'(/A)') ' Hit Enter to Escape...'
READ(5,*,ERR=2,END=2)
ENDIF
2 STOP
ENDIF

! Stop FDS if the input file cannot be found in the current directory

INQUIRE(FILE=FN_INPUT,EXIST=EX)
IF (.NOT.EX) THEN
IF (MYID==0) WRITE(LU_ERR,'(A,A,A)') "ERROR: The file, ", TRIM(FN_INPUT),", does not exist in the current
    directory"
STOP
ENDIF

! Allocate the global orientation vector

N_ORIENTATION_VECTOR = 0
ALLOCATE(ORIENTATION_VECTOR(3,10))

! Set humidity data
CALL CALC_H2O_HV

! Open the input file

OPEN(LU_INPUT,FILE=FN_INPUT,ACTION='READ')

! Read the input file, NAMELIST group by NAMELIST group

CALL READ_DEAD     ; IF (STOP_STATUS==SETUP_STOP) RETURN
CALL READ_HEAD     ; IF (STOP_STATUS==SETUP_STOP) RETURN
CALL READ_MISC     ; IF (STOP_STATUS==SETUP_STOP) RETURN
CALL READ_MULT     ; IF (STOP_STATUS==SETUP_STOP) RETURN
CALL READ_MESH(1)  ; IF (STOP_STATUS==SETUP_STOP) RETURN
CALL READ_EVAC(1)  ; IF (STOP_STATUS==SETUP_STOP) RETURN
CALL READ_MESH(2)  ; IF (STOP_STATUS==SETUP_STOP) RETURN
CALL READ_TRAN     ; IF (STOP_STATUS==SETUP_STOP) RETURN
CALL READ_WIND     ; IF (STOP_STATUS==SETUP_STOP) RETURN
CALL READ_TIME(DT) ; IF (STOP_STATUS==SETUP_STOP) RETURN
CALL READ_PRES     ; IF (STOP_STATUS==SETUP_STOP) RETURN
CALL READ_REAC     ; IF (STOP_STATUS==SETUP_STOP) RETURN
CALL READ_SPEC     ; IF (STOP_STATUS==SETUP_STOP) RETURN
CALL PROC_REAC_1   ; IF (STOP_STATUS==SETUP_STOP) RETURN
CALL READ_RADI     ; IF (STOP_STATUS==SETUP_STOP) RETURN
CALL READ_PROP     ; IF (STOP_STATUS==SETUP_STOP) RETURN
CALL READ_DEVC     ; IF (STOP_STATUS==SETUP_STOP) RETURN
CALL READ_PART     ; IF (STOP_STATUS==SETUP_STOP) RETURN
CALL READ_CTRL     ; IF (STOP_STATUS==SETUP_STOP) RETURN
CALL READ_MATL     ; IF (STOP_STATUS==SETUP_STOP) RETURN
CALL READ_SURF     ; IF (STOP_STATUS==SETUP_STOP) RETURN
CALL READ_CSVF     ; IF (STOP_STATUS==SETUP_STOP) RETURN
CALL READ_OBST     ; IF (STOP_STATUS==SETUP_STOP) RETURN
CALL READ_GEOM     ; IF (STOP_STATUS==SETUP_STOP) RETURN
CALL READ_VENT     ; IF (STOP_STATUS==SETUP_STOP) RETURN
CALL READ_ZONE     ; IF (STOP_STATUS==SETUP_STOP) RETURN
CALL READ_EVAC(2)  ; IF (STOP_STATUS==SETUP_STOP) RETURN
CALL READ_HVAC     ; IF (STOP_STATUS==SETUP_STOP) RETURN
CALL PROC_SURF_1   ; IF (STOP_STATUS==SETUP_STOP) RETURN
CALL READ_RAMP     ; IF (STOP_STATUS==SETUP_STOP) RETURN
CALL PROC_WIND     ; IF (STOP_STATUS==SETUP_STOP) RETURN
CALL PROC_SMIX     ; IF (STOP_STATUS==SETUP_STOP) RETURN
CALL PROC_REAC_2   ; IF (STOP_STATUS==SETUP_STOP) RETURN
CALL PROC_HVAC     ; IF (STOP_STATUS==SETUP_STOP) RETURN
CALL PROC_MATL     ; IF (STOP_STATUS==SETUP_STOP) RETURN
CALL PROC_SURF_2   ; IF (STOP_STATUS==SETUP_STOP) RETURN
CALL READ_DUMP     ; IF (STOP_STATUS==SETUP_STOP) RETURN
CALL READ_CLIP     ; IF (STOP_STATUS==SETUP_STOP) RETURN
CALL PROC_WALL     ; IF (STOP_STATUS==SETUP_STOP) RETURN
```

```
CALL PROC_PART     ; IF (STOP_STATUS==SETUP_STOP) RETURN
CALL READ_INIT     ; IF (STOP_STATUS==SETUP_STOP) RETURN
CALL READ_TABL     ; IF (STOP_STATUS==SETUP_STOP) RETURN
CALL PROC_CTRL     ; IF (STOP_STATUS==SETUP_STOP) RETURN
CALL PROC_PROP     ; IF (STOP_STATUS==SETUP_STOP) RETURN
CALL PROC_DEVC(DT) ; IF (STOP_STATUS==SETUP_STOP) RETURN
CALL PROC_OBST     ; IF (STOP_STATUS==SETUP_STOP) RETURN
CALL READ_PROF     ; IF (STOP_STATUS==SETUP_STOP) RETURN
CALL READ_SLCF     ; IF (STOP_STATUS==SETUP_STOP) RETURN
CALL READ_ISOF     ; IF (STOP_STATUS==SETUP_STOP) RETURN
CALL READ_BNDF     ; IF (STOP_STATUS==SETUP_STOP) RETURN
CALL READ_BNDE     ; IF (STOP_STATUS==SETUP_STOP) RETURN

!Sesa
CALL read_pen
CALL read_trunks

! Close the input file, and never open it again

CLOSE (LU_INPUT)

! Set QUANTITY ambient values

CALL SET_QUANTITIES_AMBIENT

END SUBROUTINE READ_DATA


SUBROUTINE READ_DEAD

CHARACTER(80) :: BAD_TEXT

! Look for hidden carriage return characters at the beginning of namelist input lines.

REWIND(LU_INPUT) ; INPUT_FILE_LINE_NUMBER = 0
CALL SCAN_INPUT_FILE(LU_INPUT,IOS,BAD_TEXT)
IF (IOS==0) THEN
WRITE(MESSAGE,'(3A)') 'ERROR: Hidden carriage return character in line starting with: ',BAD_TEXT(2:15),'...'
CALL SHUTDOWN(MESSAGE)
ENDIF

! Look for outdated NAMELIST groups and stop the run if any are found.

REWIND(LU_INPUT) ; INPUT_FILE_LINE_NUMBER = 0
CALL CHECKREAD('GRID',LU_INPUT,IOS)
IF (IOS==0) CALL SHUTDOWN('ERROR: GRID is no longer a valid NAMELIST group. Read User Guide discussion on MESH.')
REWIND(LU_INPUT) ; INPUT_FILE_LINE_NUMBER = 0
CALL CHECKREAD('HEAT',LU_INPUT,IOS)
IF (IOS==0) CALL SHUTDOWN('ERROR: HEAT is no longer a valid NAMELIST group. Read User Guide discussion on PROP
    and DEVC.')
REWIND(LU_INPUT) ; INPUT_FILE_LINE_NUMBER = 0
CALL CHECKREAD('PDIM',LU_INPUT,IOS)
IF (IOS==0) CALL SHUTDOWN('ERROR: PDIM is no longer a valid NAMELIST group. Read User Guide discussion on MESH.')
REWIND(LU_INPUT) ; INPUT_FILE_LINE_NUMBER = 0
CALL CHECKREAD('PIPE',LU_INPUT,IOS)
IF (IOS==0) CALL SHUTDOWN('ERROR: PIPE is no longer a valid NAMELIST group. Read User Guide discussion on PROP
    and DEVC.')
REWIND(LU_INPUT) ; INPUT_FILE_LINE_NUMBER = 0
CALL CHECKREAD('PL3D',LU_INPUT,IOS)
IF (IOS==0) CALL SHUTDOWN('ERROR: PL3D is no longer a valid NAMELIST group. Read User Guide discussion on DUMP.')
REWIND(LU_INPUT) ; INPUT_FILE_LINE_NUMBER = 0
CALL CHECKREAD('SMOD',LU_INPUT,IOS)
IF (IOS==0) CALL SHUTDOWN('ERROR: SMOD is no longer a valid NAMELIST group. Read User Guide discussion on DEVC.')
REWIND(LU_INPUT) ; INPUT_FILE_LINE_NUMBER = 0
CALL CHECKREAD('SPRK',LU_INPUT,IOS)
IF (IOS==0) CALL SHUTDOWN('ERROR: SPRK is no longer a valid NAMELIST group. Read User Guide discussion on PROP
    and DEVC.')
REWIND(LU_INPUT) ; INPUT_FILE_LINE_NUMBER = 0
CALL CHECKREAD('THCP',LU_INPUT,IOS)
IF (IOS==0) CALL SHUTDOWN('ERROR: THCP is no longer a valid NAMELIST group. Read User Guide discussion on DEVC.')

REWIND(LU_INPUT) ; INPUT_FILE_LINE_NUMBER = 0

END SUBROUTINE READ_DEAD


SUBROUTINE READ_HEAD
INTEGER :: NAMELENGTH
NAMELIST /HEAD/ CHID,FYI,STOPFDS,TITLE

CHID    = 'null'
TITLE   = '        '
STOPFDS=-1

REWIND(LU_INPUT) ; INPUT_FILE_LINE_NUMBER = 0
HEAD_LOOP: DO
CALL CHECKREAD('HEAD',LU_INPUT,IOS)
IF (IOS==1) EXIT HEAD_LOOP
```

```fortran
READ(LU_INPUT,HEAD,END=13,ERR=14,IOSTAT=IOS)
14 IF (IOS>0) THEN ; CALL SHUTDOWN('ERROR: Problem with HEAD line') ; RETURN ; ENDIF
ENDDO HEAD_LOOP
13 REWIND(LU_INPUT) ; INPUT_FILE_LINE_NUMBER = 0

CLOOP: DO I=1,39
IF (CHID(I:I)=='.') THEN ; CALL SHUTDOWN('ERROR: No periods allowed in CHID') ; RETURN ; ENDIF
IF (CHID(I:I)==' ') EXIT CLOOP
ENDDO CLOOP

IF (TRIM(CHID)=='null') THEN
NAMELENGTH = LEN_TRIM(FN_INPUT)
ROOTNAME: DO I=NAMELENGTH,2,-1
IF (FN_INPUT(I:I)=='.') THEN
WRITE(CHID,'(A)') FN_INPUT(1:I-1)
EXIT ROOTNAME
ENDIF
END DO ROOTNAME
ENDIF

! Define and look for a stop file

FN_STOP = TRIM(CHID)//'.stop'
INQUIRE(FILE=FN_STOP,EXIST=EX)
IF (EX) THEN
STOP_AT_ITER=READ_STOP() ! READ_STOP() returns 0 if there is nothing in the .stop file
IF (STOP_AT_ITER<=0) THEN
WRITE(MESSAGE,'(A,A,A)') "ERROR: Remove the file, ",TRIM(FN_STOP),", from the current directory"
CALL SHUTDOWN(MESSAGE) ; RETURN
ELSE
WRITE(LU_ERR,'(A,A,A)') "NOTE: The file, ",TRIM(FN_STOP),", was detected."
WRITE(LU_ERR,'(A,I0,A)')"This FDS run will stop after ",STOP_AT_ITER," iterations."
ENDIF
ELSE
IF(STOPFDS>=0) THEN
STOP_AT_ITER = STOPFDS
WRITE(LU_ERR,'(A,A,A)') "NOTE: The STOPFDS keyword was detected on the &HEAD line."
WRITE(LU_ERR,'(A,I0,A)')"This FDS run will stop after ",STOP_AT_ITER," iterations."
ENDIF
ENDIF

END SUBROUTINE READ_HEAD


INTEGER FUNCTION READ_STOP()

! if a stop file exists and it contains a positive integer then
! stop the fds run at when it computes that number of iterations

INTEGER :: IERROR

READ_STOP=0

! this routine is only called if the stop file exists

OPEN(UNIT=LU_STOP,FILE=FN_STOP,FORM='FORMATTED',STATUS='OLD',IOSTAT=IERROR)
IF (IERROR==0) THEN
READ(LU_STOP,'(I5)',END=10,IOSTAT=IERROR) READ_STOP
IF (IERROR/=0) READ_STOP=0
ENDIF
10 CLOSE(LU_STOP)

END FUNCTION READ_STOP


SUBROUTINE READ_MESH(IMODE)
USE GLOBAL_CONSTANTS, ONLY : OPENMP_USED_THREADS, OPENMP_USER_SET_THREADS, USE_OPENMP
USE EVAC, ONLY: N_DOORS, N_EXITS, N_CO_EXITS, EVAC_EMESH_EXITS_TYPE, EMESH_EXITS, EMESH_ID, EMESH_IJK, EMESH_XB,
     &
EMESH_NM, N_DOOR_MESHES, EMESH_NFIELDS, HUMAN_SMOKE_HEIGHT, EVAC_DELTA_SEE, &
EMESH_STAIRS, EVAC_EMESH_STAIRS_TYPE, N_STRS, INPUT_EVAC_GRIDS, NO_EVAC_MESHES
INTEGER, INTENT(IN) :: IMODE
INTEGER :: IJK(3),NM,NM2,CURRENT_MPI_PROCESS,MPI_PROCESS,RGB(3),LEVEL,N_MESH_NEW,N,II,JJ,KK,NMESHES_READ,NNN,
     NEVAC_MESHES,IERR, &
NMESHES_EVAC, NMESHES_FIRE, NM_EVAC, N_THREADS
INTEGER, ALLOCATABLE, DIMENSION(:) :: NEIGHBOR_LIST
LOGICAL :: EVACUATION, EVAC_HUMANS
REAL(EB) :: EVAC_Z_OFFSET,XB1,XB2,XB3,XB4,XB5,XB6
CHARACTER(25) :: COLOR
CHARACTER(LABEL_LENGTH) :: MULT_ID,PERIODIC_MESH_IDS(3)
NAMELIST /MESH/ COLOR,CYLINDRICAL,EVACUATION,EVAC_HUMANS,EVAC_Z_OFFSET, FYI,ID,IJK,LEVEL,MPI_PROCESS,MULT_ID,
     PERIODIC_MESH_IDS,&
RGB,XB,N_THREADS
TYPE (MESH_TYPE), POINTER :: M,M2
TYPE (MULTIPLIER_TYPE), POINTER :: MR

NMESHES = 0
NMESHES_READ = 0
```

```
NMESHES_EVAC = 0
NMESHES_FIRE = 0
NEVAC_MESHES = 0
IF (IMODE==1) THEN
NO_EVAC_MESHES = .TRUE.
INPUT_EVAC_GRIDS   = 0
IF(NO_EVACUATION) THEN
N_EVAC = 0
RETURN
END IF
END IF

REWIND(LU_INPUT) ; INPUT_FILE_LINE_NUMBER = 0
COUNT_MESH_LOOP: DO
CALL CHECKREAD('MESH',LU_INPUT,IOS)
IF (IOS==1) EXIT COUNT_MESH_LOOP
MULT_ID = 'null'
EVACUATION  = .FALSE.
EVAC_HUMANS  = .FALSE.
READ(LU_INPUT,MESH,END=15,ERR=16,IOSTAT=IOS)
NMESHES_READ = NMESHES_READ + 1
IF (NO_EVACUATION .AND. EVACUATION) CYCLE COUNT_MESH_LOOP ! skip evacuation meshes
IF (EVACUATION_DRILL .AND. .NOT.EVACUATION) CYCLE COUNT_MESH_LOOP ! skip fire meshes
IF (EVACUATION_MC_MODE .AND. .NOT.EVACUATION) CYCLE COUNT_MESH_LOOP ! skip fire meshes
IF (EVACUATION) NEVAC_MESHES = NEVAC_MESHES + 1
IF (IMODE==1 .AND. EVAC_HUMANS) NO_EVAC_MESHES = .FALSE.
IF (IMODE==1 .AND. EVAC_HUMANS) INPUT_EVAC_GRIDS = INPUT_EVAC_GRIDS + 1
N_MESH_NEW = 0
IF (MULT_ID=='null') THEN
N_MESH_NEW = 1
ELSE
DO N=1,N_MULT
MR => MULTIPLIER(N)
IF (MULT_ID==MR%ID) N_MESH_NEW = MR%N_COPIES
ENDDO
IF (N_MESH_NEW==0) THEN
WRITE(MESSAGE,'(A,A,A,I0)') 'ERROR: MULT line ', TRIM(MULT_ID),' not found on MESH line', &
NMESHES_READ
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF
ENDIF
NMESHES       = NMESHES + N_MESH_NEW
IF (.NOT.EVACUATION) NMESHES_FIRE =  NMESHES_FIRE + N_MESH_NEW
16 IF (IOS>0) THEN ; CALL SHUTDOWN('ERROR: Problem with MESH line.') ; RETURN ; ENDIF
ENDDO COUNT_MESH_LOOP
15 CONTINUE

EVAC_MODE_IF: IF (IMODE==1) THEN
REWIND(LU_INPUT) ; INPUT_FILE_LINE_NUMBER = 0
IF (NO_EVAC_MESHES) THEN
NO_EVACUATION        = .TRUE.
EVACUATION_DRILL     = .FALSE.
EVACUATION_MC_MODE = .FALSE.
N_EVAC               = 0
RETURN
END IF
ALLOCATE(EMESH_ID(      MAX(1,INPUT_EVAC_GRIDS)), STAT=IZERO)
CALL ChkMemErr('READ_EVAC','EMESH_ID',IZERO)
ALLOCATE(EMESH_XB(6,   MAX(1,INPUT_EVAC_GRIDS)), STAT=IZERO)
CALL ChkMemErr('READ_EVAC','EMESH_XB',IZERO)
ALLOCATE(EMESH_IJK(3,   MAX(1,INPUT_EVAC_GRIDS)), STAT=IZERO)
CALL ChkMemErr('READ_EVAC','EMESH_IJK',IZERO)

NM = 0
EVAC_MESH_LOOP: DO N = 1, NMESHES_READ
! Set evacuation MESH defaults
IJK(1)= 10
IJK(2)= 10
IJK(3)= 1
XB(1) = 0._EB
XB(2) = 1._EB
XB(3) = 0._EB
XB(4) = 1._EB
XB(5) = 0._EB
XB(6) = 1._EB
RGB   = -1
COLOR    = 'null'
ID       = 'null'
EVACUATION  = .FALSE.
EVAC_HUMANS = .FALSE.
! Read the MESH line
CALL CHECKREAD('MESH', LU_INPUT, IOS)
IF (IOS==1) EXIT EVAC_MESH_LOOP
READ(LU_INPUT, MESH)
IF (.NOT.EVACUATION)                      CYCLE EVAC_MESH_LOOP ! skip fire meshes
IF (.NOT.EVAC_HUMANS .AND. EVACUATION) CYCLE EVAC_MESH_LOOP ! skip additional evac meshes
NM = NM + 1
! Reorder XB coordinates if necessary
```

```fortran
CALL CHECK_XB(XB)
EMESH_ID(NM)      = TRIM(ID)
EMESH_IJK(1,NM)  = IJK(1)
EMESH_IJK(2,NM)  = IJK(2)
EMESH_IJK(3,NM)  = IJK(3)
EMESH_XB(1,NM)   = XB(1)
EMESH_XB(2,NM)   = XB(2)
EMESH_XB(3,NM)   = XB(3)
EMESH_XB(4,NM)   = XB(4)
EMESH_XB(5,NM)   = XB(5)
EMESH_XB(6,NM)   = XB(6)
END DO EVAC_MESH_LOOP
REWIND(LU_INPUT) ; INPUT_FILE_LINE_NUMBER = 0
RETURN
END IF EVAC_MODE_IF

IF (.NOT. NO_EVACUATION) NMESHES = NMESHES + N_DOOR_MESHES + NEVAC_MESHES
IF (.NOT. NO_EVACUATION) NMESHES = NMESHES + N_STRS

NMESHES_EVAC = NMESHES - NMESHES_FIRE

! Stop the calculation if the number of MPI processes is greater than the number of meshes

IF (NO_EVACUATION) THEN
IF (NMESHES<N_MPI_PROCESSES) THEN
CALL MPI_FINALIZE(IERR)
WRITE(MESSAGE,'(A,I0,A,I0)') 'ERROR: The number of MPI processes, ',N_MPI_PROCESSES,', exceeds the number of
     meshes, ',NMESHES
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF
ELSE
IF(NMESHES_FIRE+1<N_MPI_PROCESSES) THEN
CALL MPI_FINALIZE(IERR)
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
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF
ENDIF

! Allocate parameters associated with the mesh.

ALLOCATE(MESHES(NMESHES),STAT=IZERO)
CALL ChkMemErr('READ','MESHES',IZERO)
ALLOCATE(PROCESS(NMESHES),STAT=IZERO)
CALL ChkMemErr('READ','PROCESS',IZERO)
ALLOCATE(MESH_NAME(NMESHES),STAT=IZERO)
CALL ChkMemErr('READ','MESH_NAME',IZERO)
ALLOCATE(PERIODIC_MESH_NAMES(NMESHES,3),STAT=IZERO)
CALL ChkMemErr('READ','PERIODIC_MESH_NAMES',IZERO)
ALLOCATE(CHANGE_TIME_STEP_INDEX(NMESHES),STAT=IZERO)
CALL ChkMemErr('READ','CHANGE_TIME_STEP_INDEX',IZERO)
CHANGE_TIME_STEP_INDEX = 0
ALLOCATE(EVACUATION_ONLY(NMESHES),STAT=IZERO)
CALL ChkMemErr('READ','EVACUATION_ONLY',IZERO)
EVACUATION_ONLY(1:NMESHES_FIRE) = .FALSE.
IF (NMESHES_FIRE<NMESHES) EVACUATION_ONLY(NMESHES_FIRE+1:NMESHES) = .TRUE.
ALLOCATE(EVACUATION_SKIP(NMESHES),STAT=IZERO)
CALL ChkMemErr('READ','EVACUATION_SKIP',IZERO)
EVACUATION_SKIP = .FALSE.
ALLOCATE(EVACUATION_Z_OFFSET(NMESHES),STAT=IZERO)
CALL ChkMemErr('READ','EVACUATION_Z_OFFSET',IZERO)
EVACUATION_Z_OFFSET = 1.0_EB

! Read in the Mesh lines from Input file

REWIND(LU_INPUT) ; INPUT_FILE_LINE_NUMBER = 0

IF (NMESHES<1) THEN ; CALL SHUTDOWN('ERROR: No MESH line(s) defined.') ; RETURN ; ENDIF

NM = 0

MESH_LOOP: DO N=1,NMESHES_READ

! Set MESH defaults

IJK(1)= 10
IJK(2)= 10
IJK(3)= 10
TWO_D = .FALSE.
XB(1) = 0._EB
XB(2) = 1._EB
XB(3) = 0._EB
XB(4) = 1._EB
XB(5) = 0._EB
XB(6) = 1._EB
RGB   = -1
COLOR = 'null'
CYLINDRICAL = .FALSE.
```

```
ID = 'null'
EVACUATION   = .FALSE.
EVAC_Z_OFFSET = 1.0_EB
EVAC_HUMANS = .FALSE.
MPI_PROCESS = -1
LEVEL = 0
MULT_ID = 'null'
PERIODIC_MESH_IDS = 'pnull'
N_THREADS = -1

! Read the MESH line

CALL CHECKREAD('MESH',LU_INPUT,IOS)
IF (IOS==1) EXIT MESH_LOOP
READ(LU_INPUT,MESH)

IF (NO_EVACUATION .AND. EVACUATION) CYCLE MESH_LOOP ! skip evacuation meshes
IF (EVACUATION_DRILL .AND. .NOT.EVACUATION) CYCLE MESH_LOOP ! skip fire meshes
IF (EVACUATION_MC_MODE .AND. .NOT.EVACUATION) CYCLE MESH_LOOP ! skip fire meshes

! Reorder XB coordinates if necessary

CALL CHECK_XB(XB)

! Multiply meshes if need be

MR => MULTIPLIER(0)
DO NNN=1,N_MULT
IF (MULT_ID==MULTIPLIER(NNN)%ID) MR => MULTIPLIER(NNN)
ENDDO

K_MULT_LOOP: DO KK=MR%K_LOWER,MR%K_UPPER
J_MULT_LOOP: DO JJ=MR%J_LOWER,MR%J_UPPER
I_MULT_LOOP: DO II=MR%I_LOWER,MR%I_UPPER

IF (.NOT.MR%SEQUENTIAL) THEN
XB1 = XB(1) + MR%DX0 + II*MR%DXB(1)
XB2 = XB(2) + MR%DX0 + II*MR%DXB(2)
XB3 = XB(3) + MR%DY0 + JJ*MR%DXB(3)
XB4 = XB(4) + MR%DY0 + JJ*MR%DXB(4)
XB5 = XB(5) + MR%DZ0 + KK*MR%DXB(5)
XB6 = XB(6) + MR%DZ0 + KK*MR%DXB(6)
ELSE
XB1 = XB(1) + MR%DX0 + II*MR%DXB(1)
XB2 = XB(2) + MR%DX0 + II*MR%DXB(2)
XB3 = XB(3) + MR%DY0 + II*MR%DXB(3)
XB4 = XB(4) + MR%DY0 + II*MR%DXB(4)
XB5 = XB(5) + MR%DZ0 + II*MR%DXB(5)
XB6 = XB(6) + MR%DZ0 + II*MR%DXB(6)
ENDIF

! Increase the MESH counter by 1

NM = NM + 1

! Determine which PROCESS to assign the MESH to

IF (MPI_PROCESS>-1) THEN
CURRENT_MPI_PROCESS = MPI_PROCESS
IF (CURRENT_MPI_PROCESS>N_MPI_PROCESSES-1) THEN
IF (N_MPI_PROCESSES > 1) THEN
WRITE(MESSAGE,'(A,I0,A)') 'ERROR: MPI_PROCESS for MESH ',NM,' greater than total number of processes'
CALL SHUTDOWN(MESSAGE) ; RETURN
ELSE
! Prevents fatal error when testing a run on a single core with MPI_PROCESS set for meshes
WRITE(MESSAGE,'(A,I0,A)') 'WARNING: MPI_PROCESS set for MESH ',NM,' and only one MPI process exists'
IF (MYID==0) WRITE(LU_ERR,'(A)') TRIM(MESSAGE)
CURRENT_MPI_PROCESS=0
ENDIF
ENDIF
ELSE
CURRENT_MPI_PROCESS = MIN(NM-1,N_MPI_PROCESSES-1)
ENDIF

! Fill in MESH related variables

M => MESHES(NM)
M%MESH_LEVEL = LEVEL
M%IBAR = IJK(1)
M%JBAR = IJK(2)
M%KBAR = IJK(3)
IBAR_MAX = MAX(IBAR_MAX,M%IBAR)
JBAR_MAX = MAX(JBAR_MAX,M%JBAR)
KBAR_MAX = MAX(KBAR_MAX,M%KBAR)
M%N_EXTERNAL_WALL_CELLS = 2*M%IBAR*M%JBAR+2*M%IBAR*M%KBAR+2*M%JBAR*M%KBAR

IF (EVACUATION) EVACUATION_ONLY(NM) = .TRUE.
IF (EVAC_HUMANS) EVACUATION_SKIP(NM) = .TRUE.
```

```fortran
IF (EVACUATION)   EVACUATION_Z_OFFSET(NM) = EVAC_Z_OFFSET
IF (EVACUATION)   M%N_EXTERNAL_WALL_CELLS = 2*M%IBAR*M%KBAR+2*M%JBAR*M%KBAR
IF (EVACUATION .AND. .NOT.EVAC_HUMANS) THEN
WRITE(MESSAGE,'(A)') 'ERROR: NO DOOR FLOW EVACUATION MESHES IN FDS6'
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF

IF (M%JBAR==1) TWO_D = .TRUE.
IF (TWO_D .AND. M%JBAR/=1) THEN
WRITE(MESSAGE,'(A)') 'ERROR: IJK(2) must be 1 for all grids in 2D Calculation'
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF
IF (EVACUATION .AND. M%KBAR/=1) THEN
WRITE(MESSAGE,'(A)') 'ERROR: IJK(3) must be 1 for all evacuation grids'
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF

! Associate the MESH with the PROCESS

IF (MYID==CURRENT_MPI_PROCESS) THEN
LOWER_MESH_INDEX = MIN(LOWER_MESH_INDEX,NM)
UPPER_MESH_INDEX = MAX(UPPER_MESH_INDEX,NM)
ENDIF

PROCESS(NM) = CURRENT_MPI_PROCESS
IF (MYID==0 .AND. VERBOSE) &
WRITE(LU_ERR,'(A,I0,A,I0)') ' Mesh ',NM,' is assigned to MPI Process ',PROCESS(NM)
IF (EVACUATION_ONLY(NM) .AND. (N_MPI_PROCESSES>1)) EVAC_PROCESS = N_MPI_PROCESSES-1

! Check the number of OMP threads for a valid value (positive, larger than 0), -1 indicates default unchanged
    value
IF (N_THREADS < 1 .AND. N_THREADS /= -1) THEN
WRITE(MESSAGE, '(A)') 'ERROR: N_THREADS must be at least 1'
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF

! If OMP number of threads is explicitly set for this mesh and the mesh is assigned to this MPI process,
! then set this value
IF (MYID == PROCESS(NM) .AND. N_THREADS > 0) THEN
! Check if OPENMP is active
IF (USE_OPENMP .NEQV. .TRUE.) THEN
WRITE(MESSAGE, '(A)') 'ERROR: setting N_THREADS, but OPENMP is not active'
CALL SHUTDOWN(MESSAGE) ; RETURN
END IF

! Check if the process' thread number was already set in a previous mesh definition
IF (OPENMP_USER_SET_THREADS .EQV. .TRUE.) THEN
! Check if previous definitions are consistent
IF (N_THREADS .NE. OPENMP_USED_THREADS) THEN
WRITE(MESSAGE, '(A)') 'ERROR: N_THREADS not consistent for MPI process'
CALL SHUTDOWN(MESSAGE) ; RETURN
END IF
END IF

! set the value-changed-flag and the new thread number
OPENMP_USER_SET_THREADS = .TRUE.
OPENMP_USED_THREADS     = N_THREADS
END IF

! Mesh boundary colors

IF (ANY(RGB<0) .AND. COLOR=='null') COLOR = 'BLACK'
IF (COLOR /= 'null') CALL COLOR2RGB(RGB,COLOR)
ALLOCATE(M%RGB(3))
M%RGB = RGB

! Mesh Geometry and Name

PERIODIC_MESH_NAMES(NM,:) = PERIODIC_MESH_IDS(:)
WRITE(MESH_NAME(NM),'(A,I7.7)') 'MESH_',NM
IF (ID/='null') MESH_NAME(NM) = ID

! Process Physical Coordinates

IF (XB2-XB1<TWO_EPSILON_EB) THEN
WRITE(MESSAGE,'(A,I0)') 'ERROR: XMIN > XMAX on MESH ', NM
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF
IF (XB4-XB3<TWO_EPSILON_EB) THEN
WRITE(MESSAGE,'(A,I0)') 'ERROR: YMIN > YMAX on MESH ', NM
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF
IF (XB6-XB5<TWO_EPSILON_EB) THEN
WRITE(MESSAGE,'(A,I0)') 'ERROR: ZMIN > ZMAX on MESH ', NM
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF
IF (EVACUATION .AND. ABS(XB5 - XB6) <= SPACING(XB(6))) THEN
WRITE(MESSAGE,'(A,I0)') 'ERROR: ZMIN = ZMAX on evacuation MESH ', NM
```

```
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF

M%XS     = XB1
M%XF     = XB2
M%YS     = XB3
M%YF     = XB4
M%ZS     = XB5
M%ZF     = XB6
IF (.NOT.EVACUATION ) THEN
XS_MIN   = MIN(XS_MIN,M%XS)
XF_MAX   = MAX(XF_MAX,M%XF)
YS_MIN   = MIN(YS_MIN,M%YS)
YF_MAX   = MAX(YF_MAX,M%YF)
ZS_MIN   = MIN(ZS_MIN,M%ZS)
ZF_MAX   = MAX(ZF_MAX,M%ZF)
ENDIF
M%DXI    = (M%XF-M%XS)/REAL(M%IBAR,EB)
M%DETA   = (M%YF-M%YS)/REAL(M%JBAR,EB)
M%DZETA  = (M%ZF-M%ZS)/REAL(M%KBAR,EB)
M%RDXI   = 1._EB/M%DXI
M%RDETA  = 1._EB/M%DETA
M%RDZETA= 1._EB/M%DZETA
M%IBM1   = M%IBAR-1
M%JBM1   = M%JBAR-1
M%KBM1   = M%KBAR-1
M%IBP1   = M%IBAR+1
M%JBP1   = M%JBAR+1
M%KBP1   = M%KBAR+1

IF (TWO_D) THEN
M%CELL_SIZE = SQRT(M%DXI*M%DZETA)
ELSE
M%CELL_SIZE = (M%DXI*M%DETA*M%DZETA)**ONTH
ENDIF

IF (.NOT.EVACUATION_ONLY(NM)) CHARACTERISTIC_CELL_SIZE = MIN( CHARACTERISTIC_CELL_SIZE , M%CELL_SIZE )

ENDDO I_MULT_LOOP
ENDDO J_MULT_LOOP
ENDDO K_MULT_LOOP

ENDDO MESH_LOOP

NMEVAC = NM

! Check for bad mesh ordering if MPI_PROCESS used

DO NM=1,NMESHES
IF (NM==1) CYCLE
IF (EVACUATION_ONLY(NM)) CYCLE
IF (PROCESS(NM) < PROCESS(NM-1)) THEN
WRITE(MESSAGE,'(A,I0,A,I0,A)') 'ERROR: MPI_PROCESS for MESH ', NM,' < MPI_PROCESS for MESH ',NM-1,&
'. Reorder MESH lines.'
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF
ENDDO
DO NM=1,NMESHES
IF (NM==1 .OR. .NOT.EVACUATION_ONLY(NM)) CYCLE
IF (.NOT.EVACUATION_SKIP(NM)) CYCLE
IF (PROCESS(NM) < PROCESS(NM-1)) THEN
WRITE(MESSAGE,'(A,I0,A,I0,A)') 'ERROR: MPI_PROCESS for evacuation MESH ', NM,' < MPI_PROCESS for MESH ',NM-1,&
'. Reorder MESH lines.'
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF
ENDDO

!Sesa-added as in 6.2.0
! Allocation for mean forcing (required here, instead of init, because of hole feature)

IF (ANY(MEAN_FORCING)) THEN
DO NM=1,NMESHES
M=>MESHES(NM)
ALLOCATE(M%MEAN_FORCING_CELL(0:M%IBP1,0:M%JBP1,0:M%KBP1),STAT=IZERO)
CALL ChkMemErr('INIT','MEAN_FORCING_CELL',IZERO)
M%MEAN_FORCING_CELL=.TRUE.
ENDDO
ENDIF

! Min and Max values of temperature

TMPMIN = MAX(1._EB , MIN(TMPA,TMPM)-10._EB)
IF (LAPSE_RATE < 0._EB) TMPMIN = MIN(TMPMIN,TMPA+LAPSE_RATE*ZF_MAX)
TMPMAX = 3000._EB

REWIND(LU_INPUT) ; INPUT_FILE_LINE_NUMBER = 0
!Sesa-addition end
```

```

! Define the additional evacuation door flow meshes

!Timo: Mesh counter NM_EVAC is now fire meshes plus main evac meshes
IF (.NOT. NO_EVACUATION) CALL DEFINE_EVACUATION_MESHES(NM_EVAC)

! Determine mesh neighbors

ALLOCATE(NEIGHBOR_LIST(10000))
DO NM=1,NMESHES
M => MESHES(NM)
M%N_NEIGHBORING_MESHES = 0
NEIGHBOR_LIST = 0
DO NM2=1,NMESHES
IF (NM/=NM2 .AND. EVACUATION_ONLY(NM2)) CYCLE
M2 => MESHES(NM2)
IF ((M2%XS>M%XF+NANOMETER .OR. M2%XF<M%XS-NANOMETER .OR. &
M2%YS>M%YF+NANOMETER .OR. M2%YF<M%YS-NANOMETER .OR. &
M2%ZS>M%ZF+NANOMETER .OR. M2%ZF<M%ZS-NANOMETER) .AND. &
PERIODIC_MESH_NAMES(NM,1)/=MESH_NAME(NM2) .AND. &
PERIODIC_MESH_NAMES(NM,2)/=MESH_NAME(NM2) .AND. &
PERIODIC_MESH_NAMES(NM,3)/=MESH_NAME(NM2)) CYCLE
M%N_NEIGHBORING_MESHES = M%N_NEIGHBORING_MESHES + 1
NEIGHBOR_LIST(M%N_NEIGHBORING_MESHES) = NM2
ENDDO
ALLOCATE(M%NEIGHBORING_MESH(M%N_NEIGHBORING_MESHES))
DO I=1,M%N_NEIGHBORING_MESHES
M%NEIGHBORING_MESH(I) = NEIGHBOR_LIST(I)
ENDDO
ENDDO
DEALLOCATE(NEIGHBOR_LIST)

REWIND(LU_INPUT) ; INPUT_FILE_LINE_NUMBER = 0

CONTAINS

SUBROUTINE DEFINE_EVACUATION_MESHES(NM)
IMPLICIT NONE
! Passed variables
INTEGER, INTENT(INOUT) :: NM
! Local variables
INTEGER :: N, N_END, I, J, NN, JMAX, NM_OLD, I_MAIN_EVAC_MESH
REAL(EB) :: Z_MID

N = 0
DO I = 1, NM
IF (EVACUATION_SKIP(I) .AND. EVACUATION_ONLY(I)) THEN
N = N + 1 ! Main evacuation mesh index for EMESH_EXITS(N) array
EMESH_NM(N) = I
END IF
END DO

NM_OLD = NM
LOOP_EMESHES: DO N = 1, NEVAC_MESHES
! Additional meshes for the main evacuation meshes. These will be
! at different z level than the corresponding main evacuation mesh.

I_MAIN_EVAC_MESH = NM_OLD - NEVAC_MESHES + N

! Set MESH defaults

RGB   = MESHES(I_MAIN_EVAC_MESH)%RGB
COLOR = 'null'
ID = TRIM(TRIM('Emesh_' // MESH_NAME(I_MAIN_EVAC_MESH)))
MPI_PROCESS = -1
LEVEL = 0
EVACUATION = .TRUE.
EVAC_HUMANS = .FALSE.

! Increase the MESH counter by 1

NM = NM + 1

! Fill in MESH related variables

M => MESHES(NM)
M%MESH_LEVEL = LEVEL
M%IBAR = MESHES(I_MAIN_EVAC_MESH)%IBAR
M%JBAR = MESHES(I_MAIN_EVAC_MESH)%JBAR
M%KBAR = MESHES(I_MAIN_EVAC_MESH)%KBAR
IBAR_MAX = MAX(IBAR_MAX,M%IBAR)
JBAR_MAX = MAX(JBAR_MAX,M%JBAR)
KBAR_MAX = MAX(KBAR_MAX,M%KBAR)
EVACUATION_ONLY(NM) = .TRUE.
EVACUATION_SKIP(NM) = .FALSE.
EVACUATION_Z_OFFSET(NM) = EVAC_Z_OFFSET ! Not used, this line is not needed
M%N_EXTERNAL_WALL_CELLS = 2*M%IBAR*M%KBAR+2*M%JBAR*M%KBAR
IF (EVACUATION .AND. M%KBAR/=1) THEN
```

```
WRITE(MESSAGE,'(A)') 'ERROR: IJK(3) must be 1 for all evacuation grids'
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF

! Associate the MESH with the PROCESS

IF (MYID==CURRENT_MPI_PROCESS) THEN
LOWER_MESH_INDEX = MIN(LOWER_MESH_INDEX,NM)
UPPER_MESH_INDEX = MAX(UPPER_MESH_INDEX,NM)
ENDIF

PROCESS(NM) = CURRENT_MPI_PROCESS
IF (MYID==0 .AND. VERBOSE) WRITE(LU_ERR,'(A,I0,A,I0)') ' Mesh ',NM,' is assigned to MPI Process ',PROCESS(NM)
IF (EVACUATION_ONLY(NM) .AND. (N_MPI_PROCESSES>1)) EVAC_PROCESS = N_MPI_PROCESSES-1

! Mesh boundary colors

IF (ANY(RGB<0) .AND. COLOR=='null') COLOR = 'BLACK'
IF (COLOR /= 'null') CALL COLOR2RGB(RGB,COLOR)
ALLOCATE(M%RGB(3))
M%RGB = RGB

! Mesh Geometry and Name

WRITE(MESH_NAME(NM),'(A,I7.7)') 'MESH_',NM
IF (ID/='null') MESH_NAME(NM) = ID

Z_MID = 0.5_EB*(MESHES(I_MAIN_EVAC_MESH)%ZS + MESHES(I_MAIN_EVAC_MESH)%ZF)
Z_MID = Z_MID - EVACUATION_Z_OFFSET(I_MAIN_EVAC_MESH)  + HUMAN_SMOKE_HEIGHT
M%XS     = MESHES(I_MAIN_EVAC_MESH)%XS
M%XF     = MESHES(I_MAIN_EVAC_MESH)%XF
M%YS     = MESHES(I_MAIN_EVAC_MESH)%YS
M%YF     = MESHES(I_MAIN_EVAC_MESH)%YF
M%ZS     = Z_MID - EVAC_DELTA_SEE
M%ZF     = Z_MID + EVAC_DELTA_SEE
M%DXI    = MESHES(I_MAIN_EVAC_MESH)%DXI
M%DETA   = MESHES(I_MAIN_EVAC_MESH)%DETA
M%DZETA  = (M%ZF-M%ZS)/REAL(M%KBAR,EB)
M%RDXI   = MESHES(I_MAIN_EVAC_MESH)%RDXI
M%RDETA  = MESHES(I_MAIN_EVAC_MESH)%RDETA
M%RDZETA= 1._EB/M%DZETA
M%IBM1   = M%IBAR-1
M%JBM1   = M%JBAR-1
M%KBM1   = M%KBAR-1
M%IBP1   = M%IBAR+1
M%JBP1   = M%JBAR+1
M%KBP1   = M%KBAR+1
! WRITE (LU_ERR,FMT='(A,I0,3A)') ' EVAC: Mesh number ', NM, ' name ', TRIM(ID), ' defined for evacuation'

END DO LOOP_EMESHES

N_END = N_EXITS - N_CO_EXITS + N_DOORS
LOOP_EXITS: DO N = 1, N_END
I = EMESH_EXITS(N)%EMESH  ! The main evacuation mesh index (for EMESH_EXITS(I) array)
IF (.NOT.EMESH_EXITS(N)%DEFINE_MESH) CYCLE LOOP_EXITS

EMESH_EXITS(N)%MAINMESH = EMESH_NM(EMESH_EXITS(N)%EMESH)        ! The 1,...,NMESHES index
! Only main evacuation meshes in FDS6
EMESH_EXITS(N)%IMESH = EMESH_EXITS(N)%MAINMESH   ! The mesh index (all meshes included)

! Set MESH defaults

IJK(1)= EMESH_IJK(1,I)
IJK(2)= EMESH_IJK(2,I)
IJK(3)= EMESH_IJK(3,I)

ALLOCATE(EMESH_EXITS(N)%U_EVAC(0:IJK(1)+1,0:IJK(2)+1),STAT=IZERO)
CALL ChkMemErr('READ','EMESH_EXITS(N)%U_EVAC',IZERO)
ALLOCATE(EMESH_EXITS(N)%V_EVAC(0:IJK(1)+1,0:IJK(2)+1),STAT=IZERO)
CALL ChkMemErr('READ','EMESH_EXITS(N)%V_EVAC',IZERO)

CYCLE LOOP_EXITS
ENDDO LOOP_EXITS

NN = 0
JMAX = 0
DO I = 1, NM
EV_IF: IF (EVACUATION_SKIP(I) .AND. EVACUATION_ONLY(I)) THEN
J = 0  ! Index of the flow field (for a main evacuation mesh)
NN = NN + 1 ! Main evacuation mesh index
! NN = EMESH_INDEX(NM)
EMESH_NFIELDS(NN) = 0  ! How many fields for this main evacuation mesh
LOOP_EXITS_0: DO N = 1, N_END
IF (.NOT.EMESH_EXITS(N)%DEFINE_MESH) CYCLE LOOP_EXITS_0
IF (.NOT.EMESH_EXITS(N)%EMESH == NN) CYCLE LOOP_EXITS_0
J = J + 1
EMESH_EXITS(N)%I_DOORS_EMESH = J
EMESH_NFIELDS(NN) = J
```

```
END DO LOOP_EXITS_0
IF (EMESH_NFIELDS(NN)==0) THEN
WRITE(MESSAGE,'(A,I0,3A)') 'ERROR: EVAC: Emesh ',NN,' ',TRIM(EMESH_ID(NN)),' needs at least one DOOR/EXIT.'
CALL SHUTDOWN(MESSAGE) ; RETURN
ELSE
WRITE(LU_ERR,FMT='(A,I0,3A,I0,A)') ' EVAC: Emesh ',NN,' ',TRIM(EMESH_ID(NN)),' has ',&
EMESH_NFIELDS(NN),' door flow fields'
ENDIF
ENDIF EV_IF
ENDDO

! Next line should be executed only once during a FDS+Evac run
JMAX = MAXVAL(EMESH_NFIELDS,1)
EVAC_TIME_ITERATIONS = EVAC_TIME_ITERATIONS*JMAX

LOOP_STAIRS: DO N = 1, N_STRS

! Evacuation meshes for the stairs.

! Set MESH defaults

RGB   = EMESH_STAIRS(N)%RGB
COLOR = 'null'
ID = TRIM('Emesh_' // TRIM(EMESH_STAIRS(N)%ID))
MPI_PROCESS = -1
LEVEL = 0
EVACUATION = .TRUE.
EVAC_HUMANS = .TRUE.
EVAC_Z_OFFSET = EMESH_STAIRS(N)%EVAC_Z_OFFSET

! Increase the MESH counter by 1

NM = NM + 1
EMESH_STAIRS(N)%IMESH = NM

! Fill in MESH related variables

M => MESHES(NM)
M%MESH_LEVEL = LEVEL
M%IBAR = EMESH_STAIRS(N)%IBAR
M%JBAR = EMESH_STAIRS(N)%JBAR
M%KBAR = EMESH_STAIRS(N)%KBAR
IBAR_MAX = MAX(IBAR_MAX,M%IBAR)
JBAR_MAX = MAX(JBAR_MAX,M%JBAR)
KBAR_MAX = MAX(KBAR_MAX,M%KBAR)
EVACUATION_ONLY(NM) = .TRUE.
EVACUATION_SKIP(NM) = .TRUE.
EVACUATION_Z_OFFSET(NM) = EVAC_Z_OFFSET
M%N_EXTERNAL_WALL_CELLS = 2*M%IBAR*M%KBAR+2*M%JBAR*M%KBAR
IF (EVACUATION .AND. M%KBAR/=1) THEN
WRITE(MESSAGE,'(A)') 'ERROR: IJK(3) must be 1 for all evacuation grids'
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF

! Associate the MESH with the PROCESS

IF (MYID==CURRENT_MPI_PROCESS) THEN
LOWER_MESH_INDEX = MIN(LOWER_MESH_INDEX,NM)
UPPER_MESH_INDEX = MAX(UPPER_MESH_INDEX,NM)
ENDIF

PROCESS(NM) = CURRENT_MPI_PROCESS
IF (MYID==0 .AND. VERBOSE) WRITE(LU_ERR,'(A,I0,A,I0)') ' Mesh ',NM,' is assigned to MPI Process ',PROCESS(NM)
IF (EVACUATION_ONLY(NM) .AND. (N_MPI_PROCESSES>1)) EVAC_PROCESS = N_MPI_PROCESSES-1

! Mesh boundary colors

ALLOCATE(M%RGB(3))
M%RGB = EMESH_STAIRS(N)%RGB

! Mesh Geometry and Name

WRITE(MESH_NAME(NM),'(A,I7.7)') 'MESH_',NM
IF (ID/='null') MESH_NAME(NM) = ID

M%XS     = EMESH_STAIRS(N)%XB(1)
M%XF     = EMESH_STAIRS(N)%XB(2)
M%YS     = EMESH_STAIRS(N)%XB(3)
M%YF     = EMESH_STAIRS(N)%XB(4)
M%ZS     = EMESH_STAIRS(N)%XB(5)
M%ZF     = EMESH_STAIRS(N)%XB(6)
M%DXI    = (M%XF-M%XS)/REAL(M%IBAR,EB)
M%DETA   = (M%YF-M%YS)/REAL(M%JBAR,EB)
M%DZETA  = (M%ZF-M%ZS)/REAL(M%KBAR,EB)
M%RDXI   = 1._EB/M%DXI
M%RDETA  = 1._EB/M%DETA
M%RDZETA= 1._EB/M%DZETA
M%IBM1   = M%IBAR-1
```

```
M%JBM1   = M%JBAR-1
M%KBM1   = M%KBAR-1
M%IBP1   = M%IBAR+1
M%JBP1   = M%JBAR+1
M%KBP1   = M%KBAR+1
WRITE (LU_ERR,FMT='(A,I0,3A)') ' EVAC: Mesh number ', NM, ' name ', TRIM(ID), ' defined for evacuation'

ENDDO LOOP_STAIRS

IF (ALL(EVACUATION_ONLY)) THEN
DO N = 1, NMESHES
M => MESHES(NM)
XS_MIN   = MIN(XS_MIN,M%XS)
XF_MAX   = MAX(XF_MAX,M%XF)
YS_MIN   = MIN(YS_MIN,M%YS)
YF_MAX   = MAX(YF_MAX,M%YF)
ZS_MIN   = MIN(ZS_MIN,M%ZS)
ZF_MAX   = MAX(ZF_MAX,M%ZF)
ENDDO
ENDIF

RETURN
END SUBROUTINE DEFINE_EVACUATION_MESHES

END SUBROUTINE READ_MESH


SUBROUTINE READ_TRAN
USE MATH_FUNCTIONS, ONLY : GAUSSJ

! Compute the polynomial transform function for the vertical coordinate

REAL(EB), ALLOCATABLE, DIMENSION(:,:) :: A,XX
INTEGER, ALLOCATABLE, DIMENSION(:,:) :: ND
REAL(EB) :: PC,CC,COEF,XI,ETA,ZETA
INTEGER  IEXP,IC,IDERIV,N,K,IERROR,IOS,I,MESH_NUMBER, NIPX,NIPY,NIPZ,NIPXS,NIPYS,NIPZS,NIPXF,NIPYF,NIPZF,NM
LOGICAL :: PROCESS_TRANS
TYPE (MESH_TYPE), POINTER :: M=>NULL()
TYPE (TRAN_TYPE), POINTER :: T=>NULL()
NAMELIST /TRNX/ CC,FYI,IDERIV,MESH_NUMBER,PC
NAMELIST /TRNY/ CC,FYI,IDERIV,MESH_NUMBER,PC
NAMELIST /TRNZ/ CC,FYI,IDERIV,MESH_NUMBER,PC

! Scan the input file, counting the number of NAMELIST entries

ALLOCATE(TRANS(NMESHES))

MESH_LOOP: DO NM=1,NMESHES

M => MESHES(NM)

! Only read and process the TRNX, TRNY and TRNZ lines if the current MPI
! process (MYID) controls mesh NM or one of its neighbors.

PROCESS_TRANS = .FALSE.
DO N=1,M%N_NEIGHBORING_MESHES
IF (MYID==PROCESS(M%NEIGHBORING_MESH(N))) PROCESS_TRANS = .TRUE.
ENDDO

IF (PROCESS_ALL_MESHES) PROCESS_TRANS = .TRUE.

! A fast fix for fire+evacuation calculation with MPI and neighboring_mesh array problem
! Evacuation meshes need fire mesh obst information => evacuation process processes all fire meshes
IF(MYID==EVAC_PROCESS .AND. .NOT.EVACUATION_ONLY(NM)) PROCESS_TRANS = .TRUE.

IF (.NOT.PROCESS_TRANS) CYCLE MESH_LOOP

T => TRANS(NM)

DO N=1,3
T%NOC(N) = 0
TRNLOOP: DO
IF (EVACUATION_ONLY(NM)) EXIT TRNLOOP
SELECT CASE (N)
CASE(1)
CALL CHECKREAD('TRNX',LU_INPUT,IOS)
IF (IOS==1) EXIT TRNLOOP
MESH_NUMBER = 1
READ(LU_INPUT,NML=TRNX,END=17,ERR=18,IOSTAT=IOS)
IF (MESH_NUMBER>0 .AND. MESH_NUMBER/=NM) CYCLE TRNLOOP
CASE(2)
CALL CHECKREAD('TRNY',LU_INPUT,IOS)
IF (IOS==1) EXIT TRNLOOP
MESH_NUMBER = 1
READ(LU_INPUT,NML=TRNY,END=17,ERR=18,IOSTAT=IOS)
IF (MESH_NUMBER>0 .AND. MESH_NUMBER/=NM) CYCLE TRNLOOP
CASE(3)
CALL CHECKREAD('TRNZ',LU_INPUT,IOS)
```

```
IF (IOS==1) EXIT TRNLOOP
MESH_NUMBER = 1
READ(LU_INPUT,NML=TRNZ,END=17,ERR=18,IOSTAT=IOS)
IF (MESH_NUMBER>0 .AND. MESH_NUMBER/=NM) CYCLE TRNLOOP
END SELECT
T%NOC(N) = T%NOC(N) + 1
18 IF (IOS>0) THEN ; CALL SHUTDOWN('ERROR: Problem with TRN* line') ; RETURN ; ENDIF
ENDDO TRNLOOP
17 REWIND(LU_INPUT) ; INPUT_FILE_LINE_NUMBER = 0
ENDDO

T%NOCMAX = MAX(T%NOC(1),T%NOC(2),T%NOC(3))
ALLOCATE(A(T%NOCMAX+1,T%NOCMAX+1))
ALLOCATE(XX(T%NOCMAX+1,3))
ALLOCATE(ND(T%NOCMAX+1,3))
ALLOCATE(T%C1(0:T%NOCMAX+1,3))
T%C1             = 0._EB
T%C1(1,1:3)      = 1._EB
ALLOCATE(T%C2(0:T%NOCMAX+1,3))
ALLOCATE(T%C3(0:T%NOCMAX+1,3))
ALLOCATE(T%CCSTORE(T%NOCMAX,3))
ALLOCATE(T%PCSTORE(T%NOCMAX,3))
ALLOCATE(T%IDERIVSTORE(T%NOCMAX,3))

T%ITRAN  = 0

DO IC=1,3
NLOOP:  DO N=1,T%NOC(IC)
IDERIV = -1
IF (IC==1) THEN
LOOP1: DO
CALL CHECKREAD('TRNX',LU_INPUT,IOS)
IF (IOS==1) EXIT NLOOP
MESH_NUMBER = 1
READ(LU_INPUT,TRNX,END=1,ERR=2)
IF (MESH_NUMBER==0 .OR. MESH_NUMBER==NM) EXIT LOOP1
ENDDO LOOP1
ENDIF
IF (IC==2) THEN
LOOP2: DO
CALL CHECKREAD('TRNY',LU_INPUT,IOS)
IF (IOS==1) EXIT NLOOP
MESH_NUMBER = 1
READ(LU_INPUT,TRNY,END=1,ERR=2)
IF (MESH_NUMBER==0 .OR. MESH_NUMBER==NM) EXIT LOOP2
ENDDO LOOP2
ENDIF
IF (IC==3) THEN
LOOP3: DO
CALL CHECKREAD('TRNZ',LU_INPUT,IOS)
IF (IOS==1) EXIT NLOOP
MESH_NUMBER = 1
READ(LU_INPUT,TRNZ,END=1,ERR=2)
IF (MESH_NUMBER==0 .OR. MESH_NUMBER==NM) EXIT LOOP3
ENDDO LOOP3
ENDIF
T%CCSTORE(N,IC) = CC
T%PCSTORE(N,IC) = PC
T%IDERIVSTORE(N,IC) = IDERIV
IF (IDERIV>=0) T%ITRAN(IC) = 1
IF (IDERIV<0)  T%ITRAN(IC) = 2
2 CONTINUE
ENDDO NLOOP
1 REWIND(LU_INPUT) ; INPUT_FILE_LINE_NUMBER = 0
ENDDO

ICLOOP: DO IC=1,3

SELECT CASE (T%ITRAN(IC))

CASE (1)  ! polynomial transformation
ND(1,IC)  = 0
SELECT CASE(IC)
CASE(1)
XX(1,IC)     = M%XF-M%XS
T%C1(1,IC)   = M%XF-M%XS
CASE(2)
XX(1,IC)     = M%YF-M%YS
T%C1(1,IC)   = M%YF-M%YS
CASE(3)
XX(1,IC)     = M%ZF-M%ZS
T%C1(1,IC)   = M%ZF-M%ZS
END SELECT

NNLOOP:  DO N=2,T%NOC(IC)+1
IDERIV = T%IDERIVSTORE(N-1,IC)
IF (IC==1) CC = T%CCSTORE(N-1,IC)-M%XS
IF (IC==2) CC = T%CCSTORE(N-1,IC)-M%YS
```

```
IF (IC==3) CC = T%CCSTORE(N-1,IC)-M%ZS
IF (IC==1 .AND. IDERIV==0) PC = T%PCSTORE(N-1,IC)-M%XS
IF (IC==2 .AND. IDERIV==0) PC = T%PCSTORE(N-1,IC)-M%YS
IF (IC==3 .AND. IDERIV==0) PC = T%PCSTORE(N-1,IC)-M%ZS
IF (IC==1 .AND. IDERIV>0) PC = T%PCSTORE(N-1,IC)
IF (IC==2 .AND. IDERIV>0) PC = T%PCSTORE(N-1,IC)
IF (IC==3 .AND. IDERIV>0) PC = T%PCSTORE(N-1,IC)
ND(N,IC) = IDERIV
XX(N,IC) = CC
T%C1(N,IC) = PC
ENDDO NNLOOP

DO K=1,T%NOC(IC)+1
DO N=1,T%NOC(IC)+1
COEF = IFAC(K,ND(N,IC))
IEXP = K-ND(N,IC)
IF (IEXP<0) A(N,K) = 0._EB
IF (IEXP==0) A(N,K) = COEF
IF (IEXP>0) A(N,K) = COEF*XX(N,IC)**IEXP
ENDDO
ENDDO

IERROR = 0
CALL GAUSSJ(A,T%NOC(IC)+1,T%NOCMAX+1,T%C1(1:T%NOCMAX+1,IC),1,1,IERROR)
IF (IERROR/=0) THEN ; CALL SHUTDOWN('ERROR: Problem with grid transformation') ; RETURN ; ENDIF

CASE (2) ! linear transformation

T%C1(0,IC) = 0._EB
T%C2(0,IC) = 0._EB
DO N=1,T%NOC(IC)
IF (IC==1) CC = T%CCSTORE(N,IC)-M%XS
IF (IC==2) CC = T%CCSTORE(N,IC)-M%YS
IF (IC==3) CC = T%CCSTORE(N,IC)-M%ZS
IF (IC==1) PC = T%PCSTORE(N,IC)-M%XS
IF (IC==2) PC = T%PCSTORE(N,IC)-M%YS
IF (IC==3) PC = T%PCSTORE(N,IC)-M%ZS
T%C1(N,IC) = CC
T%C2(N,IC) = PC
ENDDO

SELECT CASE(IC)
CASE(1)
T%C1(T%NOC(1)+1,1) = M%XF-M%XS
T%C2(T%NOC(1)+1,1) = M%XF-M%XS
CASE(2)
T%C1(T%NOC(2)+1,2) = M%YF-M%YS
T%C2(T%NOC(2)+1,2) = M%YF-M%YS
CASE(3)
T%C1(T%NOC(3)+1,3) = M%ZF-M%ZS
T%C2(T%NOC(3)+1,3) = M%ZF-M%ZS
END SELECT

DO N=1,T%NOC(IC)+1
IF (T%C1(N,IC)-T%C1(N-1,IC)<TWO_EPSILON_EB) THEN
CALL SHUTDOWN('ERROR: Do not specify endpoints in linear grid transformation')
RETURN
ENDIF
T%C3(N,IC) = (T%C2(N,IC)-T%C2(N-1,IC))/(T%C1(N,IC)-T%C1(N-1,IC))
ENDDO
END SELECT
ENDDO ICLOOP

DEALLOCATE(A)
DEALLOCATE(XX)
DEALLOCATE(ND)

! Set up grid stretching arrays

ALLOCATE(M%R(0:M%IBAR),STAT=IZERO)
CALL ChkMemErr('READ','R',IZERO)
ALLOCATE(M%RC(0:M%IBAR+1),STAT=IZERO)
CALL ChkMemErr('READ','RC',IZERO)
M%RC = 1._EB
ALLOCATE(M%RRN(0:M%IBP1),STAT=IZERO)
CALL ChkMemErr('READ','RRN',IZERO)
M%RRN = 1._EB
ALLOCATE(M%X(0:M%IBAR),STAT=IZERO)
CALL ChkMemErr('READ','X',IZERO)
ALLOCATE(M%XC(0:M%IBP1),STAT=IZERO)
CALL ChkMemErr('READ','XC',IZERO)
ALLOCATE(M%HX(0:M%IBP1),STAT=IZERO)
CALL ChkMemErr('READ','HX',IZERO)
ALLOCATE(M%DX(0:M%IBP1),STAT=IZERO)
CALL ChkMemErr('READ','DX',IZERO)
ALLOCATE(M%RDX(0:M%IBP1),STAT=IZERO)
CALL ChkMemErr('READ','RDX',IZERO)
ALLOCATE(M%DXN(0:M%IBAR),STAT=IZERO)
```

```
CALL ChkMemErr('READ','DXN',IZERO)
ALLOCATE(M%RDXN(0:M%IBAR),STAT=IZERO)
CALL ChkMemErr('READ','RDXN',IZERO)
ALLOCATE(M%Y(0:M%JBAR),STAT=IZERO)
CALL ChkMemErr('READ','Y',IZERO)
ALLOCATE(M%YC(0:M%JBP1),STAT=IZERO)
CALL ChkMemErr('READ','YC',IZERO)
ALLOCATE(M%HY(0:M%JBP1),STAT=IZERO)
CALL ChkMemErr('READ','HY',IZERO)
ALLOCATE(M%DY(0:M%JBP1),STAT=IZERO)
CALL ChkMemErr('READ','DY',IZERO)
ALLOCATE(M%RDY(0:M%JBP1),STAT=IZERO)
CALL ChkMemErr('READ','RDY',IZERO)
ALLOCATE(M%DYN(0:M%JBAR),STAT=IZERO)
CALL ChkMemErr('READ','DYN',IZERO)
ALLOCATE(M%RDYN(0:M%JBAR),STAT=IZERO)
CALL ChkMemErr('READ','RDYN',IZERO)
ALLOCATE(M%Z(0:M%KBAR),STAT=IZERO)
CALL ChkMemErr('READ','Z',IZERO)
ALLOCATE(M%ZC(0:M%KBP1),STAT=IZERO)
CALL ChkMemErr('READ','ZC',IZERO)
ALLOCATE(M%HZ(0:M%KBP1),STAT=IZERO)
CALL ChkMemErr('READ','HZ',IZERO)
ALLOCATE(M%DZ(0:M%KBP1),STAT=IZERO)
CALL ChkMemErr('READ','DZ',IZERO)
ALLOCATE(M%RDZ(0:M%KBP1),STAT=IZERO)
CALL ChkMemErr('READ','RDZ',IZERO)
ALLOCATE(M%DZN(0:M%KBAR),STAT=IZERO)
CALL ChkMemErr('READ','DZN',IZERO)
ALLOCATE(M%RDZN(0:M%KBAR),STAT=IZERO)
CALL ChkMemErr('READ','RDZN',IZERO)

! Define X grid stretching terms

M%DXMIN = 1000._EB
DO I=1,M%IBAR
XI      = (REAL(I,EB)-.5)*M%DXI
M%HX(I) = GP(XI,1,NM)
M%DX(I) = M%HX(I)*M%DXI
M%DXMIN = MIN(M%DXMIN,M%DX(I))
IF (M%HX(I)<=0._EB) THEN
WRITE(MESSAGE,'(A,I0)') 'ERROR: x transformation not monotonic, mesh ',NM
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF
M%RDX(I) = 1._EB/M%DX(I)
ENDDO

M%HX(0)        = M%HX(1)
M%HX(M%IBP1) = M%HX(M%IBAR)
M%DX(0)        = M%DX(1)
M%DX(M%IBP1) = M%DX(M%IBAR)
M%RDX(0)       = 1._EB/M%DX(1)
M%RDX(M%IBP1) = 1._EB/M%DX(M%IBAR)

DO I=0,M%IBAR
XI        = I*M%DXI
M%X(I) = M%XS + G(XI,1,NM)
IF (CYLINDRICAL) THEN
M%R(I) = M%X(I)
ELSE
M%R(I) = 1._EB
ENDIF
M%DXN(I)   = 0.5_EB*(M%DX(I)+M%DX(I+1))
M%RDXN(I) = 1._EB/M%DXN(I)
ENDDO
M%X(0)         = M%XS
M%X(M%IBAR) = M%XF

DO I=1,M%IBAR
M%XC(I) = 0.5_EB*(M%X(I)+M%X(I-1))
ENDDO
M%XC(0)          = M%XS - 0.5_EB*M%DX(0)
M%XC(M%IBP1) = M%XF + 0.5_EB*M%DX(M%IBP1)

IF (CYLINDRICAL) THEN
DO I=1,M%IBAR
M%RRN(I) = 2._EB/(M%R(I)+M%R(I-1))
M%RC(I)   = 0.5_EB*(M%R(I)+M%R(I-1))
ENDDO
M%RRN(0)       = M%RRN(1)
M%RRN(M%IBP1) = M%RRN(M%IBAR)
ENDIF

! Define Y grid stretching terms

M%DYMIN = 1000._EB
DO J=1,M%JBAR
ETA     = (REAL(J,EB)-.5)*M%DETA
```

```
M%HY(J) = GP(ETA,2,NM)
M%DY(J) = M%HY(J)*M%DETA
M%DYMIN = MIN(M%DYMIN,M%DY(J))
 IF (M%HY(J)<=0._EB) THEN
 WRITE(MESSAGE,'(A,I0)') 'ERROR: y transformation not monotonic, mesh ',NM
 CALL SHUTDOWN(MESSAGE) ; RETURN
 ENDIF
M%RDY(J) = 1._EB/M%DY(J)
ENDDO

M%HY(0)      = M%HY(1)
M%HY(M%JBP1) = M%HY(M%JBAR)
M%DY(0)      = M%DY(1)
M%DY(M%JBP1) = M%DY(M%JBAR)
M%RDY(0)      = 1._EB/M%DY(1)
M%RDY(M%JBP1) = 1._EB/M%DY(M%JBAR)

DO J=0,M%JBAR
ETA       = J*M%DETA
M%Y(J)    = M%YS + G(ETA,2,NM)
M%DYN(J)  = 0.5_EB*(M%DY(J)+M%DY(J+1))
M%RDYN(J) = 1._EB/M%DYN(J)
ENDDO

M%Y(0)       = M%YS
M%Y(M%JBAR)  = M%YF

DO J=1,M%JBAR
M%YC(J) = 0.5_EB*(M%Y(J)+M%Y(J-1))
ENDDO
M%YC(0)       = M%YS - 0.5_EB*M%DY(0)
M%YC(M%JBP1)  = M%YF + 0.5_EB*M%DY(M%JBP1)

! Define Z grid stretching terms

M%DZMIN = 1000._EB
DO K=1,M%KBAR
ZETA   = (REAL(K,EB)-.5_EB)*M%DZETA
M%HZ(K) = GP(ZETA,3,NM)
M%DZ(K) = M%HZ(K)*M%DZETA
M%DZMIN = MIN(M%DZMIN,M%DZ(K))
 IF (M%HZ(K)<=0._EB) THEN
 WRITE(MESSAGE,'(A,I0)') 'ERROR: z transformation not monotonic, mesh ',NM
 CALL SHUTDOWN(MESSAGE) ; RETURN
 ENDIF
M%RDZ(K) = 1._EB/M%DZ(K)
ENDDO

M%HZ(0)      = M%HZ(1)
M%HZ(M%KBP1) = M%HZ(M%KBAR)
M%DZ(0)      = M%DZ(1)
M%DZ(M%KBP1) = M%DZ(M%KBAR)
M%RDZ(0)      = 1._EB/M%DZ(1)
M%RDZ(M%KBP1) = 1._EB/M%DZ(M%KBAR)

DO K=0,M%KBAR
ZETA      = K*M%DZETA
M%Z(K)    = M%ZS + G(ZETA,3,NM)
M%DZN(K)  = 0.5_EB*(M%DZ(K)+M%DZ(K+1))
M%RDZN(K) = 1._EB/M%DZN(K)
ENDDO

M%Z(0)       = M%ZS
M%Z(M%KBAR)  = M%ZF

DO K=1,M%KBAR
M%ZC(K) = 0.5_EB*(M%Z(K)+M%Z(K-1))
ENDDO
M%ZC(0)       = M%ZS - 0.5_EB*M%DZ(0)
M%ZC(M%KBP1)  = M%ZF + 0.5_EB*M%DZ(M%KBP1)

! Set up arrays that will return coordinate positions

NIPX    = 500*M%IBAR
NIPY    = 500*M%JBAR
NIPZ    = 500*M%KBAR
NIPXS   = NINT(NIPX*M%DX(0)/(M%XF-M%XS))
NIPXF   = NINT(NIPX*M%DX(M%IBP1)/(M%XF-M%XS))
NIPYS   = NINT(NIPY*M%DY(0)/(M%YF-M%YS))
NIPYF   = NINT(NIPY*M%DY(M%JBP1)/(M%YF-M%YS))
NIPZS   = NINT(NIPZ*M%DZ(0)/(M%ZF-M%ZS))
NIPZF   = NINT(NIPZ*M%DZ(M%KBP1)/(M%ZF-M%ZS))
M%RDXINT = REAL(NIPX,EB)/(M%XF-M%XS)
M%RDYINT = REAL(NIPY,EB)/(M%YF-M%YS)
M%RDZINT = REAL(NIPZ,EB)/(M%ZF-M%ZS)

ALLOCATE(M%CELLSI(-NIPXS:NIPX+NIPXF),STAT=IZERO)
CALL ChkMemErr('READ','CELLSI',IZERO)
```

```fortran
ALLOCATE(M%CELLSJ(-NIPYS:NIPY+NIPYF),STAT=IZERO)
CALL ChkMemErr('READ','CELLSJ',IZERO)
ALLOCATE(M%CELLSK(-NIPZS:NIPZ+NIPZF),STAT=IZERO)
CALL ChkMemErr('READ','CELLSK',IZERO)

DO I=-NIPXS,NIPX+NIPXF
M%CELLSI(I) = GINV(REAL(I,EB)/M%RDXINT,1,NM)*M%RDXI
M%CELLSI(I) = MAX(M%CELLSI(I),-0.9_EB)
M%CELLSI(I) = MIN(M%CELLSI(I),REAL(M%IBAR)+0.9_EB)
ENDDO
DO J=-NIPYS,NIPY+NIPYF
M%CELLSJ(J) = GINV(REAL(J,EB)/M%RDYINT,2,NM)*M%RDETA
M%CELLSJ(J) = MAX(M%CELLSJ(J),-0.9_EB)
M%CELLSJ(J) = MIN(M%CELLSJ(J),REAL(M%JBAR)+0.9_EB)
ENDDO
DO K=-NIPZS,NIPZ+NIPZF
M%CELLSK(K) = GINV(REAL(K,EB)/M%RDZINT,3,NM)*M%RDZETA
M%CELLSK(K) = MAX(M%CELLSK(K),-0.9_EB)
M%CELLSK(K) = MIN(M%CELLSK(K),REAL(M%KBAR)+0.9_EB)
ENDDO

ENDDO MESH_LOOP


CONTAINS

INTEGER FUNCTION IFAC(II,NN)
INTEGER, INTENT(IN) :: II,NN
INTEGER :: III
IFAC = 1
DO III=II-NN+1,II
IFAC = IFAC*III
ENDDO
END FUNCTION IFAC

END SUBROUTINE READ_TRAN


SUBROUTINE READ_TIME(DT)

REAL(EB) :: VEL_CHAR,DT
NAMELIST /TIME/ DT,EVAC_DT_FLOWFIELD,EVAC_DT_STEADY_STATE,FYI,LIMITING_DT_RATIO,LOCK_TIME_STEP,RESTRICT_TIME_STEP, &
T_BEGIN,T_END,T_END_GEOM,TIME_SHRINK_FACTOR,WALL_INCREMENT,WALL_INCREMENT_HT3D, &
TWFIN ! Backward compatibility

DT                   = -1._EB
EVAC_DT_FLOWFIELD    = 0.01_EB
EVAC_DT_STEADY_STATE = 0.05_EB
TIME_SHRINK_FACTOR   = 1._EB
T_BEGIN              = 0._EB
T_END                = 1._EB
T_END_GEOM           = T_END
IF (ALL(EVACUATION_ONLY)) DT = EVAC_DT_STEADY_STATE

REWIND(LU_INPUT) ; INPUT_FILE_LINE_NUMBER = 0
READ_TIME_LOOP: DO
CALL CHECKREAD('TIME',LU_INPUT,IOS)
IF (IOS==1) EXIT READ_TIME_LOOP
READ(LU_INPUT,TIME,END=21,ERR=22,IOSTAT=IOS)
22 IF (IOS>0) THEN ; CALL SHUTDOWN('ERROR: Problem with TIME line') ; RETURN ; ENDIF
ENDDO READ_TIME_LOOP
21 REWIND(LU_INPUT) ; INPUT_FILE_LINE_NUMBER = 0

IF (T_END<=T_BEGIN) SET_UP_ONLY = .TRUE.
T_END = T_BEGIN + (T_END-T_BEGIN)/TIME_SHRINK_FACTOR

! Compute the starting time step if the user has not specified it.

IF (DT<=0._EB) THEN
VEL_CHAR = 0.2_EB*SQRT(10._EB*(ZF_MAX-ZS_MIN))
IF (ABS(U0)>TWO_EPSILON_EB .OR. ABS(V0)>TWO_EPSILON_EB .OR. ABS(W0)>TWO_EPSILON_EB) &
VEL_CHAR = MAX(VEL_CHAR,SQRT(U0**2+V0**2+W0**2))
DT = CFL_MAX*CHARACTERISTIC_CELL_SIZE/VEL_CHAR
ENDIF

END SUBROUTINE READ_TIME



SUBROUTINE READ_MULT

REAL(EB) :: DX,DY,DZ,DXB(6),DX0,DY0,DZ0
CHARACTER(LABEL_LENGTH) :: ID
INTEGER :: N,I_LOWER,I_UPPER,J_LOWER,J_UPPER,K_LOWER,K_UPPER,N_LOWER,N_UPPER
TYPE(MULTIPLIER_TYPE), POINTER :: MR=>NULL()
NAMELIST /MULT/ DX,DXB,DX0,DY,DY0,DZ,DZ0,FYI,ID,I_LOWER,I_UPPER,J_LOWER,J_UPPER,K_LOWER,K_UPPER,N_LOWER,N_UPPER

```

```
N_MULT = 0
REWIND(LU_INPUT) ; INPUT_FILE_LINE_NUMBER = 0
COUNT_MULT_LOOP: DO
CALL CHECKREAD('MULT',LU_INPUT,IOS)
IF (IOS==1) EXIT COUNT_MULT_LOOP
READ(LU_INPUT,NML=MULT,END=9,ERR=10,IOSTAT=IOS)
N_MULT = N_MULT + 1
10 IF (IOS>0) THEN
WRITE(MESSAGE,'(A,I0,A,I0)') 'ERROR: Problem with MULT number',N_MULT,', line number',INPUT_FILE_LINE_NUMBER
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF
ENDDO COUNT_MULT_LOOP
9 REWIND(LU_INPUT) ; INPUT_FILE_LINE_NUMBER = 0

ALLOCATE(MULTIPLIER(0:N_MULT),STAT=IZERO)
CALL ChkMemErr('READ','MULTIPLIER',IZERO)

READ_MULT_LOOP: DO N=0,N_MULT

ID      = 'null'
IF (N==0) ID = 'MULT DEFAULT'
DX      = 0._EB
DY      = 0._EB
DZ      = 0._EB
DX0     = 0._EB
DY0     = 0._EB
DZ0     = 0._EB
DXB     = 0._EB
I_LOWER = 0
I_UPPER = 0
J_LOWER = 0
J_UPPER = 0
K_LOWER = 0
K_UPPER = 0
N_LOWER = 0
N_UPPER = 0

IF (N>0) THEN
CALL CHECKREAD('MULT',LU_INPUT,IOS)
IF (IOS==1) EXIT READ_MULT_LOOP
READ(LU_INPUT,MULT)
ENDIF

MR => MULTIPLIER(N)
MR%ID       = ID
MR%DXB      = DXB
MR%DX0      = DX0
MR%DY0      = DY0
MR%DZ0      = DZ0
IF (ABS(DX)>TWO_EPSILON_EB) MR%DXB(1:2) = DX
IF (ABS(DY)>TWO_EPSILON_EB) MR%DXB(3:4) = DY
IF (ABS(DZ)>TWO_EPSILON_EB) MR%DXB(5:6) = DZ

MR%I_LOWER = I_LOWER
MR%I_UPPER = I_UPPER
MR%J_LOWER = J_LOWER
MR%J_UPPER = J_UPPER
MR%K_LOWER = K_LOWER
MR%K_UPPER = K_UPPER
MR%N_COPIES = (I_UPPER-I_LOWER+1)*(J_UPPER-J_LOWER+1)*(K_UPPER-K_LOWER+1)

IF (N_LOWER/=0 .OR. N_UPPER/=0) THEN
MR%SEQUENTIAL = .TRUE.
MR%I_LOWER  = N_LOWER
MR%I_UPPER  = N_UPPER
MR%J_LOWER  = 0
MR%J_UPPER  = 0
MR%K_LOWER  = 0
MR%K_UPPER  = 0
MR%N_COPIES = (N_UPPER-N_LOWER+1)
ENDIF

ENDDO READ_MULT_LOOP
REWIND(LU_INPUT) ; INPUT_FILE_LINE_NUMBER = 0

END SUBROUTINE READ_MULT


SUBROUTINE READ_MISC

USE MATH_FUNCTIONS, ONLY: GET_RAMP_INDEX
REAL(EB) :: MAXIMUM_VISIBILITY
CHARACTER(LABEL_LENGTH) :: ASSUMED_GAS_TEMPERATURE_RAMP,RAMP_GX,RAMP_GY,RAMP_GZ,TURBULENCE_MODEL,&
NEAR_WALL_TURBULENCE_MODEL,COMBUSTION_MODEL_SELECT    !Sesa-added this new MISC option COMBUSTION_MODEL_SELECT

NAMELIST /MISC/ AGGLOMERATION,AEROSOL_AL2O3,ALLOW_SURFACE_PARTICLES,ALLOW_UNDERSIDE_PARTICLES,
        ASSUMED_GAS_TEMPERATURE,&
ASSUMED_GAS_TEMPERATURE_RAMP,&
```

```
BAROCLINIC,BNDF_DEFAULT,CC_IBM,CCVOL_LINK,CC_ZEROIBM_VELO,CHECK_MASS_CONSERVE, &
CNF_CUTOFF,CFL_MAX,CFL_MIN,CFL_VELOCITY_NORM,&
CHECK_HT,CHECK_REALIZABILITY,CHECK_VN,CLIP_MASS_FRACTION,COMPUTE_CUTCELLS_ONLY,&
COMPUTE_VISCOSITY_TWICE,COMPUTE_ZETA_SOURCE_TERM,CONSTANT_H_SOLID,CONSTANT_SPECIFIC_HEAT_RATIO,&
CORRECT_SUBGRID_TEMPERATURE,COUPLED_1D3D_HEAT_TRANSFER,C_DEARDORFF,C_RNG,C_RNG_CUTOFF,C_SMAGORINSKY,C_VREMAN,&
C_WALE,DNS,DO_IMPLICIT_CCREGION,DRAG_CFL_MAX,ENTHALPY_TRANSPORT,&
EVACUATION_DRILL,EVACUATION_MC_MODE,EVAC_PRESSURE_ITERATIONS,EVAC_SURF_DEFAULT,EVAC_TIME_ITERATIONS,&
EVAPORATION,EXTERNAL_BOUNDARY_CORRECTION,EXTINCTION_MODEL,HVAC_PRES_RELAX,HT3D_TEST,&
FDS5_OPTIONS,FLUX_LIMITER,FREEZE_VELOCITY,FYI,GAMMA,GRAVITATIONAL_DEPOSITION,&
GRAVITATIONAL_SETTLING,GVEC,DT_HVAC,H_F_REFERENCE_TEMPERATURE,&
HRRPUV_MAX_SMV,HUMIDITY,HVAC_MASS_TRANSPORT,&
IBLANK_SMV,IMMERSED_BOUNDARY_METHOD,INITIAL_UNMIXED_FRACTION,&
LES_FILTER_WIDTH,MAX_CHEMISTRY_ITERATIONS,&
MAX_LEAK_PATHS,MAXIMUM_VISIBILITY,MPI_TIMEOUT,&
N_FIXED_CHEMISTRY_SUBSTEPS,N_INITIAL_PARTICLE_SUBSTEPS,NEAR_WALL_TURBULENCE_MODEL,&
NOISE,NOISE_VELOCITY,NO_EVACUATION,NO_RAMPS,&
OVERWRITE,PARTICLE_CFL_MAX,PARTICLE_CFL_MIN,PARTICLE_CFL,PERIODIC_TEST,PROFILING,POROUS_FLOOR,&
PR,PROJECTION,P_INF,PROCESS_ALL_MESHES,RAMP_GX,RAMP_GY,RAMP_GZ,&
RADIATION,RESEARCH_MODE,RESTART,RESTART_CHID,RICHARDSON_ERROR_TOLERANCE,&
SC,SHARED_FILE_SYSTEM,SLIP_CONDITION,SMOKE_ALBEDO,SOLID_PHASE_ONLY,SOOT_OXIDATION,&
STRATIFICATION,SUPPRESSION,SURF_DEFAULT,TEMPERATURE_DEPENDENT_REACTION,&
TENSOR_DIFFUSIVITY,TERRAIN_CASE,TERRAIN_IMAGE,TEST_FILTER_QUADRATURE,TEXTURE_ORIGIN,&
THERMOPHORETIC_DEPOSITION,THICKEN_OBSTRUCTIONS,TRANSPORT_UNMIXED_FRACTION,TRANSPORT_ZETA_SCHEME,&
TMPA,TURBULENCE_MODEL,TURBULENT_DEPOSITION,TURB_INIT_CLOCK,UVW_FILE,&
VEG_LEVEL_SET_COUPLED,VEG_LEVEL_SET_UNCOUPLED,VERBOSE,VISIBILITY_FACTOR,VN_MAX,VN_MIN,Y_CO2_INFTY,Y_O2_INFTY,&
WD_PROPS,WIND_ONLY,COMBUSTION_MODEL_SELECT,HRRPUA_SHEET,HRRPUV_AVERAGE, SIX  !Sesa-added to namelist MISC -
        COMBUSTION_MODEL_SELECT,HRRPUA_SHEET
!HRRPUV_AVERAGE

! Physical constants

TMPA          = 20._EB                                  ! Ambient temperature (C)
GAMMA         = 1.4_EB                                  ! Heat capacity ratio for air
P_INF         = 101325._EB                              ! Ambient pressure (Pa)
MU_AIR_0      = 1.8E-5_EB                               ! Dynamic Viscosity of Air at 20 C (kg/m/s)
CP_AIR_0      = 1012._EB                                ! Specific Heat of Air at 20 C (J/kg/K)
PR_AIR        = 0.7_EB
K_AIR_0       = MU_AIR_0*CP_AIR_0/PR_AIR                ! Thermal Conductivity of Air at 20 C (W/m/K)

! Empirical heat transfer constants

PR_ONTH       = PR_AIR**ONTH
ASSUMED_GAS_TEMPERATURE = -1000.    ! Assumed gas temperature, used for diagnostics

! Miscellaneous constants

RESTART_CHID     = CHID
RESTART          = .FALSE.
NOISE            = .TRUE.
LES              = .TRUE.
DNS              = .FALSE.
SOLID_PHASE_ONLY = .FALSE.
IBLANK_SMV       = .TRUE.

TEXTURE_ORIGIN(1) = 0._EB
TEXTURE_ORIGIN(2) = 0._EB
TEXTURE_ORIGIN(3) = 0._EB

! EVACuation parameters

EVAC_PRESSURE_ITERATIONS = 50
EVAC_TIME_ITERATIONS     = 50
EVACUATION_MC_MODE       = .FALSE.
NO_EVACUATION            = .FALSE.
EVACUATION_DRILL         = .FALSE.

! LES parameters

PR                      = -1.0_EB  ! Turbulent Prandtl number
SC                      = -1.0_EB  ! Turbulent Schmidt number

! Misc

ASSUMED_GAS_TEMPERATURE_RAMP = 'null'
RAMP_GX                 = 'null'
RAMP_GY                 = 'null'
RAMP_GZ                 = 'null'
EXTINCTION_MODEL        = 'null'
GVEC(1)                 = 0._EB        ! x-component of gravity
GVEC(2)                 = 0._EB        ! y-component of gravity
GVEC(3)                 = -GRAV        ! z-component of gravity
THICKEN_OBSTRUCTIONS    = .FALSE.
CFL_MAX                 = 1.0_EB       ! Stability bounds
CFL_MIN                 = 0.8_EB
VN_MAX                  = 1.0_EB
VN_MIN                  = 0.8_EB
PARTICLE_CFL_MAX        = 1._EB
PARTICLE_CFL_MIN        = 0.8_EB
```

```fortran
DRAG_CFL_MAX           = 1.0_EB
VEG_LEVEL_SET              = .FALSE.
VEG_LEVEL_SET_COUPLED      = .FALSE.
VEG_LEVEL_SET_UNCOUPLED = .FALSE.
TERRAIN_IMAGE          = 'xxxnull'
MAXIMUM_VISIBILITY     = 30._EB ! m
VISIBILITY_FACTOR      = 3._EB
TURBULENCE_MODEL       = 'null'
NEAR_WALL_TURBULENCE_MODEL = 'null'
MAX_LEAK_PATHS         = 200
COMBUSTION_MODEL_SELECT = 'null'    !Sesa-added for selecting combustion model
HRRPUA_SHEET           = 200._EB    !Sesa-added as in 6.2.0 kW/m^2
HRRPUV_AVERAGE         = 2500._EB   !Sesa-added as in 6.2.0 kW/m^3
IF (N_MPI_PROCESSES<=50) VERBOSE = .TRUE.

! Initial read of the MISC line

REWIND(LU_INPUT) ; INPUT_FILE_LINE_NUMBER = 0
MISC_LOOP: DO
CALL CHECKREAD('MISC',LU_INPUT,IOS)
IF (IOS==1) EXIT MISC_LOOP
READ(LU_INPUT,MISC,END=23,ERR=24,IOSTAT=IOS)
24 IF (IOS>0) THEN ; CALL SHUTDOWN('ERROR: Problem with MISC line') ; RETURN ; ENDIF
ENDDO MISC_LOOP
23 REWIND(LU_INPUT) ; INPUT_FILE_LINE_NUMBER = 0

!Sesa-Combustion model select
!IF (COMBUSTION_MODEL_SELECT/='null') THEN
!       WRITE(MESSAGE,'(A,A,A)')  'ERROR: COMBUSTION_MODEL_SELECT, ',TRIM(COMBUSTION_MODEL_SELECT),', is not appropriate.'
!       CALL SHUTDOWN(MESSAGE) ; RETURN
!ELSE
!   COMBUSTION_MODEL_SELECT = 'COMBUSTION_SIX' !Sesa-default model is combustion model of 6.6.0
!ENDIF
!Sesa-Combustion


! FDS 6 options

IF (DNS) THEN
CFL_VELOCITY_NORM = 1
CFL_MAX = 0.5
CFL_MIN = 0.4
VN_MAX = 0.5
VN_MIN = 0.4
FLUX_LIMITER = CHARM_LIMITER
IF (TURBULENCE_MODEL/='null') THEN
WRITE(MESSAGE,'(A,A,A)')  'ERROR: TURBULENCE_MODEL, ',TRIM(TURBULENCE_MODEL),', is not appropriate for DNS.'
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF
ELSE
FLUX_LIMITER = SUPERBEE_LIMITER
TURBULENCE_MODEL = 'DEARDORFF'
LES_FILTER_WIDTH = 'MEAN'
ENDIF

! FDS 5 options (diagnostic timing purposes)

IF (FDS5_OPTIONS) THEN
FLUX_LIMITER = CENTRAL_LIMITER
TURBULENCE_MODEL = 'CONSTANT SMAGORINSKY'
BAROCLINIC = .FALSE.
CFL_VELOCITY_NORM = 3
CONSTANT_SPECIFIC_HEAT_RATIO = .TRUE.
MAX_PRESSURE_ITERATIONS = 1 ! see PRES
EXTINCTION_MODEL = 'EXTINCTION 1'
ENDIF

! Re-read the line to pick up any user-specified options

REWIND(LU_INPUT) ; INPUT_FILE_LINE_NUMBER = 0
CALL CHECKREAD('MISC',LU_INPUT,IOS)
IF (IOS==0) READ(LU_INPUT,MISC)
REWIND(LU_INPUT) ; INPUT_FILE_LINE_NUMBER = 0

! Temperature conversions

TMPA  = TMPA + TMPM
TMPA4 = TMPA**4

! Miscellaneous

ASSUMED_GAS_TEMPERATURE = ASSUMED_GAS_TEMPERATURE + TMPM
TEX_ORI = TEXTURE_ORIGIN
GRAV = SQRT(DOT_PRODUCT(GVEC,GVEC))

! Velocity, force, and gravity ramps

```

```
I_RAMP_AGT   = 0
I_RAMP_GX    = 0
I_RAMP_GY    = 0
I_RAMP_GZ    = 0
N_RAMP       = 0
ALLOCATE(RAMP_ID(100))
ALLOCATE(RAMP_TYPE(100))
IF (ASSUMED_GAS_TEMPERATURE_RAMP/='null') CALL GET_RAMP_INDEX(ASSUMED_GAS_TEMPERATURE_RAMP,'TIME',I_RAMP_AGT)
IF (RAMP_GX/='null') CALL GET_RAMP_INDEX(RAMP_GX,'TIME',I_RAMP_GX)
IF (RAMP_GY/='null') CALL GET_RAMP_INDEX(RAMP_GY,'TIME',I_RAMP_GY)
IF (RAMP_GZ/='null') CALL GET_RAMP_INDEX(RAMP_GZ,'TIME',I_RAMP_GZ)

! Prandtl and Schmidt numbers

IF (DNS) THEN
LES              = .FALSE.
IF (PR<0._EB) PR = 0.7_EB
IF (SC<0._EB) SC = 1.0_EB
ELSE
IF (PR<0._EB) PR = 0.5_EB
IF (SC<0._EB) SC = 0.5_EB
ENDIF

RSC = 1._EB/SC
RPR = 1._EB/PR

! Check for a restart file

APPEND = .FALSE.
IF (RESTART .AND. RESTART_CHID == CHID) APPEND = .TRUE.
IF (RESTART) NOISE  = .FALSE.

! Min and Max values of flux limiter

IF (FLUX_LIMITER<0 .OR. FLUX_LIMITER>5) THEN
WRITE(MESSAGE,'(A)') 'ERROR on MISC: Permissible values for FLUX_LIMITER=0:5'
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF

!Sesa-Combustion Model
SELECT CASE (TRIM(COMBUSTION_MODEL_SELECT))
CASE ('COMBUSTION_SIX')
COMB_MODEL=COMBUSTIONSIX
CASE ('COMBUSTION_TWO')
COMB_MODEL=COMBUSTIONTWO
END SELECT
!Sesa

! Turbulence model

SELECT CASE (TRIM(TURBULENCE_MODEL))
CASE ('CONSTANT SMAGORINSKY')
TURB_MODEL=CONSMAG
CASE ('DYNAMIC SMAGORINSKY')
TURB_MODEL=DYNSMAG
CASE ('DEARDORFF')
TURB_MODEL=DEARDORFF
CASE ('VREMAN')
TURB_MODEL=VREMAN
CASE ('RNG')
TURB_MODEL=RNG
CASE ('WALE')
TURB_MODEL=WALE
CASE ('null')
TURB_MODEL=NO_TURB_MODEL
CASE DEFAULT
WRITE(MESSAGE,'(A,A,A)') 'ERROR: TURBULENCE_MODEL, ',TRIM(TURBULENCE_MODEL),', is not recognized.'
CALL SHUTDOWN(MESSAGE) ; RETURN
END SELECT

! Near wall eddy viscosity model

SELECT CASE (TRIM(NEAR_WALL_TURBULENCE_MODEL))
CASE DEFAULT
NEAR_WALL_TURB_MODEL=CONSMAG
CASE ('WALE')
NEAR_WALL_TURB_MODEL=WALE
END SELECT

! Extinction Model

IF (TRIM(EXTINCTION_MODEL)/='null') THEN
SELECT CASE (TRIM(EXTINCTION_MODEL))
CASE ('EXTINCTION 1')
EXTINCT_MOD = EXTINCTION_1
CASE ('EXTINCTION 2')
!Single-step product based calculation
EXTINCT_MOD = EXTINCTION_2
```

```
CASE ('EXTINCTION 3')
!Two-step fuel -> CO, CO-> CO2
EXTINCT_MOD = EXTINCTION_3
CASE DEFAULT
WRITE(MESSAGE,'(A,A,A)') 'ERROR: EXTINCTION_MODEL, ',TRIM(EXTINCTION_MODEL),', is not recognized.'
CALL SHUTDOWN(MESSAGE) ; RETURN
END SELECT
ELSE
EXTINCT_MOD = EXTINCTION_2
EXTINCTION_MODEL = 'EXTINCTION 2'
ENDIF

! Check range of INITIAL_UNMIXED_FRACTION

IF (INITIAL_UNMIXED_FRACTION<0._EB .OR. INITIAL_UNMIXED_FRACTION>1._EB) THEN
WRITE(MESSAGE,'(A)') 'ERROR on MISC: Permissible values for INITIAL_UNMIXED_FRACTION=[0,1]'
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF

! Level set based model of firespread in vegetation

IF (VEG_LEVEL_SET_COUPLED)   VEG_LEVEL_SET = .TRUE.
IF (VEG_LEVEL_SET_UNCOUPLED) VEG_LEVEL_SET = .TRUE.
IF (VEG_LEVEL_SET_UNCOUPLED) WIND_ONLY = .TRUE.

IF (HRRPUV_MAX_SMV<0.0) THEN
HRRPUV_MAX_SMV=1200.0
USE_HRRPUV_MAX_SMV=0
ELSE
USE_HRRPUV_MAX_SMV=1
ENDIF

! Set the lower limit of the extinction coefficient

EC_LL = VISIBILITY_FACTOR/MAXIMUM_VISIBILITY

IF (HUMIDITY<0._EB) HUMIDITY=40._EB

! Do not allow predefined SURF as DEFAULT

IF (TRIM(SURF_DEFAULT)=='OPEN'                .OR. &
TRIM(SURF_DEFAULT)=='MIRROR'              .OR. &
TRIM(SURF_DEFAULT)=='INTERPOLATED'        .OR. &
TRIM(SURF_DEFAULT)=='PERIODIC'            .OR. &
TRIM(SURF_DEFAULT)=='HVAC'                .OR. &
TRIM(SURF_DEFAULT)=='MASSLESS TRACER'     .OR. &
TRIM(SURF_DEFAULT)=='DROPLET'             .OR. &
TRIM(SURF_DEFAULT)=='VEGETATION'          .OR. &
TRIM(SURF_DEFAULT)=='EVACUATION_OUTFLOW'  .OR. &
TRIM(SURF_DEFAULT)=='MASSLESS TARGET'         ) THEN
WRITE (MESSAGE,'(A,A,A)') 'ERROR: Problem with MISC. Cannot set predefined SURF as SURF_DEFAULT'
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF

!Sesa-added as in 6.2.0
!Change units of combustion quantities

HRRPUV_AVERAGE = HRRPUV_AVERAGE*1000._EB    !W/m3
HRRPUA_SHEET   = HRRPUA_SHEET*  1000._EB    !W.m3
!Sesa-added end


FUEL_SMIX_INDEX=2

H_F_REFERENCE_TEMPERATURE = H_F_REFERENCE_TEMPERATURE + TMPM

END SUBROUTINE READ_MISC


SUBROUTINE READ_WIND

USE MATH_FUNCTIONS, ONLY: GET_RAMP_INDEX
USE PHYSICAL_FUNCTIONS, ONLY: MONIN_OBUKHOV_SIMILARITY
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
INTEGER :: NM
LOGICAL :: INITIALIZATION_ONLY
CHARACTER(LABEL_LENGTH) :: RAMP_FVX_T,RAMP_FVY_T,RAMP_FVZ_T,RAMP_U0_T,RAMP_V0_T,RAMP_W0_T,&
RAMP_U0_Z,RAMP_V0_Z,RAMP_W0_Z,RAMP_TMP0_Z,RAMP_DIRECTION,RAMP_SPEED
TYPE(RESERVED_RAMPS_TYPE), POINTER :: RRP,RRP2
EQUIVALENCE(Z_0,AERODYNAMIC_ROUGHNESS)
EQUIVALENCE(Z_REF,REFERENCE_HEIGHT)
EQUIVALENCE(TMP_REF,REFERENCE_TEMPERATURE)
EQUIVALENCE(L,OBUKHOV_LENGTH)
REAL(EB), PARAMETER :: KAPPA_VK = 0.41_EB

NAMELIST /WIND/ CORIOLIS_VECTOR,DIRECTION,DT_MEAN_FORCING,FORCE_VECTOR,FYI,GROUND_LEVEL,INITIALIZATION_ONLY,L,&
```

```
LAPSE_RATE,MEAN_FORCING,OBUKHOV_LENGTH,&
POTENTIAL_TEMPERATURE_CORRECTION,RAMP_DIRECTION,RAMP_SPEED,RAMP_FVX_T,RAMP_FVY_T,RAMP_FVZ_T,&
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,&
SPEED,SPONGE_CELLS,STRATIFICATION,THETA_STAR,TMP_REF,U_STAR,U0,V0,W0,Z_0,Z_REF

! Default values

DIRECTION             = 270._EB    ! westerly wind
DT_MEAN_FORCING       = 1._EB      ! s
INITIALIZATION_ONLY   = .FALSE.
LAPSE_RATE            = 0._EB      ! K/m
MEAN_FORCING          = .FALSE.
OBUKHOV_LENGTH        = 0._EB      ! m
RAMP_DIRECTION        = 'null'
RAMP_SPEED            = 'null'
RAMP_U0_T             = 'null'
RAMP_V0_T             = 'null'
RAMP_W0_T             = 'null'
RAMP_U0_Z             = 'null'
RAMP_V0_Z             = 'null'
RAMP_W0_Z             = 'null'
RAMP_TMP0_Z           = 'null'
RAMP_FVX_T            = 'null'
RAMP_FVY_T            = 'null'
RAMP_FVZ_T            = 'null'
SPEED                 = -1._EB     ! m/s
SPONGE_CELLS          = 3
THETA_STAR            = 0._EB      ! K
TMP_REF               = -1._EB     ! C
U_STAR                = -1._EB     ! m/s
U0                    = 0._EB      ! m/s
V0                    = 0._EB      ! m/s
W0                    = 0._EB      ! m/s
Z_0                   = 0.03_EB    ! m
Z_REF                 = 2._EB      ! m

! Initial read of the WIND line

REWIND(LU_INPUT) ; INPUT_FILE_LINE_NUMBER = 0
WIND_LOOP: DO
CALL CHECKREAD('WIND',LU_INPUT,IOS)
IF (IOS==1) EXIT WIND_LOOP
READ(LU_INPUT,WIND,END=23,ERR=24,IOSTAT=IOS)
24 IF (IOS>0) THEN ; CALL SHUTDOWN('ERROR: Problem with WIND line') ; RETURN ; ENDIF
ENDDO WIND_LOOP
23 REWIND(LU_INPUT) ; INPUT_FILE_LINE_NUMBER = 0

! Check compatibility of constant_specific_heat_ratio and stratification

IF (CONSTANT_SPECIFIC_HEAT_RATIO .AND. STRATIFICATION) THEN
WRITE(MESSAGE,'(A,A,A)') 'ERROR: CONSTANT_SPECIFIC_HEAT_RATIO option is incompatible with STRATIFICATION.'
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF

! Do not impose MEAN_FORCING if the user just wants to initialize the wind speed

IF (INITIALIZATION_ONLY) MEAN_FORCING = .FALSE.

! Convert wind speed to directions

IF (U_STAR>0._EB) SPEED = U_STAR*LOG(Z_REF/Z_0)/KAPPA_VK
DIRECTION = DIRECTION*PI/180._EB
IF (SPEED>0._EB) THEN
IF (RAMP_DIRECTION/='null') THEN
U0 = SPEED
V0 = SPEED
ELSE
U0 = -SPEED*SIN(DIRECTION)
V0 = -SPEED*COS(DIRECTION)
ENDIF
ELSE
SPEED = SQRT(U0**2 + V0**2)
ENDIF

! Apply mean forcing in all directions if any wind component is non-sero

IF ((U0/=0._EB .OR. V0/=0._EB .OR. W0/=0._EB) .AND. .NOT. INITIALIZATION_ONLY) MEAN_FORCING = .TRUE.

! Miscellaneous

FVEC = FORCE_VECTOR
OVEC = CORIOLIS_VECTOR

! Velocity, force, and gravity ramps

I_RAMP_DIRECTION = 0
I_RAMP_SPEED     = 0
I_RAMP_U0_T = 0
```

```fortran
I_RAMP_V0_T  = 0
I_RAMP_W0_T  = 0
I_RAMP_U0_Z  = 0
I_RAMP_V0_Z  = 0
I_RAMP_W0_Z  = 0
I_RAMP_TMP0_Z  = 0
I_RAMP_FVX_T= 0
I_RAMP_FVY_T= 0
I_RAMP_FVZ_T= 0

IF (RAMP_DIRECTION/='null') THEN  ! create dummy time RAMPs for U0 and V0 and fill in later
RAMP_U0_T = RAMP_DIRECTION
RAMP_V0_T = RAMP_DIRECTION
ENDIF
IF (RAMP_U0_T/='null') CALL GET_RAMP_INDEX(RAMP_U0_T,'TIME',I_RAMP_U0_T,DUPLICATE_RAMP=.TRUE.)
IF (RAMP_V0_T/='null') CALL GET_RAMP_INDEX(RAMP_V0_T,'TIME',I_RAMP_V0_T,DUPLICATE_RAMP=.TRUE.)
IF (RAMP_W0_T/='null') CALL GET_RAMP_INDEX(RAMP_W0_T,'TIME',I_RAMP_W0_T)
IF (RAMP_U0_Z/='null') CALL GET_RAMP_INDEX(RAMP_U0_Z,'PROFILE',I_RAMP_U0_Z)
IF (RAMP_V0_Z/='null') CALL GET_RAMP_INDEX(RAMP_V0_Z,'PROFILE',I_RAMP_V0_Z)
IF (RAMP_W0_Z/='null') CALL GET_RAMP_INDEX(RAMP_W0_Z,'PROFILE',I_RAMP_W0_Z)
IF (RAMP_FVX_T/='null') CALL GET_RAMP_INDEX(RAMP_FVX_T,'TIME',I_RAMP_FVX_T)
IF (RAMP_FVY_T/='null') CALL GET_RAMP_INDEX(RAMP_FVY_T,'TIME',I_RAMP_FVY_T)
IF (RAMP_FVZ_T/='null') CALL GET_RAMP_INDEX(RAMP_FVZ_T,'TIME',I_RAMP_FVZ_T)
IF (RAMP_SPEED/='null') CALL GET_RAMP_INDEX(RAMP_SPEED,'TIME',I_RAMP_SPEED)
IF (RAMP_DIRECTION/='null') CALL GET_RAMP_INDEX(RAMP_DIRECTION,'TIME',I_RAMP_DIRECTION)

IF (STRATIFICATION) THEN

IF (RAMP_TMP0_Z=='null' .AND. ABS(OBUKHOV_LENGTH)<1.E-10_EB) THEN
N_RESERVED_RAMPS = N_RESERVED_RAMPS + 1
RRP => RESERVED_RAMPS(N_RESERVED_RAMPS)
ALLOCATE(RRP%INDEPENDENT_DATA(2))
ALLOCATE(RRP%DEPENDENT_DATA(2))
RRP%INDEPENDENT_DATA(1) = ZS_MIN
RRP%INDEPENDENT_DATA(2) = ZF_MAX
RRP%DEPENDENT_DATA(1)    = (TMPA+LAPSE_RATE*(ZS_MIN-GROUND_LEVEL))/TMPA
RRP%DEPENDENT_DATA(2)    = (TMPA+LAPSE_RATE*(ZF_MAX-GROUND_LEVEL))/TMPA
RRP%NUMBER_DATA_POINTS = 2
RAMP_TMP0_Z = 'RSRVD TEMPERATURE PROFILE'
CALL GET_RAMP_INDEX(RAMP_TMP0_Z,'PROFILE',I_RAMP_TMP0_Z)
ENDIF

IF (ABS(OBUKHOV_LENGTH)>1.E-10_EB) THEN
N_RESERVED_RAMPS = N_RESERVED_RAMPS + 1
RRP => RESERVED_RAMPS(N_RESERVED_RAMPS)
RRP%NUMBER_DATA_POINTS = 51
N_RESERVED_RAMPS = N_RESERVED_RAMPS + 1
RRP2 => RESERVED_RAMPS(N_RESERVED_RAMPS)
RRP2%NUMBER_DATA_POINTS = 51
ALLOCATE(RRP%INDEPENDENT_DATA(51))
ALLOCATE(RRP%DEPENDENT_DATA(51))
ALLOCATE(RRP2%INDEPENDENT_DATA(51))
ALLOCATE(RRP2%DEPENDENT_DATA(51))
IF (U_STAR<0._EB) U_STAR = KAPPA_VK*SPEED/LOG(Z_REF/Z_0)
IF (TMP_REF<0._EB) THEN
TMP_REF = TMPA
ELSE
TMP_REF = TMP_REF + TMPM  ! C to K
ENDIF
P_REF = P_INF - RHOA*GRAV*(Z_REF-Z_0)
THETA_REF = TMP_REF*(P_INF/P_REF)**0.286_EB
IF (ABS(THETA_STAR)<1.E-10_EB) THEN
THETA_0 = THETA_REF/(1._EB+U_STAR**2*LOG(Z_REF/Z_0)/(GRAV*KAPPA_VK**2*OBUKHOV_LENGTH))
THETA_STAR = U_STAR**2*THETA_0/(GRAV*KAPPA_VK*OBUKHOV_LENGTH)
ELSE
THETA_0 = THETA_REF - THETA_STAR*LOG(Z_REF/Z_0)/KAPPA_VK
ENDIF
TMPA = THETA_0  ! Make the ground temperature the new ambient temperature
DO I=1,51
ZETA = (I-1)*(ZF_MAX-ZS_MIN)/50.
ZZZ  = Z_0*EXP(LOG(ZF_MAX/Z_0)*(ZETA-ZS_MIN)/(ZF_MAX-ZS_MIN))
CALL MONIN_OBUKHOV_SIMILARITY(ZZZ,Z_0,OBUKHOV_LENGTH,U_STAR,THETA_STAR,THETA_0,U,TMP)
RRP%INDEPENDENT_DATA(I) = ZZZ
RRP%DEPENDENT_DATA(I)    = TMP/TMPA
RRP2%INDEPENDENT_DATA(I) = ZZZ
RRP2%DEPENDENT_DATA(I)    = U/SPEED
ENDDO
RAMP_TMP0_Z = 'RSRVD TEMPERATURE PROFILE'
CALL GET_RAMP_INDEX(RAMP_TMP0_Z,'PROFILE',I_RAMP_TMP0_Z)
RAMP_U0_Z = 'RSRVD VELOCITY PROFILE'
CALL GET_RAMP_INDEX(RAMP_U0_Z,'PROFILE',I_RAMP_U0_Z)
RAMP_V0_Z = 'RSRVD VELOCITY PROFILE'
CALL GET_RAMP_INDEX(RAMP_V0_Z,'PROFILE',I_RAMP_V0_Z)
ENDIF

! Add a RAMP for the vertical profile of pressure (the values are computed in INIT)

N_RESERVED_RAMPS = N_RESERVED_RAMPS + 1
```

```
CALL GET_RAMP_INDEX('RSRVD PRESSURE PROFILE','PROFILE',I_RAMP_P0_Z)
RRP => RESERVED_RAMPS(N_RESERVED_RAMPS)
ALLOCATE(RRP%INDEPENDENT_DATA(2))
ALLOCATE(RRP%DEPENDENT_DATA(2))
RRP%INDEPENDENT_DATA(1) = ZS_MIN
RRP%INDEPENDENT_DATA(2) = ZF_MAX
RRP%DEPENDENT_DATA(1) = 0._EB      ! Dummy values to be filled in later
RRP%DEPENDENT_DATA(2) = 1._EB      ! Dummy values to be filled in later
RRP%NUMBER_DATA_POINTS = 2

ENDIF

! External kinetic energy

H0 = 0.5_EB*(U0**2+V0**2+W0**2)

! Allocation for mean forcing (required here, instead of init, because of hole feature)

IF (ANY(MEAN_FORCING)) THEN
DO NM=1,NMESHES
IF (MYID/=PROCESS(NM) .OR. EVACUATION_ONLY(NM)) CYCLE
M=>MESHES(NM)
ALLOCATE(M%MEAN_FORCING_CELL(0:M%IBP1,0:M%JBP1,0:M%KBP1),STAT=IZERO)
CALL ChkMemErr('INIT','MEAN_FORCING_CELL',IZERO)
M%MEAN_FORCING_CELL=.TRUE.
ENDDO
ENDIF

! Min and Max values of temperature

TMPMIN = MAX(1._EB , MIN(TMPA,TMPM)-10._EB)
IF (LAPSE_RATE < 0._EB) TMPMIN = MIN(TMPMIN,TMPA+LAPSE_RATE*(ZF_MAX-GROUND_LEVEL))
TMPMAX = 3000._EB

END SUBROUTINE READ_WIND


SUBROUTINE PROC_WIND

! This short routine takes a time ramp of wind speed and direction and converts to Cartesian ramps for U0, V0

REAL(EB) :: THETA
INTEGER :: I

IF (I_RAMP_DIRECTION/=0) THEN
DO I=0,RAMPS(I_RAMP_DIRECTION)%NUMBER_INTERPOLATION_POINTS+1
THETA = RAMPS(I_RAMP_DIRECTION)%INTERPOLATED_DATA(I)*PI/180._EB
RAMPS(I_RAMP_U0_T)%INTERPOLATED_DATA(I) = -SIN(THETA)
RAMPS(I_RAMP_V0_T)%INTERPOLATED_DATA(I) = -COS(THETA)
ENDDO
ENDIF

END SUBROUTINE PROC_WIND


SUBROUTINE READ_DUMP

! Read parameters associated with output files

REAL(EB) :: DT_DEFAULT
INTEGER :: N,SIG_FIGS,SIG_FIGS_EXP
NAMELIST /DUMP/ CLIP_RESTART_FILES,COLUMN_DUMP_LIMIT,CTRL_COLUMN_LIMIT,&
DEVC_COLUMN_LIMIT,DT_BNDE,DT_BNDF,DT_CPU,DT_CTRL,DT_DEVC,DT_DEVC_LINE,DT_FLUSH,&
DT_GEOM,DT_HRR,DT_ISOF,DT_MASS,DT_PART,DT_PL3D,DT_PROF,DT_RESTART,DT_SL3D,DT_SLCF,EB_PART_FILE,&
FLUSH_FILE_BUFFERS,GEOM_DIAG,MASS_FILE,MAXIMUM_PARTICLES,MMS_TIMER,NFRAMES,PLOT3D_PART_ID,PLOT3D_QUANTITY,&
PLOT3D_SPEC_ID,PLOT3D_SPEC_ID,PLOT3D_VELO_INDEX,RENDER_FILE,SIG_FIGS,SIG_FIGS_EXP,SMOKE3D,SMOKE3D_QUANTITY,&
SMOKE3D_SPEC_ID,STATUS_FILES,SUPPRESS_DIAGNOSTICS,UVW_TIMER,VELOCITY_ERROR_FILE,WRITE_XYZ

! Set defaults

GEOM_DIAG = .FALSE.
FLUSH_FILE_BUFFERS = .TRUE.
MAXIMUM_PARTICLES    = 1000000
MMS_TIMER            = 1.E10_EB
NFRAMES              = 1000
PLOT3D_QUANTITY(1)   = 'TEMPERATURE'
PLOT3D_QUANTITY(2)   = 'U-VELOCITY'
PLOT3D_QUANTITY(3)   = 'V-VELOCITY'
PLOT3D_QUANTITY(4)   = 'W-VELOCITY'
PLOT3D_QUANTITY(5)   = 'HRRPUV'
PLOT3D_PART_ID       = 'null'
PLOT3D_SPEC_ID       = 'null'
PLOT3D_VELO_INDEX    = 0
RENDER_FILE          = 'null'
SMOKE3D              = .TRUE.
SMOKE3D_QUANTITY     = 'null'
SMOKE3D_SPEC_ID      = 'null'
SIG_FIGS             = 8
```

```
SIG_FIGS_EXP         = 3
IF (NMESHES>32) THEN
SUPPRESS_DIAGNOSTICS = .TRUE.
ELSE
SUPPRESS_DIAGNOSTICS = .FALSE.
ENDIF
UVW_TIMER            = 1.E10_EB

DT_GEOM      =  1._EB
DT_BNDE      = -1._EB
DT_BNDF      = -1._EB
DT_CPU       = 1000000._EB
DT_RESTART   = 1000000._EB
DT_FLUSH     = -1._EB
DT_DEVC      = -1._EB
DT_DEVC_LINE = 0.5_EB*(T_END-T_BEGIN)
DT_HRR       = -1._EB
DT_ISOF      = -1._EB
DT_MASS      = -1._EB
DT_PART      = -1._EB
DT_PL3D      =  1.E10_EB
DT_PROF      = -1._EB
DT_SLCF      = -1._EB
DT_SL3D      = 0.2_EB*(T_END-T_BEGIN)
DT_CTRL      = -1._EB

! Read the DUMP line

REWIND(LU_INPUT) ; INPUT_FILE_LINE_NUMBER = 0
DUMP_LOOP: DO
CALL CHECKREAD('DUMP',LU_INPUT,IOS)
IF (IOS==1) EXIT DUMP_LOOP
READ(LU_INPUT,DUMP,END=23,ERR=24,IOSTAT=IOS)
24 IF (IOS>0) THEN ; CALL SHUTDOWN('ERROR: Problem with DUMP line') ; RETURN ; ENDIF
ENDDO DUMP_LOOP
23 REWIND(LU_INPUT) ; INPUT_FILE_LINE_NUMBER = 0

! Set output time intervals

DT_DEFAULT = (T_END - T_BEGIN)/REAL(NFRAMES,EB)

IF (DT_BNDE < 0._EB) THEN ; DT_BNDE = 2._EB*DT_DEFAULT ; ELSE ; DT_BNDE = DT_BNDE /TIME_SHRINK_FACTOR ; ENDIF
IF (DT_BNDF < 0._EB) THEN ; DT_BNDF = 2._EB*DT_DEFAULT ; ELSE ; DT_BNDF = DT_BNDF /TIME_SHRINK_FACTOR ; ENDIF
IF (DT_DEVC < 0._EB) THEN ; DT_DEVC = DT_DEFAULT       ; ELSE ; DT_DEVC = DT_DEVC /TIME_SHRINK_FACTOR ; ENDIF
IF (DT_HRR  < 0._EB) THEN ; DT_HRR  = DT_DEFAULT       ; ELSE ; DT_HRR  = DT_HRR  /TIME_SHRINK_FACTOR ; ENDIF
IF (DT_ISOF < 0._EB) THEN ; DT_ISOF = DT_DEFAULT       ; ELSE ; DT_ISOF = DT_ISOF /TIME_SHRINK_FACTOR ; ENDIF
IF (DT_MASS < 0._EB) THEN ; DT_MASS = DT_DEFAULT       ; ELSE ; DT_MASS = DT_MASS /TIME_SHRINK_FACTOR ; ENDIF
IF (DT_PART < 0._EB) THEN ; DT_PART = DT_DEFAULT       ; ELSE ; DT_PART = DT_PART /TIME_SHRINK_FACTOR ; ENDIF
IF (DT_PROF < 0._EB) THEN ; DT_PROF = DT_DEFAULT       ; ELSE ; DT_PROF = DT_PROF /TIME_SHRINK_FACTOR ; ENDIF
IF (DT_SLCF < 0._EB) THEN ; DT_SLCF = DT_DEFAULT       ; ELSE ; DT_SLCF = DT_SLCF /TIME_SHRINK_FACTOR ; ENDIF
IF (DT_CTRL < 0._EB) THEN ; DT_CTRL = DT_DEFAULT       ; ELSE ; DT_CTRL = DT_CTRL /TIME_SHRINK_FACTOR ; ENDIF
IF (DT_FLUSH< 0._EB) THEN ; DT_FLUSH= DT_DEFAULT       ; ELSE ; DT_FLUSH= DT_FLUSH/TIME_SHRINK_FACTOR ; ENDIF

! Check Plot3D QUANTITIES

PLOOP: DO N=1,5
CALL GET_QUANTITY_INDEX(PLOT3D_SMOKEVIEW_LABEL(N),PLOT3D_SMOKEVIEW_BAR_LABEL(N),PLOT3D_QUANTITY_INDEX(N),I_DUM(1)
   , &
PLOT3D_Y_INDEX(N),PLOT3D_Z_INDEX(N),PLOT3D_PART_INDEX(N),I_DUM(2),I_DUM(3),I_DUM(4),'PLOT3D', &
PLOT3D_QUANTITY(N),'null',PLOT3D_SPEC_ID(N),PLOT3D_PART_ID(N),'null','null','null')
IF (OUTPUT_QUANTITY(PLOT3D_QUANTITY_INDEX(N))%INTEGRATED_PARTICLES) PL3D_PARTICLE_FLUX = .TRUE.
ENDDO PLOOP

! Check SMOKE3D viability

IF (TWO_D .OR. SOLID_PHASE_ONLY) SMOKE3D = .FALSE.

IF (SMOKE3D_QUANTITY=='null') THEN
IF (SOOT_INDEX > 0)   THEN
SMOKE3D_QUANTITY = 'MASS FRACTION'
SMOKE3D_SPEC_ID  = SPECIES(SOOT_INDEX)%ID
ELSE
IF (N_REACTIONS > 0)   THEN
SMOKE3D_QUANTITY = 'HRRPUV'
ELSE
SMOKE3D = .FALSE.
ENDIF
ENDIF
ENDIF

IF (SMOKE3D) THEN
CALL GET_QUANTITY_INDEX(SMOKE3D_SMOKEVIEW_LABEL,SMOKE3D_SMOKEVIEW_BAR_LABEL,SMOKE3D_QUANTITY_INDEX,I_DUM(1), &
SMOKE3D_Y_INDEX,SMOKE3D_Z_INDEX,I_DUM(2),I_DUM(3),I_DUM(4),I_DUM(5),'SMOKE3D', &
SMOKE3D_QUANTITY,'null',SMOKE3D_SPEC_ID,'null','null','null','null')
ENDIF

! Set format of real number output

```

```fortran
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

END SUBROUTINE READ_DUMP


SUBROUTINE READ_SPEC

USE MATH_FUNCTIONS, ONLY : GET_RAMP_INDEX
USE PHYSICAL_FUNCTIONS, ONLY : WATER_VAPOR_MASS_FRACTION
USE PROPERTY_DATA, ONLY: GAS_PROPS,FED_PROPS,CHECK_PREDEFINED
USE SOOT_ROUTINES
REAL(EB) :: MW,SIGMALJ,EPSILONKLJ,VISCOSITY,CONDUCTIVITY,DIFFUSIVITY,MASS_EXTINCTION_COEFFICIENT, &
SPECIFIC_HEAT,REFERENCE_ENTHALPY,REFERENCE_TEMPERATURE,FIC_CONCENTRATION,FLD_LETHAL_DOSE,&
SPECIFIC_HEAT_LIQUID,DENSITY_LIQUID,VAPORIZATION_TEMPERATURE,HEAT_OF_VAPORIZATION,MELTING_TEMPERATURE,&
H_V_REFERENCE_TEMPERATURE,MEAN_DIAMETER,CONDUCTIVITY_SOLID,DENSITY_SOLID,ENTHALPY_OF_FORMATION,MASS_FRACTION_0,&
CONVERSION,PR_GAS,CONDUCTIVITY_LIQUID,VISCOSITY_LIQUID,BETA_LIQUID,H_F_IN
REAL(EB):: MASS_FRACTION(MAX_SPECIES),VOLUME_FRACTION(MAX_SPECIES),MIN_DIAMETER,MAX_DIAMETER
INTEGER :: N_SPEC_READ,N,NN,NNN,NS2,NR,N_SPEC_READ_2,N_SUB_SPECIES,NS,N_BINS,N_COPY,N_FOUND
INTEGER, ALLOCATABLE, DIMENSION(:) :: Y_INDEX
LOGICAL :: LUMPED_COMPONENT_ONLY,AEROSOL,BACKGROUND,&
DEFINED_BACKGROUND,REAC_FUEL_READ=.FALSE.,PRIMITIVE,COPY_LUMPED
LOGICAL, ALLOCATABLE, DIMENSION(:) :: PREDEFINED,PREDEFINED_SMIX,NEW_PRIMITIVE
CHARACTER(LABEL_LENGTH) :: RAMP_CP,RAMP_CP_L,RAMP_K,RAMP_MU,RAMP_D,RADCAL_ID,RAMP_G_F,SPEC_ID(MAX_SPECIES)

CHARACTER(LABEL_LENGTH), ALLOCATABLE, DIMENSION(:) :: PREDEFINED_SPEC_ID,SPEC_ID_READ
CHARACTER(LABEL_LENGTH) :: FORMULA
TYPE(SPECIES_TYPE), POINTER :: SS=>NULL()
TYPE(SPECIES_MIXTURE_TYPE), POINTER :: SM=>NULL()
NAMELIST /SPEC/ AEROSOL,BACKGROUND,BETA_LIQUID,CONDUCTIVITY,CONDUCTIVITY_LIQUID,CONDUCTIVITY_SOLID,COPY_LUMPED, &
DENSITY_LIQUID,DENSITY_SOLID,DIFFUSIVITY,ENTHALPY_OF_FORMATION,EPSILONKLJ,FIC_CONCENTRATION,FLD_LETHAL_DOSE, &
FORMULA,FYI,HEAT_OF_VAPORIZATION,H_V_REFERENCE_TEMPERATURE,ID,LUMPED_COMPONENT_ONLY,&
MASS_EXTINCTION_COEFFICIENT,MASS_FRACTION,MASS_FRACTION_0,MAX_DIAMETER,MEAN_DIAMETER,MELTING_TEMPERATURE,&
MIN_DIAMETER,MW,N_BINS,PR_GAS,PRIMITIVE,RADCAL_ID,&
RAMP_CP,RAMP_CP_L,RAMP_D,RAMP_G_F,RAMP_K,RAMP_MU,REFERENCE_ENTHALPY,REFERENCE_TEMPERATURE,SIGMALJ,SPEC_ID,&
SPECIFIC_HEAT,SPECIFIC_HEAT_LIQUID,VAPORIZATION_TEMPERATURE,VISCOSITY,VISCOSITY_LIQUID,VOLUME_FRACTION

IF (SIMPLE_CHEMISTRY) THEN
MINT_SPECIES = 9
ELSE
MINT_SPECIES = 5
ENDIF
REWIND(LU_INPUT) ; INPUT_FILE_LINE_NUMBER = 0
N_SPECIES=0

COUNT_SPEC_LINES: DO
READ(LU_INPUT,NML=SPEC,END=19,ERR=20,IOSTAT=IOS)
MINT_SPECIES=MINT_SPECIES+1
IF (N_PARTICLE_BINS > 0) THEN
MINT_SPECIES=MINT_SPECIES+N_PARTICLE_BINS
ENDIF
20 IF (IOS>0) THEN
WRITE(MESSAGE,'(A,I0,A,I0)') 'ERROR: Problem with SPECies number',N_SPECIES+1,', line number',
      INPUT_FILE_LINE_NUMBER
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF
N_SPECIES = N_SPECIES+1
ENDDO COUNT_SPEC_LINES
19 REWIND(LU_INPUT) ; INPUT_FILE_LINE_NUMBER = 0

!Allocate species inputs
ALLOCATE(PREDEFINED(MINT_SPECIES))
ALLOCATE(PREDEFINED_SMIX(1:MINT_SPECIES))
ALLOCATE(NEW_PRIMITIVE(0:MINT_SPECIES))
NEW_PRIMITIVE=.FALSE.
ALLOCATE(PREDEFINED_SPEC_ID(MINT_SPECIES))
ALLOCATE(SPEC_ID_READ(MINT_SPECIES))

! Create predefined inputs related to simple chemistry mode

PREDEFINED = .FALSE.
PREDEFINED_SMIX = .FALSE.

N_SPEC_READ = 0
N_TRACKED_SPECIES = 1

IF (SIMPLE_CHEMISTRY) THEN
N_SPECIES = 7
PREDEFINED(1:7)          = .TRUE.
PREDEFINED_SPEC_ID(1) = REACTION(1)%FUEL
PREDEFINED_SPEC_ID(2) = 'NITROGEN'
PREDEFINED_SPEC_ID(3) = 'OXYGEN'
PREDEFINED_SPEC_ID(4) = 'CARBON DIOXIDE'
PREDEFINED_SPEC_ID(5) = 'CARBON MONOXIDE'
PREDEFINED_SPEC_ID(6) = 'WATER VAPOR'
PREDEFINED_SPEC_ID(7) = 'SOOT'
PREDEFINED_SMIX(1:3) = .TRUE.
IF (REACTION(1)%C <=0._EB .AND. REACTION(1)%H <=0._EB) PREDEFINED_SMIX(2)=.FALSE.
N_TRACKED_SPECIES = 3
```

```
ELSE
! If not simple chemistry look for a background species and if none force creation of AIR lumped species
REWIND(LU_INPUT) ; INPUT_FILE_LINE_NUMBER = 0
CHECK_BACKGROUND_LOOP: DO
BACKGROUND = .FALSE.
READ(LU_INPUT,NML=SPEC,END=21,IOSTAT=IOS)
IF (BACKGROUND) EXIT
ENDDO CHECK_BACKGROUND_LOOP
21 REWIND(LU_INPUT) ; INPUT_FILE_LINE_NUMBER = 0
IF (BACKGROUND) THEN
N_SPECIES = 0
DEFINED_BACKGROUND = .TRUE.
ELSE
N_SPECIES = 4
PREDEFINED(1:4)          = .TRUE.
PREDEFINED_SPEC_ID(1) = 'NITROGEN'
PREDEFINED_SPEC_ID(2) = 'OXYGEN'
PREDEFINED_SPEC_ID(3) = 'CARBON DIOXIDE'
PREDEFINED_SPEC_ID(4) = 'WATER VAPOR'
PREDEFINED_SMIX(1) = .TRUE.
DEFINED_BACKGROUND = .FALSE.
ENDIF
ENDIF

! Pass 1: Count SPEC lines determine number of primitive and tracked species and check for errors

REWIND(LU_INPUT) ; INPUT_FILE_LINE_NUMBER = 0
COUNT_SPEC_LOOP: DO
CALL SET_SPEC_DEFAULT
CALL CHECKREAD('SPEC',LU_INPUT,IOS)
IF (IOS==1) EXIT COUNT_SPEC_LOOP
READ(LU_INPUT,NML=SPEC,END=29,IOSTAT=IOS)
N_SPEC_READ = N_SPEC_READ + 1
SPEC_ID_READ(N_SPEC_READ) = ID

IF (ID=='null') THEN
WRITE(MESSAGE,'(A,I0,A)') 'ERROR: Species ',N_SPEC_READ, ' needs a name (ID=...)'
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF

! Prevent use of 'AIR' unless a new BACKGROUND has been defined.

IF (ID=='AIR' .AND. .NOT. DEFINED_BACKGROUND) THEN
WRITE(MESSAGE,'(A,I0,A)') 'ERROR: SPEC ',N_SPEC_READ,': Cannot redefine AIR without defining a BACKGROUND species
 '
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF

! Make sure both ramps and constant values have not been given

IF (SPECIFIC_HEAT > 0._EB .AND. RAMP_CP/='null') THEN
WRITE(MESSAGE,'(A,A,A)') 'ERROR: SPEC ',TRIM(ID),': Cannot specify both SPECIFIC_HEAT and RAMP_CP'
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF
IF (SPECIFIC_HEAT_LIQUID > 0._EB .AND. RAMP_CP_L/='null') THEN
WRITE(MESSAGE,'(A,A,A)') 'ERROR: SPEC ',TRIM(ID),': Cannot specify both SPECIFIC_HEAT_LIQUID and RAMP_CP_L'
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF
IF (CONDUCTIVITY > 0._EB .AND. RAMP_K/='null') THEN
WRITE(MESSAGE,'(A,A,A)') 'ERROR: SPEC ',TRIM(ID),': Cannot specify both CONDUCTIVITY and RAMP_K'
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF
IF (DIFFUSIVITY > 0._EB .AND. RAMP_D/='null') THEN
WRITE(MESSAGE,'(A,A,A)') 'ERROR: SPEC ',TRIM(ID),': Cannot specify both DIFFUSIVITY and RAMP_D'
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF
IF (VISCOSITY > 0._EB .AND. RAMP_MU/='null') THEN
WRITE(MESSAGE,'(A,A,A)') 'ERROR: SPEC ',TRIM(ID),': Cannot specify both VISCOSISTY and RAMP_MU'
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF

! REFERENCE_ENTHALPY requires additional parameters

IF (REFERENCE_ENTHALPY > -2.E20_EB .AND. (SPECIFIC_HEAT < 0._EB .AND. RAMP_CP=='null')) THEN
WRITE(MESSAGE,'(A,A,A)') 'ERROR: SPEC ',TRIM(ID),': REFERENCE_ENTHALPY requires SPECIFIC_HEAT or RAMP_CP'
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF

DO NN = 1,N_SPEC_READ-1
IF (ID==SPEC_ID_READ(NN)) THEN
WRITE(MESSAGE,'(A,I0,A,I0,A)') 'ERROR: Species ',N_SPEC_READ,' has the same ID as species ',NN, '.'
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF
ENDDO

IF (BACKGROUND) THEN
IF (LUMPED_COMPONENT_ONLY) THEN
WRITE(MESSAGE,'(A)') 'ERROR: Cannot define a LUMPED_COMPONENT_ONLY species as the BACKGROUND species'
```

```
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF
IF (SIMPLE_CHEMISTRY) THEN
WRITE(MESSAGE,'(A)') 'ERROR: Cannot define a BACKGROUND species if using the simple chemistry'
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF
ENDIF
IF (LUMPED_COMPONENT_ONLY .AND. MASS_FRACTION_0>0._EB) THEN
WRITE(MESSAGE,'(A)') 'ERROR: Cannot define MASS_FRACTION_0 for a LUMPED_COMPONENT_ONLY species'
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF

IF (PRIMITIVE) THEN
IF (SPEC_ID(1)/='null' .AND. SPEC_ID(2)=='null') THEN
NEW_PRIMITIVE(N_SPEC_READ)=.TRUE.
IF (.NOT. CHECK_PREDEFINED(SPEC_ID(1))) THEN
WRITE(MESSAGE,'(A,I0,A,I0,A)') 'ERROR: SPEC_ID(1) for species ',N_SPEC_READ,' must be a predefined species'
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF
ELSEIF (SPEC_ID(1)/='null' .AND. SPEC_ID(2)/='null') THEN
WRITE(MESSAGE,'(A,I0,A,I0,A)') 'ERROR: Species ',N_SPEC_READ,' is declared PRIMITIVE and has more than one
     SPEC_ID given'
CALL SHUTDOWN(MESSAGE)          ; RETURN
ENDIF
ENDIF

IF (SPEC_ID(1)=='null' .OR. PRIMITIVE) THEN
N_SPECIES = N_SPECIES+1
IF (SIMPLE_CHEMISTRY) THEN
IF(TRIM(ID)/=TRIM(REACTION(1)%FUEL)) THEN
IF (.NOT. LUMPED_COMPONENT_ONLY .AND. .NOT. BACKGROUND) N_TRACKED_SPECIES = N_TRACKED_SPECIES + 1
ENDIF
ELSE
IF (.NOT. LUMPED_COMPONENT_ONLY .AND. .NOT. BACKGROUND) N_TRACKED_SPECIES = N_TRACKED_SPECIES + 1
ENDIF
ELSE
IF (SIMPLE_CHEMISTRY) THEN
IF(TRIM(ID)/=TRIM(REACTION(1)%FUEL)) N_TRACKED_SPECIES = N_TRACKED_SPECIES + 1
ELSE
IF (.NOT. BACKGROUND) N_TRACKED_SPECIES = N_TRACKED_SPECIES + 1
ENDIF
ENDIF

! If predefined species, check to see if the species has already been defined.
IF (PREDEFINED_SMIX(1)) THEN
DO NN=1,N_SPECIES-1
IF (TRIM(PREDEFINED_SPEC_ID(NN))==TRIM(ID)) THEN
IF (.NOT. SIMPLE_CHEMISTRY) THEN
IF (SPEC_ID(1)/='null') THEN
WRITE(MESSAGE,'(A,I0,A)') 'ERROR: Species ',N_SPEC_READ+1, &
'. Lumped species has the same ID as a predefined species'
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF
ELSE
IF (SPEC_ID(1)/='null') THEN
IF (TRIM(ID)/=TRIM(REACTION(1)%FUEL)) THEN
WRITE(MESSAGE,'(A,I0,A)') 'ERROR: Species ',N_SPEC_READ+1, &
'. Lumped species has the same ID as a predefined species'
CALL SHUTDOWN(MESSAGE) ; RETURN
ELSE
PREDEFINED_SMIX(2) = .FALSE.
REAC_FUEL_READ = .TRUE.
ENDIF
ELSE
IF (TRIM(ID)==TRIM(REACTION(1)%FUEL)) REAC_FUEL_READ = .TRUE.
ENDIF
ENDIF
PREDEFINED(NN) = .FALSE.
N_SPECIES = N_SPECIES - 1
EXIT
ENDIF
ENDDO
ENDIF
ENDDO COUNT_SPEC_LOOP

29 REWIND(LU_INPUT) ; INPUT_FILE_LINE_NUMBER = 0

IF (SIMPLE_CHEMISTRY .AND. .NOT. (REAC_FUEL_READ .OR. SIMPLE_FUEL_DEFINED)) THEN
WRITE(MESSAGE,'(A,A,A)') 'ERROR: Simple chemistry fuel, ',TRIM(REACTION(1)%FUEL),', not defined on REAC or SPEC.'
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF

! Allocate the primitive species array.
ALLOCATE(SPECIES(N_SPECIES),STAT=IZERO)
CALL ChkMemErr('READ','SPECIES',IZERO)

ALLOCATE(Y_INDEX(N_SPECIES))

```

```
! Pass 2: read and process primitive species
N = 0
N_SPEC_READ_2 = N_SPEC_READ
N_SPEC_READ = 0

PRIMITIVE_SPEC_READ_LOOP: DO WHILE (N_SPEC_READ < N_SPEC_READ_2 .OR. N < N_SPECIES)
N = N + 1

CALL SET_SPEC_DEFAULT

IF (PREDEFINED(N)) THEN
ID = PREDEFINED_SPEC_ID(N)
LUMPED_COMPONENT_ONLY = .TRUE.
ELSE
READ(LU_INPUT,NML=SPEC)
N_SPEC_READ = N_SPEC_READ + 1
IF (SIMPLE_CHEMISTRY) THEN
IF(TRIM(ID)==TRIM(REACTION(1)%FUEL)) PREDEFINED_SMIX(2)=.FALSE.
ENDIF

IF (SPEC_ID(1)/='null' .AND. .NOT. NEW_PRIMITIVE(N_SPEC_READ)) THEN
N = N - 1
CYCLE PRIMITIVE_SPEC_READ_LOOP
ENDIF

ENDIF

SS => SPECIES(N)

SS%K_USER                          = CONDUCTIVITY
SS%CONDUCTIVITY_SOLID              = CONDUCTIVITY_SOLID
SS%D_USER                          = DIFFUSIVITY
SS%DENSITY_SOLID                   = DENSITY_SOLID
SS%EPSK                            = EPSILONKLJ
SS%FIC_CONCENTRATION               = FIC_CONCENTRATION
SS%FLD_LETHAL_DOSE                 = FLD_LETHAL_DOSE
SS%FORMULA                         = FORMULA
SS%H_F                             = ENTHALPY_OF_FORMATION*1000._EB
SS%ID                              = ID
SS%RADCAL_ID                       = RADCAL_ID
SS%MASS_EXTINCTION_COEFFICIENT     = MAX(0._EB,MASS_EXTINCTION_COEFFICIENT)
SS%MEAN_DIAMETER                   = MEAN_DIAMETER
SS%MU_USER                         = VISCOSITY
SS%MW                              = MW
SS%PR_USER                         = PR_GAS
SS%RAMP_CP                         = RAMP_CP
SS%RAMP_CP_L                       = RAMP_CP_L
SS%RAMP_D                          = RAMP_D
SS%RAMP_K                          = RAMP_K
SS%RAMP_G_F                        = RAMP_G_F
SS%RAMP_MU                         = RAMP_MU
IF (REFERENCE_TEMPERATURE < -TMPM) REFERENCE_TEMPERATURE = 25._EB
SS%REFERENCE_TEMPERATURE           = REFERENCE_TEMPERATURE + TMPM
SS%SIG                             = SIGMALJ
SS%SPECIFIC_HEAT                   = SPECIFIC_HEAT*1000._EB
SS%REFERENCE_ENTHALPY              = REFERENCE_ENTHALPY*1000._EB
SS%YY0                             = MAX(0._EB,MASS_FRACTION_0)

SS%DENSITY_LIQUID                  = DENSITY_LIQUID
SS%BETA_LIQUID                     = BETA_LIQUID
SS%K_LIQUID                        = CONDUCTIVITY_LIQUID
SS%MU_LIQUID                       = VISCOSITY_LIQUID

IF ((HEAT_OF_VAPORIZATION >  0._EB .AND. SPECIFIC_HEAT_LIQUID <= 0._EB) .OR. &
(HEAT_OF_VAPORIZATION <= 0._EB .AND. SPECIFIC_HEAT_LIQUID >  0._EB)) THEN
WRITE(MESSAGE,'(A,I0,A)') 'ERROR: SPEC ' ,N_SPEC_READ, &
': If one of SPECIFIC_HEAT_LIQUID or HEAT_OF_VAPORIZATION defined, both must be'
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF
IF (SPECIFIC_HEAT_LIQUID > 0._EB) THEN
IF (MELTING_TEMPERATURE < -TMPM) THEN
WRITE(MESSAGE,'(A,I0,A)') 'ERROR: SPEC ' ,N_SPEC_READ,': MELTING_TEMPERATURE not set'
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF
SS%SPECIFIC_HEAT_LIQUID            = SPECIFIC_HEAT_LIQUID*1000._EB
SS%HEAT_OF_VAPORIZATION            = HEAT_OF_VAPORIZATION*1000._EB
SS%TMP_MELT = MELTING_TEMPERATURE + TMPM
IF (H_V_REFERENCE_TEMPERATURE < -TMPM) H_V_REFERENCE_TEMPERATURE = MELTING_TEMPERATURE
SS%H_V_REFERENCE_TEMPERATURE = H_V_REFERENCE_TEMPERATURE + 273.15_EB
IF (VAPORIZATION_TEMPERATURE< -TMPM) THEN
WRITE(MESSAGE,'(A,I0,A)') 'ERROR: SPEC ' ,N_SPEC_READ,': VAPORIZATION_TEMPERATURE not set'
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF
SS%TMP_V = VAPORIZATION_TEMPERATURE + TMPM
ENDIF

IF (N_BINS > 0) THEN
IF (AGGLOMERATION_INDEX>0) THEN
```

```
WRITE(MESSAGE,'(A,I0,A)') 'ERROR: SPEC ',N_SPEC_READ,': Can only specify N_BINS for one &SPEC input'
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF
IF (N_BINS < 2) THEN
WRITE(MESSAGE,'(A,I0,A)') 'ERROR: SPEC ',N_SPEC_READ,': N_BINS must be >=2'
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF
IF (.NOT. AEROSOL) THEN
WRITE(MESSAGE,'(A,I0,A)') 'ERROR: SPEC ',N_SPEC_READ,': AEROSOL must be .TRUE. to use N_BINS'
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF
IF (MAX_DIAMETER < 0._EB) THEN
WRITE(MESSAGE,'(A,I0,A)') 'ERROR: SPEC ',N_SPEC_READ,': MAX_DIAMETER not set'
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF
IF (MIN_DIAMETER < 0._EB) THEN
WRITE(MESSAGE,'(A,I0,A)') 'ERROR: SPEC ',N_SPEC_READ,': MIN_DIAMETER not set'
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF
IF (MAX_DIAMETER <= MIN_DIAMETER) THEN
WRITE(MESSAGE,'(A,I0,A)') 'ERROR: SPEC ',N_SPEC_READ,': MAX_DIAMETER <= MIN_DIAMETER'
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF
IF (.NOT. LUMPED_COMPONENT_ONLY) THEN
WRITE(MESSAGE,'(A,I0,A)') 'ERROR: SPEC ',N_SPEC_READ,': LUMPED_COMPONENT_ONLY must be .TRUE. to use N_BINS'
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF
N_PARTICLE_BINS = N_BINS
MAX_PARTICLE_DIAMETER = MAX_DIAMETER
MIN_PARTICLE_DIAMETER = MIN_DIAMETER
N_TRACKED_SPECIES=N_TRACKED_SPECIES+N_PARTICLE_BINS
AGGLOMERATION_INDEX=N
MIN_PARTICLE_DIAMETER = MIN_PARTICLE_DIAMETER * 1.E-6_EB
MAX_PARTICLE_DIAMETER = MAX_PARTICLE_DIAMETER * 1.E-6_EB
SS%AGGLOMERATING = .TRUE.
CALL INITIALIZE_AGGLOMERATION
ENDIF

IF (NEW_PRIMITIVE(N_SPEC_READ)) THEN
SS%PROP_ID = SPEC_ID(1)
ELSE
SS%PROP_ID = ID
ENDIF
H_F_IN = SS%H_F
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)
IF (SIMPLE_CHEMISTRY) THEN
IF (TRIM(SS%ID)==TRIM(REACTION(1)%FUEL) .AND. .NOT. SS%LISTED) SS%H_F = H_F_IN
ENDIF
CALL FED_PROPS(SS%PROP_ID,SS%FLD_LETHAL_DOSE,SS%FIC_CONCENTRATION)

IF (SS%H_F > -1.E23_EB) SS%EXPLICIT_H_F=.TRUE.

IF (SS%SPECIFIC_HEAT > 0._EB) THEN
! H_F overrides REFERENCE_ENTHALPY
IF (ENTHALPY_OF_FORMATION > -1.E23_EB) THEN
SS%REFERENCE_ENTHALPY = SS%H_F/SS%MW*1000._EB - SS%SPECIFIC_HEAT*H_F_REFERENCE_TEMPERATURE
ELSE
IF (SS%REFERENCE_ENTHALPY < -1.E20_EB) SS%REFERENCE_ENTHALPY = SS%SPECIFIC_HEAT * SS%REFERENCE_TEMPERATURE
SS%H_F = (SS%REFERENCE_ENTHALPY + SS%SPECIFIC_HEAT * (H_F_REFERENCE_TEMPERATURE-SS%REFERENCE_TEMPERATURE))*SS%MW
      *0.001_EB
!Adjust SS%REFERENCE_ENTHALPY to 0 K
SS%REFERENCE_ENTHALPY = SS%REFERENCE_ENTHALPY - SS%SPECIFIC_HEAT * SS%REFERENCE_TEMPERATURE
ENDIF
ENDIF

IF (SS%RAMP_CP/='null' .AND. SS%REFERENCE_ENTHALPY < -1.E20_EB) SS%REFERENCE_ENTHALPY = 0._EB

IF (TRIM(SS%FORMULA)=='null') WRITE(SS%FORMULA,'(A,I0)') 'SPEC_',N

! For simple chemistry Determine if the species is the one specified on the REAC line(s)

IF (SIMPLE_CHEMISTRY) THEN
IF (TRIM(ID)==TRIM(REACTION(1)%FUEL)) THEN
FUEL_INDEX = N
WRITE(FORMULA,'(A,I0)') 'SPEC_',N
IF (TRIM(SS%FORMULA)==TRIM(FORMULA)) SS%MW = REACTION(1)%MW_FUEL
ENDIF
IF (TRIM(ID)=='SOOT') SS%MW = REACTION(1)%MW_SOOT
ENDIF

SS%RCON = R0/SS%MW
SS%MODE = GAS_SPECIES

! Special processing of certain species

SELECT CASE (ID)
CASE('WATER VAPOR')
H2O_INDEX = N
```

```
IF (MASS_FRACTION_0 > 0._EB .AND. LUMPED_COMPONENT_ONLY) THEN
WRITE(MESSAGE,'(A)') 'WARNING: MASS_FRACTION_0 specified for WATER VAPOR with LUMPED_COMPONENT_ONLY = .TRUE.'
IF (MYID==0) WRITE(LU_ERR,'(A)') TRIM(MESSAGE)
ENDIF
IF (PREDEFINED_SMIX(1)) Y_H2O_INFTY = WATER_VAPOR_MASS_FRACTION(HUMIDITY,TMPA,P_INF)
CASE('CARBON DIOXIDE')
CO2_INDEX = N
CASE('CARBON MONOXIDE')
CO_INDEX = N
CASE('OXYGEN')
O2_INDEX = N
CASE('NITROGEN')
N2_INDEX = N
CASE('HYDROGEN')
H2_INDEX = N
CASE('HYDROGEN CYANIDE')
HCN_INDEX = N
CASE('NITRIC OXIDE')
NO_INDEX = N
CASE('NITROGEN DIOXIDE')
NO2_INDEX = N
CASE('SOOT')
SOOT_INDEX = N
IF (MASS_EXTINCTION_COEFFICIENT < 0._EB) SS%MASS_EXTINCTION_COEFFICIENT = 8700._EB
END SELECT

IF (SS%RADCAL_ID=='SOOT' .AND. SOOT_INDEX==0) SOOT_INDEX = N
IF (SS%ID=='SOOT' .AND. AEROSOL_AL2O3) SS%DENSITY_SOLID = 4000.
IF (AEROSOL) SS%MODE = AEROSOL_SPECIES

! Get ramps
IF (SS%RAMP_CP/='null') THEN
CALL GET_RAMP_INDEX(SS%RAMP_CP,'TEMPERATURE',NR)
SS%RAMP_CP_INDEX = NR
ENDIF
IF (SS%RAMP_CP_L/='null') THEN
CALL GET_RAMP_INDEX(SS%RAMP_CP_L,'TEMPERATURE',NR)
SS%RAMP_CP_L_INDEX = NR
ENDIF
IF (SS%RAMP_D/='null') THEN
CALL GET_RAMP_INDEX(SS%RAMP_D,'TEMPERATURE',NR)
SS%RAMP_D_INDEX = NR
ENDIF
IF (SS%RAMP_G_F/='null') THEN
CALL GET_RAMP_INDEX(SS%RAMP_G_F,'TEMPERATURE',NR)
SS%RAMP_G_F_INDEX = NR
ENDIF
IF (SS%RAMP_K/='null') THEN
CALL GET_RAMP_INDEX(SS%RAMP_K,'TEMPERATURE',NR)
SS%RAMP_K_INDEX = NR
ENDIF
IF (SS%RAMP_MU/='null') THEN
CALL GET_RAMP_INDEX(SS%RAMP_MU,'TEMPERATURE',NR)
SS%RAMP_MU_INDEX = NR
ENDIF
ENDDO PRIMITIVE_SPEC_READ_LOOP

! Unmixed fraction
IF (TRANSPORT_UNMIXED_FRACTION) THEN
N_PASSIVE_SCALARS=1
ZETA_INDEX=N_TRACKED_SPECIES+1
ENDIF

! IMPORTANT: define number of total tracked scalars
N_TOTAL_SCALARS=N_TRACKED_SPECIES+N_PASSIVE_SCALARS

! Pass 3: process tracked species (primitive and lumped)
ALLOCATE(SPECIES_MIXTURE(1:N_TOTAL_SCALARS),STAT=IZERO)
CALL ChkMemErr('READ','SPECIES_MIXTURE',IZERO)

! Process non-predefined mixtures first
REWIND(LU_INPUT) ; INPUT_FILE_LINE_NUMBER = 0
N = 1
DEFINED_BACKGROUND = .FALSE.
N_SPEC_READ = 0
N_COPY = 0
N_FOUND = 1

TRACKED_SPEC_LOOP_1: DO WHILE (N_FOUND <= N_TRACKED_SPECIES .OR. .NOT. DEFINED_BACKGROUND)
IF (PREDEFINED_SMIX(N)) THEN
CALL SET_SPEC_DEFAULT
IF (N==1) BACKGROUND = .TRUE.
ELSE
FIND_TRACKED: DO
CALL SET_SPEC_DEFAULT
READ(LU_INPUT,NML=SPEC)
N_SPEC_READ = N_SPEC_READ + 1
IF (LUMPED_COMPONENT_ONLY .AND. N_BINS < 0) CYCLE FIND_TRACKED
```

```
IF (COPY_LUMPED) THEN
IF (N_BINS >0) THEN
WRITE(MESSAGE,'(A,A,A)') 'ERROR: SPEC ' ,TRIM(SM%ID),', cannot specify both COPY_LUMPED and N_BINS.'
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF
IF (BACKGROUND) THEN
WRITE(MESSAGE,'(A,A,A)') 'ERROR: SPEC ' ,TRIM(SM%ID),', cannot specify both COPY_LUMPED and BACKGROUND.'
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF
N_COPY = N_COPY+1
EXIT FIND_TRACKED
ENDIF
IF (ANY(MASS_FRACTION>0._EB) .AND. ANY(VOLUME_FRACTION>0._EB)) THEN
WRITE(MESSAGE,'(A,A,A)') 'ERROR: SPEC ' ,TRIM(SM%ID),', cannot specify both MASS_FRACTION and VOLUME_FRACTION.'
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF
EXIT
ENDDO FIND_TRACKED
IF (COPY_LUMPED) THEN
N_FOUND = N + N_COPY
CYCLE TRACKED_SPEC_LOOP_1
ENDIF
IF (N_BINS > 0) THEN
DO NNN=1,N_PARTICLE_BINS
MEAN_DIAMETER = 2._EB*PARTICLE_RADIUS(NNN)
SPEC_ID(1)=SPECIES(AGGLOMERATION_INDEX)%ID
WRITE(ID,'(A,A,I0)') TRIM(SPECIES(AGGLOMERATION_INDEX)%ID),'_',NNN
MASS_FRACTION(1)=1._EB
CALL DEFINE_MIXTURE
N = N + 1
ENDDO
AGGLOMERATION_INDEX = N - N_PARTICLE_BINS
N = N - 1
ELSE
CALL DEFINE_MIXTURE
ENDIF

IF (SIMPLE_CHEMISTRY) THEN
SM => SPECIES_MIXTURE(N)
IF (TRIM(SM%ID)==TRIM(REACTION(1)%FUEL)) THEN
FUEL_SMIX_INDEX = N
IF (ABS(SM%ATOMS(1)+SM%ATOMS(6)+SM%ATOMS(7)+SM%ATOMS(8) - SUM(SM%ATOMS)) > SPACING(SUM(SM%ATOMS))) THEN
WRITE(MESSAGE,'(A)') 'ERROR: Fuel FORMULA for SIMPLE_CHEMISTRY can only contain C,H,O, and N'
CALL SHUTDOWN(MESSAGE) ; RETURN
ELSE
REACTION(1)%C = SM%ATOMS(6)
REACTION(1)%H = SM%ATOMS(1)
REACTION(1)%O = SM%ATOMS(8)
REACTION(1)%N = SM%ATOMS(7)
REACTION(1)%MW_FUEL = SM%MW
ENDIF
IF (REACTION(1)%C<=TWO_EPSILON_EB .AND. REACTION(1)%H<=TWO_EPSILON_EB) THEN
WRITE(MESSAGE,'(A)') 'ERROR: Must specify fuel chemistry using C and/or H when using simple chemistry'
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF
ENDIF
ENDIF
ENDIF
IF (BACKGROUND) THEN
DEFINED_BACKGROUND = .TRUE.
IF (N==1) N = N + 1
ELSE
N = N + 1
ENDIF
N_FOUND = N + N_COPY
ENDDO TRACKED_SPEC_LOOP_1

IF (N_COPY >= 1) NNN = N

! Process predefined mixtures second

IF (ANY(PREDEFINED_SMIX)) THEN
REWIND(LU_INPUT) ; INPUT_FILE_LINE_NUMBER = 0
N = 1
DEFINED_BACKGROUND = .FALSE.
TRACKED_SPEC_LOOP_2: DO WHILE (N <= N_TOTAL_SCALARS .OR. .NOT. DEFINED_BACKGROUND)
IF (PREDEFINED_SMIX(N)) THEN
CALL SET_SPEC_DEFAULT
CALL SETUP_PREDEFINED_SMIX(N)
IF (N==1) BACKGROUND=.TRUE.
CALL DEFINE_MIXTURE
ELSE
BACKGROUND = .FALSE.
ENDIF
IF (BACKGROUND) THEN
DEFINED_BACKGROUND = .TRUE.
IF (N==1) N = N + 1
ELSE
```

```
N = N + 1
ENDIF
ENDDO TRACKED_SPEC_LOOP_2
ENDIF

REWIND (LU_INPUT)

TRACKED_SPEC_LOOP_3: DO NN = 1, N_COPY
FIND_TRACKED_2: DO
CALL SET_SPEC_DEFAULT
READ(LU_INPUT,NML=SPEC)
IF (.NOT. COPY_LUMPED) CYCLE FIND_TRACKED_2
DO N=1,NNN-1
IF (SPECIES_MIXTURE(N)%ID==SPEC_ID(1)) THEN
SPECIES_MIXTURE(NNN) = SPECIES_MIXTURE(N)
SPECIES_MIXTURE(NNN)%ID = ID
EXIT FIND_TRACKED_2
ELSE
IF (N==NNN-1) THEN
WRITE(MESSAGE,'(A,A,A,A)') 'ERROR: SPEC ' ,TRIM(ID),', cannot find tracked species ',TRIM(SPEC_ID(1))
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF
ENDIF
ENDDO
ENDDO FIND_TRACKED_2
NNN = NNN + 1
ENDDO TRACKED_SPEC_LOOP_3

REWIND (LU_INPUT) ; INPUT_FILE_LINE_NUMBER = 0

! Normalize the initial mass fractions of the lumped species if necessary
IF (SUM(SPECIES_MIXTURE(2:N_TRACKED_SPECIES)%ZZ0) > 1._EB) &
SPECIES_MIXTURE(2:N_TRACKED_SPECIES)%ZZ0 = SPECIES_MIXTURE(2:N_TRACKED_SPECIES)%ZZ0/ &
SUM(SPECIES_MIXTURE(2:N_TRACKED_SPECIES)%ZZ0)

SPECIES_MIXTURE(1)%ZZ0 = 1._EB - SUM(SPECIES_MIXTURE(2:N_TRACKED_SPECIES)%ZZ0)

DEPOSITION = ANY(SPECIES_MIXTURE%DEPOSITING) .AND. DEPOSITION

!Deallocate species inputs
DEALLOCATE(PREDEFINED)
DEALLOCATE(PREDEFINED_SMIX)
DEALLOCATE(NEW_PRIMITIVE)
DEALLOCATE(PREDEFINED_SPEC_ID)
DEALLOCATE(SPEC_ID_READ)
DEALLOCATE(Y_INDEX)

CONTAINS

SUBROUTINE DEFINE_MIXTURE
USE PROPERTY_DATA, ONLY:GET_FORMULA_WEIGHT
! Create a species mixture

IF (BACKGROUND) THEN
NN = 1
ELSE
IF (N==1) N = N + 1
NN = N
ENDIF

CONVERSION = 0._EB

SM => SPECIES_MIXTURE(NN)

IF (SPEC_ID(1)=='null') THEN
SPEC_ID(1) = ID
VOLUME_FRACTION(1) = 1.0_EB
ELSE
SM%K_USER                        = CONDUCTIVITY
SM%D_USER                        = DIFFUSIVITY
SM%EPSK                          = EPSILONKLJ
SM%FIC_CONCENTRATION             = FIC_CONCENTRATION
SM%FLD_LETHAL_DOSE               = FLD_LETHAL_DOSE
SM%MU_USER                       = VISCOSITY
SM%PR_USER                       = PR_GAS
SM%RAMP_CP                       = RAMP_CP
SM%RAMP_D                        = RAMP_D
SM%RAMP_G_F                      = RAMP_G_F
SM%RAMP_K                        = RAMP_K
SM%RAMP_MU                       = RAMP_MU
IF (REFERENCE_TEMPERATURE < -TMPM) REFERENCE_TEMPERATURE = 25._EB
SM%REFERENCE_TEMPERATURE         = REFERENCE_TEMPERATURE + TMPM
SM%SIG                           = SIGMALJ
SM%SPECIFIC_HEAT                 = SPECIFIC_HEAT*1000._EB
SM%REFERENCE_ENTHALPY            = REFERENCE_ENTHALPY*1000._EB
SM%H_F                           = ENTHALPY_OF_FORMATION*1000._EB

IF (SM%RAMP_CP/='null' .AND. SM%REFERENCE_ENTHALPY < -1.E20_EB) SM%REFERENCE_ENTHALPY = 0._EB
```

```

! Get ramps
IF (SM%RAMP_CP/='null') THEN
CALL GET_RAMP_INDEX(SM%RAMP_CP,'TEMPERATURE',NR)
SM%RAMP_CP_INDEX = NR
ENDIF
IF (SM%RAMP_D/='null') THEN
CALL GET_RAMP_INDEX(SM%RAMP_D,'TEMPERATURE',NR)
SM%RAMP_D_INDEX = NR
ENDIF
IF (SM%RAMP_G_F/='null') THEN
CALL GET_RAMP_INDEX(SM%RAMP_D,'TEMPERATURE',NR)
SM%RAMP_G_F_INDEX = NR
ENDIF
IF (SM%RAMP_K/='null') THEN
CALL GET_RAMP_INDEX(SM%RAMP_K,'TEMPERATURE',NR)
SM%RAMP_K_INDEX = NR
ENDIF
IF (SM%RAMP_MU/='null') THEN
CALL GET_RAMP_INDEX(SM%RAMP_MU,'TEMPERATURE',NR)
SM%RAMP_MU_INDEX = NR
ENDIF
ENDIF

SM%ID = ID
SM%ZZ0 = MAX(0._EB,MASS_FRACTION_0)

! Count the number of species included in the mixture

N_SUB_SPECIES = 0
COUNT_SPEC: DO NS=1,N_SPECIES
IF (TRIM(SPEC_ID(NS)) /= 'null') THEN
N_SUB_SPECIES = N_SUB_SPECIES + 1
ELSE
EXIT
ENDIF
ENDDO COUNT_SPEC

IF (N_SUB_SPECIES == 1) THEN
MASS_FRACTION=0._EB
MASS_FRACTION(1)=1._EB
VOLUME_FRACTION=0._EB
ENDIF

! Allocate arrays to store the species id, mass, volume fractions

ALLOCATE (SM%SPEC_ID(N_SPECIES),STAT=IZERO)
ALLOCATE (SM%VOLUME_FRACTION(N_SPECIES),STAT=IZERO)
ALLOCATE (SM%MASS_FRACTION(N_SPECIES),STAT=IZERO)

SM%SPEC_ID          = 'null'
SM%VOLUME_FRACTION  = 0._EB
SM%MASS_FRACTION    = 0._EB
Y_INDEX = -1
DO NS = 1,N_SUB_SPECIES
FIND_SPEC_ID: DO NS2 = 1,N_SPECIES
IF ((.NOT. NEW_PRIMITIVE(N_SPEC_READ) .AND. TRIM(SPECIES(NS2)%ID) == TRIM(SPEC_ID(NS))) .OR. &
(        NEW_PRIMITIVE(N_SPEC_READ) .AND. TRIM(SPECIES(NS2)%ID) == TRIM(ID))) THEN
SM%SPEC_ID(NS2) = SPECIES(NS2)%ID
Y_INDEX(NS)    = NS2
IF (N_SUB_SPECIES==1) THEN
SM%FORMULA = SPECIES(NS2)%FORMULA
SM%SINGLE_SPEC_INDEX=NS2
ENDIF
IF (SPECIES(NS2)%MODE == AEROSOL_SPECIES) THEN
IF (N_SUB_SPECIES == 1) THEN
SM%DEPOSITING = .TRUE.
IF (ABS(MEAN_DIAMETER-1.E-6_EB)<=TWO_EPSILON_EB) THEN
SM%MEAN_DIAMETER = SPECIES(NS2)%MEAN_DIAMETER
ELSE
SM%MEAN_DIAMETER = MEAN_DIAMETER
ENDIF
IF (ABS(DENSITY_SOLID-1800._EB) <=TWO_EPSILON_EB .AND. &
ABS(DENSITY_SOLID-SPECIES(NS2)%DENSITY_SOLID) <=TWO_EPSILON_EB) THEN
SM%DENSITY_SOLID = DENSITY_SOLID
ELSE
SM%DENSITY_SOLID = SPECIES(NS2)%DENSITY_SOLID
ENDIF
SM%CONDUCTIVITY_SOLID=SPECIES(NS2)%CONDUCTIVITY_SOLID
ELSE
WRITE(MESSAGE,'(A,A,A)') 'WARNING: Cannot do deposition with a lumped species.  Species ',TRIM(SM%ID),&
' will not have deposition'
IF (MYID==0) WRITE(LU_ERR,'(A)') TRIM(MESSAGE)
ENDIF
ENDIF
EXIT FIND_SPEC_ID
ENDIF
ENDDO FIND_SPEC_ID
```

```
IF (Y_INDEX(NS)<0) THEN
WRITE(MESSAGE,'(A,A,A,I0,A)') 'ERROR: SPEC ',TRIM(SM%ID),', sub species ',NS,' not found.'
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF
IF (MASS_FRACTION(NS)>0._EB)      CONVERSION = CONVERSION + MASS_FRACTION(NS)   / SPECIES(Y_INDEX(NS))%MW
IF (VOLUME_FRACTION(NS)>0._EB)    CONVERSION = CONVERSION + VOLUME_FRACTION(NS) * SPECIES(Y_INDEX(NS))%MW
IF (.NOT. PREDEFINED_SMIX(NN) .AND. MASS_FRACTION(NS)<=0._EB .AND. VOLUME_FRACTION(NS)<=0._EB) THEN
WRITE(MESSAGE,'(A,A,A,I0,A)') 'ERROR: SPEC ',TRIM(SM%ID),', mass or volume fraction for sub species ',NS,' not
        found.'
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF

ENDDO

IF (ANY(MASS_FRACTION>0._EB)) THEN
DO NS = 1,N_SUB_SPECIES
SM%VOLUME_FRACTION(Y_INDEX(NS)) = MASS_FRACTION(NS) / SPECIES(Y_INDEX(NS))%MW / CONVERSION
SM%MASS_FRACTION(Y_INDEX(NS))   = MASS_FRACTION(NS)
ENDDO
ENDIF

IF (ANY(VOLUME_FRACTION>0._EB)) THEN
DO NS = 1,N_SUB_SPECIES
SM%MASS_FRACTION(Y_INDEX(NS))   = VOLUME_FRACTION(NS) * SPECIES(Y_INDEX(NS))%MW / CONVERSION
SM%VOLUME_FRACTION(Y_INDEX(NS)) = VOLUME_FRACTION(NS)
ENDDO
ENDIF

! Normalize mass and volume fractions, plus stoichiometric coefficient

SM%MASS_FRACTION = SM%MASS_FRACTION / SUM(SM%MASS_FRACTION)
IF (.NOT. SIMPLE_CHEMISTRY) SM%ADJUST_NU = SUM(SM%VOLUME_FRACTION)
SM%VOLUME_FRACTION = SM%VOLUME_FRACTION / SUM(SM%VOLUME_FRACTION)

! Calculate the molecular weight and extinction coefficient

SM%MW = 0._EB
SM%MASS_EXTINCTION_COEFFICIENT = 0._EB
DO NS = 1,N_SPECIES
IF (SM%MASS_FRACTION(NS) <TWO_EPSILON_EB) CYCLE
IF (MASS_EXTINCTION_COEFFICIENT > 0._EB) THEN
SM%MASS_EXTINCTION_COEFFICIENT = MASS_EXTINCTION_COEFFICIENT
ELSE
SM%MASS_EXTINCTION_COEFFICIENT = SM%MASS_EXTINCTION_COEFFICIENT+SM%MASS_FRACTION(NS)*SPECIES(NS)%
        MASS_EXTINCTION_COEFFICIENT
ENDIF
IF (MW > 0._EB) THEN
SM%MW = MW
ELSE
SM%MW = SM%MW + SM%VOLUME_FRACTION(NS) * SPECIES(NS)%MW !! *SM%ADJUST_NU ::term for potential non-normalized
        inputs
ENDIF
IF (SPECIES(NS)%FORMULA(1:5)=='SPEC_') SM%VALID_ATOMS = .FALSE.
IF (FORMULA /= 'null' .AND. .NOT. PREDEFINED_SMIX(NN)) THEN
CALL GET_FORMULA_WEIGHT(FORMULA,SM%MW,SM%ATOMS)
ELSE
SM%ATOMS = SM%ATOMS + SM%VOLUME_FRACTION(NS)*SPECIES(NS)%ATOMS !! *SM%ADJUST_NU ::term for potential non-
        normalized inputs
ENDIF
ENDDO

SM%RCON = R0/SM%MW

IF (SM%H_F > -1.E23_EB) SM%H_F = SM%H_F/SM%MW*1000._EB !J/mol -> J/kg
IF (SM%SPECIFIC_HEAT > 0._EB) THEN
IF (SM%H_F > -1.E23_EB) THEN
SM%REFERENCE_ENTHALPY = SM%H_F - SM%SPECIFIC_HEAT*H_F_REFERENCE_TEMPERATURE
ELSE
IF (SM%REFERENCE_ENTHALPY < -1.E20_EB) SM%REFERENCE_ENTHALPY = SM%SPECIFIC_HEAT * SM%REFERENCE_TEMPERATURE
SM%H_F = SM%REFERENCE_ENTHALPY + SM%SPECIFIC_HEAT * (H_F_REFERENCE_TEMPERATURE - SM%REFERENCE_TEMPERATURE)
ENDIF
ENDIF

END SUBROUTINE DEFINE_MIXTURE

SUBROUTINE SET_SPEC_DEFAULT

AEROSOL                    = .FALSE.
BACKGROUND                 = .FALSE.
BETA_LIQUID                = -1._EB
CONDUCTIVITY               = -1._EB
CONDUCTIVITY_LIQUID        = -1._EB
CONDUCTIVITY_SOLID         = 0.26_EB !W/m/K Ben-Dor, et al. 2002. (~10 x air)
COPY_LUMPED                = .FALSE.
DENSITY_SOLID              = 1800._EB !kg/m^3 Slowik, et al. 2004
DIFFUSIVITY                = -1._EB
EPSILONKLJ                 =  0._EB
FIC_CONCENTRATION          =  0._EB
```

```
FLD_LETHAL_DOSE               =  0._EB
FORMULA                       = 'null'
FYI                           = 'null'
ENTHALPY_OF_FORMATION         = -1.E30_EB   ! J/mol
ID                            = 'null'
LUMPED_COMPONENT_ONLY         = .FALSE.
RADCAL_ID                     = 'null'
MASS_EXTINCTION_COEFFICIENT   = -1._EB   ! m2/kg
MASS_FRACTION                 =  0._EB
MASS_FRACTION_0               = -1._EB
MEAN_DIAMETER                 =  1.E-6_EB
MW                            =  0._EB
PR_GAS                        = -1._EB
PRIMITIVE                     = .FALSE.
REFERENCE_TEMPERATURE         = -300._EB  ! C
SIGMALJ                       =  0._EB
SPEC_ID                       = 'null'
SPECIFIC_HEAT                 = -1._EB
REFERENCE_ENTHALPY            = -2.E20_EB
VISCOSITY                     = -1._EB
VISCOSITY_LIQUID              =  -1._EB
VOLUME_FRACTION               =  0._EB

DENSITY_LIQUID                = -1._EB
HEAT_OF_VAPORIZATION          = -1._EB      ! kJ/kg
H_V_REFERENCE_TEMPERATURE     = -300._EB
MELTING_TEMPERATURE           = -300._EB    ! C
SPECIFIC_HEAT_LIQUID          = -1._EB      ! kJ/kg-K
VAPORIZATION_TEMPERATURE      = -300._EB    ! C

RAMP_CP                       = 'null'
RAMP_CP_L                     = 'null'
RAMP_D                        = 'null'
RAMP_G_F                      = 'null'
RAMP_K                        = 'null'
RAMP_MU                       = 'null'

N_BINS                        = -1
MIN_DIAMETER                  = -1._EB  !um
MAX_DIAMETER                  = -1._EB  !um

END SUBROUTINE SET_SPEC_DEFAULT

SUBROUTINE SETUP_PREDEFINED_SMIX(N)

! Set up the SMIX line either for the SIMPLE_CHEMISTRY mode or for a primitive species

INTEGER, INTENT(IN) :: N
TYPE(REACTION_TYPE), POINTER :: RN

MASS_FRACTION = 0._EB

SELECT CASE(N)
CASE(1)
ID                 = 'AIR'
FORMULA            = 'Z0'
SPEC_ID(1)         = 'WATER VAPOR'
SPEC_ID(2)         = 'OXYGEN'
SPEC_ID(3)         = 'CARBON DIOXIDE'
SPEC_ID(4)         = 'NITROGEN'
MASS_FRACTION(1)  = Y_H2O_INFTY
MASS_FRACTION(2)  = Y_O2_INFTY*(1._EB-Y_H2O_INFTY)
MASS_FRACTION(3)  = Y_CO2_INFTY*(1._EB-Y_H2O_INFTY)
MASS_FRACTION(4)  = 1._EB-SUM(MASS_FRACTION)
CASE(2)
RN => REACTION(1)
ID                 = RN%FUEL
FORMULA            = 'Z1'
SPEC_ID(1)         = RN%FUEL
MASS_FRACTION(1)  = 1._EB
CASE(3)
RN => REACTION(1)
ID                    = 'PRODUCTS'
FORMULA               = 'Z2'
SPEC_ID(1)            = 'CARBON MONOXIDE'
SPEC_ID(2)            = 'SOOT'
SPEC_ID(3)            = 'WATER VAPOR'
SPEC_ID(4)            = 'CARBON DIOXIDE'
SPEC_ID(5)            = 'NITROGEN'
RN%NU_CO              = (SPECIES_MIXTURE(FUEL_SMIX_INDEX)%MW/MW_CO)        *RN%CO_YIELD
RN%NU_SOOT            = (SPECIES_MIXTURE(FUEL_SMIX_INDEX)%MW/RN%MW_SOOT)*RN%SOOT_YIELD
RN%NU_H2O             = 0.5_EB*RN%H - 0.5_EB*RN%NU_SOOT*RN%SOOT_H_FRACTION
RN%NU_CO2             = RN%C - RN%NU_CO - RN%NU_SOOT*(1._EB-RN%SOOT_H_FRACTION)
IF (RN%NU_CO2 <0._EB) THEN
WRITE(MESSAGE,'(A)') 'ERROR: REAC, Not enough carbon in the fuel for the specified CO_YIELD and/or SOOT_YIELD'
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF
RN%NU_O2              = RN%NU_CO2 + 0.5_EB*(RN%NU_CO+RN%NU_H2O-RN%O)
```

```
RN%NU_N2          = RN%N*0.5_EB
VOLUME_FRACTION(1)  = RN%NU_CO
VOLUME_FRACTION(2)  = RN%NU_SOOT
VOLUME_FRACTION(3)  = RN%NU_H2O + SPECIES_MIXTURE(1)%VOLUME_FRACTION(H2O_INDEX)*RN%NU_O2 / &
SPECIES_MIXTURE(1)%VOLUME_FRACTION(O2_INDEX)
VOLUME_FRACTION(4)  = RN%NU_CO2 + SPECIES_MIXTURE(1)%VOLUME_FRACTION(CO2_INDEX)*RN%NU_O2 / &
SPECIES_MIXTURE(1)%VOLUME_FRACTION(O2_INDEX)
VOLUME_FRACTION(5)  = RN%NU_N2  + SPECIES_MIXTURE(1)%VOLUME_FRACTION(N2_INDEX)*RN%NU_O2 / &
SPECIES_MIXTURE(1)%VOLUME_FRACTION(O2_INDEX)
VOLUME_FRACTION     = VOLUME_FRACTION/SUM(VOLUME_FRACTION)
SPECIES(SOOT_INDEX)%ATOMS=0._EB
SPECIES(SOOT_INDEX)%ATOMS(1)=RN%SOOT_H_FRACTION
SPECIES(SOOT_INDEX)%ATOMS(6)=1._EB-RN%SOOT_H_FRACTION
END SELECT

END SUBROUTINE SETUP_PREDEFINED_SMIX

END SUBROUTINE READ_SPEC


SUBROUTINE PROC_SMIX

! Create the Z to Y transformation matrix and fill up the gas property tables

USE PHYSICAL_FUNCTIONS, ONLY: GET_SPECIFIC_GAS_CONSTANT
USE MATH_FUNCTIONS, ONLY: EVALUATE_RAMP
USE PROPERTY_DATA, ONLY: JANAF_TABLE, CALC_GAS_PROPS, GAS_PROPS, CALC_MIX_PROPS
REAL(EB), ALLOCATABLE, DIMENSION(:) :: MU_TMP,CP_TMP,K_TMP,H_TMP,D_TMP,G_F_TMP,ZZ_GET, &
MU_TMP_Z,CP_TMP_Z,K_TMP_Z,H_TMP_Z,D_TMP_Z,G_F_TMP_Z,RSQ_MW_Y
REAL(EB) :: CP1,CP2,H1,H2,H_REF_SENSIBLE(1:N_TRACKED_SPECIES),REF_TEMP
INTEGER :: NN,N
TYPE(SPECIES_TYPE), POINTER :: SS=>NULL()
TYPE(SPECIES_MIXTURE_TYPE), POINTER :: SM=>NULL()

! Setup the array to convert the tracked species array to array of all primitive species

ALLOCATE(Z2Y(N_SPECIES,N_TRACKED_SPECIES),STAT=IZERO)
CALL ChkMemErr('READ','Z2Y',IZERO)
Z2Y = 0._EB

DO N=1,N_TRACKED_SPECIES
SM => SPECIES_MIXTURE(N)
DO NN=1,N_SPECIES
Z2Y(NN,N) = SM%MASS_FRACTION(NN)
ENDDO
ENDDO

ALLOCATE(RSQ_MW_Y(N_SPECIES),STAT=IZERO)
CALL ChkMemErr('READ','RSQ_MW_Y',IZERO)

RSQ_MW_Y=1._EB/SQRT(SPECIES%MW)

! Set up the arrays of molecular weights

ALLOCATE(MWR_Z(N_TRACKED_SPECIES),STAT=IZERO)
CALL ChkMemErr('READ','MW_AVG_Y',IZERO)

ALLOCATE(RSQ_MW_Z(N_TRACKED_SPECIES),STAT=IZERO)
CALL ChkMemErr('READ','RSQ_MW_Z',IZERO)

MWR_Z = 1._EB/SPECIES_MIXTURE%MW
RSQ_MW_Z = 1._EB/SQRT(SPECIES_MIXTURE%MW)

ALLOCATE(ZZ_GET(N_TRACKED_SPECIES))
ZZ_GET = SPECIES_MIXTURE%ZZ0
CALL GET_SPECIFIC_GAS_CONSTANT(ZZ_GET,RSUM0)
DEALLOCATE(ZZ_GET)

MW_MIN = MINVAL(SPECIES_MIXTURE(1:N_TRACKED_SPECIES)%MW)
MW_MAX = MAXVAL(SPECIES_MIXTURE(1:N_TRACKED_SPECIES)%MW)

! Compute background density from other background quantities

RHOA = P_INF/(TMPA*RSUM0)

! Compute constant-temperature specific heats

GM1OG = (GAMMA-1._EB)/GAMMA
CP_GAMMA = SPECIES_MIXTURE(1)%RCON/GM1OG
CPOPR = CP_GAMMA/PR

! Compute gas properties for primitive species 1 to N_SPECIES.

ALLOCATE(D_TMP(N_SPECIES))
D_TMP = 0._EB
ALLOCATE(MU_TMP(N_SPECIES))
MU_TMP = 0._EB
ALLOCATE(CP_TMP(N_SPECIES))
```

```
CP_TMP = 0._EB
ALLOCATE(H_TMP(N_SPECIES))
H_TMP = 0._EB
ALLOCATE(K_TMP(N_SPECIES))
K_TMP = 0._EB
ALLOCATE(G_F_TMP(N_SPECIES))
G_F_TMP = 0._EB

ALLOCATE(D_TMP_Z(N_TOTAL_SCALARS))
D_TMP_Z = -1.E30_EB
ALLOCATE(MU_TMP_Z(N_TOTAL_SCALARS))
MU_TMP_Z = -1.E30_EB
ALLOCATE(CP_TMP_Z(N_TOTAL_SCALARS))
CP_TMP_Z = -1.E30_EB
ALLOCATE(H_TMP_Z(N_TOTAL_SCALARS))
H_TMP_Z = -1.E30_EB
ALLOCATE(K_TMP_Z(N_TOTAL_SCALARS))
K_TMP_Z = -1.E30_EB
ALLOCATE(G_F_TMP_Z(N_TOTAL_SCALARS))
G_F_TMP_Z = -1.E30_EB

ALLOCATE(CPBAR_Z(0:5000,N_TOTAL_SCALARS))
CALL ChkMemErr('READ','CPBAR_Z',IZERO)
CPBAR_Z = 0._EB

!ALLOCATE(CP_AVG_Z(0:5000,N_TOTAL_SCALARS))
!CALL ChkMemErr('READ','CP_AVG_Z',IZERO)
!CP_AVG_Z = 0._EB

ALLOCATE(H_SENS_Z(0:5000,N_TOTAL_SCALARS))
CALL ChkMemErr('READ','H_SENS_Z',IZERO)
H_SENS_Z = 0._EB

ALLOCATE(K_RSQMW_Z(0:5000,N_TOTAL_SCALARS))
CALL ChkMemErr('READ','K_RSQMW_Z',IZERO)
K_RSQMW_Z = 0._EB

ALLOCATE(MU_RSQMW_Z(0:5000,N_TOTAL_SCALARS))
CALL ChkMemErr('READ','MU_RSQMW_Z',IZERO)
MU_RSQMW_Z = 0._EB

ALLOCATE(CP_Z(0:5000,N_TOTAL_SCALARS))
CALL ChkMemErr('READ','CP_Z',IZERO)
CP_Z = 0._EB

ALLOCATE(D_Z(0:5000,N_TOTAL_SCALARS))
CALL ChkMemErr('READ','D_Z',IZERO)
D_Z = 0._EB

ALLOCATE(G_F_Z(0:5000,N_TOTAL_SCALARS))
CALL ChkMemErr('READ','G_F_Z',IZERO)
G_F_Z = 0._EB

! Adjust reference enthalpy to 0 K if a RAMP_CP is given
DO N=1,N_SPECIES
SS => SPECIES(N)
IF(SS%RAMP_CP_INDEX > 0) THEN
IF (SS%H_F > -1.E20_EB) THEN
CP2 = EVALUATE_RAMP(1._EB,1._EB,SS%RAMP_CP_INDEX)*1000._EB
H2 = 0._EB
DO J=1,INT(H_F_REFERENCE_TEMPERATURE)+1
H1 = H2
CP1 = CP2
CP2 = EVALUATE_RAMP(REAL(J,EB),1._EB,SS%RAMP_CP_INDEX)*1000._EB
H2 = H2 + 0.5_EB*(CP1+CP2)
ENDDO
SS%REFERENCE_ENTHALPY = SS%H_F/SS%MW*1000._EB - &
(H1 + (H2-H1)*(H_F_REFERENCE_TEMPERATURE-INT(H_F_REFERENCE_TEMPERATURE)))
ELSE
IF (SS%REFERENCE_TEMPERATURE<=TWO_EPSILON_EB) CYCLE
CP2 = EVALUATE_RAMP(1._EB,1._EB,SS%RAMP_CP_INDEX)*1000._EB
H2 = 0._EB
DO J=1,INT(SS%REFERENCE_TEMPERATURE)+1
H1 = H2
CP1 = CP2
CP2 = EVALUATE_RAMP(REAL(J,EB),1._EB,SS%RAMP_CP_INDEX)*1000._EB
H2 = H2 + 0.5_EB*(CP1+CP2)
ENDDO
SS%REFERENCE_ENTHALPY = SS%REFERENCE_ENTHALPY - &
(H1 + (H2-H1)*(SS%REFERENCE_TEMPERATURE-INT(SS%REFERENCE_TEMPERATURE)))
IF (SS%H_F <= -1.E20) THEN
CP2 = EVALUATE_RAMP(1._EB,1._EB,SS%RAMP_CP_INDEX)*1000._EB
H2 = SS%REFERENCE_ENTHALPY
DO J=1,INT(H_F_REFERENCE_TEMPERATURE)+1
H1 = H2
CP1 = CP2
CP2 = EVALUATE_RAMP(REAL(J,EB),1._EB,SS%RAMP_CP_INDEX)*1000._EB
H2 = H2 + 0.5_EB*(CP1+CP2)
```

```
ENDDO
SS%H_F = (H1 + (H2-H1)*(H_F_REFERENCE_TEMPERATURE-INT(H_F_REFERENCE_TEMPERATURE)))*SS%MW*0.001_EB
ENDIF
ENDIF
ENDIF
END DO

DO N=1,N_TRACKED_SPECIES
SM => SPECIES_MIXTURE(N)
IF(SM%RAMP_CP_INDEX > 0) THEN
IF (SM%H_F > -1.E20_EB) THEN
CP2 = EVALUATE_RAMP(1._EB,1._EB,SM%RAMP_CP_INDEX)*1000._EB
H2 = 0._EB
DO J=1,INT(H_F_REFERENCE_TEMPERATURE)+1
H1 = H2
CP1 = CP2
CP2 = EVALUATE_RAMP(REAL(J,EB),1._EB,SM%RAMP_CP_INDEX)*1000._EB
H2 = H2 + 0.5_EB*(CP1+CP2)
ENDDO
SM%REFERENCE_ENTHALPY = SM%H_F - &
(H1 + (H2-H1)*(H_F_REFERENCE_TEMPERATURE-INT(H_F_REFERENCE_TEMPERATURE)))
ELSE
IF (SM%REFERENCE_TEMPERATURE<=TWO_EPSILON_EB) CYCLE
CP2 = EVALUATE_RAMP(1._EB,1._EB,SM%RAMP_CP_INDEX)*1000._EB
H2 = 0._EB
DO J=1,INT(SM%REFERENCE_TEMPERATURE)+1
H1 = H2
CP1 = CP2
CP2 = EVALUATE_RAMP(REAL(J,EB),1._EB,SM%RAMP_CP_INDEX)*1000._EB
H2 = H2 + 0.5_EB*(CP1+CP2)
ENDDO
SM%REFERENCE_ENTHALPY = SM%REFERENCE_ENTHALPY - &
(H1 + (H2-H1)*(SM%REFERENCE_TEMPERATURE-INT(SM%REFERENCE_TEMPERATURE)))
IF (SM%H_F <= -1.E20) THEN
CP2 = EVALUATE_RAMP(1._EB,1._EB,SM%RAMP_CP_INDEX)*1000._EB
H2 = SM%REFERENCE_ENTHALPY
DO J=1,INT(H_F_REFERENCE_TEMPERATURE)+1
H1 = H2
CP1 = CP2
CP2 = EVALUATE_RAMP(REAL(J,EB),1._EB,SM%RAMP_CP_INDEX)*1000._EB
H2 = H2 + 0.5_EB*(CP1+CP2)
ENDDO
SM%H_F = H1 + (H2-H1)*(H_F_REFERENCE_TEMPERATURE-INT(H_F_REFERENCE_TEMPERATURE))
ENDIF
ENDIF
ENDIF
IF(SM%H_F <= -1.E20_EB) THEN
SM%H_F = 0._EB
DO J=1,N_SPECIES
SM%H_F = SM%H_F + SM%VOLUME_FRACTION(J) * SPECIES(J)%H_F ! Calculate H_F of mixtures
ENDDO
SM%H_F = SM%H_F/SM%MW*1000._EB
ENDIF
END DO

! Loop through temperatures from 1 K to 5000 K to get temperature-specific gas properties.  Data from JANAF 4

TABLE_LOOP: DO J=1,5000

! For each primitive species, get its property values at temperature J

DO N=1,N_SPECIES
SS => SPECIES(N)
CALL CALC_GAS_PROPS(J,N,D_TMP(N),MU_TMP(N),K_TMP(N),CP_TMP(N),H_TMP(N),SS%ISFUEL,G_F_TMP(N))
IF (SS%RAMP_CP_INDEX>0) THEN
CP_TMP(N) = EVALUATE_RAMP(REAL(J,EB),0._EB,SS%RAMP_CP_INDEX)*1000._EB
H_TMP(N) = SS%REFERENCE_ENTHALPY
ENDIF
IF (SS%RAMP_D_INDEX>0)    D_TMP(N)   = EVALUATE_RAMP(REAL(J,EB),1._EB,SS%RAMP_D_INDEX)
IF (SS%RAMP_G_F_INDEX>0) G_F_TMP(N) = EVALUATE_RAMP(REAL(J,EB),1._EB,SS%RAMP_G_F_INDEX)
IF (SS%RAMP_K_INDEX>0)    K_TMP(N)   = EVALUATE_RAMP(REAL(J,EB),1._EB,SS%RAMP_K_INDEX)/SQRT(SS%MW)
IF (SS%RAMP_MU_INDEX>0)   MU_TMP(N)  = EVALUATE_RAMP(REAL(J,EB),1._EB,SS%RAMP_MU_INDEX)/SQRT(SS%MW)
ENDDO

DO N=1,N_TRACKED_SPECIES
SM => SPECIES_MIXTURE(N)
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,&
SM%MU_USER,SM%K_USER,SM%MW,SM%SPECIFIC_HEAT,SM%REFERENCE_ENTHALPY,SM%REFERENCE_TEMPERATURE,SM%PR_USER)
IF (SM%RAMP_CP_INDEX>0) THEN
CP_TMP_Z(N) = EVALUATE_RAMP(REAL(J,EB),0._EB,SM%RAMP_CP_INDEX)*1000._EB
H_TMP_Z(N) = SM%REFERENCE_ENTHALPY
ENDIF
IF (SM%RAMP_D_INDEX>0)    D_TMP_Z(N)   = EVALUATE_RAMP(REAL(J,EB),1._EB,SM%RAMP_D_INDEX)
IF (SM%RAMP_G_F_INDEX>0) G_F_TMP_Z(N) = EVALUATE_RAMP(REAL(J,EB),1._EB,SM%RAMP_G_F_INDEX)
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)
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)
ENDDO
```

```fortran
! For each tracked species, store the mass-weighted property values

DO N=1,N_TRACKED_SPECIES
IF (SPECIES_MIXTURE(N)%REFERENCE_ENTHALPY < -1.E20_EB) SPECIES_MIXTURE(N)%REFERENCE_ENTHALPY = SUM(Z2Y(:,N) *
    H_TMP(:))
IF (D_TMP_Z(N) > 0._EB) THEN
D_Z(J,N) = D_TMP_Z(N)
ELSE
D_Z(J,N) = SPECIES_MIXTURE(N)%MW*SUM(Z2Y(:,N)*D_TMP(:)/SPECIES(:)%MW)
ENDIF
IF (CP_TMP_Z(N) > 0._EB) THEN
CP_Z(J,N) = CP_TMP_Z(N)
IF (J==1) CP_Z(0,N) = CP_Z(1,N)
H_SENS_Z(J,N) = H_SENS_Z(J-1,N) + 0.5_EB*(CP_Z(J,N)+CP_Z(J-1,N))
IF (J>1) THEN
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)
ELSE
CPBAR_Z(0,N) = H_TMP_Z(N)
CPBAR_Z(J,N) = CPBAR_Z(0,N) + CP_Z(J,N)
ENDIF
ELSE
CP_Z(J,N) = SUM(Z2Y(:,N) * CP_TMP(:))
IF (J==1) CP_Z(0,N) = CP_Z(1,N)
H_SENS_Z(J,N) = H_SENS_Z(J-1,N) + 0.5_EB*(CP_Z(J,N)+CP_Z(J-1,N))
IF (J>1) THEN
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)
ELSE
CPBAR_Z(0,N) = SUM(Z2Y(:,N) * H_TMP(:))
CPBAR_Z(J,N) = CPBAR_Z(0,N) + CP_Z(J,N)
ENDIF
ENDIF
IF (MU_TMP_Z(N) > 0._EB) THEN
MU_RSQMW_Z(J,N) = MU_TMP_Z(N)
ELSE
MU_RSQMW_Z(J,N) = SUM(Z2Y(:,N) * MU_TMP(:)) / SUM(Z2Y(:,N) * RSQ_MW_Y(:)) * RSQ_MW_Z(N)
ENDIF
IF (K_TMP_Z(N) > 0._EB) THEN
K_RSQMW_Z(J,N)  = K_TMP_Z(N)
ELSE
K_RSQMW_Z(J,N)  = SUM(Z2Y(:,N) * K_TMP(:)) / SUM(Z2Y(:,N) * RSQ_MW_Y(:)) * RSQ_MW_Z(N)
ENDIF
IF (G_F_TMP_Z(N) > 0._EB) THEN
G_F_Z(J,N) = G_F_TMP_Z(N)
ELSE
G_F_Z(J,N) = SUM(Z2Y(:,N) * G_F_TMP(:))
ENDIF
ENDDO
ENDDO TABLE_LOOP

! Adjust H_SENS_Z to 0 at the H_F_REFERENCE_TEMPERATURE
IF (CONSTANT_SPECIFIC_HEAT_RATIO) THEN
REF_TEMP = 0._EB
ELSE
REF_TEMP = H_F_REFERENCE_TEMPERATURE
ENDIF
J = INT(REF_TEMP)
H_REF_SENSIBLE(:) = H_SENS_Z(J,:)+(REF_TEMP-REAL(J,EB))*(H_SENS_Z(J+1,:)-H_SENS_Z(J,:))
H_SENS_Z(0,:) = -H_REF_SENSIBLE(:)
DO J = 1, 5000
H_SENS_Z(J,:) = H_SENS_Z(J,:)-H_REF_SENSIBLE(:)
!   CP_AVG_Z(J,:) = H_SENS_Z(J,:)/REAL(J,EB)
ENDDO

DO N=1,N_TRACKED_SPECIES
SM=>SPECIES_MIXTURE(N)
IF (SM%H_F <=-1.E20_EB) THEN
H1=CPBAR_Z(INT(H_F_REFERENCE_TEMPERATURE),N)*REAL(INT(H_F_REFERENCE_TEMPERATURE),EB)
H2=CPBAR_Z(INT(H_F_REFERENCE_TEMPERATURE)+1,N)*REAL(INT(H_F_REFERENCE_TEMPERATURE)+1,EB)
SM%H_F = H1+(H2-H1)*(H_F_REFERENCE_TEMPERATURE-REAL(INT(H_F_REFERENCE_TEMPERATURE),EB))
ENDIF
END DO

DEALLOCATE(RSQ_MW_Y)

DEALLOCATE(D_TMP)
DEALLOCATE(MU_TMP)
DEALLOCATE(CP_TMP)
DEALLOCATE(H_TMP)
DEALLOCATE(K_TMP)
DEALLOCATE(G_F_TMP)

DEALLOCATE(D_TMP_Z)
DEALLOCATE(MU_TMP_Z)
DEALLOCATE(CP_TMP_Z)
DEALLOCATE(H_TMP_Z)
DEALLOCATE(K_TMP_Z)

END SUBROUTINE PROC_SMIX
```

```fortran


SUBROUTINE READ_REAC

USE PROPERTY_DATA, ONLY : ELEMENT,GET_FORMULA_WEIGHT,MAKE_PERIODIC_TABLE,SIMPLE_SPECIES_MW,GAS_PROPS,LOOKUP_CHI_R
USE MATH_FUNCTIONS, ONLY : GET_RAMP_INDEX,GET_TABLE_INDEX
CHARACTER(LABEL_LENGTH) :: FUEL,RADCAL_ID='null',SPEC_ID_NU(MAX_SPECIES),SPEC_ID_N_S(MAX_SPECIES),RAMP_FS,TABLE_FS,&
RAMP_CHI_R
CHARACTER(LABEL_LENGTH) :: FORMULA
CHARACTER(255) :: EQUATION
CHARACTER(100) :: FWD_ID
INTEGER :: NR,NS,NS2,NFR
REAL(EB) :: SOOT_YIELD,CO_YIELD,EPUMO2,A, &
CRITICAL_FLAME_TEMPERATURE,HEAT_OF_COMBUSTION,E,C,H,N,O, &
AUTO_IGNITION_TEMPERATURE,SOOT_H_FRACTION,N_T,K,NU(MAX_SPECIES),N_S(MAX_SPECIES),&
FLAME_SPEED,FLAME_SPEED_EXPONENT,FLAME_SPEED_TEMPERATURE,&
TURBULENT_FLAME_SPEED_ALPHA,TURBULENT_FLAME_SPEED_EXPONENT,RADIATIVE_FRACTION !RADIATIVE_FRACTION_1 !Sesa-added RADIATIVE_FRACTION_1
REAL(EB) :: E_TMP=0._EB,S_TMP=0._EB,ATOM_COUNTS(118),MW_FUEL=0._EB,H_F=0._EB,PR_TMP
LOGICAL :: L_TMP,CHECK_ATOM_BALANCE,FAST_CHEMISTRY,REVERSE,THIRD_BODY,SERIES_REACTION      !Sesa-added SERIES_REACTION as in 6.2.0
NAMELIST /REAC/ A,AUTO_IGNITION_TEMPERATURE,C,CHECK_ATOM_BALANCE,CO_YIELD,CRITICAL_FLAME_TEMPERATURE,&
E,EPUMO2,K,EQUATION,FIXED_MIX_TIME,FLAME_SPEED,FLAME_SPEED_EXPONENT,FLAME_SPEED_TEMPERATURE,FORMULA,FUEL,&
FUEL_RADCAL_ID,FWD_ID,FYI,H,HEAT_OF_COMBUSTION,&
ID,IDEAL,N,NU,N_S,N_T,O,ODE_SOLVER,RADIATIVE_FRACTION,RAMP_CHI_R,RAMP_FS,REAC_ATOM_ERROR,&
REAC_MASS_ERROR,REVERSE,SOOT_H_FRACTION,SOOT_YIELD,&
SPEC_ID_N_S,SPEC_ID_NU,TABLE_FS,TAU_CHEM,TAU_FLAME,&
THIRD_BODY,TURBULENT_FLAME_SPEED_ALPHA,TURBULENT_FLAME_SPEED_EXPONENT,Y_P_MIN_EDC,&
HRRPUA_SHEET,HRRPUV_AVERAGE !RADIATIVE_FRACTION_1            !Sesa-added HRRPUA_SHEET and HRRPUV_AVERAGE RADIATIVE_FRACTION_1 as in 6.2.0

CALL MAKE_PERIODIC_TABLE
CALL SIMPLE_SPECIES_MW
ATOM_COUNTS = 0._EB
N_REACTIONS = 0
REWIND(LU_INPUT) ; INPUT_FILE_LINE_NUMBER = 0

COUNT_REAC_LOOP: DO
CALL CHECKREAD('REAC',LU_INPUT,IOS)
IF (IOS==1) EXIT COUNT_REAC_LOOP
CALL SET_REAC_DEFAULTS
READ(LU_INPUT,REAC,END=435,ERR=434,IOSTAT=IOS)
N_REACTIONS = N_REACTIONS + 1
IF (A < 0._EB .AND. E < 0._EB .AND. TRIM(SPEC_ID_NU(1))=='null' .AND. TRIM(EQUATION)=='null') SIMPLE_CHEMISTRY = .TRUE.
IF ((A > 0._EB .OR. E > 0._EB) .AND. (SPEC_ID_N_S(1)=='null' .OR. N_S(1) < -998._EB)) THEN
WRITE(MESSAGE,'(A,I0,A)') 'ERROR: Problem with REAC ',N_REACTIONS,'. SPEC_ID_N_S and N_S arrays must be defined'
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF
IF (.NOT.SIMPLE_CHEMISTRY .AND. TRIM(SPEC_ID_NU(1))=='null' .AND. TRIM(EQUATION)=='null') THEN
WRITE(MESSAGE,'(A,I0,A)') 'ERROR: Problem with REAC ',N_REACTIONS,'. SPEC_ID_NU and NU arrays or EQUATION must be defined'
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF
434 IF (IOS>0) THEN
WRITE(MESSAGE,'(A,I0)') 'ERROR: Problem with REAC ',N_REACTIONS+1
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF
ENDDO COUNT_REAC_LOOP

435 REWIND(LU_INPUT) ; INPUT_FILE_LINE_NUMBER = 0

ALLOCATE(REACTION(N_REACTIONS),STAT=IZERO)

! Read and store the reaction parameters

NFR = 0 ! Number of fast reactions

REAC_READ_LOOP: DO NR=1,N_REACTIONS

! Read the REAC line

CALL CHECKREAD('REAC',LU_INPUT,IOS)
IF (IOS==1) EXIT REAC_READ_LOOP
CALL SET_REAC_DEFAULTS
READ(LU_INPUT,REAC)

! Ensure that there is a specified fuel

IF (FUEL=='null' .AND. ID/='null') FUEL = ID ! Backward compatibility

IF (FUEL=='null' .AND. ID=='null') THEN
WRITE(MESSAGE,'(A,I0,A)') 'ERROR: REAC ',NR,' requires a FUEL'
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF

! Set up the SIMPLE_CHEMISTRY model
```

```

RN => REACTION(NR)
IF (SIMPLE_CHEMISTRY) THEN
IF(C<=TWO_EPSILON_EB .AND. H<=TWO_EPSILON_EB) THEN
IF (TRIM(FORMULA)=='null') THEN
CALL GAS_PROPS(FUEL,S_TMP,E_TMP,PR_TMP,MW_FUEL,FORMULA,L_TMP,ATOM_COUNTS,H_F,RADCAL_ID)
ELSE
CALL GET_FORMULA_WEIGHT(FORMULA,MW_FUEL,ATOM_COUNTS)
L_TMP = .TRUE.
ENDIF
IF (L_TMP) THEN
SIMPLE_FUEL_DEFINED = .TRUE.
IF (ATOM_COUNTS(1)+ATOM_COUNTS(6)+ATOM_COUNTS(7)+ATOM_COUNTS(8) - SUM(ATOM_COUNTS) < 0._EB) THEN
WRITE(MESSAGE,'(A)') 'ERROR: Fuel FORMULA for SIMPLE_CHEMISTRY can only contain C,H,O, and N'
CALL SHUTDOWN(MESSAGE) ; RETURN
ELSE
C = ATOM_COUNTS(6)
H = ATOM_COUNTS(1)
O = ATOM_COUNTS(8)
N = ATOM_COUNTS(7)
ENDIF
IF (C<=TWO_EPSILON_EB .AND. H<=TWO_EPSILON_EB) THEN
WRITE(MESSAGE,'(A)') 'ERROR: Must specify fuel chemistry using C and/or H when using simple chemistry'
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF
ENDIF
ELSE
SIMPLE_FUEL_DEFINED = .TRUE.
MW_FUEL = ELEMENT(6)%MASS*C+ELEMENT(1)%MASS*H+ELEMENT(8)%MASS*O+ELEMENT(7)%MASS*N
ENDIF
ENDIF
RN%A_IN                        = A
RN%A_PRIME                     = A
RN%AUTO_IGNITION_TEMPERATURE   = AUTO_IGNITION_TEMPERATURE + TMPM
RN%C                           = C
RN%CHECK_ATOM_BALANCE          = CHECK_ATOM_BALANCE
RN%CO_YIELD                    = CO_YIELD
RN%CRIT_FLAME_TMP              = CRITICAL_FLAME_TEMPERATURE + TMPM
RN%E                           = E*1000._EB
RN%E_IN                        = E
RN%K                           = K
RN%EQUATION                    = EQUATION
RN%EPUMO2                      = EPUMO2*1000._EB
RN%FAST_CHEMISTRY              = FAST_CHEMISTRY
RN%FLAME_SPEED                 = FLAME_SPEED
RN%FLAME_SPEED_EXPONENT        = FLAME_SPEED_EXPONENT
RN%FLAME_SPEED_TEMPERATURE     = FLAME_SPEED_TEMPERATURE + TMPM
IF (RN%FLAME_SPEED_TEMPERATURE <=0._EB) RN%FLAME_SPEED_TEMPERATURE = TMPA
RN%FUEL                        = FUEL
RN%FWD_ID                      = FWD_ID
RN%FYI                         = FYI
RN%H                           = H
RN%HEAT_OF_COMBUSTION          = HEAT_OF_COMBUSTION*1000._EB
RN%ID                          = ID
RN%MW_FUEL                     = MW_FUEL
RN%MW_SOOT                     = ELEMENT(6)%MASS * (1._EB-SOOT_H_FRACTION) + ELEMENT(1)%MASS*SOOT_H_FRACTION
RN%N                           = N
RN%N_T                         = N_T
RN%O                           = O
RN%RAMP_CHI_R                  = RAMP_CHI_R
RN%RAMP_FS                     = RAMP_FS
RN%TABLE_FS                    = TABLE_FS
RN%REVERSE                     = REVERSE
RN%SOOT_H_FRACTION             = SOOT_H_FRACTION
RN%SOOT_YIELD                  = SOOT_YIELD
RN%THIRD_BODY                  = THIRD_BODY
RN%TURBULENT_FLAME_SPEED_ALPHA    = TURBULENT_FLAME_SPEED_ALPHA
RN%TURBULENT_FLAME_SPEED_EXPONENT = TURBULENT_FLAME_SPEED_EXPONENT
RN%SERIES_REACTION             = SERIES_REACTION     !Sesa-added as in 6.2.0

IF (RN%RAMP_FS/='null' .AND. RN%TABLE_FS/='null') THEN
WRITE(MESSAGE,'(A)') 'ERROR: Can only specify one of RAMP_FS or TABLE_FS'
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF
IF (RN%RAMP_FS/='null') CALL GET_RAMP_INDEX(RN%RAMP_FS,'EQUIVALENCE RATIO',RN%RAMP_FS_INDEX)
IF (RN%RAMP_CHI_R/='null') CALL GET_RAMP_INDEX(RN%RAMP_CHI_R,'TIME',RN%RAMP_CHI_R_INDEX)
IF (RN%TABLE_FS/='null') CALL GET_TABLE_INDEX(RN%TABLE_FS,FLAME_SPEED_TABLE,RN%TABLE_FS_INDEX)

IF (RN%A_PRIME==-1._EB .AND. RN%E==-1000._EB .AND. .NOT.RN%REVERSE) THEN
RN%FAST_CHEMISTRY=.TRUE.
NFR = NFR + 1
ENDIF

! Check appropriate extinction model

IF (NFR > 1 .AND. (EXTINCT_MOD == 2 .OR. EXTINCT_MOD == 6) .AND. SUPPRESSION) THEN
WRITE(MESSAGE,'(A)') 'ERROR: The default EXTINCTION MODEL is designed for 1 reaction. See Tech Guide'
CALL SHUTDOWN(MESSAGE) ; RETURN
```

```
ENDIF

! Determine the number of stoichiometric coefficients for this reaction

IF (.NOT.SIMPLE_CHEMISTRY) THEN
NS2 = 0
DO NS=1,MAX_SPECIES
IF (TRIM(SPEC_ID_NU(NS))/='null') THEN
NS2=NS2+1
ELSE
EXIT
ENDIF
ENDDO
RN%N_SMIX = NS2
NS2 = 0
IF(TRIM(RN%EQUATION)/='null') RN%N_SMIX = MAX_SPECIES
DO NS=1,MAX_SPECIES
IF (TRIM(SPEC_ID_N_S(NS))/='null') THEN
NS2=NS2+1
ELSE
EXIT
ENDIF
ENDDO
RN%N_SPEC = NS2
ELSE
RN%N_SMIX = 3
RN%N_SPEC = 0
ENDIF

! Store the "read in" values of N_S, NU, and SPEC_ID_NU for use in PROC_REAC.

IF (RN%N_SPEC > 0) THEN
ALLOCATE(RN%N_S_READ(RN%N_SPEC))
RN%N_S_READ(1:RN%N_SPEC) = N_S(1:RN%N_SPEC)
ALLOCATE(RN%SPEC_ID_N_S_READ(RN%N_SPEC))
RN%SPEC_ID_N_S_READ = 'null'
RN%SPEC_ID_N_S_READ(1:RN%N_SPEC)=SPEC_ID_N_S(1:RN%N_SPEC)
ENDIF
ALLOCATE(RN%NU_READ(RN%N_SMIX))
RN%NU_READ(1:RN%N_SMIX) = NU(1:RN%N_SMIX)

ALLOCATE(RN%SPEC_ID_NU_READ(RN%N_SMIX))
RN%SPEC_ID_NU_READ = 'null'
RN%SPEC_ID_NU_READ(1:RN%N_SMIX)=SPEC_ID_NU(1:RN%N_SMIX)

IF (RADIATIVE_FRACTION < 0._EB) THEN
IF (DNS) RN%CHI_R = 0._EB
IF (LES) CALL LOOKUP_CHI_R(FUEL,RN%CHI_R)
ELSE
RN%CHI_R = RADIATIVE_FRACTION
ENDIF

!Sesa-added for RADIATIVE_FRACTION_1
! IF (LES) RADIATIVE_FRACTION_1 = 0.35_EB
! IF (DNS) RADIATIVE_FRACTION_1 = 0.00_EB
!Sesa-adding end

ENDDO REAC_READ_LOOP

REWIND(LU_INPUT) ; INPUT_FILE_LINE_NUMBER = 0

CONTAINS

SUBROUTINE SET_REAC_DEFAULTS

AUTO_IGNITION_TEMPERATURE    = -TMPM
A                            = -1._EB
C                            = 0._EB
CHECK_ATOM_BALANCE           = .TRUE.
CO_YIELD                     = 0._EB
CRITICAL_FLAME_TEMPERATURE   = 1427._EB    ! C (See C. Beyler, Ch 7, Eq 6, SFPE Handbook, 4th Ed.)
E                            = -1._EB      ! kJ/kmol
EPUMO2                       = 13100._EB   ! kJ/kg
K                            = 1._EB
EQUATION                     = 'null'
FAST_CHEMISTRY               = .FALSE.
FLAME_SPEED                  = -1._EB
FLAME_SPEED_EXPONENT         = 0._EB
FLAME_SPEED_TEMPERATURE      = -273.15_EB
FORMULA                      = 'null'
FUEL                         = 'null'
FWD_ID                       = 'null'
FYI                          = 'null'
H                            = 0._EB
HEAT_OF_COMBUSTION           = -2.E20_EB
ID                           = 'null'
N                            = 0._EB
NU                           = 0._EB
```

```
N_S                            = -999._EB
N_T                            = 0._EB
O                              = 0._EB
ODE_SOLVER                     = 'null'
RADIATIVE_FRACTION             = -1._EB
RAMP_CHI_R                     = 'null'
RAMP_FS                        = 'null'
REAC_ATOM_ERROR                = 1.E-4_EB
REAC_MASS_ERROR                = 1.E-4_EB
REVERSE                        = .FALSE.
SOOT_H_FRACTION                = 0.1_EB
SOOT_YIELD                     = 0.0_EB
SPEC_ID_NU                     = 'null'
SPEC_ID_N_S                    = 'null'
TABLE_FS                       = 'null'
THIRD_BODY                     = .FALSE.
TURBULENT_FLAME_SPEED_ALPHA     = 1._EB ! see O. Gulder. 23rd Int. Symp. on Comb. 1990.
TURBULENT_FLAME_SPEED_EXPONENT = 2._EB
SERIES_REACTION                = .FALSE.     !Sesa-added as in 6.2.0

END SUBROUTINE SET_REAC_DEFAULTS

END SUBROUTINE READ_REAC


SUBROUTINE PROC_REAC_1
USE PROPERTY_DATA, ONLY : PARSE_EQUATION, SHUTDOWN_ATOM
REAL(EB) :: MASS_PRODUCT,MASS_REACTANT,REACTION_BALANCE(118)
INTEGER :: NS,NS2,NR,NSPEC
LOGICAL :: NAME_FOUND,SKIP_ATOM_BALANCE
TYPE (SPECIES_MIXTURE_TYPE), POINTER :: SM

IF (N_REACTIONS <=0) RETURN

! Basic input error checking

IF (SIMPLE_CHEMISTRY .AND. N_REACTIONS > 1) THEN
WRITE(MESSAGE,'(A)') 'ERROR: can not have more than one reaction when using simple chemistry'
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF

! The following information is what the user would have entered into the input file in the more general case

IF (SIMPLE_CHEMISTRY) THEN
RN => REACTION(1)
IF (RN%NU_O2<=0._EB) THEN
WRITE(MESSAGE,'(A)') 'ERROR: Fuel specified for simple chemistry has NU_O2 <=0 and it must require air for
       combustion.'
CALL SHUTDOWN(MESSAGE)          ; RETURN
ENDIF
RN%SPEC_ID_NU_READ(1) = RN%FUEL
RN%SPEC_ID_NU_READ(2) = 'AIR'
RN%SPEC_ID_NU_READ(3) = 'PRODUCTS'
RN%NU_READ(1)         = -1._EB
RN%NU_READ(2)         = -RN%NU_O2/SPECIES_MIXTURE(1)%VOLUME_FRACTION(O2_INDEX)
RN%NU_READ(3)         = -(RN%NU_READ(1)*SPECIES_MIXTURE(FUEL_SMIX_INDEX)%MW+RN%NU_READ(2)*SPECIES_MIXTURE(1)%MW)/ &
SPECIES_MIXTURE(3)%MW
RN%N_SMIX             = 3
ENDIF

REAC_LOOP: DO NR=1,N_REACTIONS

RN => REACTION(NR)

IF ((RN%A_PRIME > 0._EB .OR. RN%E > 0._EB) .AND. (RN%C>TWO_EPSILON_EB .OR. RN%H>TWO_EPSILON_EB)) THEN
WRITE(MESSAGE,'(A)') 'ERROR: cannot use both finite rate REAC and simple chemistry'
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF
IF (TRIM(RN%EQUATION)/='null') THEN
IF(ANY(ABS(RN%NU_READ)>TWO_EPSILON_EB)) THEN
WRITE(MESSAGE,'(A,I0,A)') 'ERROR: Problem with REAC ',NR,'. Cannot set NUs if an EQUATION is specified.'
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF
CALL PARSE_EQUATION(NR)
RN%N_SMIX = 0
DO NS=1,N_TRACKED_SPECIES+1
IF(ABS(RN%NU_READ(NS))>TWO_EPSILON_EB) THEN
RN%N_SMIX = RN%N_SMIX+1
ENDIF
ENDDO
ENDIF

IF (TRIM(RN%FUEL)=='null') THEN
WRITE(MESSAGE,'(A,I0,A)') 'ERROR: Problem with REAC ',NR,'. FUEL must be defined'
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF

! Allocate the arrays that are going to carry the mixture stoichiometry to the rest of the code
```

```
ALLOCATE(RN%SPEC_ID_NU(1:N_TRACKED_SPECIES))
ALLOCATE(RN%NU(1:N_TRACKED_SPECIES))
ALLOCATE(RN%NU_MW_O_MW_F(1:N_TRACKED_SPECIES))
ALLOCATE(RN%SPEC_ID_N_S(1:N_SPECIES))
ALLOCATE(RN%N_S(1:N_SPECIES))
RN%SPEC_ID_NU   = 'null'
RN%SPEC_ID_N_S  = 'null'
RN%NU           = 0._EB
RN%N_S          = -999._EB

! Transfer SPEC_ID_NU, SPEC_ID_N, NU, and N_S that were indexed by the order they were read in
! to now be indexed by the SMIX or SPEC index

DO NS=1,RN%N_SMIX
IF (TRIM(RN%SPEC_ID_NU_READ(NS))=='null') CYCLE
NAME_FOUND = .FALSE.
DO NS2=1,N_TRACKED_SPECIES
IF (TRIM(RN%SPEC_ID_NU_READ(NS))==TRIM(SPECIES_MIXTURE(NS2)%ID)) THEN
RN%SPEC_ID_NU(NS2) = RN%SPEC_ID_NU_READ(NS)
RN%NU(NS2)         = RN%NU_READ(NS)
NAME_FOUND = .TRUE.
EXIT
ENDIF
IF (TRIM(RN%EQUATION)/='null') THEN
IF (TRIM(RN%SPEC_ID_NU_READ(NS))==TRIM(SPECIES_MIXTURE(NS2)%FORMULA)) THEN
RN%SPEC_ID_NU(NS2) = SPECIES_MIXTURE(NS2)%ID
RN%NU(NS2)         = RN%NU_READ(NS)
NAME_FOUND = .TRUE.
EXIT
ENDIF
ENDIF
ENDDO
IF (.NOT. NAME_FOUND) THEN
WRITE(MESSAGE,'(A,I0,A,A,A)') 'ERROR: Problem with REAC ',NR,'. Tracked species ',TRIM(RN%SPEC_ID_NU_READ(NS)),&
' not found.'
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF
ENDDO

! Look for indices of fuels, oxidizers, and products. Normalize the stoichiometric coefficients by that of the fuel.

DO NS2=1,N_TRACKED_SPECIES
IF (ABS(RN%NU(NS2))<TWO_EPSILON_EB) CYCLE
DO NSPEC=1,N_SPECIES
IF (SPECIES_MIXTURE(NS2)%SPEC_ID(NSPEC)==RN%FUEL .OR. SPECIES_MIXTURE(NS2)%ID==RN%FUEL) THEN
RN%FUEL_SMIX_INDEX = NS2
RN%NU = -RN%NU/RN%NU(NS2)
EXIT
ENDIF
ENDDO
ENDDO

! Find AIR index

GET_AIR_INDEX_LOOP: DO NS = 1,N_TRACKED_SPECIES
IF (RN%NU(NS) < 0._EB .AND. NS /= RN%FUEL_SMIX_INDEX) THEN
RN%AIR_SMIX_INDEX = NS
EXIT GET_AIR_INDEX_LOOP
ENDIF
ENDDO GET_AIR_INDEX_LOOP

! Adjust mol/cm^3/s based rate to kg/m^3/s rate

RN%RHO_EXPONENT = 0._EB
DO NS=1,RN%N_SPEC
IF (TRIM(RN%SPEC_ID_N_S_READ(NS))=='null') CYCLE
IF (RN%A_PRIME < 0.0_EB) CYCLE
NAME_FOUND = .FALSE.
DO NS2=1,N_SPECIES
IF (TRIM(RN%SPEC_ID_N_S_READ(NS))==TRIM(SPECIES(NS2)%ID)) THEN
RN%SPEC_ID_N_S(NS2) = RN%SPEC_ID_N_S_READ(NS)
RN%N_S(NS2)         = RN%N_S_READ(NS)
RN%A_PRIME          = RN%A_PRIME * (1000._EB*SPECIES(NS2)%MW)**(-RN%N_S(NS2)) ! FDS Tech Guide, Eq. (5.46), product term
RN%RHO_EXPONENT     = RN%RHO_EXPONENT + RN%N_S(NS2)
NAME_FOUND = .TRUE.
EXIT
ENDIF
ENDDO
IF (.NOT. NAME_FOUND) THEN
WRITE(MESSAGE,'(A,I0,A,A,A)') &
'ERROR: Problem with REAC ',NR,'. Primitive species ',TRIM(RN%SPEC_ID_N_S_READ(NS)),' not found.'
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF
ENDDO
RN%RHO_EXPONENT = RN%RHO_EXPONENT - 1._EB ! subtracting 1 accounts for division by rho in Eq. (5.49)
```

```
RN%A_PRIME = RN%A_PRIME * 1000._EB*SPECIES_MIXTURE(RN%FUEL_SMIX_INDEX)%MW ! conversion terms in Eq. (5.46)

! Adjust mol/cm^3/s based rate to kg/m^3/s rate for FAST_CHEMISTRY (this will get removed when we overhaul
    combustion)
! Fictitious Arrhenius rate is dC_F/dt = -1E10*C_F*C_A

IF (RN%FAST_CHEMISTRY) THEN
IF (RN%AIR_SMIX_INDEX > -1) THEN
RN%RHO_EXPONENT_FAST = 1._EB
RN%A_PRIME_FAST = 1.E10_EB*(1000._EB*SPECIES_MIXTURE(RN%AIR_SMIX_INDEX)%MW)**(-1._EB)
ELSE
RN%RHO_EXPONENT_FAST = 0._EB
RN%A_PRIME_FAST = 1.E10_EB
ENDIF
ENDIF

! Compute the primitive species reaction coefficients

ALLOCATE(RN%NU_SPECIES(N_SPECIES))
RN%NU_SPECIES = 0._EB
DO NS=1,N_TRACKED_SPECIES
SM => SPECIES_MIXTURE(NS)
RN%NU(NS) = RN%NU(NS)*SM%ADJUST_NU
DO NS2 = 1,N_SPECIES
RN%NU_SPECIES(NS2) = RN%NU_SPECIES(NS2) + RN%NU(NS)*SM%VOLUME_FRACTION(NS2)
ENDDO
IF (SM%ID=='WATER VAPOR') I_WATER = NS
IF (SM%ID=='CARBON DIOXIDE') I_CO2 = NS
ENDDO

! Check atom balance of the reaction

IF (.NOT. SIMPLE_CHEMISTRY .AND. RN%CHECK_ATOM_BALANCE) THEN
SKIP_ATOM_BALANCE = .FALSE.
REACTION_BALANCE = 0._EB
DO NS=1,N_TRACKED_SPECIES
IF (ABS(RN%NU(NS))>TWO_EPSILON_EB .AND. .NOT. SPECIES_MIXTURE(NS)%VALID_ATOMS) SKIP_ATOM_BALANCE = .TRUE.
REACTION_BALANCE = REACTION_BALANCE + RN%NU(NS)*SPECIES_MIXTURE(NS)%ATOMS
ENDDO
IF (ANY(ABS(REACTION_BALANCE)>REAC_ATOM_ERROR) .AND. .NOT. SKIP_ATOM_BALANCE) THEN
CALL SHUTDOWN_ATOM(REACTION_BALANCE,NR,REAC_ATOM_ERROR) ; RETURN
ENDIF
ENDIF

! Check the mass balance of the reaction

MASS_REACTANT = 0._EB
MASS_PRODUCT  = 0._EB

DO NS=1,N_TRACKED_SPECIES
IF (RN%NU(NS) < -TWO_EPSILON_EB) MASS_REACTANT = MASS_REACTANT + RN%NU(NS)*SPECIES_MIXTURE(NS)%MW
IF (RN%NU(NS) >  TWO_EPSILON_EB) MASS_PRODUCT  = MASS_PRODUCT  + RN%NU(NS)*SPECIES_MIXTURE(NS)%MW
ENDDO
IF (ABS(MASS_PRODUCT) < TWO_EPSILON_EB .OR. ABS(MASS_REACTANT) < TWO_EPSILON_EB) THEN
IF (ABS(MASS_PRODUCT) <TWO_EPSILON_EB) WRITE(MESSAGE,'(A,I0,A)') 'ERROR: Problem with REAC ',NR,'. Products not
    specified.'
IF (ABS(MASS_REACTANT)<TWO_EPSILON_EB) WRITE(MESSAGE,'(A,I0,A)') 'ERROR: Problem with REAC ',NR,'. Reactants not
    specified.'
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF
IF (ABS(MASS_PRODUCT+MASS_REACTANT)/ABS(MASS_PRODUCT) > REAC_MASS_ERROR) THEN
WRITE(MESSAGE,'(A,I0,A,F8.3,A,F8.3)') 'ERROR: Problem with REAC ',NR,'. Mass of products, ',MASS_PRODUCT, &
', does not equal mass of reactants,',-MASS_REACTANT
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF

! Mass stoichiometric coefficient of oxidizer

DO NS=1,N_TRACKED_SPECIES
RN%NU_MW_O_MW_F(NS) = RN%NU(NS)*SPECIES_MIXTURE(NS)%MW/SPECIES_MIXTURE(RN%FUEL_SMIX_INDEX)%MW
IF (RN%NU(NS)< 0._EB .AND. NS /= RN%FUEL_SMIX_INDEX) THEN
RN%S = -RN%NU_MW_O_MW_F(NS)
ENDIF
ENDDO

ENDDO REAC_LOOP

! Select integrator

IF (TRIM(ODE_SOLVER)/='null') THEN
SELECT CASE (TRIM(ODE_SOLVER))
CASE ('EXPLICIT EULER')
COMBUSTION_ODE_SOLVER = EXPLICIT_EULER
CASE ('RK2')
COMBUSTION_ODE_SOLVER = RK2
CASE ('RK3')
COMBUSTION_ODE_SOLVER = RK3
CASE ('RK2 RICHARDSON')
```

```
COMBUSTION_ODE_SOLVER = RK2_RICHARDSON
CASE DEFAULT
WRITE(MESSAGE,'(A)') 'ERROR: Problem with REAC. Name of ODE_SOLVER is not recognized.'
CALL SHUTDOWN(MESSAGE) ; RETURN
END SELECT
ELSE
FAST_CHEM_LOOP: DO NR = 1,N_REACTIONS
RN => REACTION(NR)
IF (.NOT. RN%FAST_CHEMISTRY) THEN
COMBUSTION_ODE_SOLVER = RK2_RICHARDSON
EXIT FAST_CHEM_LOOP
ELSE
COMBUSTION_ODE_SOLVER = EXPLICIT_EULER
ENDIF
ENDDO FAST_CHEM_LOOP
ENDIF

END SUBROUTINE PROC_REAC_1


SUBROUTINE PROC_REAC_2
INTEGER :: NS,NR,HF_COUNT,I_HFRT,NRR          !Sesa-added NRR as in 6.2.0
REAL(EB) :: H_F_OLD(1:N_TRACKED_SPECIES),D_HFRT,D_H
LOGICAL :: REDEFINE_H_F(1:N_TRACKED_SPECIES),LISTED_FUEL
TYPE (SPECIES_MIXTURE_TYPE), POINTER :: SM,SMF
TYPE(REACTION_TYPE), POINTER :: RN=>NULL()

IF (N_REACTIONS <=0) RETURN

REDEFINE_H_F = .FALSE.
LISTED_FUEL = .FALSE.
H_F_OLD = SPECIES_MIXTURE%H_F

REAC_LOOP: DO NR=1,N_REACTIONS

RN => REACTION(NR)
SMF => SPECIES_MIXTURE(RN%FUEL_SMIX_INDEX)
IF (SIMPLE_CHEMISTRY) THEN
LISTED_FUEL = .TRUE.
DO NS=1,N_SPECIES
IF (SMF%VOLUME_FRACTION(NS)>0._EB .AND. .NOT. SPECIES(NS)%EXPLICIT_H_F) LISTED_FUEL=.FALSE.
IF (.NOT. SPECIES(NS)%LISTED .AND. SPECIES(NS)%RADCAL_ID=='null') SPECIES(NS)%RADCAL_ID=FUEL_RADCAL_ID
ENDDO
ENDIF

! Heat of Combustion calculation
HOC_IF: IF (RN%HEAT_OF_COMBUSTION > -1.E21) THEN ! User specified heat of combustion
IF (SIMPLE_CHEMISTRY) THEN
IF (IDEAL) THEN
RN%HEAT_OF_COMBUSTION = RN%HEAT_OF_COMBUSTION * SMF%MW * 0.001 !J/kg -> J/mol
RN%HEAT_OF_COMBUSTION = RN%HEAT_OF_COMBUSTION + ( &
RN%NU_CO * (SPECIES(CO2_INDEX)%H_F - SPECIES(CO_INDEX)%H_F) &
+ RN%NU_SOOT * SPECIES(CO2_INDEX)%H_F * (1._EB - RN%SOOT_H_FRACTION) &
+ RN%NU_SOOT * SPECIES(H2O_INDEX)%H_F * RN%SOOT_H_FRACTION * 0.5_EB)
RN%HEAT_OF_COMBUSTION = RN%HEAT_OF_COMBUSTION / SMF%MW * 1000._EB !J/mol->J/kg
ENDIF
RN%EPUMO2 = RN%HEAT_OF_COMBUSTION*SMF%MW / (RN%NU_O2 * SPECIES(O2_INDEX)%MW)
ENDIF
HF_COUNT = 0
DO NS = 1,N_TRACKED_SPECIES
IF (RN%NU(NS) /= 0._EB) THEN
IF (SPECIES_MIXTURE(NS)%H_F <= -1.E21) HF_COUNT = HF_COUNT +1
IF (HF_COUNT > 1) THEN
WRITE(MESSAGE,'(A,I0,A)') 'ERROR: Problem with REAC ',NR,'. Missing more than 1 species heat of formation.'
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF
ENDIF
ENDDO
! Find heat of formation of lumped fuel to satisfy specified heat of combustion
IF (REDEFINE_H_F(RN%FUEL_SMIX_INDEX)) THEN
WRITE(MESSAGE,'(A,I0,A)') 'WARNING: H_F for FUEL for REACtion ',NR,' was redefined multiple times.'
IF (MYID==0) WRITE(LU_ERR,'(A)') TRIM(MESSAGE)
ENDIF
REDEFINE_H_F(RN%FUEL_SMIX_INDEX) = .TRUE.
SMF%H_F = RN%HEAT_OF_COMBUSTION * ABS(RN%NU(RN%FUEL_SMIX_INDEX)) * SMF%MW * 0.001_EB
DO NS = 1,N_TRACKED_SPECIES
IF (NS == RN%FUEL_SMIX_INDEX) CYCLE
SM=>SPECIES_MIXTURE(NS)
SMF%H_F = SMF%H_F + RN%NU(NS) * SM%H_F * SM%MW * 0.001_EB
ENDDO
IF (SMF%SINGLE_SPEC_INDEX>0) SPECIES(SMF%SINGLE_SPEC_INDEX)%H_F = -SMF%H_F/ RN%NU(RN%FUEL_SMIX_INDEX)
SMF%H_F = -SMF%H_F/ (RN%NU(RN%FUEL_SMIX_INDEX) * SMF%MW * 0.001_EB)


ELSE HOC_IF ! Heat of combustion not specified
IF (SIMPLE_CHEMISTRY) THEN ! Calculate heat of combustion based oxygen consumption
IF (RN%EPUMO2 > 0._EB .AND. .NOT. LISTED_FUEL ) THEN
RN%HEAT_OF_COMBUSTION = -RN%EPUMO2 * RN%NU_SPECIES(O2_INDEX) * SPECIES(O2_INDEX)%MW / SMF%MW
```

```
REDEFINE_H_F(RN%FUEL_SMIX_INDEX) = .TRUE.
SMF%H_F = RN%HEAT_OF_COMBUSTION * ABS(RN%NU(RN%FUEL_SMIX_INDEX)) * SMF%MW * 0.001_EB
DO NS = 1,N_TRACKED_SPECIES
IF  (NS == RN%FUEL_SMIX_INDEX) CYCLE
SM=>SPECIES_MIXTURE(NS)
SMF%H_F = SMF%H_F + RN%NU(NS) * SM%H_F * SM%MW * 0.001_EB
ENDDO
IF (SMF%SINGLE_SPEC_INDEX>0) SPECIES(SMF%SINGLE_SPEC_INDEX)%H_F = -SMF%H_F/ RN%NU(RN%FUEL_SMIX_INDEX)
SMF%H_F = -SMF%H_F/ (RN%NU(RN%FUEL_SMIX_INDEX) * SMF%MW * 0.001_EB)
ELSE
RN%HEAT_OF_COMBUSTION = 0._EB
DO NS = 1,N_TRACKED_SPECIES
SM=>SPECIES_MIXTURE(NS)
RN%HEAT_OF_COMBUSTION = RN%HEAT_OF_COMBUSTION - RN%NU(NS) * SM%H_F * SM%MW
ENDDO
RN%HEAT_OF_COMBUSTION =  RN%HEAT_OF_COMBUSTION / SMF%MW
RN%EPUMO2 = RN%HEAT_OF_COMBUSTION * SMF%MW / (RN%NU_O2 * SPECIES(O2_INDEX)%MW)
ENDIF
ELSE
RN%HEAT_OF_COMBUSTION = 0._EB
DO NS = 1,N_TRACKED_SPECIES
SM=>SPECIES_MIXTURE(NS)
RN%HEAT_OF_COMBUSTION = RN%HEAT_OF_COMBUSTION - RN%NU(NS) * SM%H_F * SM%MW
ENDDO
RN%HEAT_OF_COMBUSTION = RN%HEAT_OF_COMBUSTION / SPECIES_MIXTURE(RN%FUEL_SMIX_INDEX)%MW
ENDIF
ENDIF HOC_IF

IF (NR==1) REACTION%HOC_COMPLETE = RN%HEAT_OF_COMBUSTION

ENDDO REAC_LOOP

!Sesa-Added this as in 6.2.0
! Determine number of fast/potentially fast series reactions
N_SERIES_REACTIONS = 0
SERIES_REACTION_LOOP: DO NR = 1,N_REACTIONS
RN => REACTION(NR)
RN%SERIES_REACTION = .FALSE.
! Special treatment of series reactions is only needed for fast chemistry
FAST_CHEM_IF: IF (RN%FAST_CHEMISTRY) THEN

! Determine whether a product of reaction RN is fuel of a different reaction RNN
SERIES_SPECIES_LOOP: DO NS = 1,N_TRACKED_SPECIES
PRODUCTS_IF: IF (RN%NU(NS) > 0._EB) THEN

RNN_LOOP: DO NRR = 1,N_REACTIONS
RNN => REACTION(NRR)
IF (RNN%FAST_CHEMISTRY) THEN
IF (NRR == NR) CYCLE RNN_LOOP
IF (NS == RNN%FUEL_SMIX_INDEX) THEN
N_SERIES_REACTIONS = N_SERIES_REACTIONS + 1
RN%SERIES_REACTION = .TRUE.
ENDIF
ENDIF
ENDDO RNN_LOOP

ENDIF PRODUCTS_IF
ENDDO SERIES_SPECIES_LOOP

ENDIF FAST_CHEM_IF
ENDDO SERIES_REACTION_LOOP
!Sesa-end added as in 6.2.0

! Correct CP_BAR array if H_F is redefined.
IF (ANY(REDEFINE_H_F)) THEN
DO NS=1,N_TRACKED_SPECIES
IF (.NOT.REDEFINE_H_F(NS)) CYCLE
CPBAR_Z(0,NS) = CPBAR_Z(0,NS) + SPECIES_MIXTURE(NS)%H_F - H_F_OLD(NS)
DO J=1,5000
CPBAR_Z(J,NS) = CPBAR_Z(J,NS) + (SPECIES_MIXTURE(NS)%H_F - H_F_OLD(NS))/REAL(J,EB)
ENDDO
ENDDO
ELSE
IF (SIMPLE_CHEMISTRY .AND. CONSTANT_SPECIFIC_HEAT_RATIO) THEN
H_F_OLD = 0._EB
I_HFRT = INT(H_F_REFERENCE_TEMPERATURE)
D_HFRT = H_F_REFERENCE_TEMPERATURE - REAL(I_HFRT,EB)
RN => REACTION(1)
DO NS=1,N_TRACKED_SPECIES
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
        )))
ENDDO
H_F_OLD = H_F_OLD*H_F_REFERENCE_TEMPERATURE
D_H = -SUM(H_F_OLD)/SPECIES_MIXTURE(RN%FUEL_SMIX_INDEX)%MW-RN%HEAT_OF_COMBUSTION
CPBAR_Z(0,RN%FUEL_SMIX_INDEX) = CPBAR_Z(0,RN%FUEL_SMIX_INDEX) - D_H
DO J=1,5000
CPBAR_Z(J,RN%FUEL_SMIX_INDEX) = CPBAR_Z(J,RN%FUEL_SMIX_INDEX) -D_H/REAL(J,EB)
ENDDO
```

```fortran
ENDIF
ENDIF

END SUBROUTINE PROC_REAC_2


SUBROUTINE READ_PART

USE MATH_FUNCTIONS, ONLY : GET_RAMP_INDEX
USE DEVICE_VARIABLES, ONLY : PROPERTY_TYPE
USE RADCONS, ONLY : MIE_NDG
USE MATH_FUNCTIONS, ONLY: GET_TABLE_INDEX
INTEGER :: SAMPLING_FACTOR,N,NN,ILPC,IPC,RGB(3),N_STRATA,N_LAGRANGIAN_CLASSES_READ
REAL(EB) :: DIAMETER, GAMMA_D,AGE,INITIAL_TEMPERATURE,HEAT_OF_COMBUSTION, &
VERTICAL_VELOCITY,HORIZONTAL_VELOCITY,MAXIMUM_DIAMETER,MINIMUM_DIAMETER,SIGMA_D, &
SURFACE_TENSION,BREAKUP_RATIO,BREAKUP_GAMMA_D,BREAKUP_SIGMA_D,&
DENSE_VOLUME_FRACTION,REAL_REFRACTIVE_INDEX,COMPLEX_REFRACTIVE_INDEX,RUNNING_AVERAGE_FACTOR
REAL(EB) :: DRAG_COEFFICIENT(3),FREE_AREA_FRACTION,PERMEABILITY(3),POROUS_VOLUME_FRACTION
REAL(EB), DIMENSION(3,10) :: ORIENTATION
REAL(EB), DIMENSION(3) :: OR_TEMP
CHARACTER(LABEL_LENGTH) :: SPEC_ID,DEVC_ID,CTRL_ID,QUANTITIES(1:10),QUANTITIES_SPEC_ID(1:10),SURF_ID,DRAG_LAW,
    PROP_ID, &
RADIATIVE_PROPERTY_TABLE='null',CNF_RAMP_ID='null',BREAKUP_CNF_RAMP_ID='null',DISTRIBUTION,BREAKUP_DISTRIBUTION
CHARACTER(25) :: COLOR
LOGICAL :: TARGET_ONLY,MASSLESS,STATIC,MONODISPERSE,BREAKUP,CHECK_DISTRIBUTION,TURBULENT_DISPERSION,&
PERIODIC_X,PERIODIC_Y,PERIODIC_Z,SECOND_ORDER_PARTICLE_TRANSPORT
TYPE(LAGRANGIAN_PARTICLE_CLASS_TYPE), POINTER :: LPC=>NULL()
NAMELIST /PART/ AGE,BREAKUP,BREAKUP_CNF_RAMP_ID,BREAKUP_DISTRIBUTION,BREAKUP_GAMMA_D,BREAKUP_RATIO,&
BREAKUP_SIGMA_D,CHECK_DISTRIBUTION,CNF_RAMP_ID,COLOR,COMPLEX_REFRACTIVE_INDEX,&
CTRL_ID,DENSE_VOLUME_FRACTION,&
DEVC_ID,DIAMETER,DISTRIBUTION,DRAG_COEFFICIENT,DRAG_LAW,FREE_AREA_FRACTION,FYI,GAMMA_D,HEAT_OF_COMBUSTION,&
HORIZONTAL_VELOCITY,ID,INITIAL_TEMPERATURE,MASSLESS,MAXIMUM_DIAMETER,MINIMUM_DIAMETER,MONODISPERSE,&
N_STRATA,ORIENTATION,PERMEABILITY,PERIODIC_X,PERIODIC_Y,PERIODIC_Z,POROUS_VOLUME_FRACTION,PROP_ID,QUANTITIES,&
QUANTITIES_SPEC_ID,RADIATIVE_PROPERTY_TABLE,REAL_REFRACTIVE_INDEX,RGB,RUNNING_AVERAGE_FACTOR,&
SAMPLING_FACTOR,SECOND_ORDER_PARTICLE_TRANSPORT,SIGMA_D,SPEC_ID,STATIC,&
SURFACE_TENSION,SURF_ID,TARGET_ONLY,TURBULENT_DISPERSION,VERTICAL_VELOCITY

! Determine total number of PART lines in the input file

REWIND(LU_INPUT) ; INPUT_FILE_LINE_NUMBER = 0
N_LAGRANGIAN_CLASSES = 0
N_LAGRANGIAN_CLASSES_READ = 0

COUNT_PART_LOOP: DO
CALL CHECKREAD('PART',LU_INPUT,IOS)
IF (IOS==1) EXIT COUNT_PART_LOOP
READ(LU_INPUT,PART,END=219,ERR=220,IOSTAT=IOS)
N_LAGRANGIAN_CLASSES_READ = N_LAGRANGIAN_CLASSES_READ + 1
220 IF (IOS>0) THEN ; CALL SHUTDOWN('ERROR: Problem with PART line') ; RETURN ; ENDIF
ENDDO COUNT_PART_LOOP
219 REWIND(LU_INPUT) ; INPUT_FILE_LINE_NUMBER = 0

! Add reserved INIT lines to account for devices for 'RADIATIVE HEAT FLUX GAS' or 'ADIABATIC SURFACE TEMPERATURE
    GAS'

N_LAGRANGIAN_CLASSES = N_LAGRANGIAN_CLASSES_READ

IF (TARGET_PARTICLES_INCLUDED) N_LAGRANGIAN_CLASSES = N_LAGRANGIAN_CLASSES + 1

! Allocate the derived type array to hold information about the particle classes

IF (N_LAGRANGIAN_CLASSES>0) PARTICLE_FILE = .TRUE.
ALLOCATE(LAGRANGIAN_PARTICLE_CLASS(N_LAGRANGIAN_CLASSES),STAT=IZERO)
CALL ChkMemErr('READ','N_LAGRANGIAN_CLASSES',IZERO)

N_LP_ARRAY_INDICES = 0
IPC = 0
ILPC = 0

READ_PART_LOOP: DO N=1,N_LAGRANGIAN_CLASSES

! Read the PART line from the input file or set up special PARTICLE_CLASS class for water PARTICLEs or tracers

IF (N<=N_LAGRANGIAN_CLASSES_READ) THEN

CALL CHECKREAD('PART',LU_INPUT,IOS)
IF (IOS==1) EXIT READ_PART_LOOP
CALL SET_PART_DEFAULTS
READ(LU_INPUT,PART)

ELSE

! Create a class of particles that is just a target

CALL SET_PART_DEFAULTS
WRITE(ID,'(A)') 'RESERVED TARGET PARTICLE'
TARGET_ONLY = .TRUE.
STATIC = .TRUE.
```

```
ORIENTATION(1:3,1) = (/1._EB , 0._EB , 0._EB/)  ! This is just a dummy orientation

ENDIF

LPC => LAGRANGIAN_PARTICLE_CLASS(N)

! Identify the different types of Lagrangian particles, like massless tracers, targets, droplets, etc.

IF (SURF_ID/='null') THEN
SOLID_PARTICLES = .TRUE.
IF (CNF_RAMP_ID=='null') MONODISPERSE = .TRUE.
LPC%SOLID_PARTICLE = .TRUE.
IF (SAMPLING_FACTOR<=0) SAMPLING_FACTOR = 1
IF (DIAMETER>0._EB) THEN
WRITE(MESSAGE,'(A,I0,A)') 'ERROR: PART ',N,' cannot have both a specified DIAMETER and a SURF_ID.'
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF
ENDIF

IF (TARGET_ONLY) THEN
LPC%MASSLESS_TARGET = .TRUE.
SURF_ID  = 'MASSLESS TARGET'
SOLID_PARTICLES = .TRUE.
IF (CNF_RAMP_ID=='null') MONODISPERSE = .TRUE.
STATIC = .TRUE.
IF (SAMPLING_FACTOR<=0) SAMPLING_FACTOR = 1
ENDIF

IF (SPEC_ID/='null') THEN
SURF_ID = 'DROPLET'
LPC%LIQUID_DROPLET = .TRUE.
IF (SAMPLING_FACTOR<=0) SAMPLING_FACTOR = 10
IF (DIAMETER<=0._EB .AND. CNF_RAMP_ID=='null') THEN
WRITE(MESSAGE,'(A,I0,A)') 'ERROR: PART ',N,' requires a specified DIAMETER.'
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF
IF (MASSLESS) THEN
WRITE(MESSAGE,'(A)') 'ERROR: Cannot have MASSLESS=.TRUE. with evaporating PARTICLEs'
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF
ENDIF

IF (MASSLESS) THEN
LPC%MASSLESS_TRACER = .TRUE.
DIAMETER = 0._EB
SURF_ID  = 'MASSLESS TRACER'
IF (SAMPLING_FACTOR<=0) SAMPLING_FACTOR = 1
ENDIF

! If particle class has no ID at this point, stop.

IF (SURF_ID=='null') THEN
WRITE(MESSAGE,'(A,I0,A)') 'ERROR: PART ',N,' needs a SURF_ID.'
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF

! If particle class has no ID at this point, stop.

DO I=1,10
IF (QUANTITIES(I)=='MASS FLUX' .AND. QUANTITIES_SPEC_ID(I)=='null') THEN
WRITE(MESSAGE,'(A)') "ERROR: PART QUANTITIES 'MASS FLUX' requires QUANTITIES_SPEC_ID."
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF
ENDDO

! Set default colors for Smokeview. Water droplets are BLUE. Fuel droplets are YELLOW. Everything else is BLACK.

IF (TRIM(SPEC_ID)=='WATER VAPOR') THEN
IF (ANY(RGB<0) .AND. COLOR=='null') COLOR='BLUE'
ENDIF

IF (SIMPLE_CHEMISTRY) THEN
IF(TRIM(SPEC_ID)==TRIM(REACTION(1)%FUEL)) THEN
IF (ANY(RGB<0) .AND. COLOR=='null') COLOR='YELLOW'
ENDIF
ENDIF

IF (ANY(RGB<0) .AND. COLOR=='null') COLOR = 'BLACK'

IF (COLOR /= 'null') CALL COLOR2RGB(RGB,COLOR)

! Determine if the SPEC_ID is OK
LPC%SPEC_ID = SPEC_ID
IF (LPC%LIQUID_DROPLET) THEN
DO NN=1,N_TRACKED_SPECIES
IF (TRIM(SPECIES_MIXTURE(NN)%ID)==TRIM(LPC%SPEC_ID)) THEN
LPC%Z_INDEX = NN
SPECIES_MIXTURE(NN)%EVAPORATING = .TRUE.
```

```
EXIT
ENDIF
ENDDO
IF(LPC%Z_INDEX < 0) THEN
WRITE(MESSAGE,'(A,A,A)') 'ERROR: PART SPEC_ID ',TRIM(LPC%SPEC_ID),' not found'
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF
IF (SPECIES_MIXTURE(LPC%Z_INDEX)%SINGLE_SPEC_INDEX < 0) THEN
WRITE(MESSAGE,'(A,I0,A)') 'ERROR: PART line ',N,'. Particles cannot evaporate to a lumped species.'
CALL SHUTDOWN(MESSAGE) ; RETURN
ELSE
LPC%Y_INDEX = SPECIES_MIXTURE(LPC%Z_INDEX)%SINGLE_SPEC_INDEX
ENDIF
IF (SPECIES(LPC%Y_INDEX)%DENSITY_LIQUID > 0._EB) LPC%DENSITY=SPECIES(LPC%Y_INDEX)%DENSITY_LIQUID
ENDIF

! Arrays for particle size distribution

IF (MONODISPERSE) THEN
LPC%N_STRATA = 1
ELSE
LPC%N_STRATA = N_STRATA
ENDIF

IF (DIAMETER > 0._EB .OR. CNF_RAMP_ID/='null') THEN
ALLOCATE(LPC%CNF(0:NDC),STAT=IZERO)
CALL ChkMemErr('READ','CNF',IZERO)
ALLOCATE(LPC%CVF(0:NDC),STAT=IZERO)
CALL ChkMemErr('READ','CVF',IZERO)
ALLOCATE(LPC%R_CNF(0:NDC),STAT=IZERO)
CALL ChkMemErr('READ','R_CNF',IZERO)
ALLOCATE(LPC%IL_CNF(LPC%N_STRATA),STAT=IZERO)
CALL ChkMemErr('READ','IL_CNF',IZERO)
ALLOCATE(LPC%IU_CNF(LPC%N_STRATA),STAT=IZERO)
CALL ChkMemErr('READ','IU_CNF',IZERO)
ALLOCATE(LPC%W_CNF(LPC%N_STRATA),STAT=IZERO)
CALL ChkMemErr('READ','W_CNF',IZERO)
ENDIF

! Arrays related to particle break-up model

IF (BREAKUP) THEN
ALLOCATE(LPC%BREAKUP_CNF(0:NDC),STAT=IZERO)
CALL ChkMemErr('READ','BREAKUP_CNF',IZERO)
ALLOCATE(LPC%BREAKUP_R_CNF(0:NDC),STAT=IZERO)
CALL ChkMemErr('READ','BREAKUP_R_CNF',IZERO)
ALLOCATE(LPC%BREAKUP_CVF(0:NDC),STAT=IZERO)
CALL ChkMemErr('READ','BREAKUP_CVF',IZERO)
ENDIF

! Radiative property table

IF (RADIATIVE_PROPERTY_TABLE /= 'null') THEN
CALL GET_TABLE_INDEX(RADIATIVE_PROPERTY_TABLE,PART_RADIATIVE_PROPERTY,LPC%RADIATIVE_PROPERTY_INDEX)
LPC%RADIATIVE_PROPERTY_TABLE_ID = RADIATIVE_PROPERTY_TABLE
ELSE
LPC%RADIATIVE_PROPERTY_INDEX = 0
ENDIF

! Assign property data to LAGRANGIAN_PARTICLE_CLASS class

LPC%ID                          = ID
LPC%BREAKUP                     = BREAKUP
LPC%BREAKUP_RATIO               = BREAKUP_RATIO
LPC%BREAKUP_GAMMA               = BREAKUP_GAMMA_D
IF ( BREAKUP_SIGMA_D > 0._EB ) THEN
LPC%BREAKUP_SIGMA             = BREAKUP_SIGMA_D
ELSE
! per tech guide, sigma*gamma=1.15 smoothly joins Rosin-Rammler and lognormal distribustions
LPC%BREAKUP_SIGMA             = 1.15_EB/BREAKUP_GAMMA_D
ENDIF
LPC%CTRL_ID                     = CTRL_ID
LPC%DENSE_VOLUME_FRACTION       = DENSE_VOLUME_FRACTION
LPC%DEVC_ID                     = DEVC_ID
LPC%TMP_INITIAL                 = INITIAL_TEMPERATURE + TMPM
LPC%SAMPLING                    = SAMPLING_FACTOR
LPC%RGB                         = RGB
LPC%DIAMETER                    = DIAMETER*1.E-6_EB
LPC%MEAN_DROPLET_VOLUME         = FOTHPI*(0.5_EB*LPC%DIAMETER)**3 ! recomputed for distributions
LPC%MAXIMUM_DIAMETER            = MAXIMUM_DIAMETER*1.E-6_EB
IF (MINIMUM_DIAMETER<0._EB) THEN
MINIMUM_DIAMETER              = 0.005_EB*DIAMETER
ENDIF
LPC%MINIMUM_DIAMETER            = MINIMUM_DIAMETER*1.E-6_EB
LPC%KILL_RADIUS                 = MINIMUM_DIAMETER*0.5_EB*1.E-6_EB*0.171_EB ! 0.171 ???
LPC%MONODISPERSE                = MONODISPERSE
LPC%PERIODIC_X                  = PERIODIC_X
LPC%PERIODIC_Y                  = PERIODIC_Y
```

```
LPC%PERIODIC_Z                          = PERIODIC_Z
LPC%PROP_ID                             = PROP_ID
LPC%QUANTITIES                          = QUANTITIES
LPC%QUANTITIES_SPEC_ID                  = QUANTITIES_SPEC_ID
LPC%GAMMA                               = GAMMA_D
IF ( SIGMA_D > 0._EB ) THEN
LPC%SIGMA                            = SIGMA_D
ELSE
LPC%SIGMA                            = 1.15_EB/GAMMA_D
END IF
LPC%DISTRIBUTION                        = DISTRIBUTION
LPC%CHECK_DISTRIBUTION                  = CHECK_DISTRIBUTION
LPC%BREAKUP_DISTRIBUTION                = BREAKUP_DISTRIBUTION
LPC%CNF_RAMP_ID                         = CNF_RAMP_ID
LPC%BREAKUP_CNF_RAMP_ID                 = BREAKUP_CNF_RAMP_ID

IF(LPC%CNF_RAMP_ID/='null') THEN
CALL GET_RAMP_INDEX(LPC%CNF_RAMP_ID,'DIAMETER',LPC%CNF_RAMP_INDEX)
ENDIF
IF(LPC%BREAKUP_CNF_RAMP_ID/='null') THEN
CALL GET_RAMP_INDEX(LPC%BREAKUP_CNF_RAMP_ID,'DIAMETER',LPC%BREAKUP_CNF_RAMP_INDEX)
ENDIF

LPC%TMP_INITIAL          = INITIAL_TEMPERATURE + TMPM
LPC%REAL_REFRACTIVE_INDEX = REAL_REFRACTIVE_INDEX
LPC%COMPLEX_REFRACTIVE_INDEX = COMPLEX_REFRACTIVE_INDEX
IF (LPC%REAL_REFRACTIVE_INDEX <= 0._EB .OR. LPC%COMPLEX_REFRACTIVE_INDEX < 0._EB) THEN
WRITE(MESSAGE,'(A,A)') 'Bad refractive index on PART line ',LPC%ID
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF
LPC%HEAT_OF_COMBUSTION = HEAT_OF_COMBUSTION*1000._EB
LPC%FTPR               = FOTHPI*LPC%DENSITY
LPC%KILL_MASS          = LPC%FTPR*LPC%KILL_RADIUS**3
LPC%LIFETIME           = AGE
LPC%TURBULENT_DISPERSION = TURBULENT_DISPERSION
LPC%STATIC             = STATIC
LPC%SPEC_ID            = SPEC_ID
LPC%SURF_ID            = SURF_ID
LPC%SURF_INDEX         = -1
LPC%SURFACE_TENSION    = SURFACE_TENSION
LPC%ADJUST_EVAPORATION  = 1._EB   ! If H_O_C>0. this parameter will have to be reset later
LPC%VERTICAL_VELOCITY   = VERTICAL_VELOCITY
LPC%HORIZONTAL_VELOCITY = HORIZONTAL_VELOCITY
LPC%SECOND_ORDER_PARTICLE_TRANSPORT = SECOND_ORDER_PARTICLE_TRANSPORT
LPC%DRAG_COEFFICIENT    = DRAG_COEFFICIENT
IF (DRAG_COEFFICIENT(1)>=0._EB .AND. DRAG_LAW=='SPHERE') DRAG_LAW = 'USER'

! Count and process the number of orientations for the particle

LPC%N_ORIENTATION = 0

DO NN=1,10
IF (ANY(ABS(ORIENTATION(1:3,NN))>TWO_EPSILON_EB)) LPC%N_ORIENTATION = LPC%N_ORIENTATION + 1
ENDDO
IF (TRIM(DRAG_LAW)=='SCREEN' .AND. LPC%N_ORIENTATION/=1) THEN
WRITE(MESSAGE,'(A,I0,A)') 'ERROR: PART line ',N,'. Must specify exactly one ORIENTATION for SCREEN drag law.'
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF

IF (LPC%N_ORIENTATION>0) THEN
ALLOCATE(LPC%SOLID_ANGLE(1:LPC%N_ORIENTATION))
LPC%SOLID_ANGLE = 4._EB*PI
LPC%ORIENTATION_INDEX = N_ORIENTATION_VECTOR + 1
DO NN=1,LPC%N_ORIENTATION
OR_TEMP(1:3) = ORIENTATION(1:3,NN)
N_ORIENTATION_VECTOR = N_ORIENTATION_VECTOR + 1
IF (N_ORIENTATION_VECTOR>UBOUND(ORIENTATION_VECTOR,DIM=2)) THEN
ORIENTATION_VECTOR => REALLOCATE2D(ORIENTATION_VECTOR,1,3,1,N_ORIENTATION_VECTOR+10)
ENDIF
IF (ALL(ABS(OR_TEMP(1:3))<TWO_EPSILON_EB)) THEN
WRITE(MESSAGE,'(A,I0,A)') 'ERROR: PART line ',N,'. All components of ORIENTATION are zero.'
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF
ORIENTATION_VECTOR(1:3,N_ORIENTATION_VECTOR) = ORIENTATION(1:3,NN)/ NORM2(OR_TEMP)
ENDDO
ENDIF
LPC%FREE_AREA_FRACTION = FREE_AREA_FRACTION

SELECT CASE(DRAG_LAW)
CASE('SPHERE')
LPC%DRAG_LAW = SPHERE_DRAG
CASE('CYLINDER')
LPC%DRAG_LAW = CYLINDER_DRAG
CASE('USER')
LPC%DRAG_LAW = USER_DRAG
CASE('SCREEN')
LPC%DRAG_LAW = SCREEN_DRAG
LPC%PERMEABILITY(1:3) = 3.44E-9_EB*LPC%FREE_AREA_FRACTION**1.6_EB
```

```
LPC%DRAG_COEFFICIENT(1:3) = 4.30E-2_EB*LPC%FREE_AREA_FRACTION**2.13_EB
CASE('POROUS MEDIA')
IF (ANY(DRAG_COEFFICIENT<TWO_EPSILON_EB) .OR. ANY(PERMEABILITY<TWO_EPSILON_EB)) THEN
WRITE(MESSAGE,'(A,I0,A)') 'ERROR: PART line ',N,&
'. For POROUS MEDIA must specify all compoents for DRAG_COEFFICIENT and PERMIABILTIY.'
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF
LPC%DRAG_LAW = POROUS_DRAG
LPC%PERMEABILITY = PERMEABILITY
LPC%POROUS_VOLUME_FRACTION = POROUS_VOLUME_FRACTION
CASE DEFAULT
WRITE(MESSAGE,'(A)') 'ERROR: unrecognized drag law on PART line'
CALL SHUTDOWN(MESSAGE) ; RETURN
END SELECT

! Determine the number of slots to create in the particle evaporation and radiation arrays

IF (LPC%LIQUID_DROPLET .OR. LPC%SOLID_PARTICLE) THEN
N_LP_ARRAY_INDICES = N_LP_ARRAY_INDICES + 1
LPC%ARRAY_INDEX =  N_LP_ARRAY_INDICES
IF (LPC%SOLID_PARTICLE .AND. RUNNING_AVERAGE_FACTOR<0._EB) LPC%RUNNING_AVERAGE_FACTOR = 0.0_EB
IF (LPC%LIQUID_DROPLET .AND. RUNNING_AVERAGE_FACTOR<0._EB) LPC%RUNNING_AVERAGE_FACTOR = 0.5_EB
ENDIF

ENDDO READ_PART_LOOP

! Allocate radiation arrays

PLOOP2: DO ILPC=1,N_LAGRANGIAN_CLASSES
LPC=>LAGRANGIAN_PARTICLE_CLASS(ILPC)
IF (LPC%LIQUID_DROPLET) THEN
ALLOCATE(LPC%WQABS(0:MIE_NDG,1:NUMBER_SPECTRAL_BANDS))
CALL ChkMemErr('INIT','WQABS',IZERO)
LPC%WQABS = 0._EB
ALLOCATE(LPC%WQSCA(0:MIE_NDG,1:NUMBER_SPECTRAL_BANDS))
CALL ChkMemErr('INIT','WQSCA',IZERO)
LPC%WQSCA = 0._EB
ALLOCATE(LPC%R50(0:MIE_NDG))
CALL ChkMemErr('INIT','R50',IZERO)
LPC%R50 = 0._EB
ENDIF
ENDDO PLOOP2

! Determine output quantities

DO ILPC=1,N_LAGRANGIAN_CLASSES
LPC=>LAGRANGIAN_PARTICLE_CLASS(ILPC)
LPC%N_QUANTITIES = 0
IF (ANY(LPC%QUANTITIES/='null')) THEN
QUANTITIES_LOOP: DO N=1,10
IF (LPC%QUANTITIES(N)=='null') CYCLE QUANTITIES_LOOP
LPC%N_QUANTITIES = LPC%N_QUANTITIES + 1
CALL GET_QUANTITY_INDEX(LPC%SMOKEVIEW_LABEL(LPC%N_QUANTITIES),LPC%SMOKEVIEW_BAR_LABEL(LPC%N_QUANTITIES), &
LPC%QUANTITIES_INDEX(LPC%N_QUANTITIES),I_DUM(1), &
LPC%QUANTITIES_Y_INDEX(LPC%N_QUANTITIES),LPC%QUANTITIES_Z_INDEX(LPC%N_QUANTITIES),&
I_DUM(4),I_DUM(5),I_DUM(6),I_DUM(7),'PART', &
LPC%QUANTITIES(N),'null',LPC%QUANTITIES_SPEC_ID(N),'null','null','null','null')
ENDDO QUANTITIES_LOOP
ENDIF
ENDDO

CONTAINS


SUBROUTINE SET_PART_DEFAULTS

BREAKUP                    = .FALSE.
BREAKUP_RATIO              = 3._EB/7._EB  ! ratio of child Sauter mean to parent size in Bag breakup regime
BREAKUP_GAMMA_D            = 2.4_EB
BREAKUP_SIGMA_D            = -99999.9_EB
CTRL_ID                    = 'null'
DENSE_VOLUME_FRACTION      = 1.E-5_EB     ! Limiting volume fraction for drag reduction
DEVC_ID                    = 'null'
INITIAL_TEMPERATURE        = TMPA - TMPM  ! C
HEAT_OF_COMBUSTION         = -1._EB       ! kJ/kg
DIAMETER                   = -1._EB       !
MAXIMUM_DIAMETER           = 1.E9_EB      ! microns, meant to be infinitely large and not used
MINIMUM_DIAMETER           = -1._EB       ! microns, below which the PARTICLE evaporates in one time step
MONODISPERSE               = .FALSE.
N_STRATA                   = 6
GAMMA_D                    = 2.4_EB
SIGMA_D                    = -99999.9_EB
AGE                        = 1.E6_EB      ! s
ID                         = 'null'
PERIODIC_X                 = .FALSE.
PERIODIC_Y                 = .FALSE.
PERIODIC_Z                 = .FALSE.
PROP_ID                    = 'null'
```

```
ORIENTATION              = 0._EB
QUANTITIES               = 'null'
QUANTITIES_SPEC_ID       = 'null'
RADIATIVE_PROPERTY_TABLE = 'null'
RGB                      = -1
SPEC_ID                  = 'null'
SURF_ID                  = 'null'
SURFACE_TENSION          = 72.8E-3_EB  ! N/m, applies for water
COLOR                    = 'null'
SAMPLING_FACTOR          = -1
STATIC                   = .FALSE.
MASSLESS                 = .FALSE.
TARGET_ONLY              = .FALSE.
TURBULENT_DISPERSION     = .FALSE.
REAL_REFRACTIVE_INDEX    = 1.33_EB
RUNNING_AVERAGE_FACTOR   = -1._EB
COMPLEX_REFRACTIVE_INDEX = 0.01_EB
VERTICAL_VELOCITY        = 0.5_EB
HORIZONTAL_VELOCITY      = 0.2_EB
DRAG_LAW                 = 'SPHERE'
DRAG_COEFFICIENT         = -1._EB
PERMEABILITY             = -1._EB
DISTRIBUTION             = 'ROSIN-RAMMLER-LOGNORMAL'
CNF_RAMP_ID              = 'null'
CHECK_DISTRIBUTION       = .FALSE.
BREAKUP_DISTRIBUTION     = 'ROSIN-RAMMLER-LOGNORMAL'
BREAKUP_CNF_RAMP_ID      = 'null'
FREE_AREA_FRACTION       = 0.5_EB
SECOND_ORDER_PARTICLE_TRANSPORT = .FALSE.

END SUBROUTINE SET_PART_DEFAULTS

END SUBROUTINE READ_PART


SUBROUTINE PROC_PART

USE PROPERTY_DATA, ONLY: JANAF_TABLE_LIQUID
USE MATH_FUNCTIONS, ONLY: EVALUATE_RAMP
INTEGER :: N,NN,J,ITMP,I_MELT,I_BOIL
REAL(EB) :: H_L,H_V,CPBAR,H_G_S,H_G_S_REF,H_L_REF,TMP_REF,TMP_MELT,TMP_V,TMP_WGT,DENSITY,MASS,VOLUME,R_O,R_I,&
MU_LIQUID,K_LIQUID,BETA_LIQUID
TYPE(LAGRANGIAN_PARTICLE_CLASS_TYPE), POINTER :: LPC=>NULL()
TYPE(SPECIES_TYPE),POINTER:: SS=>NULL()
TYPE(SURFACE_TYPE),POINTER:: SF=>NULL()

IF (N_LAGRANGIAN_CLASSES == 0) RETURN

PART_LOOP: DO N=1,N_LAGRANGIAN_CLASSES

LPC => LAGRANGIAN_PARTICLE_CLASS(N)
SF  => SURFACE(LPC%SURF_INDEX)

! Assign device or controller

CALL SEARCH_CONTROLLER('PART',LPC%CTRL_ID,LPC%DEVC_ID,LPC%DEVC_INDEX,LPC%CTRL_INDEX,N)

! Get density if the particles are liquid droplets or have mass

IF (LPC%LIQUID_DROPLET) THEN
CALL JANAF_TABLE_LIQUID (1,CPBAR,H_V,H_L,TMP_REF,TMP_MELT,TMP_V,SPECIES(LPC%Y_INDEX)%ID,LPC%FUEL,DENSITY,&
MU_LIQUID,K_LIQUID,BETA_LIQUID)
IF (LPC%DENSITY < 0._EB) THEN
LPC%DENSITY = DENSITY
LPC%FTPR = FOTH*PI*DENSITY
LPC%KILL_MASS = LPC%FTPR*LPC%KILL_RADIUS**3
IF (SPECIES(LPC%Y_INDEX)%DENSITY_LIQUID<0._EB) SPECIES(LPC%Y_INDEX)%DENSITY_LIQUID = DENSITY
ENDIF
IF (SPECIES(LPC%Y_INDEX)%MU_LIQUID < 0._EB) SPECIES(LPC%Y_INDEX)%MU_LIQUID = MU_LIQUID
IF (SPECIES(LPC%Y_INDEX)%K_LIQUID < 0._EB) SPECIES(LPC%Y_INDEX)%K_LIQUID = K_LIQUID
IF (SPECIES(LPC%Y_INDEX)%BETA_LIQUID < 0._EB) SPECIES(LPC%Y_INDEX)%BETA_LIQUID = BETA_LIQUID
ENDIF

IF (SF%THERMALLY_THICK) THEN
MASS = 0._EB
VOLUME = 0._EB
DO NN=1,SF%N_CELLS_INI
SELECT CASE (SF%GEOMETRY)
CASE (SURF_CARTESIAN)
MASS = MASS + (SF%X_S(NN)-SF%X_S(NN-1))*SUM(SF%RHO_0(NN,1:SF%N_MATL))
VOLUME = VOLUME + (SF%X_S(NN)-SF%X_S(NN-1))
CASE (SURF_CYLINDRICAL)
R_I = SF%INNER_RADIUS + SF%THICKNESS - SF%X_S(NN)
R_O = SF%INNER_RADIUS + SF%THICKNESS - SF%X_S(NN-1)
MASS    = MASS    + (R_O**2-R_I**2)*SUM(SF%RHO_0(NN,1:SF%N_MATL))
VOLUME = VOLUME + (R_O**2-R_I**2)
CASE (SURF_SPHERICAL)
R_I = SF%INNER_RADIUS + SF%THICKNESS - SF%X_S(NN)
```

```
R_O = SF%INNER_RADIUS + SF%THICKNESS - SF%X_S(NN-1)
MASS   = MASS   + (R_O**3-R_I**3)*SUM(SF%RHO_0(NN,1:SF%N_MATL))
VOLUME = VOLUME + (R_O**3-R_I**3)
END SELECT
ENDDO
LPC%DENSITY = MASS/VOLUME
LPC%FTPR = FOTH*PI*LPC%DENSITY
ENDIF

IF (SF%HEAT_TRANSFER_MODEL>0) THEN ; CALL SHUTDOWN('ERROR: HEAT_TRANSFER_MODEL not appropriate for PART') ;
      RETURN ; ENDIF

! Set the flag to do particle exchanges between meshes

OMESH_PARTICLES=.TRUE.

! Only process DROPLETs

SURF_OR_SPEC: IF (LPC%SURF_INDEX==DROPLET_SURF_INDEX) THEN

SS => SPECIES(LPC%Y_INDEX)

IZERO = 0
IF (.NOT.ALLOCATED(SS%C_P_L)) ALLOCATE(SS%C_P_L(0:5000),STAT=IZERO)
CALL ChkMemErr('PROC_PART','SS%C_P_L',IZERO)
SS%C_P_L=SS%SPECIFIC_HEAT_LIQUID
IF (.NOT.ALLOCATED(SS%C_P_L_BAR)) ALLOCATE(SS%C_P_L_BAR(0:5000),STAT=IZERO)
CALL ChkMemErr('PROC_PART','SS%C_P_L_BAR',IZERO)
IF (.NOT.ALLOCATED(SS%H_L)) ALLOCATE(SS%H_L(0:5000),STAT=IZERO)
CALL ChkMemErr('PROC_PART','SS%H_L',IZERO)
IF (.NOT.ALLOCATED(SS%H_V)) ALLOCATE(SS%H_V(0:5000),STAT=IZERO)
CALL ChkMemErr('PROC_PART','SS%H_V',IZERO)

TMP_REF = -1._EB
TMP_MELT = -1._EB
TMP_V = -1._EB
DO J = 1, 5000
IF (SS%C_P_L(J) > 0._EB) THEN
SS%H_L(J) = (REAL(J,EB)-SS%TMP_MELT)*SS%C_P_L(J)
IF (J==1) THEN
CALL JANAF_TABLE_LIQUID (J,CPBAR,H_V,H_L,TMP_REF,TMP_MELT,TMP_V,SS%PROP_ID,LPC%FUEL,DENSITY,&
MU_LIQUID,K_LIQUID,BETA_LIQUID)
IF (SS%H_V_REFERENCE_TEMPERATURE < 0._EB) SS%H_V_REFERENCE_TEMPERATURE=TMP_REF
IF (SS%TMP_V < 0._EB) SS%TMP_V = TMP_V
IF (SS%TMP_V < 0._EB) THEN
WRITE(MESSAGE,'(A,A,A)') 'ERROR: SPEC ',TRIM(SS%ID),' requires a VAPORIZATION_TEMPERATURE'
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF
IF (SS%TMP_MELT < 0._EB) SS%TMP_MELT = TMP_MELT
IF (SS%TMP_MELT < 0._EB) THEN
WRITE(MESSAGE,'(A,A,A)') 'ERROR: PARTicle class ',TRIM(SS%ID),' requires a TMP_MELT'
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF
IF (LPC%DENSITY < 0._EB) LPC%DENSITY = DENSITY
IF (LPC%DENSITY < 0._EB) THEN
WRITE(MESSAGE,'(A,A,A)') 'ERROR: PARTicle class ',TRIM(SS%ID),' requires a density'
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF
LPC%FTPR = FOTH*PI*LPC%DENSITY
ENDIF
ELSE
CALL JANAF_TABLE_LIQUID (J,SS%C_P_L(J),H_V,H_L,TMP_REF,TMP_MELT,TMP_V,SS%ID,LPC%FUEL,DENSITY,&
MU_LIQUID,K_LIQUID,BETA_LIQUID)
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
IF (J==1) THEN
IF (SS%H_V_REFERENCE_TEMPERATURE < 0._EB) SS%H_V_REFERENCE_TEMPERATURE=TMP_REF
IF (SS%TMP_V < 0._EB) SS%TMP_V = TMP_V
IF (SS%TMP_V < 0._EB) THEN
WRITE(MESSAGE,'(A,A,A)') 'ERROR: PARTicle class ',TRIM(SS%ID),' requires a TMP_V'
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF
IF (SS%TMP_MELT < 0._EB) SS%TMP_MELT = TMP_MELT
IF (SS%TMP_MELT < 0._EB) THEN
WRITE(MESSAGE,'(A,A,A)') 'ERROR: PARTicle class ',TRIM(SS%ID),' requires a TMP_MELT'
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF
IF (LPC%DENSITY < 0._EB) LPC%DENSITY = DENSITY
IF (LPC%DENSITY < 0._EB) THEN
WRITE(MESSAGE,'(A,A,A)') 'ERROR: PARTicle class ',TRIM(SS%ID),' requires a density'
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF
LPC%FTPR = FOTH*PI*LPC%DENSITY
IF (SS%C_P_L(J) < 0._EB) THEN
WRITE(MESSAGE,'(A,A,A)') 'ERROR: PARTicle class ',TRIM(SS%ID),' requires CP, H_V, TMP_MELT, TMP_V, and T_REF'
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF
SS%H_L(J) = H_L + SS%C_P_L(J)
ELSE
```

```
SS%H_L(J) = SS%H_L(J-1) + 0.5_EB*(SS%C_P_L(J)+SS%C_P_L(J-1))
ENDIF
ENDIF
END DO

SS%C_P_L(0) = SS%C_P_L(1)
SS%H_L(0) = SS%H_L(1) - SS%C_P_L(1)

! Adjust liquid H_L to force H_V at H_V_REFERENCE_TEMPERATURE

IF(SS%HEAT_OF_VAPORIZATION > 0._EB) H_V = SS%HEAT_OF_VAPORIZATION
ITMP = INT(SS%H_V_REFERENCE_TEMPERATURE)
TMP_WGT  = SS%H_V_REFERENCE_TEMPERATURE - REAL(ITMP,EB)
H_L_REF = SS%H_L(ITMP)+TMP_WGT*(SS%H_L(ITMP+1)-SS%H_L(ITMP))
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)))*&
SS%H_V_REFERENCE_TEMPERATURE
SS%H_L = SS%H_L + (H_G_S_REF - H_L_REF) - H_V
I_MELT = INT(SS%TMP_MELT) - 1
I_BOIL = INT(SS%TMP_V) + 1

! Determine the properties of the PARTICLE

DO J=1,5000
H_G_S = CPBAR_Z(J,LPC%Z_INDEX)*REAL(J,EB)
SS%H_V(J) = H_G_S - SS%H_L(J)
IF (SS%H_V(J) < 0._EB .AND. J > I_MELT .AND. J < I_BOIL) THEN
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'
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF
IF (J==1) THEN
SS%C_P_L_BAR(J) =  SS%H_L(J)
ELSE
SS%C_P_L_BAR(J) = SS%H_L(J) / REAL(J,EB)
ENDIF
ENDDO

SS%H_V(0) = SS%H_V(1)
SS%C_P_L_BAR(0) = SS%H_L(1)
TMPMIN = MIN(TMPMIN,SS%TMP_MELT)

SS%PR_LIQUID = SS%MU_LIQUID*SS%C_P_L(NINT(TMPA))/SS%K_LIQUID

ENDIF SURF_OR_SPEC

! Adjust the evaporation rate of fuel PARTICLEs to account for difference in HoC.

IF (LPC%HEAT_OF_COMBUSTION > 0._EB) LPC%ADJUST_EVAPORATION = LPC%HEAT_OF_COMBUSTION/REACTION(1)%
    HEAT_OF_COMBUSTION

ENDDO PART_LOOP

END SUBROUTINE PROC_PART


SUBROUTINE READ_PROP

USE DEVICE_VARIABLES
USE MATH_FUNCTIONS, ONLY : GET_RAMP_INDEX,GET_TABLE_INDEX
USE PHYSICAL_FUNCTIONS, ONLY : SPRAY_ANGLE_DISTRIBUTION
REAL(EB) :: ACTIVATION_OBSCURATION,ACTIVATION_TEMPERATURE,ALPHA_C,ALPHA_E,BETA_C,BETA_E, &
BEAD_DIAMETER,BEAD_EMISSIVITY,BEAD_SPECIFIC_HEAT,BEAD_DENSITY, &
BEAD_HEAT_TRANSFER_COEFFICIENT,HEAT_TRANSFER_COEFFICIENT,DIAMETER,DENSITY,SPECIFIC_HEAT, &
C_FACTOR,CHARACTERISTIC_VELOCITY,ORIFICE_DIAMETER,DROPLET_VELOCITY,EMISSIVITY, &
PARTICLE_VELOCITY,FLOW_RATE,FLOW_TAU,GAUGE_EMISSIVITY,GAUGE_TEMPERATURE,INITIAL_TEMPERATURE,K_FACTOR,&
LENGTH,SPRAY_ANGLE(2,2),OFFSET,OPERATING_PRESSURE,RTI,PDPA_START,PDPA_END,PDPA_RADIUS,MASS_FLOW_RATE,&
SPRAY_PATTERN_MU,SPRAY_PATTERN_BETA, &
PDPA_HISTOGRAM_LIMITS(2), &
P0,PX(3),PXX(3,3)
EQUIVALENCE(PARTICLE_VELOCITY,DROPLET_VELOCITY)
EQUIVALENCE(EMISSIVITY,BEAD_EMISSIVITY)
EQUIVALENCE(HEAT_TRANSFER_COEFFICIENT,BEAD_HEAT_TRANSFER_COEFFICIENT)
EQUIVALENCE(DENSITY,BEAD_DENSITY)
EQUIVALENCE(DIAMETER,BEAD_DIAMETER)
EQUIVALENCE(SPECIFIC_HEAT,BEAD_SPECIFIC_HEAT)
INTEGER ::I,N,NN,PDPA_M,PDPA_N,PARTICLES_PER_SECOND,VELOCITY_COMPONENT,PDPA_HISTOGRAM_NBINS,FED_ACTIVITY
LOGICAL :: PDPA_INTEGRATE,PDPA_NORMALIZE,PDPA_HISTOGRAM,PDPA_HISTOGRAM_CUMULATIVE
EQUIVALENCE(LENGTH,ALPHA_C)
CHARACTER(LABEL_LENGTH) :: SMOKEVIEW_ID(SMOKEVIEW_OBJECTS_DIMENSION),QUANTITY='null',PART_ID='null',FLOW_RAMP='
    null', &
SPRAY_PATTERN_TABLE='null',SPEC_ID='null',&
PRESSURE_RAMP='null',SMOKEVIEW_PARAMETERS(SMOKEVIEW_OBJECTS_DIMENSION), &
SPRAY_PATTERN_SHAPE='GAUSSIAN'
TYPE (PROPERTY_TYPE), POINTER :: PY=>NULL()

NAMELIST /PROP/ ACTIVATION_OBSCURATION,ACTIVATION_TEMPERATURE,ALPHA_C,ALPHA_E,BETA_C,BETA_E,FED_ACTIVITY,&
CHARACTERISTIC_VELOCITY,C_FACTOR,DENSITY,DIAMETER,EMISSIVITY,FLOW_RAMP,FLOW_RATE,FLOW_TAU,&
GAUGE_EMISSIVITY,GAUGE_TEMPERATURE,HEAT_TRANSFER_COEFFICIENT,ID,&
```

```
INITIAL_TEMPERATURE,K_FACTOR,LENGTH,MASS_FLOW_RATE,OFFSET,OPERATING_PRESSURE,ORIFICE_DIAMETER,P0,&
PARTICLES_PER_SECOND,&
PARTICLE_VELOCITY,PART_ID,PDPA_END,PDPA_HISTOGRAM, PDPA_HISTOGRAM_LIMITS,PDPA_HISTOGRAM_NBINS,&
PDPA_HISTOGRAM_CUMULATIVE,PDPA_INTEGRATE,PDPA_M,PDPA_N,PDPA_NORMALIZE,PDPA_RADIUS,&
PDPA_START,PRESSURE_RAMP,PX,PXX,QUANTITY,RTI,SMOKEVIEW_ID,SMOKEVIEW_PARAMETERS,SPEC_ID,SPECIFIC_HEAT,SPRAY_ANGLE,&
SPRAY_PATTERN_BETA,SPRAY_PATTERN_MU,SPRAY_PATTERN_SHAPE,SPRAY_PATTERN_TABLE,VELOCITY_COMPONENT,&
BEAD_EMISSIVITY,BEAD_HEAT_TRANSFER_COEFFICIENT,DROPLET_VELOCITY,& ! Backward compatability
BEAD_DENSITY,BEAD_DIAMETER,BEAD_SPECIFIC_HEAT                    ! Backward compatability

! Count the PROP lines in the input file. Note how many of these are cables.

N_PROP=0
REWIND(LU_INPUT) ; INPUT_FILE_LINE_NUMBER = 0
COUNT_PROP_LOOP: DO
CALL CHECKREAD('PROP',LU_INPUT,IOS)
IF (IOS==1) EXIT COUNT_PROP_LOOP
READ(LU_INPUT,PROP,ERR=34,IOSTAT=IOS)
N_PROP = N_PROP + 1
34 IF (IOS>0) THEN
WRITE(MESSAGE,'(A,I0,A,I0)') 'ERROR: Problem with PROP number', N_PROP+1,', line number',INPUT_FILE_LINE_NUMBER
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF
ENDDO COUNT_PROP_LOOP

! Allocate the PROPERTY derived types

ALLOCATE(PROPERTY(0:N_PROP),STAT=IZERO)
CALL ChkMemErr('READ','PROPERTY',IZERO)

! Read the PROP lines in the order listed in the input file

REWIND(LU_INPUT) ; INPUT_FILE_LINE_NUMBER = 0

READ_PROP_LOOP: DO N=0,N_PROP

CALL CHECKREAD('PROP',LU_INPUT,IOS)  ! Look for PROP lines in the input file
CALL SET_PROP_DEFAULTS           ! Reset PROP NAMELIST parameters to default values
IF (N > 0) READ(LU_INPUT,PROP)

! Pack PROP parameters into the appropriate property derived types

PY => PROPERTY(N)
PY%ACTIVATION_OBSCURATION   = ACTIVATION_OBSCURATION
PY%ACTIVATION_TEMPERATURE   = ACTIVATION_TEMPERATURE   ! NOTE: Act_Temp remains in degrees C. It is just a SETPOINT.
PY%ALPHA_C                  = ALPHA_C
PY%ALPHA_E                  = ALPHA_E
PY%BETA_C                   = BETA_C
PY%BETA_E                   = BETA_E
PY%DENSITY                  = DENSITY
PY%DIAMETER                 = DIAMETER
PY%EMISSIVITY               = EMISSIVITY
PY%HEAT_TRANSFER_COEFFICIENT= HEAT_TRANSFER_COEFFICIENT
PY%SPECIFIC_HEAT            = SPECIFIC_HEAT*1000._EB/TIME_SHRINK_FACTOR
PY%C_FACTOR                 = C_FACTOR
PY%CHARACTERISTIC_VELOCITY  = CHARACTERISTIC_VELOCITY
PY%GAUGE_EMISSIVITY         = GAUGE_EMISSIVITY
PY%GAUGE_TEMPERATURE        = GAUGE_TEMPERATURE + TMPM
PY%ID                       = ID
PY%INITIAL_TEMPERATURE      = INITIAL_TEMPERATURE + TMPM
PY%PARTICLES_PER_SECOND     = PARTICLES_PER_SECOND
PY%OFFSET                   = OFFSET
PY%OPERATING_PRESSURE       = OPERATING_PRESSURE
PY%PART_ID                  = PART_ID
PY%QUANTITY                 = QUANTITY
IF (PY%PART_ID/='null' .AND. PY%QUANTITY == 'null' ) PY%QUANTITY = 'NOZZLE'
PY%RTI                      = RTI
IF (SMOKEVIEW_ID(1)/='null') THEN
PY%SMOKEVIEW_ID           = SMOKEVIEW_ID
PY%N_SMOKEVIEW_IDS = 0
DO NN=1,SMOKEVIEW_OBJECTS_DIMENSION
IF (SMOKEVIEW_ID(NN)/='null') PY%N_SMOKEVIEW_IDS = PY%N_SMOKEVIEW_IDS + 1
ENDDO
ELSE
PY%N_SMOKEVIEW_IDS = 1
SELECT CASE(PY%QUANTITY)
CASE DEFAULT
PY%SMOKEVIEW_ID(1) = 'sensor'
CASE('SPRINKLER LINK TEMPERATURE')
PY%SMOKEVIEW_ID(1) = 'sprinkler_pendent'
CASE('NOZZLE')
PY%SMOKEVIEW_ID(1) = 'nozzle'
CASE('LINK TEMPERATURE')
PY%SMOKEVIEW_ID(1) = 'heat_detector'
CASE('spot obscuration','CHAMBER OBSCURATION')
PY%SMOKEVIEW_ID(1) = 'smoke_detector'
CASE('THERMOCOUPLE')
```

```
PY%SMOKEVIEW_ID(1) = 'thermocouple'
END SELECT
ENDIF
PY%SMOKEVIEW_PARAMETERS = SMOKEVIEW_PARAMETERS
PY%N_SMOKEVIEW_PARAMETERS = 0
DO I=1,SMOKEVIEW_OBJECTS_DIMENSION
IF (PY%SMOKEVIEW_PARAMETERS(I)/='null') PY%N_SMOKEVIEW_PARAMETERS = PY%N_SMOKEVIEW_PARAMETERS + 1
ENDDO
PY%SPEC_ID              = SPEC_ID
IF (PART_ID/='null' .AND. SPRAY_PATTERN_TABLE /= 'null') THEN
CALL GET_TABLE_INDEX(SPRAY_PATTERN_TABLE,SPRAY_PATTERN,PY%SPRAY_PATTERN_INDEX)
PY%TABLE_ID = SPRAY_PATTERN_TABLE
ELSE
PY%SPRAY_PATTERN_INDEX = 0
ENDIF
PY%SPRAY_ANGLE = SPRAY_ANGLE*PI/180._EB
IF(ANY(PY%SPRAY_ANGLE(1:2,2)<0)) PY%SPRAY_ANGLE(1:2,2)=PY%SPRAY_ANGLE(1:2,1)
SPRAY_PATTERN_MU=SPRAY_PATTERN_MU*PI/180._EB
IF (PART_ID/='null' .AND. SPRAY_PATTERN_TABLE == 'null' ) THEN
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))
IF(SPRAY_PATTERN_MU<0._EB) THEN
IF(SPRAY_ANGLE(1,1)>0._EB) THEN
SPRAY_PATTERN_MU=0.5_EB*SUM(PY%SPRAY_ANGLE(1:2,1))
ELSE
SPRAY_PATTERN_MU=0._EB
ENDIF
ENDIF
CALL SPRAY_ANGLE_DISTRIBUTION(PY%SPRAY_LON,PY%SPRAY_LAT,PY%SPRAY_LON_CDF,PY%SPRAY_LAT_CDF, &
SPRAY_PATTERN_BETA,SPRAY_PATTERN_MU,PY%SPRAY_ANGLE &
,SPRAY_PATTERN_SHAPE,NDC2)
ENDIF

! PDPA model

PY%PDPA_START          = PDPA_START
PY%PDPA_END            = PDPA_END
PY%PDPA_RADIUS         = PDPA_RADIUS
PY%PDPA_M              = PDPA_M
PY%PDPA_N              = PDPA_N
PY%PDPA_INTEGRATE      = PDPA_INTEGRATE
PY%PDPA_NORMALIZE      = PDPA_NORMALIZE
IF (TRIM(PY%QUANTITY) == 'NUMBER CONCENTRATION') THEN
PY%PDPA_M          = 0
PY%PDPA_N          = 0
ENDIF
IF ((TRIM(PY%QUANTITY) == 'MASS CONCENTRATION') .OR. &
(TRIM(PY%QUANTITY) == 'ENTHALPY')           .OR. &
(TRIM(PY%QUANTITY) == 'PARTICLE FLUX X')    .OR. &
(TRIM(PY%QUANTITY) == 'PARTICLE FLUX Y')    .OR. &
(TRIM(PY%QUANTITY) == 'PARTICLE FLUX Z')) THEN
PY%PDPA_M          = 3
PY%PDPA_N          = 0
ENDIF

! Histograms of PDPA data
PY%PDPA_HISTOGRAM              = PDPA_HISTOGRAM
PY%PDPA_HISTOGRAM_NBINS        = PDPA_HISTOGRAM_NBINS
PY%PDPA_HISTOGRAM_LIMITS       = PDPA_HISTOGRAM_LIMITS
PY%PDPA_HISTOGRAM_CUMULATIVE   = PDPA_HISTOGRAM_CUMULATIVE
IF(PDPA_HISTOGRAM) THEN
IF(PDPA_HISTOGRAM_NBINS<2) THEN
WRITE(MESSAGE,'(A,A,A)') 'ERROR: Problem with PROP ',TRIM(PY%ID),', PDPA_HISTOGRAM needs PDPA_NBINS>2'
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF
IF(ABS(PDPA_HISTOGRAM_LIMITS(1)-PDPA_HISTOGRAM_LIMITS(2)) < TWO_EPSILON_EB) THEN
WRITE(MESSAGE,'(A,A,A)') 'ERROR: Problem with PROP ',TRIM(PY%ID),', PDPA_HISTOGRAM needs PDPA_HISTOGRAM_LIMITS'
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF

ENDIF

PY%FED_ACTIVITY = FED_ACTIVITY
IF(FED_ACTIVITY < 1 .OR. FED_ACTIVITY > 3) THEN
WRITE(MESSAGE,'(A,A,A,I0)') 'ERROR: Problem with PROP ',TRIM(PY%ID),', FED_ACTIVITY out of range: ',FED_ACTIVITY
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF

PATCH_VELOCITY_IF: IF (VELOCITY_COMPONENT>0) THEN
IF(VELOCITY_COMPONENT > 3) THEN
WRITE(MESSAGE,'(A,A,A,I0)') 'ERROR: Problem with PROP ',TRIM(PY%ID),', VELOCITY_COMPONENT > 3: ',
    VELOCITY_COMPONENT
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF
IF(P0<-1.E9_EB) THEN
WRITE(MESSAGE,'(A,A,A)') 'ERROR: Problem with PROP ',TRIM(PY%ID),', VELOCITY_PATCH requires P0'
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF
```

```
PY%I_VEL = VELOCITY_COMPONENT
PY%P0 = P0  ! value at origin of Taylor expansion
DO J=1,3
PY%PX(J) = PX(J)  ! first derivative of P evaluated at origin
DO I=1,3
IF (I>J) PXX(I,J)=PXX(J,I) ! make symmetric
PY%PXX(I,J) = PXX(I,J) ! second derivative of P evaluated at origin
ENDDO
ENDDO
ENDIF PATCH_VELOCITY_IF

! Set flow variables
PY%MASS_FLOW_RATE =MASS_FLOW_RATE
PY%FLOW_RATE      =FLOW_RATE

IF (PART_ID/='null' .AND. PRESSURE_RAMP /= 'null') THEN
CALL GET_RAMP_INDEX(PRESSURE_RAMP,'PRESSURE',PY%PRESSURE_RAMP_INDEX)
ELSE
PY%PRESSURE_RAMP_INDEX = 0
ENDIF

! Check sufficient input

IF (PY%PRESSURE_RAMP_INDEX == 0 .AND. FLOW_RATE > 0._EB) THEN
IF (K_FACTOR < 0._EB) K_FACTOR = 10.0_EB
ENDIF

IF (PART_ID /='null' .AND. ABS(PDPA_RADIUS) <= TWO_EPSILON_EB) THEN
IF (MASS_FLOW_RATE > 0._EB) THEN
PY%MASS_FLOW_RATE = MASS_FLOW_RATE
IF (ABS(PARTICLE_VELOCITY) <= TWO_EPSILON_EB) THEN
WRITE(MESSAGE,'(A,A,A)') 'ERROR: Problem with PROP ',TRIM(PY%ID),', must specify PARTICLE_VELOCITY with
    MASS_FLOW_RATE'
CALL SHUTDOWN(MESSAGE) ; RETURN
ELSE
PY%PARTICLE_VELOCITY  = PARTICLE_VELOCITY
ENDIF
ELSE
IF ((FLOW_RATE>0._EB .AND. K_FACTOR<=0._EB .AND. OPERATING_PRESSURE<=0._EB) .OR. &
(FLOW_RATE<0._EB .AND. K_FACTOR>=0._EB .AND. OPERATING_PRESSURE<=0._EB) .OR. &
(FLOW_RATE<0._EB .AND. K_FACTOR<=0._EB .AND. OPERATING_PRESSURE>0._EB)) THEN
WRITE(MESSAGE,'(A,A,A)') 'ERROR: Problem with PROP ',TRIM(PY%ID),', too few flow parameters'
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF
IF (K_FACTOR < 0._EB .AND. OPERATING_PRESSURE > 0._EB)  K_FACTOR           = FLOW_RATE/SQRT(OPERATING_PRESSURE)
IF (FLOW_RATE < 0._EB .AND. OPERATING_PRESSURE > 0._EB) FLOW_RATE          = K_FACTOR*SQRT(OPERATING_PRESSURE)
IF (OPERATING_PRESSURE < 0._EB .AND. K_FACTOR > 0._EB)  OPERATING_PRESSURE = (FLOW_RATE/K_FACTOR)**2
PY%K_FACTOR           = K_FACTOR
PY%FLOW_RATE          = FLOW_RATE
PY%OPERATING_PRESSURE = OPERATING_PRESSURE

IF (PARTICLE_VELOCITY<=TWO_EPSILON_EB .AND. ORIFICE_DIAMETER<=TWO_EPSILON_EB .AND. &
PRESSURE_RAMP=='null' .AND. SPRAY_PATTERN_TABLE=='null') THEN
WRITE(MESSAGE,'(A,A,A)') 'WARNING: PROP ',TRIM(PY%ID),' PARTICLE velocity is not defined.'
IF (MYID==0) WRITE(LU_ERR,'(A)') TRIM(MESSAGE)
ENDIF

IF (PARTICLE_VELOCITY > 0._EB) THEN
PY%PARTICLE_VELOCITY  = PARTICLE_VELOCITY
ELSEIF ((ORIFICE_DIAMETER > 0._EB) .AND. (FLOW_RATE > 0._EB)) THEN
PY%PARTICLE_VELOCITY  = (FLOW_RATE/60._EB/1000._EB)/(PI*(ORIFICE_DIAMETER/2._EB)**2)
ENDIF
ENDIF
ENDIF
IF (FLOW_RAMP /= 'null') THEN
CALL GET_RAMP_INDEX(FLOW_RAMP,'TIME',PY%FLOW_RAMP_INDEX)
ELSE
PY%FLOW_RAMP_INDEX = 0
ENDIF
IF (ABS(FLOW_TAU) > TWO_EPSILON_EB) THEN
PY%FLOW_TAU = FLOW_TAU/TIME_SHRINK_FACTOR
IF (FLOW_TAU > 0._EB) PY%FLOW_RAMP_INDEX = TANH_RAMP
IF (FLOW_TAU < 0._EB) PY%FLOW_RAMP_INDEX = TSQR_RAMP
ENDIF

! Check for SPEC_ID

IF (PY%SPEC_ID/='null') THEN
CALL GET_SPEC_OR_SMIX_INDEX(PY%SPEC_ID,PY%Y_INDEX,PY%Z_INDEX)
IF (PY%Z_INDEX>=0 .AND. PY%Y_INDEX>=1) THEN
IF (TRIM(PY%QUANTITY)=='DIFFUSIVITY') THEN
PY%Y_INDEX=-999
ELSE
PY%Z_INDEX=-999
ENDIF
ENDIF
IF (PY%Y_INDEX<1 .AND. PY%Z_INDEX<0) THEN
WRITE(MESSAGE,'(A,A,A)') 'ERROR: PROP SPEC_ID ',TRIM(PY%SPEC_ID),' not found'
```

```
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF
ENDIF

ENDDO READ_PROP_LOOP


CONTAINS


SUBROUTINE SET_PROP_DEFAULTS

ACTIVATION_OBSCURATION    = 3.24_EB     ! %/m
ACTIVATION_TEMPERATURE    = -273.15_EB  ! C
ALPHA_C                   = 1.8_EB      ! m, Heskestad Length Scale
ALPHA_E                   = 0.0_EB
BETA_C                    = -1.0_EB
BETA_E                    = -1.0_EB
DENSITY                   = 8908._EB    ! kg/m3 (Nickel)
DIAMETER                  = 0.001       ! m
EMISSIVITY                = 0.85_EB
HEAT_TRANSFER_COEFFICIENT= -1._EB       ! W/m2/K
SPECIFIC_HEAT             = 0.44_EB     ! kJ/kg/K (Nickel)
C_FACTOR                  = 0.0_EB
CHARACTERISTIC_VELOCITY   = 1.0_EB      ! m/s
PARTICLE_VELOCITY          = 0._EB       ! m/s
PARTICLES_PER_SECOND       = 5000
!DT_INSERT                 = 0.01        ! s
FLOW_RATE                 = -1._EB      ! L/min
MASS_FLOW_RATE            = -1._EB
FLOW_RAMP                 = 'null'
FLOW_TAU                  = 0._EB
GAUGE_EMISSIVITY          = 1._EB
GAUGE_TEMPERATURE         = TMPA - TMPM
INITIAL_TEMPERATURE       = TMPA - TMPM
ID                        = 'null'
K_FACTOR                  = -1.0_EB     ! L/min/bar**0.5
MASS_FLOW_RATE            = -1._EB      ! kg/s
OFFSET                    = 0.05_EB     ! m
OPERATING_PRESSURE        = -1.0_EB     ! bar
ORIFICE_DIAMETER          = 0.0_EB      ! m
PART_ID                   = 'null'
PDPA_START                = T_BEGIN
PDPA_END                  = T_END + 1.0_EB
PDPA_RADIUS               = 0.0_EB
PDPA_M                    = 0
PDPA_N                    = 0
PDPA_HISTOGRAM            =.FALSE.
PDPA_HISTOGRAM_CUMULATIVE =.FALSE.
PDPA_HISTOGRAM_NBINS       = -1
PDPA_HISTOGRAM_LIMITS      = 0._EB
PDPA_HISTOGRAM_CUMULATIVE = .FALSE.
PDPA_INTEGRATE            = .TRUE.
PDPA_NORMALIZE            = .TRUE.
PRESSURE_RAMP             = 'null'
P0                        = -1.E10_EB
PX                        = 0._EB
PXX                       = 0._EB
QUANTITY                  = 'null'
RTI                       = 100._EB     ! (ms)**0.5
SMOKEVIEW_ID              = 'null'
SMOKEVIEW_PARAMETERS      = 'null'
SPEC_ID                   = 'null'
SPRAY_ANGLE(1,1)            = 60._EB      ! degrees
SPRAY_ANGLE(2,1)            = 75._EB      ! degrees
SPRAY_ANGLE(1,2)            = -999._EB      ! degrees
SPRAY_ANGLE(2,2)            = -999._EB      ! degrees
SPRAY_PATTERN_TABLE       = 'null'
SPRAY_PATTERN_SHAPE       = 'GAUSSIAN'
SPRAY_PATTERN_MU          = -1._EB
SPRAY_PATTERN_BETA        = 5.0_EB
FED_ACTIVITY              = 2 ! light work
VELOCITY_COMPONENT        = 0
END SUBROUTINE SET_PROP_DEFAULTS

END SUBROUTINE READ_PROP



SUBROUTINE PROC_PROP
USE DEVICE_VARIABLES
REAL(EB) :: TOTAL_FLOWRATE, SUBTOTAL_FLOWRATE
INTEGER :: N,NN,N_V_FACTORS,ILPC
LOGICAL :: TABLE_NORMED(1:N_TABLE)
TYPE (PROPERTY_TYPE), POINTER :: PY=>NULL()
TYPE (TABLES_TYPE),  POINTER :: TA=>NULL()
TYPE (LAGRANGIAN_PARTICLE_CLASS_TYPE),POINTER :: LPC=>NULL()

```

```
TABLE_NORMED = .FALSE.

PROP_LOOP: DO N=0,N_PROP
PY => PROPERTY(N)

! Assign PART_INDEX to Device PROPERTY array

IF (PY%PART_ID/='null') THEN

DO ILPC=1,N_LAGRANGIAN_CLASSES
LPC => LAGRANGIAN_PARTICLE_CLASS(ILPC)
IF (LPC%ID==PY%PART_ID) THEN
PY%PART_INDEX = ILPC
EXIT
ENDIF
ENDDO

IF (PY%PART_INDEX<0) THEN
WRITE(MESSAGE,'(A,I0,A)') 'ERROR: PART_ID for PROP ',N,' not found'
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF

IF (LPC%ID==PY%PART_ID .AND. LPC%MASSLESS_TRACER) THEN
IF ( .NOT.(TRIM(PY%QUANTITY)=='NUMBER CONCENTRATION' .OR. &
TRIM(PY%QUANTITY)=='U-VELOCITY'           .OR. &
TRIM(PY%QUANTITY)=='V-VELOCITY'           .OR. &
TRIM(PY%QUANTITY)=='W-VELOCITY'           .OR. &
TRIM(PY%QUANTITY)=='VELOCITY')                ) THEN
WRITE(MESSAGE,'(A,I0,A)') 'ERROR: PART_ID for PROP ',N,' cannot refer to MASSLESS particles'
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF
ENDIF

PARTICLE_FILE=.TRUE.

ENDIF

! Set up spinkler distributrion if needed

IF (PY%SPRAY_PATTERN_INDEX > 0) THEN
TA => TABLES(PY%SPRAY_PATTERN_INDEX)
ALLOCATE(PY%TABLE_ROW(1:TA%NUMBER_ROWS))
TOTAL_FLOWRATE=0._EB
SUBTOTAL_FLOWRATE=0._EB
DO NN=1,TA%NUMBER_ROWS
IF (TA%TABLE_DATA(NN,6) <=0._EB) THEN
WRITE(MESSAGE,'(A,A,A,I0)') 'ERROR: Spray Pattern Table, ',TRIM(PY%TABLE_ID),', massflux <= 0 for line ',NN
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF
TOTAL_FLOWRATE = TOTAL_FLOWRATE + TA%TABLE_DATA(NN,6)
ENDDO
IF (TABLE_NORMED(PY%SPRAY_PATTERN_INDEX)) THEN
DO NN=1,TA%NUMBER_ROWS
SUBTOTAL_FLOWRATE = SUBTOTAL_FLOWRATE + TA%TABLE_DATA(NN,6)
PY%TABLE_ROW(NN) = SUBTOTAL_FLOWRATE/TOTAL_FLOWRATE
ENDDO
ELSE
DO NN=1,TA%NUMBER_ROWS
TA%TABLE_DATA(NN,1) = TA%TABLE_DATA(NN,1) * PI/180._EB
TA%TABLE_DATA(NN,2) = TA%TABLE_DATA(NN,2) * PI/180._EB
TA%TABLE_DATA(NN,3) = TA%TABLE_DATA(NN,3) * PI/180._EB
TA%TABLE_DATA(NN,4) = TA%TABLE_DATA(NN,4) * PI/180._EB
SUBTOTAL_FLOWRATE = SUBTOTAL_FLOWRATE + TA%TABLE_DATA(NN,6)
PY%TABLE_ROW(NN) = SUBTOTAL_FLOWRATE/TOTAL_FLOWRATE
ENDDO
TABLE_NORMED(PY%SPRAY_PATTERN_INDEX) = .TRUE.
ENDIF
PY%TABLE_ROW(TA%NUMBER_ROWS) = 1._EB
END IF

! Set up pressure dependence
IF (PY%PRESSURE_RAMP_INDEX > 0) THEN
IF (PY%SPRAY_PATTERN_INDEX > 0) THEN
N_V_FACTORS = TA%NUMBER_ROWS
ELSE
N_V_FACTORS = 1
ENDIF
ALLOCATE(PY%V_FACTOR(1:N_V_FACTORS))
IF (PY%SPRAY_PATTERN_INDEX > 0) THEN
DO NN=1,TA%NUMBER_ROWS
PY%V_FACTOR(NN) = TA%TABLE_DATA(NN,5)/SQRT(PY%OPERATING_PRESSURE)
ENDDO
ELSE
PY%V_FACTOR = PY%PARTICLE_VELOCITY/SQRT(PY%OPERATING_PRESSURE)
ENDIF
ENDIF

ENDDO PROP_LOOP
```

```fortran

END SUBROUTINE PROC_PROP



SUBROUTINE READ_MATL

USE MATH_FUNCTIONS, ONLY : GET_RAMP_INDEX
CHARACTER(LABEL_LENGTH) :: CONDUCTIVITY_RAMP,SPECIFIC_HEAT_RAMP
CHARACTER(LABEL_LENGTH) :: SPEC_ID(MAX_SPECIES,MAX_REACTIONS)
REAL(EB) :: EMISSIVITY,CONDUCTIVITY,SPECIFIC_HEAT,DENSITY,ABSORPTION_COEFFICIENT,BOILING_TEMPERATURE, &
PEAK_REACTION_RATE,POROSITY
REAL(EB), DIMENSION(MAX_MATERIALS,MAX_REACTIONS) :: NU_MATL
REAL(EB), DIMENSION(MAX_REACTIONS) :: A,E,HEATING_RATE,PYROLYSIS_RANGE,HEAT_OF_REACTION, &
N_S,N_T,N_O2,REFERENCE_RATE,REFERENCE_TEMPERATURE,THRESHOLD_TEMPERATURE,HEAT_OF_COMBUSTION, &
THRESHOLD_SIGN,GAS_DIFFUSION_DEPTH,NU_O2,BETA_CHAR
REAL(EB) :: NU_SPEC(MAX_SPECIES,MAX_REACTIONS)
LOGICAL, DIMENSION(MAX_REACTIONS) :: PCR
LOGICAL :: ALLOW_SHRINKING, ALLOW_SWELLING,VEGETATION
CHARACTER(25) :: COLOR
INTEGER :: RGB(3)
CHARACTER(LABEL_LENGTH), DIMENSION(MAX_MATERIALS,MAX_REACTIONS) :: MATL_ID
INTEGER :: N,NN,NNN,IOS,NR,N_REACTIONS
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

! Count the MATL lines in the input file

REWIND(LU_INPUT) ; INPUT_FILE_LINE_NUMBER = 0
N_MATL = 0
COUNT_MATL_LOOP: DO
CALL CHECKREAD('MATL',LU_INPUT,IOS)
IF (IOS==1) EXIT COUNT_MATL_LOOP
READ(LU_INPUT,MATL,ERR=34,IOSTAT=IOS)
N_MATL = N_MATL + 1
MATL_NAME(N_MATL) = ID
34 IF (IOS>0) THEN
WRITE(MESSAGE,'(A,I0,A,I0)') 'ERROR: Problem with MATL number',N_MATL+1,', line number',INPUT_FILE_LINE_NUMBER
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF
ENDDO COUNT_MATL_LOOP

! Allocate the MATERIAL derived type

ALLOCATE(MATERIAL(1:N_MATL),STAT=IZERO)
CALL ChkMemErr('READ','MATERIAL',IZERO)

! Read the MATL lines in the order listed in the input file

REWIND(LU_INPUT) ; INPUT_FILE_LINE_NUMBER = 0

READ_MATL_LOOP: DO N=1,N_MATL

ML => MATERIAL(N)

! Read user defined MATL lines

CALL CHECKREAD('MATL',LU_INPUT,IOS)
CALL SET_MATL_DEFAULTS
READ(LU_INPUT,MATL)

! Do some error checking on the inputs

NOT_BOILING: IF (BOILING_TEMPERATURE>4000._EB) THEN

IF ( ( ANY(THRESHOLD_TEMPERATURE>-TMPM) .OR. ANY(REFERENCE_TEMPERATURE>-TMPM) .OR. ANY(A>=0._EB) .OR. ANY(E>=0.
      _EB) .OR. &
ANY(ABS(HEAT_OF_REACTION)>TWO_EPSILON_EB) ) .AND. N_REACTIONS==0) THEN
WRITE(MESSAGE,'(A,A,A)') 'ERROR: Problem with MATL number ',TRIM(ID),'. A reaction parameter is used, but
      N_REACTIONS=0'
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF

DO NR=1,N_REACTIONS
IF (REFERENCE_TEMPERATURE(NR)<-TMPM .AND. (E(NR)< 0._EB .OR. A(NR)<0._EB)) THEN
WRITE(MESSAGE,'(A,A,A,I0,A)') 'ERROR: Problem with MATL ',TRIM(ID),', REAC ',NR,'. Set REFERENCE_TEMPERATURE or E
      , A'
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF
IF (ABS(SUM(NU_MATL(:,NR)))<=TWO_EPSILON_EB .AND. ABS(SUM(NU_SPEC(:,NR)))<=TWO_EPSILON_EB) THEN
WRITE(MESSAGE,'(A,A,A,I0,A)') 'WARNING: MATL ',TRIM(ID),', REAC ',NR,'. No product yields (NUs) set'
IF (MYID==0) WRITE(LU_ERR,'(A)') TRIM(MESSAGE)
ENDIF
ENDDO
```

```
ELSE NOT_BOILING ! Is liquid

N_REACTIONS = 1
IF (ABS(HEAT_OF_REACTION(1))<=TWO_EPSILON_EB) THEN
WRITE(MESSAGE,'(A,A)') 'ERROR: HEAT_OF_REACTION should be greater than zero for liquid MATL ',TRIM(ID)
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF

ENDIF NOT_BOILING

! Error checking for thermal properties

IF (ABS(DENSITY) <=TWO_EPSILON_EB ) THEN
WRITE(MESSAGE,'(A,A,A)') 'ERROR: Problem with MATL ',TRIM(ID),': DENSITY=0'
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF
IF (ABS(CONDUCTIVITY) <=TWO_EPSILON_EB .AND. CONDUCTIVITY_RAMP == 'null' ) THEN
WRITE(MESSAGE,'(A,A,A)') 'ERROR: Problem with MATL ',TRIM(ID),': CONDUCTIVITY = 0'
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF
IF (ABS(SPECIFIC_HEAT)<=TWO_EPSILON_EB .AND. SPECIFIC_HEAT_RAMP == 'null' ) THEN
WRITE(MESSAGE,'(A,A,A)') 'ERROR: Problem with MATL ',TRIM(ID),': SPECIFIC_HEAT = 0'
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF
IF (SPECIFIC_HEAT > 10._EB) WRITE(LU_ERR,'(A,A)') 'WARNING: SPECIFIC_HEAT units are kJ/kg/K check MATL ',TRIM(ID)

! Pack MATL parameters into the MATERIAL derived type

IF (COLOR/='null') THEN
CALL COLOR2RGB(RGB,COLOR)
ENDIF
IF (ANY(RGB< 0)) THEN
RGB(1) = 255
RGB(2) = 204
RGB(3) = 102
ENDIF
ML%RGB                    = RGB

ML%A(:)                   = A(:)

ALLOCATE(ML%ADJUST_BURN_RATE(N_TRACKED_SPECIES,MAX(1,N_REACTIONS)),STAT=IZERO)
CALL ChkMemErr('READ','MATERIAL',IZERO)
ML%ADJUST_BURN_RATE       = 1._EB
ML%ALLOW_SHRINKING        = ALLOW_SHRINKING
ML%ALLOW_SWELLING         = ALLOW_SWELLING
ML%BETA_CHAR(:)           = BETA_CHAR(:)
ML%C_S                    = 1000._EB*SPECIFIC_HEAT/TIME_SHRINK_FACTOR
ML%E(:)                   = 1000._EB*E(:)
ML%EMISSIVITY             = EMISSIVITY
ML%FYI                    = FYI
ML%GAS_DIFFUSION_DEPTH(:) = GAS_DIFFUSION_DEPTH(:)
ML%HEAT_OF_COMBUSTION     = 1000._EB*HEAT_OF_COMBUSTION
ML%H_R(:)                 = 1000._EB*HEAT_OF_REACTION(:)
ML%ID                     = ID
ML%KAPPA_S                = ABSORPTION_COEFFICIENT
ML%K_S                    = CONDUCTIVITY
ML%N_REACTIONS            = N_REACTIONS
ML%N_O2(:)                = N_O2(:)
ML%NU_O2(:)               = NU_O2(:)
ML%N_S(:)                 = N_S(:)
ML%N_T(:)                 = N_T(:)
ML%NU_RESIDUE             = NU_MATL
ML%NU_SPEC                = NU_SPEC
ML%POROSITY               = POROSITY
ML%SPEC_ID                = SPEC_ID
ML%RAMP_C_S               = SPECIFIC_HEAT_RAMP
ML%RAMP_K_S               = CONDUCTIVITY_RAMP
ML%RHO_S                  = DENSITY      ! This is bulk density of pure material.
ML%RESIDUE_MATL_NAME      = MATL_ID
ML%HEATING_RATE(:)        = HEATING_RATE(:)/60._EB
ML%PYROLYSIS_RANGE(:)     = PYROLYSIS_RANGE(:)
ML%PCR(:)                 = PCR(:)
ML%TMP_BOIL               = BOILING_TEMPERATURE + TMPM
ML%TMP_THR(:)             = THRESHOLD_TEMPERATURE(:) + TMPM
ML%TMP_REF(:)             = REFERENCE_TEMPERATURE(:) + TMPM
ML%THR_SIGN(:)            = THRESHOLD_SIGN
ML%RATE_REF(:)            = REFERENCE_RATE(:)

ALLOCATE(ML%NU_GAS(N_TRACKED_SPECIES,N_REACTIONS),STAT=IZERO)
CALL ChkMemErr('READ','MATERIAL',IZERO)
ML%NU_GAS=0._EB

! Additional logic

IF (BOILING_TEMPERATURE<5000._EB) THEN
ML%PYROLYSIS_MODEL = PYROLYSIS_LIQUID
ML%N_REACTIONS = 1
```

```
ELSEIF (VEGETATION) THEN
ML%PYROLYSIS_MODEL = PYROLYSIS_VEGETATION
ML%ALLOW_SHRINKING = .FALSE.
ML%ALLOW_SWELLING  = .FALSE.
ELSE
ML%PYROLYSIS_MODEL = PYROLYSIS_SOLID
ENDIF

IF (N_REACTIONS==0) ML%PYROLYSIS_MODEL = PYROLYSIS_NONE

IF (ML%RAMP_K_S/='null') THEN
CALL GET_RAMP_INDEX(ML%RAMP_K_S,'TEMPERATURE',NR)
ML%K_S = -NR
ENDIF

IF (ML%RAMP_C_S/='null') THEN
CALL GET_RAMP_INDEX(ML%RAMP_C_S,'TEMPERATURE',NR)
ML%C_S = -NR
ENDIF

! Determine A and E if REFERENCE_TEMPERATURE is specified

DO NR=1,ML%N_REACTIONS
IF (ML%TMP_REF(NR) > 0._EB) THEN
IF (ML%RATE_REF(NR) > 0._EB) THEN
PEAK_REACTION_RATE = ML%RATE_REF(NR)
ELSE
PEAK_REACTION_RATE = 2._EB*ML%HEATING_RATE(NR)*(1._EB-SUM(ML%NU_RESIDUE(:,NR)))/ML%PYROLYSIS_RANGE(NR)
ENDIF
ML%E(NR) = EXP(1._EB)*PEAK_REACTION_RATE*R0*ML%TMP_REF(NR)**2/ML%HEATING_RATE(NR)
ML%A(NR) = EXP(1._EB)*PEAK_REACTION_RATE*EXP(ML%E(NR)/(R0*ML%TMP_REF(NR)))
ENDIF

ML%N_RESIDUE(NR) = 0
DO NN=1,MAX_MATERIALS
IF (ML%RESIDUE_MATL_NAME(NN,NR)/='null') ML%N_RESIDUE(NR) = ML%N_RESIDUE(NR) + 1
ENDDO
ENDDO

ENDDO READ_MATL_LOOP

! Assign a material index to the RESIDUEs

DO N=1,N_MATL
ML => MATERIAL(N)
ML%RESIDUE_MATL_INDEX = 0
DO NR=1,ML%N_REACTIONS
DO NN=1,ML%N_RESIDUE(NR)
DO NNN=1,N_MATL
IF (MATL_NAME(NNN)==ML%RESIDUE_MATL_NAME(NN,NR)) ML%RESIDUE_MATL_INDEX(NN,NR) = NNN
ENDDO
IF (ML%RESIDUE_MATL_INDEX(NN,NR)==0 .AND. ML%NU_RESIDUE(NN,NR)>0._EB) THEN
WRITE(MESSAGE,'(5A)') 'ERROR: Residue ',TRIM(ML%RESIDUE_MATL_NAME(NN,NR)),' of ',TRIM(MATL_NAME(N)),' is not
     defined.'
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF
ENDDO
ENDDO
ENDDO

! Check for duplicate names

IF (N_MATL>1) THEN
DO N=1,N_MATL-1
DO NN=N+1,N_MATL
IF(MATL_NAME(N)==MATL_NAME(NN)) THEN
WRITE(MESSAGE,'(A,A)') 'ERROR: Duplicate material name: ',TRIM(MATL_NAME(N))
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF
ENDDO
ENDDO
ENDIF

! Check material porosity values

DO N=1,N_MATL
ML => MATERIAL(N)
IF ((1.0_EB-ML%POROSITY) < 1.0E-5_EB) THEN
WRITE(MESSAGE,'(2A)') 'ERROR: Too high porosity for material ',TRIM(MATL_NAME(N))
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF
!   ML%RHO_NONPOROUS          = DENSITY/(1._EB-POROSITY)
ENDDO

CONTAINS

SUBROUTINE SET_MATL_DEFAULTS
```

```
A                        = -1._EB        ! 1/s
ABSORPTION_COEFFICIENT   = 5.0E4_EB      ! 1/m, corresponds to 99.3% drop within 1E-4 m distance.
ALLOW_SHRINKING          = .TRUE.
ALLOW_SWELLING           = .TRUE.
BOILING_TEMPERATURE      = 5000._EB      ! C
BETA_CHAR                = 0.2_EB
COLOR                    = 'null'
RGB                       = -1
CONDUCTIVITY             = 0.0_EB        ! W/m/K
CONDUCTIVITY_RAMP        = 'null'
DENSITY                  = 0._EB         ! kg/m3
E                        = -1._EB        ! kJ/kmol
EMISSIVITY               = 0.9_EB
FYI                      = 'null'
GAS_DIFFUSION_DEPTH      = 0.001_EB      ! m
HEAT_OF_COMBUSTION       = -1._EB        ! kJ/kg
HEAT_OF_REACTION         = 0._EB         ! kJ/kg
ID                       = 'null'
THRESHOLD_TEMPERATURE    = -TMPM         ! 0 K
THRESHOLD_SIGN           = 1.0
N_REACTIONS              = 0
N_O2                     = 0._EB
NU_O2                    = 0._EB
N_S                      = 1._EB
N_T                      = 0._EB
NU_SPEC                  = 0._EB
NU_MATL                  = 0._EB
PCR                      = .FALSE.
POROSITY                 = 0._EB
REFERENCE_RATE           = -1._EB
REFERENCE_TEMPERATURE    = -1000._EB
MATL_ID                  = 'null'
SPECIFIC_HEAT            = 0.0_EB        ! kJ/kg/K
SPECIFIC_HEAT_RAMP       = 'null'
SPEC_ID                  = 'null'
HEATING_RATE             = 5._EB         ! K/min
PYROLYSIS_RANGE          = 80._EB        ! K or C
VEGETATION               = .FALSE.

END SUBROUTINE SET_MATL_DEFAULTS

END SUBROUTINE READ_MATL



SUBROUTINE PROC_MATL

! Process Materials -- do some additional set-up work with materials

INTEGER :: N,J,JJ,NS,NS2,NR,Z_INDEX(N_TRACKED_SPECIES,MAX_REACTIONS)

PROC_MATL_LOOP: DO N=1,N_MATL

ML => MATERIAL(N)

! Convert ML%NU_SPEC(I_ORDINAL,I_REACTION) and ML%SPEC_ID(I_ORDINAL,I_REACTION) to ML%NU_GAS(I_SPECIES,I_REACTION)

Z_INDEX = -1
DO NR=1,ML%N_REACTIONS
DO NS=1,MAX_SPECIES

IF (TRIM(ML%SPEC_ID(NS,NR))=='null' .AND. ML%NU_SPEC(NS,NR)>TWO_EPSILON_EB) THEN
WRITE(MESSAGE,'(A,A,A,I0,A,I0)') 'ERROR: MATL ',TRIM(MATL_NAME(N)),' requires a SPEC_ID for the ',&
NS, 'th yield of reaction', NR
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF
IF (TRIM(ML%SPEC_ID(NS,NR))=='null') EXIT
IF (NS==2 .AND. ML%PYROLYSIS_MODEL==PYROLYSIS_LIQUID) THEN
WRITE(MESSAGE,'(A,A,A)') 'ERROR: MATL ',TRIM(MATL_NAME(N)),' can only specify one SPEC_ID for a liquid'
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF
DO NS2=1,N_TRACKED_SPECIES
IF (TRIM(ML%SPEC_ID(NS,NR))==TRIM(SPECIES_MIXTURE(NS2)%ID)) THEN
Z_INDEX(NS,NR) = NS2
ML%NU_GAS(Z_INDEX(NS,NR),NR) = ML%NU_SPEC(NS,NR)
EXIT
ENDIF
ENDDO
IF (Z_INDEX(NS,NR)==-1) THEN
WRITE(MESSAGE,'(A,A,A,A,A)') 'ERROR: SPECies ',TRIM(ML%SPEC_ID(NS,NR)),&
' corresponding to MATL ',TRIM(MATL_NAME(N)),' is not a tracked species'
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF

ENDDO
ENDDO

```

```
! Adjust burn rate if heat of combustion is different from the gas phase reaction value

IF (N_REACTIONS>0) THEN
RN => REACTION(1)
DO NS = 1,N_TRACKED_SPECIES
DO J=0,MAX(1,ML%N_REACTIONS)
JJ = MAX(J,1)
IF (ML%HEAT_OF_COMBUSTION(JJ)>0._EB .AND. RN%HEAT_OF_COMBUSTION>0._EB) &
ML%ADJUST_BURN_RATE(NS,JJ) = ML%HEAT_OF_COMBUSTION(JJ)/RN%HEAT_OF_COMBUSTION
ENDDO
ENDDO
ENDIF

! Check units of specific heat

IF (ML%RAMP_C_S/='null') THEN
NR = -NINT(ML%C_S)
IF (.NOT.RAMPS(NR)%DEP_VAR_UNITS_CONVERTED) THEN
RAMPS(NR)%INTERPOLATED_DATA(:) = RAMPS(NR)%INTERPOLATED_DATA(:)*1000._EB/TIME_SHRINK_FACTOR
RAMPS(NR)%DEP_VAR_UNITS_CONVERTED = .TRUE.
ENDIF
IF (RAMPS(NR)%DEPENDENT_DATA(1) > 10._EB) &
WRITE(LU_ERR,'(A,A)') 'WARNING: SPECIFIC_HEAT units are kJ/kg/K check MATL ',TRIM(ID)
ENDIF

ENDDO PROC_MATL_LOOP

END SUBROUTINE PROC_MATL


SUBROUTINE READ_SURF

USE MATH_FUNCTIONS, ONLY : GET_RAMP_INDEX
USE DEVICE_VARIABLES, ONLY : PROPERTY_TYPE
CHARACTER(LABEL_LENGTH) :: PART_ID,RAMP_MF(MAX_SPECIES),RAMP_Q,RAMP_V,RAMP_T,MATL_ID(MAX_LAYERS,MAX_MATERIALS),&
PROFILE,BACKING,GEOMETRY,NAME_LIST(MAX_MATERIALS*MAX_LAYERS),EXTERNAL_FLUX_RAMP,RAMP_EF,RAMP_PART,&
SPEC_ID(MAX_SPECIES),RAMP_T_I,RAMP_V_X,RAMP_V_Y,RAMP_V_Z
EQUIVALENCE(EXTERNAL_FLUX_RAMP,RAMP_EF)
LOGICAL :: ADIABATIC,BURN_AWAY,FREE_SLIP,NO_SLIP,CONVERT_VOLUME_TO_MASS
CHARACTER(60) :: TEXTURE_MAP,HEAT_TRANSFER_MODEL
CHARACTER(25) :: COLOR
REAL(EB) :: TAU_Q,TAU_V,TAU_T,TAU_MF(MAX_SPECIES),HRRPUA,MLRPUA,TEXTURE_WIDTH,TEXTURE_HEIGHT,VEL_T(2),&
TAU_EXTERNAL_FLUX,TAU_EF,E_COEFFICIENT,VOLUME_FLOW,VOLUME_FLUX,&
TMP_FRONT,TMP_INNER(MAX_LAYERS),THICKNESS(MAX_LAYERS),VEL,VEL_BULK,INTERNAL_HEAT_SOURCE(MAX_LAYERS),&
MASS_FLUX(MAX_SPECIES),Z0,PLE,CONVECTIVE_HEAT_FLUX,PARTICLE_MASS_FLUX,&
TRANSPARENCY,EXTERNAL_FLUX,TMP_BACK,MASS_FLUX_TOTAL,MASS_FLUX_VAR,STRETCH_FACTOR(MAX_LAYERS),
    CONVECTION_LENGTH_SCALE,&
MATL_MASS_FRACTION(MAX_LAYERS,MAX_MATERIALS),CELL_SIZE_FACTOR,MAX_PRESSURE,&
IGNITION_TEMPERATURE,HEAT_OF_VAPORIZATION,NET_HEAT_FLUX,LAYER_DIVIDE,&
ROUGHNESS,RADIUS,INNER_RADIUS,LENGTH,WIDTH,DT_INSERT,HEAT_TRANSFER_COEFFICIENT,HEAT_TRANSFER_COEFFICIENT_BACK,&
TAU_PART,EMISSIVITY,EMISSIVITY_BACK,EMISSIVITY_DEFAULT,SPREAD_RATE,XYZ(3),MINIMUM_LAYER_THICKNESS,VEL_GRAD,&
MASS_FRACTION(MAX_SPECIES),MASS_TRANSFER_COEFFICIENT,&
C_FORCED_CONSTANT,C_FORCED_PR_EXP,C_FORCED_RE,C_FORCED_RE_EXP,C_VERTICAL,C_HORIZONTAL,ZETA_FRONT,&
AUTO_IGNITION_TEMPERATURE
EQUIVALENCE(TAU_EXTERNAL_FLUX,TAU_EF)
INTEGER :: NPPC,N,IOS,NL,NN,NNN,NRM,N_LIST,N_LIST2,INDEX_LIST(MAX_MATERIALS_TOTAL),LEAK_PATH(2),DUCT_PATH(2),RGB
    (3),&
NR,IL,N_CELLS_MAX
INTEGER :: VEGETATION_LAYERS,N_LAYER_CELLS_MAX(MAX_LAYERS)
REAL(EB) :: VEGETATION_CDRAG,VEGETATION_CHAR_FRACTION,VEGETATION_ELEMENT_DENSITY,VEGETATION_HEIGHT, &
VEGETATION_INITIAL_TEMP,VEGETATION_LOAD,VEGETATION_LSET_IGNITE_TIME,VEG_LSET_QCON,VEGETATION_MOISTURE, &
VEGETATION_SVRATIO, &
FIRELINE_MLR_MAX,VEGETATION_GROUND_TEMP,VEG_LSET_ROS_HEAD,VEG_LSET_ROS_FLANK,VEG_LSET_ROS_BACK, &
VEG_LSET_WIND_EXP,VEG_LSET_BETA,VEG_LSET_HT,VEG_LSET_SIGMA,VEG_LSET_ELLIPSE_HEAD
LOGICAL :: VEGETATION,VEGETATION_NO_BURN,VEGETATION_LINEAR_DEGRAD,VEGETATION_ARRHENIUS_DEGRAD,
    VEG_LEVEL_SET_SPREAD,&
DEFAULT,EVAC_DEFAULT,VEG_LSET_ELLIPSE,VEG_LSET_TAN2,TGA_ANALYSIS,COMPUTE_EMISSIVITY,COMPUTE_EMISSIVITY_BACK,&
HT3D

NAMELIST /SURF/ ADIABATIC,AUTO_IGNITION_TEMPERATURE,&
BACKING,BURN_AWAY,CELL_SIZE_FACTOR,C_FORCED_CONSTANT,C_FORCED_PR_EXP,C_FORCED_RE,C_FORCED_RE_EXP,&
C_HORIZONTAL,C_VERTICAL,COLOR,&
CONVECTION_LENGTH_SCALE,CONVECTIVE_HEAT_FLUX,CONVERT_VOLUME_TO_MASS,DEFAULT,&
DT_INSERT,EMISSIVITY,EMISSIVITY_BACK,EVAC_DEFAULT,EXTERNAL_FLUX,E_COEFFICIENT,FIRELINE_MLR_MAX,&
FREE_SLIP,FYI,GEOMETRY,HEAT_OF_VAPORIZATION,HEAT_TRANSFER_COEFFICIENT,HEAT_TRANSFER_COEFFICIENT_BACK,&
HEAT_TRANSFER_MODEL,HRRPUA,HT3D,ID,IGNITION_TEMPERATURE,INNER_RADIUS,INTERNAL_HEAT_SOURCE,LAYER_DIVIDE,&
LEAK_PATH,LENGTH,MASS_FLUX,MASS_FLUX_TOTAL,MASS_FLUX_VAR,MASS_FRACTION,MASS_TRANSFER_COEFFICIENT,MATL_ID, &
MATL_MASS_FRACTION,MINIMUM_LAYER_THICKNESS,MLRPUA,N_CELLS_MAX,N_LAYER_CELLS_MAX,NET_HEAT_FLUX,&
NO_SLIP,NPPC,PARTICLE_MASS_FLUX,PART_ID,PLE,PROFILE,RADIUS,RAMP_EF,RAMP_MF,RAMP_PART,RAMP_Q,RAMP_T,RAMP_T_I, &
RAMP_V,RAMP_V_X,RAMP_V_Y,RAMP_V_Z,&
RGB,ROUGHNESS,SPEC_ID,SPREAD_RATE,STRETCH_FACTOR,&
TAU_EF,TAU_MF,TAU_PART,TAU_Q,TAU_T,TAU_V,TEXTURE_HEIGHT,TEXTURE_MAP,TEXTURE_WIDTH,&
TGA_ANALYSIS,TGA_FINAL_TEMPERATURE,TGA_HEATING_RATE,THICKNESS,&
TMP_BACK,TMP_FRONT,TMP_INNER,TRANSPARENCY,VEGETATION,VEGETATION_ARRHENIUS_DEGRAD,VEGETATION_CDRAG,&
VEGETATION_CHAR_FRACTION,VEGETATION_ELEMENT_DENSITY,VEGETATION_GROUND_TEMP,VEGETATION_HEIGHT,&
VEGETATION_INITIAL_TEMP,VEGETATION_LAYERS,VEGETATION_LINEAR_DEGRAD,VEGETATION_LOAD,VEGETATION_LSET_IGNITE_TIME,&
VEG_LSET_QCON,VEGETATION_MOISTURE,VEGETATION_NO_BURN,VEGETATION_SVRATIO,VEG_LEVEL_SET_SPREAD,&
VEG_LSET_ROS_BACK,VEG_LSET_ROS_FLANK,VEG_LSET_ROS_HEAD,VEG_LSET_WIND_EXP,&
```

```
VEG_LSET_SIGMA,VEG_LSET_HT,VEG_LSET_BETA,VEG_LSET_ELLIPSE,VEG_LSET_TAN2,VEG_LSET_ELLIPSE_HEAD,&
VEL,VEL_BULK,VEL_GRAD,VEL_T,VOLUME_FLOW,WIDTH,XYZ,Z0,ZETA_FRONT,&
EXTERNAL_FLUX_RAMP,TAU_EXTERNAL_FLUX,VOLUME_FLUX ! Backwards compatability??

! Count the SURF lines in the input file

REWIND(LU_INPUT) ; INPUT_FILE_LINE_NUMBER = 0
N_SURF = 0
COUNT_SURF_LOOP: DO
HRRPUA = 0._EB
MLRPUA = 0._EB
CALL CHECKREAD('SURF',LU_INPUT,IOS)
IF (IOS==1) EXIT COUNT_SURF_LOOP
READ(LU_INPUT,SURF,ERR=34,IOSTAT=IOS)
N_SURF = N_SURF + 1
34 IF (IOS>0) THEN
WRITE(MESSAGE,'(A,I0,A,I0)') 'ERROR: Problem with SURF number', N_SURF+1,', line number',INPUT_FILE_LINE_NUMBER
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF
ENDDO COUNT_SURF_LOOP

! Allocate the SURFACE derived type, leaving space for SURF entries not defined explicitly by the user

N_SURF_RESERVED = 11
ALLOCATE(SURFACE(0:N_SURF+N_SURF_RESERVED),STAT=IZERO)
CALL ChkMemErr('READ','SURFACE',IZERO)

! Count the SURF lines in the input file

REWIND(LU_INPUT) ; INPUT_FILE_LINE_NUMBER = 0
NN = 0
COUNT_SURF_LOOP_AGAIN: DO
CALL CHECKREAD('SURF',LU_INPUT,IOS)
IF (IOS==1) EXIT COUNT_SURF_LOOP_AGAIN
READ(LU_INPUT,SURF)
NN = NN+1
SURFACE(NN)%ID = ID
ENDDO COUNT_SURF_LOOP_AGAIN

! Add extra surface types to the list that has already been compiled

INERT_SURF_INDEX               = 0
OPEN_SURF_INDEX                = N_SURF + 1
MIRROR_SURF_INDEX              = N_SURF + 2
INTERPOLATED_SURF_INDEX        = N_SURF + 3
PERIODIC_SURF_INDEX            = N_SURF + 4
HVAC_SURF_INDEX                = N_SURF + 5
MASSLESS_TRACER_SURF_INDEX     = N_SURF + 6
DROPLET_SURF_INDEX             = N_SURF + 7
VEGETATION_SURF_INDEX          = N_SURF + 8
EVACUATION_SURF_INDEX          = N_SURF + 9
MASSLESS_TARGET_SURF_INDEX     = N_SURF + 10
PERIODIC_WIND_SURF_INDEX       = N_SURF + 11

N_SURF = N_SURF + N_SURF_RESERVED

SURFACE(INERT_SURF_INDEX)%ID              = 'INERT'
SURFACE(OPEN_SURF_INDEX)%ID               = 'OPEN'
SURFACE(MIRROR_SURF_INDEX)%ID             = 'MIRROR'
SURFACE(INTERPOLATED_SURF_INDEX)%ID       = 'INTERPOLATED'
SURFACE(PERIODIC_SURF_INDEX)%ID           = 'PERIODIC'
SURFACE(HVAC_SURF_INDEX)%ID               = 'HVAC'
SURFACE(MASSLESS_TRACER_SURF_INDEX)%ID    = 'MASSLESS TRACER'
SURFACE(DROPLET_SURF_INDEX)%ID            = 'DROPLET'
SURFACE(VEGETATION_SURF_INDEX)%ID         = 'VEGETATION'
SURFACE(EVACUATION_SURF_INDEX)%ID         = 'EVACUATION_OUTFLOW'
SURFACE(MASSLESS_TARGET_SURF_INDEX)%ID    = 'MASSLESS TARGET'
SURFACE(PERIODIC_WIND_SURF_INDEX)%ID      = 'PERIODIC WIND'

SURFACE(0)%USER_DEFINED                              = .FALSE.
SURFACE(N_SURF-N_SURF_RESERVED+1:N_SURF)%USER_DEFINED = .FALSE.

! Check if SURF_DEFAULT exists

CALL CHECK_SURF_NAME(SURF_DEFAULT,EX)
IF (.NOT.EX) THEN
WRITE(MESSAGE,'(A)') 'ERROR: SURF_DEFAULT not found'
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF

! Add evacuation boundary type if necessary

CALL CHECK_SURF_NAME(EVAC_SURF_DEFAULT,EX)
IF (.NOT.EX) THEN
WRITE(MESSAGE,'(A)') 'ERROR: EVAC_SURF_DEFAULT not found'
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF
```

```
! Read the SURF lines

REWIND(LU_INPUT) ; INPUT_FILE_LINE_NUMBER = 0
READ_SURF_LOOP: DO N=0,N_SURF

SF => SURFACE(N)

! Allocate arrays associated with the SURF line

ALLOCATE(SF%MASS_FRACTION(1:N_TRACKED_SPECIES),STAT=IZERO)
CALL ChkMemErr('READ','SURFACE',IZERO) ; SF%MASS_FRACTION = 0._EB
ALLOCATE(SF%MASS_FLUX(1:N_TRACKED_SPECIES),STAT=IZERO)
CALL ChkMemErr('READ','SURFACE',IZERO) ; SF%MASS_FLUX = 0._EB
ALLOCATE(SF%TAU(-5:N_TRACKED_SPECIES),STAT=IZERO)
CALL ChkMemErr('READ','SURFACE',IZERO) ; SF%TAU = 0._EB
ALLOCATE(SF%ADJUST_BURN_RATE(-5:N_TRACKED_SPECIES),STAT=IZERO)
CALL ChkMemErr('READ','SURFACE',IZERO) ; SF%ADJUST_BURN_RATE = 1._EB
ALLOCATE(SF%RAMP_INDEX(-8:N_TRACKED_SPECIES),STAT=IZERO)
CALL ChkMemErr('READ','SURFACE',IZERO) ; SF%RAMP_INDEX = 0
ALLOCATE(SF%RAMP_MF(1:N_TRACKED_SPECIES),STAT=IZERO)
CALL ChkMemErr('READ','SURFACE',IZERO) ; SF%RAMP_MF = 'null'

! Read the user defined SURF lines

CALL SET_SURF_DEFAULTS

IF (SF%USER_DEFINED) THEN
CALL CHECKREAD('SURF',LU_INPUT,IOS)
READ(LU_INPUT,SURF)
ENDIF

! Check to make sure that a DEFAULT SURF has an ID

IF (DEFAULT) THEN
IF (ID=='null') ID = 'DEFAULT SURF'
SURF_DEFAULT = TRIM(ID)
ENDIF

IF (EVAC_DEFAULT) EVAC_SURF_DEFAULT = TRIM(ID)

! Look for special TGA_ANALYSIS=.TRUE. to indicate that only a TGA analysis is to be done

IF (TGA_ANALYSIS) THEN
GEOMETRY = 'CARTESIAN'
LENGTH    = 0.1
WIDTH     = 0.1
BACKING   = 'INSULATED'
IF (THICKNESS(2)>0._EB) THEN
WRITE(MESSAGE,'(A)') 'ERROR: IF TGA_ANALYSIS=.TRUE., the surface can only be one layer thick'
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF
THICKNESS = 1.E-6_EB
HEAT_TRANSFER_COEFFICIENT = 1000._EB
MINIMUM_LAYER_THICKNESS = 1.E-12_EB
TGA_SURF_INDEX = N
INITIAL_RADIATION_ITERATIONS = 0
ENDIF

! Vegetation parameters

IF (VEGETATION) WFDS_BNDRYFUEL = .TRUE.

! Level set vegetation fire spread specific
SF%VEG_LSET_SPREAD       = VEG_LEVEL_SET_SPREAD
SF%VEG_LSET_ROS_HEAD     = VEG_LSET_ROS_HEAD !head fire rate of spread m/s
SF%VEG_LSET_ELLIPSE_HEAD = VEG_LSET_ELLIPSE_HEAD !no-wind, no-slope ros for elliptical model in level set
SF%VEG_LSET_ROS_FLANK = VEG_LSET_ROS_FLANK !flank fire rate of spread
SF%VEG_LSET_ROS_BACK  = VEG_LSET_ROS_BACK !back fire rate of spread
SF%VEG_LSET_WIND_EXP  = VEG_LSET_WIND_EXP !exponent on wind cosine in ROS formula
SF%VEG_LSET_SIGMA        = VEG_LSET_SIGMA * 0.01 !SAV for Farsite emulation in LSET converted to 1/cm
SF%VEG_LSET_HT           = VEG_LSET_HT
SF%VEG_LSET_BETA         = VEG_LSET_BETA
SF%VEG_LSET_ELLIPSE      = VEG_LSET_ELLIPSE
SF%VEG_LSET_TAN2         = VEG_LSET_TAN2

! Boundary Vegetation specific

SF%VEGETATION     = VEGETATION !T or F
SF%VEG_NO_BURN    = VEGETATION_NO_BURN
IF(WIND_ONLY) SF%VEG_NO_BURN = .TRUE.
!  IF(SF%VEGETATION) ADIABATIC = .TRUE.
SF%VEG_CHARFRAC = VEGETATION_CHAR_FRACTION
SF%VEG_MOISTURE = VEGETATION_MOISTURE
SF%VEG_HEIGHT     = VEGETATION_HEIGHT
SF%VEG_INITIAL_TEMP = VEGETATION_INITIAL_TEMP
SF%VEG_GROUND_TEMP   = VEGETATION_GROUND_TEMP
IF (ABS(VEGETATION_GROUND_TEMP+99._EB)>TWO_EPSILON_EB) SF%VEG_GROUND_ZERO_RAD = .FALSE.
SF%VEG_LOAD         = VEGETATION_LOAD
```

```
SF%FIRELINE_MLR_MAX = FIRELINE_MLR_MAX
!   SF%VEG_DEHYDRATION_RATE_MAX = SRF_VEG_DEHYDRATION_RATE_MAX
SF%VEG_PACKING   = VEGETATION_LOAD/VEGETATION_HEIGHT/VEGETATION_ELEMENT_DENSITY
SF%VEG_SVRATIO   = VEGETATION_SVRATIO
SF%VEG_KAPPA     = 0.25_EB*VEGETATION_SVRATIO*SF%VEG_PACKING
SF%NVEG_L        = INT(1. + VEGETATION_HEIGHT*3._EB*SF%VEG_KAPPA)
IF(VEGETATION_LAYERS > 0) SF%NVEG_L = VEGETATION_LAYERS
SF%VEG_DRAG_INI = VEGETATION_CDRAG*SF%VEG_PACKING*SF%VEG_SVRATIO
SF%VEG_LSET_IGNITE_T = VEGETATION_LSET_IGNITE_TIME
IF(SF%VEG_LSET_IGNITE_T > -1._EB) SF%VEG_LSET_SPREAD = .TRUE.
SF%VEG_LSET_QCON      = -VEG_LSET_QCON*1000._EB !convert from kW/m^2 to W/m^2
SF%VEG_LINEAR_DEGRAD      = VEGETATION_LINEAR_DEGRAD
SF%VEG_ARRHENIUS_DEGRAD  = VEGETATION_ARRHENIUS_DEGRAD
IF(VEGETATION_ARRHENIUS_DEGRAD) SF%VEG_LINEAR_DEGRAD = .FALSE.

ALLOCATE(SF%VEG_FUEL_FLUX_L(SF%NVEG_L),STAT=IZERO)
CALL ChkMemErr('READ_SURF','VEG_FUEL_FLUX_L',IZERO)
ALLOCATE(SF%VEG_MOIST_FLUX_L(SF%NVEG_L),STAT=IZERO)
CALL ChkMemErr('READ_SURF','VEG_MOIST_FLUX_L',IZERO)
ALLOCATE(SF%VEG_DIVQNET_L(SF%NVEG_L),STAT=IZERO)
CALL ChkMemErr('READ_SURF','VEG_DIVQNET',IZERO)

ALLOCATE(SF%VEG_FINCM_RADFCT_L(0:SF%NVEG_L),STAT=IZERO) !add index for mult veg
CALL ChkMemErr('READ_SURF','VEG_FINCM_RADFCT_L',IZERO)
ALLOCATE(SF%VEG_FINCP_RADFCT_L(0:SF%NVEG_L),STAT=IZERO)
CALL ChkMemErr('READ','VEG_FINCP_RADFCT_L',IZERO)

ALLOCATE(SF%VEG_SEMISSP_RADFCT_L(0:SF%NVEG_L,0:SF%NVEG_L),STAT=IZERO) !add index for mult veg
CALL ChkMemErr('READ','VEG_SEMISSP_RADFCT_L',IZERO)
ALLOCATE(SF%VEG_SEMISSM_RADFCT_L(0:SF%NVEG_L,0:SF%NVEG_L),STAT=IZERO)
CALL ChkMemErr('READ','VEG_SEMISSM_RADFCT_L',IZERO)

! If a RADIUS is specified, consider it the same as THICKNESS(1)

IF (RADIUS>0._EB) THICKNESS(1) = RADIUS

! Check SURF parameters for potential problems

LAYER_LOOP: DO IL=1,MAX_LAYERS
IF ((ADIABATIC.OR.NET_HEAT_FLUX<1.E12_EB.OR.ABS(CONVECTIVE_HEAT_FLUX)>TWO_EPSILON_EB.OR.TMP_FRONT>-TMPM) &
.AND. MATL_ID(IL,1)/='null') THEN
WRITE(MESSAGE,'(A)') 'ERROR: SURF '//TRIM(SF%ID)//' cannot have a specified flux or temperature and a MATL_ID'
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF
IF (THICKNESS(IL)<=0._EB .AND. MATL_ID(IL,1)/='null') THEN
WRITE(MESSAGE,'(A,I0)') 'ERROR: SURF '//TRIM(SF%ID)// ' must have a specified THICKNESS for Layer ',IL
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF
ENDDO LAYER_LOOP

IF ((GEOMETRY=='CYLINDRICAL' .OR. GEOMETRY=='SPHERICAL') .AND. RADIUS<0._EB .AND. THICKNESS(1)<0._EB) THEN
WRITE(MESSAGE,'(A,A,A)') 'ERROR: SURF ',TRIM(SF%ID),' needs a RADIUS or THICKNESS'
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF

! Identify the default SURF

IF (ID==SURF_DEFAULT) DEFAULT_SURF_INDEX = N

! Pack SURF parameters into the SURFACE derived type

SF                          => SURFACE(N)
SF%ADIABATIC                = ADIABATIC
SF%AUTO_IGNITION_TEMPERATURE = AUTO_IGNITION_TEMPERATURE + TMPM
SELECT CASE(BACKING)
CASE('VOID')
SF%BACKING            = VOID
CASE('INSULATED')
SF%BACKING            = INSULATED
CASE('EXPOSED')
SF%BACKING            = EXPOSED
CASE DEFAULT
WRITE(MESSAGE,'(A)') 'ERROR: SURF '//TRIM(SF%ID)//', BACKING '//TRIM(BACKING)//' not recognized'
CALL SHUTDOWN(MESSAGE) ; RETURN
END SELECT
SF%BURN_AWAY                = BURN_AWAY
SF%CELL_SIZE_FACTOR         = CELL_SIZE_FACTOR
SF%CONVECTIVE_HEAT_FLUX = 1000._EB*CONVECTIVE_HEAT_FLUX
SF%C_FORCED_CONSTANT        = C_FORCED_CONSTANT
SF%C_FORCED_PR_EXP          = C_FORCED_PR_EXP
SF%C_FORCED_RE              = C_FORCED_RE
SF%C_FORCED_RE_EXP          = C_FORCED_RE_EXP
SF%C_HORIZONTAL             = C_HORIZONTAL
SF%C_VERTICAL               = C_VERTICAL
SF%CONV_LENGTH              = CONVECTION_LENGTH_SCALE
SF%CONVERT_VOLUME_TO_MASS = CONVERT_VOLUME_TO_MASS
IF (SF%CONVERT_VOLUME_TO_MASS .AND. TMP_FRONT<0._EB) THEN
WRITE(MESSAGE,'(A,A,A)') 'ERROR: SURF ',TRIM(SF%ID),' must specify TMP_FRONT for CONVERT_VOLUME_TO_MASS'
```

```fortran
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF
SF%NET_HEAT_FLUX          = 1000._EB*NET_HEAT_FLUX
SF%DUCT_PATH              = DUCT_PATH
SF%DT_INSERT              = DT_INSERT
SF%E_COEFFICIENT          = E_COEFFICIENT
SF%EMISSIVITY             = EMISSIVITY
SF%EMISSIVITY_BACK        = EMISSIVITY_BACK
SF%FIRE_SPREAD_RATE       = SPREAD_RATE / TIME_SHRINK_FACTOR
SF%FREE_SLIP              = FREE_SLIP
SF%NO_SLIP                = NO_SLIP
SF%FYI                    = FYI
SF%EXTERNAL_FLUX          = 1000._EB*EXTERNAL_FLUX
SF%INNER_RADIUS           = INNER_RADIUS
SELECT CASE(GEOMETRY)
CASE('CARTESIAN')
SF%GEOMETRY       = SURF_CARTESIAN
IF (SF%WIDTH>0._EB)                   SF%BACKING = INSULATED
CASE('CYLINDRICAL')
SF%GEOMETRY       = SURF_CYLINDRICAL
IF (SF%INNER_RADIUS<TWO_EPSILON_EB) SF%BACKING = INSULATED
CASE('SPHERICAL')
SF%GEOMETRY       = SURF_SPHERICAL
IF (SF%INNER_RADIUS<TWO_EPSILON_EB) SF%BACKING = INSULATED
CASE DEFAULT
WRITE(MESSAGE,'(A,A,A)') 'ERROR: SURF ',TRIM(SF%ID),' GEOMETRY not recognized'
CALL SHUTDOWN(MESSAGE) ; RETURN
END SELECT
SF%H_V                    = 1000._EB*HEAT_OF_VAPORIZATION
SELECT CASE(HEAT_TRANSFER_MODEL)
CASE DEFAULT
IF (ABS(C_FORCED_CONSTANT)>TWO_EPSILON_EB .OR. ABS(C_FORCED_RE)>TWO_EPSILON_EB) THEN
SF%HEAT_TRANSFER_MODEL = H_CUSTOM
ELSE
SF%HEAT_TRANSFER_MODEL = H_DEFAULT
ENDIF
CASE('LOGLAW','LOG LAW')
SF%HEAT_TRANSFER_MODEL = H_LOGLAW
CASE('ABL')
SF%HEAT_TRANSFER_MODEL = H_ABL
CASE('RAYLEIGH')
SF%HEAT_TRANSFER_MODEL = H_RAYLEIGH
CASE('YUAN')
SF%HEAT_TRANSFER_MODEL = H_YUAN
END SELECT
SF%HRRPUA                 = 1000._EB*HRRPUA
SF%MLRPUA                 = MLRPUA
SF%LAYER_DIVIDE           = LAYER_DIVIDE
IF (LEAK_PATH(2) < LEAK_PATH(1)) THEN
SF%LEAK_PATH(2)        = LEAK_PATH(1)
SF%LEAK_PATH(1)        = LEAK_PATH(2)
ELSE
SF%LEAK_PATH           = LEAK_PATH
ENDIF
SF%LENGTH                 = LENGTH
SF%MASS_FLUX              = 0._EB
SF%MASS_FLUX_VAR          = MASS_FLUX_VAR
SF%MASS_FRACTION          = 0._EB
SF%MAX_PRESSURE           = MAX_PRESSURE
SF%MINIMUM_LAYER_THICKNESS = MINIMUM_LAYER_THICKNESS
SF%N_CELLS_MAX            = N_CELLS_MAX
IF (ANY(N_LAYER_CELLS_MAX<1)) THEN
WRITE(MESSAGE,'(A,A,A)') 'ERROR: SURF ',TRIM(SF%ID),' N_LAYER_CELLS_MAX must be >= 2'
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF
SF%N_LAYER_CELLS_MAX      = N_LAYER_CELLS_MAX+1
SF%NRA                    = NUMBER_RADIATION_ANGLES
SF%NSB                    = NUMBER_SPECTRAL_BANDS
ALLOCATE(SF%PARTICLE_INSERT_CLOCK(NMESHES),STAT=IZERO)
CALL ChkMemErr('READ','PARTICLE_INSERT_CLOCK',IZERO)
SF%PARTICLE_INSERT_CLOCK = T_BEGIN
SF%NPPC                   = NPPC
SF%PARTICLE_MASS_FLUX     = PARTICLE_MASS_FLUX
SF%PART_ID                = PART_ID
SF%PLE                    = PLE
SELECT CASE (PROFILE)
CASE('null')
SF%PROFILE        = 0
CASE('ATMOSPHERIC')
SF%PROFILE        = ATMOSPHERIC_PROFILE
CASE('PARABOLIC')
SF%PROFILE        = PARABOLIC_PROFILE
CASE('BOUNDARY LAYER')
SF%PROFILE        = BOUNDARY_LAYER_PROFILE
CASE('RAMP')
SF%PROFILE        = RAMP_PROFILE
END SELECT
SF%RAMP_EF                = EXTERNAL_FLUX_RAMP
```

```
SF%RAMP_MF              = 'null'
SF%RAMP_Q               = RAMP_Q
SF%RAMP_V               = RAMP_V
SF%RAMP_T               = RAMP_T
SF%RAMP_T_I             = RAMP_T_I
SF%RAMP_PART            = RAMP_PART
SF%RAMP_V_X             = RAMP_V_X
SF%RAMP_V_Y             = RAMP_V_Y
SF%RAMP_V_Z             = RAMP_V_Z
IF (COLOR/='null') THEN
IF (COLOR=='INVISIBLE') THEN
TRANSPARENCY = 0._EB
ELSE
CALL COLOR2RGB(RGB,COLOR)
ENDIF
ENDIF
IF (ANY(RGB< 0)) THEN
RGB(1) = 255
RGB(2) = 204
RGB(3) = 102
ENDIF
IF (SF%ID=="OPEN") THEN
RGB(1) = 255
RGB(2) = 0
RGB(3) = 255
ENDIF
SF%RGB                  = RGB
SF%ROUGHNESS            = ROUGHNESS
SF%TRANSPARENCY         = TRANSPARENCY
SF%STRETCH_FACTOR       = STRETCH_FACTOR
SF%STRETCH_FACTOR       = MAX(1.0_EB,SF%STRETCH_FACTOR)
SF%TAU(TIME_HEAT)       = TAU_Q/TIME_SHRINK_FACTOR
SF%TAU(TIME_VELO)       = TAU_V/TIME_SHRINK_FACTOR
SF%TAU(TIME_TEMP)       = TAU_T/TIME_SHRINK_FACTOR
SF%TAU(TIME_EFLUX)      = TAU_EXTERNAL_FLUX/TIME_SHRINK_FACTOR
SF%TAU(TIME_PART)       = TAU_PART/TIME_SHRINK_FACTOR
SF%TEXTURE_MAP          = TEXTURE_MAP
SF%TEXTURE_WIDTH        = TEXTURE_WIDTH
SF%TEXTURE_HEIGHT       = TEXTURE_HEIGHT
SF%THERMALLY_THICK_HT3D = HT3D
SF%TMP_IGN              = IGNITION_TEMPERATURE + TMPM
SF%VEL                  = VEL
SF%VEL_BULK             = VEL_BULK
SF%VEL_GRAD             = VEL_GRAD
SF%VEL_T                = VEL_T
SF%VOLUME_FLOW          = VOLUME_FLOW
SF%WIDTH                = WIDTH
SF%Z0                   = Z0
SF%ZETA_FRONT           = ZETA_FRONT
IF (HEAT_TRANSFER_COEFFICIENT_BACK < 0._EB) HEAT_TRANSFER_COEFFICIENT_BACK=HEAT_TRANSFER_COEFFICIENT
SF%H_FIXED              = HEAT_TRANSFER_COEFFICIENT
SF%H_FIXED_B            = HEAT_TRANSFER_COEFFICIENT_BACK
SF%HM_FIXED             = MASS_TRANSFER_COEFFICIENT
SF%XYZ                  = XYZ

! Convert inflowing MASS_FLUX_TOTAL to MASS_FLUX

IF (MASS_FLUX_TOTAL >= 0._EB) THEN
SF%MASS_FLUX_TOTAL = MASS_FLUX_TOTAL
ELSE
WRITE (MESSAGE,'(A,A,A)') 'ERROR: Problem with SURF: ',TRIM(SF%ID), &
'. MASS_FLUX_TOTAL should only be used for outflow. Use MASS_FLUX for inflow'
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF

! Error checking

IF (DEFAULT .AND. &
(TRIM(ID)=='OPEN'                .OR. &
TRIM(ID)=='MIRROR'               .OR. &
TRIM(ID)=='INTERPOLATED'         .OR. &
TRIM(ID)=='PERIODIC'             .OR. &
TRIM(ID)=='HVAC'                 .OR. &
TRIM(ID)=='MASSLESS TRACER'      .OR. &
TRIM(ID)=='DROPLET'              .OR. &
TRIM(ID)=='VEGETATION'           .OR. &
TRIM(ID)=='EVACUATION_OUTFLOW'   .OR. &
TRIM(ID)=='MASSLESS TARGET')          ) THEN
WRITE (MESSAGE,'(A,A,A)') 'ERROR: Problem with SURF: ',TRIM(SF%ID),'. Cannot set predefined SURF as DEFAULT'
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF

IF (ABS(VOLUME_FLUX)>0._EB)  THEN
WRITE (MESSAGE,'(A,A,A)') 'ERROR: Problem with SURF: ',TRIM(SF%ID),'. VOLUME_FLUX is deprecated; use VOLUME_FLOW'
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF

IF (ANY(MASS_FLUX>0._EB) .AND. ANY(MASS_FRACTION>0._EB))  THEN
```

```fortran
WRITE(MESSAGE,'(A,A,A)') 'ERROR: Problem with SURF: ',TRIM(SF%ID),'. Cannot use both MASS_FLUX and MASS_FRACTION'
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF

IF (ANY(MASS_FLUX<0._EB) .OR. PARTICLE_MASS_FLUX<0._EB) THEN
WRITE(MESSAGE,'(A,A,A)') 'ERROR: Problem with SURF: ',TRIM(SF%ID),'. MASS_FLUX cannot be less than zero'
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF

IF (ANY(MASS_FLUX>0._EB) .AND. ABS(VEL)>TWO_EPSILON_EB) THEN
WRITE(MESSAGE,'(A,A,A)') 'ERROR: Problem with SURF: ',TRIM(SF%ID),'. Cannot use both MASS_FLUX and VEL'
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF

IF (ANY(MASS_FLUX>0._EB) .AND. ABS(MASS_FLUX_TOTAL)>TWO_EPSILON_EB) THEN
WRITE(MESSAGE,'(A,A,A)') 'ERROR: Problem with SURF: ',TRIM(SF%ID),'. Cannot use both MASS_FLUX and
    MASS_FLUX_TOTAL'
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF

IF (ABS(MASS_FLUX_TOTAL)>TWO_EPSILON_EB .AND. ABS(VEL)>TWO_EPSILON_EB) THEN
WRITE(MESSAGE,'(A,A,A)') 'ERROR: Problem with SURF: ',TRIM(SF%ID),'. Cannot use both MASS_FLUX_TOTAL and VEL'
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF

IF (ANY(MASS_FRACTION<0._EB)) THEN
WRITE(MESSAGE,'(A,A,A)') 'ERROR: Problem with SURF: ',TRIM(SF%ID),'. Cannot use a negative MASS_FRACTION'
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF

IF (ANY(MASS_FLUX/=0._EB) .OR. ANY(MASS_FRACTION>0._EB)) THEN
IF (SPEC_ID(1)=='null') THEN
WRITE(MESSAGE,'(A,A,A)') 'ERROR: Problem with SURF: ',TRIM(SF%ID),&
'. Must define SPEC_ID when using MASS_FLUX or MASS_FRACTION'
CALL SHUTDOWN(MESSAGE) ; RETURN
ELSE
DO NN=1,MAX_SPECIES
IF (TRIM(SPEC_ID(NN))=='null') EXIT
DO NNN=1,N_TRACKED_SPECIES
IF (TRIM(SPECIES_MIXTURE(NNN)%ID)==TRIM(SPEC_ID(NN))) THEN
SF%MASS_FLUX(NNN)      = MASS_FLUX(NN)
SF%MASS_FRACTION(NNN)= MASS_FRACTION(NN)
SF%TAU(NNN)            = TAU_MF(NN)/TIME_SHRINK_FACTOR
SF%RAMP_MF(NNN)        = RAMP_MF(NN)
EXIT
ENDIF
IF (NNN==N_TRACKED_SPECIES) THEN
WRITE(MESSAGE,'(A,A,A,A,A)') 'ERROR: Problem with SURF: ',TRIM(SF%ID),' SPEC ',TRIM(SPEC_ID(NN)),' not found'
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF
ENDDO
ENDDO
ENDIF
IF (SUM(SF%MASS_FRACTION) > TWO_EPSILON_EB) THEN
IF (SUM(SF%MASS_FRACTION) > 1._EB) THEN
WRITE(MESSAGE,'(A,A,A)') 'ERROR: Problem with SURF: ',TRIM(SF%ID),'. SUM(MASS_FRACTION) > 1'
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF
IF (SF%MASS_FRACTION(1) > 0._EB) THEN
WRITE(MESSAGE,'(A,A,A)') 'ERROR: Problem with SURF: ',TRIM(SF%ID), &
'. Cannot use background species with MASS_FRACTION.'
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF
SF%MASS_FRACTION(1) = 1._EB - SUM(SF%MASS_FRACTION(2:N_TRACKED_SPECIES))
ENDIF
ENDIF

IF (SF%HEAT_TRANSFER_MODEL==H_RAYLEIGH .AND. GRAV<TWO_EPSILON_EB) THEN
WRITE(MESSAGE,'(A,A,A)') 'ERROR: Problem with SURF: ',TRIM(SF%ID),'. Cannot use a RAYLEIGH model with GRAV=0'
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF

! Set various logical parameters

IF (ABS(SF%VEL_T(1))>TWO_EPSILON_EB .OR. ABS(SF%VEL_T(2))>TWO_EPSILON_EB) SF%SPECIFIED_TANGENTIAL_VELOCITY = .
    TRUE.

! Count the number of layers for the surface, and compile a LIST of all material names and indices

COMPUTE_EMISSIVITY      = .FALSE.
COMPUTE_EMISSIVITY_BACK = .FALSE.
IF (SF%EMISSIVITY      <0._EB) COMPUTE_EMISSIVITY      = .TRUE.
IF (SF%EMISSIVITY_BACK<0._EB) COMPUTE_EMISSIVITY_BACK = .TRUE.

SF%N_LAYERS = 0
N_LIST = 0
NAME_LIST = 'null'
```

```
SF%THICKNESS    = 0._EB
SF%LAYER_MATL_INDEX = 0
SF%LAYER_DENSITY    = 0._EB
INDEX_LIST = -1
ALLOCATE(SF%LAYER_THICKNESS(MAX_LAYERS))
SF%LAYER_THICKNESS = 0._EB
COUNT_LAYERS: DO NL=1,MAX_LAYERS
IF (THICKNESS(NL) < 0._EB) EXIT COUNT_LAYERS
SF%N_LAYERS = SF%N_LAYERS + 1
SF%LAYER_THICKNESS(NL) = THICKNESS(NL)
SF%N_LAYER_MATL(NL) = 0
EMISSIVITY = 0._EB
COUNT_LAYER_MATL: DO NN=1,MAX_MATERIALS
IF (MATL_ID(NL,NN) == 'null') CYCLE COUNT_LAYER_MATL
N_LIST = N_LIST + 1
NAME_LIST(N_LIST) = MATL_ID(NL,NN)
SF%N_LAYER_MATL(NL) = SF%N_LAYER_MATL(NL) + 1
SF%LAYER_MATL_NAME(NL,NN) = MATL_ID(NL,NN)
SF%LAYER_MATL_FRAC(NL,NN) = MATL_MASS_FRACTION(NL,NN)
DO NNN=1,N_MATL
IF (MATL_NAME(NNN)==NAME_LIST(N_LIST)) THEN
INDEX_LIST(N_LIST) = NNN
SF%LAYER_MATL_INDEX(NL,NN) = NNN
SF%LAYER_DENSITY(NL) = SF%LAYER_DENSITY(NL)+SF%LAYER_MATL_FRAC(NL,NN)/MATERIAL(NNN)%RHO_S
EMISSIVITY = EMISSIVITY + &
MATERIAL(NNN)%EMISSIVITY*SF%LAYER_MATL_FRAC(NL,NN)/MATERIAL(NNN)%RHO_S ! volume based
ENDIF
ENDDO
IF (INDEX_LIST(N_LIST)<0) THEN
WRITE(MESSAGE,'(A,A,A,A,A)') 'ERROR: MATL_ID ',TRIM(NAME_LIST(N_LIST)),', on SURF: ',TRIM(SF%ID),', does not
     exist'
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF
ENDDO COUNT_LAYER_MATL
IF (SF%LAYER_DENSITY(NL) > 0._EB) SF%LAYER_DENSITY(NL) = 1./SF%LAYER_DENSITY(NL)
IF (COMPUTE_EMISSIVITY_BACK)         SF%EMISSIVITY_BACK = EMISSIVITY*SF%LAYER_DENSITY(NL)
IF (NL==1 .AND. COMPUTE_EMISSIVITY) SF%EMISSIVITY      = EMISSIVITY*SF%LAYER_DENSITY(NL)
SF%THICKNESS = SF%THICKNESS + SF%LAYER_THICKNESS(NL)
ENDDO COUNT_LAYERS

! Set emissivity to default value if no other method applies.

IF (SF%EMISSIVITY      < 0._EB) SF%EMISSIVITY      = EMISSIVITY_DEFAULT
IF (SF%EMISSIVITY_BACK < 0._EB) SF%EMISSIVITY_BACK = EMISSIVITY_DEFAULT

! Define mass flux division point

IF (SF%LAYER_DIVIDE < 0._EB) THEN
IF (SF%BACKING==EXPOSED) THEN
SF%LAYER_DIVIDE = 0.5_EB * REAL(SF%N_LAYERS,EB)
ELSE
SF%LAYER_DIVIDE = REAL(SF%N_LAYERS+1)
ENDIF
ENDIF

! Add residue materials

DO I = 1,MAX_STEPS    ! repeat the residue loop to find chained reactions - allows MAX_STEPS steps
N_LIST2 = N_LIST
DO NN = 1,N_LIST2
ML=>MATERIAL(INDEX_LIST(NN))
DO NR=1,ML%N_REACTIONS
DO NNN=1,ML%N_RESIDUE(NR)
IF (ML%RESIDUE_MATL_NAME(NNN,NR) == 'null') CYCLE
IF (ANY(NAME_LIST==ML%RESIDUE_MATL_NAME(NNN,NR))) CYCLE
N_LIST = N_LIST + 1
IF (N_LIST>MAX_MATERIALS_TOTAL) THEN ; CALL SHUTDOWN('ERROR: Too many materials in the surface.') ; RETURN ;
     ENDIF
NAME_LIST (N_LIST) = ML%RESIDUE_MATL_NAME(NNN,NR)
INDEX_LIST(N_LIST) = ML%RESIDUE_MATL_INDEX(NNN,NR)
ENDDO
ENDDO
ENDDO
ENDDO

! Eliminate multiply counted materials from the list

N_LIST2 = N_LIST
WEED_MATL_LIST: DO NN=1,N_LIST
DO NNN=1,NN-1
IF (NAME_LIST(NNN)==NAME_LIST(NN)) THEN
NAME_LIST(NN)   = 'null'
INDEX_LIST(NN) = 0
N_LIST2 = N_LIST2-1
CYCLE WEED_MATL_LIST
ENDIF
ENDDO
ENDDO WEED_MATL_LIST
```

```fortran

! Allocate parameters indexed by layer

IF (TMP_FRONT >= -TMPM) TMPMIN = MIN(TMPMIN,TMP_FRONT+TMPM)
IF (TMP_BACK >= -TMPM) TMPMIN = MIN(TMPMIN,TMP_BACK+TMPM)
IF (ASSUMED_GAS_TEMPERATURE >= 0._EB) TMPMIN = MIN(TMPMIN,ASSUMED_GAS_TEMPERATURE)

SF%N_MATL = N_LIST2
SF%THERMALLY_THICK = .FALSE.
IF (SF%LAYER_DENSITY(1) > 0._EB) THEN
SF%THERMALLY_THICK = .TRUE.
SF%TMP_INNER = TMP_INNER + TMPM
IF (SF%TMP_INNER(1)>=0._EB) THEN
SF%TMP_FRONT = SF%TMP_INNER(1)
SF%TAU(TIME_TEMP) = 0._EB
ELSE
SF%TMP_FRONT = TMP_FRONT + TMPM
ENDIF
SF%TMP_BACK = TMP_BACK + TMPM
ALLOCATE(SF%N_LAYER_CELLS(SF%N_LAYERS))              ! The number of cells in each layer
ALLOCATE(SF%MIN_DIFFUSIVITY(SF%N_LAYERS))            ! The smallest diffusivity of materials in each layer
ALLOCATE(SF%MATL_NAME(SF%N_MATL))                    ! The list of all material names associated with the surface
ALLOCATE(SF%MATL_INDEX(SF%N_MATL))                   ! The list of all material indices associated with the surface
ALLOCATE(SF%RESIDUE_INDEX(SF%N_MATL,MAX_MATERIALS,MAX_REACTIONS))! Each material associated with the surface has a RESIDUE
ALLOCATE(SF%INTERNAL_HEAT_SOURCE(SF%N_LAYERS))      ! Volumetric source term set by the user
ELSE
SF%TMP_FRONT = TMP_FRONT + TMPM
SF%TMP_INNER = SF%TMP_FRONT
SF%TMP_BACK  = SF%TMP_FRONT
ENDIF
DO NN = 1,SF%N_LAYERS
IF (TMP_INNER(NN)>= -TMPM) TMPMIN = MIN(TMPMIN,TMP_INNER(NN)+TMPM)
ENDDO

! Store the names and indices of all materials associated with the surface

NNN = 0
DO NN=1,N_LIST
IF (NAME_LIST(NN)/='null') THEN
NNN = NNN + 1
SF%MATL_NAME(NNN)  = NAME_LIST(NN)
SF%MATL_INDEX(NNN) = INDEX_LIST(NN)
ENDIF
ENDDO

! Store the RESIDUE indices

DO NN=1,SF%N_MATL
ML => MATERIAL(SF%MATL_INDEX(NN))
IF (ML%N_REACTIONS>0 .AND. SF%TMP_IGN<5000._EB) THEN
WRITE(MESSAGE,'(A)') 'ERROR: SURF '//TRIM(SF%ID)//' cannot have a REACting MATL and IGNITION_TEMPERATURE'
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF
DO NR=1,ML%N_REACTIONS
DO NRM=1,ML%N_RESIDUE(NR)
DO NNN=1,SF%N_MATL
IF (ML%RESIDUE_MATL_INDEX(NRM,NR)==SF%MATL_INDEX(NNN)) SF%RESIDUE_INDEX(NN,NRM,NR) = NNN
ENDDO
ENDDO
ENDDO
ENDDO

! Specified source term

IF (SF%N_LAYERS>0 .AND. SF%LAYER_DENSITY(1)>0._EB) THEN
SF%INTERNAL_HEAT_SOURCE(1:SF%N_LAYERS) = 1000._EB*INTERNAL_HEAT_SOURCE(1:SF%N_LAYERS)
IF (MAXVAL(ABS(SF%INTERNAL_HEAT_SOURCE)) > TWO_EPSILON_EB) SF%SPECIFIED_HEAT_SOURCE = .TRUE.
ENDIF

! Thermal boundary conditions

IF (SF%ADIABATIC .AND. (SF%NET_HEAT_FLUX < 1.E12_EB .OR. ABS(SF%CONVECTIVE_HEAT_FLUX)>TWO_EPSILON_EB)) THEN
WRITE(MESSAGE,'(A)') 'ERROR: SURF '//TRIM(SF%ID)//&
' cannot have both ADIABATIC and NET_HEAT_FLUX or CONVECTIVE_HEAT_FLUX'
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF
IF (SF%NET_HEAT_FLUX < 1.E12_EB .AND. ABS(SF%CONVECTIVE_HEAT_FLUX)>TWO_EPSILON_EB) THEN
WRITE(MESSAGE,'(A)') 'ERROR: SURF '//TRIM(SF%ID)//' cannot have both NET_HEAT_FLUX or CONVECTIVE_HEAT_FLUX'
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF
IF (SF%THERMALLY_THICK_HT3D) THEN
IF ( SF%NET_HEAT_FLUX < 1.E12_EB .OR. ABS(SF%CONVECTIVE_HEAT_FLUX) > TWO_EPSILON_EB .OR. TMP_FRONT >= -TMPM )
    THEN
WRITE(MESSAGE,'(A)') 'ERROR: SURF '//TRIM(SF%ID)//' cannot have HT3D with specified TMP or FLUX bc'
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF
ENDIF
```

```

SF%THERMAL_BC_INDEX = SPECIFIED_TEMPERATURE
IF (SF%ADIABATIC) THEN
SF%THERMAL_BC_INDEX = NET_FLUX_BC
SF%NET_HEAT_FLUX = 0._EB
SF%EMISSIVITY = 1._EB
ENDIF
IF (SF%NET_HEAT_FLUX < 1.E12_EB)                   SF%THERMAL_BC_INDEX = NET_FLUX_BC
IF (ABS(SF%CONVECTIVE_HEAT_FLUX)>TWO_EPSILON_EB) SF%THERMAL_BC_INDEX = CONVECTIVE_FLUX_BC
IF (SF%THERMALLY_THICK)                            SF%THERMAL_BC_INDEX = THERMALLY_THICK
IF (SF%THERMALLY_THICK_HT3D)                       SF%THERMAL_BC_INDEX = THERMALLY_THICK_HT3D
IF (SF%PROFILE==ATMOSPHERIC_PROFILE)               SF%THERMAL_BC_INDEX = INFLOW_OUTFLOW
IF (SF%VEGETATION)                                 SF%THERMAL_BC_INDEX = VEG_BNDRY_FUEL

! Boundary layer profile

IF (SF%PROFILE==BOUNDARY_LAYER_PROFILE) THEN
IF ( ABS(VEL_BULK)>ABS(VEL) ) THEN
WRITE(MESSAGE,'(A)') 'ERROR: SURF '//TRIM(SF%ID)//' VEL_BULK invalid, must have VEL_BULK <= VEL'
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF
ENDIF

! Set convection length scale automatically for spheres. Set to 1 m for everything else.

IF (SF%CONV_LENGTH<0._EB) THEN
SELECT CASE(SF%GEOMETRY)
CASE(SURF_SPHERICAL)    ; SF%CONV_LENGTH = 2._EB*(SF%INNER_RADIUS+SF%THICKNESS)
CASE(SURF_CYLINDRICAL) ; SF%CONV_LENGTH = 2._EB*SF%THICKNESS
CASE DEFAULT            ; SF%CONV_LENGTH = 1._EB
END SELECT
ENDIF

! Determine if REIGNITION_MODEL is to be used

IF (SF%AUTO_IGNITION_TEMPERATURE < 1.E20_EB ) REIGNITION_MODEL = .TRUE.

! Ramps

IF (SF%RAMP_Q/='null') THEN
CALL GET_RAMP_INDEX(SF%RAMP_Q,'TIME',NR)
SF%RAMP_INDEX(TIME_HEAT) = NR
ELSE
IF (SF%TAU(TIME_HEAT) > 0._EB) SF%RAMP_INDEX(TIME_HEAT) = TANH_RAMP
IF (SF%TAU(TIME_HEAT) < 0._EB) SF%RAMP_INDEX(TIME_HEAT) = TSQR_RAMP
ENDIF

IF (SF%RAMP_V/='null') THEN
CALL GET_RAMP_INDEX(SF%RAMP_V,'TIME',NR)
SF%RAMP_INDEX(TIME_VELO) = NR
ELSE
IF (SF%TAU(TIME_VELO) > 0._EB) SF%RAMP_INDEX(TIME_VELO) = TANH_RAMP
IF (SF%TAU(TIME_VELO) < 0._EB) SF%RAMP_INDEX(TIME_VELO) = TSQR_RAMP
ENDIF

IF (SF%RAMP_T/='null') THEN
CALL GET_RAMP_INDEX(SF%RAMP_T,'TIME',NR)
SF%RAMP_INDEX(TIME_TEMP) = NR
ELSE
IF (SF%TAU(TIME_TEMP) > 0._EB) SF%RAMP_INDEX(TIME_TEMP) = TANH_RAMP
IF (SF%TAU(TIME_TEMP) < 0._EB) SF%RAMP_INDEX(TIME_TEMP) = TSQR_RAMP
ENDIF

IF (SF%RAMP_T_I/='null') THEN
CALL GET_RAMP_INDEX(SF%RAMP_T_I,'TIME',NR)
SF%RAMP_T_I_INDEX = NR
ENDIF

IF (SF%RAMP_EF/='null') THEN
CALL GET_RAMP_INDEX(SF%RAMP_EF,'TIME',NR)
SF%RAMP_INDEX(TIME_EFLUX) = NR
ELSE
IF (SF%TAU(TIME_EFLUX) > 0._EB) SF%RAMP_INDEX(TIME_EFLUX) = TANH_RAMP
IF (SF%TAU(TIME_EFLUX) < 0._EB) SF%RAMP_INDEX(TIME_EFLUX) = TSQR_RAMP
ENDIF

IF (SF%RAMP_PART/='null') THEN
CALL GET_RAMP_INDEX(SF%RAMP_PART,'TIME',NR)
SF%RAMP_INDEX(TIME_PART) = NR
ELSE
IF (SF%TAU(TIME_PART) > 0._EB) SF%RAMP_INDEX(TIME_PART) = TANH_RAMP
IF (SF%TAU(TIME_PART) < 0._EB) SF%RAMP_INDEX(TIME_PART) = TSQR_RAMP
ENDIF

IF (SF%RAMP_V_X/='null') THEN
CALL GET_RAMP_INDEX(SF%RAMP_V_X,'PROFILE',NR)
SF%RAMP_INDEX(VELO_PROF_X) = NR
ENDIF
```

```fortran

IF (SF%RAMP_V_Y/='null') THEN
CALL GET_RAMP_INDEX(SF%RAMP_V_Y,'PROFILE',NR)
SF%RAMP_INDEX(VELO_PROF_Y) = NR
ENDIF

IF (SF%RAMP_V_Z/='null') THEN
CALL GET_RAMP_INDEX(SF%RAMP_V_Z,'PROFILE',NR)
SF%RAMP_INDEX(VELO_PROF_Z) = NR
ENDIF

ENDDO READ_SURF_LOOP

CONTAINS

SUBROUTINE SET_SURF_DEFAULTS

ADIABATIC                  = .FALSE.
AUTO_IGNITION_TEMPERATURE  = 1.E20_EB
BACKING                    = 'EXPOSED'
BURN_AWAY                  = .FALSE.
CELL_SIZE_FACTOR           = 1.0
C_FORCED_CONSTANT          = 0._EB
C_FORCED_PR_EXP            = 0._EB
C_FORCED_RE                = 0._EB
C_FORCED_RE_EXP            = 0._EB
C_VERTICAL                 = 1.31_EB  ! Vertical free convection (Holman, Table 7-2)
C_HORIZONTAL               = 1.52_EB  ! Horizontal free convection
COLOR                      = 'null'
CONVECTIVE_HEAT_FLUX       = 0._EB
CONVECTION_LENGTH_SCALE    = -1._EB
CONVERT_VOLUME_TO_MASS     = .FALSE.
NET_HEAT_FLUX              = 1.E12_EB
DEFAULT                    = .FALSE.
DT_INSERT                  = 0.01_EB
DUCT_PATH                  = 0
E_COEFFICIENT              = 0._EB
EMISSIVITY                 = -1._EB
EMISSIVITY_DEFAULT         = 0.9_EB
EMISSIVITY_BACK            = -1._EB
EVAC_DEFAULT               = .FALSE.
EXTERNAL_FLUX              = 0._EB
EXTERNAL_FLUX_RAMP         = 'null'
FREE_SLIP                  = .FALSE.
NO_SLIP                    = .FALSE.
FYI                        = 'null'
GEOMETRY                   = 'CARTESIAN'
HEAT_OF_VAPORIZATION       = 0._EB
HEAT_TRANSFER_MODEL        = 'null'
HEAT_TRANSFER_COEFFICIENT  = -1._EB
HEAT_TRANSFER_COEFFICIENT_BACK = -1._EB
MASS_TRANSFER_COEFFICIENT  = -1._EB
HRRPUA                     = 0._EB
HT3D                       = .FALSE.
ID                         = 'null'
IGNITION_TEMPERATURE       = 5000._EB
INNER_RADIUS               =  0._EB
INTERNAL_HEAT_SOURCE       = 0._EB
LAYER_DIVIDE               = -1._EB
LEAK_PATH                  = -1
LENGTH                     = -1._EB
MASS_FLUX                  = 0._EB
MASS_FLUX_TOTAL            = 0._EB
MASS_FLUX_VAR              = -1._EB
MASS_FRACTION              = 0._EB
MATL_ID                    = 'null'
MATL_MASS_FRACTION         = 0._EB
MATL_MASS_FRACTION(:,1)    = 1._EB
MAX_PRESSURE               = 1.E12_EB
MINIMUM_LAYER_THICKNESS    = 1.E-6_EB
MLRPUA                     = 0._EB
N_CELLS_MAX                = 0
N_LAYER_CELLS_MAX          = 999
NPPC                       = 1
PARTICLE_MASS_FLUX         = 0._EB
PART_ID                    = 'null'
PLE                        = 0.3_EB
PROFILE                    = 'null'
RADIUS                     = -1._EB
RAMP_MF                    = 'null'
RAMP_Q                     = 'null'
RAMP_V                     = 'null'
RAMP_T                     = 'null'
RAMP_T_I                   = 'null'
RAMP_PART                  = 'null'
RAMP_V_X                   = 'null'
RAMP_V_Y                   = 'null'
RAMP_V_Z                   = 'null'
```

```fortran
RGB                     = -1
IF (LES) ROUGHNESS      = 0._EB !4.5E-5_EB  ! meters, commercial steel
IF (DNS) ROUGHNESS      = 0._EB
SPEC_ID                 = 'null'
SPREAD_RATE             = -1._EB
STRETCH_FACTOR          = 2._EB
TAU_MF                  = 1._EB
TAU_Q                   = 1._EB
TAU_V                   = 1._EB
TAU_T                   = 1._EB
TAU_PART                = 1._EB
TAU_EXTERNAL_FLUX       = 0.001_EB
TEXTURE_MAP             = 'null'
TEXTURE_WIDTH           = 1._EB
TEXTURE_HEIGHT          = 1._EB
TGA_ANALYSIS            = .FALSE.
THICKNESS               = -1._EB
TMP_BACK                = -TMPM-1._EB
TMP_FRONT               = -TMPM-1._EB
TMP_INNER               = -TMPM-1._EB
TRANSPARENCY            = 1._EB
VEL                     = 0._EB
VEL_BULK                = 0._EB
VEL_GRAD                = -999999._EB
VEL_T                   = 0._EB
VOLUME_FLUX             = 0._EB ! deprecated
VOLUME_FLOW             = 0._EB
WIDTH                   = -1._EB
XYZ                     = -1.E6_EB
Z0                      = 10._EB
ZETA_FRONT              = INITIAL_UNMIXED_FRACTION

VEGETATION                      = .FALSE.
VEGETATION_NO_BURN              = .FALSE.
VEGETATION_CDRAG                = 1.0_EB
VEGETATION_CHAR_FRACTION        = 0.20_EB
VEGETATION_ELEMENT_DENSITY      = 512._EB !kg/m^3
VEGETATION_HEIGHT               = 0.50_EB !m
VEGETATION_INITIAL_TEMP         = TMPA-TMPM
VEGETATION_GROUND_TEMP          = -99._EB
VEGETATION_LOAD                 = 0.30_EB !kg/m^2
FIRELINE_MLR_MAX                = 999. !kg/m/s w*R*(1-ChiChar)
!SRF_VEG_DEHYDRATION_RATE_MAX = 999. !kg/m^2/s
VEGETATION_LAYERS               = 0
VEGETATION_MOISTURE             = 0.06_EB
VEGETATION_SVRATIO              = 12000_EB !1/m
VEGETATION_LSET_IGNITE_TIME     = -1._EB
VEGETATION_LINEAR_DEGRAD        = .TRUE.
VEGETATION_ARRHENIUS_DEGRAD     = .FALSE.
VEG_LSET_ROS_HEAD               = 0.0_EB
VEG_LSET_ELLIPSE_HEAD           = 0.0_EB
VEG_LSET_ROS_FLANK              = 0.0_EB
VEG_LSET_ROS_BACK               = 0.0_EB
VEG_LEVEL_SET_SPREAD            = .FALSE.
VEG_LSET_WIND_EXP               = 1.0_EB
VEG_LSET_ELLIPSE                = .FALSE.
VEG_LSET_TAN2                   = .FALSE.
VEG_LSET_HT                     = 0.0_EB
VEG_LSET_BETA                   = 0.0_EB
VEG_LSET_SIGMA                  = 0.0_EB
VEG_LSET_QCON                   = 0.0_EB

END SUBROUTINE SET_SURF_DEFAULTS

END SUBROUTINE READ_SURF


SUBROUTINE PROC_SURF_1

! Go through the SURF types and process

USE MATH_FUNCTIONS, ONLY : GET_RAMP_INDEX
INTEGER :: N,NSPC,NR,ILPC
TYPE (LAGRANGIAN_PARTICLE_CLASS_TYPE), POINTER :: LPC=>NULL()

PROCESS_SURF_LOOP: DO N=0,N_SURF

SF => SURFACE(N)

! Get ramps for the surface mass fraction and flux

DO NSPC=1,N_TRACKED_SPECIES
IF (TRIM(SF%RAMP_MF(NSPC))/='null') THEN
CALL GET_RAMP_INDEX(SF%RAMP_MF(NSPC),'TIME',NR)
SF%RAMP_INDEX(NSPC) = NR
ELSE
IF (SF%TAU(NSPC) > 0._EB) SF%RAMP_INDEX(NSPC) = TANH_RAMP
IF (SF%TAU(NSPC) < 0._EB) SF%RAMP_INDEX(NSPC) = TSQR_RAMP
```

```
ENDIF
ENDDO

! Look for particle classes that use SURF for property info

DO ILPC=1,N_LAGRANGIAN_CLASSES

LPC=>LAGRANGIAN_PARTICLE_CLASS(ILPC)

IF (LPC%SURF_ID==SF%ID) THEN
LPC%SURF_INDEX = N

IF (.NOT.LPC%SOLID_PARTICLE) CYCLE
IF (LPC%DRAG_LAW==SCREEN_DRAG) CYCLE
SELECT CASE (SF%GEOMETRY)
CASE(SURF_CARTESIAN)
IF (SF%THICKNESS<=0._EB) THEN
WRITE(MESSAGE,'(A,A,A)') 'ERROR: SURF ',TRIM(SF%ID),' needs a THICKNESS'
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF
IF (.NOT. LPC%DRAG_LAW==POROUS_DRAG) THEN
IF (SF%LENGTH<=0._EB) THEN
WRITE(MESSAGE,'(A,A,A)') 'ERROR: SURF ',TRIM(SF%ID),' needs a LENGTH'
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF
IF (SF%WIDTH<=0._EB) THEN
WRITE(MESSAGE,'(A,A,A)') 'ERROR: SURF ',TRIM(SF%ID),' needs a WIDTH'
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF
ENDIF
CASE(SURF_CYLINDRICAL)
IF (.NOT. LPC%DRAG_LAW==POROUS_DRAG) THEN
IF (SF%LENGTH <0._EB) THEN
WRITE(MESSAGE,'(A,A,A)') 'ERROR: SURF ',TRIM(SF%ID),' needs a LENGTH'
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF
ENDIF
END SELECT
ENDIF
ENDDO

ENDDO PROCESS_SURF_LOOP

! If a particle class uses a SURF line, make sure the SURF ID exists

DO ILPC=1,N_LAGRANGIAN_CLASSES
LPC=>LAGRANGIAN_PARTICLE_CLASS(ILPC)
IF (LPC%SURF_INDEX<0) THEN
WRITE(MESSAGE,'(A,A,A)') 'ERROR: SURF ',TRIM(LPC%SURF_ID),' not found'
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF
ENDDO

END SUBROUTINE PROC_SURF_1


SUBROUTINE PROC_SURF_2

! Go through the SURF types and process

INTEGER :: ILPC,N,NN,NNN,NL
REAL(EB) :: ADJUSTED_LAYER_DENSITY,R_L(0:MAX_LAYERS)
INTEGER  :: IVEG_L,IIVEG_L,I_FUEL,I_GRAD
REAL(EB) :: DETA_VEG,DZVEG_L,ETA_H,ETAFM_VEG,ETAFP_VEG
LOGICAL :: BURNING,BLOWING,SUCKING

TYPE(LAGRANGIAN_PARTICLE_CLASS_TYPE), POINTER :: LPC=>NULL()

PROCESS_SURF_LOOP: DO N=0,N_SURF

SF => SURFACE(N)
IF (SF%THERMALLY_THICK) ML => MATERIAL(SF%LAYER_MATL_INDEX(1,1))

SELECT CASE(SF%GEOMETRY)
CASE(SURF_CARTESIAN)     ; I_GRAD = 1
CASE(SURF_CYLINDRICAL)   ; I_GRAD = 2
CASE(SURF_SPHERICAL)     ; I_GRAD = 3
END SELECT

! Particle Information

SF%PART_INDEX = 0
IF (SF%PART_ID/='null') THEN
DO ILPC=1,N_LAGRANGIAN_CLASSES
LPC=>LAGRANGIAN_PARTICLE_CLASS(ILPC)
IF (LPC%ID==SF%PART_ID) SF%PART_INDEX = ILPC
ENDDO
IF (SF%PART_INDEX==0) THEN
```

```
WRITE(MESSAGE,'(A)') 'ERROR: PART_ID '//TRIM(SF%PART_ID)//' not found'
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF
PARTICLE_FILE=.TRUE.
ENDIF

! Determine if surface has internal radiation

SF%INTERNAL_RADIATION = .FALSE.
DO NL=1,SF%N_LAYERS
DO NN =1,SF%N_LAYER_MATL(NL)
ML => MATERIAL(SF%LAYER_MATL_INDEX(NL,NN))
IF (ML%KAPPA_S<5.0E4_EB) SF%INTERNAL_RADIATION = .TRUE.
ENDDO
ENDDO

! In case of internal radiation, do not allow zero-emissivity

IF (SF%INTERNAL_RADIATION) THEN
DO NL=1,SF%N_LAYERS
DO NN =1,SF%N_LAYER_MATL(NL)
ML => MATERIAL(SF%LAYER_MATL_INDEX(NL,NN))
IF (ML%EMISSIVITY == 0._EB) THEN
WRITE(MESSAGE,'(A)') 'ERROR: Zero emissivity of MATL '//TRIM(MATL_NAME(SF%LAYER_MATL_INDEX(NL,NN)))// &
' is inconsistent with internal radiation in SURF '//TRIM(SF%ID)//'.'
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF
ENDDO
ENDDO
ENDIF

! Determine if the surface is combustible/burning

SF%PYROLYSIS_MODEL = PYROLYSIS_NONE
BURNING = .FALSE.
DO NL=1,SF%N_LAYERS
DO NN=1,SF%N_LAYER_MATL(NL)
NNN = SF%LAYER_MATL_INDEX(NL,NN)
ML => MATERIAL(NNN)
IF (ML%PYROLYSIS_MODEL/=PYROLYSIS_NONE) THEN
SF%PYROLYSIS_MODEL = PYROLYSIS_PREDICTED
SF%STRETCH_FACTOR(NL) = 1._EB
IF (N_REACTIONS>0) THEN
IF (REACTION(1)%FUEL_SMIX_INDEX>=0) THEN
IF (ANY(ML%NU_SPEC(REACTION(1)%FUEL_SMIX_INDEX,:)>0._EB)) THEN
BURNING = .TRUE.
SF%TAU(TIME_HEAT) = 0._EB
ENDIF
ENDIF
ENDIF
ENDIF
ENDDO
ENDDO

IF (SF%HRRPUA>0._EB .OR. SF%MLRPUA>0._EB) THEN
IF (SF%PYROLYSIS_MODEL==PYROLYSIS_PREDICTED) THEN
WRITE(MESSAGE,'(A)') 'ERROR: SURF '//TRIM(SF%ID)//' has a specified HRRPUA or MLRPUA plus another pyrolysis model'
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF
IF (N_REACTIONS > 1) THEN
WRITE(MESSAGE,'(A)') 'ERROR: SURF '//TRIM(SF%ID)//' has HRRPUA or MLRPUA set and there is more than one reaction'
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF
BURNING = .TRUE.
SF%PYROLYSIS_MODEL = PYROLYSIS_SPECIFIED
ENDIF

IF (BURNING .AND. N_REACTIONS==0) THEN
WRITE(MESSAGE,'(A)') 'ERROR: SURF '//TRIM(SF%ID)//' indicates burning, but there is no REAC line'
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF

! Make decisions based on whether there is forced ventilation at the surface

BLOWING = .FALSE.
SUCKING = .FALSE.
IF (SF%VEL<0._EB .OR. SF%VOLUME_FLOW<0._EB .OR. SF%MASS_FLUX_TOTAL < 0._EB) BLOWING = .TRUE.
IF (SF%VEL>0._EB .OR. SF%VOLUME_FLOW>0._EB .OR. SF%MASS_FLUX_TOTAL > 0._EB) SUCKING = .TRUE.
IF (BLOWING .OR. SUCKING) SF%SPECIFIED_NORMAL_VELOCITY = .TRUE.
IF (SUCKING) SF%FREE_SLIP = .TRUE.

IF (BURNING .AND. (BLOWING .OR. SUCKING)) THEN
WRITE(MESSAGE,'(A)') 'ERROR: SURF '//TRIM(SF%ID)//' cannot have a specified velocity or volume flux'
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF

! Neumann for normal component of velocity
```

```
IF (SF%VEL_GRAD > -999998._EB) THEN
SF%SPECIFIED_NORMAL_GRADIENT = .TRUE.
SF%SPECIFIED_NORMAL_VELOCITY = .FALSE.
ENDIF

! Set predefined HRRPUA

BURNING_IF: IF (BURNING .AND. .NOT.ALL(EVACUATION_ONLY)) THEN
IF (SF%HRRPUA>0._EB) THEN
RN => REACTION(1)
SF%MASS_FLUX(RN%FUEL_SMIX_INDEX) = SF%HRRPUA/RN%HOC_COMPLETE
ENDIF
IF (SF%MLRPUA>0._EB) THEN
RN => REACTION(1)
SF%MASS_FLUX(RN%FUEL_SMIX_INDEX) = SF%MLRPUA
ENDIF
I_FUEL = REACTION(1)%FUEL_SMIX_INDEX
IF (SF%N_LAYERS > 0 .AND. SF%THERMALLY_THICK) THEN
SF%ADJUST_BURN_RATE(I_FUEL) = MATERIAL(SF%MATL_INDEX(1))%ADJUST_BURN_RATE(I_FUEL,1)
SF%MASS_FLUX(I_FUEL) = SF%MASS_FLUX(I_FUEL)/SF%ADJUST_BURN_RATE(I_FUEL) ! This is the true burning rate of the fuel.
ENDIF
SF%TAU(I_FUEL)          = SF%TAU(TIME_HEAT)
SF%RAMP_MF(I_FUEL)      = SF%RAMP_Q
SF%RAMP_INDEX(I_FUEL) = SF%RAMP_INDEX(TIME_HEAT)
ENDIF BURNING_IF

! Compute surface density

SF%SURFACE_DENSITY = 0._EB
R_L(0) = SF%THICKNESS
DO NL=1,SF%N_LAYERS
ADJUSTED_LAYER_DENSITY = 0._EB
MATL_LOOP:DO NN=1,SF%N_LAYER_MATL(NL)
NNN = SF%LAYER_MATL_INDEX(NL,NN)
ML => MATERIAL(NNN)
ADJUSTED_LAYER_DENSITY = ADJUSTED_LAYER_DENSITY + SF%LAYER_MATL_FRAC(NL,NN)/ML%RHO_S
ENDDO MATL_LOOP
IF (ADJUSTED_LAYER_DENSITY > 0._EB) ADJUSTED_LAYER_DENSITY = 1./ADJUSTED_LAYER_DENSITY
R_L(NL) = R_L(NL-1)-SF%LAYER_THICKNESS(NL)
SF%SURFACE_DENSITY = SF%SURFACE_DENSITY + ADJUSTED_LAYER_DENSITY * &
(R_L(NL-1)**I_GRAD-R_L(NL)**I_GRAD)/(REAL(I_GRAD,EB)*SF%THICKNESS**(I_GRAD-1))
ENDDO

IF ((ABS(SF%SURFACE_DENSITY) <= TWO_EPSILON_EB) .AND. SF%BURN_AWAY) THEN
WRITE(MESSAGE,'(A,A,A)') 'WARNING: SURF ',TRIM(SF%ID),' has BURN_AWAY set but zero combustible density'
IF (MYID==0) WRITE(LU_ERR,'(A)') TRIM(MESSAGE)
ENDIF

! Ignition Time

SF%T_IGN = T_BEGIN
IF (SF%TMP_IGN<5000._EB)                         SF%T_IGN = HUGE(T_END)
IF (SF%PYROLYSIS_MODEL==PYROLYSIS_PREDICTED) SF%T_IGN = HUGE(T_END)

! Species Arrays and Method of Mass Transfer (SPECIES_BC_INDEX)

SF%SPECIES_BC_INDEX = NO_MASS_FLUX

IF (ANY(SF%MASS_FRACTION>0._EB) .AND. (ANY(ABS(SF%MASS_FLUX)>TWO_EPSILON_EB) .OR. SF%PYROLYSIS_MODEL/=PYROLYSIS_NONE)) THEN
WRITE(MESSAGE,'(A)') 'ERROR: SURF '//TRIM(SF%ID)//' cannot specify mass fraction with mass flux and/or pyrolysis'
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF
IF (ANY(SF%MASS_FRACTION>0._EB) .AND. SUCKING) THEN
WRITE(MESSAGE,'(A)') 'ERROR: SURF '//TRIM(SF%ID)//' cannot specify both mass fraction and outflow velocity'
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF
IF (ANY(SF%LEAK_PATH>=0) .AND. (BLOWING .OR. SUCKING .OR. SF%PYROLYSIS_MODEL/= PYROLYSIS_NONE)) THEN
WRITE(MESSAGE,'(A)') 'ERROR: SURF '//TRIM(SF%ID)//' cannot leak and specify flow or pyrolysis at the same time'
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF
IF (ANY(ABS(SF%MASS_FLUX)>TWO_EPSILON_EB) .AND. (BLOWING .OR. SUCKING)) THEN
WRITE(MESSAGE,'(A)') 'ERROR: SURF '//TRIM(SF%ID)//' cannot have both a mass flux and specified velocity'
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF

IF (BLOWING .OR. SUCKING)           SF%SPECIES_BC_INDEX = SPECIFIED_MASS_FRACTION
IF (ANY(SF%MASS_FRACTION>0._EB)) SF%SPECIES_BC_INDEX = SPECIFIED_MASS_FRACTION
IF (ANY(ABS(SF%MASS_FLUX)>TWO_EPSILON_EB) .OR. &
SF%PYROLYSIS_MODEL==PYROLYSIS_PREDICTED) SF%SPECIES_BC_INDEX = SPECIFIED_MASS_FLUX

IF (SF%SPECIES_BC_INDEX==SPECIFIED_MASS_FRACTION) THEN
IF (ALL(ABS(SF%MASS_FRACTION)< TWO_EPSILON_EB)) &
SF%MASS_FRACTION(1:N_TRACKED_SPECIES) = SPECIES_MIXTURE(1:N_TRACKED_SPECIES)%ZZ0
ENDIF
```

```
! Boundary fuel model for vegetation

IF (SF%VEGETATION) SF%SPECIES_BC_INDEX = SPECIFIED_MASS_FLUX

! Texture map info

SF%SURF_TYPE = 0
IF (SF%TEXTURE_MAP/='null') SF%SURF_TYPE = 1

! Set BCs for various boundary types

SF%VELOCITY_BC_INDEX = WALL_MODEL_BC
IF (DNS) SF%VELOCITY_BC_INDEX = NO_SLIP_BC
IF (SF%FREE_SLIP) SF%VELOCITY_BC_INDEX = FREE_SLIP_BC
IF (SF%NO_SLIP)   SF%VELOCITY_BC_INDEX = NO_SLIP_BC

IF (N==OPEN_SURF_INDEX) THEN
SF%THERMAL_BC_INDEX = INFLOW_OUTFLOW
SF%SPECIES_BC_INDEX = INFLOW_OUTFLOW_MASS_FLUX
SF%VELOCITY_BC_INDEX = FREE_SLIP_BC
SF%SURF_TYPE = 2
SF%EMISSIVITY = 1._EB
ENDIF
IF (N==MIRROR_SURF_INDEX) THEN
SF%THERMAL_BC_INDEX = NO_CONVECTION
SF%SPECIES_BC_INDEX = NO_MASS_FLUX
SF%VELOCITY_BC_INDEX = FREE_SLIP_BC
SF%SURF_TYPE = -2
SF%EMISSIVITY = 0._EB
ENDIF
IF (N==INTERPOLATED_SURF_INDEX) THEN
SF%THERMAL_BC_INDEX = INTERPOLATED_BC
SF%SPECIES_BC_INDEX = INTERPOLATED_BC
SF%VELOCITY_BC_INDEX = INTERPOLATED_VELOCITY_BC
ENDIF
IF (N==PERIODIC_SURF_INDEX) THEN
SF%THERMAL_BC_INDEX = INTERPOLATED_BC
SF%SPECIES_BC_INDEX = INTERPOLATED_BC
SF%VELOCITY_BC_INDEX = INTERPOLATED_VELOCITY_BC
ENDIF
IF (N==PERIODIC_WIND_SURF_INDEX) THEN
SF%THERMAL_BC_INDEX = INFLOW_OUTFLOW
SF%SPECIES_BC_INDEX = INFLOW_OUTFLOW_MASS_FLUX
SF%VELOCITY_BC_INDEX = INTERPOLATED_VELOCITY_BC
ENDIF
IF (N==HVAC_SURF_INDEX) THEN
SF%THERMAL_BC_INDEX = HVAC_BOUNDARY
SF%SPECIES_BC_INDEX = HVAC_BOUNDARY
ENDIF
IF (N==MASSLESS_TRACER_SURF_INDEX) THEN
SF%NRA = 1
SF%NSB = 1
ENDIF
IF (N==DROPLET_SURF_INDEX) THEN
SF%NRA = 1
SF%NSB = 1
ENDIF
IF (N==VEGETATION_SURF_INDEX) THEN
SF%NRA = 1
SF%NSB = 1
ENDIF
IF (N==EVACUATION_SURF_INDEX) THEN
SF%THERMAL_BC_INDEX = INFLOW_OUTFLOW
SF%SPECIES_BC_INDEX = SPECIFIED_MASS_FRACTION
SF%SPECIFIED_NORMAL_VELOCITY = .TRUE.
SF%FREE_SLIP = .TRUE.
SF%VELOCITY_BC_INDEX = FREE_SLIP_BC
SF%VEL                     = +0.000001_EB ! VEL
SF%TAU(TIME_VELO)          = 0.1_EB ! TAU_V
SF%RAMP_INDEX(TIME_VELO) = TANH_RAMP
ENDIF
IF (N==MASSLESS_TARGET_SURF_INDEX) THEN
SF%EMISSIVITY = 1._EB
ENDIF

! Do not allow N_LAYERS or N_CELLS_INI to be zero

IF (.NOT.SF%THERMALLY_THICK) THEN
SF%N_LAYERS      = 1
SF%N_CELLS_MAX   = 1
SF%N_CELLS_INI   = 1
SF%N_MATL    = 1
ALLOCATE(SF%N_LAYER_CELLS(SF%N_LAYERS))
ALLOCATE(SF%X_S(0:SF%N_CELLS_MAX))
SF%X_S(0) = 0._EB
SF%X_S(1) = SF%THICKNESS
ALLOCATE(SF%RHO_0(0:SF%N_CELLS_MAX+1,SF%N_MATL))
SF%RHO_0 = 0._EB
```

```
SF%TMP_INNER(:) = TMPA
ENDIF

! Boundary surface vegetation

DZVEG_L   = SF%VEG_HEIGHT/REAL(SF%NVEG_L,EB)
DETA_VEG = SF%VEG_KAPPA*DZVEG_L

! Factors for computing decay of +/- incident fluxes

SF%VEG_FINCM_RADFCT_L(:) =   0.0_EB
SF%VEG_FINCP_RADFCT_L(:) =   0.0_EB
ETA_H = SF%VEG_KAPPA*SF%VEG_HEIGHT
DO IVEG_L = 0,SF%NVEG_L
ETAFM_VEG = IVEG_L*DETA_VEG
ETAFP_VEG = ETA_H - ETAFM_VEG
SF%VEG_FINCM_RADFCT_L(IVEG_L) = EXP(-ETAFM_VEG)
SF%VEG_FINCP_RADFCT_L(IVEG_L) = EXP(-ETAFP_VEG)
ENDDO

!  Integrand for computing +/- self emission fluxes

SF%VEG_SEMISSP_RADFCT_L(:,:) = 0.0_EB
SF%VEG_SEMISSM_RADFCT_L(:,:) = 0.0_EB
! q+
DO IIVEG_L = 0,SF%NVEG_L !grid coordinate
DO IVEG_L = IIVEG_L,SF%NVEG_L !integrand index
ETAFM_VEG = (IVEG_L-IIVEG_L)*DETA_VEG
ETAFP_VEG = ETAFM_VEG + DETA_VEG
SF%VEG_SEMISSP_RADFCT_L(IVEG_L,IIVEG_L) = EXP(-ETAFM_VEG) - EXP(-ETAFP_VEG)
ENDDO
ENDDO
! q-
DO IIVEG_L = 0,SF%NVEG_L
DO IVEG_L = 1,IIVEG_L
ETAFM_VEG = (IIVEG_L-IVEG_L)*DETA_VEG
ETAFP_VEG = ETAFM_VEG + DETA_VEG
SF%VEG_SEMISSM_RADFCT_L(IVEG_L,IIVEG_L) = EXP(-ETAFM_VEG) - EXP(-ETAFP_VEG)
ENDDO
ENDDO

ENDDO PROCESS_SURF_LOOP

END SUBROUTINE PROC_SURF_2



SUBROUTINE PROC_WALL

! Set up 1-D grids and arrays for thermally-thick calcs

USE GEOMETRY_FUNCTIONS
USE MATH_FUNCTIONS, ONLY: EVALUATE_RAMP

INTEGER :: SURF_INDEX,N,NL,II,IL,NN,N_CELLS_MAX
REAL(EB) :: K_S_0,C_S_0,SMALLEST_CELL_SIZE(MAX_LAYERS),SWELL_RATIO,DENSITY_MAX,DENSITY_MIN

! Calculate ambient temperature thermal DIFFUSIVITY for each MATERIAL, to be used in determining number of solid
     cells

DO N=1,N_MATL
ML => MATERIAL(N)
IF (ML%K_S>0._EB) THEN
K_S_0 = ML%K_S
ELSE
K_S_0 = EVALUATE_RAMP(TMPA,0._EB,-NINT(ML%K_S))
ENDIF
IF (ML%C_S>0._EB) THEN
C_S_0 = ML%C_S
ELSE
C_S_0 = EVALUATE_RAMP(TMPA,0._EB,-NINT(ML%C_S))*1000._EB
ENDIF
ML%DIFFUSIVITY = K_S_0/(C_S_0*ML%RHO_S)
ENDDO

NWP_MAX = 0  ! For some utility arrays, need to know the greatest number of points of all surface types

! Loop through all surfaces, looking for those that are thermally-thick (have layers).
! Compute smallest cell size for each layer such that internal cells double in size.
! Each layer should have an odd number of cells.

SURF_GRID_LOOP: DO SURF_INDEX=0,N_SURF

SF => SURFACE(SURF_INDEX)
IF (SF%THERMAL_BC_INDEX /= THERMALLY_THICK) CYCLE SURF_GRID_LOOP

! Compute number of points per layer, and then sum up to get total points for the surface

```

```
SF%N_CELLS_INI = 0
N_CELLS_MAX    = 0

LAYER_LOOP: DO NL=1,SF%N_LAYERS

SF%MIN_DIFFUSIVITY(NL) = 1000000._EB
DO N = 1,SF%N_LAYER_MATL(NL)
ML => MATERIAL(SF%LAYER_MATL_INDEX(NL,N))
SF%MIN_DIFFUSIVITY(NL) = MIN(SF%MIN_DIFFUSIVITY(NL),ML%DIFFUSIVITY)
ENDDO

DENSITY_MAX = 0._EB
DENSITY_MIN = 10000000._EB
DO N = 1,SF%N_MATL
ML => MATERIAL(SF%MATL_INDEX(N))
DO NN = 1,SF%N_LAYER_MATL(NL)
IF ((ML%PYROLYSIS_MODEL==PYROLYSIS_SOLID.OR.ML%PYROLYSIS_MODEL==PYROLYSIS_VEGETATION) .AND. &
SF%LAYER_MATL_INDEX(NL,NN)==SF%MATL_INDEX(N)) THEN
DENSITY_MAX = MAX(DENSITY_MAX,SF%LAYER_MATL_FRAC(NL,NN)*SF%LAYER_DENSITY(NL))
ENDIF
ENDDO
DENSITY_MIN = MIN(DENSITY_MIN,ML%RHO_S)
ENDDO

SWELL_RATIO = 1._EB
IF (SF%PYROLYSIS_MODEL==PYROLYSIS_PREDICTED .AND. DENSITY_MIN>TWO_EPSILON_EB) SWELL_RATIO = DENSITY_MAX/
       DENSITY_MIN
SWELL_RATIO = MAX(1.0_EB, SWELL_RATIO)

! Get highest possible number of cells for this layer

CALL GET_N_LAYER_CELLS(SF%MIN_DIFFUSIVITY(NL),SWELL_RATIO*SF%LAYER_THICKNESS(NL),SF%STRETCH_FACTOR(NL), &
SF%CELL_SIZE_FACTOR,SF%N_LAYER_CELLS_MAX(NL),SF%N_LAYER_CELLS(NL),SMALLEST_CELL_SIZE(NL))
N_CELLS_MAX = N_CELLS_MAX + SF%N_LAYER_CELLS(NL)

! Get initial number of cells for this layer

CALL GET_N_LAYER_CELLS(SF%MIN_DIFFUSIVITY(NL),SF%LAYER_THICKNESS(NL),SF%STRETCH_FACTOR(NL), &
SF%CELL_SIZE_FACTOR,SF%N_LAYER_CELLS_MAX(NL),SF%N_LAYER_CELLS(NL),SMALLEST_CELL_SIZE(NL))
SF%N_CELLS_INI= SF%N_CELLS_INI + SF%N_LAYER_CELLS(NL)

ENDDO LAYER_LOOP

IF (SF%N_CELLS_MAX==0) SF%N_CELLS_MAX = N_CELLS_MAX

! Allocate arrays to hold x_s, 1/dx_s (center to center, RDXN), 1/dx_s (edge to edge, RDX)

NWP_MAX = MAX(NWP_MAX,SF%N_CELLS_MAX)
ALLOCATE(SF%DX(1:SF%N_CELLS_MAX))
ALLOCATE(SF%RDX(0:SF%N_CELLS_MAX+1))
ALLOCATE(SF%RDXN(0:SF%N_CELLS_MAX))
ALLOCATE(SF%DX_WGT(0:SF%N_CELLS_MAX))
ALLOCATE(SF%X_S(0:SF%N_CELLS_MAX))
ALLOCATE(SF%LAYER_INDEX(0:SF%N_CELLS_MAX+1))
ALLOCATE(SF%MF_FRAC(1:SF%N_CELLS_MAX))
ALLOCATE(SF%RHO_0(0:SF%N_CELLS_MAX+1,SF%N_MATL))

! Compute node coordinates

CALL GET_WALL_NODE_COORDINATES(SF%N_CELLS_INI,SF%N_LAYERS,SF%N_LAYER_CELLS, &
SMALLEST_CELL_SIZE(1:SF%N_LAYERS),SF%STRETCH_FACTOR(1:SF%N_LAYERS),SF%X_S)

CALL GET_WALL_NODE_WEIGHTS(SF%N_CELLS_INI,SF%N_LAYERS,SF%N_LAYER_CELLS,SF%LAYER_THICKNESS,SF%GEOMETRY, &
SF%X_S,SF%LAYER_DIVIDE,SF%DX,SF%RDX,SF%RDXN,SF%DX_WGT,SF%DXF,SF%DXB,SF%LAYER_INDEX,SF%MF_FRAC,SF%INNER_RADIUS)

! Initialize the material densities of the solid

SF%RHO_0 = 0._EB

DO II=0,SF%N_CELLS_INI+1
IL = SF%LAYER_INDEX(II)
IF (SF%TMP_INNER(IL)<=0._EB) SF%TMP_INNER(IL) = TMPA
DO NN=1,SF%N_LAYER_MATL(IL)
DO N=1,SF%N_MATL
IF (SF%LAYER_MATL_INDEX(IL,NN)==SF%MATL_INDEX(N)) &
SF%RHO_0(II,N) = SF%LAYER_MATL_FRAC(IL,NN)*SF%LAYER_DENSITY(IL)
ENDDO
ENDDO
ENDDO

ENDDO SURF_GRID_LOOP

ALLOCATE(AAS(NWP_MAX),STAT=IZERO)
CALL ChkMemErr('INIT','AAS',IZERO)
ALLOCATE(CCS(NWP_MAX),STAT=IZERO)
CALL ChkMemErr('INIT','CCS',IZERO)
ALLOCATE(BBS(NWP_MAX),STAT=IZERO)
CALL ChkMemErr('INIT','BBS',IZERO)
```

```fortran
ALLOCATE(DDS(NWP_MAX) ,STAT=IZERO)
CALL ChkMemErr('INIT','DDS',IZERO)
ALLOCATE(DDT(NWP_MAX) ,STAT=IZERO)
CALL ChkMemErr('INIT','DDT',IZERO)
ALLOCATE(K_S(0:NWP_MAX+1) ,STAT=IZERO)
CALL ChkMemErr('INIT','K_S',IZERO)
ALLOCATE(C_S(0:NWP_MAX+1) ,STAT=IZERO)
CALL ChkMemErr('INIT','C_S',IZERO)
ALLOCATE(Q_S(1:NWP_MAX) ,STAT=IZERO)
CALL ChkMemErr('INIT','Q_S',IZERO)
ALLOCATE(RHO_S(0:NWP_MAX+1) ,STAT=IZERO)
CALL ChkMemErr('INIT','RHO_S',IZERO)
ALLOCATE(RHOCBAR(1:NWP_MAX) ,STAT=IZERO)
CALL ChkMemErr('INIT','RHOCBAR',IZERO)
ALLOCATE(KAPPA_S(1:NWP_MAX) ,STAT=IZERO)
CALL ChkMemErr('INIT','KAPPA_S',IZERO)
ALLOCATE(X_S_NEW(0:NWP_MAX) ,STAT=IZERO)
CALL ChkMemErr('INIT','X_S_NEW',IZERO)
ALLOCATE(DX_S(1:NWP_MAX) ,STAT=IZERO)
CALL ChkMemErr('INIT','DX_S',IZERO)
ALLOCATE(RDX_S(0:NWP_MAX+1) ,STAT=IZERO)
CALL ChkMemErr('INIT','RDX_S',IZERO)
ALLOCATE(RDXN_S(0:NWP_MAX) ,STAT=IZERO)
CALL ChkMemErr('INIT','RDXN_S',IZERO)
ALLOCATE(R_S(0:NWP_MAX) ,STAT=IZERO)
CALL ChkMemErr('INIT','R_S',IZERO)
ALLOCATE(R_S_NEW(0:NWP_MAX) ,STAT=IZERO)
CALL ChkMemErr('INIT','R_S_NEW',IZERO)
ALLOCATE(DX_WGT_S(0:NWP_MAX) ,STAT=IZERO)
CALL ChkMemErr('INIT','DX_WGT_S',IZERO)
ALLOCATE(LAYER_INDEX(0:NWP_MAX+1) ,STAT=IZERO)
CALL ChkMemErr('INIT','LAYER_INDEX',IZERO)
ALLOCATE(MF_FRAC(1:NWP_MAX) ,STAT=IZERO)
CALL ChkMemErr('INIT','MF_FRAC',IZERO)
ALLOCATE(REGRID_FACTOR(1:NWP_MAX) ,STAT=IZERO)
CALL ChkMemErr('INIT','REGRID_FACTOR',IZERO)

END SUBROUTINE PROC_WALL


SUBROUTINE READ_PRES

USE SCRC, ONLY: SCARC_METHOD , SCARC_KRYLOV , SCARC_MULTIGRID, SCARC_SMOOTH , SCARC_PRECON, &
SCARC_COARSE , SCARC_INITIAL, SCARC_ACCURACY, SCARC_DEBUG , &
SCARC_MULTIGRID_CYCLE, SCARC_MULTIGRID_LEVEL , SCARC_MULTIGRID_COARSENING , &
SCARC_MULTIGRID_ITERATIONS, SCARC_MULTIGRID_ACCURACY, SCARC_MULTIGRID_INTERPOL, &
SCARC_KRYLOV_ITERATIONS, SCARC_KRYLOV_ACCURACY, &
SCARC_SMOOTH_ITERATIONS, SCARC_SMOOTH_ACCURACY, SCARC_SMOOTH_OMEGA, &
SCARC_PRECON_ITERATIONS, SCARC_PRECON_ACCURACY, SCARC_PRECON_OMEGA, &
SCARC_COARSE_ITERATIONS, SCARC_COARSE_ACCURACY

CHARACTER(60) :: SOLVER='FFT'

NAMELIST /PRES/ CHECK_POISSON,FISHPAK_BC,ITERATION_SUSPEND_FACTOR,LAPLACE_PRESSURE_CORRECTION, &
MAX_PRESSURE_ITERATIONS,PRESSURE_RELAX_TIME,PRESSURE_TOLERANCE,RELAXATION_FACTOR, &
SCARC_METHOD , SCARC_KRYLOV , SCARC_MULTIGRID, SCARC_SMOOTH , SCARC_PRECON, &
SCARC_COARSE , SCARC_INITIAL, SCARC_ACCURACY, SCARC_DEBUG , &
SCARC_MULTIGRID_CYCLE, SCARC_MULTIGRID_LEVEL, SCARC_MULTIGRID_COARSENING , &
SCARC_MULTIGRID_ITERATIONS, SCARC_MULTIGRID_ACCURACY, SCARC_MULTIGRID_INTERPOL, &
SCARC_KRYLOV_ITERATIONS, SCARC_KRYLOV_ACCURACY, &
SCARC_SMOOTH_ITERATIONS, SCARC_SMOOTH_ACCURACY, SCARC_SMOOTH_OMEGA, &
SCARC_PRECON_ITERATIONS, SCARC_PRECON_ACCURACY, SCARC_PRECON_OMEGA, &
SCARC_COARSE_ITERATIONS, SCARC_COARSE_ACCURACY, &
SOLVER,SUSPEND_PRESSURE_ITERATIONS,VELOCITY_TOLERANCE

! Read the single PRES line

REWIND(LU_INPUT) ; INPUT_FILE_LINE_NUMBER = 0
READ_LOOP: DO
CALL CHECKREAD('PRES',LU_INPUT,IOS)
IF (IOS==1) EXIT READ_LOOP
READ(LU_INPUT,PRES,END=23,ERR=24,IOSTAT=IOS)
24 IF (IOS>0) THEN
CALL SHUTDOWN('ERROR: Problem with PRES line') ; RETURN
ENDIF
ENDDO READ_LOOP
23 REWIND(LU_INPUT) ; INPUT_FILE_LINE_NUMBER = 0

! Given the chosen SOLVER, define internal variable PRES_METHOD:

SELECT CASE(TRIM(SOLVER))
CASE('SCARC')
PRES_METHOD = 'SCARC'
ITERATE_PRESSURE = .FALSE.
IF (SCARC_METHOD == 'null') SCARC_METHOD = 'KRYLOV' ! Taken as default for SCARC when SOLVER is SCARC and
! SCARC_METHOD is not defined.
CASE('GLMAT')
PRES_METHOD = 'GLMAT'
```

```fortran
GLMAT_SOLVER = .TRUE.
PRES_ON_WHOLE_DOMAIN = .FALSE.

CASE('GLMAT IBM')
PRES_METHOD = 'GLMAT'
GLMAT_SOLVER = .TRUE.
PRES_ON_WHOLE_DOMAIN = .TRUE.

CASE DEFAULT
! Nothing to do. By default PRES_METHOD is set to 'FFT' in cons.f90
END SELECT

! Determine how many pressure iterations to perform per half time step.

IF (VELOCITY_TOLERANCE>100._EB) THEN
ITERATE_PRESSURE = .FALSE.
ELSE
ITERATE_PRESSURE = .TRUE.
IF (VELOCITY_TOLERANCE>TWO_EPSILON_EB .OR. PRESSURE_TOLERANCE>TWO_EPSILON_EB .OR. MAX_PRESSURE_ITERATIONS/=10) &
SUSPEND_PRESSURE_ITERATIONS=.FALSE.
IF (VELOCITY_TOLERANCE<TWO_EPSILON_EB) VELOCITY_TOLERANCE =  0.5_EB*CHARACTERISTIC_CELL_SIZE
IF (PRESSURE_TOLERANCE<TWO_EPSILON_EB) PRESSURE_TOLERANCE = 20.0_EB/CHARACTERISTIC_CELL_SIZE**2
ENDIF

IF (NMESHES>1 .AND. ANY(FISHPAK_BC==FISHPAK_BC_PERIODIC)) THEN
CALL SHUTDOWN('ERROR: Cannot use FISHPAK_BC_PERIODIC with NMESHES>1') ; RETURN
ENDIF

IF (ANY(FISHPAK_BC>0)) THEN
CALL SHUTDOWN('ERROR: Cannot have FISHPAK_BC>0') ; RETURN
ENDIF

END SUBROUTINE READ_PRES


SUBROUTINE READ_RADI

USE RADCONS
REAL(EB) :: BAND_LIMITS(MAX_NUMBER_SPECTRAL_BANDS+1), RADIATIVE_FRACTION=-1._EB
NAMELIST /RADI/ ANGLE_INCREMENT,BAND_LIMITS,C_MAX,C_MIN,INITIAL_RADIATION_ITERATIONS,KAPPA0,NMIEANG,
        NUMBER_RADIATION_ANGLES,&
PATH_LENGTH,&
QR_CLIP,RADIATION,RADIATION_ITERATIONS,RADIATIVE_FRACTION,RADTMP,RTE_SOURCE_CORRECTION,TIME_STEP_INCREMENT,&
WIDE_BAND_MODEL,MIE_MINIMUM_DIAMETER,MIE_MAXIMUM_DIAMETER,MIE_NDG, &
NUMBER_INITIAL_ITERATIONS !, RADIATIVE_FRACTION_1 ! Backward compatibility   Sesa-added RADIATIVE_FRACTION_1
REAL(EB) THETALOW,THETAUP
INTEGER  NRA,N

! Set default values

BAND_LIMITS(:) = -1._EB


INITIAL_RADIATION_ITERATIONS = 3
NUMBER_RADIATION_ANGLES = 100
TIME_STEP_INCREMENT     = 3
IF (TWO_D) THEN
NUMBER_RADIATION_ANGLES = 60
TIME_STEP_INCREMENT     = 2
ENDIF

KAPPA0               =   0._EB
RADTMP               = 900._EB
WIDE_BAND_MODEL      = .FALSE.
NMIEANG              = 15
PATH_LENGTH          = -1.0_EB ! calculate path based on the geometry
ANGLE_INCREMENT      = -1
MIE_MAXIMUM_DIAMETER = 0._EB
MIE_MINIMUM_DIAMETER = 0._EB
MIE_NDG              = 50
QR_CLIP              = 10._EB  ! kW/m3, lower bound for radiation source correction

! Read radiation parameters

REWIND(LU_INPUT) ; INPUT_FILE_LINE_NUMBER = 0
READ_LOOP: DO
CALL CHECKREAD('RADI',LU_INPUT,IOS)
IF (IOS==1) EXIT READ_LOOP
READ(LU_INPUT,RADI,END=23,ERR=24,IOSTAT=IOS)
24 IF (IOS>0) THEN
CALL SHUTDOWN('ERROR: Problem with RADI line') ; RETURN
ENDIF
ENDDO READ_LOOP
23 REWIND(LU_INPUT) ; INPUT_FILE_LINE_NUMBER = 0

IF (RADIATIVE_FRACTION >=0._EB) THEN
CALL SHUTDOWN('ERROR: RADIATIVE_FRACTION is now specified on REAC.')
RETURN
```

```
ENDIF

RADTMP = RADTMP + TMPM
QR_CLIP = QR_CLIP*1000._EB   ! kW/m3 to W/m3

! Define band parameters

IF (WIDE_BAND_MODEL) THEN
NUMBER_SPECTRAL_BANDS = 6
TIME_STEP_INCREMENT=MAX(1,TIME_STEP_INCREMENT)
ANGLE_INCREMENT = 1
UIIDIM=NUMBER_SPECTRAL_BANDS
ELSE
NUMBER_SPECTRAL_BANDS = 1
IF (ANGLE_INCREMENT < 0) ANGLE_INCREMENT = MAX(1,MIN(5,NUMBER_RADIATION_ANGLES/15))
UIIDIM = ANGLE_INCREMENT
ENDIF

! Define custom wavelength band limits

IF (ANY(BAND_LIMITS>0._EB)) THEN
NUMBER_SPECTRAL_BANDS = COUNT(BAND_LIMITS>0._EB) - 1
IF (NUMBER_SPECTRAL_BANDS<2) THEN ; CALL SHUTDOWN('ERROR: Need more spectral band limits.') ; RETURN ; ENDIF
IF (ANY((BAND_LIMITS(2:NUMBER_SPECTRAL_BANDS+1)-BAND_LIMITS(1:NUMBER_SPECTRAL_BANDS))<0._EB)) THEN
CALL SHUTDOWN('ERROR: Spectral band limits should be given in ascending order.')
RETURN
ENDIF
ALLOCATE(WL_HIGH(1:NUMBER_SPECTRAL_BANDS))
ALLOCATE(WL_LOW(1:NUMBER_SPECTRAL_BANDS))
DO I=1,NUMBER_SPECTRAL_BANDS
WL_LOW(I) = BAND_LIMITS(I)
WL_HIGH(I)= BAND_LIMITS(I+1)
ENDDO

TIME_STEP_INCREMENT=MAX(1,TIME_STEP_INCREMENT)
ANGLE_INCREMENT = 1
UIIDIM=NUMBER_SPECTRAL_BANDS
ENDIF

! Calculate actual number of radiation angles and determine the angular discretization

IF (.NOT.RADIATION) THEN

NUMBER_RADIATION_ANGLES = 1
INITIAL_RADIATION_ITERATIONS = 1

ELSE

NRA = NUMBER_RADIATION_ANGLES

! Determine the number of polar angles (theta)

IF (CYLINDRICAL) THEN
NRT = NINT(SQRT(REAL(NRA)))
ELSEIF (TWO_D) THEN
NRT = 1
ELSE
NRT = 2*NINT(0.5_EB*1.17*REAL(NRA)**(1._EB/2.26))
ENDIF

! Determine number of azimuthal angles (phi)

ALLOCATE(NRP(1:NRT),STAT=IZERO)
CALL ChkMemErr('INIT','NRP',IZERO)

N = 0
DO I=1,NRT
IF (CYLINDRICAL) THEN
NRP(I) = NINT(REAL(NRA)/(REAL(NRT)))
ELSEIF (TWO_D) THEN
NRP(I) = 4*NINT(0.25_EB*REAL(NRA))
ELSE
THETALOW = PI*REAL(I-1)/REAL(NRT)
THETAUP  = PI*REAL(I)/REAL(NRT)
NRP(I) = NINT(0.5_EB*REAL(NRA)*(COS(THETALOW)-COS(THETAUP)))
NRP(I) = MAX(4,NRP(I))
NRP(I) = 4*NINT(0.25_EB*REAL(NRP(I)))
ENDIF
N = N + NRP(I)
ENDDO
NUMBER_RADIATION_ANGLES = N

ENDIF

END SUBROUTINE READ_RADI


SUBROUTINE READ_CLIP
```

```fortran

REAL(EB) :: MAXIMUM_DENSITY,MINIMUM_DENSITY,MINIMUM_TEMPERATURE,MAXIMUM_TEMPERATURE
NAMELIST /CLIP/ FYI,MAXIMUM_DENSITY,MAXIMUM_TEMPERATURE,MINIMUM_DENSITY,MINIMUM_TEMPERATURE

! Check for user-defined mins and maxes.

MINIMUM_DENSITY      = -999._EB
MAXIMUM_DENSITY      = -999._EB
MINIMUM_TEMPERATURE  = -999._EB
MAXIMUM_TEMPERATURE  = -999._EB

REWIND(LU_INPUT) ; INPUT_FILE_LINE_NUMBER = 0
CLIP_LOOP: DO
CALL CHECKREAD('CLIP',LU_INPUT,IOS)
IF (IOS==1) EXIT CLIP_LOOP
READ(LU_INPUT,CLIP,END=431,ERR=432,IOSTAT=IOS)
432 IF (IOS>0) THEN ; CALL SHUTDOWN('ERROR: Problem with CLIP line') ; RETURN ; ENDIF
ENDDO CLIP_LOOP
431 REWIND(LU_INPUT) ; INPUT_FILE_LINE_NUMBER = 0

IF (MINIMUM_TEMPERATURE>-TMPM) TMPMIN = MINIMUM_TEMPERATURE + TMPM
IF (MAXIMUM_TEMPERATURE>-TMPM) TMPMAX = MAXIMUM_TEMPERATURE + TMPM

IF (TMPMAX > 5000._EB) THEN ; CALL SHUTDOWN('MAXIMUM_TEMPERATURE cannot be greater than 4726.85 C (5000 K)') ; RETURN ; ENDIF

IF (MINIMUM_DENSITY>0._EB) THEN
RHOMIN = MINIMUM_DENSITY
ELSE
RHOMIN = MIN(0.1_EB*RHOA,P_INF*MW_MIN/(R0*TMPMAX))
ENDIF

IF (MAXIMUM_DENSITY>0._EB) THEN
RHOMAX = MAXIMUM_DENSITY
ELSE
RHOMAX = 3.0_EB*P_INF*MW_MAX/(R0*MAX(TMPMIN,1._EB))
ENDIF

END SUBROUTINE READ_CLIP


SUBROUTINE READ_RAMP

REAL(EB) :: X,Z,T,F,TM
INTEGER  :: I,II,NN,N,NUMBER_INTERPOLATION_POINTS,N_RES_RAMP
CHARACTER(LABEL_LENGTH) :: DEVC_ID,CTRL_ID
TYPE(RAMPS_TYPE), POINTER :: RP
TYPE(RESERVED_RAMPS_TYPE), POINTER :: RRP
NAMELIST /RAMP/ CTRL_ID,DEVC_ID,F,FYI,ID,NUMBER_INTERPOLATION_POINTS,T,X,Z

IF (N_RAMP==0) RETURN
ALLOCATE(RAMPS(N_RAMP),STAT=IZERO)
CALL ChkMemErr('READ','RAMPS',IZERO)

! Count the number of points in each ramp

N_RES_RAMP = 0

COUNT_RAMP_POINTS: DO N=1,N_RAMP

RP => RAMPS(N)

IF (RAMP_ID(N)(1:5)=='RSRVD') THEN
N_RES_RAMP = N_RES_RAMP + 1
RRP => RESERVED_RAMPS(N_RES_RAMP)
RP%RESERVED = .TRUE.
RP%NUMBER_DATA_POINTS = RRP%NUMBER_DATA_POINTS
ELSE
REWIND(LU_INPUT) ; INPUT_FILE_LINE_NUMBER = 0
RP%NUMBER_DATA_POINTS = 0
SEARCH_LOOP: DO
CALL CHECKREAD('RAMP',LU_INPUT,IOS)
IF (IOS==1) EXIT SEARCH_LOOP
READ(LU_INPUT,NML=RAMP,ERR=56,IOSTAT=IOS)
IF (ID/=RAMP_ID(N)) CYCLE SEARCH_LOOP
RP%NUMBER_DATA_POINTS = RP%NUMBER_DATA_POINTS + 1
56 IF (IOS>0) THEN ; CALL SHUTDOWN('ERROR: Problem with RAMP '//TRIM(RAMP_ID(N)) ) ; RETURN ; ENDIF
ENDDO SEARCH_LOOP
ENDIF

IF (RP%NUMBER_DATA_POINTS<2) THEN
IF (RP%NUMBER_DATA_POINTS==0) WRITE(MESSAGE,'(A,A,A)') 'ERROR: RAMP ',TRIM(RAMP_ID(N)), ' not found'
IF (RP%NUMBER_DATA_POINTS==1) WRITE(MESSAGE,'(A,A,A)') 'ERROR: RAMP ',TRIM(RAMP_ID(N)), ' has only one point'
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF

ENDDO COUNT_RAMP_POINTS

```

```
! Read and process the ramp functions

N_RES_RAMP = 0

READ_RAMP_LOOP: DO N=1,N_RAMP

RP => RAMPS(N)

RP%DEVC_ID = 'null'
RP%CTRL_ID = 'null'
ALLOCATE(RP%INDEPENDENT_DATA(1:RP%NUMBER_DATA_POINTS))
ALLOCATE(RP%DEPENDENT_DATA(1:RP%NUMBER_DATA_POINTS))
NUMBER_INTERPOLATION_POINTS=5000

IF (RP%RESERVED) THEN

N_RES_RAMP = N_RES_RAMP + 1
RRP => RESERVED_RAMPS(N_RES_RAMP)
RP%INDEPENDENT_DATA(1:RP%NUMBER_DATA_POINTS) = RRP%INDEPENDENT_DATA(1:RRP%NUMBER_DATA_POINTS)
RP%DEPENDENT_DATA(1:RP%NUMBER_DATA_POINTS) = RRP%DEPENDENT_DATA(1:RRP%NUMBER_DATA_POINTS)
RP%NUMBER_INTERPOLATION_POINTS = NUMBER_INTERPOLATION_POINTS

ELSE

REWIND(LU_INPUT) ; INPUT_FILE_LINE_NUMBER = 0
NN = 0
SEARCH_LOOP2: DO
DEVC_ID = 'null'
CTRL_ID = 'null'
X       = -1.E6_EB
Z       = -1.E6_EB
CALL CHECKREAD('RAMP',LU_INPUT,IOS)
IF (IOS==1) EXIT SEARCH_LOOP2
READ(LU_INPUT,RAMP)
IF (ID/=RAMP_ID(N)) CYCLE SEARCH_LOOP2
IF (RP%DEVC_ID =='null') RP%DEVC_ID = DEVC_ID
IF (RP%CTRL_ID =='null') RP%CTRL_ID = CTRL_ID
IF (X>-1.E5_EB) THEN
RAMP_TYPE(N) = 'X COORDINATE'
SPATIAL_GRAVITY_VARIATION = .TRUE.
STRATIFICATION = .FALSE.
T = X
ENDIF
IF (Z>-1.E5_EB) THEN
RAMP_TYPE(N) = 'Z COORDINATE'
T = Z
ENDIF
IF (RAMP_TYPE(N)=='TEMPERATURE') T = T + TMPM
IF (RAMP_TYPE(N)=='TIME')        T = T_BEGIN + (T-T_BEGIN)/TIME_SHRINK_FACTOR
NN = NN+1
RP%INDEPENDENT_DATA(NN) = T
IF (NN>1) THEN
IF (T<=RP%INDEPENDENT_DATA(NN-1)) THEN
WRITE(MESSAGE,'(A,A,A)') 'ERROR: RAMP ',TRIM(RAMP_ID(N)), ' variable T must be monotonically increasing'
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF
ENDIF
RP%DEPENDENT_DATA(NN) = F
RP%NUMBER_INTERPOLATION_POINTS = NUMBER_INTERPOLATION_POINTS
ENDDO SEARCH_LOOP2

ENDIF

RP%T_MIN = MINVAL(RP%INDEPENDENT_DATA)
RP%T_MAX = MAXVAL(RP%INDEPENDENT_DATA)
RP%SPAN = RP%T_MAX - RP%T_MIN

ENDDO READ_RAMP_LOOP

! Set up interpolated ramp values in INTERPOLATED_DATA and get control or device index

DO N=1,N_RAMP
RP => RAMPS(N)
RP%RDT = REAL(RP%NUMBER_INTERPOLATION_POINTS,EB)/RP%SPAN
ALLOCATE(RAMPS(N)%INTERPOLATED_DATA(0:RP%NUMBER_INTERPOLATION_POINTS+1))
RAMPS(N)%INTERPOLATED_DATA(0) = RP%DEPENDENT_DATA(1)
DO I=1,RP%NUMBER_INTERPOLATION_POINTS-1
TM = RP%INDEPENDENT_DATA(1) + REAL(I,EB)/RP%RDT
TLOOP: DO II=1,RP%NUMBER_DATA_POINTS-1
IF (TM>=RP%INDEPENDENT_DATA(II) .AND. TM<RP%INDEPENDENT_DATA(II+1)) THEN
RP%INTERPOLATED_DATA(I) = RP%DEPENDENT_DATA(II) + (TM-RP%INDEPENDENT_DATA(II)) * &
(RP%DEPENDENT_DATA(II+1)-RP%DEPENDENT_DATA(II))/(RP%INDEPENDENT_DATA(II+1)-RP%INDEPENDENT_DATA(II))
EXIT TLOOP
ENDIF
ENDDO TLOOP
ENDDO
RP%INTERPOLATED_DATA(RP%NUMBER_INTERPOLATION_POINTS)   = RP%DEPENDENT_DATA(RP%NUMBER_DATA_POINTS)
RP%INTERPOLATED_DATA(RP%NUMBER_INTERPOLATION_POINTS+1) = RP%DEPENDENT_DATA(RP%NUMBER_DATA_POINTS)
```

```fortran

! Get Device or Control Index

CALL SEARCH_CONTROLLER('RAMP',RP%CTRL_ID,RP%DEVC_ID,RP%DEVC_INDEX,RP%CTRL_INDEX,N)

ENDDO

END SUBROUTINE READ_RAMP


SUBROUTINE READ_TABL
REAL(EB) :: TABLE_DATA(9)
INTEGER  :: NN,N,I,J
TYPE(TABLES_TYPE), POINTER :: TA=>NULL()
NAMELIST /TABL/ FYI,ID,TABLE_DATA

IF (N_TABLE==0) RETURN

ALLOCATE(TABLES(N_TABLE),STAT=IZERO)
CALL ChkMemErr('READ','TABLES',IZERO)

! Count the number of points in each table

REWIND(LU_INPUT) ; INPUT_FILE_LINE_NUMBER = 0
COUNT_TABLE_POINTS: DO N=1,N_TABLE
TA => TABLES(N)
REWIND(LU_INPUT) ; INPUT_FILE_LINE_NUMBER = 0
TA%NUMBER_ROWS = 0
SELECT CASE (TABLE_TYPE(N))
CASE (SPRAY_PATTERN)
TA%NUMBER_COLUMNS = 6
CASE (PART_RADIATIVE_PROPERTY)
TA%NUMBER_COLUMNS = 3
CASE (TABLE_2D_TYPE,FLAME_SPEED_TABLE)
TA%NUMBER_COLUMNS = 3
END SELECT
SEARCH_LOOP: DO
CALL CHECKREAD('TABL',LU_INPUT,IOS)
IF (IOS==1) EXIT SEARCH_LOOP
TABLE_DATA = -999._EB
READ(LU_INPUT,NML=TABL,ERR=56,IOSTAT=IOS)
IF (ID/=TABLE_ID(N)) CYCLE SEARCH_LOOP
TA%NUMBER_ROWS = TA%NUMBER_ROWS + 1
MESSAGE='null'
SELECT CASE(TABLE_TYPE(N))
CASE (SPRAY_PATTERN)
IF (TABLE_DATA(1)<0._EB .OR.           TABLE_DATA(1)>180._EB) THEN
WRITE(MESSAGE,'(A,I0,A,A,A)') 'ERROR: Row ',TA%NUMBER_ROWS,' of ',TRIM(TABLE_ID(N)),' has a bad 1st lattitude'
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF
IF (TABLE_DATA(2)<TABLE_DATA(1).OR. TABLE_DATA(2)>180._EB) THEN
WRITE(MESSAGE,'(A,I0,A,A,A)') 'ERROR: Row ',TA%NUMBER_ROWS,' of ',TRIM(TABLE_ID(N)),' has a bad 2nd lattitude'
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF
IF (TABLE_DATA(3)<-180._EB .OR.           TABLE_DATA(3)>360._EB) THEN
WRITE(MESSAGE,'(A,I0,A,A,A)') 'ERROR: Row ',TA%NUMBER_ROWS,' of ',TRIM(TABLE_ID(N)),' has a bad 1st longitude'
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF
IF (TABLE_DATA(4)<TABLE_DATA(3).OR. TABLE_DATA(4)>360._EB) THEN
WRITE(MESSAGE,'(A,I0,A,A,A)') 'ERROR: Row ',TA%NUMBER_ROWS,' of ',TRIM(TABLE_ID(N)),' has a bad 2nd longitude'
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF
IF (TABLE_DATA(5)<0._EB) THEN
WRITE(MESSAGE,'(A,I0,A,A,A)') 'ERROR: Row ',TA%NUMBER_ROWS,' of ',TRIM(TABLE_ID(N)),' has a bad velocity'
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF
IF (TABLE_DATA(6)<0._EB) THEN
WRITE(MESSAGE,'(A,I0,A,A,A)') 'ERROR: Row ',TA%NUMBER_ROWS,' of ',TRIM(TABLE_ID(N)),' has a bad mass flow'
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF
CASE (PART_RADIATIVE_PROPERTY)
IF (TABLE_DATA(1)<0._EB) THEN
WRITE(MESSAGE,'(A,I0,A,A,A)') 'ERROR: Row ',TA%NUMBER_ROWS,' of ',TRIM(TABLE_ID(N)),' has a bad wave length'
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF
IF (TABLE_DATA(2)<=0._EB) THEN
WRITE(MESSAGE,'(A,I0,A,A,A)') 'ERROR: Row ',TA%NUMBER_ROWS,' of ',TRIM(TABLE_ID(N)),' has a bad real index'
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF
IF (TABLE_DATA(3)< 0._EB) THEN
WRITE(MESSAGE,'(A,I0,A,A,A)') 'ERROR: Row ',TA%NUMBER_ROWS,' of ',TRIM(TABLE_ID(N)),' has a bad complex index'
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF
CASE (TABLE_2D_TYPE,FLAME_SPEED_TABLE)
IF (TA%NUMBER_ROWS == 1) THEN
IF (INT(TABLE_DATA(1)) <= 0._EB) THEN
WRITE(MESSAGE,'(A,I0,A,A,A)') 'ERROR: Row ',TA%NUMBER_ROWS,' of ',TRIM(TABLE_ID(N)),&
' has < 1 x entries'
```

```
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF
IF (INT(TABLE_DATA(2)) < 0._EB) THEN
WRITE(MESSAGE,'(A,I0,A,A,A)') 'ERROR: Row ',TA%NUMBER_ROWS,' of ',TRIM(TABLE_ID(N)),&
' has < 1 y entries'
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF
ENDIF
END SELECT

56 IF (IOS>0) THEN ; CALL SHUTDOWN('ERROR: Problem with TABLE '//TRIM(TABLE_ID(N)) ) ; RETURN ; ENDIF
ENDDO SEARCH_LOOP
IF (TA%NUMBER_ROWS<=0) THEN
WRITE(MESSAGE,'(A,A,A)') 'ERROR: TABLE ',TRIM(TABLE_ID(N)), ' not found'
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF
IF (TABLE_TYPE(N) == TABLE_2D_TYPE .OR. TABLE_TYPE(N) == FLAME_SPEED_TABLE) THEN
IF (TA%NUMBER_ROWS<=1) THEN
WRITE(MESSAGE,'(A,A,A)') 'ERROR: 2D TABLE ',TRIM(TABLE_ID(N)), ' must have at least one row of data'
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF
ENDIF
ENDDO COUNT_TABLE_POINTS

READ_TABL_LOOP: DO N=1,N_TABLE
TA => TABLES(N)
ALLOCATE(TA%TABLE_DATA(TA%NUMBER_ROWS,TA%NUMBER_COLUMNS),STAT=IZERO)
CALL ChkMemErr('READ','TA%TABLE_DATA',IZERO)
REWIND(LU_INPUT) ; INPUT_FILE_LINE_NUMBER = 0
NN = 0
SEARCH_LOOP2: DO
CALL CHECKREAD('TABL',LU_INPUT,IOS)
IF (IOS==1) EXIT SEARCH_LOOP2
READ(LU_INPUT,TABL)
IF (ID/=TABLE_ID(N)) CYCLE SEARCH_LOOP2
NN = NN+1
TA%TABLE_DATA(NN,:) = TABLE_DATA(1:TA%NUMBER_COLUMNS)
ENDDO SEARCH_LOOP2
TABLE_2D_IF: IF (TABLE_TYPE(N)==TABLE_2D_TYPE) THEN
IF (TA%NUMBER_ROWS-1/=INT(TA%TABLE_DATA(1,1))*INT(TA%TABLE_DATA(1,2))) THEN
WRITE(MESSAGE,'(A,A,A)') 'ERROR: 2D TABLE ',TRIM(TABLE_ID(N)), ' is not rectangular'
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF
TA%LX = MINVAL(TA%TABLE_DATA(2:TA%NUMBER_ROWS,1),1)
TA%UX = MAXVAL(TA%TABLE_DATA(2:TA%NUMBER_ROWS,1),1)
TA%LY = MINVAL(TA%TABLE_DATA(2:TA%NUMBER_ROWS,2),1)
TA%UY = MAXVAL(TA%TABLE_DATA(2:TA%NUMBER_ROWS,2),1)
ALLOCATE(TA%X(INT(TA%TABLE_DATA(1,1))),STAT=IZERO)
CALL ChkMemErr('READ','TA%X',IZERO)
ALLOCATE(TA%Y(INT(TA%TABLE_DATA(1,2))),STAT=IZERO)
CALL ChkMemErr('READ','TA%Y',IZERO)
ALLOCATE(TA%Z(INT(TA%TABLE_DATA(1,1)),INT(TA%TABLE_DATA(1,2))),STAT=IZERO)
CALL ChkMemErr('READ','TA%Z',IZERO)
NN = 1
TA%NUMBER_ROWS = INT(TA%TABLE_DATA(1,1))
TA%NUMBER_COLUMNS = INT(TA%TABLE_DATA(1,2))
DO I = 1, TA%NUMBER_ROWS
DO J = 1, TA%NUMBER_COLUMNS
NN = NN + 1
IF (J==1) THEN
TA%X(I)=TA%TABLE_DATA(NN,1)
ELSE
IF (TA%TABLE_DATA(NN,1) /= TA%X(I)) THEN
WRITE(MESSAGE,'(A,A,A)') 'ERROR: 2D TABLE ',TRIM(TABLE_ID(N)), ' x value must be the same for each row'
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF
ENDIF
IF (I==1) THEN
TA%Y(J)=TA%TABLE_DATA(NN,2)
ELSE
IF (TA%TABLE_DATA(NN,2) /= TA%Y(J)) THEN
WRITE(MESSAGE,'(A,A,A)') 'ERROR: 2D TABLE ',TRIM(TABLE_ID(N)), ' y value must be the same for each column'
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF
ENDIF
TA%Z(I,J) = TA%TABLE_DATA(NN,3)
ENDDO
ENDDO
IF (TABLE_TYPE(N)==FLAME_SPEED_TABLE) TA%Y=TA%Y+TMPM
ENDIF TABLE_2D_IF
ENDDO READ_TABL_LOOP

END SUBROUTINE READ_TABL


SUBROUTINE READ_OBST

USE GEOMETRY_FUNCTIONS, ONLY: BLOCK_CELL
```

```
TYPE(OBSTRUCTION_TYPE), POINTER :: OB2=>NULL(),OBT=>NULL()
TYPE(MULTIPLIER_TYPE), POINTER :: MR=>NULL()
TYPE(OBSTRUCTION_TYPE), DIMENSION(:), ALLOCATABLE, TARGET :: TEMP_OBSTRUCTION
INTEGER :: NM,NOM,N_OBST_O,NNN,IC,N,NN,NNNN,N_NEW_OBST,RGB(3),N_OBST_NEW,II,JJ,KK,EVAC_N
CHARACTER(LABEL_LENGTH) :: ID,DEVC_ID,PROP_ID,SURF_ID,SURF_IDS(3),SURF_ID6(6),CTRL_ID,MULT_ID,MATL_ID
CHARACTER(60) :: MESH_ID
CHARACTER(25) :: COLOR
LOGICAL :: EVACUATION_OBST,OVERLAY
REAL(EB) :: TRANSPARENCY,XB1,XB2,XB3,XB4,XB5,XB6,BULK_DENSITY,VOL_ADJUSTED,VOL_SPECIFIED,UNDIVIDED_INPUT_AREA(3)
  ,&
INTERNAL_HEAT_SOURCE
LOGICAL :: EMBEDDED,THICKEN,PERMIT_HOLE,ALLOW_VENT,EVACUATION,REMOVABLE,BNDF_FACE(-3:3),BNDF_OBST,OUTLINE,
    NOTERRAIN,HT3D
NAMELIST /OBST/ ALLOW_VENT,BNDF_FACE,BNDF_OBST,BULK_DENSITY,&
COLOR,CTRL_ID,DEVC_ID,EVACUATION,FYI,HT3D,ID,INTERNAL_HEAT_SOURCE,MATL_ID,MESH_ID,MULT_ID,NOTERRAIN,&
OUTLINE,OVERLAY,PERMIT_HOLE,PROP_ID,REMOVABLE,RGB,SURF_ID,SURF_ID6,SURF_IDS,TEXTURE_ORIGIN,THICKEN,&
TRANSPARENCY,XB

MESH_LOOP: DO NM=1,NMESHES

IF (PROCESS(NM)/=MYID .AND. MYID/=EVAC_PROCESS) CYCLE MESH_LOOP

M=>MESHES(NM)
CALL POINT_TO_MESH(NM)
! Count OBST lines

REWIND(LU_INPUT) ; INPUT_FILE_LINE_NUMBER = 0
N_OBST = 0
COUNT_OBST_LOOP: DO
CALL CHECKREAD('OBST',LU_INPUT,IOS)
IF (IOS==1) EXIT COUNT_OBST_LOOP
MULT_ID = 'null'
READ(LU_INPUT,NML=OBST,END=1,ERR=2,IOSTAT=IOS)
N_OBST_NEW = 0
IF (MULT_ID=='null') THEN
N_OBST_NEW = 1
ELSE
DO N=1,N_MULT
MR => MULTIPLIER(N)
IF (MULT_ID==MR%ID) N_OBST_NEW = MR%N_COPIES
ENDDO
IF (N_OBST_NEW==0) THEN
WRITE(MESSAGE,'(A,A,A,I0,A,I0)') 'ERROR: MULT line ', TRIM(MULT_ID),' not found on OBST ', N_OBST+1,&
', line number',INPUT_FILE_LINE_NUMBER
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF
ENDIF
N_OBST = N_OBST + N_OBST_NEW
2 IF (IOS>0) THEN
WRITE(MESSAGE,'(A,I0,A,I0)') 'ERROR: Problem with OBST number',N_OBST+1,', line number',INPUT_FILE_LINE_NUMBER
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF
ENDDO COUNT_OBST_LOOP
1 REWIND(LU_INPUT) ; INPUT_FILE_LINE_NUMBER = 0

IF (EVACUATION_ONLY(NM)) CALL DEFINE_EVACUATION_OBSTS(NM,1,0)

! Allocate OBSTRUCTION array

ALLOCATE(M%OBSTRUCTION(0:N_OBST),STAT=IZERO)
CALL ChkMemErr('READ','OBSTRUCTION',IZERO)
OBSTRUCTION=>M%OBSTRUCTION

N        = 0
N_OBST_O = N_OBST
EVAC_N   = 1

READ_OBST_LOOP: DO NN=1,N_OBST_O

ID       = 'null'
MATL_ID  = 'null' ! for HT3D only
MULT_ID  = 'null'
PROP_ID  = 'null'
SURF_ID  = 'null'
SURF_IDS = 'null'
SURF_ID6 = 'null'
COLOR    = 'null'
MESH_ID  = 'null'
RGB          = -1
BULK_DENSITY= -1._EB
HT3D         = .FALSE.
INTERNAL_HEAT_SOURCE = 0._EB  ! for HT3D only
TRANSPARENCY= 1._EB
BNDF_FACE    = BNDF_DEFAULT
BNDF_OBST    = BNDF_DEFAULT
NOTERRAIN    = .FALSE.
THICKEN      = THICKEN_OBSTRUCTIONS
OUTLINE      = .FALSE.
```

```
OVERLAY        = .TRUE.
TEXTURE_ORIGIN = -999._EB
DEVC_ID        = 'null'
CTRL_ID        = 'null'
PERMIT_HOLE    = .TRUE.
ALLOW_VENT     = .TRUE.
REMOVABLE      = .TRUE.
XB             = -9.E30_EB
IF (.NOT.EVACUATION_ONLY(NM)) EVACUATION = .FALSE.
IF (     EVACUATION_ONLY(NM)) EVACUATION = .TRUE.
IF (     EVACUATION_ONLY(NM)) REMOVABLE  = .FALSE.

! Read the OBST line

EVACUATION_OBST = .FALSE.
IF (EVACUATION_ONLY(NM)) CALL DEFINE_EVACUATION_OBSTS(NM,2,EVAC_N)
EVACUATION_OBSTS: IF (.NOT. EVACUATION_OBST) THEN
CALL CHECKREAD('OBST',LU_INPUT,IOS)
IF (IOS==1) EXIT READ_OBST_LOOP
READ(LU_INPUT,OBST,END=35)
END IF EVACUATION_OBSTS

! Reorder OBST coordinates if necessary

CALL CHECK_XB(XB)

! If any obstruction is to do 3D heat transfer (HT3D), set a global parameter

IF (HT3D) SOLID_HT3D = .TRUE.

! No device and controls for evacuation obstructions

IF (EVACUATION_ONLY(NM)) THEN
DEVC_ID    = 'null'
CTRL_ID    = 'null'
PROP_ID    = 'null'
END IF

! Loop over all possible multiples of the OBST

MR => MULTIPLIER(0)
DO NNN=1,N_MULT
IF (MULT_ID==MULTIPLIER(NNN)%ID) MR => MULTIPLIER(NNN)
ENDDO

K_MULT_LOOP: DO KK=MR%K_LOWER,MR%K_UPPER
J_MULT_LOOP: DO JJ=MR%J_LOWER,MR%J_UPPER
I_MULT_LOOP: DO II=MR%I_LOWER,MR%I_UPPER

IF (.NOT.MR%SEQUENTIAL) THEN
XB1 = XB(1) + MR%DX0 + II*MR%DXB(1)
XB2 = XB(2) + MR%DX0 + II*MR%DXB(2)
XB3 = XB(3) + MR%DY0 + JJ*MR%DXB(3)
XB4 = XB(4) + MR%DY0 + JJ*MR%DXB(4)
XB5 = XB(5) + MR%DZ0 + KK*MR%DXB(5)
XB6 = XB(6) + MR%DZ0 + KK*MR%DXB(6)
ELSE
XB1 = XB(1) + MR%DX0 + II*MR%DXB(1)
XB2 = XB(2) + MR%DX0 + II*MR%DXB(2)
XB3 = XB(3) + MR%DY0 + II*MR%DXB(3)
XB4 = XB(4) + MR%DY0 + II*MR%DXB(4)
XB5 = XB(5) + MR%DZ0 + II*MR%DXB(5)
XB6 = XB(6) + MR%DZ0 + II*MR%DXB(6)
ENDIF

! Increase the OBST counter

N = N + 1

! Evacuation criteria

EVAC_N = EVAC_N + 1
IF (MESH_ID/=MESH_NAME(NM) .AND. MESH_ID/='null') THEN
N = N-1
N_OBST = N_OBST-1
CYCLE I_MULT_LOOP
ENDIF

IF ((.NOT.EVACUATION .AND. EVACUATION_ONLY(NM)) .OR. (EVACUATION .AND. .NOT.EVACUATION_ONLY(NM))) THEN
N = N-1
N_OBST = N_OBST-1
CYCLE I_MULT_LOOP
ENDIF

! Look for obstructions that are within a half grid cell of the current mesh. If the obstruction is thin and has the
! THICKEN attribute, look for it within an entire grid cell.

```

```
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
XB1 = XS
XB2 = XS
THICKEN = .FALSE.
ENDIF
IF ( (XB1<XF+0.5_EB*DX(IBP1) .AND. XB1>XF) .OR. (THICKEN .AND. 0.5_EB*(XB1+XB2)< XF+DX(IBP1) .AND. XB1>XF) ) THEN
XB1 = XF
XB2 = XF
THICKEN = .FALSE.
ENDIF
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
XB3 = YS
XB4 = YS
THICKEN = .FALSE.
ENDIF
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
XB3 = YF
XB4 = YF
THICKEN = .FALSE.
ENDIF
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
     . &
 .NOT.EVACUATION_ONLY(NM)) THEN
XB5 = ZS
XB6 = ZS
THICKEN = .FALSE.
ENDIF
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
     . &
 .NOT.EVACUATION_ONLY(NM)) THEN
XB5 = ZF
XB6 = ZF
THICKEN = .FALSE.
ENDIF

! Save the original, undivided obstruction face areas.

UNDIVIDED_INPUT_AREA(1) = (XB4-XB3)*(XB6-XB5)
UNDIVIDED_INPUT_AREA(2) = (XB2-XB1)*(XB6-XB5)
UNDIVIDED_INPUT_AREA(3) = (XB2-XB1)*(XB4-XB3)

! Throw out obstructions that are not within computational domain

XB1 = MAX(XB1,XS)
XB2 = MIN(XB2,XF)
XB3 = MAX(XB3,YS)
XB4 = MIN(XB4,YF)
XB5 = MAX(XB5,ZS)
XB6 = MIN(XB6,ZF)
IF (XB1>XF .OR. XB2<XS .OR. XB3>YF .OR. XB4<YS .OR. XB5>ZF .OR. XB6<ZS) THEN
N = N-1
N_OBST = N_OBST-1
CYCLE I_MULT_LOOP
ENDIF

! Begin processing of OBSTruction

OB=>OBSTRUCTION(N)

OB%UNDIVIDED_INPUT_AREA(1) = UNDIVIDED_INPUT_AREA(1)
OB%UNDIVIDED_INPUT_AREA(2) = UNDIVIDED_INPUT_AREA(2)
OB%UNDIVIDED_INPUT_AREA(3) = UNDIVIDED_INPUT_AREA(3)

OB%X1 = XB1
OB%X2 = XB2
OB%Y1 = XB3
OB%Y2 = XB4
OB%Z1 = XB5
OB%Z2 = XB6

OB%NOTERRAIN = NOTERRAIN

! Thicken evacuation mesh obstructions in the z direction

IF (EVACUATION_ONLY(NM) .AND. EVACUATION) THEN
OB%Z1 = ZS
OB%Z2 = ZF
XB5   = ZS
XB6   = ZF
ENDIF

! Determine the indices of the obstruction according to cell edges, not centers.

OB%I1 = NINT( GINV(XB1-XS,1,NM)*RDXI   )
OB%I2 = NINT( GINV(XB2-XS,1,NM)*RDXI   )
OB%J1 = NINT( GINV(XB3-YS,2,NM)*RDETA  )
OB%J2 = NINT( GINV(XB4-YS,2,NM)*RDETA  )
OB%K1 = NINT( GINV(XB5-ZS,3,NM)*RDZETA )
```

```
OB%K2 = NINT( GINV(XB6-ZS,3,NM)*RDZETA )

! If desired, thicken small obstructions

IF (THICKEN) THEN
IF (OB%I1==OB%I2) THEN
OB%I1 = INT(GINV(.5_EB*(XB1+XB2)-XS,1,NM)*RDXI)
OB%I2 = MIN(OB%I1+1,IBAR)
ENDIF
IF (OB%J1==OB%J2) THEN
OB%J1 = INT(GINV(.5_EB*(XB3+XB4)-YS,2,NM)*RDETA)
OB%J2 = MIN(OB%J1+1,JBAR)
ENDIF
IF (OB%K1==OB%K2) THEN
OB%K1 = INT(GINV(.5_EB*(XB5+XB6)-ZS,3,NM)*RDZETA)
OB%K2 = MIN(OB%K1+1,KBAR)
ENDIF
ELSE
!Don't allow thickening if an OBST straddles the midpoint and is small compared to grid cell
IF (GINV(XB2-XS,1,NM)-GINV(XB1-XS,1,NM)<0.25_EB/RDXI .AND. OB%I1 /= OB%I2) THEN
IF(GINV(XB1-XS,1,NM)-REAL(OB%I1,EB) < REAL(OB%I2,EB) - GINV(XB2-XS,1,NM)) THEN
OB%I2=OB%I1
ELSE
OB%I1=OB%I2
ENDIF
ENDIF
IF (GINV(XB4-YS,2,NM)-GINV(XB3-YS,2,NM)<0.25_EB/RDETA .AND. OB%J1 /= OB%J2) THEN
IF(GINV(XB3-XS,2,NM)-REAL(OB%J1,EB) < REAL(OB%J2,EB) - GINV(XB4-YS,2,NM)) THEN
OB%J2=OB%J1
ELSE
OB%J1=OB%J2
ENDIF
ENDIF
IF (GINV(XB6-ZS,3,NM)-GINV(XB5-ZS,3,NM)<0.25_EB/RDZETA .AND. OB%K1 /= OB%K2) THEN
IF(GINV(XB5-ZS,3,NM)-REAL(OB%I1,EB) < REAL(OB%I2,EB) - GINV(XB6-ZS,3,NM)) THEN
OB%K2=OB%K1
ELSE
OB%K1=OB%K2
ENDIF
ENDIF
ENDIF

! Throw out obstructions that are too small

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
N = N-1
N_OBST= N_OBST-1
CYCLE I_MULT_LOOP
ENDIF

! Check to see if obstruction is completely embedded in another

EMBEDDED = .FALSE.
EMBED_LOOP: DO NNN=1,N-1
OB2=>OBSTRUCTION(NNN)
IF (OB%I1>OB2%I1 .AND. OB%I2<OB2%I2 .AND. &
OB%J1>OB2%J1 .AND. OB%J2<OB2%J2 .AND. &
OB%K1>OB2%K1 .AND. OB%K2<OB2%K2) THEN
EMBEDDED = .TRUE.
EXIT EMBED_LOOP
ENDIF
ENDDO EMBED_LOOP

IF (EMBEDDED .AND. DEVC_ID=='null' .AND. REMOVABLE .AND. CTRL_ID=='null' ) THEN
N = N-1
N_OBST= N_OBST-1
CYCLE I_MULT_LOOP
ENDIF

! Check if the SURF IDs exist

IF (EVACUATION_ONLY(NM)) SURF_ID=EVAC_SURF_DEFAULT

IF (SURF_ID/='null') CALL CHECK_SURF_NAME(SURF_ID,EX)
IF (.NOT.EX) THEN
WRITE(MESSAGE,'(A,A,A)') 'ERROR: SURF_ID ',TRIM(SURF_ID),' does not exist'
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF

DO NNNN=1,3
IF (EVACUATION_ONLY(NM)) SURF_IDS(NNNN)=EVAC_SURF_DEFAULT
IF (SURF_IDS(NNNN)/='null') CALL CHECK_SURF_NAME(SURF_IDS(NNNN),EX)
IF (.NOT.EX) THEN
WRITE(MESSAGE,'(A,A,A)') 'ERROR: SURF_ID ',TRIM(SURF_IDS(NNNN)),' does not exist'
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF
ENDDO
```

```
DO NNNN=1,6
IF (EVACUATION_ONLY(NM)) SURF_ID6(NNNN)=EVAC_SURF_DEFAULT
IF (SURF_ID6(NNNN)/='null') CALL CHECK_SURF_NAME(SURF_ID6(NNNN),EX)
IF (.NOT.EX) THEN
WRITE(MESSAGE,'(A,A,A)') 'ERROR: SURF_ID ',TRIM(SURF_ID6(NNNN)),' does not exist'
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF
ENDDO

! Save boundary condition info for obstacles

OB%SURF_INDEX(:) = DEFAULT_SURF_INDEX

NNNN = 0
DO NNN=0,N_SURF
IF (SURF_ID      ==SURFACE(NNN)%ID) OB%SURF_INDEX(:)     = NNN
IF (SURF_IDS(1)==SURFACE(NNN)%ID) OB%SURF_INDEX(3)     = NNN
IF (SURF_IDS(2)==SURFACE(NNN)%ID) OB%SURF_INDEX(-2:2) = NNN
IF (SURF_IDS(3)==SURFACE(NNN)%ID) OB%SURF_INDEX(-3)    = NNN
IF (SURF_ID6(1)==SURFACE(NNN)%ID) OB%SURF_INDEX(-1)    = NNN
IF (SURF_ID6(2)==SURFACE(NNN)%ID) OB%SURF_INDEX( 1)    = NNN
IF (SURF_ID6(3)==SURFACE(NNN)%ID) OB%SURF_INDEX(-2)    = NNN
IF (SURF_ID6(4)==SURFACE(NNN)%ID) OB%SURF_INDEX( 2)    = NNN
IF (SURF_ID6(5)==SURFACE(NNN)%ID) OB%SURF_INDEX(-3)    = NNN
IF (SURF_ID6(6)==SURFACE(NNN)%ID) OB%SURF_INDEX( 3)    = NNN
IF (TRIM(SURFACE(NNN)%ID)==TRIM(EVAC_SURF_DEFAULT)) NNNN = NNN
ENDDO

! Fire + evacuation calculation: draw obsts as outlines by default

IF (.NOT.OUTLINE .AND. EVACUATION_ONLY(NM) .AND. .NOT.ALL(EVACUATION_ONLY)) THEN
IF (SURFACE(NNNN)%TRANSPARENCY < 0.99999_EB .AND. .NOT.OUTLINE) THEN
OUTLINE = .FALSE.
ELSE
OUTLINE = .TRUE.
ENDIF
ENDIF

! Determine if the OBST is CONSUMABLE

FACE_LOOP: DO NNN=-3,3
IF (NNN==0) CYCLE FACE_LOOP
IF (SURFACE(OB%SURF_INDEX(NNN))%BURN_AWAY) OB%CONSUMABLE = .TRUE.
ENDDO FACE_LOOP

! Calculate the increase or decrease in the obstruction volume over the user-specified

VOL_SPECIFIED = (OB%X2-OB%X1)*(OB%Y2-OB%Y1)*(OB%Z2-OB%Z1)
VOL_ADJUSTED  = (X(OB%I2)-X(OB%I1))*(Y(OB%J2)-Y(OB%J1))*(Z(OB%K2)-Z(OB%K1))
IF (VOL_SPECIFIED>0._EB .AND..NOT.EVACUATION_ONLY(NM)) THEN
OB%VOLUME_ADJUST = VOL_ADJUSTED/VOL_SPECIFIED
ELSE
OB%VOLUME_ADJUST = 0._EB
ENDIF

! Creation and removal logic

OB%DEVC_ID = DEVC_ID
OB%CTRL_ID = CTRL_ID
OB%HIDDEN = .FALSE.

! Property ID

OB%PROP_ID = PROP_ID

CALL SEARCH_CONTROLLER('OBST',CTRL_ID,DEVC_ID,OB%DEVC_INDEX,OB%CTRL_INDEX,N)
IF (DEVC_ID /='null' .OR. CTRL_ID /='null') OB%REMOVABLE = .TRUE.

IF (OB%CONSUMABLE .AND..NOT.EVACUATION_ONLY(NM)) OB%REMOVABLE = .TRUE.

! Choose obstruction color index

SELECT CASE (COLOR)
CASE ('INVISIBLE')
OB%COLOR_INDICATOR = -3
RGB(1) = 255
RGB(2) = 204
RGB(3) = 102
TRANSPARENCY = 0._EB
CASE ('null')
IF (ANY (RGB<0)) THEN
OB%COLOR_INDICATOR = -1
ELSE
OB%COLOR_INDICATOR = -3
ENDIF
CASE DEFAULT
CALL COLOR2RGB(RGB,COLOR)
```

```
OB%COLOR_INDICATOR = -3
END SELECT
OB%RGB  = RGB
OB%TRANSPARENCY = TRANSPARENCY

! Miscellaneous assignments

OB%TEXTURE(:) = TEXTURE_ORIGIN(:)  ! Origin of texture map
OB%ORDINAL = NN  ! Order of OBST in original input file
OB%PERMIT_HOLE = PERMIT_HOLE
OB%ALLOW_VENT  = ALLOW_VENT
OB%OVERLAY     = OVERLAY

! Only allow the use of BULK_DENSITY if the obstruction has a non-zero volume

IF (EVACUATION_ONLY(NM)) BULK_DENSITY = -1._EB
OB%BULK_DENSITY = BULK_DENSITY
IF (ABS(OB%VOLUME_ADJUST)<TWO_EPSILON_EB .AND. OB%BULK_DENSITY>0._EB) OB%BULK_DENSITY = -1._EB

! Error traps and warnings for HT3D

IF (.NOT.HT3D .AND. ABS(INTERNAL_HEAT_SOURCE)>TWO_EPSILON_EB) THEN
WRITE(MESSAGE,'(A,I0,A)') 'ERROR: Problem with OBST number ',NN,', INTERNAL_HEAT_SOURCE requires HT3D=T.'
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF

! No HT3D for EVAC or zero volume OBST

IF (EVACUATION_ONLY(NM)) HT3D=.FALSE.
OB%HT3D = HT3D
IF (OB%HT3D .AND. ABS(OB%VOLUME_ADJUST)<TWO_EPSILON_EB) THEN
WRITE(LU_ERR,'(A,I0,A)') 'WARNING: OBST number ',NN,' has zero volume, consider THICKEN=T, HT3D set to F.'
OB%HT3D=.FALSE. ! later add capability for 2D lateral ht on thin obst
ENDIF

IF (OB%HT3D .AND. TRIM(MATL_ID)=='null') THEN
WRITE(MESSAGE,'(A,I0,A)') 'ERROR: Problem with OBST number ',NN,', HT3D requires MATL_ID.'
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF

! Set MATL_INDEX for HT3D

IF (OB%HT3D) THEN
OB%MATL_ID = MATL_ID
DO NNN=1,N_MATL
ML=>MATERIAL(NNN)
IF (TRIM(OB%MATL_ID)==TRIM(ML%ID)) THEN
OB%MATL_INDEX=NNN
OB%BULK_DENSITY=ML%RHO_S
EXIT
ENDIF
ENDDO
IF (OB%MATL_INDEX<0) THEN
WRITE(MESSAGE,'(A,I0,A)') 'ERROR: Problem with OBST number ',NN,', MATL_ID not found.'
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF
OB%INTERNAL_HEAT_SOURCE = INTERNAL_HEAT_SOURCE * 1000._EB ! W/m^3
ENDIF

! Make obstruction invisible if it's within a finer mesh

DO NOM=1,NM-1
IF (EVACUATION_ONLY(NOM)) CYCLE
IF (EVACUATION_ONLY(NM)) CYCLE
IF (XB1>MESHES(NOM)%XS .AND. XB2<MESHES(NOM)%XF .AND. &
XB3>MESHES(NOM)%YS .AND. XB4<MESHES(NOM)%YF .AND. &
XB5>MESHES(NOM)%ZS .AND. XB6<MESHES(NOM)%ZF) OB%COLOR_INDICATOR=-2
ENDDO

! Prevent drawing of boundary info if desired

IF (BNDF_DEFAULT) THEN
OB%SHOW_BNDF(:) = BNDF_FACE(:)
IF (.NOT.BNDF_OBST) OB%SHOW_BNDF(:) = .FALSE.
ELSE
OB%SHOW_BNDF(:) = BNDF_FACE(:)
IF (BNDF_OBST) OB%SHOW_BNDF(:) = .TRUE.
ENDIF
IF (EVACUATION_ONLY(NM)) OB%SHOW_BNDF(:) = .FALSE.

! In Smokeview, draw the outline of the obstruction

IF (OUTLINE) OB%TYPE_INDICATOR = 2

ENDDO I_MULT_LOOP
ENDDO J_MULT_LOOP
ENDDO K_MULT_LOOP
```

```
ENDDO READ_OBST_LOOP
35 REWIND(LU_INPUT) ; INPUT_FILE_LINE_NUMBER = 0

ENDDO MESH_LOOP

! Read HOLEs and cut out blocks

CALL READ_HOLE

! Look for OBSTructions that are meant to BURN_AWAY and break them up into single cell blocks

MESH_LOOP_2: DO NM=1,NMESHES

IF (PROCESS(NM)/=MYID .AND. MYID/=EVAC_PROCESS) CYCLE MESH_LOOP_2

M=>MESHES(NM)
CALL POINT_TO_MESH(NM)

N_OBST_O = N_OBST
DO N=1,N_OBST_O
OB => OBSTRUCTION(N)
IF (OB%CONSUMABLE .AND..NOT.EVACUATION_ONLY(NM)) THEN

N_NEW_OBST = MAX(1,OB%I2-OB%I1)*MAX(1,OB%J2-OB%J1)*MAX(1,OB%K2-OB%K1)
IF (N_NEW_OBST > 1) THEN

! Create a temporary array of obstructions with the same properties as the one being replaced, except coordinates

ALLOCATE(TEMP_OBSTRUCTION(N_NEW_OBST))
TEMP_OBSTRUCTION = OBSTRUCTION(N)
NN = 0
DO K=OB%K1,MAX(OB%K1,OB%K2-1)
DO J=OB%J1,MAX(OB%J1,OB%J2-1)
DO I=OB%I1,MAX(OB%I1,OB%I2-1)
NN = NN + 1
OBT=>TEMP_OBSTRUCTION(NN)
OBT%I1 = I
OBT%I2 = MIN(I+1,OB%I2)
OBT%J1 = J
OBT%J2 = MIN(J+1,OB%J2)
OBT%K1 = K
OBT%K2 = MIN(K+1,OB%K2)
OBT%X1 = M%X(OBT%I1)
OBT%X2 = M%X(OBT%I2)
OBT%Y1 = M%Y(OBT%J1)
OBT%Y2 = M%Y(OBT%J2)
OBT%Z1 = M%Z(OBT%K1)
OBT%Z2 = M%Z(OBT%K2)
ENDDO
ENDDO
ENDDO

CALL RE_ALLOCATE_OBST(NM,N_OBST,N_NEW_OBST-1)
OBSTRUCTION=>M%OBSTRUCTION
OBSTRUCTION(N) = TEMP_OBSTRUCTION(1)
OBSTRUCTION(N_OBST+1:N_OBST+N_NEW_OBST-1) = TEMP_OBSTRUCTION(2:N_NEW_OBST)
N_OBST = N_OBST + N_NEW_OBST-1
DEALLOCATE(TEMP_OBSTRUCTION)

ENDIF
ENDIF
ENDDO

ENDDO MESH_LOOP_2

! Allocate the number of cells for each mesh that are SOLID or border a boundary

ALLOCATE(CELL_COUNT(NMESHES)) ; CELL_COUNT = 0

! Go through all meshes, recording which cells are solid

MESH_LOOP_3: DO NM=1,NMESHES

IF (PROCESS(NM)/=MYID .AND. MYID/=EVAC_PROCESS) CYCLE MESH_LOOP_3

M=>MESHES(NM)
CALL POINT_TO_MESH(NM)

! Compute areas of obstruction faces, both actual (AB0) and FDS approximated (AB)

DO N=1,N_OBST
OB=>OBSTRUCTION(N)
OB%INPUT_AREA(1) = (OB%Y2-OB%Y1)*(OB%Z2-OB%Z1)
OB%INPUT_AREA(2) = (OB%X2-OB%X1)*(OB%Z2-OB%Z1)
OB%INPUT_AREA(3) = (OB%X2-OB%X1)*(OB%Y2-OB%Y1)
OB%FDS_AREA(1)   = (Y(OB%J2)-Y(OB%J1))*(Z(OB%K2)-Z(OB%K1))
OB%FDS_AREA(2)   = (X(OB%I2)-X(OB%I1))*(Z(OB%K2)-Z(OB%K1))
OB%FDS_AREA(3)   = (X(OB%I2)-X(OB%I1))*(Y(OB%J2)-Y(OB%J1))
```

```
OB%DIMENSIONS(1) = OB%I2 - OB%I1
OB%DIMENSIONS(2) = OB%J2 - OB%J1
OB%DIMENSIONS(3) = OB%K2 - OB%K1
ENDDO

! Create main blockage index array (ICA)

ALLOCATE(M%CELL_INDEX(0:IBP1,0:JBP1,0:KBP1),STAT=IZERO)
CALL ChkMemErr('READ','CELL_INDEX',IZERO) ; CELL_INDEX=>M%CELL_INDEX ; CELL_INDEX = 0

DO K=0,KBP1
IF (EVACUATION_ONLY(NM) .AND. .NOT.(K==1)) CYCLE
DO J=0,JBP1
DO I=0,1
IF (CELL_INDEX(I,J,K)==0) THEN
CELL_COUNT(NM) = CELL_COUNT(NM) + 1
CELL_INDEX(I,J,K) = CELL_COUNT(NM)
ENDIF
ENDDO
DO I=IBAR,IBP1
IF (CELL_INDEX(I,J,K)==0) THEN
CELL_COUNT(NM) = CELL_COUNT(NM) + 1
CELL_INDEX(I,J,K) = CELL_COUNT(NM)
ENDIF
ENDDO
ENDDO
ENDDO

DO K=0,KBP1
IF (EVACUATION_ONLY(NM) .AND. .NOT.(K==1)) CYCLE
DO I=0,IBP1
DO J=0,1
IF (CELL_INDEX(I,J,K)==0) THEN
CELL_COUNT(NM) = CELL_COUNT(NM) + 1
CELL_INDEX(I,J,K) = CELL_COUNT(NM)
ENDIF
ENDDO
DO J=JBAR,JBP1
IF (CELL_INDEX(I,J,K)==0) THEN
CELL_COUNT(NM) = CELL_COUNT(NM) + 1
CELL_INDEX(I,J,K) = CELL_COUNT(NM)
ENDIF
ENDDO
ENDDO
ENDDO

DO J=0,JBP1
DO I=0,IBP1
DO K=0,1
IF (EVACUATION_ONLY(NM) .AND. .NOT.(K==1)) CYCLE
IF (CELL_INDEX(I,J,K)==0) THEN
CELL_COUNT(NM) = CELL_COUNT(NM) + 1
CELL_INDEX(I,J,K) = CELL_COUNT(NM)
ENDIF
ENDDO
DO K=KBAR,KBP1
IF (EVACUATION_ONLY(NM) .AND. .NOT.(K==1)) CYCLE
IF (CELL_INDEX(I,J,K)==0) THEN
CELL_COUNT(NM) = CELL_COUNT(NM) + 1
CELL_INDEX(I,J,K) = CELL_COUNT(NM)
ENDIF
ENDDO
ENDDO
ENDDO

DO N=1,N_OBST
OB=>OBSTRUCTION(N)
DO K=OB%K1,OB%K2+1
IF (EVACUATION_ONLY(NM) .AND. .NOT.(K==1)) CYCLE
DO J=OB%J1,OB%J2+1
DO I=OB%I1,OB%I2+1
IF (CELL_INDEX(I,J,K)==0) THEN
CELL_COUNT(NM) = CELL_COUNT(NM) + 1
CELL_INDEX(I,J,K) = CELL_COUNT(NM)
ENDIF
ENDDO
ENDDO
ENDDO
ENDDO

! Store in SOLID which cells are solid and which are not

ALLOCATE(M%SOLID(0:CELL_COUNT(NM)),STAT=IZERO)
CALL ChkMemErr('READ','SOLID',IZERO)
M%SOLID = .FALSE.
ALLOCATE(M%EXTERIOR(0:CELL_COUNT(NM)),STAT=IZERO)
CALL ChkMemErr('READ','EXTERIOR',IZERO)
M%EXTERIOR = .FALSE.
```

```
SOLID=>M%SOLID
ALLOCATE(M%OBST_INDEX_C(0:CELL_COUNT(NM)),STAT=IZERO)
CALL ChkMemErr('READ','OBST_INDEX_C',IZERO)
M%OBST_INDEX_C = 0
OBST_INDEX_C=>M%OBST_INDEX_C

! Make all exterior cells solid

CALL BLOCK_CELL(NM,    0,    0,    0,JBP1,    0,KBP1,1,0)
CALL BLOCK_CELL(NM,IBP1,IBP1,    0,JBP1,    0,KBP1,1,0)
CALL BLOCK_CELL(NM,    0,IBP1,    0,    0,    0,KBP1,1,0)
CALL BLOCK_CELL(NM,    0,IBP1,JBP1,JBP1,    0,KBP1,1,0)
CALL BLOCK_CELL(NM,    0,IBP1,    0,JBP1,    0,    0,1,0)
CALL BLOCK_CELL(NM,    0,IBP1,    0,JBP1,KBP1,KBP1,1,0)

! Block off cells filled by obstructions

DO N=1,N_OBST
OB=>OBSTRUCTION(N)
CALL BLOCK_CELL(NM,OB%I1+1,OB%I2,OB%J1+1,OB%J2,OB%K1+1,OB%K2,1,N)
ENDDO

! Create arrays to hold cell indices

ALLOCATE(M%I_CELL(CELL_COUNT(NM)),STAT=IZERO)
CALL ChkMemErr('READ','I_CELL',IZERO)
M%I_CELL = -1
ALLOCATE(M%J_CELL(CELL_COUNT(NM)),STAT=IZERO)
CALL ChkMemErr('READ','J_CELL',IZERO)
M%J_CELL = -1
ALLOCATE(M%K_CELL(CELL_COUNT(NM)),STAT=IZERO)
CALL ChkMemErr('READ','K_CELL',IZERO)
M%K_CELL = -1
I_CELL=>M%I_CELL
J_CELL=>M%J_CELL
K_CELL=>M%K_CELL

DO K=0,KBP1
DO J=0,JBP1
DO I=0,IBP1
IC = CELL_INDEX(I,J,K)
IF (IC>0) THEN
I_CELL(IC) = I
J_CELL(IC) = J
K_CELL(IC) = K
ENDIF
ENDDO
ENDDO
ENDDO

ENDDO MESH_LOOP_3

CONTAINS

SUBROUTINE DEFINE_EVACUATION_OBSTS(NM,IMODE,EVAC_N)
!
! Define the evacuation OBSTs for the doors/exits, if needed.  A VENT should always
! be defined on an OBST that is at least one grid cell thick or the VENT should be
! on the outer boundary of the evacuation mesh, which is by default solid.
! The core of the STRS meshes are also defined.
!
USE EVAC, ONLY: N_DOORS, N_EXITS, N_CO_EXITS, EVAC_EMESH_EXITS_TYPE, EMESH_EXITS, &
N_STRS, EMESH_STAIRS, EVAC_EMESH_STAIRS_TYPE
IMPLICIT NONE
! Passed variables
INTEGER, INTENT(IN) :: NM, IMODE, EVAC_N
! Local variables
INTEGER :: N, N_END, I1, I2, J1, J2
REAL(EB) :: TINY

TINY = 0.1_EB*MIN(MESHES(NM)%DXI, MESHES(NM)%DETA)
N_END = N_EXITS - N_CO_EXITS + N_DOORS
IMODE_1_IF: IF (IMODE==1) THEN
NEND_LOOP_1: DO N = 1, N_END

IF (.NOT.EMESH_EXITS(N)%DEFINE_MESH) CYCLE NEND_LOOP_1
IF (EMESH_EXITS(N)%IMESH==NM .OR. EMESH_EXITS(N)%MAINMESH==NM) THEN
EMESH_EXITS(N)%I_OBST = 0

! Move EMESH_EXITS(N)%XB to mesh cell boundaries
EMESH_EXITS(N)%XB(1) = MAX(EMESH_EXITS(N)%XB(1),MESHES(NM)%XS)
EMESH_EXITS(N)%XB(2) = MIN(EMESH_EXITS(N)%XB(2),MESHES(NM)%XF)
EMESH_EXITS(N)%XB(3) = MAX(EMESH_EXITS(N)%XB(3),MESHES(NM)%YS)
EMESH_EXITS(N)%XB(4) = MIN(EMESH_EXITS(N)%XB(4),MESHES(NM)%YF)

I1 = NINT(GINV(EMESH_EXITS(N)%XB(1)-MESHES(NM)%XS,1,NM)*MESHES(NM)%RDXI  )
I2 = NINT(GINV(EMESH_EXITS(N)%XB(2)-MESHES(NM)%XS,1,NM)*MESHES(NM)%RDXI  )
J1 = NINT(GINV(EMESH_EXITS(N)%XB(3)-MESHES(NM)%YS,2,NM)*MESHES(NM)%RDETA)
```

```
J2 = NINT(GINV(EMESH_EXITS(N)%XB(4)-MESHES(NM)%YS,2,NM)*MESHES(NM)%RDETA)

EMESH_EXITS(N)%XB(1) = MESHES(NM)%X(I1)
EMESH_EXITS(N)%XB(2) = MESHES(NM)%X(I2)
EMESH_EXITS(N)%XB(3) = MESHES(NM)%Y(J1)
EMESH_EXITS(N)%XB(4) = MESHES(NM)%Y(J2)

! Check if the exit/door is at the mesh boundary, then no OBST is needed.
SELECT CASE (EMESH_EXITS(N)%IOR)
CASE (-1)
IF (MESHES(NM)%XS >= EMESH_EXITS(N)%XB(1)-TINY) CYCLE NEND_LOOP_1
CASE (+1)
IF (MESHES(NM)%XF <= EMESH_EXITS(N)%XB(2)+TINY) CYCLE NEND_LOOP_1
CASE (-2)
IF (MESHES(NM)%YS >= EMESH_EXITS(N)%XB(3)-TINY) CYCLE NEND_LOOP_1
CASE (+2)
IF (MESHES(NM)%YF <= EMESH_EXITS(N)%XB(4)+TINY) CYCLE NEND_LOOP_1
END SELECT
N_OBST = N_OBST + 1
EMESH_EXITS(N)%I_OBST = N_OBST
EVACUATION_OBST = .TRUE.
END IF
END DO NEND_LOOP_1

NSTRS_LOOP_1: DO N = 1, N_STRS

IF (.NOT.EMESH_STAIRS(N)%DEFINE_MESH) CYCLE NSTRS_LOOP_1
IF (EMESH_STAIRS(N)%IMESH==NM) THEN

! Move EMESH_STAIRS(N)%XB_CORE to mesh cell boundaries
EMESH_STAIRS(N)%XB_CORE(1) = MAX(EMESH_STAIRS(N)%XB_CORE(1),MESHES(NM)%XS)
EMESH_STAIRS(N)%XB_CORE(2) = MIN(EMESH_STAIRS(N)%XB_CORE(2),MESHES(NM)%XF)
EMESH_STAIRS(N)%XB_CORE(3) = MAX(EMESH_STAIRS(N)%XB_CORE(3),MESHES(NM)%YS)
EMESH_STAIRS(N)%XB_CORE(4) = MIN(EMESH_STAIRS(N)%XB_CORE(4),MESHES(NM)%YF)

I1 = NINT(GINV(EMESH_STAIRS(N)%XB_CORE(1)-MESHES(NM)%XS,1,NM)*MESHES(NM)%RDXI )
I2 = NINT(GINV(EMESH_STAIRS(N)%XB_CORE(2)-MESHES(NM)%XS,1,NM)*MESHES(NM)%RDXI )
J1 = NINT(GINV(EMESH_STAIRS(N)%XB_CORE(3)-MESHES(NM)%YS,2,NM)*MESHES(NM)%RDETA)
J2 = NINT(GINV(EMESH_STAIRS(N)%XB_CORE(4)-MESHES(NM)%YS,2,NM)*MESHES(NM)%RDETA)

EMESH_STAIRS(N)%XB_CORE(1) = MESHES(NM)%X(I1)
EMESH_STAIRS(N)%XB_CORE(2) = MESHES(NM)%X(I2)
EMESH_STAIRS(N)%XB_CORE(3) = MESHES(NM)%Y(J1)
EMESH_STAIRS(N)%XB_CORE(4) = MESHES(NM)%Y(J2)

N_OBST = N_OBST + 1
EMESH_STAIRS(N)%I_OBST = N_OBST
EVACUATION_OBST = .TRUE.
END IF
END DO NSTRS_LOOP_1

END IF IMODE_1_IF

IMODE_2_IF: IF (IMODE==2) THEN
NEND_LOOP_2: DO N = 1, N_END
IF (.NOT.EMESH_EXITS(N)%DEFINE_MESH) CYCLE NEND_LOOP_2
IF (EMESH_EXITS(N)%I_OBST==EVAC_N .AND. (EMESH_EXITS(N)%IMESH==NM .OR. EMESH_EXITS(N)%MAINMESH==NM)) THEN
EVACUATION_OBST = .TRUE.
EVACUATION = .TRUE.
REMOVABLE = .FALSE.
THICKEN = .TRUE.
PERMIT_HOLE = .FALSE.
ALLOW_VENT = .TRUE.
MESH_ID = TRIM(MESH_NAME(NM))
XB(1) = EMESH_EXITS(N)%XB(1)
XB(2) = EMESH_EXITS(N)%XB(2)
XB(3) = EMESH_EXITS(N)%XB(3)
XB(4) = EMESH_EXITS(N)%XB(4)
XB(5) = EMESH_EXITS(N)%XB(5)
XB(6) = EMESH_EXITS(N)%XB(6)
SELECT CASE (EMESH_EXITS(N)%IOR)
CASE (-1)
XB(1) = MAX(MESHES(NM)%XS, XB(1) - 0.49_EB*MESHES(NM)%DXI)
CASE (+1)
XB(2) = MIN(MESHES(NM)%XF, XB(2) + 0.49_EB*MESHES(NM)%DXI)
CASE (-2)
XB(3) = MAX(MESHES(NM)%YS, XB(3) - 0.49_EB*MESHES(NM)%DETA)
CASE (+2)
XB(4) = MIN(MESHES(NM)%YF, XB(4) + 0.49_EB*MESHES(NM)%DETA)
END SELECT
RGB(:) = EMESH_EXITS(N)%RGB(:)
ID = TRIM('Eobst_' // TRIM(MESH_NAME(NM)))
END IF
END DO NEND_LOOP_2

NSTRS_LOOP_2: DO N = 1, N_STRS
IF (.NOT.EMESH_STAIRS(N)%DEFINE_MESH) CYCLE NSTRS_LOOP_2
IF (EMESH_STAIRS(N)%I_OBST==EVAC_N .AND. EMESH_STAIRS(N)%IMESH==NM) THEN
```

```
EVACUATION_OBST = .TRUE.
EVACUATION = .TRUE.
REMOVABLE = .FALSE.
! THICKEN = .TRUE.
PERMIT_HOLE = .FALSE.
ALLOW_VENT = .FALSE.
MESH_ID = TRIM(MESH_NAME(NM))
XB(1) = EMESH_STAIRS(N)%XB_CORE(1)
XB(2) = EMESH_STAIRS(N)%XB_CORE(2)
XB(3) = EMESH_STAIRS(N)%XB_CORE(3)
XB(4) = EMESH_STAIRS(N)%XB_CORE(4)
XB(5) = EMESH_STAIRS(N)%XB(5)
XB(6) = EMESH_STAIRS(N)%XB(6)
RGB(:) = EMESH_STAIRS(N)%RGB(:)
ID = TRIM('Eobst_' // TRIM(MESH_NAME(NM)))
END IF
END DO NSTRS_LOOP_2

END IF IMODE_2_IF

RETURN
END SUBROUTINE DEFINE_EVACUATION_OBSTS

END SUBROUTINE READ_OBST


SUBROUTINE READ_HOLE

CHARACTER(LABEL_LENGTH) :: DEVC_ID,CTRL_ID,MULT_ID
CHARACTER(60) :: MESH_ID
CHARACTER(25) :: COLOR
LOGICAL :: EVACUATION_HOLE,EVACUATION,BLOCK_WIND
INTEGER :: NM,N_HOLE,NN,NDO,N,I1,I2,J1,J2,K1,K2,RGB(3),N_HOLE_NEW,N_HOLE_O,II,JJ,KK,NNN,DEVC_INDEX_O,CTRL_INDEX_O
REAL(EB) :: X1,X2,Y1,Y2,Z1,Z2,TRANSPARENCY
NAMELIST /HOLE/ BLOCK_WIND,COLOR,CTRL_ID,DEVC_ID,EVACUATION,FYI,ID,MESH_ID,MULT_ID,RGB,TRANSPARENCY,XB
REAL(EB), ALLOCATABLE, DIMENSION(:,:) :: TEMP_XB
LOGICAL, ALLOCATABLE, DIMENSION(:) :: CONTROLLED
TYPE(OBSTRUCTION_TYPE), ALLOCATABLE, DIMENSION(:) :: TEMP_OBST
TYPE(MULTIPLIER_TYPE), POINTER :: MR=>NULL()
LOGICAL, ALLOCATABLE, DIMENSION(:) :: TEMP_HOLE_EVAC

ALLOCATE(TEMP_OBST(0:6))

N_HOLE   = 0
N_HOLE_O = 0
REWIND(LU_INPUT) ; INPUT_FILE_LINE_NUMBER = 0

COUNT_LOOP: DO
CALL CHECKREAD('HOLE',LU_INPUT,IOS)
IF (IOS==1) EXIT COUNT_LOOP
MULT_ID = 'null'
READ(LU_INPUT,NML=HOLE,END=1,ERR=2,IOSTAT=IOS)
N_HOLE_O = N_HOLE_O + 1
N_HOLE_NEW = 0
IF (MULT_ID=='null') THEN
N_HOLE_NEW = 1
ELSE
DO N=1,N_MULT
MR => MULTIPLIER(N)
IF (MULT_ID==MR%ID) N_HOLE_NEW = MR%N_COPIES
ENDDO
IF (N_HOLE_NEW==0) THEN
WRITE(MESSAGE,'(A,A,A,I0)') 'ERROR: MULT line ', TRIM(MULT_ID),' not found on HOLE line', N_HOLE_O
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF
ENDIF
N_HOLE  = N_HOLE  + N_HOLE_NEW
2 IF (IOS>0) THEN
WRITE(MESSAGE,'(A,I0,A,I0)') 'ERROR: Problem with HOLE number',N_HOLE_O+1,', line number',INPUT_FILE_LINE_NUMBER
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF
ENDDO COUNT_LOOP
1 REWIND(LU_INPUT) ; INPUT_FILE_LINE_NUMBER = 0

CALL DEFINE_EVACUATION_HOLES(1)

ALLOCATE (TEMP_XB(N_HOLE_O,6))
TEMP_XB = 0._EB
ALLOCATE (CONTROLLED(N_HOLE_O))
CONTROLLED = .FALSE.

! TEMP_HOLE_EVAC(:) indicates if the given HOLE is to be used in the EVACUATION routine

IF (ANY(EVACUATION_ONLY)) THEN
ALLOCATE(TEMP_HOLE_EVAC(1:N_HOLE_O))
READ_HOLE_EVAC_LOOP: DO N=1,N_HOLE_O
EVACUATION_HOLE = .FALSE.
CALL DEFINE_EVACUATION_HOLES(2)
```

```
EVACUATION = .TRUE.
IF (.NOT.EVACUATION_HOLE) THEN
CALL CHECKREAD('HOLE',LU_INPUT,IOS)
IF (IOS==1) EXIT READ_HOLE_EVAC_LOOP
READ(LU_INPUT,HOLE)
END IF
TEMP_HOLE_EVAC(N) = EVACUATION
ENDDO READ_HOLE_EVAC_LOOP
REWIND(LU_INPUT) ; INPUT_FILE_LINE_NUMBER = 0
ENDIF

READ_HOLE_LOOP: DO N=1,N_HOLE_O

! Set default values for the HOLE namelist parameters

BLOCK_WIND = .FALSE.
DEVC_ID    = 'null'
CTRL_ID    = 'null'
ID         = 'null'
MESH_ID    = 'null'
MULT_ID    = 'null'
COLOR      = 'null'
RGB        = -1
TRANSPARENCY = 1._EB
EVACUATION   = .FALSE.
XB         = -9.E30_EB

! Read the HOLE line

EVACUATION_HOLE = .FALSE.
IF (ANY(EVACUATION_ONLY)) CALL DEFINE_EVACUATION_HOLES(3)
EVACUATION_HOLES: IF (.NOT. EVACUATION_HOLE) THEN
CALL CHECKREAD('HOLE',LU_INPUT,IOS)
IF (IOS==1) EXIT READ_HOLE_LOOP
READ(LU_INPUT,HOLE)
END IF EVACUATION_HOLES

! Re-order coordinates, if necessary

CALL CHECK_XB(XB)
TEMP_XB(N,:) = XB
! Check for overlap if controlled
IF (DEVC_ID/='null' .OR. CTRL_ID/='null') CONTROLLED(N) = .TRUE.
IF (N>1) THEN
DO NN = 1,N-1
IF (TEMP_XB(NN,1) >= XB(2) .OR. TEMP_XB(NN,3) >= XB(4) .OR. TEMP_XB(NN,5) >= XB(6) .OR. &
TEMP_XB(NN,2) <= XB(1) .OR. TEMP_XB(NN,4) <= XB(3) .OR. TEMP_XB(NN,6) <= XB(5)) CYCLE
IF ((TEMP_XB(NN,1) <= XB(2) .AND. TEMP_XB(NN,2) >= XB(1)) .AND. &
(TEMP_XB(NN,3) <= XB(4) .AND. TEMP_XB(NN,4) >= XB(3)) .AND. &
(TEMP_XB(NN,5) <= XB(6) .AND. TEMP_XB(NN,6) >= XB(5))) THEN
IF (CONTROLLED(N) .OR. CONTROLLED(NN)) THEN
WRITE(MESSAGE,'(A,I0,A,I0)') 'ERROR: Cannot overlap HOLEs with a DEVC_ID or CTRL_ID. HOLE number',N_HOLE_O+1,&
', line number',INPUT_FILE_LINE_NUMBER
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF
ENDIF
ENDDO
ENDIF

! Loop over all the meshes to determine where the HOLE is

MESH_LOOP: DO NM=1,NMESHES

IF (PROCESS(NM)/=MYID .AND. MYID/=EVAC_PROCESS) CYCLE MESH_LOOP

M=>MESHES(NM)
CALL POINT_TO_MESH(NM)

! Evacuation criteria

IF (MESH_ID/='null' .AND. MESH_ID/=MESH_NAME(NM)) CYCLE MESH_LOOP
IF (EVACUATION .AND. .NOT.EVACUATION_ONLY(NM)) CYCLE MESH_LOOP
IF (EVACUATION_ONLY(NM)) THEN
IF (.NOT.TEMP_HOLE_EVAC(N)) CYCLE MESH_LOOP
DEVC_ID    = 'null'
CTRL_ID    = 'null'
ENDIF

! Loop over all possible multiples of the HOLE

MR => MULTIPLIER(0)
DO NNN=1,N_MULT
IF (MULT_ID==MULTIPLIER(NNN)%ID) MR => MULTIPLIER(NNN)
ENDDO

K_MULT_LOOP: DO KK=MR%K_LOWER,MR%K_UPPER
J_MULT_LOOP: DO JJ=MR%J_LOWER,MR%J_UPPER
I_MULT_LOOP: DO II=MR%I_LOWER,MR%I_UPPER
```

```
IF (.NOT.MR%SEQUENTIAL) THEN
X1 = XB(1) + MR%DX0 + II*MR%DXB(1)
X2 = XB(2) + MR%DX0 + II*MR%DXB(2)
Y1 = XB(3) + MR%DY0 + JJ*MR%DXB(3)
Y2 = XB(4) + MR%DY0 + JJ*MR%DXB(4)
Z1 = XB(5) + MR%DZ0 + KK*MR%DXB(5)
Z2 = XB(6) + MR%DZ0 + KK*MR%DXB(6)
ELSE
X1 = XB(1) + MR%DX0 + II*MR%DXB(1)
X2 = XB(2) + MR%DX0 + II*MR%DXB(2)
Y1 = XB(3) + MR%DY0 + II*MR%DXB(3)
Y2 = XB(4) + MR%DY0 + II*MR%DXB(4)
Z1 = XB(5) + MR%DZ0 + II*MR%DXB(5)
Z2 = XB(6) + MR%DZ0 + II*MR%DXB(6)
ENDIF

! Check if hole is contained within the current mesh

IF (X1>=XF .OR. X2<=XS .OR. Y1>YF .OR. Y2<=YS .OR. Z1>ZF .OR. Z2<=ZS) CYCLE I_MULT_LOOP

! Assign mesh-limited bounds

X1 = MAX(X1,XS-0.001_EB*DX(0))
X2 = MIN(X2,XF+0.001_EB*DX(IBP1))
Y1 = MAX(Y1,YS-0.001_EB*DY(0))
Y2 = MIN(Y2,YF+0.001_EB*DY(JBP1))
Z1 = MAX(Z1,ZS-0.001_EB*DZ(0))
Z2 = MIN(Z2,ZF+0.001_EB*DZ(KBP1))

I1 = NINT( GINV(X1-XS,1,NM)*RDXI    )
I2 = NINT( GINV(X2-XS,1,NM)*RDXI    )
J1 = NINT( GINV(Y1-YS,2,NM)*RDETA   )
J2 = NINT( GINV(Y2-YS,2,NM)*RDETA   )
K1 = NINT( GINV(Z1-ZS,3,NM)*RDZETA  )
K2 = NINT( GINV(Z2-ZS,3,NM)*RDZETA  )

! Remove mean forcing in hole region

IF (ANY(MEAN_FORCING) .AND. BLOCK_WIND) THEN
DO K=K1,K2+1
DO J=J1,J2+1
DO I=I1,I2+1
M%MEAN_FORCING_CELL(I,J,K) = .FALSE.
ENDDO
ENDDO
ENDDO
CYCLE I_MULT_LOOP
ENDIF

NN=0
OBST_LOOP: DO
NN=NN+1
IF (NN>N_OBST) EXIT OBST_LOOP
OB=>OBSTRUCTION(NN)
DEVC_INDEX_O = OB%DEVC_INDEX
CTRL_INDEX_O = OB%CTRL_INDEX
IF (.NOT.OB%PERMIT_HOLE) CYCLE OBST_LOOP

! TEMP_OBST(0) is the intersection of HOLE and OBST

TEMP_OBST(0)      = OBSTRUCTION(NN)

TEMP_OBST(0)%I1 = MAX(I1,OB%I1)
TEMP_OBST(0)%I2 = MIN(I2,OB%I2)
TEMP_OBST(0)%J1 = MAX(J1,OB%J1)
TEMP_OBST(0)%J2 = MIN(J2,OB%J2)
TEMP_OBST(0)%K1 = MAX(K1,OB%K1)
TEMP_OBST(0)%K2 = MIN(K2,OB%K2)

TEMP_OBST(0)%X1 = MAX(X1,OB%X1)
TEMP_OBST(0)%X2 = MIN(X2,OB%X2)
TEMP_OBST(0)%Y1 = MAX(Y1,OB%Y1)
TEMP_OBST(0)%Y2 = MIN(Y2,OB%Y2)
TEMP_OBST(0)%Z1 = MAX(Z1,OB%Z1)
TEMP_OBST(0)%Z2 = MIN(Z2,OB%Z2)

! Ignore OBSTs that do not intersect with HOLE or are merely sliced by the hole.

IF (TEMP_OBST(0)%I2-TEMP_OBST(0)%I1<0 .OR. TEMP_OBST(0)%J2-TEMP_OBST(0)%J1<0 .OR. &
TEMP_OBST(0)%K2-TEMP_OBST(0)%K1<0) CYCLE OBST_LOOP
IF (TEMP_OBST(0)%I2-TEMP_OBST(0)%I1==0) THEN
IF (OB%I1<TEMP_OBST(0)%I1 .OR. OB%I2>TEMP_OBST(0)%I2) CYCLE OBST_LOOP
ENDIF
IF (TEMP_OBST(0)%J2-TEMP_OBST(0)%J1==0) THEN
IF (OB%J1<TEMP_OBST(0)%J1 .OR. OB%J2>TEMP_OBST(0)%J2) CYCLE OBST_LOOP
ENDIF
IF (TEMP_OBST(0)%K2-TEMP_OBST(0)%K1==0) THEN
```

```
IF (OB%K1<TEMP_OBST(0)%K1 .OR. OB%K2>TEMP_OBST(0)%K2) CYCLE OBST_LOOP
ENDIF

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. &
TEMP_OBST(0)%Z2<=Z1 .OR. TEMP_OBST(0)%Z1>=Z2) CYCLE OBST_LOOP

! Start counting new OBSTs that need to be created

NDO=0

IF ((OB%I1<I1 .AND. I1<OB%I2) .OR. (XB(1)>=XS.AND. I1==0.AND.OB%I1==0)) THEN
NDO=NDO+1
TEMP_OBST(NDO)=OBSTRUCTION(NN)
TEMP_OBST(NDO)%I1 = OB%I1
TEMP_OBST(NDO)%I2 = I1
TEMP_OBST(NDO)%X1 = OB%X1
TEMP_OBST(NDO)%X2 = X1
ENDIF

IF ((OB%I1<I2 .AND. I2<OB%I2) .OR. (XB(2)<=XF.AND. I2==IBAR.AND.OB%I2==IBAR)) THEN
NDO=NDO+1
TEMP_OBST(NDO)=OBSTRUCTION(NN)
TEMP_OBST(NDO)%I1 = I2
TEMP_OBST(NDO)%I2 = OB%I2
TEMP_OBST(NDO)%X1 = X2
TEMP_OBST(NDO)%X2 = OB%X2
ENDIF

IF ((OB%J1<J1 .AND. J1<OB%J2) .OR. (XB(3)>=YS.AND. J1==0.AND.OB%J1==0)) THEN
NDO=NDO+1
TEMP_OBST(NDO)=OBSTRUCTION(NN)
TEMP_OBST(NDO)%I1 = MAX(I1,OB%I1)
TEMP_OBST(NDO)%I2 = MIN(I2,OB%I2)
TEMP_OBST(NDO)%X1 = MAX(X1,OB%X1)
TEMP_OBST(NDO)%X2 = MIN(X2,OB%X2)
TEMP_OBST(NDO)%J1 = OB%J1
TEMP_OBST(NDO)%J2 = J1
TEMP_OBST(NDO)%Y1 = OB%Y1
TEMP_OBST(NDO)%Y2 = Y1
ENDIF

IF ((OB%J1<J2 .AND. J2<OB%J2) .OR. (XB(4)<=YF.AND. J2==JBAR.AND.OB%J2==JBAR)) THEN
NDO=NDO+1
TEMP_OBST(NDO)=OBSTRUCTION(NN)
TEMP_OBST(NDO)%I1 = MAX(I1,OB%I1)
TEMP_OBST(NDO)%I2 = MIN(I2,OB%I2)
TEMP_OBST(NDO)%X1 = MAX(X1,OB%X1)
TEMP_OBST(NDO)%X2 = MIN(X2,OB%X2)
TEMP_OBST(NDO)%J1 = J2
TEMP_OBST(NDO)%J2 = OB%J2
TEMP_OBST(NDO)%Y1 = Y2
TEMP_OBST(NDO)%Y2 = OB%Y2
ENDIF

IF ((OB%K1<K1 .AND. K1<OB%K2) .OR. (XB(5)>=ZS.AND. K1==0.AND.OB%K1==0)) THEN
NDO=NDO+1
TEMP_OBST(NDO)=OBSTRUCTION(NN)
TEMP_OBST(NDO)%I1 = MAX(I1,OB%I1)
TEMP_OBST(NDO)%I2 = MIN(I2,OB%I2)
TEMP_OBST(NDO)%X1 = MAX(X1,OB%X1)
TEMP_OBST(NDO)%X2 = MIN(X2,OB%X2)
TEMP_OBST(NDO)%J1 = MAX(J1,OB%J1)
TEMP_OBST(NDO)%J2 = MIN(J2,OB%J2)
TEMP_OBST(NDO)%Y1 = MAX(Y1,OB%Y1)
TEMP_OBST(NDO)%Y2 = MIN(Y2,OB%Y2)
TEMP_OBST(NDO)%K1 = OB%K1
TEMP_OBST(NDO)%K2 = K1
TEMP_OBST(NDO)%Z1 = OB%Z1
TEMP_OBST(NDO)%Z2 = Z1
ENDIF

IF ((OB%K1<K2 .AND. K2<OB%K2) .OR. (XB(6)<=ZF.AND. K2==KBAR.AND.OB%K2==KBAR)) THEN
NDO=NDO+1
TEMP_OBST(NDO)=OBSTRUCTION(NN)
TEMP_OBST(NDO)%I1 = MAX(I1,OB%I1)
TEMP_OBST(NDO)%I2 = MIN(I2,OB%I2)
TEMP_OBST(NDO)%X1 = MAX(X1,OB%X1)
TEMP_OBST(NDO)%X2 = MIN(X2,OB%X2)
TEMP_OBST(NDO)%J1 = MAX(J1,OB%J1)
TEMP_OBST(NDO)%J2 = MIN(J2,OB%J2)
TEMP_OBST(NDO)%Y1 = MAX(Y1,OB%Y1)
TEMP_OBST(NDO)%Y2 = MIN(Y2,OB%Y2)
TEMP_OBST(NDO)%K1 = K2
TEMP_OBST(NDO)%K2 = OB%K2
TEMP_OBST(NDO)%Z1 = Z2
TEMP_OBST(NDO)%Z2 = OB%Z2
ENDIF

```

```
! Maintain ordinal rank of original obstruction, but negate it. This will be a code for Smokeview.

TEMP_OBST(:)%ORDINAL = -OB%ORDINAL

! Re-allocate space of new OBSTs, or remove entry for dead OBST

NEW_OBST_IF: IF (NDO>0) THEN
CALL RE_ALLOCATE_OBST(NM,N_OBST,NDO)
OBSTRUCTION=>M%OBSTRUCTION
OBSTRUCTION(N_OBST+1:N_OBST+NDO) = TEMP_OBST(1:NDO)
N_OBST = N_OBST + NDO
ENDIF NEW_OBST_IF

! If the HOLE is to be created or removed, save it in OBSTRUCTION(NN), the original OBST that was broken up

DEVC_OR_CTRL: IF (DEVC_ID/='null' .OR. CTRL_ID/='null') THEN

OBSTRUCTION(NN) = TEMP_OBST(0)
OB => OBSTRUCTION(NN)
OB%DEVC_INDEX_O = DEVC_INDEX_O
OB%CTRL_INDEX_O = CTRL_INDEX_O
OB%DEVC_ID = DEVC_ID
OB%CTRL_ID = CTRL_ID
CALL SEARCH_CONTROLLER('HOLE',CTRL_ID,DEVC_ID,OB%DEVC_INDEX,OB%CTRL_INDEX,N)
IF (DEVC_ID/='null' .OR. CTRL_ID /='null') THEN
OB%REMOVABLE    = .TRUE.
OB%HOLE_FILLER = .TRUE.
IF (DEVC_ID/='null') OB%CTRL_INDEX = -1
IF (CTRL_ID/='null') OB%DEVC_INDEX = -1
ENDIF
IF (OB%CONSUMABLE)      OB%REMOVABLE = .TRUE.

SELECT CASE (COLOR)
CASE ('INVISIBLE')
OB%COLOR_INDICATOR = -3
OB%RGB(1) = 255
OB%RGB(2) = 204
OB%RGB(3) = 102
OB%TRANSPARENCY = 0._EB
CASE ('null')
IF (ANY(RGB>0)) THEN
OB%COLOR_INDICATOR = -3
OB%RGB  = RGB
OB%TRANSPARENCY = TRANSPARENCY
ENDIF
CASE DEFAULT
CALL COLOR2RGB(RGB,COLOR)
OB%COLOR_INDICATOR = -3
OB%RGB  = RGB
OB%TRANSPARENCY = TRANSPARENCY
END SELECT

ELSE DEVC_OR_CTRL

OBSTRUCTION(NN) = OBSTRUCTION(N_OBST)
N_OBST = N_OBST-1
NN = NN-1

ENDIF DEVC_OR_CTRL

ENDDO OBST_LOOP
ENDDO I_MULT_LOOP
ENDDO J_MULT_LOOP
ENDDO K_MULT_LOOP
ENDDO MESH_LOOP
ENDDO READ_HOLE_LOOP

REWIND(LU_INPUT) ; INPUT_FILE_LINE_NUMBER = 0

IF(ANY(EVACUATION_ONLY)) DEALLOCATE(TEMP_HOLE_EVAC)
DEALLOCATE(TEMP_OBST)
DEALLOCATE (CONTROLLED)
DEALLOCATE (TEMP_XB)

CONTAINS

SUBROUTINE DEFINE_EVACUATION_HOLES(IMODE)
!
! Clear the STRS meshes by a hole with size of XB of the stairs.
! The core is put there applying permit_hole=false.
USE EVAC, ONLY: N_STRS, EMESH_STAIRS, EVAC_EMESH_STAIRS_TYPE
IMPLICIT NONE
! Passed variables
INTEGER, INTENT(IN) :: IMODE
! Local variables
INTEGER :: I
REAL(EB) :: TINY_X, TINY_Y, TINY_Z
```

```
IF (.NOT.ANY(EVACUATION_ONLY)) RETURN
IMODE_1_IF: IF (IMODE==1) THEN
NSTRS_LOOP_1: DO I = 1, N_STRS
IF (.NOT.EMESH_STAIRS(I)%DEFINE_MESH) CYCLE NSTRS_LOOP_1
N_HOLE_O = N_HOLE_O + 1
N_HOLE    = N_HOLE    + 1 ! No mult for evacuation strs meshes
EMESH_STAIRS(I)%I_HOLE = N_HOLE_O
EVACUATION_HOLE = .TRUE.
END DO NSTRS_LOOP_1
END IF IMODE_1_IF

IMODE_2_IF: IF (IMODE==2) THEN
EVACUATION_HOLE = .FALSE.
NSTRS_LOOP_2: DO I = 1, N_STRS
IF (.NOT.EMESH_STAIRS(I)%DEFINE_MESH) CYCLE NSTRS_LOOP_2
IF (.NOT.EMESH_STAIRS(I)%I_HOLE==N) CYCLE NSTRS_LOOP_2
EVACUATION_HOLE = .TRUE.
EXIT NSTRS_LOOP_2
END DO NSTRS_LOOP_2
END IF IMODE_2_IF

IMODE_3_IF: IF (IMODE==3) THEN
EVACUATION_HOLE = .FALSE.
NSTRS_LOOP_3: DO I = 1, N_STRS
IF (.NOT.EMESH_STAIRS(I)%DEFINE_MESH) CYCLE NSTRS_LOOP_3
IF (.NOT.EMESH_STAIRS(I)%I_HOLE==N) CYCLE NSTRS_LOOP_3
EVACUATION_HOLE = .TRUE.
RGB = EMESH_STAIRS(I)%RGB
XB = EMESH_STAIRS(I)%XB
TINY_X = 0.01_EB*(EMESH_STAIRS(I)%XB(2)-EMESH_STAIRS(I)%XB(1))/EMESH_STAIRS(I)%IBAR
TINY_Y = 0.01_EB*(EMESH_STAIRS(I)%XB(4)-EMESH_STAIRS(I)%XB(3))/EMESH_STAIRS(I)%JBAR
TINY_Z = 0.01_EB
XB(1) = XB(1)-TINY_X ; XB(2) = XB(2)+TINY_X
XB(3) = XB(3)-TINY_Y ; XB(4) = XB(4)+TINY_Y
XB(5) = XB(5)-TINY_Z ; XB(6) = XB(6)+TINY_Z
EVACUATION = .TRUE.
MESH_ID = TRIM(MESH_NAME(EMESH_STAIRS(I)%IMESH))
EXIT NSTRS_LOOP_3
END DO NSTRS_LOOP_3
END IF IMODE_3_IF

END SUBROUTINE DEFINE_EVACUATION_HOLES

END SUBROUTINE READ_HOLE


SUBROUTINE RE_ALLOCATE_OBST(NM,N_OBST,NOO)

TYPE (OBSTRUCTION_TYPE), ALLOCATABLE, DIMENSION(:) :: DUMMY
INTEGER, INTENT(IN) :: NM,NOO,N_OBST
TYPE (MESH_TYPE), POINTER :: M=>NULL()
M=>MESHES(NM)
ALLOCATE(DUMMY(0:N_OBST))
DUMMY(0:N_OBST) = M%OBSTRUCTION(0:N_OBST)
DEALLOCATE(M%OBSTRUCTION)
ALLOCATE(M%OBSTRUCTION(0:N_OBST+NOO))
M%OBSTRUCTION(0:N_OBST) = DUMMY(0:N_OBST)
DEALLOCATE(DUMMY)
END SUBROUTINE RE_ALLOCATE_OBST


SUBROUTINE READ_VENT

USE GEOMETRY_FUNCTIONS, ONLY : BLOCK_CELL,CIRCLE_CELL_INTERSECTION_AREA
USE DEVICE_VARIABLES, ONLY : DEVICE
USE CONTROL_VARIABLES, ONLY : CONTROL
USE MATH_FUNCTIONS, ONLY: GET_RAMP_INDEX

INTEGER :: N,NN,NM,NNN,N_VENT_O,IOR,I1,I2,J1,J2,K1,K2,RGB(3),N_EDDY,N_VENT_NEW,II,JJ,KK
REAL(EB) :: SPREAD_RATE,TRANSPARENCY,XYZ(3),TMP_EXTERIOR,DYNAMIC_PRESSURE,XB1,XB2,XB3,XB4,XB5,XB6, &
REYNOLDS_STRESS(3,3),L_EDDY,VEL_RMS,L_EDDY_IJ(3,3),UVW(3),RADIUS
CHARACTER(LABEL_LENGTH) :: ID,DEVC_ID,CTRL_ID,SURF_ID,PRESSURE_RAMP,TMP_EXTERIOR_RAMP,MULT_ID
CHARACTER(60) :: MESH_ID
CHARACTER(25) :: COLOR
TYPE(MULTIPLIER_TYPE), POINTER :: MR
LOGICAL :: REJECT_VENT,EVACUATION,OUTLINE,EVACUATION_VENT,WIND
NAMELIST /VENT/ COLOR,CTRL_ID,DEVC_ID,DYNAMIC_PRESSURE,EVACUATION,FYI,ID,IOR,L_EDDY,L_EDDY_IJ,MB,MESH_ID,MULT_ID,
        N_EDDY,OUTLINE,&
PBX,PBY,PBZ,PRESSURE_RAMP,RADIUS,REYNOLDS_STRESS,RGB,SPREAD_RATE,SURF_ID,TEXTURE_ORIGIN,TMP_EXTERIOR,&
TMP_EXTERIOR_RAMP,TRANSPARENCY,UVW,VEL_RMS,WIND,XB,XYZ

MESH_LOOP_1: DO NM=1,NMESHES

M=>MESHES(NM)
CALL POINT_TO_MESH(NM)

REWIND(LU_INPUT) ; INPUT_FILE_LINE_NUMBER = 0
N_VENT = 0
```

```
COUNT_VENT_LOOP: DO
CALL CHECKREAD('VENT',LU_INPUT,IOS)
IF (IOS==1) EXIT COUNT_VENT_LOOP
ID       = 'null'
MULT_ID  = 'null'
SURF_ID  = 'null'
READ(LU_INPUT,NML=VENT,END=3,ERR=4,IOSTAT=IOS)
N_VENT_NEW = 0
IF (MULT_ID=='null') THEN
N_VENT_NEW = 1
ELSE
IF (SURF_ID=='HVAC') THEN
WRITE(MESSAGE,'(A,I0,A,I0)') 'ERROR: Cannot use MULT with an HVAC VENT, VENT ', N_VENT+1,&
', line number',INPUT_FILE_LINE_NUMBER
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF
DO N=1,N_MULT
MR => MULTIPLIER(N)
IF (MULT_ID==MR%ID) N_VENT_NEW = MR%N_COPIES
ENDDO
IF (N_VENT_NEW==0) THEN
WRITE(MESSAGE,'(A,A,A,I0,A,I0)') 'ERROR: MULT line ', TRIM(MULT_ID),' not found on VENT ', N_VENT+1,&
', line number',INPUT_FILE_LINE_NUMBER
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF
ENDIF
IF (SURF_ID=='HVAC' .AND. ID=='null') THEN
WRITE(MESSAGE,'(A,I0,A,I0)') 'ERROR: must specify an ID for an HVAC VENT, VENT ', N_VENT+1,&
', line number',INPUT_FILE_LINE_NUMBER
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF
N_VENT = N_VENT + N_VENT_NEW
4 IF (IOS>0) THEN
WRITE(MESSAGE,'(A,I0,A,I0)') 'ERROR: Problem with VENT ',N_VENT+1,', line number',INPUT_FILE_LINE_NUMBER
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF
ENDDO COUNT_VENT_LOOP
3 REWIND(LU_INPUT) ; INPUT_FILE_LINE_NUMBER = 0

IF (EVACUATION_ONLY(NM)) CALL DEFINE_EVACUATION_VENTS(NM,1)

IF (TWO_D)                              N_VENT = N_VENT + 2
IF (CYLINDRICAL .AND. M%XS<=TWO_EPSILON_EB) N_VENT = N_VENT + 1
IF (EVACUATION_ONLY(NM))                N_VENT = N_VENT + 2

ALLOCATE(M%VENTS(N_VENT),STAT=IZERO)
CALL ChkMemErr('READ','VENTS',IZERO)
VENTS=>M%VENTS

N_VENT_O = N_VENT
N        = 0

REWIND(LU_INPUT) ; INPUT_FILE_LINE_NUMBER = 0
READ_VENT_LOOP: DO NN=1,N_VENT_O

IOR      = 0
MB       = 'null'
PBX      = -1.E6_EB
PBY      = -1.E6_EB
PBZ      = -1.E6_EB
SURF_ID  = 'null'
COLOR    = 'null'
MESH_ID  = 'null'
MULT_ID  = 'null'
ID       = 'null'
RGB      =-1
TRANSPARENCY = 1._EB
DYNAMIC_PRESSURE = 0._EB
PRESSURE_RAMP = 'null'
XYZ      = -1.E6_EB
SPREAD_RATE = -1._EB
TMP_EXTERIOR = -1000.
TMP_EXTERIOR_RAMP = 'null'
TEXTURE_ORIGIN = -999._EB
OUTLINE       = .FALSE.
DEVC_ID   = 'null'
CTRL_ID   = 'null'
EVACUATION = .FALSE.
N_EDDY=0
L_EDDY=0._EB
L_EDDY_IJ=0._EB
VEL_RMS=0._EB
REYNOLDS_STRESS=0._EB
UVW = -1.E12_EB
RADIUS = -1._EB
WIND = .FALSE.

IF (NN==N_VENT_O-2 .AND. CYLINDRICAL .AND. XS<=TWO_EPSILON_EB) MB='XMIN'
```

```
IF (NN==N_VENT_O-1 .AND. TWO_D)                    MB='YMIN'
IF (NN==N_VENT_O   .AND. TWO_D)                    MB='YMAX'
IF (NN==N_VENT_O-1 .AND. EVACUATION_ONLY(NM))      MB='ZMIN'
IF (NN==N_VENT_O   .AND. EVACUATION_ONLY(NM))      MB='ZMAX'

IF (MB=='null') THEN
EVACUATION_VENT = .FALSE.
IF (EVACUATION_ONLY(NM)) CALL DEFINE_EVACUATION_VENTS(NM,2)
EVACUATION_VENTS: IF (.NOT. EVACUATION_VENT) THEN
CALL CHECKREAD('VENT',LU_INPUT,IOS)
IF (IOS==1) EXIT READ_VENT_LOOP
READ(LU_INPUT,VENT,END=37)      ! Read in info for VENT N
END IF EVACUATION_VENTS
ELSE
SURF_ID = 'MIRROR'
ENDIF

IF (MESH_ID/='null' .AND. MESH_ID/=MESH_NAME(NM))  CYCLE READ_VENT_LOOP

IF (PBX>-1.E5_EB .OR. PBY>-1.E5_EB .OR. PBZ>-1.E5_EB) THEN
IF (MULT_ID/='null') THEN
WRITE(MESSAGE,'(A,I0,A)') 'ERROR: MULT_ID cannot be applied to VENT',NN,' because it uses PBX, PBY or PBZ.'
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF
XB(1) = XS
XB(2) = XF
XB(3) = YS
XB(4) = YF
XB(5) = ZS
XB(6) = ZF
IF (PBX>-1.E5_EB) XB(1:2) = PBX
IF (PBY>-1.E5_EB) XB(3:4) = PBY
IF (PBZ>-1.E5_EB) XB(5:6) = PBZ
ENDIF

IF (MB/='null') THEN
IF (NMESHES>1 .AND. SURF_ID=='PERIODIC') THEN
WRITE(MESSAGE,'(A,I0,A)') 'ERROR: Use PBX,PBY,PBZ or XB for VENT',NN,' multi-mesh PERIODIC boundary'
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF
IF (MULT_ID/='null') THEN
WRITE(MESSAGE,'(A,I0,A)') 'ERROR: MULT_ID cannot be applied to VENT',NN,' because it uses MB.'
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF
XB(1) = XS
XB(2) = XF
XB(3) = YS
XB(4) = YF
XB(5) = ZS
XB(6) = ZF
SELECT CASE (MB)
CASE('XMIN')
XB(2) = XS
CASE('XMAX')
XB(1) = XF
CASE('YMIN')
XB(4) = YS
CASE('YMAX')
XB(3) = YF
CASE('ZMIN')
XB(6) = ZS
CASE('ZMAX')
XB(5) = ZF
CASE DEFAULT
WRITE(MESSAGE,'(A,I0,A)') 'ERROR: MB specified for VENT',NN,' is not XMIN, XMAX, YMIN, YMAX, ZMIN, or ZMAX'
CALL SHUTDOWN(MESSAGE) ; RETURN
END SELECT
ENDIF

! Check that the vent is properly specified

IF (ABS(XB(3)-XB(4))<=SPACING(XB(4))  .AND. TWO_D .AND. NN<N_VENT_O-1) THEN
IF (ID=='null')WRITE(MESSAGE,'(A,I0,A)')'ERROR: VENT ',NN,       ' cannot be specified on a y boundary in a 2D calculation'
IF (ID/='null')WRITE(MESSAGE,'(A,A,A)') 'ERROR: VENT ',TRIM(ID),' cannot be specified on a y boundary in a 2D calculation'
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF

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
IF (ID=='null') WRITE(MESSAGE,'(A,I0,A)')  'ERROR: VENT ',NN,       ' must be a plane'
IF (ID/='null') WRITE(MESSAGE,'(A,A,A)')  'ERROR: VENT ',TRIM(ID),' must be a plane'
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF

CALL CHECK_XB(XB)

```

```
IF (ALL(EVACUATION_ONLY)) THEN
DEVC_ID    = 'null'
CTRL_ID    = 'null'
END IF

! Loop over all possible multiples of the OBST

MR => MULTIPLIER(0)
DO NNN=1,N_MULT
IF (MULT_ID==MULTIPLIER(NNN)%ID) MR => MULTIPLIER(NNN)
ENDDO

K_MULT_LOOP: DO KK=MR%K_LOWER,MR%K_UPPER
J_MULT_LOOP: DO JJ=MR%J_LOWER,MR%J_UPPER
I_MULT_LOOP: DO II=MR%I_LOWER,MR%I_UPPER

REJECT_VENT = .FALSE.

IF (.NOT.MR%SEQUENTIAL) THEN
XB1 = XB(1) + MR%DX0 + II*MR%DXB(1)
XB2 = XB(2) + MR%DX0 + II*MR%DXB(2)
XB3 = XB(3) + MR%DY0 + JJ*MR%DXB(3)
XB4 = XB(4) + MR%DY0 + JJ*MR%DXB(4)
XB5 = XB(5) + MR%DZ0 + KK*MR%DXB(5)
XB6 = XB(6) + MR%DZ0 + KK*MR%DXB(6)
ELSE
XB1 = XB(1) + MR%DX0 + II*MR%DXB(1)
XB2 = XB(2) + MR%DX0 + II*MR%DXB(2)
XB3 = XB(3) + MR%DY0 + II*MR%DXB(3)
XB4 = XB(4) + MR%DY0 + II*MR%DXB(4)
XB5 = XB(5) + MR%DZ0 + II*MR%DXB(5)
XB6 = XB(6) + MR%DZ0 + II*MR%DXB(6)
ENDIF

! Increase the VENT counter

N = N + 1

VT=>VENTS(N)

IF (ABS(XB1-XB2)<=SPACING(XB2) ) VT%UNDIVIDED_INPUT_AREA = (XB4-XB3)*(XB6-XB5)
IF (ABS(XB3-XB4)<=SPACING(XB4) ) VT%UNDIVIDED_INPUT_AREA = (XB2-XB1)*(XB6-XB5)
IF (ABS(XB5-XB6)<=SPACING(XB6) ) VT%UNDIVIDED_INPUT_AREA = (XB2-XB1)*(XB4-XB3)
IF (RADIUS>0._EB)                VT%UNDIVIDED_INPUT_AREA = PI*RADIUS**2

VT%X1_ORIG = XB1
VT%X2_ORIG = XB2
VT%Y1_ORIG = XB3
VT%Y2_ORIG = XB4
VT%Z1_ORIG = XB5
VT%Z2_ORIG = XB6

XB1 = MAX(XB1,XS)
XB2 = MIN(XB2,XF)
XB3 = MAX(XB3,YS)
XB4 = MIN(XB4,YF)
XB5 = MAX(XB5,ZS)
XB6 = MIN(XB6,ZF)

IF ((XB1-XF)>SPACING(XF) .OR. (XS-XB2)>SPACING(XS) .OR. &
(XB3-YF)>SPACING(YF) .OR. (YS-XB4)>SPACING(YS) .OR. &
(XB5-ZF)>SPACING(ZF) .OR. (ZS-XB6)>SPACING(ZS)) REJECT_VENT = .TRUE.

VT%I1 = MAX(0,    NINT(GINV(XB1-XS,1,NM)*RDXI   ))
VT%I2 = MIN(IBAR,NINT(GINV(XB2-XS,1,NM)*RDXI   ))
VT%J1 = MAX(0,    NINT(GINV(XB3-YS,2,NM)*RDETA  ))
VT%J2 = MIN(JBAR,NINT(GINV(XB4-YS,2,NM)*RDETA  ))
VT%K1 = MAX(0,    NINT(GINV(XB5-ZS,3,NM)*RDZETA ))
VT%K2 = MIN(KBAR,NINT(GINV(XB6-ZS,3,NM)*RDZETA ))

! Thicken evacuation mesh vents in the z direction

IF (EVACUATION_ONLY(NM) .AND. EVACUATION .AND. VT%K1==VT%K2 .AND. .NOT.REJECT_VENT) THEN
VT%K1 = INT(GINV(.5_EB*(XB5+XB6)-ZS,3,NM)*RDZETA)
VT%K2 = KBAR
XB5 = ZS
XB6 = ZF
IF (ABS(XB1-XB2)>SPACING(XB2) .AND. ABS(XB3-XB4)>SPACING(XB4) ) THEN
IF (ID=='null') WRITE(MESSAGE,'(A,I0,A)') 'ERROR: Evacuation VENT ',NN,       ' must be a vertical plane'
IF (ID/='null') WRITE(MESSAGE,'(A,A,A)')  'ERROR: Evacuation VENT ',TRIM(ID),' must be a vertical plane'
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF
ENDIF

IF (ABS(XB1-XB2)<=SPACING(XB2) ) THEN
IF (VT%J1==VT%J2 .OR. VT%K1==VT%K2) REJECT_VENT=.TRUE.
IF (VT%I1>IBAR .OR. VT%I2<0)        REJECT_VENT=.TRUE.
ENDIF
```

```
IF (ABS(XB3-XB4)<=SPACING(XB4) ) THEN
IF (VT%I1==VT%I2 .OR. VT%K1==VT%K2) REJECT_VENT=.TRUE.
IF (VT%J1>JBAR .OR. VT%J2 <0)         REJECT_VENT=.TRUE.
ENDIF
IF (ABS(XB5-XB6)<=SPACING(XB6) ) THEN
IF (VT%I1==VT%I2 .OR. VT%J1==VT%J2) REJECT_VENT=.TRUE.
IF (VT%K1>KBAR .OR. VT%K2<0)          REJECT_VENT=.TRUE.
ENDIF

! Evacuation criteria

IF (.NOT.EVACUATION .AND. EVACUATION_ONLY(NM)) REJECT_VENT=.TRUE.
IF (EVACUATION .AND. .NOT.EVACUATION_ONLY(NM)) REJECT_VENT=.TRUE.

IF (ALL(EVACUATION_ONLY)) THEN
DEVC_ID    = 'null'
CTRL_ID    = 'null'
END IF

! If the VENT is to rejected

IF (REJECT_VENT) THEN
N = N-1
N_VENT = N_VENT-1
CYCLE I_MULT_LOOP
ENDIF

! Vent area

VT%X1 = XB1
VT%X2 = XB2
VT%Y1 = XB3
VT%Y2 = XB4
VT%Z1 = XB5
VT%Z2 = XB6

IF (ABS(XB1-XB2)<=SPACING(XB2) ) VT%INPUT_AREA = (XB4-XB3)*(XB6-XB5)
IF (ABS(XB3-XB4)<=SPACING(XB4) ) VT%INPUT_AREA = (XB2-XB1)*(XB6-XB5)
IF (ABS(XB5-XB6)<=SPACING(XB6) ) VT%INPUT_AREA = (XB2-XB1)*(XB4-XB3)

! Check the SURF_ID against the list of SURF's

CALL CHECK_SURF_NAME(SURF_ID,EX)
IF (.NOT.EX) THEN
WRITE(MESSAGE,'(A,A,A,I0,A,I0)') 'ERROR: SURF_ID ',TRIM(SURF_ID),' not found for VENT ',N_VENT,&
', line number ',INPUT_FILE_LINE_NUMBER
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF

! Assign SURF_INDEX, Index of the Boundary Condition

VT%SURF_INDEX = DEFAULT_SURF_INDEX
DO NNN=0,N_SURF
IF (SURF_ID==SURFACE(NNN)%ID) VT%SURF_INDEX = NNN
ENDDO

IF (SURF_ID=='OPEN')                          VT%TYPE_INDICATOR =  2
IF (SURF_ID=='MIRROR' .OR. SURF_ID=='PERIODIC') VT%TYPE_INDICATOR = -2
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
    =-2
IF (SURF_ID=='PERIODIC' .AND. WIND) VT%SURF_INDEX = PERIODIC_WIND_SURF_INDEX

VT%BOUNDARY_TYPE = SOLID_BOUNDARY
IF (VT%SURF_INDEX==OPEN_SURF_INDEX)          VT%BOUNDARY_TYPE = OPEN_BOUNDARY
IF (VT%SURF_INDEX==MIRROR_SURF_INDEX)        VT%BOUNDARY_TYPE = MIRROR_BOUNDARY
IF (VT%SURF_INDEX==PERIODIC_SURF_INDEX)      VT%BOUNDARY_TYPE = PERIODIC_BOUNDARY
IF (VT%SURF_INDEX==PERIODIC_WIND_SURF_INDEX) VT%BOUNDARY_TYPE = PERIODIC_BOUNDARY
IF (VT%SURF_INDEX==HVAC_SURF_INDEX)          VT%BOUNDARY_TYPE = HVAC_BOUNDARY

VT%IOR = IOR
VT%ORDINAL = NN

! Activate and Deactivate logic

VT%ACTIVATED = .TRUE.
VT%DEVC_ID   = DEVC_ID
VT%CTRL_ID   = CTRL_ID
VT%ID        = ID
CALL SEARCH_CONTROLLER('VENT',CTRL_ID,DEVC_ID,VT%DEVC_INDEX,VT%CTRL_INDEX,N)
IF (DEVC_ID /= 'null') THEN
IF (.NOT.DEVICE(VT%DEVC_INDEX)%INITIAL_STATE) VT%ACTIVATED = .FALSE.
ENDIF
IF (CTRL_ID /= 'null') THEN
IF (.NOT.CONTROL(VT%CTRL_INDEX)%INITIAL_STATE) VT%ACTIVATED = .FALSE.
ENDIF

IF ( (VT%BOUNDARY_TYPE==OPEN_BOUNDARY .OR. VT%BOUNDARY_TYPE==MIRROR_BOUNDARY .OR. &
VT%BOUNDARY_TYPE==PERIODIC_BOUNDARY) .AND. &
```

```
(VT%DEVC_ID /= 'null' .OR. VT%CTRL_ID /= 'null') ) THEN
IF (ID=='null') WRITE(MESSAGE,'(A,I0,A,I0)') 'ERROR: VENT ',NN, &
' cannot be controlled by a device, line number ',INPUT_FILE_LINE_NUMBER
IF (ID/='null') WRITE(MESSAGE,'(A,A,A)') 'ERROR: VENT ',TRIM(ID),' cannot be controlled by a device'
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF

! Set the VENT color index

SELECT CASE(COLOR)
CASE('INVISIBLE')
VT%COLOR_INDICATOR = 8
TRANSPARENCY = 0._EB
CASE('null')
VT%COLOR_INDICATOR = 99
CASE DEFAULT
VT%COLOR_INDICATOR = 99
CALL COLOR2RGB(RGB,COLOR)
END SELECT
IF (VT%COLOR_INDICATOR==8) VT%TYPE_INDICATOR = -2
IF (OUTLINE)               VT%TYPE_INDICATOR =  2
VT%RGB = RGB
VT%TRANSPARENCY = TRANSPARENCY

! Parameters for specified spread of a fire over a VENT

VT%X0 = XYZ(1)
VT%Y0 = XYZ(2)
VT%Z0 = XYZ(3)
VT%FIRE_SPREAD_RATE = SPREAD_RATE / TIME_SHRINK_FACTOR

! Circular VENT

IF (RADIUS>0._EB) THEN
IF (ANY(XYZ<-1.E5_EB)) THEN
WRITE(MESSAGE,'(A,I0,A)') 'ERROR: VENT ',NN,' requires center point XYZ'
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF
VT%RADIUS = RADIUS
ENDIF

! Dynamic Pressure

VT%DYNAMIC_PRESSURE = DYNAMIC_PRESSURE
IF (PRESSURE_RAMP/='null') CALL GET_RAMP_INDEX(PRESSURE_RAMP,'TIME',VT%PRESSURE_RAMP_INDEX)

! Synthetic Eddy Method

VT%N_EDDY = N_EDDY
IF (L_EDDY>TWO_EPSILON_EB) THEN
VT%SIGMA_IJ = L_EDDY
ELSE
VT%SIGMA_IJ = L_EDDY_IJ ! Modified SEM (Jarrin, Ch. 7)
VT%SIGMA_IJ = MAX(VT%SIGMA_IJ,1.E-10_EB)
ENDIF
IF (VEL_RMS>0._EB) THEN
VT%R_IJ=0._EB
VT%R_IJ(1,1)=VEL_RMS**2
VT%R_IJ(2,2)=VEL_RMS**2
VT%R_IJ(3,3)=VEL_RMS**2
ELSE
VT%R_IJ = REYNOLDS_STRESS
VT%R_IJ = MAX(VT%R_IJ,1.E-10_EB)
ENDIF

! Check SEM parameters

IF (N_EDDY>0) THEN
SYNTHETIC_EDDY_METHOD = .TRUE.
IF (ANY(VT%SIGMA_IJ<TWO_EPSILON_EB)) THEN
WRITE(MESSAGE,'(A,I0,A)') 'ERROR: VENT ',NN,' L_EDDY = 0 in Synthetic Eddy Method'
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF
IF (ALL(ABS(VT%R_IJ)<TWO_EPSILON_EB)) THEN
WRITE(MESSAGE,'(A,I0,A)') 'ERROR: VENT ',NN,' VEL_RMS (or Reynolds Stress) = 0 in Synthetic Eddy Method'
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF
IF (TRIM(SURF_ID)=='HVAC') THEN
WRITE(MESSAGE,'(A,I0,A)') 'ERROR: VENT ',NN,' Synthetic Eddy Method not permitted with HVAC'
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF
ENDIF

! Miscellaneous

VT%TMP_EXTERIOR = TMP_EXTERIOR + TMPM
IF (VT%TMP_EXTERIOR>0._EB) TMPMIN = MIN(TMPMIN,VT%TMP_EXTERIOR)
IF (TMP_EXTERIOR_RAMP/='null') CALL GET_RAMP_INDEX(TMP_EXTERIOR_RAMP,'TIME',VT%TMP_EXTERIOR_RAMP_INDEX)
```

```
VT%TEXTURE(:) = TEXTURE_ORIGIN(:)

VT%UVW = UVW
IF (ALL(VT%UVW > -1.E12_EB)) THEN
VT%UVW = VT%UVW/SQRT(VT%UVW(1)**2+VT%UVW(2)**2+VT%UVW(3)**2)
ENDIF

ENDDO I_MULT_LOOP
ENDDO J_MULT_LOOP
ENDDO K_MULT_LOOP

ENDDO READ_VENT_LOOP
37 REWIND(LU_INPUT) ; INPUT_FILE_LINE_NUMBER = 0

ENDDO MESH_LOOP_1

! Go through all the meshes again, but this time only if PROCESS(NM)==MYID

MESH_LOOP_2: DO NM=1,NMESHES

IF (PROCESS(NM)/=MYID) CYCLE MESH_LOOP_2

M=>MESHES(NM)
CALL POINT_TO_MESH(NM)

! Get total number of vents (needed for detailed wind BC)

N_VENT_TOTAL = N_VENT_TOTAL + N_VENT

! Check vents and assign orientations

VENT_LOOP_2: DO N=1,N_VENT

VT => VENTS(N)

I1 = MAX(0,VT%I1)
I2 = MIN(IBAR,VT%I2)
J1 = MAX(0,VT%J1)
J2 = MIN(JBAR,VT%J2)
K1 = MAX(0,VT%K1)
K2 = MIN(KBAR,VT%K2)

IF (VT%IOR==0) THEN
IF (I1==    0 .AND. I2==0) VT%IOR =  1
IF (I1==IBAR .AND. I2==IBAR) VT%IOR = -1
IF (J1==    0 .AND. J2==0) VT%IOR =  2
IF (J1==JBAR .AND. J2==JBAR) VT%IOR = -2
IF (K1==    0 .AND. K2==0) VT%IOR =  3
IF (K1==KBAR .AND. K2==KBAR) VT%IOR = -3
ENDIF

ORIENTATION_IF: IF (VT%IOR==0) THEN
IF (I1==I2) THEN
DO K=K1+1,K2
DO J=J1+1,J2
IF (.NOT.SOLID(CELL_INDEX(I2+1,J,K))) VT%IOR =  1
IF (.NOT.SOLID(CELL_INDEX(I2  ,J,K))) VT%IOR = -1
ENDDO
ENDDO
ENDIF
IF (J1==J2) THEN
DO K=K1+1,K2
DO I=I1+1,I2
IF (.NOT.SOLID(CELL_INDEX(I,J2+1,K))) VT%IOR =  2
IF (.NOT.SOLID(CELL_INDEX(I,J2  ,K))) VT%IOR = -2
ENDDO
ENDDO
ENDIF
IF (K1==K2) THEN
DO J=J1+1,J2
DO I=I1+1,I2
IF (.NOT.SOLID(CELL_INDEX(I,J,K2+1))) VT%IOR =  3
IF (.NOT.SOLID(CELL_INDEX(I,J,K2  ))) VT%IOR = -3
ENDDO
ENDDO
ENDIF
ENDIF ORIENTATION_IF

IF (VT%IOR==0) THEN
WRITE(MESSAGE,'(A,I0,A,I0)') 'ERROR: Specify orientation of VENT ',VT%ORDINAL, ', MESH NUMBER',NM
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF

! Other error messages for VENTs

SELECT CASE(ABS(VT%IOR))
CASE(1)
```

```
IF (I1>=1 .AND. I1<=IBM1) THEN
IF (VT%BOUNDARY_TYPE==OPEN_BOUNDARY.OR.VT%BOUNDARY_TYPE==MIRROR_BOUNDARY.OR.VT%BOUNDARY_TYPE==PERIODIC_BOUNDARY)
    THEN
WRITE(MESSAGE,'(A,I0,A)') 'ERROR: OPEN, MIRROR, OR PERIODIC VENT ',VT%ORDINAL, ' must be an exterior boundary.'
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF
IF (VT%BOUNDARY_TYPE/=HVAC_BOUNDARY) VT%BOUNDARY_TYPE = SOLID_BOUNDARY
IF (.NOT.SOLID(CELL_INDEX(I2+1,J2,K2)) .AND. .NOT.SOLID(CELL_INDEX(I2,J2,K2))) THEN
WRITE(MESSAGE,'(A,I0,A)') 'ERROR: VENT ',VT%ORDINAL, ' must be attached to a solid obstruction'
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF
ENDIF
CASE(2)
IF (J1>=1 .AND. J1<=JBM1) THEN
IF (VT%BOUNDARY_TYPE==OPEN_BOUNDARY.OR.VT%BOUNDARY_TYPE==MIRROR_BOUNDARY.OR.VT%BOUNDARY_TYPE==PERIODIC_BOUNDARY)
    THEN
WRITE(MESSAGE,'(A,I0,A)') 'ERROR: OPEN, MIRROR, OR PERIODIC VENT ',VT%ORDINAL, ' must be an exterior boundary.'
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF
IF (VT%BOUNDARY_TYPE/=HVAC_BOUNDARY) VT%BOUNDARY_TYPE = SOLID_BOUNDARY
IF (.NOT.SOLID(CELL_INDEX(I2,J2+1,K2)) .AND. .NOT.SOLID(CELL_INDEX(I2,J2,K2))) THEN
WRITE(MESSAGE,'(A,I0,A)') 'ERROR: VENT ',VT%ORDINAL, ' must be attached to a solid obstruction'
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF
ENDIF
CASE(3)
IF (K1>=1 .AND. K1<=KBM1) THEN
IF (VT%BOUNDARY_TYPE==OPEN_BOUNDARY.OR.VT%BOUNDARY_TYPE==MIRROR_BOUNDARY.OR.VT%BOUNDARY_TYPE==PERIODIC_BOUNDARY)
    THEN
WRITE(MESSAGE,'(A,I0,A)') 'ERROR: OPEN, MIRROR, OR PERIODIC VENT ',VT%ORDINAL, ' must be an exterior boundary.'
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF
IF (VT%BOUNDARY_TYPE/=HVAC_BOUNDARY) VT%BOUNDARY_TYPE = SOLID_BOUNDARY
IF (.NOT.SOLID(CELL_INDEX(I2,J2,K2+1)) .AND. .NOT.SOLID(CELL_INDEX(I2,J2,K2))) THEN
WRITE(MESSAGE,'(A,I0,A)') 'ERROR: VENT ',VT%ORDINAL, ' must be attached to a solid obstruction'
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF
ENDIF
END SELECT

! Open up boundary cells if it is an open vent

IF (VT%BOUNDARY_TYPE==OPEN_BOUNDARY) THEN
SELECT CASE(VT%IOR)
CASE( 1)
CALL BLOCK_CELL(NM,    0,    0,J1+1,   J2,K1+1,   K2,0,0)
CASE(-1)
CALL BLOCK_CELL(NM,IBP1,IBP1,J1+1,   J2,K1+1,   K2,0,0)
CASE( 2)
CALL BLOCK_CELL(NM,I1+1,   I2,    0,    0,K1+1,   K2,0,0)
CASE(-2)
CALL BLOCK_CELL(NM,I1+1,   I2,JBP1,JBP1,K1+1,   K2,0,0)
CASE( 3)
CALL BLOCK_CELL(NM,I1+1,   I2,J1+1,   J2,    0,    0,0,0)
CASE(-3)
CALL BLOCK_CELL(NM,I1+1,   I2,J1+1,   J2,KBP1,KBP1,0,0)
END SELECT
ENDIF

! Check UVW
IF (ABS(VT%UVW(ABS(VT%IOR))) < TWO_EPSILON_EB) THEN
WRITE(MESSAGE,'(A,I0,A)') 'ERROR: VENT ',VT%ORDINAL, ' cannot have normal component of UVW equal to 0'
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF

ENDDO VENT_LOOP_2

! Compute vent areas and check for passive openings

VENT_LOOP_3: DO N=1,N_VENT

VT => VENTS(N)

IF (VT%SURF_INDEX==HVAC_SURF_INDEX .AND. N>1) THEN
DO NN=1,N-1
IF (TRIM(VT%ID)==TRIM(VENTS(NN)%ID) .AND. VENTS(NN)%SURF_INDEX==HVAC_SURF_INDEX) THEN
WRITE(MESSAGE,'(A,A)') 'ERROR: Two HVAC VENTS have the same ID. VENT ID: ',TRIM(VT%ID)
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF
ENDDO
ENDIF

VT%FDS_AREA = 0._EB
IF (VT%RADIUS>0._EB) VT%INPUT_AREA = 0._EB

I1 = VT%I1
I2 = VT%I2
J1 = VT%J1
```

```
J2 = VT%J2
K1 = VT%K1
K2 = VT%K2

VT%GHOST_CELLS_ONLY = .TRUE.

SELECT CASE(ABS(VT%IOR))
CASE(1)
DO K=K1+1,K2
DO J=J1+1,J2
IF (J>=1 .AND. J<=JBAR .AND. K>=1 .AND. K<=KBAR) VT%GHOST_CELLS_ONLY = .FALSE.
IF ( VT%RADIUS>0._EB) THEN
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))
IF (((YC(J)-VT%Y0)**2+(ZC(K)-VT%Z0)**2)>VT%RADIUS**2) CYCLE
ENDIF
VT%FDS_AREA = VT%FDS_AREA + DY(J)*DZ(K)
ENDDO
ENDDO
CASE(2)
DO K=K1+1,K2
DO I=I1+1,I2
IF (I>=1 .AND. I<=IBAR .AND. K>=1 .AND. K<=KBAR) VT%GHOST_CELLS_ONLY = .FALSE.
IF ( VT%RADIUS>0._EB) THEN
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))
IF (((XC(I)-VT%X0)**2+(ZC(K)-VT%Z0)**2)>VT%RADIUS**2) CYCLE
ENDIF
VT%FDS_AREA = VT%FDS_AREA + DX(I)*DZ(K)
ENDDO
ENDDO
CASE(3)
DO J=J1+1,J2
DO I=I1+1,I2
IF (I>=1 .AND. I<=IBAR .AND. J>=1 .AND. J<=JBAR) VT%GHOST_CELLS_ONLY = .FALSE.
IF ( VT%RADIUS>0._EB) THEN
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))
IF (((XC(I)-VT%X0)**2+(YC(J)-VT%Y0)**2)>VT%RADIUS**2) CYCLE
ENDIF
VT%FDS_AREA = VT%FDS_AREA + DX(I)*DY(J)
ENDDO
ENDDO
END SELECT

ENDDO  VENT_LOOP_3

ENDDO MESH_LOOP_2

CONTAINS

SUBROUTINE DEFINE_EVACUATION_VENTS(NM,IMODE)
!
! Define the evacuation outflow VENTs for the doors/exits.
!
USE EVAC, ONLY: N_DOORS, N_EXITS, N_CO_EXITS, EVAC_EMESH_EXITS_TYPE, EMESH_EXITS
IMPLICIT NONE
! Passed variables
INTEGER, INTENT(IN) :: NM, IMODE
! Local variables
INTEGER :: N, N_END

N_END = N_EXITS - N_CO_EXITS + N_DOORS
IMODE_1_IF: IF (IMODE==1) THEN
NEND_LOOP_1: DO N = 1, N_END
IF (.NOT.EMESH_EXITS(N)%DEFINE_MESH) CYCLE NEND_LOOP_1
IF (EMESH_EXITS(N)%IMESH==NM .OR. EMESH_EXITS(N)%MAINMESH==NM) THEN
N_VENT = N_VENT + 1
EMESH_EXITS(N)%I_VENT = N_VENT
EVACUATION_VENT = .TRUE.
EVACUATION = .TRUE.
END IF
END DO NEND_LOOP_1
END IF IMODE_1_IF

IMODE_2_IF: IF (IMODE==2) THEN
! Evacuation VENTs (for the outflow vents) need: XB, EVACUATION, RGB, MESH_ID, SURF_ID, IOR
NEND_LOOP_2: DO N = 1, N_END
IF (.NOT.EMESH_EXITS(N)%DEFINE_MESH) CYCLE NEND_LOOP_2
IF (EMESH_EXITS(N)%I_VENT==NN .AND. (EMESH_EXITS(N)%IMESH==NM .OR. EMESH_EXITS(N)%MAINMESH==NM)) THEN
EVACUATION_VENT = .TRUE.
EVACUATION = .TRUE.
SURF_ID = 'EVACUATION_OUTFLOW'
MESH_ID = TRIM(MESH_NAME(NM))
XB(1) = EMESH_EXITS(N)%XB(1)
XB(2) = EMESH_EXITS(N)%XB(2)
XB(3) = EMESH_EXITS(N)%XB(3)
XB(4) = EMESH_EXITS(N)%XB(4)
XB(5) = EMESH_EXITS(N)%XB(5)
XB(6) = EMESH_EXITS(N)%XB(6)
RGB(:) = EMESH_EXITS(N)%RGB(:)
```

```
ID = TRIM('Event_' // TRIM(MESH_NAME(NM)))
END IF
END DO NEND_LOOP_2
END IF IMODE_2_IF

RETURN
END SUBROUTINE DEFINE_EVACUATION_VENTS

END SUBROUTINE READ_VENT


SUBROUTINE READ_INIT

USE PHYSICAL_FUNCTIONS, ONLY: GET_SPECIFIC_GAS_CONSTANT
USE COMP_FUNCTIONS, ONLY: GET_FILE_NUMBER
USE DEVICE_VARIABLES, ONLY: DEVICE_TYPE, DEVICE, N_DEVC
REAL(EB) :: DIAMETER,TEMPERATURE,DENSITY,RR_SUM,ZZ_GET(1:N_TRACKED_SPECIES),MASS_PER_VOLUME, &
MASS_PER_TIME,DT_INSERT,UVW(3),HRRPUV,XYZ(3),DX,DY,DZ,HEIGHT,RADIUS,MASS_FRACTION(MAX_SPECIES), &
PARTICLE_WEIGHT_FACTOR,AUTO_IGNITION_TEMPERATURE,VOLUME_FRACTION(MAX_SPECIES)
INTEGER :: NM,N,NN,NNN,II,JJ,KK,NS,NS2,NUMBER_INITIAL_PARTICLES,N_PARTICLES,N_INIT_NEW,N_INIT_READ,
       N_PARTICLES_PER_CELL
LOGICAL :: CELL_CENTERED
EQUIVALENCE(NUMBER_INITIAL_PARTICLES,N_PARTICLES)
CHARACTER(LABEL_LENGTH) :: ID,CTRL_ID,DEVC_ID,PART_ID,SHAPE,MULT_ID,SPEC_ID(1:MAX_SPECIES)
TYPE(INITIALIZATION_TYPE), POINTER :: IN=>NULL()
TYPE(MULTIPLIER_TYPE), POINTER :: MR=>NULL()
TYPE(LAGRANGIAN_PARTICLE_CLASS_TYPE), POINTER :: LPC=>NULL()
TYPE(DEVICE_TYPE), POINTER :: DV
NAMELIST /INIT/ AUTO_IGNITION_TEMPERATURE,CELL_CENTERED,CTRL_ID,DENSITY,DEVC_ID,DIAMETER,DT_INSERT,DX,DY,DZ,&
HEIGHT,HRRPUV,ID,MASS_FRACTION,&
MASS_PER_TIME,MASS_PER_VOLUME,MULT_ID,N_PARTICLES,N_PARTICLES_PER_CELL,PART_ID,PARTICLE_WEIGHT_FACTOR,&
RADIUS,SHAPE,SPEC_ID,TEMPERATURE,UVW,VOLUME_FRACTION,XB,XYZ,&
NUMBER_INITIAL_PARTICLES !Backwards compatability

N_INIT = 0
N_INIT_READ = 0
REWIND(LU_INPUT) ; INPUT_FILE_LINE_NUMBER = 0

COUNT_LOOP: DO
CALL CHECKREAD('INIT',LU_INPUT,IOS)
IF (IOS==1) EXIT COUNT_LOOP
MULT_ID = 'null'
READ(LU_INPUT,NML=INIT,END=11,ERR=12,IOSTAT=IOS)
N_INIT_READ = N_INIT_READ + 1
12 IF (IOS>0) THEN
WRITE(MESSAGE,'(A,I0,A,I0)') 'ERROR: Problem with INIT number ',N_INIT_READ+1,', line number',
       INPUT_FILE_LINE_NUMBER
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF
N_INIT_NEW = 0
IF (MULT_ID=='null') THEN
N_INIT_NEW = 1
ELSE
DO N=1,N_MULT
MR => MULTIPLIER(N)
IF (MULT_ID==MR%ID) N_INIT_NEW = MR%N_COPIES
ENDDO
IF (N_INIT_NEW==0) THEN
WRITE(MESSAGE,'(A,A,A,I0)') 'ERROR: MULT line ',TRIM(MULT_ID),' not found on INIT line', N_INIT_READ
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF
ENDIF
N_INIT = N_INIT + N_INIT_NEW
ENDDO COUNT_LOOP
11 REWIND(LU_INPUT) ; INPUT_FILE_LINE_NUMBER = 0

! Add reserved INIT lines

N_INIT = N_INIT + N_INIT_RESERVED

! If there are no INIT lines, return

IF (N_INIT==0) RETURN

ALLOCATE(INITIALIZATION(N_INIT),STAT=IZERO)
CALL ChkMemErr('READ','INITIALIZATION',IZERO)

DO NN=1,N_INIT
ALLOCATE(INITIALIZATION(NN)%MASS_FRACTION(N_TRACKED_SPECIES),STAT=IZERO)
CALL ChkMemErr('READ','INITIALIZATION',IZERO)
INITIALIZATION(NN)%MASS_FRACTION=0._EB
ENDDO

NN = 0

INIT_LOOP: DO N=1,N_INIT_READ+N_INIT_RESERVED

IF (N<=N_INIT_READ) THEN
```

```fortran

! Read in the INIT lines

CALL CHECKREAD('INIT',LU_INPUT,IOS)
IF (IOS==1) EXIT INIT_LOOP
CALL SET_INIT_DEFAULTS
READ(LU_INPUT,INIT)
IF(ANY(MASS_FRACTION>0._EB) .AND. ANY(VOLUME_FRACTION>0._EB)) THEN
WRITE(MESSAGE,'(A,I0,A)') 'ERROR: INIT line ', N, ". Do not specify both MASS_FRACTION and VOLUME_FRACTION."
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF

ELSE

! Use information from DEVC line to create an INIT line for 'RADIATIVE HEAT FLUX GAS' or 'ADIABATIC SURFACE
    TEMPERATURE GAS'

CALL SET_INIT_DEFAULTS
DV => DEVICE(INIT_RESERVED(N-N_INIT_READ)%DEVC_INDEX)
DX = INIT_RESERVED(N-N_INIT_READ)%DX
DY = INIT_RESERVED(N-N_INIT_READ)%DY
DZ = INIT_RESERVED(N-N_INIT_READ)%DZ
WRITE(PART_ID,'(A)') 'RESERVED TARGET PARTICLE'
N_PARTICLES = INIT_RESERVED(N-N_INIT_READ)%N_PARTICLES
XB(1) = DV%X
XB(2) = DV%X + (N_PARTICLES-1)*DX
XB(3) = DV%Y
XB(4) = DV%Y + (N_PARTICLES-1)*DY
XB(5) = DV%Z
XB(6) = DV%Z + (N_PARTICLES-1)*DZ
ID = DV%ID
ENDIF

! Transform XYZ into XB if necessary, and move XYZ points off of mesh boundaries.

IF (ANY(XYZ>-100000._EB)) THEN

MESH_LOOP: DO NM=1,NMESHES
IF (EVACUATION_ONLY(NM)) CYCLE MESH_LOOP
M=>MESHES(NM)
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)
    THEN
IF (ABS(XYZ(1)-M%XS)<TWO_EPSILON_EB) XYZ(1) = XYZ(1) + 0.01_EB*M%DXI
IF (ABS(XYZ(1)-M%XF)<TWO_EPSILON_EB) XYZ(1) = XYZ(1) - 0.01_EB*M%DXI
IF (ABS(XYZ(2)-M%YS)<TWO_EPSILON_EB) XYZ(2) = XYZ(2) + 0.01_EB*M%DETA
IF (ABS(XYZ(2)-M%YF)<TWO_EPSILON_EB) XYZ(2) = XYZ(2) - 0.01_EB*M%DETA
IF (ABS(XYZ(3)-M%ZS)<TWO_EPSILON_EB) XYZ(3) = XYZ(3) + 0.01_EB*M%DZETA
IF (ABS(XYZ(3)-M%ZF)<TWO_EPSILON_EB) XYZ(3) = XYZ(3) - 0.01_EB*M%DZETA
EXIT MESH_LOOP
ENDIF
ENDDO MESH_LOOP

XB(1:2) = XYZ(1)
XB(3:4) = XYZ(2)
XB(5:6) = XYZ(3)
ENDIF

! If an offset has been specified, set the SHAPE to LINE.

IF (DX>0._EB .OR. DY>0._EB .OR. DZ>0._EB) SHAPE = 'LINE'

IF (N_PARTICLES>0 .AND. SHAPE=='LINE') THEN
XB(2) = XB(1) + DX*(N_PARTICLES-1)
XB(4) = XB(3) + DY*(N_PARTICLES-1)
XB(6) = XB(5) + DZ*(N_PARTICLES-1)
ENDIF

! Create a box around a CONE

IF (SHAPE=='CONE' .OR. SHAPE=='RING') THEN
XB(1) = XYZ(1) - RADIUS
XB(2) = XYZ(1) + RADIUS
XB(3) = XYZ(2) - RADIUS
XB(4) = XYZ(2) + RADIUS
XB(5) = XYZ(3)
XB(6) = XYZ(3) + HEIGHT
IF (SHAPE=='RING') XB(6) = XB(5)
ENDIF

! Reorder XB coordinates if necessary

CALL CHECK_XB(XB)

! Loop over all possible multiples of the INIT

MR => MULTIPLIER(0)
DO NNN=1,N_MULT
IF (MULT_ID==MULTIPLIER(NNN)%ID) MR => MULTIPLIER(NNN)
```

```
ENDDO

NNN = 0
K_MULT_LOOP: DO KK=MR%K_LOWER,MR%K_UPPER
J_MULT_LOOP: DO JJ=MR%J_LOWER,MR%J_UPPER
I_MULT_LOOP: DO II=MR%I_LOWER,MR%I_UPPER

NNN = NNN + 1  ! Counter for MULT INIT lines

NN = NN + 1
IN => INITIALIZATION(NN)

! Store the input parameters

IF (.NOT.MR%SEQUENTIAL) THEN
IN%X1 = XB(1) + MR%DX0 + II*MR%DXB(1)
IN%X2 = XB(2) + MR%DX0 + II*MR%DXB(2)
IN%Y1 = XB(3) + MR%DY0 + JJ*MR%DXB(3)
IN%Y2 = XB(4) + MR%DY0 + JJ*MR%DXB(4)
IN%Z1 = XB(5) + MR%DZ0 + KK*MR%DXB(5)
IN%Z2 = XB(6) + MR%DZ0 + KK*MR%DXB(6)
ELSE
IN%X1 = XB(1) + MR%DX0 + II*MR%DXB(1)
IN%X2 = XB(2) + MR%DX0 + II*MR%DXB(2)
IN%Y1 = XB(3) + MR%DY0 + II*MR%DXB(3)
IN%Y2 = XB(4) + MR%DY0 + II*MR%DXB(4)
IN%Z1 = XB(5) + MR%DZ0 + II*MR%DXB(5)
IN%Z2 = XB(6) + MR%DZ0 + II*MR%DXB(6)
ENDIF

IF (MR%N_COPIES>1) THEN
WRITE(IN%ID,'(A,A,I5.5)') TRIM(ID),'-',NNN
ELSE
IN%ID = ID
ENDIF

IN%CELL_CENTERED  = CELL_CENTERED
IN%DIAMETER       = DIAMETER*1.E-6_EB
IN%DX             = DX
IN%DY             = DY
IN%DZ             = DZ
IN%CTRL_ID        = CTRL_ID
IN%DEVC_ID        = DEVC_ID
CALL SEARCH_CONTROLLER('INIT',IN%CTRL_ID,IN%DEVC_ID,IN%DEVC_INDEX,IN%CTRL_INDEX,N)
IN%VOLUME         = (IN%X2-IN%X1)*(IN%Y2-IN%Y1)*(IN%Z2-IN%Z1)
IN%TEMPERATURE    = TEMPERATURE + TMPM
IN%DENSITY        = DENSITY
IN%SHAPE          = SHAPE
IN%HEIGHT         = HEIGHT
IN%RADIUS         = RADIUS
IN%HRRPUV         = HRRPUV*1000._EB
IN%AIT            = AUTO_IGNITION_TEMPERATURE + TMPM
IF (HRRPUV > TWO_EPSILON_EB) INIT_HRRPUV = .TRUE.
IF (DENSITY > 0._EB) RHOMAX = MAX(RHOMAX,IN%DENSITY)
IF (AUTO_IGNITION_TEMPERATURE < 1.E20_EB) REIGNITION_MODEL = .TRUE.

SPEC_INIT_IF:IF (ANY(MASS_FRACTION > 0._EB)) THEN
IF (SPEC_ID(1)=='null') THEN
WRITE(MESSAGE,'(A,I0,A,A)') 'ERROR: Problem with INIT number ',N,'. SPEC_ID must be used with MASS_FRACTION'
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF
DO NS=1,MAX_SPECIES
IF (SPEC_ID(NS)=='null') EXIT
DO NS2=1,N_TRACKED_SPECIES
IF (NS2>0 .AND. TRIM(SPEC_ID(NS))==TRIM(SPECIES_MIXTURE(NS2)%ID)) THEN
IN%MASS_FRACTION(NS2)=MASS_FRACTION(NS)
EXIT
ENDIF
IF (NS2==N_TRACKED_SPECIES)  THEN
WRITE(MESSAGE,'(A,I0,A,A,A)') 'ERROR: Problem with INIT number ',N,' tracked species ',&
TRIM(SPEC_ID(NS)),' not found'
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF
ENDDO
ENDDO

IF (SUM(IN%MASS_FRACTION) > 1._EB) THEN
WRITE(MESSAGE,'(A,I0,A,A)') 'ERROR: Problem with INIT number ',N,'. Sum of specified mass fractions > 1'
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF
IF (IN%MASS_FRACTION(1) <=TWO_EPSILON_EB) THEN
IN%MASS_FRACTION(1) = 1._EB-SUM(IN%MASS_FRACTION(2:N_TRACKED_SPECIES))
ELSE
WRITE(MESSAGE,'(A,I0,A,A)') 'ERROR: Problem with INIT number ',N,&
'. Cannot specify background species for MASS_FRACTION'
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF
ZZ_GET(1:N_TRACKED_SPECIES) = IN%MASS_FRACTION(1:N_TRACKED_SPECIES)
```

```
CALL GET_SPECIFIC_GAS_CONSTANT(ZZ_GET,RR_SUM)

ELSEIF (ANY(VOLUME_FRACTION>0._EB)) THEN SPEC_INIT_IF
IF (SUM(VOLUME_FRACTION) > 1._EB) THEN
WRITE(MESSAGE,'(A,I0,A,A)') 'ERROR: Problem with INIT number ',N,'. Sum of specified volume fractions > 1'
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF
IF (SPEC_ID(1)=='null') THEN
WRITE(MESSAGE,'(A,I0,A,A)') 'ERROR: Problem with INIT number ',N,'. SPEC_ID must be used with VOLUME_FRACTION'
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF
DO NS=1,MAX_SPECIES
IF (SPEC_ID(NS)=='null') EXIT
DO NS2=1,N_TRACKED_SPECIES
IF (NS2>0 .AND. TRIM(SPEC_ID(NS))==TRIM(SPECIES_MIXTURE(NS2)%ID)) THEN
MASS_FRACTION(NS2)=VOLUME_FRACTION(NS)*SPECIES_MIXTURE(NS2)%MW
EXIT
ENDIF
IF (NS2==N_TRACKED_SPECIES)  THEN
WRITE(MESSAGE,'(A,I0,A,A,A)') 'ERROR: Problem with INIT number ',N,' tracked species ',&
TRIM(SPEC_ID(NS)),' not found'
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF
ENDDO
ENDDO
IF (MASS_FRACTION(1) <=TWO_EPSILON_EB) THEN
MASS_FRACTION(1) = (1._EB-SUM(VOLUME_FRACTION))*SPECIES_MIXTURE(1)%MW
ELSE
WRITE(MESSAGE,'(A,I0,A,A)') 'ERROR: Problem with INIT number ',N,&
'. Cannot specify background species for VOLUME_FRACTION'
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF
MASS_FRACTION(1:N_TRACKED_SPECIES) = MASS_FRACTION(1:N_TRACKED_SPECIES)/SUM(MASS_FRACTION(1:N_TRACKED_SPECIES))
IN%MASS_FRACTION(1:N_TRACKED_SPECIES) = MASS_FRACTION(1:N_TRACKED_SPECIES)
ZZ_GET(1:N_TRACKED_SPECIES) = IN%MASS_FRACTION(1:N_TRACKED_SPECIES)
CALL GET_SPECIFIC_GAS_CONSTANT(ZZ_GET,RR_SUM)

ELSE SPEC_INIT_IF
IN%MASS_FRACTION(1:N_TRACKED_SPECIES) = SPECIES_MIXTURE(1:N_TRACKED_SPECIES)%ZZ0
RR_SUM = RSUM0

ENDIF SPEC_INIT_IF

IF (TEMPERATURE > 0._EB) TMPMIN = MIN(TMPMIN,IN%TEMPERATURE)

IF (IN%TEMPERATURE > 0._EB .AND. IN%DENSITY < 0._EB) THEN
IN%DENSITY        = P_INF/(IN%TEMPERATURE*RR_SUM)
IN%ADJUST_DENSITY = .TRUE.
ENDIF
IF (IN%TEMPERATURE < 0._EB .AND. IN%DENSITY > 0._EB) THEN
IN%TEMPERATURE        = P_INF/(IN%DENSITY*RR_SUM)
IN%ADJUST_TEMPERATURE = .TRUE.
ENDIF
IF (IN%TEMPERATURE < 0._EB .AND. IN%DENSITY < 0._EB) THEN
IN%TEMPERATURE    = TMPA
IN%DENSITY        = P_INF/(IN%TEMPERATURE*RR_SUM)
IN%ADJUST_DENSITY = .TRUE.
ENDIF

! Special case where INIT is used to introduce a block of particles

IN%MASS_PER_TIME   = MASS_PER_TIME
IN%MASS_PER_VOLUME = MASS_PER_VOLUME

IF(N_PARTICLES_PER_CELL>0 .AND. N_PARTICLES>0) THEN
WRITE(MESSAGE,'(A,I0,A)') 'ERROR: INIT ',N,' Cannot use both N_PARTICLES and N_PARTICLES_PER_CELL'
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF

IN%N_PARTICLES        = N_PARTICLES
IN%N_PARTICLES_PER_CELL = N_PARTICLES_PER_CELL
IN%PARTICLE_WEIGHT_FACTOR = PARTICLE_WEIGHT_FACTOR

IF ( IN%MASS_PER_VOLUME>0._EB .AND. IN%VOLUME<=TWO_EPSILON_EB) THEN
WRITE(MESSAGE,'(A,I0,A)') 'ERROR: INIT ',N,' XB has no volume'
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF

IN%DT_INSERT = DT_INSERT
IF (DT_INSERT>0._EB) IN%SINGLE_INSERTION = .FALSE.

! Set up a clock to keep track of particle insertions

ALLOCATE(IN%PARTICLE_INSERT_CLOCK(NMESHES),STAT=IZERO)
CALL ChkMemErr('READ','PARTICLE_INSERT_CLOCK',IZERO)
IN%PARTICLE_INSERT_CLOCK = T_BEGIN

ALLOCATE(IN%ALREADY_INSERTED(NMESHES),STAT=IZERO)
```

```
CALL ChkMemErr('READ','ALREADY_INSERTED',IZERO)
IN%ALREADY_INSERTED = .FALSE.

! Assign an index to identify the particle class

PART_ID_IF: IF (PART_ID/='null') THEN

DO NS=1,N_LAGRANGIAN_CLASSES
IF (PART_ID==LAGRANGIAN_PARTICLE_CLASS(NS)%ID) THEN
IN%PART_INDEX = NS
PARTICLE_FILE = .TRUE.
EXIT
ENDIF
ENDDO
IF (IN%PART_INDEX<1) THEN
WRITE(MESSAGE,'(A,A,A)') 'ERROR: PART_ID ',TRIM(PART_ID),' does not exist'
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF
LPC => LAGRANGIAN_PARTICLE_CLASS(IN%PART_INDEX)
IN%N_PARTICLES = N_PARTICLES*MAX(1,LPC%N_ORIENTATION)
IN%N_PARTICLES_PER_CELL = N_PARTICLES_PER_CELL*MAX(1,LPC%N_ORIENTATION)
IF (IN%MASS_PER_TIME>0._EB .OR. IN%MASS_PER_VOLUME>0._EB) THEN
IF (LPC%DENSITY < 0._EB) THEN
WRITE(MESSAGE,'(A,A,A)') 'INIT ERROR: PARTicle class ',TRIM(LPC%ID),' requires a density'
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF
ENDIF

! Make sure that all particles are inside of the domain
IF ( LPC%PERIODIC_X .AND. (IN%X2>=XF_MAX .OR. IN%X1<=XS_MIN) ) THEN
WRITE(MESSAGE,'(A,I0,A,A)') 'ERROR: Problem with INIT number ',N,'. Particle at boundary or outside of domain.'
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF
IF ( LPC%PERIODIC_Y .AND. (IN%Y2>=YF_MAX .OR. IN%Y1<=YS_MIN) ) THEN
WRITE(MESSAGE,'(A,I0,A,A)') 'ERROR: Problem with INIT number ',N,'. Particle at boundary or outside of domain.'
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF
IF ( LPC%PERIODIC_Z .AND. (IN%Z2>=ZF_MAX .OR. IN%Z1<=ZS_MIN) ) THEN
WRITE(MESSAGE,'(A,I0,A,A)') 'ERROR: Problem with INIT number ',N,'. Particle at boundary or outside of domain.'
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF

ENDIF PART_ID_IF

! Initial velocity components

IN%U0 = UVW(1)
IN%V0 = UVW(2)
IN%W0 = UVW(3)

ENDDO I_MULT_LOOP
ENDDO J_MULT_LOOP
ENDDO K_MULT_LOOP

ENDDO INIT_LOOP

! Check if there are any devices that refer to INIT lines

DEVICE_LOOP: DO NN=1,N_DEVC
DV => DEVICE(NN)
IF (DV%INIT_ID=='null') CYCLE
DO I=1,N_INIT
IN => INITIALIZATION(I)
IF (IN%ID==DV%INIT_ID) CYCLE DEVICE_LOOP
ENDDO
WRITE(MESSAGE,'(A,A,A)') 'ERROR: The INIT_ID for DEVC ',TRIM(DV%ID),' cannot be found.'
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDDO DEVICE_LOOP

! Rewind the input file and return

REWIND(LU_INPUT) ; INPUT_FILE_LINE_NUMBER = 0

CONTAINS


SUBROUTINE SET_INIT_DEFAULTS

! Set default values

AUTO_IGNITION_TEMPERATURE = 1.E20_EB
CELL_CENTERED             = .FALSE.
CTRL_ID                   = 'null'
DENSITY                   = -1000._EB
DEVC_ID                   = 'null'
DIAMETER                  = -1._EB
DT_INSERT                 = -1._EB
DX                        =  0._EB
```

```
DY                          =  0._EB
DZ                          =  0._EB
HEIGHT                      = -1._EB
HRRPUV                      =  0._EB
ID                          = 'null'
MASS_FRACTION               =  0._EB
MASS_PER_TIME               = -1._EB
MASS_PER_VOLUME             = -1._EB
MULT_ID                     = 'null'
N_PARTICLES                 = 0
N_PARTICLES_PER_CELL        = 0
PARTICLE_WEIGHT_FACTOR      = 1.0_EB
PART_ID                     = 'null'
RADIUS                      = -1._EB
SHAPE                       = 'BLOCK'
SPEC_ID                     = 'null'
TEMPERATURE                 = -1000._EB
UVW                         = 0._EB
VOLUME_FRACTION             =  0._EB
XB(1)                       = -1000000._EB
XB(2)                       =  1000000._EB
XB(3)                       = -1000000._EB
XB(4)                       =  1000000._EB
XB(5)                       = -1000000._EB
XB(6)                       =  1000000._EB
XYZ                         = -1000000._EB

END SUBROUTINE SET_INIT_DEFAULTS

END SUBROUTINE READ_INIT


SUBROUTINE READ_ZONE

REAL(EB), ALLOCATABLE, DIMENSION(:) :: LEAK_AREA, LEAK_REFERENCE_PRESSURE, LEAK_PRESSURE_EXPONENT
INTEGER  :: N,NM,NN,N_EVAC_ZONE,N_EVAC_MESH,NM_EVAC
LOGICAL :: SEALED,READ_ZONE_LINES,PERIODIC
CHARACTER(LABEL_LENGTH) :: ID
NAMELIST /ZONE/ ID,LEAK_AREA,LEAK_PRESSURE_EXPONENT,LEAK_REFERENCE_PRESSURE,XB,PERIODIC

ALLOCATE (LEAK_AREA(0:MAX_LEAK_PATHS))
ALLOCATE (LEAK_REFERENCE_PRESSURE(0:MAX_LEAK_PATHS))
ALLOCATE (LEAK_PRESSURE_EXPONENT(0:MAX_LEAK_PATHS))

N_ZONE = 0
REWIND(LU_INPUT) ; INPUT_FILE_LINE_NUMBER = 0
COUNT_ZONE_LOOP: DO
CALL CHECKREAD('ZONE',LU_INPUT,IOS)
IF (IOS==1) EXIT COUNT_ZONE_LOOP
READ(LU_INPUT,NML=ZONE,END=11,ERR=12,IOSTAT=IOS)
N_ZONE = N_ZONE + 1
12 IF (IOS>0) THEN
WRITE(MESSAGE,'(A,I0,A,I0)') 'ERROR: Problem with ZONE number ',N_ZONE+1,', line number',INPUT_FILE_LINE_NUMBER
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF
ENDDO COUNT_ZONE_LOOP
11 REWIND(LU_INPUT) ; INPUT_FILE_LINE_NUMBER = 0

! Check to see if there are any OPEN vents. If there are not, and there are no declared pressure ZONEs, stop with an error.

SEALED = .TRUE.

N_EVAC_ZONE = 0
DO NM=1,NMESHES
IF (.NOT.EVACUATION_ONLY(NM)) THEN
M => MESHES(NM)
DO N=1,M%N_VENT
VT => M%VENTS(N)
IF (VT%BOUNDARY_TYPE==OPEN_BOUNDARY)      SEALED = .FALSE.
IF (VT%BOUNDARY_TYPE==PERIODIC_BOUNDARY) SEALED = .FALSE.
ENDDO
ELSE
IF (EVACUATION_SKIP(NM)) N_EVAC_ZONE = N_EVAC_ZONE + 1
END IF
ENDDO

! If the whole domain lacks on OPEN or PERIODIC boundary, assume it to be one big pressure zone

READ_ZONE_LINES = .TRUE.
IF (SEALED .AND. N_ZONE==0) THEN
N_ZONE = 1
READ_ZONE_LINES = .FALSE.
ENDIF

IF (ANY(EVACUATION_SKIP)) THEN
IF (READ_ZONE_LINES) THEN
N_ZONE = N_ZONE + N_EVAC_ZONE
```

```
ELSE
N_ZONE = N_EVAC_ZONE
ENDIF
END IF

! Make sure that there are no leak paths to undefined pressure ZONEs

DO N=0,N_SURF
SF => SURFACE(N)
IF (SF%LEAK_PATH(1)>(N_ZONE-N_EVAC_ZONE) .OR. SF%LEAK_PATH(2)>(N_ZONE-N_EVAC_ZONE)) SF%LEAK_PATH = -1
ENDDO

! Allocate array to indicate if pressure ZONEs are connected

ALLOCATE(CONNECTED_ZONES(0:N_ZONE,0:N_ZONE,NMESHES),STAT=IZERO)
CALL ChkMemErr('READ','CONNECTED_ZONES',IZERO)
CONNECTED_ZONES = .FALSE.

! If there are no ZONE lines, return

IF (N_ZONE==0) RETURN

! Allocate ZONE arrays

ALLOCATE(P_ZONE(N_ZONE),STAT=IZERO)
CALL ChkMemErr('READ','P_ZONE',IZERO)

! Read in and process ZONE lines

READ_ZONE_LOOP: DO N=1,N_ZONE-N_EVAC_ZONE

ALLOCATE(P_ZONE(N)%LEAK_AREA(0:N_ZONE),STAT=IZERO)
CALL ChkMemErr('READ','LEAK_AREA',IZERO)
ALLOCATE(P_ZONE(N)%LEAK_PRESSURE_EXPONENT(0:N_ZONE),STAT=IZERO)
CALL ChkMemErr('READ','LEAK_PRESSURE_EXPONENT',IZERO)
ALLOCATE(P_ZONE(N)%LEAK_REFERENCE_PRESSURE(0:N_ZONE),STAT=IZERO)
CALL ChkMemErr('READ','LEAK_REFERENCE_PRESSURE',IZERO)

IF (N<1000) WRITE(ID,'(A,I3)') 'ZONE_',N
IF (N<100)  WRITE(ID,'(A,I2)') 'ZONE_',N
IF (N<10)   WRITE(ID,'(A,I1)') 'ZONE_',N
LEAK_AREA      = 0._EB
LEAK_REFERENCE_PRESSURE = 4._EB
LEAK_PRESSURE_EXPONENT = 0.5_EB
XB(1)          = -1000000._EB
XB(2)          =  1000000._EB
XB(3)          = -1000000._EB
XB(4)          =  1000000._EB
XB(5)          = -1000000._EB
XB(6)          =  1000000._EB
PERIODIC       = .FALSE.

IF (READ_ZONE_LINES) THEN
CALL CHECKREAD('ZONE',LU_INPUT,IOS)
IF (IOS==1) EXIT READ_ZONE_LOOP
READ(LU_INPUT,ZONE)
ENDIF

CALL CHECK_XB(XB)

P_ZONE(N)%ID = ID
P_ZONE(N)%LEAK_AREA(0:N_ZONE) = LEAK_AREA(0:N_ZONE)
P_ZONE(N)%LEAK_REFERENCE_PRESSURE(0:N_ZONE) = LEAK_REFERENCE_PRESSURE(0:N_ZONE)
P_ZONE(N)%LEAK_PRESSURE_EXPONENT(0:N_ZONE) = LEAK_PRESSURE_EXPONENT(0:N_ZONE)
P_ZONE(N)%X1 = XB(1)
P_ZONE(N)%X2 = XB(2)
P_ZONE(N)%Y1 = XB(3)
P_ZONE(N)%Y2 = XB(4)
P_ZONE(N)%Z1 = XB(5)
P_ZONE(N)%Z2 = XB(6)
P_ZONE(N)%EVACUATION = .FALSE.
P_ZONE(N)%PERIODIC = PERIODIC
IF (N > 1) THEN
DO NN = 1,N-1
IF(P_ZONE(NN)%LEAK_AREA(N)>0._EB) THEN
IF(P_ZONE(N)%LEAK_AREA(NN) > 0._EB) THEN
WRITE(MESSAGE,'(A,I0,A,I0)') 'ERROR: LEAK_AREA specified twice for ZONE ',N,' and ',NN
CALL SHUTDOWN(MESSAGE) ; RETURN
ELSE
P_ZONE(N)%LEAK_AREA(NN) = P_ZONE(NN)%LEAK_AREA(N)
P_ZONE(N)%LEAK_REFERENCE_PRESSURE(NN) = P_ZONE(NN)%LEAK_REFERENCE_PRESSURE(N)
P_ZONE(N)%LEAK_PRESSURE_EXPONENT(NN) = P_ZONE(NN)%LEAK_PRESSURE_EXPONENT(N)
ENDIF
ENDIF
ENDDO
ENDIF

ENDDO READ_ZONE_LOOP
```

```

READ_EVACUATION_ZONE_LOOP: DO N=N_ZONE-N_EVAC_ZONE+1,N_ZONE
ALLOCATE(P_ZONE(N)%LEAK_AREA(0:N_ZONE),STAT=IZERO)
CALL ChkMemErr('READ','LEAK_AREA',IZERO)

WRITE(ID,'(A,I2.2)') 'ZONE_',N
LEAK_AREA      = 0._EB
N_EVAC_MESH = 0
NM_EVAC = 0
EVAC_MESH_NUMBER: DO NM=1,NMESHES
IF (EVACUATION_SKIP(NM)) THEN
N_EVAC_MESH = N_EVAC_MESH + 1
NM_EVAC = NM
IF (N_EVAC_MESH == N-(N_ZONE-N_EVAC_ZONE)) EXIT EVAC_MESH_NUMBER
END IF
ENDDO EVAC_MESH_NUMBER

XB(1)          = 0.5_EB*(MESHES(NM_EVAC)%XS+MESHES(NM_EVAC)%XF)
XB(2)          = 0.5_EB*(MESHES(NM_EVAC)%XS+MESHES(NM_EVAC)%XF)
XB(3)          = 0.5_EB*(MESHES(NM_EVAC)%YS+MESHES(NM_EVAC)%YF)
XB(4)          = 0.5_EB*(MESHES(NM_EVAC)%YS+MESHES(NM_EVAC)%YF)
XB(5)          = MESHES(NM_EVAC)%ZS
XB(6)          = MESHES(NM_EVAC)%ZF

P_ZONE(N)%ID = ID
P_ZONE(N)%ID = TRIM('EvacPzone_') // TRIM(MESH_NAME(NM_EVAC))
P_ZONE(N)%LEAK_AREA(0:N_ZONE) = LEAK_AREA(0:N_ZONE)
P_ZONE(N)%X1 = XB(1)-0.5_EB
P_ZONE(N)%X2 = XB(2)+0.5_EB
P_ZONE(N)%Y1 = XB(3)-0.5_EB
P_ZONE(N)%Y2 = XB(4)+0.5_EB
P_ZONE(N)%Z1 = XB(5)
P_ZONE(N)%Z2 = XB(6)
P_ZONE(N)%EVACUATION = .TRUE.
P_ZONE(N)%MESH_INDEX = NM_EVAC

ENDDO READ_EVACUATION_ZONE_LOOP

REWIND(LU_INPUT) ; INPUT_FILE_LINE_NUMBER = 0

DEALLOCATE (LEAK_AREA)

END SUBROUTINE READ_ZONE


SUBROUTINE READ_DEVC

! Just read in the DEViCes and the store the info in DEVICE()

USE DEVICE_VARIABLES, ONLY: DEVICE_TYPE, DEVICE, N_DEVC, N_DEVC_TIME, N_DEVC_LINE,MAX_DEVC_LINE_POINTS,
      DEVC_PIPE_OPERATING
INTEGER  :: NN,NM,MESH_NUMBER,N_DEVC_READ,IOR,TRIP_DIRECTION,VELO_INDEX,POINTS,I_POINT,PIPE_INDEX,
      ORIENTATION_INDEX, &
ORIENTATION_NUMBER,STATISTICS_LOCATION_INDEX,GHOST_CELL_IOR(3)
REAL(EB) :: DEPTH,ORIENTATION(3),ROTATION,SETPOINT,FLOWRATE,BYPASS_FLOWRATE,DELAY,XYZ(3),CONVERSION_FACTOR,
      SMOOTHING_FACTOR,&
OR_TEMP(3),QUANTITY_RANGE(2),STATISTICS_START,COORD_FACTOR
CHARACTER(LABEL_LENGTH) :: QUANTITY,QUANTITY2,PROP_ID,CTRL_ID,DEVC_ID,INIT_ID,SURF_ID,STATISTICS,PART_ID,MATL_ID,
      SPEC_ID,UNITS, &
DUCT_ID,NODE_ID(2),R_ID,X_ID,Y_ID,Z_ID,NO_UPDATE_DEVC_ID,NO_UPDATE_CTRL_ID,REAC_ID
LOGICAL :: INITIAL_STATE,LATCH,DRY,TIME_AVERAGED,EVACUATION,HIDE_COORDINATES,RELATIVE,OUTPUT,
      NEW_ORIENTATION_VECTOR,TIME_HISTORY,&
LINE_DEVICE
TYPE (DEVICE_TYPE), POINTER :: DV=>NULL()
NAMELIST /DEVC/ BYPASS_FLOWRATE,CONVERSION_FACTOR,COORD_FACTOR,CTRL_ID,DELAY,DEPTH,DEVC_ID,DRY,DUCT_ID,EVACUATION
      ,FLOWRATE,FYI,&
GHOST_CELL_IOR,HIDE_COORDINATES,ID,INITIAL_STATE,INIT_ID,IOR,LATCH,MATL_ID,NODE_ID, &
NO_UPDATE_DEVC_ID,NO_UPDATE_CTRL_ID,ORIENTATION,ORIENTATION_NUMBER,OUTPUT,PART_ID,PIPE_INDEX,POINTS,&
PROP_ID,QUANTITY,QUANTITY2,QUANTITY_RANGE,&
REAC_ID,RELATIVE,R_ID,ROTATION,SETPOINT,SMOOTHING_FACTOR,SPEC_ID,STATISTICS,STATISTICS_START,SURF_ID,&
TIME_AVERAGED,TIME_HISTORY,TRIP_DIRECTION,UNITS,VELO_INDEX,XB,XYZ,X_ID,Y_ID,Z_ID

! Read the input file and count the number of DEVC lines

N_DEVC = 0
N_DEVC_READ = 0
N_DEVC_TIME = 0
N_DEVC_LINE = 0

REWIND(LU_INPUT) ; INPUT_FILE_LINE_NUMBER = 0
COUNT_DEVC_LOOP: DO
CALL CHECKREAD('DEVC',LU_INPUT,IOS)
IF (IOS==1) EXIT COUNT_DEVC_LOOP
POINTS = 1
TIME_HISTORY = .FALSE.
READ(LU_INPUT,NML=DEVC,END=11,ERR=12,IOSTAT=IOS)
N_DEVC       = N_DEVC       + POINTS
N_DEVC_READ = N_DEVC_READ + 1
```

```
IF (POINTS>1 .AND. .NOT.TIME_HISTORY) MAX_DEVC_LINE_POINTS = MAX(MAX_DEVC_LINE_POINTS,POINTS)
12 IF (IOS>0) THEN
WRITE(MESSAGE,'(A,I0,A,I0)') 'ERROR: Problem with DEVC number ',N_DEVC_READ+1,', line number',
       INPUT_FILE_LINE_NUMBER
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF
ENDDO COUNT_DEVC_LOOP
11 REWIND(LU_INPUT) ; INPUT_FILE_LINE_NUMBER = 0

IF (N_DEVC==0) RETURN

! Allocate DEVICE array to hold all information for each device

ALLOCATE(DEVICE(N_DEVC),STAT=IZERO) ; CALL ChkMemErr('READ','DEVICE',IZERO)

! Speceial case for QUANTITY='RADIATIVE HEAT FLUX GAS' or 'ADIABATIC SURFACE TEMPERATURE GAS'

ALLOCATE(INIT_RESERVED(N_DEVC),STAT=IZERO) ; CALL ChkMemErr('READ','INIT_RESERVED',IZERO)
N_INIT_RESERVED = 0

! Read in the DEVC lines, keeping track of TIME-history devices, and LINE array devices

N_DEVC = 0

READ_DEVC_LOOP: DO NN=1,N_DEVC_READ

CALL CHECKREAD('DEVC',LU_INPUT,IOS)
IF (IOS==1) EXIT READ_DEVC_LOOP
CALL SET_DEVC_DEFAULTS
READ(LU_INPUT,DEVC)

IF (QUANTITY_RANGE(2) <= QUANTITY_RANGE(1)) THEN
WRITE(MESSAGE,'(A,A,A)')  'ERROR: DEVC ',TRIM(ID),' has QUANTITY_RANGE(2) <= QUANTITY_RANGE(1)'
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF

! Determine if the device is a steady-state "line" device or the usual time-history device.

LINE_DEVICE = .FALSE.
IF (POINTS>1 .AND. .NOT.TIME_HISTORY) LINE_DEVICE = .TRUE.

! Parse 'MAXLOC X', etc.

IF (STATISTICS(1:6)=='MAXLOC' .OR. STATISTICS(1:6)=='MINLOC') THEN
IF (STATISTICS(8:8)=='X') STATISTICS_LOCATION_INDEX = 1
IF (STATISTICS(8:8)=='Y') STATISTICS_LOCATION_INDEX = 2
IF (STATISTICS(8:8)=='Z') STATISTICS_LOCATION_INDEX = 3
IF (STATISTICS(1:6)=='MAXLOC') STATISTICS = 'MAX'
IF (STATISTICS(1:6)=='MINLOC') STATISTICS = 'MIN'
UNITS = 'm'
ENDIF

! Error statement involving POINTS

IF (POINTS>1 .AND. ANY(XB<-1.E5_EB) .AND. INIT_ID=='null') THEN
WRITE(MESSAGE,'(A,A,A)')  'ERROR: DEVC ',TRIM(ID),' must have coordinates given in terms of XB'
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF
IF (LINE_DEVICE .AND. TRIM(STATISTICS)/='null'      .AND. &
TRIM(STATISTICS)/='RMS'        .AND. &
TRIM(STATISTICS)/='COV'        .AND. &
TRIM(STATISTICS)/='CORRCOEF'   .AND. &
TRIM(STATISTICS)/='TIME MIN'   .AND. &
TRIM(STATISTICS)/='TIME MAX') THEN
WRITE(MESSAGE,'(A,A,A)')  'ERROR: DEVC ',TRIM(ID),' cannot use a steady-state line device and STATISTICS at the
       same time'
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF
IF (.NOT.LINE_DEVICE .AND. (TRIM(STATISTICS)=='TIME MIN' .OR. &
TRIM(STATISTICS)=='TIME MAX') ) THEN
WRITE(MESSAGE,'(A,A,A)')  'ERROR: DEVC ',TRIM(ID),' cannot compute STATISTICS. Set POINTS>1.'
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF

! Make ORIENTATION consistent with IOR

SELECT CASE(IOR)
CASE( 1) ; ORIENTATION=(/ 1._EB, 0._EB, 0._EB/)
CASE(-1) ; ORIENTATION=(/-1._EB, 0._EB, 0._EB/)
CASE( 2) ; ORIENTATION=(/ 0._EB, 1._EB, 0._EB/)
CASE(-2) ; ORIENTATION=(/ 0._EB,-1._EB, 0._EB/)
CASE( 3) ; ORIENTATION=(/ 0._EB, 0._EB, 1._EB/)
CASE(-3) ; ORIENTATION=(/ 0._EB, 0._EB,-1._EB/)
END SELECT

! Add ORIENTATION to global list

NEW_ORIENTATION_VECTOR = .TRUE.
```

```
ORIENTATION_INDEX = 0

DO I=1,N_ORIENTATION_VECTOR
IF (ORIENTATION(1)==ORIENTATION_VECTOR(1,I) .AND. &
ORIENTATION(2)==ORIENTATION_VECTOR(2,I) .AND. &
ORIENTATION(3)==ORIENTATION_VECTOR(3,I)) THEN
NEW_ORIENTATION_VECTOR = .FALSE.
ORIENTATION_INDEX = I
EXIT
ENDIF
ENDDO

IF (NEW_ORIENTATION_VECTOR) THEN
OR_TEMP(1:3) = ORIENTATION(1:3)
N_ORIENTATION_VECTOR = N_ORIENTATION_VECTOR + 1
IF (N_ORIENTATION_VECTOR>UBOUND(ORIENTATION_VECTOR,DIM=2)) THEN
ORIENTATION_VECTOR => REALLOCATE2D(ORIENTATION_VECTOR,1,3,1,N_ORIENTATION_VECTOR+10)
ENDIF
IF (ALL(ABS(OR_TEMP(1:3))<TWO_EPSILON_EB)) THEN
WRITE(MESSAGE,'(A,A,A)') 'ERROR: DEVC ',TRIM(ID),'. All components of ORIENTATION are zero.'
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF
ORIENTATION_VECTOR(1:3,N_ORIENTATION_VECTOR) = ORIENTATION(1:3)/ NORM2(OR_TEMP)
ORIENTATION_INDEX = N_ORIENTATION_VECTOR
ENDIF

! Check if there are any devices with specified XB that do not fall within a mesh.

IF (POINTS==1 .AND. XB(1)>-1.E5_EB) THEN

IF (QUANTITY/='PATH OBSCURATION' .AND. QUANTITY/='TRANSMISSION') CALL CHECK_XB(XB)

BAD = .TRUE.
CHECK_MESH_LOOP: DO NM=1,NMESHES
IF (EVACUATION_ONLY(NM)) CYCLE CHECK_MESH_LOOP
M=>MESHES(NM)
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
BAD = .FALSE.
EXIT CHECK_MESH_LOOP
ENDIF
ENDDO CHECK_MESH_LOOP

IF (BAD .AND. .NOT.ALL(EVACUATION_ONLY)) THEN
WRITE(MESSAGE,'(A,A,A)')  'ERROR: XB for DEVC ',TRIM(ID),' must be completely within a mesh.'
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF

ENDIF

! Process the point devices along a line, if necessary

POINTS_LOOP: DO I_POINT=1,POINTS

IF (XB(1)>-1.E5_EB) THEN
IF (TRIM(QUANTITY)=='VELOCITY PATCH') THEN
IF (XYZ(1) <-1.E5_EB) THEN
XYZ(1) = XB(1) + (XB(2)-XB(1))/2._EB
XYZ(2) = XB(3) + (XB(4)-XB(3))/2._EB
XYZ(3) = XB(5) + (XB(6)-XB(5))/2._EB
ENDIF
ELSE
IF (POINTS > 1) THEN
XYZ(1) = XB(1) + (XB(2)-XB(1))*REAL(I_POINT-1,EB)/REAL(MAX(POINTS-1,1),EB)
XYZ(2) = XB(3) + (XB(4)-XB(3))*REAL(I_POINT-1,EB)/REAL(MAX(POINTS-1,1),EB)
XYZ(3) = XB(5) + (XB(6)-XB(5))*REAL(I_POINT-1,EB)/REAL(MAX(POINTS-1,1),EB)
ELSE
XYZ(1) = XB(1) + (XB(2)-XB(1))/2._EB
XYZ(2) = XB(3) + (XB(4)-XB(3))/2._EB
XYZ(3) = XB(5) + (XB(6)-XB(5))/2._EB
ENDIF
ENDIF
ELSE
IF (XYZ(1) < -1.E5_EB .AND. DUCT_ID=='null' .AND. NODE_ID(1)=='null' .AND. INIT_ID=='null') THEN
WRITE(MESSAGE,'(A,A,A)')  'ERROR: DEVC ',TRIM(ID),' must have coordinates, even if it is not a point quantity'
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF
ENDIF

! Determine which mesh the device is in

BAD = .TRUE.
MESH_LOOP: DO NM=1,NMESHES
IF (EVACUATION_ONLY(NM)) CYCLE MESH_LOOP
M=>MESHES(NM)
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)
    THEN
IF (ABS(XYZ(1)-M%XS)<TWO_EPSILON_EB) XYZ(1) = XYZ(1) + 0.01_EB*M%DXI
IF (ABS(XYZ(1)-M%XF)<TWO_EPSILON_EB) XYZ(1) = XYZ(1) - 0.01_EB*M%DXI
```

```
IF (ABS(XYZ(2)-M%YS)<TWO_EPSILON_EB) XYZ(2) = XYZ(2) + 0.01_EB*M%DETA
IF (ABS(XYZ(2)-M%YF)<TWO_EPSILON_EB) XYZ(2) = XYZ(2) - 0.01_EB*M%DETA
IF (ABS(XYZ(3)-M%ZS)<TWO_EPSILON_EB) XYZ(3) = XYZ(3) + 0.01_EB*M%DZETA
IF (ABS(XYZ(3)-M%ZF)<TWO_EPSILON_EB) XYZ(3) = XYZ(3) - 0.01_EB*M%DZETA
MESH_NUMBER = NM
BAD = .FALSE.
EXIT MESH_LOOP
ENDIF
ENDDO MESH_LOOP

! Process EVAC meshes

EVACUATION_MESH_LOOP: DO NM=1,NMESHES
IF (.NOT.EVACUATION_SKIP(NM)) CYCLE EVACUATION_MESH_LOOP
M=>MESHES(NM)
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) THEN
IF (BAD) MESH_NUMBER = NM
IF (.NOT.BAD .AND. EVACUATION .AND. QUANTITY=='TIME' .AND. SETPOINT<=T_BEGIN) THEN
MESH_NUMBER = NM
BAD = .FALSE.
END IF
EXIT EVACUATION_MESH_LOOP
ENDIF
ENDDO EVACUATION_MESH_LOOP

! Make sure there is either a QUANTITY or PROP_ID for the DEVICE

IF (QUANTITY=='null' .AND. PROP_ID=='null') THEN
WRITE(MESSAGE,'(A,A,A)') 'ERROR: DEVC ',TRIM(ID),' must have either an output QUANTITY or PROP_ID'
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF

IF (BAD) THEN
IF (DUCT_ID/='null' .OR. NODE_ID(1)/='null' .OR. INIT_ID/='null') THEN
XYZ(1) = MESHES(1)%XS
XYZ(2) = MESHES(1)%YS
XYZ(3) = MESHES(1)%ZS
MESH_NUMBER = 1
ELSE
IF (ALL(EVACUATION_ONLY)) CYCLE READ_DEVC_LOOP
WRITE(MESSAGE,'(A,A,A)') 'WARNING: DEVC ',TRIM(ID),' is not within any mesh.'
IF (MYID==0) WRITE(LU_ERR,'(A)') TRIM(MESSAGE)
CYCLE READ_DEVC_LOOP
ENDIF
ENDIF

! Don't print out clocks

IF (QUANTITY=='TIME' .AND. NO_UPDATE_DEVC_ID=='null' .AND. NO_UPDATE_CTRL_ID=='null' ) OUTPUT = .FALSE.

! Determine if the DEVC is a TIME or LINE device

IF (.NOT.LINE_DEVICE .AND. OUTPUT)  N_DEVC_TIME = N_DEVC_TIME + 1
IF (LINE_DEVICE .AND. I_POINT==1)   N_DEVC_LINE = N_DEVC_LINE + 1

! Assign properties to the DEVICE array

N_DEVC = N_DEVC + 1

DV => DEVICE(N_DEVC)

DV%RELATIVE           = RELATIVE
DV%CONVERSION_FACTOR  = CONVERSION_FACTOR
DV%COORD_FACTOR       = COORD_FACTOR
DV%DEPTH              = DEPTH
DV%IOR                = IOR
IF (POINTS>1 .AND. POINTS<=99 .AND. .NOT.LINE_DEVICE) THEN
WRITE(DV%ID,'(A,A,I2.2)') TRIM(ID),'-',I_POINT
ELSEIF (POINTS>99 .AND. POINTS<=999 .AND. .NOT.LINE_DEVICE) THEN
WRITE(DV%ID,'(A,A,I3.3)') TRIM(ID),'-',I_POINT
ELSEIF (POINTS>999 .AND. .NOT.LINE_DEVICE) THEN
WRITE(DV%ID,'(A,A,I6.6)') TRIM(ID),'-',I_POINT
ELSE
DV%ID               = ID
ENDIF
IF (LINE_DEVICE) DV%LINE = N_DEVC_LINE
DV%POINT              = I_POINT
DV%MESH               = MESH_NUMBER
DV%ORDINAL            = NN
DV%ORIENTATION_INDEX  = ORIENTATION_INDEX
DV%PROP_ID            = PROP_ID
DV%DEVC_ID            = DEVC_ID
DV%CTRL_ID            = CTRL_ID
DV%SURF_ID            = SURF_ID
DV%PART_ID            = PART_ID
DV%MATL_ID            = MATL_ID
DV%SPEC_ID            = SPEC_ID
```

```fortran
DV%DUCT_ID               = DUCT_ID
DV%INIT_ID               = INIT_ID
DV%NODE_ID               = NODE_ID
DV%REAC_ID               = REAC_ID
DV%QUANTITY              = QUANTITY
DV%QUANTITY2             = QUANTITY2
DV%ROTATION              = ROTATION*TWOPI/360._EB
DV%SETPOINT              = SETPOINT
DV%LATCH                 = LATCH
DV%OUTPUT                = OUTPUT
DV%ORIENTATION_NUMBER = ORIENTATION_NUMBER
DV%TRIP_DIRECTION        = TRIP_DIRECTION
DV%INITIAL_STATE         = INITIAL_STATE
DV%CURRENT_STATE         = INITIAL_STATE
DV%PRIOR_STATE           = INITIAL_STATE
DV%FLOWRATE              = FLOWRATE
DV%BYPASS_FLOWRATE       = BYPASS_FLOWRATE
DV%SMOOTHING_FACTOR      = SMOOTHING_FACTOR
DV%STATISTICS            = STATISTICS
DV%STATISTICS_LOCATION_INDEX = STATISTICS_LOCATION_INDEX
DV%STATISTICS_START      = STATISTICS_START
DV%TIME_AVERAGED         = TIME_AVERAGED
DV%SURF_INDEX            = 0
DV%UNITS                 = UNITS
DV%DELAY                 = DELAY / TIME_SHRINK_FACTOR
DV%X1                    = XB(1)
DV%X2                    = XB(2)
DV%Y1                    = XB(3)
DV%Y2                    = XB(4)
DV%Z1                    = XB(5)
DV%Z2                    = XB(6)
DV%X                     = XYZ(1)
DV%Y                     = XYZ(2)
DV%Z                     = XYZ(3)
IF (X_ID=='null') X_ID = TRIM(ID)//'-x'
IF (Y_ID=='null') Y_ID = TRIM(ID)//'-y'
IF (Z_ID=='null') Z_ID = TRIM(ID)//'-z'
DV%R_ID                  = R_ID
DV%X_ID                  = X_ID
DV%Y_ID                  = Y_ID
DV%Z_ID                  = Z_ID
DV%DRY                   = DRY
DV%EVACUATION            = EVACUATION
DV%VELO_INDEX            = VELO_INDEX
DV%PIPE_INDEX            = PIPE_INDEX
DV%NO_UPDATE_DEVC_ID = NO_UPDATE_DEVC_ID
DV%NO_UPDATE_CTRL_ID = NO_UPDATE_CTRL_ID
DV%QUANTITY_RANGE        = QUANTITY_RANGE
DV%GHOST_CELL_IOR        = GHOST_CELL_IOR

IF (LINE_DEVICE) THEN
IF (.NOT.HIDE_COORDINATES) THEN
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))) DV%LINE_COORD_CODE = 1
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))) DV%LINE_COORD_CODE = 2
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))) DV%LINE_COORD_CODE = 3
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))) DV%LINE_COORD_CODE = 12
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))) DV%LINE_COORD_CODE = 13
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))) DV%LINE_COORD_CODE = 23
IF (DV%R_ID/='null') DV%LINE_COORD_CODE = 4  ! Special case where radial coordinates are requested
ELSE
DV%LINE_COORD_CODE = 0
ENDIF
ENDIF

! Special case for QUANTITY='RADIATIVE HEAT FLUX GAS' or 'ADIABATIC SURFACE TEMPERATURE GAS'.
! Save information to create INIT line.

IF (DV%QUANTITY=='RADIATIVE HEAT FLUX GAS' .OR. &
DV%QUANTITY=='RADIANCE' .OR. &
DV%QUANTITY=='ADIABATIC SURFACE TEMPERATURE GAS') THEN
DV%INIT_ID = DV%ID
TARGET_PARTICLES_INCLUDED = .TRUE.
IF (DV%POINT==1) THEN
N_INIT_RESERVED = N_INIT_RESERVED + 1
INIT_RESERVED(N_INIT_RESERVED)%DEVC_INDEX = N_DEVC
INIT_RESERVED(N_INIT_RESERVED)%N_PARTICLES = POINTS
IF (POINTS>1) THEN
INIT_RESERVED(N_INIT_RESERVED)%XYZ(1) = XB(1)
INIT_RESERVED(N_INIT_RESERVED)%XYZ(2) = XB(3)
INIT_RESERVED(N_INIT_RESERVED)%XYZ(3) = XB(5)
INIT_RESERVED(N_INIT_RESERVED)%DX = (XB(2)-XB(1))/(REAL(POINTS,EB)-1)
INIT_RESERVED(N_INIT_RESERVED)%DY = (XB(4)-XB(3))/(REAL(POINTS,EB)-1)
```

```fortran
INIT_RESERVED(N_INIT_RESERVED)%DZ = (XB(6)-XB(5))/(REAL(POINTS,EB)-1)
ENDIF
ENDIF
ENDIF

ENDDO POINTS_LOOP

! Coordinates for non-point devices

IF ((XB(1)>-1.E5_EB .OR. STATISTICS/='null') .AND. POINTS==1) THEN
NM = DV%MESH
M=>MESHES(NM)
XB(1) = MAX(XB(1),M%XS)
XB(2) = MIN(XB(2),M%XF)
XB(3) = MAX(XB(3),M%YS)
XB(4) = MIN(XB(4),M%YF)
XB(5) = MAX(XB(5),M%ZS)
XB(6) = MIN(XB(6),M%ZF)
DV%X1 = XB(1)
DV%X2 = XB(2)
DV%Y1 = XB(3)
DV%Y2 = XB(4)
DV%Z1 = XB(5)
DV%Z2 = XB(6)
DV%I1 = NINT( GINV(XB(1)-M%XS,1,NM)*M%RDXI )
DV%I2 = NINT( GINV(XB(2)-M%XS,1,NM)*M%RDXI )
DV%J1 = NINT( GINV(XB(3)-M%YS,2,NM)*M%RDETA )
DV%J2 = NINT( GINV(XB(4)-M%YS,2,NM)*M%RDETA )
DV%K1 = NINT( GINV(XB(5)-M%ZS,3,NM)*M%RDZETA )
DV%K2 = NINT( GINV(XB(6)-M%ZS,3,NM)*M%RDZETA )
IF (DV%I1<DV%I2) DV%I1 = DV%I1 + 1
IF (DV%J1<DV%J2) DV%J1 = DV%J1 + 1
IF (DV%K1<DV%K2) DV%K1 = DV%K1 + 1
IF (ABS(XB(1)-XB(2))<=SPACING(XB(2)) .AND. IOR==0) DV%IOR_ASSUMED = 1
IF (ABS(XB(3)-XB(4))<=SPACING(XB(4)) .AND. IOR==0) DV%IOR_ASSUMED = 2
IF (ABS(XB(5)-XB(6))<=SPACING(XB(6)) .AND. IOR==0) DV%IOR_ASSUMED = 3
ENDIF

IF (TRIM(DV%QUANTITY) == 'CHEMISTRY SUBITERATIONS') OUTPUT_CHEM_IT = .TRUE.

IF (TRIM(DV%QUANTITY) == 'REAC SOURCE TERM' .OR. TRIM(DV%QUANTITY) == 'HRRPUV REAC') REAC_SOURCE_CHECK=.TRUE.

IF (TRIM(QUANTITY)=='SOLID CELL Q_S') STORE_Q_DOT_PPP_S = .TRUE.

IF (TRIM(QUANTITY)=='DUDT' .OR. TRIM(QUANTITY)=='DVDT' .OR. TRIM(QUANTITY)=='DWDT') STORE_OLD_VELOCITY=.TRUE.

ENDDO READ_DEVC_LOOP

ALLOCATE (DEVC_PIPE_OPERATING(MAXVAL(DEVICE%PIPE_INDEX)))
DEVC_PIPE_OPERATING = 0

REWIND(LU_INPUT) ; INPUT_FILE_LINE_NUMBER = 0

CONTAINS

SUBROUTINE SET_DEVC_DEFAULTS

RELATIVE          = .FALSE.
CONVERSION_FACTOR = 1._EB
COORD_FACTOR      = 1._EB
DEPTH             = 0._EB
IOR               = 0
ID                = 'null'
ORIENTATION(1:3)  = (/0._EB,0._EB,-1._EB/)
PROP_ID           = 'null'
CTRL_ID           = 'null'
DEVC_ID           = 'null'
SURF_ID           = 'null'
PART_ID           = 'null'
MATL_ID           = 'null'
SPEC_ID           = 'null'
DUCT_ID           = 'null'
INIT_ID           = 'null'
NODE_ID           = 'null'
REAC_ID           = 'null'
FLOWRATE          = 0._EB
DELAY             = 0._EB
BYPASS_FLOWRATE   = 0._EB
QUANTITY          = 'null'
QUANTITY2         = 'null'
ROTATION          = 0._EB
XB(1)             = -1.E6_EB
XB(2)             =  1.E6_EB
XB(3)             = -1.E6_EB
XB(4)             =  1.E6_EB
XB(5)             = -1.E6_EB
XB(6)             =  1.E6_EB
INITIAL_STATE     = .FALSE.
```

```
LATCH              = .TRUE.
OUTPUT             = .TRUE.
ORIENTATION_NUMBER = 1
POINTS             = 1
SETPOINT           = 1.E20_EB
SMOOTHING_FACTOR   = 0._EB
STATISTICS         = 'null'
STATISTICS_START   = -1.E20_EB
TRIP_DIRECTION     = 1
TIME_AVERAGED      = .TRUE.
TIME_HISTORY       = .FALSE.
UNITS              = 'null'
VELO_INDEX         = 0
XYZ                = -1.E6_EB
R_ID               = 'null'
X_ID               = 'null'
Y_ID               = 'null'
Z_ID               = 'null'
HIDE_COORDINATES   = .FALSE.
DRY                = .FALSE.
EVACUATION         = .FALSE.
PIPE_INDEX         = 1
NO_UPDATE_DEVC_ID= 'null'
NO_UPDATE_CTRL_ID= 'null'
QUANTITY_RANGE(1)= -1.E50_EB
QUANTITY_RANGE(2)=  1.E50_EB
STATISTICS_LOCATION_INDEX = 0
GHOST_CELL_IOR(1:3) = (/0,0,0/)

END SUBROUTINE SET_DEVC_DEFAULTS

END SUBROUTINE READ_DEVC


SUBROUTINE READ_CTRL

! Just read in the ConTRoL parameters and store in the array CONTROL

USE CONTROL_VARIABLES
USE MATH_FUNCTIONS, ONLY : GET_RAMP_INDEX

LOGICAL :: INITIAL_STATE, LATCH, EVACUATION
INTEGER :: CYCLES,N,NC,TRIP_DIRECTION
REAL(EB) :: SETPOINT(2), DELAY, CYCLE_TIME,CONSTANT,PROPORTIONAL_GAIN,INTEGRAL_GAIN,DIFFERENTIAL_GAIN, TARGET_VALUE
CHARACTER(LABEL_LENGTH) :: ID,FUNCTION_TYPE,INPUT_ID(40),RAMP_ID,ON_BOUND
TYPE (CONTROL_TYPE), POINTER :: CF=>NULL()
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

N_CTRL = 0
REWIND(LU_INPUT) ; INPUT_FILE_LINE_NUMBER = 0
COUNT_CTRL_LOOP: DO
CALL CHECKREAD('CTRL',LU_INPUT,IOS)
IF (IOS==1) EXIT COUNT_CTRL_LOOP
READ(LU_INPUT,NML=CTRL,END=11,ERR=12,IOSTAT=IOS)
N_CTRL = N_CTRL + 1
12 IF (IOS>0) THEN
WRITE(MESSAGE,'(A,I0,A,I0)') 'ERROR: Problem with CTRL number ',N_CTRL+1,', line number',INPUT_FILE_LINE_NUMBER
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF
ENDDO COUNT_CTRL_LOOP
11 REWIND(LU_INPUT) ; INPUT_FILE_LINE_NUMBER = 0

IF (N_CTRL==0) RETURN

! Allocate CONTROL array and set initial values of all to 0

ALLOCATE(CONTROL(N_CTRL),STAT=IZERO)
CALL ChkMemErr('READ','CONTROL',IZERO)

! Read in the CTRL lines

READ_CTRL_LOOP: DO NC=1,N_CTRL

CALL CHECKREAD('CTRL',LU_INPUT,IOS)
IF (IOS==1) EXIT READ_CTRL_LOOP
CALL SET_CTRL_DEFAULTS
READ(LU_INPUT,CTRL)

! Make sure there is either a FUNCTION_TYPE type for the CTRL

IF (FUNCTION_TYPE=='null') THEN
WRITE(MESSAGE,'(A,I0,A)') 'ERROR: CTRL ',NC,' must have a FUNCTION_TYPE'
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF
```

```
! Assign properties to the CONTROL array

CF => CONTROL(NC)
CF%CONSTANT            = CONSTANT
CF%ID                  = ID
CF%LATCH               = LATCH
CF%INITIAL_STATE       = INITIAL_STATE
CF%CURRENT_STATE       = INITIAL_STATE
CF%PRIOR_STATE         = INITIAL_STATE
CF%SETPOINT            = SETPOINT
CF%DELAY               = DELAY / TIME_SHRINK_FACTOR
CF%CYCLE_TIME          = CYCLE_TIME
CF%CYCLES              = CYCLES
CF%RAMP_ID             = RAMP_ID
CF%N                   = N
CF%INPUT_ID            = INPUT_ID
CF%EVACUATION          = EVACUATION
CF%TRIP_DIRECTION      = TRIP_DIRECTION
CF%PROPORTIONAL_GAIN =PROPORTIONAL_GAIN
CF%INTEGRAL_GAIN = INTEGRAL_GAIN
CF%DIFFERENTIAL_GAIN = DIFFERENTIAL_GAIN
CF%TARGET_VALUE = TARGET_VALUE
IF (ON_BOUND=='UPPER') THEN
CF%ON_BOUND = 1
ELSE
CF%ON_BOUND = -1
ENDIF
!Assign control index
SELECT CASE(FUNCTION_TYPE)
CASE('ALL')
CF%CONTROL_INDEX = AND_GATE
CASE('ANY')
CF%CONTROL_INDEX = OR_GATE
CASE('ONLY')
CF%CONTROL_INDEX = XOR_GATE
CASE('AT_LEAST')
CF%CONTROL_INDEX = X_OF_N_GATE
CASE('TIME_DELAY')
CF%CONTROL_INDEX = TIME_DELAY
CASE('DEADBAND')
CF%CONTROL_INDEX = DEADBAND
CASE('CYCLING')
CF%CONTROL_INDEX = CYCLING
CASE('CUSTOM')
CF%CONTROL_INDEX = CUSTOM
CALL GET_RAMP_INDEX(RAMP_ID,'CONTROL',CF%RAMP_INDEX)
CF%LATCH = .FALSE.
CASE('KILL')
CF%CONTROL_INDEX = KILL
CASE('RESTART')
CF%CONTROL_INDEX = CORE_DUMP
CASE('SUM')
CF%CONTROL_INDEX = CF_SUM
CASE('SUBTRACT')
CF%CONTROL_INDEX = CF_SUBTRACT
CASE('MULTIPLY')
CF%CONTROL_INDEX = CF_MULTIPLY
CASE('DIVIDE')
CF%CONTROL_INDEX = CF_DIVIDE
CASE('POWER')
CF%CONTROL_INDEX = CF_POWER
CASE('EXP')
CF%CONTROL_INDEX = CF_EXP
CASE('LOG')
CF%CONTROL_INDEX = CF_LOG
CASE('SIN')
CF%CONTROL_INDEX = CF_SIN
CASE('COS')
CF%CONTROL_INDEX = CF_COS
CASE('ASIN')
CF%CONTROL_INDEX = CF_ASIN
CASE('ACOS')
CF%CONTROL_INDEX = CF_ACOS
CASE('PID')
CF%CONTROL_INDEX = CF_PID
IF (CF%TARGET_VALUE<-1.E30_EB) THEN
WRITE(MESSAGE,'(A,I0,A)') 'ERROR: CTRL ',NC,', PID controller must be given a TARGET_VALUE'
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF
CASE DEFAULT
WRITE(MESSAGE,'(A,I0,A)') 'ERROR: CTRL ',NC,' FUNCTION_TYPE not recognized'
CALL SHUTDOWN(MESSAGE) ; RETURN
END SELECT

ENDDO READ_CTRL_LOOP
REWIND(LU_INPUT) ; INPUT_FILE_LINE_NUMBER = 0

```

```
CONTAINS

SUBROUTINE SET_CTRL_DEFAULTS
CONSTANT       = -9.E30_EB
ID             = 'null'
LATCH          = .TRUE.
INITIAL_STATE  = .FALSE.
SETPOINT       = 1.E30_EB
DELAY          = 0._EB
CYCLE_TIME     = 1000000._EB
CYCLES         = 1
FUNCTION_TYPE  = 'null'
RAMP_ID        = 'null'
INPUT_ID       = 'null'
ON_BOUND       = 'LOWER'
N              = 1
EVACUATION     = .FALSE.
TRIP_DIRECTION = 1
PROPORTIONAL_GAIN  = 1._EB
INTEGRAL_GAIN      = 0._EB
DIFFERENTIAL_GAIN  = 0._EB
TARGET_VALUE       = 0._EB

END SUBROUTINE SET_CTRL_DEFAULTS

END SUBROUTINE READ_CTRL


SUBROUTINE PROC_CTRL

! Process the CONTROL function parameters

USE CONTROL_VARIABLES
USE DEVICE_VARIABLES, ONLY: N_DEVC,DEVICE
LOGICAL :: CONSTANT_SPECIFIED,TSF_WARNING=.FALSE.
INTEGER :: NC,NN,NNN
TYPE (CONTROL_TYPE), POINTER :: CF=>NULL()

PROC_CTRL_LOOP: DO NC = 1, N_CTRL

CF => CONTROL(NC)
CONSTANT_SPECIFIED = .FALSE.
IF (CF%CONTROL_INDEX== TIME_DELAY) TSF_WARNING=.TRUE.
! setup input array

CF%N_INPUTS = 0
INPUT_COUNT: DO NN=1,40
IF (CF%INPUT_ID(NN)=='null') EXIT INPUT_COUNT
END DO INPUT_COUNT
CF%N_INPUTS=NN-1
IF (CF%N_INPUTS==0) THEN
WRITE(MESSAGE,'(A,I0,A)')  'ERROR: CTRL ',NC,' must have at least one input'
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF
SELECT CASE (CF%CONTROL_INDEX)
CASE (CF_SUBTRACT,CF_DIVIDE,CF_POWER)
IF (CF%N_INPUTS /= 2) THEN
WRITE(MESSAGE,'(A,I0,A)')  'ERROR: CTRL ',NC,' must have at only two inputs'
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF
CASE (CF_SUM,CF_MULTIPLY)
CASE DEFAULT
IF (ANY(CF%INPUT_ID=='CONSTANT')) THEN
WRITE(MESSAGE,'(A,I0,A)')  'ERROR: CTRL ',NC,' the INTPUT_ID of CONSTANT cannot be used'
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF
END SELECT
ALLOCATE (CF%INPUT(CF%N_INPUTS),STAT=IZERO)
CALL ChkMemErr('READ','CF%INPUT',IZERO)
ALLOCATE (CF%INPUT_TYPE(CF%N_INPUTS),STAT=IZERO)
CALL ChkMemErr('READ','CF%INPUT_TYPE',IZERO)
CF%INPUT_TYPE = -1

BUILD_INPUT: DO NN = 1, CF%N_INPUTS
IF (TRIM(CF%INPUT_ID(NN))==TRIM(CF%ID)) THEN
WRITE(MESSAGE,'(A,I0,A)')  'ERROR: CTRL ',NC,' cannot use a control function as an input to itself'
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF
IF (CF%INPUT_ID(NN)=='CONSTANT') THEN
IF (CONSTANT_SPECIFIED) THEN
WRITE(MESSAGE,'(A,I0,A)')  'ERROR: CTRL ',NC,' can only specify one input as a constant value'
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF
IF (CF%CONSTANT < -8.E30_EB) THEN
WRITE(MESSAGE,'(A,I0,A)')  'ERROR: CTRL ',NC,' has the INPUT_ID CONSTANT but no constant value was specified'
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF
CF%INPUT_TYPE(NN) = CONSTANT_INPUT
```

```
CONSTANT_SPECIFIED = .TRUE.
CYCLE BUILD_INPUT
ENDIF
CTRL_LOOP: DO NNN = 1, N_CTRL
IF (CONTROL(NNN)%ID == CP%INPUT_ID(NN)) THEN
CP%INPUT(NN) = NNN
CP%INPUT_TYPE(NN) = CONTROL_INPUT
IF (CP%CONTROL_INDEX == CUSTOM) THEN
WRITE(MESSAGE,'(A,I0,A)') 'ERROR: CUSTOM CTRL ',NC,' cannot have another CTRL as input'
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF
EXIT CTRL_LOOP
ENDIF
END DO CTRL_LOOP
DEVC_LOOP: DO NNN = 1, N_DEVC
IF (DEVICE(NNN)%ID == CP%INPUT_ID(NN)) THEN
IF (DEVICE(NNN)%OUTPUT_INDEX==41 .OR. DEVICE(NNN)%OUTPUT2_INDEX==41) TSF_WARNING=.TRUE.
IF (CP%INPUT_TYPE(NN) > 0) THEN
WRITE(MESSAGE,'(A,I0,A,I0,A)') 'ERROR: CTRL ',NC,' input ',NN,' is the ID for both a DEVC and a CTRL'
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF
CP%INPUT(NN) = NNN
CP%INPUT_TYPE(NN) = DEVICE_INPUT
EXIT DEVC_LOOP
ENDIF
END DO DEVC_LOOP
IF (CP%INPUT_TYPE(NN) > 0) CYCLE BUILD_INPUT
WRITE(MESSAGE,'(A,I0,A,A)') 'ERROR: CTRL ',NC,' cannot locate item for input ', TRIM(CP%INPUT_ID(NN))
CALL SHUTDOWN(MESSAGE) ; RETURN
IF (ALL(EVACUATION_ONLY)) CYCLE BUILD_INPUT
END DO BUILD_INPUT

END DO PROC_CTRL_LOOP

IF (ABS(TIME_SHRINK_FACTOR-1._EB)>TWO_EPSILON_EB .AND. TSF_WARNING) THEN
IF (MYID==0) WRITE(LU_ERR,'(A)') 'WARNING: One or more time based CTRL functions are being used with
    TIME_SHRINK_FACTOR'
ENDIF

END SUBROUTINE PROC_CTRL


SUBROUTINE PROC_OBST

USE GEOMETRY_FUNCTIONS, ONLY: BLOCK_CELL
INTEGER :: NM,N,I,J,K,IS,JS,KS,IC1,IC2

MESH_LOOP: DO NM=1,NMESHES

IF (PROCESS(NM)/=MYID)   CYCLE MESH_LOOP
IF (EVACUATION_ONLY(NM)) CYCLE MESH_LOOP

M=>MESHES(NM)
CALL POINT_TO_MESH(NM)

! Assign a property index to the obstruction for use in Smokeview

DO N=1,N_OBST
OB=>OBSTRUCTION(N)
IF (OB%PROP_ID /='null') THEN
CALL GET_PROPERTY_INDEX(OB%PROP_INDEX,'OBST',OB%PROP_ID)
ENDIF
ENDDO

! Make mesh edge cells not solid if cells on either side are not solid

DO K=0,KBP1,KBP1
IF (K==0) THEN ; KS=1 ; ELSE ; KS=-1 ; ENDIF
DO J=0,JBP1,JBP1
IF (J==0) THEN ; JS=1 ; ELSE ; JS=-1 ; ENDIF
DO I=1,IBAR
IC1 = CELL_INDEX(I,J+JS,K) ; IC2 = CELL_INDEX(I,J,K+KS)
IF (.NOT.SOLID(IC1) .AND. .NOT.SOLID(IC2)) CALL BLOCK_CELL(NM,I,I,J,J,K,K,0,0)
ENDDO
ENDDO
ENDDO

DO K=0,KBP1,KBP1
IF (K==0) THEN ; KS=1 ; ELSE ; KS=-1 ; ENDIF
DO I=0,IBP1,IBP1
IF (I==0) THEN ; IS=1 ; ELSE ; IS=-1 ; ENDIF
DO J=1,JBAR
IC1 = CELL_INDEX(I+IS,J,K) ; IC2 = CELL_INDEX(I,J,K+KS)
IF (.NOT.SOLID(IC1) .AND. .NOT.SOLID(IC2)) CALL BLOCK_CELL(NM,I,I,J,J,K,K,0,0)
ENDDO
ENDDO
ENDDO

```

```
DO J=0,JBP1,JBP1
IF (J==0) THEN ; JS=1 ; ELSE ; JS=-1 ; ENDIF
DO I=0,IBP1,IBP1
IF (I==0) THEN ; IS=1 ; ELSE ; IS=-1 ; ENDIF
DO K=1,KBAR
IC1 = CELL_INDEX(I+IS,J,K) ; IC2 = CELL_INDEX(I,J+JS,K)
IF (.NOT.SOLID(IC1) .AND. .NOT.SOLID(IC2)) CALL BLOCK_CELL(NM,I,I,J,J,K,K,0,0)
ENDDO
ENDDO
ENDDO

ENDDO MESH_LOOP


END SUBROUTINE PROC_OBST


SUBROUTINE PROC_DEVC(DT)

! Process the DEViCes

USE COMP_FUNCTIONS, ONLY : CHANGE_UNITS
USE CONTROL_VARIABLES
USE DEVICE_VARIABLES, ONLY : DEVICE_TYPE, DEVICE, N_DEVC, PROPERTY, PROPERTY_TYPE, MAX_PDPA_HISTOGRAM_NBINS,
      N_PDPA_HISTOGRAM

REAL(EB), INTENT(IN) :: DT
INTEGER  :: N,NN,NNN,NM,QUANTITY_INDEX,QUANTITY2_INDEX,MAXCELLS,I,J,K
REAL(EB) :: XX,YY,ZZ,XX1,YY1,ZZ1,DISTANCE,SCANDISTANCE,DX,DY,DZ
TYPE (DEVICE_TYPE), POINTER :: DV=>NULL()

IF (N_DEVC==0) RETURN

! Set initial values for DEViCes

DEVICE(1:N_DEVC)%VALUE = 0._EB
DEVICE(1:N_DEVC)%TIME_INTERVAL = 0._EB

PROC_DEVC_LOOP: DO N=1,N_DEVC

DV => DEVICE(N)

! Check for HVAC outputs with no HVAC inputs

IF ((DV%DUCT_ID/='null' .OR. DV%NODE_ID(1)/='null') .AND. .NOT. HVAC_SOLVE) THEN
WRITE(MESSAGE,'(A)') 'ERROR: HVAC outputs specified with no HVAC inputs'
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF

! If the Device has a SURF_ID, get the SURF_INDEX

IF (DV%SURF_ID/='null') THEN
DO NN=1,N_SURF
IF (SURFACE(NN)%ID==DV%SURF_ID) DV%SURF_INDEX = NN
ENDDO
ENDIF

! Check if the device PROPERTY exists and is appropriate

DV%PROP_INDEX = 0
IF (DV%PROP_ID /='null') THEN
CALL GET_PROPERTY_INDEX(DV%PROP_INDEX,'DEVC',DV%PROP_ID)
IF (DV%QUANTITY=='null' .AND. PROPERTY(DV%PROP_INDEX)%QUANTITY=='null') THEN
WRITE(MESSAGE,'(5A)') 'ERROR: DEVC ',TRIM(DV%ID),' or DEVC PROPerty ',TRIM(DV%PROP_ID),' must have a QUANTITY'
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF
IF (DV%QUANTITY=='null' .AND. PROPERTY(DV%PROP_INDEX)%QUANTITY/='null') DV%QUANTITY = PROPERTY(DV%PROP_INDEX)%
      QUANTITY
ENDIF

! Check if the output QUANTITY exists and is appropriate

QUANTITY_IF: IF (DV%QUANTITY /= 'null') THEN
CALL GET_QUANTITY_INDEX(DV%SMOKEVIEW_LABEL,DV%SMOKEVIEW_BAR_LABEL,QUANTITY_INDEX,I_DUM(1), &
DV%Y_INDEX,DV%Z_INDEX,DV%PART_INDEX,DV%DUCT_INDEX,DV%NODE_INDEX(1),DV%REAC_INDEX,'DEVC', &
DV%QUANTITY,'null',DV%SPEC_ID,DV%PART_ID,DV%DUCT_ID,DV%NODE_ID(1),DV%REAC_ID)

IF (DV%QUANTITY=='CONTROL' .OR. DV%QUANTITY=='CONTROL VALUE') UPDATE_DEVICES_AGAIN = .TRUE.

IF (OUTPUT_QUANTITY(QUANTITY_INDEX)%INTEGRATED .AND. DV%X1<=-1.E6_EB) THEN
WRITE(MESSAGE,'(3A)') 'ERROR: DEVC QUANTITY ',TRIM(DV%QUANTITY),' requires coordinates using XB'
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF

IF (QUANTITY_INDEX<0 .AND. DV%IOR==0 .AND. (DV%STATISTICS=='null'.OR.DV%LINE>0) .AND. DV%INIT_ID=='null') THEN
WRITE(MESSAGE,'(A,A,A)') 'ERROR: Specify orientation of DEVC ',TRIM(DV%ID),' using the parameter IOR'
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF
```

```

IF (QUANTITY_INDEX < 0 .AND. (DV%STATISTICS=='MASS MEAN' .OR. DV%STATISTICS=='VOLUME MEAN' .OR. &
DV%STATISTICS=='VOLUME INTEGRAL' .OR. DV%STATISTICS=='MASS INTEGRAL' .OR. &
DV%STATISTICS=='AREA INTEGRAL' .OR. DV%STATISTICS=='MASS' .OR. &
DV%STATISTICS=='VOLUME')) THEN
WRITE(MESSAGE,'(A,A)') 'ERROR: Invalid STATISTICS specified for wall DEVC ',TRIM(DV%ID)
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF

IF (QUANTITY_INDEX > 0 .AND. DV%STATISTICS=='SURFACE INTEGRAL' .OR. DV%STATISTICS=='SURFACE AREA') THEN
WRITE(MESSAGE,'(A,A)') 'ERROR: Invalid STATISTICS specified for gas DEVC ',TRIM(DV%ID)
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF

IF (QUANTITY_INDEX > 0              .AND. &
DV%STATISTICS/='null'             .AND. &
DV%STATISTICS/='TIME INTEGRAL'    .AND. &
DV%STATISTICS/='RMS'              .AND. &
DV%STATISTICS/='COV'              .AND. &
DV%STATISTICS/='CORRCOEF'         .AND. &
DV%STATISTICS/='TIME MIN'         .AND. &
DV%STATISTICS/='TIME MAX'         .AND. DV%I1<0) THEN
WRITE(MESSAGE,'(A,A)') 'ERROR: XB required when geometrical STATISTICS specified for gas DEVC ',TRIM(DV%ID)
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF

IF (TRIM(DV%QUANTITY)=='NODE PRESSURE DIFFERENCE') THEN
CALL GET_QUANTITY_INDEX(DV%SMOKEVIEW_LABEL,DV%SMOKEVIEW_BAR_LABEL,QUANTITY_INDEX,I_DUM(1), &
DV%Y_INDEX,DV%Z_INDEX,DV%PART_INDEX,DV%DUCT_INDEX,DV%NODE_INDEX(2),I_DUM(2),'DEVC', &
DV%QUANTITY,'null',DV%SPEC_ID,DV%PART_ID,DV%DUCT_ID,DV%NODE_ID(2),'null')
IF (DV%NODE_INDEX(1)==DV%NODE_INDEX(2)) THEN
WRITE(MESSAGE,'(A,A)') 'ERROR: NODE PRESSURE DIFFERENCE node 1 = node 2 ',TRIM(DV%ID)
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF
ENDIF

IF (OUTPUT_QUANTITY(QUANTITY_INDEX)%INTEGRATED_PARTICLES) DEVC_PARTICLE_FLUX = .TRUE.

IF (ANY(ABS(DV%GHOST_CELL_IOR)>0)) THEN
IF (DV%STATISTICS/='null') THEN
WRITE(MESSAGE,'(A,A)') 'ERROR: Statistics not appropriate for GHOST_CELL_IOR DEVC ',TRIM(DV%ID)
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF
ENDIF

ENDIF QUANTITY_IF

! Even if the device is not in a mesh that is handled by the current MPI process, assign its unit.

DV%OUTPUT_INDEX      = QUANTITY_INDEX
DV%QUANTITY          = OUTPUT_QUANTITY(QUANTITY_INDEX)%NAME
IF (DV%UNITS=='null') DV%UNITS = OUTPUT_QUANTITY(DV%OUTPUT_INDEX)%UNITS

! Only process the device if it belongs to the current MPI process.

! IF (PROCESS(DV%MESH)/=MYID) CYCLE PROC_DEVC_LOOP

! Assign properties to the DEVICE array

M => MESHES(DV%MESH)

DV%T_CHANGE          = 1.E7_EB
DV%I                 = MAX( 1 , MIN( M%IBAR , FLOOR(GINV(DV%X-M%XS,1,DV%MESH)*M%RDXI)   +1 ) )
DV%J                 = MAX( 1 , MIN( M%JBAR , FLOOR(GINV(DV%Y-M%YS,2,DV%MESH)*M%RDETA)  +1 ) )
DV%K                 = MAX( 1 , MIN( M%KBAR , FLOOR(GINV(DV%Z-M%ZS,3,DV%MESH)*M%RDZETA)+1 ) )
DV%CTRL_INDEX        = 0
DV%T                 = T_BEGIN
DV%TMP_L             = TMPA
DV%TI_VALUE          = 0._EB
DV%TI_T              = 0._EB

! COVariance and CORRelation COEFficient STATISTICS requiring QUANTITY2

QUANTITY2_IF: IF (TRIM(DV%QUANTITY2)/='null') THEN
IF (TRIM(DV%STATISTICS)/='COV' .AND. TRIM(DV%STATISTICS)/='CORRCOEF') THEN
WRITE(MESSAGE,'(A,A)') 'ERROR: QUANTITY2 only applicable to COV and CORRCOEF STATISTICS for DEVC ',TRIM(DV%ID)
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF
IF (DV%RELATIVE) THEN
WRITE(MESSAGE,'(A,A)') 'ERROR: RELATIVE not applicable for COV and CORRCOEF STATISTICS for DEVC ',TRIM(DV%ID)
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF
DV%RELATIVE=.FALSE.
CALL GET_QUANTITY_INDEX(DV%SMOKEVIEW_LABEL,DV%SMOKEVIEW_BAR_LABEL,QUANTITY2_INDEX,I_DUM(1), &
DV%Y_INDEX,DV%Z_INDEX,DV%PART_INDEX,DV%DUCT_INDEX,DV%NODE_INDEX(1),DV%REAC_INDEX,'DEVC', &
DV%QUANTITY2,'null',DV%SPEC_ID,DV%PART_ID,DV%DUCT_ID,DV%NODE_ID(1),DV%REAC_ID)
DV%OUTPUT2_INDEX = QUANTITY2_INDEX
DV%QUANTITY2      = OUTPUT_QUANTITY(QUANTITY2_INDEX)%NAME
```

```
DV%SMOKEVIEW_LABEL = TRIM(DV%QUANTITY)//' '//TRIM(DV%QUANTITY2)//' '//TRIM(DV%STATISTICS)
DV%SMOKEVIEW_BAR_LABEL = TRIM(OUTPUT_QUANTITY(DV%OUTPUT_INDEX)%SHORT_NAME)//' '// &
TRIM(OUTPUT_QUANTITY(DV%OUTPUT2_INDEX)%SHORT_NAME)//' '//TRIM(DV%STATISTICS)
SELECT CASE(TRIM(DV%STATISTICS))
CASE('COV')
DV%UNITS = TRIM(OUTPUT_QUANTITY(DV%OUTPUT_INDEX)%UNITS)//'*'//TRIM(OUTPUT_QUANTITY(DV%OUTPUT2_INDEX)%UNITS)
CASE('CORRCOEF')
DV%UNITS = ''
END SELECT
ENDIF QUANTITY2_IF

! Initialize histogram

IF(PROPERTY(DV%PROP_INDEX)%PDPA_HISTOGRAM) THEN
ALLOCATE(DV%PDPA_HISTOGRAM_COUNTS(PROPERTY(DV%PROP_INDEX)%PDPA_HISTOGRAM_NBINS))
DV%PDPA_HISTOGRAM_COUNTS(:)=0._EB
N_PDPA_HISTOGRAM=N_PDPA_HISTOGRAM+1
MAX_PDPA_HISTOGRAM_NBINS =MAX(MAX_PDPA_HISTOGRAM_NBINS,PROPERTY(DV%PROP_INDEX)%PDPA_HISTOGRAM_NBINS)
ENDIF

! Do initialization of special models

SPECIAL_QUANTITIES: SELECT CASE (DV%QUANTITY)

CASE ('CHAMBER OBSCURATION')

IF (DV%PROP_INDEX<1) THEN
WRITE(MESSAGE,'(A,A,A)') 'ERROR: DEVC ',TRIM(DV%ID),' is a smoke detector and must have a PROP_ID'
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF
IF (PROPERTY(DV%PROP_INDEX)%Y_INDEX<0 .AND. PROPERTY(DV%PROP_INDEX)%Z_INDEX<0) THEN
IF (SOOT_INDEX<1) THEN
WRITE(MESSAGE,'(A,A,A)') 'ERROR: DEVC ',TRIM(DV%ID),' is a smoke detector and requires a smoke source'
CALL SHUTDOWN(MESSAGE) ; RETURN
ELSE
PROPERTY(DV%PROP_INDEX)%Y_INDEX = SOOT_INDEX
ENDIF
ENDIF
ALLOCATE(DV%T_E(-1:1000))
ALLOCATE(DV%Y_E(-1:1000))
DV%T_E        = T_BEGIN - DT
DV%Y_E        = 0._EB
DV%N_T_E      = -1
DV%Y_C        = 0._EB
DV%SETPOINT = PROPERTY(DV%PROP_INDEX)%ACTIVATION_OBSCURATION
IF (PROPERTY(DV%PROP_INDEX)%Y_INDEX>0) DV%Y_INDEX = PROPERTY(DV%PROP_INDEX)%Y_INDEX
IF (PROPERTY(DV%PROP_INDEX)%Z_INDEX>0) DV%Z_INDEX = PROPERTY(DV%PROP_INDEX)%Z_INDEX

CASE ('LINK TEMPERATURE','SPRINKLER LINK TEMPERATURE')

IF (DV%PROP_INDEX<1) THEN
WRITE(MESSAGE,'(A,A,A)') 'ERROR: DEVC ',TRIM(DV%ID),' must have a PROP_ID'
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF
IF (PROPERTY(DV%PROP_INDEX)%ACTIVATION_TEMPERATURE <= -273.15_EB) THEN
WRITE(MESSAGE,'(A,A)') 'ERROR: ACTIVATION_TEMPERATURE needed for PROP ',TRIM(DV%PROP_ID)
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF

DV%SETPOINT = PROPERTY(DV%PROP_INDEX)%ACTIVATION_TEMPERATURE
DV%TMP_L      = PROPERTY(DV%PROP_INDEX)%INITIAL_TEMPERATURE

CASE ('THERMOCOUPLE')

IF (DV%STATISTICS=='MAX'.OR.DV%STATISTICS=='MIN'.OR.DV%STATISTICS=='MASS MEAN'.OR.DV%STATISTICS=='VOLUME MEAN'.OR.&
DV%STATISTICS=='MASS INTEGRAL'.OR.DV%STATISTICS=='VOLUME INTEGRAL'.OR.DV%STATISTICS=='MEAN'.OR.&
DV%STATISTICS=='AREA INTEGRAL') THEN
WRITE(MESSAGE,'(A,A,A)') 'ERROR: DEVC ',TRIM(DV%ID),' cannot use volume STATISTICS'
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF
DV%TMP_L = PROPERTY(DV%PROP_INDEX)%INITIAL_TEMPERATURE

CASE ('SOLID DENSITY')

IF (DV%MATL_ID=='null') THEN
WRITE(MESSAGE,'(A,A,A)') 'ERROR: DEVC ',TRIM(DV%ID),' must have a MATL_ID'
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF

CASE ('LAYER HEIGHT','UPPER TEMPERATURE','LOWER TEMPERATURE')

DV%K1 = MAX(1        ,DV%K1)
DV%K2 = MIN(M%KBAR,DV%K2)

CASE ('TRANSMISSION','PATH OBSCURATION')

IF (DV%PROP_INDEX>0) THEN
```

```
IF (PROPERTY(DV%PROP_INDEX)%Y_INDEX<1 .AND. PROPERTY(DV%PROP_INDEX)%Z_INDEX<1) THEN
IF (SOOT_INDEX<1) THEN
WRITE(MESSAGE,'(A,A,A)') 'ERROR: DEVC ',TRIM(DV%ID),' is a smoke detector and requires a smoke source'
CALL SHUTDOWN(MESSAGE) ; RETURN
ELSE
PROPERTY(DV%PROP_INDEX)%Y_INDEX = SOOT_INDEX
ENDIF
ENDIF
ELSE
IF (SOOT_INDEX <=0) THEN
WRITE(MESSAGE,'(A,A,A)') 'ERROR: DEVC ',TRIM(DV%ID),' is a smoke detector and requires a smoke source'
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF
ENDIF
IF (PROPERTY(DV%PROP_INDEX)%Y_INDEX>0) DV%Y_INDEX = PROPERTY(DV%PROP_INDEX)%Y_INDEX
IF (PROPERTY(DV%PROP_INDEX)%Z_INDEX>0) DV%Z_INDEX = PROPERTY(DV%PROP_INDEX)%Z_INDEX
NM = DV%MESH
M=>MESHES(NM)
DISTANCE = SQRT((DV%X1-DV%X2)**2 + (DV%Y1-DV%Y2)**2 + (DV%Z1-DV%Z2)**2)
SCANDISTANCE = 0.0001_EB * DISTANCE
DX = (DV%X2-DV%X1) * 0.0001_EB
DY = (DV%Y2-DV%Y1) * 0.0001_EB
DZ = (DV%Z2-DV%Z1) * 0.0001_EB
XX = DV%X1
YY = DV%Y1
ZZ = DV%Z1
MAXCELLS = 2*MAX(M%IBAR,M%JBAR,M%KBAR)
ALLOCATE(DV%I_PATH(MAXCELLS))
ALLOCATE(DV%J_PATH(MAXCELLS))
ALLOCATE(DV%K_PATH(MAXCELLS))
ALLOCATE(DV%D_PATH(MAXCELLS))
DV%D_PATH    = 0._EB
DV%I_PATH = MIN(M%IBAR , INT(GINV(DV%X1-M%XS,1,NM)*M%RDXI)   + 1)
DV%J_PATH = MIN(M%JBAR , INT(GINV(DV%Y1-M%YS,2,NM)*M%RDETA)  + 1)
DV%K_PATH = MIN(M%KBAR , INT(GINV(DV%Z1-M%ZS,3,NM)*M%RDZETA) + 1)
DV%N_PATH = 1
NN = 1
DO NNN=1,10000
XX = XX + DX
I = MIN(M%IBAR , INT(GINV(XX-M%XS,1,NM)*M%RDXI)   + 1)
YY = YY + DY
J = MIN(M%JBAR , INT(GINV(YY-M%YS,2,NM)*M%RDETA)  + 1)
ZZ = ZZ + DZ
K = MIN(M%KBAR , INT(GINV(ZZ-M%ZS,3,NM)*M%RDZETA) + 1)
IF (I==DV%I_PATH(NN) .AND. J==DV%J_PATH(NN) .AND. K==DV%K_PATH(NN)) THEN
DV%D_PATH(NN) = DV%D_PATH(NN) + SCANDISTANCE
ELSE
NN = NN + 1
DV%I_PATH(NN) = I
DV%J_PATH(NN) = J
DV%K_PATH(NN) = K
XX1 = DX
YY1 = DY
ZZ1 = DZ
IF (PROCESS(DV%MESH)==MYID) THEN
IF (I<DV%I_PATH(NN-1)) XX1 = XX-M%X(DV%I_PATH(NN-1)-1)
IF (I>DV%I_PATH(NN-1)) XX1 = XX-M%X(DV%I_PATH(NN-1))
IF (J<DV%J_PATH(NN-1)) YY1 = YY-M%Y(DV%J_PATH(NN-1)-1)
IF (J>DV%J_PATH(NN-1)) YY1 = YY-M%Y(DV%J_PATH(NN-1))
IF (K<DV%K_PATH(NN-1)) ZZ1 = ZZ-M%Z(DV%K_PATH(NN-1)-1)
IF (K>DV%K_PATH(NN-1)) ZZ1 = ZZ-M%Z(DV%K_PATH(NN-1))
ENDIF
DV%D_PATH(NN)    = SCANDISTANCE - SQRT(XX1**2+YY1**2+ZZ1**2)
DV%D_PATH(NN-1) = DV%D_PATH(NN-1) + SCANDISTANCE - DV%D_PATH(NN)
ENDIF
ENDDO
DV%N_PATH = NN

CASE ('CONTROL')

DO NN=1,N_CTRL
IF (CONTROL(NN)%ID==DV%CTRL_ID) DV%CTRL_INDEX = NN
ENDDO
IF (DV%CTRL_ID/='null' .AND. DV%CTRL_INDEX<=0) THEN
WRITE(MESSAGE,'(A,A,A)')  'ERROR: CONTROL ',TRIM(DV%CTRL_ID),' does not exist'
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF
DV%SETPOINT = 0.5
DV%TRIP_DIRECTION = 1

CASE ('CONTROL VALUE')

DO NN=1,N_CTRL
IF (CONTROL(NN)%ID==DV%CTRL_ID) DV%CTRL_INDEX = NN
ENDDO
IF (DV%CTRL_ID/='null' .AND. DV%CTRL_INDEX<=0) THEN
WRITE(MESSAGE,'(A,A,A)')  'ERROR: CONTROL ',TRIM(DV%CTRL_ID),' does not exist'
CALL SHUTDOWN(MESSAGE) ; RETURN
```

```
ENDIF

CASE ('ASPIRATION')

! Check either for a specified SMOKE SPECies, or if simple chemistry model is being used
IF (DV%PROP_INDEX>0) THEN
IF (PROPERTY(DV%PROP_INDEX)%Y_INDEX<1 .AND. PROPERTY(DV%PROP_INDEX)%Z_INDEX<1 .AND. SOOT_INDEX<1) THEN
WRITE(MESSAGE,'(A,A,A)') 'ERROR: DEVC ',TRIM(DV%ID),' is a smoke detector and requires a smoke source'
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF
ENDIF
! Count number of inputs for detector and verify that input is DENSITY with a specified SPEC_ID for smoke
NNN = 0
DO NN=1,N_DEVC
IF (DEVICE(NN)%DEVC_ID==DV%ID) THEN
IF (DEVICE(NN)%QUANTITY/='DENSITY' .OR. DEVICE(NN)%SPEC_ID=='null') THEN
WRITE(MESSAGE,'(A,A,A)') &
'ERROR: DEVICE ',TRIM(DEVICE(NN)%ID)," must use QUANTITY='DENSITY' and a SPEC_ID"
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF
NNN = NNN + 1
ENDIF
ENDDO
ALLOCATE(DV%DEVC_INDEX(NNN),STAT=IZERO)
CALL ChkMemErr('READ','DV%DEVC_INDEX',IZERO)
DV%DEVC_INDEX = -1
ALLOCATE(DV%YY_SOOT(NNN,0:100))
CALL ChkMemErr('READ','DV%YY_SOOT',IZERO)
DV%YY_SOOT = 0._EB
ALLOCATE(DV%TIME_ARRAY(0:100))
CALL ChkMemErr('READ','DV%TIME_ARRAY',IZERO)
DV%TIME_ARRAY = 0._EB
DV%TOTAL_FLOWRATE = DV%BYPASS_FLOWRATE
DV%DT            = -1._EB
DV%N_INPUTS = NNN
NNN = 1
DO NN=1,N_DEVC
IF (DEVICE(NN)%DEVC_ID==DV%ID) THEN
DV%TOTAL_FLOWRATE  = DV%TOTAL_FLOWRATE + DEVICE(NN)%FLOWRATE
DV%DT = MAX(DV%DT,DEVICE(NN)%DELAY)
IF (NN > N) THEN
WRITE(MESSAGE,'(A,A,A)') 'ERROR: ASPIRATION DEVICE ',TRIM(DV%ID),' is not listed after all its inputs'
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF
DV%DEVC_INDEX(NNN)       = NN
NNN = NNN + 1
ENDIF
ENDDO
DV%DT = DV%DT * 0.01_EB

CASE ('FED')
IF (DV%STATISTICS /= 'null' .AND. DV%STATISTICS /= 'TIME INTEGRAL') THEN
WRITE(MESSAGE,'(A)') 'ERROR: Only TIME INTEGRAL statistics can be used with FED devices'
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF
DV%STATISTICS = 'TIME INTEGRAL'
IF (DV%PROP_INDEX>0) THEN
IF (PROPERTY(DV%PROP_INDEX)%FED_ACTIVITY<1 .AND. PROPERTY(DV%PROP_INDEX)%FED_ACTIVITY>3) THEN
WRITE(MESSAGE,'(A,A,A)') 'ERROR: DEVC ',TRIM(DV%ID),' is a FED detector and requires an activity'
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF
DV%FED_ACTIVITY = PROPERTY(DV%PROP_INDEX)%FED_ACTIVITY
ENDIF

CASE ('VELOCITY PATCH')
PATCH_VELOCITY = .TRUE.
ALLOCATE(DV%DEVC_INDEX(1),STAT=IZERO)
DV%DEVC_INDEX(1) = 0
DO NN=1,N_DEVC
IF (DEVICE(NN)%ID==DV%DEVC_ID) DV%DEVC_INDEX(1) = NN
ENDDO
IF (DV%DEVC_INDEX(1)==0) THEN
WRITE(MESSAGE,'(A)') 'ERROR: A VELOCITY PATCH DEVC line needs a DEVC_ID to control it'
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF

END SELECT SPECIAL_QUANTITIES

IF (DV%STATISTICS/='null') CALL CHANGE_UNITS(DV%QUANTITY,DV%UNITS,DV%STATISTICS,MYID,LU_ERR)

IF (DV%LINE == 0 .AND. (DV%STATISTICS=='RMS' .OR. DV%STATISTICS=='COV' .OR. DV%STATISTICS=='CORRCOEF')) THEN
DV%TIME_AVERAGED=.FALSE.
IF (DV%STATISTICS_START < T_BEGIN) DV%STATISTICS_START = T_BEGIN
ENDIF

IF (DV%NO_UPDATE_DEVC_ID/='null' .OR. DV%NO_UPDATE_CTRL_ID/='null') &
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)
```

```fortran

ENDDO PROC_DEVC_LOOP

END SUBROUTINE PROC_DEVC


SUBROUTINE READ_PROF

INTEGER :: N,NM,MESH_NUMBER,NN,N_PROFO,IOR,FORMAT_INDEX
REAL(EB) :: XYZ(3)
CHARACTER(LABEL_LENGTH) :: QUANTITY
TYPE (PROFILE_TYPE), POINTER :: PF=>NULL()
NAMELIST /PROF/ FORMAT_INDEX,FYI,ID,IOR,QUANTITY,XYZ

N_PROF = 0
REWIND(LU_INPUT) ; INPUT_FILE_LINE_NUMBER = 0
COUNT_PROF_LOOP: DO
CALL CHECKREAD('PROF',LU_INPUT,IOS)
IF (IOS==1) EXIT COUNT_PROF_LOOP
READ(LU_INPUT,NML=PROF,END=11,ERR=12,IOSTAT=IOS)
N_PROF = N_PROF + 1
12 IF (IOS>0) THEN
WRITE(MESSAGE,'(A,I0,A,I0)') 'ERROR: Problem with PROF number ',N_PROF+1,', line number',INPUT_FILE_LINE_NUMBER
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF
ENDDO COUNT_PROF_LOOP
11 REWIND(LU_INPUT) ; INPUT_FILE_LINE_NUMBER = 0

IF (N_PROF==0) RETURN

ALLOCATE(PROFILE(N_PROF),STAT=IZERO)
CALL ChkMemErr('READ','PROFILE',IZERO)

PROFILE(1:N_PROF)%QUANTITY = 'TEMPERATURE'
PROFILE(1:N_PROF)%IOR      = 0
PROFILE(1:N_PROF)%IW       = 0

N_PROFO = N_PROF
N        = 0

PROF_LOOP: DO NN=1,N_PROFO
N    = N+1
FORMAT_INDEX = 1
IOR  = 0
SELECT CASE(N)
CASE(1:9)
WRITE(ID,'(A,I1)') 'PROFILE ',N
CASE(10:99)
WRITE(ID,'(A,I2)') 'PROFILE ',N
CASE(100:999)
WRITE(ID,'(A,I3)') 'PROFILE ',N
END SELECT

CALL CHECKREAD('PROF',LU_INPUT,IOS)
IF (IOS==1) EXIT PROF_LOOP
READ(LU_INPUT,PROF)

! Check for bad PROF quantities or coordinates

IF (IOR==0) THEN
WRITE(MESSAGE,'(A,I0,A)') 'ERROR: Specify orientation of PROF ' ,NN,' using the parameter IOR'
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF

BAD = .FALSE.

MESH_LOOP: DO NM=1,NMESHES
IF (.NOT.EVACUATION_ONLY(NM)) THEN
M=>MESHES(NM)
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) THEN
MESH_NUMBER = NM
EXIT MESH_LOOP
ENDIF
ENDIF
IF (NM==NMESHES) BAD = .TRUE.
ENDDO MESH_LOOP

IF (BAD) THEN
N      = N-1
N_PROF = N_PROF-1
CYCLE PROF_LOOP
ENDIF

! Assign parameters to the PROFILE array

PF => PROFILE(N)
PF%FORMAT_INDEX = FORMAT_INDEX
```

```
PP%ORDINAL = NN
PP%MESH     = MESH_NUMBER
PP%ID    = ID
PP%QUANTITY = QUANTITY
PP%X        = XYZ(1)
PP%Y        = XYZ(2)
PP%Z        = XYZ(3)
PP%IOR      = IOR

ENDDO PROF_LOOP
REWIND(LU_INPUT) ; INPUT_FILE_LINE_NUMBER = 0

END SUBROUTINE READ_PROF



SUBROUTINE READ_ISOF

REAL(EB) :: VALUE(10)
CHARACTER(LABEL_LENGTH) :: QUANTITY,SPEC_ID
INTEGER :: N,VELO_INDEX
TYPE(ISOSURFACE_FILE_TYPE), POINTER :: IS=>NULL()
NAMELIST /ISOF/ FYI,QUANTITY,SPEC_ID,VALUE,VELO_INDEX

N_ISOF = 0
REWIND(LU_INPUT) ; INPUT_FILE_LINE_NUMBER = 0
COUNT_ISOF_LOOP: DO
CALL CHECKREAD('ISOF',LU_INPUT,IOS)
IF (IOS==1) EXIT COUNT_ISOF_LOOP
READ(LU_INPUT,NML=ISOF,END=9,ERR=10,IOSTAT=IOS)
N_ISOF = N_ISOF + 1
10 IF (IOS>0) THEN
WRITE(MESSAGE,'(A,I0,A,I0)') 'ERROR: Problem with ISOF number ',N_ISOF,', line number',INPUT_FILE_LINE_NUMBER
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF
ENDDO COUNT_ISOF_LOOP
9 REWIND(LU_INPUT) ; INPUT_FILE_LINE_NUMBER = 0

ALLOCATE(ISOSURFACE_FILE(N_ISOF),STAT=IZERO)
CALL ChkMemErr('READ','ISOSURFACE_FILE',IZERO)

READ_ISOF_LOOP: DO N=1,N_ISOF
IS => ISOSURFACE_FILE(N)
QUANTITY          = 'null'
SPEC_ID           = 'null'
VALUE             = -999._EB
VELO_INDEX        = 0

CALL CHECKREAD('ISOF',LU_INPUT,IOS)
IF (IOS==1) EXIT READ_ISOF_LOOP
READ(LU_INPUT,ISOF)

IS%VELO_INDEX         = VELO_INDEX

CALL GET_QUANTITY_INDEX(IS%SMOKEVIEW_LABEL,IS%SMOKEVIEW_BAR_LABEL,IS%INDEX,I_DUM(1), &
IS%Y_INDEX,IS%Z_INDEX,I_DUM(2),I_DUM(3),I_DUM(4),I_DUM(5),'ISOF', &
QUANTITY,'null',SPEC_ID,'null','null','null','null')

VALUE_LOOP: DO I=1,10
IF (VALUE(I)<=-998._EB) EXIT VALUE_LOOP
IS%N_VALUES = I
IS%VALUE(I) = REAL(VALUE(I),FB)
ENDDO VALUE_LOOP

ENDDO READ_ISOF_LOOP

REWIND(LU_INPUT) ; INPUT_FILE_LINE_NUMBER = 0

END SUBROUTINE READ_ISOF


SUBROUTINE READ_SLCF

REAL(EB) :: MAXIMUM_VALUE,MINIMUM_VALUE
REAL(EB) :: AGL_SLICE
INTEGER :: N,NN,NM,MESH_NUMBER,N_SLCF_O,NITER,ITER,VELO_INDEX,IOR
LOGICAL :: VECTOR,CELL_CENTERED, FACE_CENTERED, FIRE_LINE, EVACUATION,LEVEL_SET_FIRE_LINE
CHARACTER(LABEL_LENGTH) :: QUANTITY,SPEC_ID,PART_ID,QUANTITY2,PROP_ID,REAC_ID,SLICETYPE
REAL(EB), PARAMETER :: TOL=1.E-10_EB
REAL(EB) :: DELX, DELY, DELZ, SMV_OFFSET
TYPE (SLICE_TYPE), POINTER :: SL=>NULL()
NAMELIST /SLCF/ AGL_SLICE,CELL_CENTERED,EVACUATION,FACE_CENTERED,FIRE_LINE,FYI,ID,IOR,LEVEL_SET_FIRE_LINE,
      MAXIMUM_VALUE,&
MESH_NUMBER,MINIMUM_VALUE,PART_ID,PBX,PBY,PBZ,PROP_ID,QUANTITY,QUANTITY2,REAC_ID,SLICETYPE,SMV_OFFSET,SPEC_ID,&
VECTOR,VELO_INDEX,XB

MESH_LOOP: DO NM=1,NMESHES

```

```
IF (MYID/=PROCESS(NM)) CYCLE MESH_LOOP

M=>MESHES(NM)
CALL POINT_TO_MESH(NM)

N_SLCF    = 0
N_SLCF_O = 0
REWIND(LU_INPUT) ; INPUT_FILE_LINE_NUMBER = 0
COUNT_SLCF_LOOP: DO
VECTOR   = .FALSE.
EVACUATION   = .FALSE.
MESH_NUMBER=NM
CALL CHECKREAD('SLCF',LU_INPUT,IOS)
IF (IOS==1) EXIT COUNT_SLCF_LOOP
READ(LU_INPUT,NML=SLCF,END=9,ERR=10,IOSTAT=IOS)
N_SLCF_O = N_SLCF_O + 1
IF (MESH_NUMBER/=NM) CYCLE COUNT_SLCF_LOOP
IF (.NOT.EVACUATION_ONLY(NM) .AND.      EVACUATION) CYCLE COUNT_SLCF_LOOP
IF (     EVACUATION_ONLY(NM) .AND. .NOT.EVACUATION) CYCLE COUNT_SLCF_LOOP
IF (EVACUATION_ONLY(NM) .AND. .NOT.EVACUATION_SKIP(NM)) CYCLE COUNT_SLCF_LOOP
N_SLCF   = N_SLCF + 1
IF (VECTOR .AND. TWO_D) N_SLCF = N_SLCF + 2
IF (VECTOR .AND. .NOT. TWO_D) N_SLCF = N_SLCF + 3
10 IF (IOS>0) THEN
WRITE(MESSAGE,'(A,I0,A,I0)') 'ERROR: Problem with SLCF number ',N_SLCF_O+1,', line number',INPUT_FILE_LINE_NUMBER
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF
ENDDO COUNT_SLCF_LOOP
9 CONTINUE

ALLOCATE(M%SLICE(N_SLCF),STAT=IZERO)
CALL ChkMemErr('READ','ISP1',IZERO)
CALL POINT_TO_MESH(NM)  ! Reset the pointers after the allocation

N = 0
N_TERRAIN_SLCF = 0

REWIND(LU_INPUT) ; INPUT_FILE_LINE_NUMBER = 0
SLCF_LOOP: DO NN=1,N_SLCF_O
QUANTITY   = 'null'
QUANTITY2 = 'null'
SMV_OFFSET = 0.0_EB
PBX        = -1.E6_EB
PBY        = -1.E6_EB
PBZ        = -1.E6_EB
VECTOR     = .FALSE.
ID         = 'null'
MESH_NUMBER=NM
MINIMUM_VALUE = 0._EB
MAXIMUM_VALUE = 0._EB
AGL_SLICE = -1._EB
REAC_ID   = 'null'
SPEC_ID   = 'null'
PART_ID   = 'null'
PROP_ID   = 'null'
SLICETYPE = 'STRUCTURED'
IOR = 0
CELL_CENTERED = .FALSE.
FACE_CENTERED = .FALSE.
FIRE_LINE=.FALSE.
EVACUATION   = .FALSE.
VELO_INDEX = 0
LEVEL_SET_FIRE_LINE = .FALSE.

CALL CHECKREAD('SLCF',LU_INPUT,IOS)
IF (IOS==1) EXIT SLCF_LOOP
READ(LU_INPUT,SLCF)
IF (MESH_NUMBER/=NM) CYCLE SLCF_LOOP
IF (.NOT.EVACUATION_ONLY(NM) .AND.      EVACUATION) CYCLE SLCF_LOOP
IF (     EVACUATION_ONLY(NM) .AND. .NOT.EVACUATION) CYCLE SLCF_LOOP
IF (EVACUATION_ONLY(NM) .AND. .NOT.EVACUATION_SKIP(NM)) CYCLE SLCF_LOOP
IF (CELL_CENTERED .AND. FACE_CENTERED) FACE_CENTERED = .FALSE.

IF (PBX>-1.E5_EB .OR. PBY>-1.E5_EB .OR. PBZ>-1.E5_EB) THEN
XB(1) = XS
XB(2) = XF
XB(3) = YS
XB(4) = YF
XB(5) = ZS
XB(6) = ZF
IF (PBX>-1.E5_EB) XB(1:2) = PBX
IF (PBY>-1.E5_EB) XB(3:4) = PBY
IF (PBZ>-1.E5_EB) XB(5:6) = PBZ
ENDIF

CALL CHECK_XB(XB)

XB(1) = MAX(XB(1),XS)
```

```
XB(2) = MIN(XB(2),XF)
XB(3) = MAX(XB(3),YS)
XB(4) = MIN(XB(4),YF)
XB(5) = MAX(XB(5),ZS)
XB(6) = MIN(XB(6),ZF)
IF (IOR == 0) THEN ! determine slice orientation if not already specified
DELX = ABS(XB(1)-XB(2))
DELY = ABS(XB(3)-XB(4))
DELZ = ABS(XB(5)-XB(6))
IF( DELX < MIN(DELY,DELZ) )THEN
IOR = 1
ELSE IF( DELY < MIN(DELX,DELZ) )THEN
IOR = 2
ELSE
IOR = 3
ENDIF
ENDIF

! Reject a slice if it is beyond the bounds of the current mesh

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
N_SLCF = N_SLCF - 1
IF (VECTOR .AND. TWO_D) N_SLCF = N_SLCF - 2
IF (VECTOR .AND. .NOT. TWO_D) N_SLCF = N_SLCF - 3
CYCLE SLCF_LOOP
ENDIF

! Process vector quantities

NITER = 1
IF (VECTOR .AND. TWO_D) NITER = 3
IF (VECTOR .AND. .NOT. TWO_D) NITER = 4

VECTORLOOP: DO ITER=1,NITER
N = N + 1
SL=>SLICE(N)
SL%ID = ID
SL%SLICETYPE = TRIM(SLICETYPE)
SL%IOR = IOR
IF ((FACE_CENTERED .OR. CELL_CENTERED) .AND. NITER==1) THEN ! scalar raw data
DO I=1,IBAR
IF ( ABS(XB(1)-XC(I)) < 0.5_EB*DX(I) + TOL ) SL%I1 = I
IF ( ABS(XB(2)-XC(I)) < 0.5_EB*DX(I) + TOL ) SL%I2 = I
ENDDO
DO J=1,JBAR
IF ( ABS(XB(3)-YC(J)) < 0.5_EB*DY(J) + TOL ) SL%J1 = J
IF ( ABS(XB(4)-YC(J)) < 0.5_EB*DY(J) + TOL ) SL%J2 = J
ENDDO
DO K=1,KBAR
IF ( ABS(XB(5)-ZC(K)) < 0.5_EB*DZ(K) + TOL ) SL%K1 = K
IF ( ABS(XB(6)-ZC(K)) < 0.5_EB*DZ(K) + TOL ) SL%K2 = K
ENDDO
IF (SL%I1<SL%I2) SL%I1=SL%I1-1
IF (SL%J1<SL%J2) SL%J1=SL%J1-1
IF (SL%K1<SL%K2) SL%K1=SL%K1-1
ELSE
SL%I1 = NINT( GINV(XB(1)-XS,1,NM)*RDXI )
SL%I2 = NINT( GINV(XB(2)-XS,1,NM)*RDXI )
SL%J1 = NINT( GINV(XB(3)-YS,2,NM)*RDETA )
SL%J2 = NINT( GINV(XB(4)-YS,2,NM)*RDETA )
SL%K1 = NINT( GINV(XB(5)-ZS,3,NM)*RDZETA )
SL%K2 = NINT( GINV(XB(6)-ZS,3,NM)*RDZETA )
ENDIF
SL%MINMAX(1) = REAL(MINIMUM_VALUE,FB)
SL%MINMAX(2) = REAL(MAXIMUM_VALUE,FB)
IF (ITER==2)                    QUANTITY = 'U-VELOCITY'
IF (ITER==3 .AND. .NOT. TWO_D)  QUANTITY = 'V-VELOCITY'
IF (ITER==3 .AND. TWO_D)        QUANTITY = 'W-VELOCITY'
IF (ITER==4)                    QUANTITY = 'W-VELOCITY'
IF (ITER==1 .AND. FIRE_LINE)    QUANTITY = 'TEMPERATURE'
IF (ITER==1 .AND. LEVEL_SET_FIRE_LINE) QUANTITY = 'TEMPERATURE'
IF (ITER==1) THEN
SL%FIRE_LINE = FIRE_LINE
SL%LEVEL_SET_FIRE_LINE = LEVEL_SET_FIRE_LINE
IF (LEVEL_SET_FIRE_LINE .AND. .NOT. VEG_LEVEL_SET) THEN
WRITE(MESSAGE,'(A)') "ERROR: VEG_LEVEL_SET must be TRUE on MISC line to run the LS model"
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF
ELSE
SL%FIRE_LINE = .FALSE.
SL%LEVEL_SET_FIRE_LINE = .FALSE.
SPEC_ID        = 'null'
ENDIF
SL%VELO_INDEX = VELO_INDEX
CALL GET_QUANTITY_INDEX(SL%SMOKEVIEW_LABEL,SL%SMOKEVIEW_BAR_LABEL,SL%INDEX,SL%INDEX2, &
SL%Y_INDEX,SL%Z_INDEX,SL%PART_INDEX,I_DUM(1),I_DUM(2),SL%REAC_INDEX,'SLCF', &
QUANTITY,QUANTITY2,SPEC_ID,PART_ID,'null','null',REAC_ID)
```

```fortran
! If the user needs to do a particle flux calculation, detect that here.

IF (OUTPUT_QUANTITY(SL%INDEX)%INTEGRATED_PARTICLES) SLCF_PARTICLE_FLUX = .TRUE.

! For terrain slices, AGL=above ground level
! FIRE_LINE==.TRUE. means a terrain slice at one grid cell above ground with quantity temperature.
! Smokeview will display only regions where temperature is above 200 C. This is currently hard wired.
! LEVEL_SET_FIRE_LINE = .TRUE. will create a slice file !!! this is not fully functional !!!

IF (ITER == 1 .AND. (AGL_SLICE > -1._EB .OR. FIRE_LINE .OR. LEVEL_SET_FIRE_LINE ) ) THEN
SL%TERRAIN_SLICE = .TRUE.
IF (FIRE_LINE) THEN
SL%SMOKEVIEW_LABEL = "Fire line"
SL%SMOKEVIEW_BAR_LABEL = "Fire_line"
ENDIF
IF (LEVEL_SET_FIRE_LINE) THEN
SL%SMOKEVIEW_LABEL = "Level Set Fire line"
SL%SMOKEVIEW_BAR_LABEL = "LS_Fire_line"
ENDIF
IF (AGL_SLICE <= -1._EB .AND. FIRE_LINE)           AGL_SLICE = M%Z(1) - M%Z(0)
IF (AGL_SLICE <= -1._EB .AND. LEVEL_SET_FIRE_LINE) AGL_SLICE = 0._EB
SL%SLICE_AGL      = AGL_SLICE
N_TERRAIN_SLCF    = N_TERRAIN_SLCF + 1
ENDIF
IF (ITER==2 .OR. ITER==3 .OR. ITER ==4) THEN
IF (SLICE(N-1)%TERRAIN_SLICE) THEN
SL%TERRAIN_SLICE =  .TRUE.
SL%SLICE_AGL      = SLICE(N-1)%SLICE_AGL
N_TERRAIN_SLCF    = N_TERRAIN_SLCF + 1
ENDIF
ENDIF

! Disable cell centered for velocity

! IF (QUANTITY=='VELOCITY'   .OR. &
!     QUANTITY=='U-VELOCITY' .OR. &
!     QUANTITY=='V-VELOCITY' .OR. &
!     QUANTITY=='W-VELOCITY') THEN
!     CELL_CENTERED = .FALSE.
! ENDIF
SL%CELL_CENTERED = CELL_CENTERED
SL%FACE_CENTERED = FACE_CENTERED

! Check if the slcf PROPERTY exists (for FED_ACTIVITY input)

SL%PROP_INDEX = 0
IF (PROP_ID /='null') THEN
CALL GET_PROPERTY_INDEX(SL%PROP_INDEX,'SLCF',PROP_ID)
ENDIF

SL%SMV_OFFSET = SMV_OFFSET

ENDDO VECTORLOOP

IF (TRIM(QUANTITY)=='CHEMISTRY SUBITERATIONS') OUTPUT_CHEM_IT = .TRUE.

IF (TRIM(QUANTITY)=='REAC SOURCE TERM' .OR. TRIM(QUANTITY)=='HRRPUV REAC') REAC_SOURCE_CHECK = .TRUE.

IF (TRIM(QUANTITY)=='H PRIME' .AND. .NOT.EXTERNAL_BOUNDARY_CORRECTION) THEN
WRITE(MESSAGE,'(A,I0,A)') 'ERROR: Problem with SCLF ',NN,', H PRIME requires EXTERNAL_BOUNDARY_CORRECTION=T on
  MISC'
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF

IF (TRIM(QUANTITY)=='SOLID CELL Q_S') STORE_Q_DOT_PPP_S = .TRUE.

IF (TRIM(QUANTITY)=='SUBGRID TEMPERATURE CORRECTION' .AND. .NOT.CORRECT_SUBGRID_TEMPERATURE) THEN
WRITE(MESSAGE,'(A,I0,A)') 'ERROR: Promblem with SCLF ',NN,', requires CORRECT_SUBGRID_TEMPERATURE=T on MISC'
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF

IF (TRIM(QUANTITY)=='DUDT' .OR. TRIM(QUANTITY)=='DVDT' .OR. TRIM(QUANTITY)=='DWDT') STORE_OLD_VELOCITY=.TRUE.

ENDDO SLCF_LOOP

ALLOCATE(M%K_AGL_SLICE(0:IBP1,0:JBP1,N_TERRAIN_SLCF),STAT=IZERO)
CALL ChkMemErr('READ','K_AGL_SLICE',IZERO)
M%K_AGL_SLICE = 0
N = 0
DO NN = 1,N_SLCF
SL=>SLICE(NN)
IF(SL%TERRAIN_SLICE) THEN
TERRAIN_CASE = .TRUE.
N = N + 1
M%K_AGL_SLICE(0:IBP1,0:JBP1,N) =  INT(SL%SLICE_AGL*M%RDZ(1))
! Subtract one because bottom of domain will be accounted for when cycling through walls cells
M%K_AGL_SLICE(0:IBP1,0:JBP1,N) =  MAX(0,M%K_AGL_SLICE(0:IBP1,0:JBP1,N)-1)
ENDIF
```

```
ENDDO

N_SLCF_MAX = MAX(N_SLCF_MAX,N_SLCF)

IF(VEG_LEVEL_SET) THEN
ALLOCATE(M%LS_Z_TERRAIN(0:IBP1,0:JBP1),STAT=IZERO) ; CALL ChkMemErr('READ','LS_Z_TERRAIN',IZERO)
ENDIF

ENDDO MESH_LOOP

END SUBROUTINE READ_SLCF


SUBROUTINE READ_BNDF

USE DEVICE_VARIABLES
USE COMP_FUNCTIONS, ONLY : CHANGE_UNITS
INTEGER :: N
LOGICAL :: CELL_CENTERED
CHARACTER(LABEL_LENGTH) :: QUANTITY,PROP_ID,SPEC_ID,PART_ID,STATISTICS
NAMELIST /BNDF/ CELL_CENTERED,FYI,PART_ID,PROP_ID,QUANTITY,SPEC_ID,STATISTICS
TYPE(BOUNDARY_FILE_TYPE), POINTER :: BF=>NULL()

N_BNDF = 0
REWIND(LU_INPUT) ; INPUT_FILE_LINE_NUMBER = 0
COUNT_BNDF_LOOP: DO
CALL CHECKREAD('BNDF',LU_INPUT,IOS)
IF (IOS==1) EXIT COUNT_BNDF_LOOP
READ(LU_INPUT,NML=BNDF,END=209,ERR=210,IOSTAT=IOS)
N_BNDF = N_BNDF + 1
210 IF (IOS>0) THEN
WRITE(MESSAGE,'(A,I0,A,I0)') 'ERROR: Problem with BNDF number ',N_BNDF+1,', line number',INPUT_FILE_LINE_NUMBER
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF
ENDDO COUNT_BNDF_LOOP
209 REWIND(LU_INPUT) ; INPUT_FILE_LINE_NUMBER = 0

ALLOCATE(BOUNDARY_FILE(N_BNDF),STAT=IZERO)
CALL ChkMemErr('READ','BOUNDARY_FILE',IZERO)

BNDF_TIME_INTEGRALS = 0

READ_BNDF_LOOP: DO N=1,N_BNDF
BF => BOUNDARY_FILE(N)
CELL_CENTERED = .FALSE.
PART_ID  = 'null'
PROP_ID  = 'null'
SPEC_ID  = 'null'
STATISTICS = 'null'
QUANTITY = 'WALL_TEMPERATURE'
CALL CHECKREAD('BNDF',LU_INPUT,IOS)
IF (IOS==1) EXIT READ_BNDF_LOOP
READ(LU_INPUT,BNDF)

IF (TRIM(QUANTITY)=='AMPUA_Z' .OR. TRIM(QUANTITY)=='CPUA_Z' .OR. TRIM(QUANTITY)=='MPUA_Z') THEN
IF (N_LP_ARRAY_INDICES == 0) THEN
WRITE(MESSAGE,'(A,I0)') 'ERROR: CPUA_Z, MPUA_Z, and AMPUA_Z require liquid droplets. BNDF line ',N
CALL SHUTDOWN(MESSAGE) ; RETURN
ELSE
IF (.NOT. ALL(LAGRANGIAN_PARTICLE_CLASS%LIQUID_DROPLET)) THEN
WRITE(MESSAGE,'(A,I0)') 'ERROR: CPUA_Z, MPUA_Z, and AMPUA_Z require liquid droplets. BNDF line ',N
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF
ENDIF
ENDIF

! Look to see if output QUANTITY exists

CALL GET_QUANTITY_INDEX(BF%SMOKEVIEW_LABEL,BF%SMOKEVIEW_BAR_LABEL,BF%INDEX,I_DUM(1), &
BF%Y_INDEX,BF%Z_INDEX,BF%PART_INDEX,I_DUM(2),I_DUM(3),I_DUM(4),'BNDF', &
QUANTITY,'null',SPEC_ID,PART_ID,'null','null','null')

BF%UNITS = OUTPUT_QUANTITY(BF%INDEX)%UNITS

! Assign miscellaneous attributes to the boundary file

BF%CELL_CENTERED = CELL_CENTERED

! Check to see if PROP_ID exists

BF%PROP_INDEX = 0
IF(PROP_ID/='null')  CALL GET_PROPERTY_INDEX(BF%PROP_INDEX,'BNDF',PROP_ID)

! Check to see if the QUANTITY is to be time integrated

IF (STATISTICS=='TIME INTEGRAL') THEN
BNDF_TIME_INTEGRALS = BNDF_TIME_INTEGRALS + 1
BF%TIME_INTEGRAL_INDEX = BNDF_TIME_INTEGRALS
```

```fortran
CALL CHANGE_UNITS(QUANTITY,BF%UNITS,STATISTICS,MYID,LU_ERR)
ENDIF

ENDDO READ_BNDF_LOOP
REWIND(LU_INPUT) ; INPUT_FILE_LINE_NUMBER = 0

END SUBROUTINE READ_BNDF


SUBROUTINE READ_BNDE

USE DEVICE_VARIABLES
INTEGER :: N
LOGICAL :: CELL_CENTERED
CHARACTER(LABEL_LENGTH) :: QUANTITY,PROP_ID,SPEC_ID,PART_ID
NAMELIST /BNDE/ CELL_CENTERED,FYI,PART_ID,PROP_ID,QUANTITY,SPEC_ID
TYPE(BOUNDARY_ELEMENT_FILE_TYPE), POINTER :: BE=>NULL()

N_BNDE = 0
REWIND(LU_INPUT) ; INPUT_FILE_LINE_NUMBER = 0
COUNT_BNDE_LOOP: DO
CALL CHECKREAD('BNDE',LU_INPUT,IOS)
IF (IOS==1) EXIT COUNT_BNDE_LOOP
READ(LU_INPUT,NML=BNDE,END=309,ERR=310,IOSTAT=IOS)
N_BNDE = N_BNDE + 1
310 IF (IOS>0) THEN
WRITE(MESSAGE,'(A,I0,A,I0)') 'ERROR: Problem with BNDE number ',N_BNDE+1,', line number',INPUT_FILE_LINE_NUMBER
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF
ENDDO COUNT_BNDE_LOOP
309 REWIND(LU_INPUT) ; INPUT_FILE_LINE_NUMBER = 0

ALLOCATE(BOUNDARY_ELEMENT_FILE(N_BNDE),STAT=IZERO)
CALL ChkMemErr('READ','BOUNDARY_ELEMENT_FILE',IZERO)

READ_BNDE_LOOP: DO N=1,N_BNDE
BE => BOUNDARY_ELEMENT_FILE(N)
CELL_CENTERED = .TRUE.
PART_ID  = 'null'
PROP_ID  = 'null'
SPEC_ID  = 'null'
QUANTITY = 'WALL_TEMPERATURE'
CALL CHECKREAD('BNDE',LU_INPUT,IOS)
IF (IOS==1) EXIT READ_BNDE_LOOP
READ(LU_INPUT,BNDE)

! Look to see if output QUANTITY exists

CALL GET_QUANTITY_INDEX(BE%SMOKEVIEW_LABEL,BE%SMOKEVIEW_BAR_LABEL,BE%INDEX,I_DUM(1), &
BE%Y_INDEX,BE%Z_INDEX,BE%PART_INDEX,I_DUM(2),I_DUM(3),I_DUM(4),'BNDE', &
QUANTITY,'null',SPEC_ID,PART_ID,'null','null','null')

! Assign miscellaneous attributes to the boundary file

BE%CELL_CENTERED = CELL_CENTERED

! Check to see if PROP_ID exists

BE%PROP_INDEX = 0
IF(PROP_ID/='null')  CALL GET_PROPERTY_INDEX(BE%PROP_INDEX,'BNDE',PROP_ID)

ENDDO READ_BNDE_LOOP
REWIND(LU_INPUT) ; INPUT_FILE_LINE_NUMBER = 0

END SUBROUTINE READ_BNDE


SUBROUTINE CHECK_SURF_NAME(NAME,EXISTS)

LOGICAL, INTENT(OUT) :: EXISTS
CHARACTER(*), INTENT(IN) :: NAME
INTEGER :: NS

EXISTS = .FALSE.
DO NS=0,N_SURF
IF (NAME==SURFACE(NS)%ID) EXISTS = .TRUE.
ENDDO

END SUBROUTINE CHECK_SURF_NAME


SUBROUTINE GET_QUANTITY_INDEX(SMOKEVIEW_LABEL,SMOKEVIEW_BAR_LABEL,OUTPUT_INDEX,OUTPUT2_INDEX, &
Y_INDEX,Z_INDEX,PART_INDEX,DUCT_INDEX,NODE_INDEX,REAC_INDEX,OUTTYPE, &
QUANTITY,QUANTITY2,SPEC_ID_IN,PART_ID,DUCT_ID,NODE_ID,REAC_ID)
CHARACTER(*), INTENT(INOUT) :: QUANTITY
CHARACTER(*), INTENT(OUT) :: SMOKEVIEW_LABEL,SMOKEVIEW_BAR_LABEL
CHARACTER(*) :: SPEC_ID_IN,PART_ID,DUCT_ID,NODE_ID
CHARACTER(LABEL_LENGTH) :: SPEC_ID
```

```
CHARACTER(*), INTENT(IN) :: OUTTYPE,QUANTITY2,REAC_ID
INTEGER, INTENT(OUT) :: OUTPUT_INDEX,Y_INDEX,Z_INDEX,PART_INDEX,DUCT_INDEX,NODE_INDEX,REAC_INDEX,OUTPUT2_INDEX
INTEGER :: ND,NS,NN,NR,N_PLUS,N_MINUS

! Backward compatibility

IF (QUANTITY=='VEG_TEMPERATURE') QUANTITY='PARTICLE TEMPERATURE'

IF (QUANTITY=='oxygen') THEN
QUANTITY     = 'VOLUME FRACTION'
SPEC_ID_IN   = 'OXYGEN'
ENDIF
IF (QUANTITY=='carbon monoxide') THEN
QUANTITY     = 'VOLUME FRACTION'
SPEC_ID_IN   = 'CARBON MONOXIDE'
ENDIF
IF (QUANTITY=='carbon dioxide') THEN
QUANTITY     = 'VOLUME FRACTION'
SPEC_ID_IN   = 'CARBON DIOXIDE'
ENDIF
IF (QUANTITY=='soot') THEN
QUANTITY     = 'VOLUME FRACTION'
SPEC_ID_IN   = 'SOOT'
ENDIF
IF (QUANTITY=='soot density') THEN
QUANTITY     = 'DENSITY'
SPEC_ID_IN   = 'SOOT'
ENDIF
IF (QUANTITY=='fuel') THEN
QUANTITY     = 'VOLUME FRACTION'
WRITE(SPEC_ID_IN,'(A)') REACTION(1)%FUEL
ENDIF

DO ND=-N_OUTPUT_QUANTITIES,N_OUTPUT_QUANTITIES
IF (QUANTITY==OUTPUT_QUANTITY(ND)%OLD_NAME) QUANTITY = OUTPUT_QUANTITY(ND)%NAME
ENDDO

! Initialize indices

Y_INDEX = -1
Z_INDEX = -1

SPEC_ID = SPEC_ID_IN

IF (QUANTITY=='OPTICAL DENSITY'           .AND. SPEC_ID=='null') SPEC_ID='SOOT'
IF (QUANTITY=='EXTINCTION COEFFICIENT'    .AND. SPEC_ID=='null') SPEC_ID='SOOT'
IF (QUANTITY=='AEROSOL VOLUME FRACTION'   .AND. SPEC_ID=='null') SPEC_ID='SOOT'
IF (QUANTITY=='VISIBILITY'                .AND. SPEC_ID=='null') SPEC_ID='SOOT'

PART_INDEX = 0
DUCT_INDEX = 0
NODE_INDEX = 0
OUTPUT2_INDEX = 0
REAC_INDEX = 0

! Look for the appropriate SPEC or SMIX index

IF (SPEC_ID/='null') THEN
CALL GET_SPEC_OR_SMIX_INDEX(SPEC_ID,Y_INDEX,Z_INDEX)
IF (Z_INDEX>=0  .AND. Y_INDEX>=1) THEN
IF(TRIM(QUANTITY)=='DIFFUSIVITY') THEN
Y_INDEX=-999
ELSE
Z_INDEX=-999
ENDIF
ENDIF
IF (Z_INDEX<0 .AND. Y_INDEX<1) THEN
WRITE(MESSAGE,'(A,A,A,A)')  'ERROR: SPEC_ID ',TRIM(SPEC_ID),' is not explicitly specified for QUANTITY ',TRIM(
      QUANTITY)
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF
ENDIF

! Assign HVAC indexes

IF (DUCT_ID/='null') THEN
DO ND = 1, N_DUCTS
IF (DUCT_ID==DUCT(ND)%ID) THEN
DUCT_INDEX = ND
EXIT
ENDIF
ENDDO
ENDIF

IF (NODE_ID/='null') THEN
DO NN = 1, N_DUCTNODES
IF (NODE_ID==DUCTNODE(NN)%ID) THEN
NODE_INDEX = NN
```

```
EXIT
ENDIF
ENDDO
ENDIF

IF (TRIM(QUANTITY)=='FILTER LOADING') THEN
Y_INDEX = -999
DO NS = 1,N_TRACKED_SPECIES
IF (TRIM(SPECIES_MIXTURE(NS)%ID)==TRIM(SPEC_ID)) THEN
Z_INDEX = NS
EXIT
ENDIF
ENDDO
IF (Z_INDEX<0) THEN
WRITE(MESSAGE,'(A,A,A)')  'ERROR: FILTER LOADING. ',TRIM(SPEC_ID),' is not a tracked species'
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF
ENDIF


IF (TRIM(QUANTITY)=='EQUILIBRIUM VAPOR FRACTION' .OR. TRIM(QUANTITY)=='EQUILIBRIUM TEMPERATURE') THEN
Y_INDEX = -999
DO NS = 1,N_TRACKED_SPECIES
IF (TRIM(SPECIES_MIXTURE(NS)%ID)==TRIM(SPEC_ID)) THEN
Z_INDEX = NS
EXIT
ENDIF
ENDDO
IF (Z_INDEX<0) THEN
WRITE(MESSAGE,'(A,A,A)')  'ERROR: EQUILIBRIUM VAPOR FRACTION. ',TRIM(SPEC_ID),' is not a tracked species'
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF
IF (.NOT. SPECIES_MIXTURE(Z_INDEX)%EVAPORATING) THEN
WRITE(MESSAGE,'(A,A,A)')  'ERROR: EQUILIBRIUM VAPOR FRACTION. ',TRIM(SPEC_ID),' is not an evaporating species'
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF
ENDIF

IF (TRIM(QUANTITY)=='MIXTURE FRACTION') THEN
IF (N_REACTIONS/=1) THEN
WRITE(MESSAGE,'(A)') 'ERROR: MIXTURE FRACTION requires one and only one REAC input'
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF
N_PLUS = 0
N_MINUS = 0
DO NN = 1,N_TRACKED_SPECIES
IF (REACTION(1)%NU(NN) > 0)  THEN
N_PLUS = N_PLUS + 1
Z_INDEX = NN
ELSEIF (REACTION(1)%NU(NN) < 0)  THEN
N_MINUS = N_MINUS + 1
ENDIF
ENDDO
IF (N_PLUS/=1 .AND. N_MINUS/=2) THEN
WRITE(MESSAGE,'(A)') 'ERROR: MIXTURE FRACTION requires REAC of the form A + B -> C'
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF
ENDIF

IF (TRIM(QUANTITY)=='HRRPUV REAC') THEN
DO NR = 1,N_REACTIONS
IF (TRIM(REAC_ID)==TRIM(REACTION(NR)%ID)) REAC_INDEX = NR
ENDDO
IF (REAC_INDEX==0) THEN
WRITE(MESSAGE,'(3A)') 'ERROR: Output QUANTITY ',TRIM(QUANTITY),' requires a REAC_ID'
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF
ENDIF

! Assign PART_INDEX when PART_ID is specified

IF (PART_ID/='null') THEN
DO NS=1,N_LAGRANGIAN_CLASSES
IF (PART_ID==LAGRANGIAN_PARTICLE_CLASS(NS)%ID) THEN
PART_INDEX = NS
EXIT
ENDIF
ENDDO
ENDIF

! Loop over all possible output quantities and assign an index number to match the desired QUANTITY

DO ND=-N_OUTPUT_QUANTITIES,N_OUTPUT_QUANTITIES
IF (OUTPUT_QUANTITY(ND)%NAME=='null') CYCLE
IF (QUANTITY2==OUTPUT_QUANTITY(ND)%NAME) THEN

OUTPUT2_INDEX=ND

```

```
IF (OUTPUT_QUANTITY(ND)%SPEC_ID_REQUIRED .AND. (Y_INDEX<1 .AND. Z_INDEX<0)) THEN
WRITE(MESSAGE,'(3A)') 'ERROR: Output QUANTITY2 ',TRIM(QUANTITY2),' requires a SPEC_ID'
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF

! QUANTITY2 only works with SLCF at the moment
IF (.NOT.OUTPUT_QUANTITY(ND)%SLCF_APPROPRIATE) THEN
WRITE(MESSAGE,'(3A)') 'ERROR: The QUANTITY2 ',TRIM(QUANTITY2),' is not appropriate for SLCF'
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF

ENDIF
ENDDO

QUANTITY_INDEX_LOOP: DO ND=-N_OUTPUT_QUANTITIES,N_OUTPUT_QUANTITIES

QUANTITY_IF: IF (QUANTITY==OUTPUT_QUANTITY(ND)%NAME) THEN

OUTPUT_INDEX = ND

IF (OUTPUT_QUANTITY(ND)%QUANTITY2_REQUIRED .AND. OUTPUT2_INDEX==0) THEN
WRITE(MESSAGE,'(3A)') 'ERROR: Output QUANTITY ',TRIM(QUANTITY),' requires a QUANTITY2'
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF

IF (OUTPUT_QUANTITY(ND)%SPEC_ID_REQUIRED .AND. (Y_INDEX<1 .AND. Z_INDEX<0)) THEN
IF (SPEC_ID=='null') THEN
WRITE(MESSAGE,'(3A)') 'ERROR: Output QUANTITY ',TRIM(QUANTITY),' requires a SPEC_ID'
ELSE
WRITE(MESSAGE,'(5A)') 'ERROR: Output QUANTITY ',TRIM(QUANTITY),'. SPEC_ID ',TRIM(SPEC_ID),' not found.'
ENDIF
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF

IF (OUTPUT_QUANTITY(ND)%PART_ID_REQUIRED .AND. PART_INDEX<1) THEN
IF (PART_ID=='null') THEN
WRITE(MESSAGE,'(3A)') 'ERROR: Output QUANTITY ',TRIM(QUANTITY),' requires a PART_ID'
ELSE
WRITE(MESSAGE,'(5A)') 'ERROR: Output QUANTITY ',TRIM(QUANTITY),'. PART_ID ',TRIM(PART_ID),' not found.'
ENDIF
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF

IF (OUTPUT_QUANTITY(ND)%DUCT_ID_REQUIRED .AND. DUCT_INDEX<1) THEN
IF (DUCT_ID=='null') THEN
WRITE(MESSAGE,'(3A)') 'ERROR: Output QUANTITY ',TRIM(QUANTITY),' requires a DUCT_ID'
ELSE
WRITE(MESSAGE,'(5A)') 'ERROR: Output QUANTITY ',TRIM(QUANTITY),'. DUCT_ID ',TRIM(DUCT_ID),' not found.'
ENDIF
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF

IF (OUTPUT_QUANTITY(ND)%NODE_ID_REQUIRED .AND. NODE_INDEX<1) THEN
IF (NODE_ID=='null') THEN
WRITE(MESSAGE,'(3A)') 'ERROR: Output QUANTITY ',TRIM(QUANTITY),' requires a NODE_ID'
ELSE
WRITE(MESSAGE,'(5A)') 'ERROR: Output QUANTITY ',TRIM(QUANTITY),'. NODE_ID ',TRIM(NODE_ID),' not found.'
ENDIF
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF

IF (( QUANTITY=='RELATIVE HUMIDITY' .OR. QUANTITY=='HUMIDITY').AND. H2O_INDEX==0) THEN
WRITE(MESSAGE,'(A)') 'ERROR: RELATIVE HUMIDITY and HUMIDITY require SPEC=WATER VAPOR'
CALL SHUTDOWN(MESSAGE) ; RETURN
END IF

IF (TRIM(QUANTITY)=='DIFFUSIVITY' .AND. DNS .AND. Z_INDEX < 0) THEN
WRITE(MESSAGE,'(A)') 'ERROR: DIFFUSIVITY requires a tracked species SPEC_ID when using DNS'
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF

IF (TRIM(QUANTITY)=='SURFACE DEPOSITION') THEN
Y_INDEX = -999
DO NS=1,N_TRACKED_SPECIES
IF (TRIM(SPEC_ID)==TRIM(SPECIES_MIXTURE(NS)%ID)) THEN
Z_INDEX = NS
EXIT
ENDIF
ENDDO
IF (Z_INDEX < 0) THEN
DO NS=1,N_SPECIES
IF (TRIM(SPEC_ID)==TRIM(SPECIES(NS)%ID)) THEN
Y_INDEX = NS
EXIT
ENDIF
ENDDO
IF (Y_INDEX < 0) THEN
WRITE(MESSAGE,'(A,A,A,A)')'ERROR: SURFACE DEPOSITION for ',TRIM(SPEC_ID),' is invalid as species', &
```

```
' is not a tracked species'
CALL SHUTDOWN(MESSAGE) ; RETURN
ELSE
IF (SPECIES(Y_INDEX)%MODE /= AEROSOL_SPECIES) THEN
WRITE(MESSAGE,'(A,A,A,A)') 'ERROR: SURFACE DEPOSITION for ',TRIM(SPEC_ID),' is invalid as species', &
' is not an aerosol species'
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF
IF (SPECIES(Y_INDEX)%AWM_INDEX < 0) THEN
N_SURFACE_DENSITY_SPECIES = N_SURFACE_DENSITY_SPECIES + 1
SPECIES(Y_INDEX)%AWM_INDEX = N_SURFACE_DENSITY_SPECIES
ENDIF
ENDIF
ELSEIF (Z_INDEX==0) THEN
WRITE(MESSAGE,'(A)') 'ERROR: Cannot select background species for deposition'
CALL SHUTDOWN(MESSAGE) ; RETURN
ELSE
IF (.NOT. SPECIES_MIXTURE(Z_INDEX)%DEPOSITING) THEN
WRITE(MESSAGE,'(A,A,A)')'ERROR: SURFACE DEPOSITION for ',TRIM(SPEC_ID),' is not an aerosol tracked species'
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF
IF (SPECIES_MIXTURE(Z_INDEX)%AWM_INDEX < 0) THEN
N_SURFACE_DENSITY_SPECIES = N_SURFACE_DENSITY_SPECIES + 1
SPECIES_MIXTURE(Z_INDEX)%AWM_INDEX = N_SURFACE_DENSITY_SPECIES
ENDIF
ENDIF
ENDIF

IF (TRIM(QUANTITY)=='MPUV_Z' .OR. TRIM(QUANTITY)=='ADD_Z' .OR. TRIM(QUANTITY)=='ADT_Z' .OR. TRIM(QUANTITY)=='ADA_Z' .OR. &
TRIM(QUANTITY)=='QABS_Z' .OR. TRIM(QUANTITY)=='QSCA_Z' .OR. TRIM(QUANTITY)=='MPUA_Z' .OR. TRIM(QUANTITY)=='CPUA_Z' .OR. &
TRIM(QUANTITY)=='AMPUA_Z') THEN
IF (N_LAGRANGIAN_CLASSES==0) THEN
WRITE(MESSAGE,'(3A)') 'ERROR: The QUANTITY ',TRIM(QUANTITY),' requires liquid droplets'
CALL SHUTDOWN(MESSAGE) ; RETURN
ELSE
IF (.NOT. ALL(LAGRANGIAN_PARTICLE_CLASS%LIQUID_DROPLET)) THEN
WRITE(MESSAGE,'(3A)') 'ERROR: The QUANTITY ',TRIM(QUANTITY),' requires liquid droplets'
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF
ENDIF
ENDIF

SELECT CASE (TRIM(OUTTYPE))
CASE ('SLCF')
! Throw out bad slices
IF (.NOT. OUTPUT_QUANTITY(ND)%SLCF_APPROPRIATE) THEN
WRITE(MESSAGE,'(3A)') 'ERROR: The QUANTITY ',TRIM(QUANTITY),' is not appropriate for SLCF'
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF
CASE ('DEVC')
IF (.NOT.OUTPUT_QUANTITY(ND)%DEVC_APPROPRIATE) THEN
WRITE(MESSAGE,'(3A)') 'ERROR: The QUANTITY ',TRIM(QUANTITY),' is not appropriate for DEVC'
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF
IF (QUANTITY=='AMPUA' .OR. QUANTITY=='AMPUA_Z') ACCUMULATE_WATER = .TRUE.
CASE ('PART')
IF (.NOT. OUTPUT_QUANTITY(ND)%PART_APPROPRIATE) THEN
WRITE(MESSAGE,'(3A)') 'ERROR: ',TRIM(QUANTITY),' is not a particle output QUANTITY'
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF
CASE ('BNDF')
IF (.NOT. OUTPUT_QUANTITY(ND)%BNDF_APPROPRIATE) THEN
WRITE(MESSAGE,'(3A)') 'ERROR: The QUANTITY ',TRIM(QUANTITY),' is not appropriate for BNDF'
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF
IF (QUANTITY=='AMPUA' .OR. QUANTITY=='AMPUA_Z') ACCUMULATE_WATER = .TRUE.
CASE ('BNDE')
IF (.NOT. OUTPUT_QUANTITY(ND)%BNDE_APPROPRIATE) THEN
WRITE(MESSAGE,'(3A)') 'ERROR: The QUANTITY ',TRIM(QUANTITY),' is not appropriate for BNDE'
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF
IF (QUANTITY=='AMPUA' .OR. QUANTITY=='AMPUA_Z') ACCUMULATE_WATER = .TRUE.
CASE('ISOF')
IF (.NOT.OUTPUT_QUANTITY(ND)%ISOF_APPROPRIATE) THEN
WRITE(MESSAGE,'(3A)') 'ERROR: ISOF quantity ',TRIM(QUANTITY),' not appropriate for isosurface'
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF
CASE ('PLOT3D')
IF (OUTPUT_QUANTITY(ND)%SOLID_PHASE) THEN
WRITE(MESSAGE,'(5A)') 'ERROR: ',TRIM(OUTTYPE),'_QUANTITY ',TRIM(QUANTITY), ' not appropriate for gas phase'
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF
IF (.NOT.OUTPUT_QUANTITY(ND)%SLCF_APPROPRIATE) THEN
WRITE(MESSAGE,'(5A)') 'ERROR: ',TRIM(OUTTYPE),'_QUANTITY ',TRIM(QUANTITY), ' not appropriate for Plot3D'
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF
```

```fortran
CASE ('SMOKE3D')
IF (SMOKE3D .AND. (.NOT.OUTPUT_QUANTITY(ND)%MASS_FRACTION .AND. ND/=11)) THEN
WRITE(MESSAGE,'(5A)') 'ERROR: ',TRIM(OUTTYPE),' QUANTITY ',TRIM(QUANTITY),' must be a mass fraction'
CALL SHUTDOWN(MESSAGE) ; RETURN
ENDIF
CASE DEFAULT
END SELECT

! Assign Smokeview Label

IF (Z_INDEX>=0) THEN
IF (TRIM(QUANTITY)=='MIXTURE FRACTION') THEN
SMOKEVIEW_LABEL = TRIM(QUANTITY)
SMOKEVIEW_BAR_LABEL = TRIM(OUTPUT_QUANTITY(ND)%SHORT_NAME)
ELSE
SMOKEVIEW_LABEL = TRIM(SPECIES_MIXTURE(Z_INDEX)%ID)//' '//TRIM(QUANTITY)
SMOKEVIEW_BAR_LABEL = TRIM(OUTPUT_QUANTITY(ND)%SHORT_NAME)//'_'//TRIM(SPECIES_MIXTURE(Z_INDEX)%ID)
ENDIF
ELSEIF (Y_INDEX>0) THEN
SMOKEVIEW_LABEL = TRIM(SPECIES(Y_INDEX)%ID)//' '//TRIM(QUANTITY)
SMOKEVIEW_BAR_LABEL = TRIM(OUTPUT_QUANTITY(ND)%SHORT_NAME)//'_'//TRIM(SPECIES(Y_INDEX)%FORMULA)
ELSEIF (PART_INDEX>0) THEN
SMOKEVIEW_LABEL = TRIM(LAGRANGIAN_PARTICLE_CLASS(PART_INDEX)%ID)//' '//TRIM(QUANTITY)
SMOKEVIEW_BAR_LABEL = TRIM(OUTPUT_QUANTITY(ND)%SHORT_NAME)
ELSEIF (OUTPUT2_INDEX/=0) THEN
SMOKEVIEW_LABEL = TRIM(QUANTITY)//' '//TRIM(QUANTITY2)
SMOKEVIEW_BAR_LABEL = TRIM(OUTPUT_QUANTITY(ND)%SHORT_NAME)//' '//TRIM(OUTPUT_QUANTITY(OUTPUT2_INDEX)%SHORT_NAME)
ELSEIF (REAC_INDEX/=0) THEN
SMOKEVIEW_LABEL = TRIM(QUANTITY)//' '//TRIM(REACTION(REAC_INDEX)%ID)
SMOKEVIEW_BAR_LABEL = TRIM(OUTPUT_QUANTITY(ND)%SHORT_NAME)//' '//TRIM(REACTION(REAC_INDEX)%ID)
ELSE
SMOKEVIEW_LABEL = TRIM(QUANTITY)
SMOKEVIEW_BAR_LABEL = TRIM(OUTPUT_QUANTITY(ND)%SHORT_NAME)
ENDIF

RETURN
ENDIF QUANTITY_IF

ENDDO QUANTITY_INDEX_LOOP

! If no match for desired QUANTITY is found, stop the job

WRITE(MESSAGE,'(5A)') 'ERROR: ',TRIM(OUTTYPE),' QUANTITY ',TRIM(QUANTITY),' not found'
CALL SHUTDOWN(MESSAGE) ; RETURN

END SUBROUTINE GET_QUANTITY_INDEX


SUBROUTINE GET_SPEC_OR_SMIX_INDEX(SPEC_ID,Y_INDX,Z_INDX)

! Find the appropriate SPEC or SMIX index for the given SPEC_ID

CHARACTER(*), INTENT(IN) :: SPEC_ID
INTEGER, INTENT(OUT) :: Y_INDX,Z_INDX
INTEGER :: NS

Y_INDX = -999
Z_INDX = -999

DO NS=1,N_SPECIES
IF (TRIM(SPEC_ID)==TRIM(SPECIES(NS)%ID)) THEN
Y_INDX = NS
EXIT
ENDIF
ENDDO

DO NS=1,N_TRACKED_SPECIES
IF (TRIM(SPEC_ID)==TRIM(SPECIES_MIXTURE(NS)%ID)) THEN
Z_INDX = NS
RETURN
ENDIF
ENDDO

END SUBROUTINE GET_SPEC_OR_SMIX_INDEX



SUBROUTINE GET_PROPERTY_INDEX(P_INDEX,OUTTYPE,PROP_ID)

USE DEVICE_VARIABLES
CHARACTER(*), INTENT(IN) :: PROP_ID
CHARACTER(*), INTENT(IN) :: OUTTYPE
INTEGER, INTENT(INOUT) :: P_INDEX
INTEGER :: NN

DO NN=1,N_PROP
IF (PROP_ID==PROPERTY(NN)%ID) THEN
P_INDEX = NN
```

```
SELECT CASE (TRIM(OUTTYPE))
CASE ('SLCF')
CASE ('DEVC')
CASE ('PART')
CASE ('OBST')
CASE ('BNDF')
CASE ('PLOT3D')
CASE DEFAULT
END SELECT
RETURN
ENDIF
ENDDO

WRITE(MESSAGE,'(5A)') 'ERROR: ',TRIM(OUTTYPE),' PROP_ID ',TRIM(PROP_ID),' not found'
CALL SHUTDOWN(MESSAGE) ; RETURN

END SUBROUTINE GET_PROPERTY_INDEX

SUBROUTINE READ_CSVF
USE OUTPUT_DATA

CHARACTER(256) :: CSVFILE,UVWFILE='null'
NAMELIST /CSVF/ CSVFILE,UVWFILE

N_CSVF=0
REWIND(LU_INPUT) ; INPUT_FILE_LINE_NUMBER = 0
COUNT_CSVF_LOOP: DO
CALL CHECKREAD('CSVF',LU_INPUT,IOS)
IF (IOS==1) EXIT COUNT_CSVF_LOOP
READ(LU_INPUT,NML=CSVF,END=16,ERR=17,IOSTAT=IOS)
N_CSVF=N_CSVF+1
16 IF (IOS>0) THEN ; CALL SHUTDOWN('ERROR: problem with CSVF line') ; RETURN ; ENDIF
ENDDO COUNT_CSVF_LOOP
17 REWIND(LU_INPUT) ; INPUT_FILE_LINE_NUMBER = 0

IF (N_CSVF==0) RETURN

! Allocate CSVFINFO array

ALLOCATE(CSVFINFO(N_CSVF),STAT=IZERO)
CALL ChkMemErr('READ','CSVF',IZERO)

READ_CSVF_LOOP: DO I=1,N_CSVF

CALL CHECKREAD('CSVF',LU_INPUT,IOS)
IF (IOS==1) EXIT READ_CSVF_LOOP

! Read the CSVF line

READ(LU_INPUT,CSVF,END=37)

CSVFINFO(I)%CSVFILE = TRIM(CSVFILE)
IF (TRIM(UVWFILE)/='null' .AND. I>NMESHES) THEN
CALL SHUTDOWN('Problem with CSVF line: UVWFILE must be in order with MESH.') ; RETURN
ELSE
CSVFINFO(I)%UVWFILE = UVWFILE
UVW_RESTART = .TRUE.
ENDIF

ENDDO READ_CSVF_LOOP
37 REWIND(LU_INPUT) ; INPUT_FILE_LINE_NUMBER = 0

END SUBROUTINE READ_CSVF


SUBROUTINE CALC_H2O_HV
USE PROPERTY_DATA, ONLY: JANAF_TABLE,JANAF_TABLE_LIQUID
CHARACTER(LABEL_LENGTH) :: WATER_VAPOR='WATER VAPOR'
INTEGER :: I
REAL(EB) :: CP_G,CP_G_O,CP_L,CP_L_O,H_G,H_L,H_G_0,H_L_0,G_F,RCON,H_V,T_R,T_M,T_B,DENSITY,MU_LIQUID,K_LIQUID,
      BETA_LIQUID
LOGICAL :: FUEL

DO I=0,5000
CALL JANAF_TABLE (I,CP_G,H_G_0,WATER_VAPOR,RCON,FUEL,G_F)
CALL JANAF_TABLE_LIQUID (I,CP_L,H_V,H_L_0,T_R,T_M,T_B,WATER_VAPOR,FUEL,DENSITY,MU_LIQUID,K_LIQUID,BETA_LIQUID)
IF (I==0) THEN
H_G = H_G_0
H_L = H_L_0
ELSE
H_G = H_G + 0.5_EB*(CP_G+CP_G_O)
H_L = H_L + 0.5_EB*(CP_L+CP_L_O)
ENDIF
H_V_H2O(I) = H_G-H_L
CP_G_O=CP_G
CP_L_O=CP_L
END DO
```

```
END SUBROUTINE CALC_H2O_HV

!Sesa
SUBROUTINE read_pen

integer :: ll

!creating a namelist group name PENA
NAMELIST /PENA/ penalizationParameter, blendingParameter, dampingParameter, &
penXmin, penXmax, penYmin, penYmax, penZmin, penZmax, &
mX, mY, mZ, b , dataFileName, pena_I, pena_J, pena_K!,penU0, penV0, penW0,

!opening the input file and reading the PENA namelist
OPEN(LU_INPUT,FILE=FN_INPUT,ACTION='READ')

npen=0
pendat_size=0

!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)
npen=npen+1
pendat_size(npen)=(pena_I +1)*(pena_J +1)*(pena_K +1)
ENDDO COUNT_PEN_LOOP

66 if (npen.gt.0) allocate(pendat(npen,(15+3*(maxval(pendat_size)))))

pendat=0d0

REWIND(LU_INPUT)

!reading the penalisation regions
READ_PEN_LOOP: DO ll=1,npen

!make sure everything is read fresh
penalizationParameter=0d0
penXmin=0d0
penXmax=0d0
penYmin=0d0
penYmax=0d0
penZmin=0d0
penZmax=0d0
penU0=0d0
penV0=0d0
penW0=0d0
blendingParameter=0d0
mX=0d0
mY=0d0
mZ=0d0
b=0d0
dampingParameter=0d0

READ(LU_INPUT,NML=PENA,END=16,ERR=17,IOSTAT=IOS)
!opening and reading the csv files
OPEN(UNIT=fileread,FILE=dataFileName, STATUS='OLD', FORM='FORMATTED',ACTION='READ', IOSTAT=IERROR)

IF (IERROR/=0) THEN
MESSAGE = 'ERROR: Problem with the csv file '
CALL SHUTDOWN(MESSAGE)
RETURN
ENDIF

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


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

16 IF (IOS>0) THEN ; CALL SHUTDOWN('ERROR: problem with PENALIZATION line') ; RETURN ;ENDIF

ENDDO READ_PEN_LOOP

close(fileread)
17 close(LU_INPUT)

END SUBROUTINE read_pen

SUBROUTINE read_trunks
!open the input file again look for the &trnk line
!read it and close the file
namelist /trnk/ ntrunks, trunks, eta,trnk_min,trnk_max
namelist /tloc/ trnk_loc

!open the file with the trunk details in it
OPEN(LU_INPUT,FILE=FN_INPUT,ACTION='READ')

COUNT_TRUNK_LOOP: DO
CALL CHECKREAD('TRNK',LU_INPUT,IOS)
IF (IOS==1) EXIT COUNT_TRUNK_LOOP
READ(LU_INPUT,NML=trnk,END=16,ERR=17,IOSTAT=IOS)
16 IF (IOS>0) THEN ; CALL SHUTDOWN('ERROR: problem with TRNK line') ; RETURN ;ENDIF

ENDDO COUNT_TRUNK_LOOP

rewind(LU_INPUT)
allocate(trnk_loc(2,ntrunks))

GET_TRUNK_LOOP: DO
CALL CHECKREAD('TLOC',LU_INPUT,IOS)
IF (IOS==1) EXIT GET_TRUNK_LOOP
READ(LU_INPUT,NML=tloc,END=18,ERR=17,IOSTAT=IOS)
18 IF (IOS>0) THEN ; CALL SHUTDOWN('ERROR: problem with TLOC line') ; RETURN ;ENDIF
ENDDO GET_TRUNK_LOOP

17 close(LU_INPUT)


END SUBROUTINE read_trunks

END MODULE READ_INPUT
```

A.3 *velo.f90*

```

MODULE VELO

! Module computes the velocity flux terms, baroclinic torque correction terms, and performs the CFL Check

USE PRECISION_PARAMETERS
USE GLOBAL_CONSTANTS
USE MESH_POINTERS
USE COMP_FUNCTIONS, ONLY: SECOND

IMPLICIT NONE
PRIVATE

PUBLIC COMPUTE_VELOCITY_FLUX,VELOCITY_PREDICTOR,VELOCITY_CORRECTOR,NO_FLUX,BAROCLINIC_CORRECTION, &
MATCH_VELOCITY,MATCH_VELOCITY_FLUX,VELOCITY_BC,COMPUTE_VISCOSITY,VISCOSITY_BC
PRIVATE VELOCITY_FLUX,VELOCITY_FLUX_CYLINDRICAL


CONTAINS


SUBROUTINE COMPUTE_VELOCITY_FLUX(T,DT,NM,FUNCTION_CODE)

REAL(EB), INTENT(IN) :: T,DT
REAL(EB) :: TNOW
INTEGER, INTENT(IN) :: NM,FUNCTION_CODE

IF (SOLID_PHASE_ONLY .OR. FREEZE_VELOCITY) RETURN

```

```
TNOW = SECOND()

SELECT CASE(FUNCTION_CODE)
CASE(1)
IF (PREDICTOR .OR. COMPUTE_VISCOSITY_TWICE) CALL COMPUTE_VISCOSITY(T,NM)
CASE(2)
BAROCLINIC_TERMS_ATTACHED = .FALSE.
IF (PREDICTOR .OR. COMPUTE_VISCOSITY_TWICE) CALL VISCOSITY_BC(NM)
IF (.NOT.CYLINDRICAL) CALL VELOCITY_FLUX(T,DT,NM)
IF (     CYLINDRICAL) CALL VELOCITY_FLUX_CYLINDRICAL(T,NM)
END SELECT

T_USED(4) = T_USED(4) + SECOND() - TNOW
END SUBROUTINE COMPUTE_VELOCITY_FLUX


SUBROUTINE COMPUTE_VISCOSITY(T,NM)

USE PHYSICAL_FUNCTIONS, ONLY: GET_VISCOSITY,LES_FILTER_WIDTH_FUNCTION,GET_POTENTIAL_TEMPERATURE
USE TURBULENCE, ONLY: VARDEN_DYNSMAG,TEST_FILTER,FILL_EDGES,WALL_MODEL,RNG_EDDY_VISCOSITY,WALE_VISCOSITY
USE MATH_FUNCTIONS, ONLY:EVALUATE_RAMP
REAL(EB), INTENT(IN) :: T
INTEGER, INTENT(IN) :: NM
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),&
DUDX,DUDY,DUDZ,DVDX,DVDY,DVDZ,DWDX,DWDY,DWDZ,MU_EFF,SLIP_COEF,VEL_GAS,VEL_T,RAMP_T,TSI,&
VDF,LS,THETA_0,THETA_1,THETA_2,DTDZBAR,WGT
REAL(EB), PARAMETER :: RAPLUS=1._EB/26._EB, C_LS=0.76_EB
INTEGER :: I,J,K,IIG,JJG,KKG,II,JJ,KK,IW,IOR
REAL(EB), POINTER, DIMENSION(:,:,:) :: RHOP=>NULL(),UP=>NULL(),VP=>NULL(),WP=>NULL(), &
UP_HAT=>NULL(),VP_HAT=>NULL(),WP_HAT=>NULL(), &
UU=>NULL(),VV=>NULL(),WW=>NULL(),DTDZ=>NULL()
REAL(EB), POINTER, DIMENSION(:,:,:,:) :: ZZP=>NULL()
INTEGER, POINTER, DIMENSION(:,:,:) :: CELL_COUNTER=>NULL()
TYPE(WALL_TYPE), POINTER :: WC=>NULL()
TYPE(SURFACE_TYPE), POINTER :: SF=>NULL()

IF (EVACUATION_ONLY(NM)) RETURN ! No need to update viscosity, use initial one

CALL POINT_TO_MESH(NM)

IF (PREDICTOR) THEN
RHOP => RHO
UU   => U
VV   => V
WW   => W
ZZP  => ZZ
ELSE
RHOP => RHOS
UU   => US
VV   => VS
WW   => WS
ZZP  => ZZS
ENDIF

! Compute viscosity for DNS using primitive species

!$OMP PARALLEL DO FIRSTPRIVATE(ZZ_GET) SCHEDULE(guided)
DO K=1,KBAR
DO J=1,JBAR
DO I=1,IBAR
IF (SOLID(CELL_INDEX(I,J,K))) CYCLE
ZZ_GET(1:N_TRACKED_SPECIES) = ZZP(I,J,K,1:N_TRACKED_SPECIES)
CALL GET_VISCOSITY(ZZ_GET,MU_DNS(I,J,K),TMP(I,J,K))
ENDDO
ENDDO
ENDDO
!$OMP END PARALLEL DO

CALL COMPUTE_STRAIN_RATE(NM)

SELECT_TURB: SELECT CASE (TURB_MODEL)

CASE (NO_TURB_MODEL)

MU = MU_DNS

CASE (CONSMAG,DYNSMAG) SELECT_TURB ! Smagorinsky (1963) eddy viscosity

IF (PREDICTOR .AND. TURB_MODEL==DYNSMAG) CALL VARDEN_DYNSMAG(NM) ! dynamic procedure, Moin et al. (1991)

DO K=1,KBAR
DO J=1,JBAR
DO I=1,IBAR
IF (SOLID(CELL_INDEX(I,J,K))) CYCLE
MU(I,J,K) = MU_DNS(I,J,K) + RHOP(I,J,K)*CSD2(I,J,K)*STRAIN_RATE(I,J,K)
ENDDO
ENDDO
ENDDO
```

```

CASE (DEARDORFF) SELECT_TURB ! Deardorff (1980) eddy viscosity model (current default)

! Velocities relative to the p-cell center

UP => WORK1
VP => WORK2
WP => WORK3
UP=0._EB
VP=0._EB
WP=0._EB

DO K=1,KBAR
DO J=1,JBAR
DO I=1,IBAR
IF (SOLID(CELL_INDEX(I,J,K))) CYCLE
UP(I,J,K) = 0.5_EB*(UU(I,J,K) + UU(I-1,J,K))
VP(I,J,K) = 0.5_EB*(VV(I,J,K) + VV(I,J-1,K))
WP(I,J,K) = 0.5_EB*(WW(I,J,K) + WW(I,J,K-1))
ENDDO
ENDDO
ENDDO

! fill mesh boundary ghost cells

DO IW=1,N_EXTERNAL_WALL_CELLS
WC=>WALL(IW)
SELECT CASE(WC%BOUNDARY_TYPE)
CASE(INTERPOLATED_BOUNDARY)
II = WC%ONE_D%II
JJ = WC%ONE_D%JJ
KK = WC%ONE_D%KK
UP(II,JJ,KK) = U_GHOST(IW)
VP(II,JJ,KK) = V_GHOST(IW)
WP(II,JJ,KK) = W_GHOST(IW)
CASE(OPEN_BOUNDARY,MIRROR_BOUNDARY)
II = WC%ONE_D%II
JJ = WC%ONE_D%JJ
KK = WC%ONE_D%KK
IIG = WC%ONE_D%IIG
JJG = WC%ONE_D%JJG
KKG = WC%ONE_D%KKG
UP(II,JJ,KK) = UP(IIG,JJG,KKG)
VP(II,JJ,KK) = VP(IIG,JJG,KKG)
WP(II,JJ,KK) = WP(IIG,JJG,KKG)
END SELECT
ENDDO

! fill edge and corner ghost cells

CALL FILL_EDGES(UP)
CALL FILL_EDGES(VP)
CALL FILL_EDGES(WP)

UP_HAT => WORK4
VP_HAT => WORK5
WP_HAT => WORK6
UP_HAT=0._EB
VP_HAT=0._EB
WP_HAT=0._EB

CALL TEST_FILTER(UP_HAT,UP)
CALL TEST_FILTER(VP_HAT,VP)
CALL TEST_FILTER(WP_HAT,WP)

POTENTIAL_TEMPERATURE_IF: IF (.NOT.POTENTIAL_TEMPERATURE_CORRECTION) THEN
!$OMP PARALLEL DO PRIVATE(DELTA, KSGS, NU_EDDY) SCHEDULE(static)
DO K=1,KBAR
DO J=1,JBAR
DO I=1,IBAR
IF (SOLID(CELL_INDEX(I,J,K))) CYCLE
DELTA = LES_FILTER_WIDTH_FUNCTION(DX(I),DY(J),DZ(K))
KSGS = 0.5_EB*( (UP(I,J,K)-UP_HAT(I,J,K))**2 + (VP(I,J,K)-VP_HAT(I,J,K))**2 + (WP(I,J,K)-WP_HAT(I,J,K))**2 )
NU_EDDY = C_DEARDORFF*DELTA*SQRT(KSGS)
MU(I,J,K) = MU_DNS(I,J,K) + RHOP(I,J,K)*NU_EDDY
ENDDO
ENDDO
ENDDO
!$OMP END PARALLEL DO
ELSE POTENTIAL_TEMPERATURE_IF
DTDZ => WORK7
DTDZ = 0._EB
DO K=0,KBAR
DO J=0,JBAR
DO I=0,IBAR
THETA_1 = GET_POTENTIAL_TEMPERATURE(TMP(I,J,K),ZC(K))
THETA_2 = GET_POTENTIAL_TEMPERATURE(TMP(I,J,K+1),ZC(K+1))
DTDZ(I,J,K) = (THETA_2-THETA_1)*RDZN(K)
```

```fortran
ENDDO
ENDDO
ENDDO
DO K=1,KBAR
DO J=1,JBAR
DO I=1,IBAR
IF (SOLID(CELL_INDEX(I,J,K))) CYCLE
DELTA = LES_FILTER_WIDTH_FUNCTION(DX(I),DY(J),DZ(K))
LS = DELTA
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 )
DTDZBAR = 0.5_EB*(DTDZ(I,J,K)+DTDZ(I,J,K+1))
IF (DTDZBAR>0._EB) THEN
THETA_0 = GET_POTENTIAL_TEMPERATURE(TMP_0(K),ZC(K))
LS = C_LS*SQRT(KSGS)/SQRT(ABS(GVEC(3))/THETA_0*DTDZBAR) ! von Schoenberg Eq. (3.19)
ENDIF
NU_EDDY = C_DEARDORFF*MIN(LS,DELTA)*SQRT(KSGS)
MU(I,J,K) = MU_DNS(I,J,K) + RHOP(I,J,K)*NU_EDDY
ENDDO
ENDDO
ENDDO
ENDIF POTENTIAL_TEMPERATURE_IF

CASE (VREMAN) SELECT_TURB ! Vreman (2004) eddy viscosity model (experimental)

! A. W. Vreman. An eddy-viscosity subgrid-scale model for turbulent shear flow: Algebraic theory and applications.
! Phys. Fluids, 16(10):3670-3681, 2004.

DO K=1,KBAR
DO J=1,JBAR
DO I=1,IBAR
IF (SOLID(CELL_INDEX(I,J,K))) CYCLE
DUDX = RDX(I)*(UU(I,J,K)-UU(I-1,J,K))
DVDY = RDY(J)*(VV(I,J,K)-VV(I,J-1,K))
DWDZ = RDZ(K)*(WW(I,J,K)-WW(I,J,K-1))
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))
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))
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))
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))
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))
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))

! Vreman, Eq. (6)
A_IJ(1,1)=DUDX; A_IJ(2,1)=DUDY; A_IJ(3,1)=DUDZ
A_IJ(1,2)=DVDX; A_IJ(2,2)=DVDY; A_IJ(3,2)=DVDZ
A_IJ(1,3)=DWDX; A_IJ(2,3)=DWDY; A_IJ(3,3)=DWDZ

AA=0._EB
DO JJ=1,3
DO II=1,3
AA = AA + A_IJ(II,JJ)*A_IJ(II,JJ)
ENDDO
ENDDO

! Vreman, Eq. (7)
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
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
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

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)
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)
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)

BB = B_IJ(1,1)*B_IJ(2,2) - B_IJ(1,2)**2 &
+ B_IJ(1,1)*B_IJ(3,3) - B_IJ(1,3)**2 &
+ B_IJ(2,2)*B_IJ(3,3) - B_IJ(2,3)**2    ! Vreman, Eq. (8)

IF (ABS(AA)>TWO_EPSILON_EB .AND. BB>TWO_EPSILON_EB) THEN
NU_EDDY = C_VREMAN*SQRT(BB/AA)  ! Vreman, Eq. (5)
ELSE
NU_EDDY=0._EB
ENDIF

MU(I,J,K) = MU_DNS(I,J,K) + RHOP(I,J,K)*NU_EDDY

ENDDO
ENDDO
ENDDO

CASE (RNG) SELECT_TURB

! A. Yakhot, S. A. Orszag, V. Yakhot, and M. Israeli. Renormalization Group Formulation of Large-Eddy Simulation.
! Journal of Scientific Computing, 1(1):1-51, 1989.

DO K=1,KBAR
DO J=1,JBAR
DO I=1,IBAR
IF (SOLID(CELL_INDEX(I,J,K))) CYCLE
```

```fortran
DELTA = LES_FILTER_WIDTH_FUNCTION(DX(I),DY(J),DZ(K))
CALL RNG_EDDY_VISCOSITY(MU_EFF,MU_DNS(I,J,K),RHOP(I,J,K),STRAIN_RATE(I,J,K),DELTA)
MU(I,J,K) = MU_EFF
ENDDO
ENDDO
ENDDO

CASE (WALE) SELECT_TURB

DO K=1,KBAR
DO J=1,JBAR
DO I=1,IBAR
IF (SOLID(CELL_INDEX(I,J,K))) CYCLE
DELTA = LES_FILTER_WIDTH_FUNCTION(DX(I),DY(J),DZ(K))
! compute velocity gradient tensor
DUDX = RDX(I)*(UU(I,J,K)-UU(I-1,J,K))
DVDY = RDY(J)*(VV(I,J,K)-VV(I,J-1,K))
DWDZ = RDZ(K)*(WW(I,J,K)-WW(I,J,K-1))
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))
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))
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))
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))
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))
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))
A_IJ(1,1)=DUDX; A_IJ(1,2)=DUDY; A_IJ(1,3)=DUDZ
A_IJ(2,1)=DVDX; A_IJ(2,2)=DVDY; A_IJ(2,3)=DVDZ
A_IJ(3,1)=DWDX; A_IJ(3,2)=DWDY; A_IJ(3,3)=DWDZ

CALL WALE_VISCOSITY(NU_EDDY,A_IJ,DELTA)

MU(I,J,K) = MU_DNS(I,J,K) + RHOP(I,J,K)*NU_EDDY
ENDDO
ENDDO
ENDDO

END SELECT SELECT_TURB

! Compute resolved kinetic energy per unit mass

DO K=1,KBAR
DO J=1,JBAR
DO I=1,IBAR
IF (SOLID(CELL_INDEX(I,J,K))) CYCLE
U2 = 0.25_EB*(UU(I-1,J,K)+UU(I,J,K))**2
V2 = 0.25_EB*(VV(I,J-1,K)+VV(I,J,K))**2
W2 = 0.25_EB*(WW(I,J,K-1)+WW(I,J,K))**2
KRES(I,J,K) = 0.5_EB*(U2+V2+W2)
ENDDO
ENDDO
ENDDO

! Mirror viscosity into solids and exterior boundary cells

CELL_COUNTER => IWORK1 ; CELL_COUNTER = 0

WALL_LOOP: DO IW=1,N_EXTERNAL_WALL_CELLS+N_INTERNAL_WALL_CELLS

WC=>WALL(IW)
IF (WC%BOUNDARY_TYPE==NULL_BOUNDARY) CYCLE WALL_LOOP
II  = WC%ONE_D%II
JJ  = WC%ONE_D%JJ
KK  = WC%ONE_D%KK
IOR = WC%ONE_D%IOR
IIG = WC%ONE_D%IIG
JJG = WC%ONE_D%JJG
KKG = WC%ONE_D%KKG
SF=>SURFACE(WC%SURF_INDEX)

SELECT CASE(WC%BOUNDARY_TYPE)

CASE(SOLID_BOUNDARY)

IF (ABS(SF%T_IGN-T_BEGIN)<=SPACING(SF%T_IGN) .AND. SF%RAMP_INDEX(TIME_VELO)>=1) THEN
TSI = T
ELSE
TSI = T-SF%T_IGN
ENDIF
RAMP_T = EVALUATE_RAMP(TSI,SF%TAU(TIME_VELO),SF%RAMP_INDEX(TIME_VELO))
VEL_T = RAMP_T*SQRT(SF%VEL_T(1)**2 + SF%VEL_T(2)**2)

SELECT CASE(ABS(IOR))
CASE(1)
VEL_GAS = SQRT( 0.25_EB*( (VV(IIG,JJG,KKG)+VV(IIG,JJG-1,KKG))**2 + (WW(IIG,JJG,KKG)+WW(IIG,JJG,KKG-1))**2 ) )
CASE(2)
VEL_GAS = SQRT( 0.25_EB*( (UU(IIG,JJG,KKG)+UU(IIG-1,JJG,KKG))**2 + (WW(IIG,JJG,KKG)+WW(IIG,JJG,KKG-1))**2 ) )
CASE(3)
VEL_GAS = SQRT( 0.25_EB*( (UU(IIG,JJG,KKG)+UU(IIG-1,JJG,KKG))**2 + (VV(IIG,JJG,KKG)+VV(IIG,JJG-1,KKG))**2 ) )
END SELECT
```

```fortran

CALL WALL_MODEL(SLIP_COEF,WC%U_TAU,WC%Y_PLUS,VEL_GAS-VEL_T,&
MU_DNS(IIG,JJG,KKG)/RHO(IIG,JJG,KKG),1._EB/WC%ONE_D%RDN,SURFACE(WC%SURF_INDEX)%ROUGHNESS)

IF (LES) THEN
DELTA = LES_FILTER_WIDTH_FUNCTION(DX(IIG),DY(JJG),DZ(KKG))
SELECT CASE(NEAR_WALL_TURB_MODEL)
CASE DEFAULT ! Constant Smagorinsky with Van Driest damping
VDF = 1._EB-EXP(-WC%Y_PLUS*RAPLUS)
NU_EDDY = (VDF*C_SMAGORINSKY*DELTA)**2*STRAIN_RATE(IIG,JJG,KKG)
CASE(WALE)
! compute velocity gradient tensor
DUDX = RDX(IIG)*(UU(IIG,JJG,KKG)-UU(IIG-1,JJG,KKG))
DVDY = RDY(JJG)*(VV(IIG,JJG,KKG)-VV(IIG,JJG-1,KKG))
DWDZ = RDZ(KKG)*(WW(IIG,JJG,KKG)-WW(IIG,JJG,KKG-1))
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))
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))
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))
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))
DWDX = 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))
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))
A_IJ(1,1)=DUDX; A_IJ(1,2)=DUDY; A_IJ(1,3)=DUDZ
A_IJ(2,1)=DVDX; A_IJ(2,2)=DVDY; A_IJ(2,3)=DVDZ
A_IJ(3,1)=DWDX; A_IJ(3,2)=DWDY; A_IJ(3,3)=DWDZ
CALL WALE_VISCOSITY(NU_EDDY,A_IJ,DELTA)
END SELECT
IF (CELL_COUNTER(IIG,JJG,KKG)==0) MU(IIG,JJG,KKG) = 0._EB
CELL_COUNTER(IIG,JJG,KKG) = CELL_COUNTER(IIG,JJG,KKG) + 1
WGT = 1._EB/REAL(CELL_COUNTER(IIG,JJG,KKG),EB)
MU(IIG,JJG,KKG) = (1._EB-WGT)*MU(IIG,JJG,KKG) + WGT*(MU_DNS(IIG,JJG,KKG) + RHOP(IIG,JJG,KKG)*NU_EDDY)
ELSE
MU(IIG,JJG,KKG) = MU_DNS(IIG,JJG,KKG)
ENDIF

IF (SOLID(CELL_INDEX(II,JJ,KK))) MU(II,JJ,KK) = MU(IIG,JJG,KKG)

CASE(OPEN_BOUNDARY,MIRROR_BOUNDARY)

MU(II,JJ,KK) = MU(IIG,JJG,KKG)
KRES(II,JJ,KK) = KRES(IIG,JJG,KKG)

END SELECT

ENDDO WALL_LOOP

MU(   0,0:JBP1,   0) = MU(   1,0:JBP1,1)
MU(IBP1,0:JBP1,   0) = MU(IBAR,0:JBP1,1)
MU(IBP1,0:JBP1,KBP1) = MU(IBAR,0:JBP1,KBAR)
MU(   0,0:JBP1,KBP1) = MU(   1,0:JBP1,KBAR)
MU(0:IBP1,   0,   0) = MU(0:IBP1,   1,1)
MU(0:IBP1,JBP1,0)    = MU(0:IBP1,JBAR,1)
MU(0:IBP1,JBP1,KBP1) = MU(0:IBP1,JBAR,KBAR)
MU(0:IBP1,0,KBP1)    = MU(0:IBP1,   1,KBAR)
MU(0,   0,0:KBP1)    = MU(   1,   1,0:KBP1)
MU(IBP1,0,0:KBP1)    = MU(IBAR,   1,0:KBP1)
MU(IBP1,JBP1,0:KBP1) = MU(IBAR,JBAR,0:KBP1)
MU(0,JBP1,0:KBP1)    = MU(   1,JBAR,0:KBP1)

END SUBROUTINE COMPUTE_VISCOSITY


SUBROUTINE COMPUTE_STRAIN_RATE(NM)

INTEGER, INTENT(IN) :: NM
REAL(EB) :: DUDX,DUDY,DUDZ,DVDX,DVDY,DVDZ,DWDX,DWDY,DWDZ,S11,S22,S33,S12,S13,S23,ONTHDIV
INTEGER :: I,J,K,IOR,IIG,JJG,KKG,IW,SURF_INDEX
REAL(EB), POINTER, DIMENSION(:,:,:) :: UU=>NULL(),VV=>NULL(),WW=>NULL()
TYPE(WALL_TYPE), POINTER :: WC=>NULL()

CALL POINT_TO_MESH(NM)

IF (PREDICTOR) THEN
UU => U
VV => V
WW => W
ELSE
UU => US
VV => VS
WW => WS
ENDIF

SELECT CASE (TURB_MODEL)
CASE DEFAULT
DO K=1,KBAR
DO J=1,JBAR
DO I=1,IBAR
IF (SOLID(CELL_INDEX(I,J,K))) CYCLE
DUDX = RDX(I)*(UU(I,J,K)-UU(I-1,J,K))
```

```
DVDY = RDY(J)*(VV(I,J,K)-VV(I,J-1,K))
DWDZ = RDZ(K)*(WW(I,J,K)-WW(I,J,K-1))
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))
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))
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))
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))
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))
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))
ONTHDIV = ONTH*(DUDX+DVDY+DWDZ)
S11 = DUDX - ONTHDIV
S22 = DVDY - ONTHDIV
S33 = DWDZ - ONTHDIV
S12 = 0.5_EB*(DUDY+DVDX)
S13 = 0.5_EB*(DUDZ+DWDX)
S23 = 0.5_EB*(DVDZ+DWDY)
STRAIN_RATE(I,J,K) = SQRT(2._EB*(S11**2 + S22**2 + S33**2 + 2._EB*(S12**2 + S13**2 + S23**2)))
ENDDO
ENDDO
ENDDO
CASE (DEARDORFF)
! Here we omit the 3D loop, we only need the wall cell values of STRAIN_RATE
END SELECT

WALL_LOOP: DO IW=1,N_EXTERNAL_WALL_CELLS+N_INTERNAL_WALL_CELLS
WC=>WALL(IW)
IF (WC%BOUNDARY_TYPE/=SOLID_BOUNDARY) CYCLE WALL_LOOP

SURF_INDEX = WC%SURF_INDEX
IIG = WC%ONE_D%IIG
JJG = WC%ONE_D%JJG
KKG = WC%ONE_D%KKG
IOR = WC%ONE_D%IOR

! Handle the case where OBST lives on an external boundary
IF (IW>N_EXTERNAL_WALL_CELLS) THEN
SELECT CASE(IOR)
CASE( 1); IF (IIG>IBAR) CYCLE WALL_LOOP
CASE(-1); IF (IIG<1)    CYCLE WALL_LOOP
CASE( 2); IF (JJG>JBAR) CYCLE WALL_LOOP
CASE(-2); IF (JJG<1)    CYCLE WALL_LOOP
CASE( 3); IF (KKG>KBAR) CYCLE WALL_LOOP
CASE(-3); IF (KKG<1)    CYCLE WALL_LOOP
END SELECT
ENDIF

DUDX = RDX(IIG)*(UU(IIG,JJG,KKG)-UU(IIG-1,JJG,KKG))
DVDY = RDY(JJG)*(VV(IIG,JJG,KKG)-VV(IIG,JJG-1,KKG))
DWDZ = RDZ(KKG)*(WW(IIG,JJG,KKG)-WW(IIG,JJG,KKG-1))
ONTHDIV = ONTH*(DUDX+DVDY+DWDZ)
S11 = DUDX - ONTHDIV
S22 = DVDY - ONTHDIV
S33 = DWDZ - ONTHDIV

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))
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))
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))
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))
DWDX = 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))
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))

S12 = 0.5_EB*(DUDY+DVDX)
S13 = 0.5_EB*(DUDZ+DWDX)
S23 = 0.5_EB*(DVDZ+DWDY)

STRAIN_RATE(IIG,JJG,KKG) = SQRT(2._EB*(S11**2 + S22**2 + S33**2 + 2._EB*(S12**2 + S13**2 + S23**2)))
ENDDO WALL_LOOP

END SUBROUTINE COMPUTE_STRAIN_RATE


SUBROUTINE VISCOSITY_BC(NM)

! Specify ghost cell values of the viscosity array MU

INTEGER, INTENT(IN) :: NM
REAL(EB) :: MU_OTHER,DP_OTHER,KRES_OTHER
INTEGER :: II,JJ,KK,IW,IIO,JJO,KKO,NOM,N_INT_CELLS
TYPE(WALL_TYPE),POINTER :: WC=>NULL()
TYPE(EXTERNAL_WALL_TYPE),POINTER :: EWC=>NULL()

CALL POINT_TO_MESH(NM)

! Mirror viscosity into solids and exterior boundary cells

WALL_LOOP: DO IW=1,N_EXTERNAL_WALL_CELLS
WC =>WALL(IW)
EWC=>EXTERNAL_WALL(IW)
IF (EWC%NOM==0) CYCLE WALL_LOOP
```

```fortran
II    = WC%ONE_D%II
JJ    = WC%ONE_D%JJ
KK    = WC%ONE_D%KK
NOM = EWC%NOM
MU_OTHER     = 0._EB
DP_OTHER     = 0._EB
KRES_OTHER = 0._EB
DO KKO=EWC%KKO_MIN,EWC%KKO_MAX
DO JJO=EWC%JJO_MIN,EWC%JJO_MAX
DO IIO=EWC%IIO_MIN,EWC%IIO_MAX
MU_OTHER = MU_OTHER + OMESH(NOM)%MU(IIO,JJO,KKO)
KRES_OTHER = KRES_OTHER + OMESH(NOM)%KRES(IIO,JJO,KKO)
IF (PREDICTOR) THEN
DP_OTHER = DP_OTHER + OMESH(NOM)%D(IIO,JJO,KKO)
ELSE
DP_OTHER = DP_OTHER + OMESH(NOM)%DS(IIO,JJO,KKO)
ENDIF
ENDDO
ENDDO
ENDDO
N_INT_CELLS = (EWC%IIO_MAX-EWC%IIO_MIN+1) * (EWC%JJO_MAX-EWC%JJO_MIN+1) * (EWC%KKO_MAX-EWC%KKO_MIN+1)
MU_OTHER = MU_OTHER/REAL(N_INT_CELLS,EB)
KRES_OTHER = KRES_OTHER/REAL(N_INT_CELLS,EB)
DP_OTHER = DP_OTHER/REAL(N_INT_CELLS,EB)
MU(II,JJ,KK) = MU_OTHER
KRES(II,JJ,KK) = KRES_OTHER
IF (PREDICTOR) THEN
D(II,JJ,KK) = DP_OTHER
ELSE
DS(II,JJ,KK) = DP_OTHER
ENDIF
ENDDO WALL_LOOP

END SUBROUTINE VISCOSITY_BC


SUBROUTINE VELOCITY_FLUX(T,DT,NM)

! Compute convective and diffusive terms of the momentum equations

USE MATH_FUNCTIONS, ONLY: EVALUATE_RAMP
USE COMPLEX_GEOMETRY, ONLY: CCIBM_VELOCITY_FLUX

!Sesa
USE MESH_POINTERS
USE penalization

INTEGER, INTENT(IN) :: NM
REAL(EB), INTENT(IN) :: T,DT
REAL(EB) :: MUX,MUY,MUZ,UP,UM,VP,VM,WP,WM,VTRM,OMXP,OMXM,OMYP,OMYM,OMZP,OMZM,TXYP,TXYM,TXZP,TXZM,TYZP,TYZM, &
DTXYDY,DTXZDZ,DTYZDZ,DTXYDX,DTXZDX,DTYZDY, &
DUDX,DVDY,DWDZ,DUDY,DUDZ,DVDX,DVDZ,DWDX,DWDY, &
VOMZ,WOMY,UOMY,VOMX,UOMZ,WOMX, &
RRHO,GX(0:IBAR_MAX),GY(0:IBAR_MAX),GZ(0:IBAR_MAX),TXXP,TXXM,TYYP,TYYM,TZZP,TZZM,DTXXDX,DTYYDY,DTZZDZ, &
DUMMY=0._EB
REAL(EB) :: VEG_UMAG
INTEGER :: I,J,K,IEXP,IEXM,IEYP,IEYM,IEZP,IEZM,IC,IC1,IC2
REAL(EB), POINTER, DIMENSION(:,:,:) :: TXY=>NULL(),TXZ=>NULL(),TYZ=>NULL(),OMX=>NULL(),OMY=>NULL(),OMZ=>NULL(), &
UU=>NULL(),VV=>NULL(),WW=>NULL(),RHOP=>NULL(),DP=>NULL()

!Sesa
integer :: ll
double precision :: xloc,yloc

CALL POINT_TO_MESH(NM)

IF (PREDICTOR) THEN
UU => U
VV => V
WW => W
DP => D
RHOP => RHO
ELSE
UU => US
VV => VS
WW => WS
DP => DS
RHOP => RHOS
ENDIF

TXY => WORK1
TXZ => WORK2
TYZ => WORK3
OMX => WORK4
OMY => WORK5
OMZ => WORK6

```

```
! Compute vorticity and stress tensor components

!$OMP PARALLEL DO PRIVATE(DUDY, DVDX, DUDZ, DWDX, DVDZ, DWDY, &
!$OMP& MUX, MUY, MUZ) SCHEDULE(STATIC)
DO K=0,KBAR
DO J=0,JBAR
DO I=0,IBAR
DUDY = RDYN(J)*(UU(I,J+1,K)-UU(I,J,K))
DVDX = RDXN(I)*(VV(I+1,J,K)-VV(I,J,K))
DUDZ = RDZN(K)*(UU(I,J,K+1)-UU(I,J,K))
DWDX = RDXN(I)*(WW(I+1,J,K)-WW(I,J,K))
DVDZ = RDZN(K)*(VV(I,J,K+1)-VV(I,J,K))
DWDY = RDYN(J)*(WW(I,J+1,K)-WW(I,J,K))
OMX(I,J,K) = DWDY - DVDZ
OMY(I,J,K) = DUDZ - DWDX
OMZ(I,J,K) = DVDX - DUDY
MUX = 0.25_EB*(MU(I,J+1,K)+MU(I,J,K)+MU(I,J,K+1)+MU(I,J+1,K+1))
MUY = 0.25_EB*(MU(I+1,J,K)+MU(I,J,K)+MU(I,J,K+1)+MU(I+1,J,K+1))
MUZ = 0.25_EB*(MU(I+1,J,K)+MU(I,J,K)+MU(I,J+1,K)+MU(I+1,J+1,K))
TXY(I,J,K) = MUZ*(DVDX + DUDY)
TXZ(I,J,K) = MUY*(DUDZ + DWDX)
TYZ(I,J,K) = MUX*(DVDZ + DWDY)
ENDDO
ENDDO
ENDDO
!$OMP END PARALLEL DO

! Wannier Flow (Stokes flow) test case

IF (PERIODIC_TEST==5) THEN
OMX=0._EB
OMY=0._EB
OMZ=0._EB
OME_E = -1.E6_EB
ENDIF

! Compute gravity components

IF (.NOT.SPATIAL_GRAVITY_VARIATION) THEN
GX(0:IBAR) = EVALUATE_RAMP(T,DUMMY,I_RAMP_GX)*GVEC(1)
GY(0:IBAR) = EVALUATE_RAMP(T,DUMMY,I_RAMP_GY)*GVEC(2)
GZ(0:IBAR) = EVALUATE_RAMP(T,DUMMY,I_RAMP_GZ)*GVEC(3)
ELSE
DO I=0,IBAR
GX(I) = EVALUATE_RAMP(X(I),DUMMY,I_RAMP_GX)*GVEC(1)
GY(I) = EVALUATE_RAMP(X(I),DUMMY,I_RAMP_GY)*GVEC(2)
GZ(I) = EVALUATE_RAMP(X(I),DUMMY,I_RAMP_GZ)*GVEC(3)
ENDDO
ENDIF

! Compute x-direction flux term FVX

!$OMP PARALLEL PRIVATE(WP, WM, VP, VM, UP, UM, &
!$OMP& OMXP, OMXM, OMYP, OMYM, OMZP, OMZM, &
!$OMP& TXZP, TXZM, TXYP, TXYM, TYZP, TYZM, &
!$OMP& IC, IEXP, IEXM, IEYP, IEYM, IEZP, IEZM, &
!$OMP& RRHO, DUDX, DVDY, DWDZ, VTRM)
!$OMP DO SCHEDULE(static) &
!$OMP& PRIVATE(WOMY, VOMZ, TXXP, TXXM, DTXXDX, DTXYDY, DTXZDZ)
DO K=1,KBAR
DO J=1,JBAR
DO I=0,IBAR
WP    = WW(I,J,K)    + WW(I+1,J,K)
WM    = WW(I,J,K-1)  + WW(I+1,J,K-1)
VP    = VV(I,J,K)    + VV(I+1,J,K)
VM    = VV(I,J-1,K)  + VV(I+1,J-1,K)
OMYP  = OMY(I,J,K)
OMYM  = OMY(I,J,K-1)
OMZP  = OMZ(I,J,K)
OMZM  = OMZ(I,J-1,K)
TXZP  = TXZ(I,J,K)
TXZM  = TXZ(I,J,K-1)
TXYP  = TXY(I,J,K)
TXYM  = TXY(I,J-1,K)
IC    = CELL_INDEX(I,J,K)
IEYP  = EDGE_INDEX(8,IC)
IEYM  = EDGE_INDEX(6,IC)
IEZP  = EDGE_INDEX(12,IC)
IEZM  = EDGE_INDEX(10,IC)
IF (OME_E(-1,IEYP)>-1.E5_EB) THEN
OMYP = OME_E(-1,IEYP)
TXZP = TAU_E(-1,IEYP)
ENDIF
IF (OME_E( 1,IEYM)>-1.E5_EB) THEN
OMYM = OME_E( 1,IEYM)
TXZM = TAU_E( 1,IEYM)
ENDIF
IF (OME_E(-2,IEZP)>-1.E5_EB) THEN
```

```
OMZP = OME_E(-2,IEZP)
TXYP = TAU_E(-2,IEZP)
ENDIF
IF (OME_E( 2,IEZM)>-1.E5_EB) THEN
OMZM = OME_E( 2,IEZM)
TXYM = TAU_E( 2,IEZM)
ENDIF
WOMY   = WP*OMYP + WM*OMYM
VOMZ   = VP*OMZP + VM*OMZM
RRHO   = 2._EB/(RHOP(I,J,K)+RHOP(I+1,J,K))
DVDY   = (VV(I+1,J,K)-VV(I+1,J-1,K))*RDY(J)
DWDZ   = (WW(I+1,J,K)-WW(I+1,J,K-1))*RDZ(K)
TXXP   = MU(I+1,J,K)*( FOTH*DP(I+1,J,K) - 2._EB*(DVDY+DWDZ) )
DVDY   = (VV(I,J,K)-VV(I,J-1,K))*RDY(J)
DWDZ   = (WW(I,J,K)-WW(I,J,K-1))*RDZ(K)
TXXM   = MU(I,J,K)   *( FOTH*DP(I,J,K)   - 2._EB*(DVDY+DWDZ) )
DTXXDX= RDXN(I)*(TXXP-TXXM)
DTXYDY= RDY(J)  *(TXYP-TXYM)
DTXZDZ= RDZ(K)  *(TXZP-TXZM)
VTRM   = DTXXDX + DTXYDY + DTXZDZ
FVX(I,J,K) = 0.25_EB*(WOMY - VOMZ) - GX(I) + RRHO*(GX(I)*RHO_0(K) - VTRM)
ENDDO
ENDDO
ENDDO
!$OMP END DO NOWAIT

! Compute y-direction flux term FVY

!$OMP DO SCHEDULE(static) &
!$OMP& PRIVATE(WOMX, UOMZ, TYYP, TYYM, DTXYDX, DTYYDY, DTYZDZ)
DO K=1,KBAR
DO J=0,JBAR
DO I=1,IBAR
UP     = UU(I,J,K)     + UU(I,J+1,K)
UM     = UU(I-1,J,K)  + UU(I-1,J+1,K)
WP     = WW(I,J,K)     + WW(I,J+1,K)
WM     = WW(I,J,K-1)  + WW(I,J+1,K-1)
OMXP   = OMX(I,J,K)
OMXM   = OMX(I,J,K-1)
OMZP   = OMZ(I,J,K)
OMZM   = OMZ(I-1,J,K)
TYZP   = TYZ(I,J,K)
TYZM   = TYZ(I,J,K-1)
TXYP   = TXY(I,J,K)
TXYM   = TXY(I-1,J,K)
IC     = CELL_INDEX(I,J,K)
IEXP   = EDGE_INDEX(4,IC)
IEXM   = EDGE_INDEX(2,IC)
IEZP   = EDGE_INDEX(12,IC)
IEZM   = EDGE_INDEX(11,IC)
IF (OME_E(-2,IEXP)>-1.E5_EB) THEN
OMXP = OME_E(-2,IEXP)
TYZP = TAU_E(-2,IEXP)
ENDIF
IF (OME_E( 2,IEXM)>-1.E5_EB) THEN
OMXM = OME_E( 2,IEXM)
TYZM = TAU_E( 2,IEXM)
ENDIF
IF (OME_E(-1,IEZP)>-1.E5_EB) THEN
OMZP = OME_E(-1,IEZP)
TXYP = TAU_E(-1,IEZP)
ENDIF
IF (OME_E( 1,IEZM)>-1.E5_EB) THEN
OMZM = OME_E( 1,IEZM)
TXYM = TAU_E( 1,IEZM)
ENDIF
WOMX   = WP*OMXP + WM*OMXM
UOMZ   = UP*OMZP + UM*OMZM
RRHO   = 2._EB/(RHOP(I,J,K)+RHOP(I,J+1,K))
DUDX   = (UU(I,J+1,K)-UU(I-1,J+1,K))*RDX(I)
DWDZ   = (WW(I,J+1,K)-WW(I,J+1,K-1))*RDZ(K)
TYYP   = MU(I,J+1,K)*( FOTH*DP(I,J+1,K) - 2._EB*(DUDX+DWDZ) )
DUDX   = (UU(I,J,K)-UU(I-1,J,K))*RDX(I)
DWDZ   = (WW(I,J,K)-WW(I,J,K-1))*RDZ(K)
TYYM   = MU(I,J,K)   *( FOTH*DP(I,J,K)   - 2._EB*(DUDX+DWDZ) )
DTXYDX= RDX(I)  *(TXYP-TXYM)
DTYYDY= RDYN(J)*(TYYP-TYYM)
DTYZDZ= RDZ(K)  *(TYZP-TYZM)
VTRM   = DTXYDX + DTYYDY + DTYZDZ
FVY(I,J,K) = 0.25_EB*(UOMZ - WOMX) - GY(I) + RRHO*(GY(I)*RHO_0(K) - VTRM)
ENDDO
ENDDO
ENDDO
!$OMP END DO NOWAIT

! Compute z-direction flux term FVZ

!$OMP DO SCHEDULE(static) &
```

```fortran
!$OMP& PRIVATE(UOMY, VOMX, TZZP, TZZM, DTXZDX, DTYZDY, DTZZDZ)
DO K=0,KBAR
DO J=1,JBAR
DO I=1,IBAR
UP    = UU(I,J,K)    + UU(I,J,K+1)
UM    = UU(I-1,J,K)  + UU(I-1,J,K+1)
VP    = VV(I,J,K)    + VV(I,J,K+1)
VM    = VV(I,J-1,K)  + VV(I,J-1,K+1)
OMYP  = OMY(I,J,K)
OMYM  = OMY(I-1,J,K)
OMXP  = OMX(I,J,K)
OMXM  = OMX(I,J-1,K)
TXZP  = TXZ(I,J,K)
TXZM  = TXZ(I-1,J,K)
TYZP  = TYZ(I,J,K)
TYZM  = TYZ(I,J-1,K)
IC    = CELL_INDEX(I,J,K)
IEXP  = EDGE_INDEX(4,IC)
IEXM  = EDGE_INDEX(3,IC)
IEYP  = EDGE_INDEX(8,IC)
IEYM  = EDGE_INDEX(7,IC)
IF (OME_E(-1,IEXP)>-1.E5_EB) THEN
OMXP = OME_E(-1,IEXP)
TYZP = TAU_E(-1,IEXP)
ENDIF
IF (OME_E( 1,IEXM)>-1.E5_EB) THEN
OMXM = OME_E( 1,IEXM)
TYZM = TAU_E( 1,IEXM)
ENDIF
IF (OME_E(-2,IEYP)>-1.E5_EB) THEN
OMYP = OME_E(-2,IEYP)
TXZP = TAU_E(-2,IEYP)
ENDIF
IF (OME_E( 2,IEYM)>-1.E5_EB) THEN
OMYM = OME_E( 2,IEYM)
TXZM = TAU_E( 2,IEYM)
ENDIF
UOMY  = UP*OMYP + UM*OMYM
VOMX  = VP*OMXP + VM*OMXM
RRHO  = 2._EB/(RHOP(I,J,K)+RHOP(I,J,K+1))
DUDX  = (UU(I,J,K+1)-UU(I-1,J,K+1))*RDX(I)
DVDY  = (VV(I,J,K+1)-VV(I,J-1,K+1))*RDY(J)
TZZP  = MU(I,J,K+1)*( FOTH*DP(I,J,K+1) - 2._EB*(DUDX+DVDY) )
DUDX  = (UU(I,J,K)-UU(I-1,J,K))*RDX(I)
DVDY  = (VV(I,J,K)-VV(I,J-1,K))*RDY(J)
TZZM  = MU(I,J,K)  *( FOTH*DP(I,J,K)   - 2._EB*(DUDX+DVDY) )
DTXZDX= RDX(I) *(TXZP-TXZM)
DTYZDY= RDY(J) *(TYZP-TYZM)
DTZZDZ= RDZN(K)*(TZZP-TZZM)
VTRM  = DTXZDX + DTYZDY + DTZZDZ
FVZ(I,J,K) = 0.25_EB*(VOMX - UOMY) - GZ(I) + RRHO*(GZ(I)*0.5_EB*(RHO_0(K)+RHO_0(K+1)) - VTRM)
ENDDO
ENDDO
ENDDO
!$OMP END DO NOWAIT
!$OMP END PARALLEL

IF (EVACUATION_ONLY(NM)) THEN
FVZ = 0._EB
RETURN
END IF

! Additional force terms

IF (ANY(MEAN_FORCING))                   CALL MOMENTUM_NUDGING     ! Mean forcing
IF (ANY(ABS(FVEC)>TWO_EPSILON_EB))  CALL DIRECT_FORCE         ! Direct force
IF (ANY(ABS(OVEC)>TWO_EPSILON_EB))  CALL CORIOLIS_FORCE       ! Coriolis force

!Sesa
do ll=1,size(pendat,1)

cntr=0

if (pendat(ll,14).ge.0) then
if(pendat(ll,14).le.floor(T)) then
if(ll.le.4) then

DO K=0,KBAR
DO J=0,JBAR
DO I=0,IBAR
!penalisation region start
if (pendat(ll,13).eq.1) then
if ((Z(K).le.pendat(ll,9)).and.(Z(K).ge.pendat(ll,8))) then
if ((Y(J).le.pendat(ll,7)).and.(Y(J).ge.pendat(ll,6))) then
if ((X(I).le.pendat(ll,5)).and.(X(I).ge.pendat(ll,4))) then

cntr=cntr+1

```

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

endif
endif
endif
else
!blending region start
if ((Z(K).le.pendat(ll,9)).and.(Z(K).ge.pendat(ll,8))) then
if ((Y(J).le.pendat(ll,7)).and.(Y(J).ge.pendat(ll,6))) then
if ((X(I).le.pendat(ll,5)).and.(X(I).ge.pendat(ll,4))) then

cntr=cntr+1

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

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

FVZ(I,J,K) = FVZ(I,J,K) +(pendat(ll,13) + pendat(ll,10)*X(I)-((pendat(ll,10)*(1-pendat(ll,10))*pendat(ll,5))/2d0) &
-(pendat(ll,10)*(1+pendat(ll,10))*pendat(ll,4)/2d0 ))*(WW(I,J,K)-pendat(ll,14+size(penU0)+size(penV0)+cntr))/pendat(ll,2) &
+(pendat(ll,13) + pendat(ll,11)*Y(J)-((pendat(ll,11)*(1-pendat(ll,11))*pendat(ll,7))/2d0) &
-(pendat(ll,11)*(1+pendat(ll,11))*pendat(ll,6)/2d0 ))*(WW(I,J,K)-pendat(ll,14+size(penU0)+size(penV0)+cntr))/pendat(ll,2) &
+ (pendat(ll,13) + pendat(ll,12)*Z(K)-((pendat(ll,12)*(1-pendat(ll,12))*pendat(ll,9))/2d0) &
-(pendat(ll,12)*(1+pendat(ll,12))*pendat(ll,8)/2d0 ))*(WW(I,J,K)-pendat(ll,14+size(penU0)+size(penV0)+cntr))/pendat(ll,2)

endif
endif
endif
endif
ENDDO
ENDDO
ENDDO

endif

if((ll.gt.4) .AND. pendat(ll,14).le. floor(T)) then

if(mod(ll,4).eq. 1) then
pendat(ll-1,14)=-1
pendat(ll-2,14)=-1
pendat(ll-3,14)=-1
pendat(ll-4,14)=-1
endif

DO K=0,KBAR
DO J=0,JBAR
DO I=0,IBAR

!penalisation region starts
if (pendat(ll,13).eq.1) then

if ((Z(K).le.pendat(ll,9)).and.(Z(K).ge.pendat(ll,8))) then
if ((Y(J).le.pendat(ll,7)).and.(Y(J).ge.pendat(ll,6))) then
if ((X(I).le.pendat(ll,5)).and.(X(I).ge.pendat(ll,4))) then

cntr=cntr+1

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

endif
endif
endif
else
!blending region starts
if ((Z(K).le.pendat(ll,9)).and.(Z(K).ge.pendat(ll,8))) then
if ((Y(J).le.pendat(ll,7)).and.(Y(J).ge.pendat(ll,6))) then
if ((X(I).le.pendat(ll,5)).and.(X(I).ge.pendat(ll,4))) then

cntr=cntr+1
```

```
FVX(I,J,K) = FVX(I,J,K) + (pendat(ll,13) + pendat(ll,10)*X(I)-((pendat(ll,10)*(1-pendat(ll,10))*pendat(ll,5))/2d0) &
-(pendat(ll,10)*(1+pendat(ll,10))*pendat(ll,4)/2d0))*(UU(I,J,K)-pendat(ll,14+cntr))/pendat(ll,2) &
+ (pendat(ll,13) + pendat(ll,11)*Y(J)-((pendat(ll,11)*(1-pendat(ll,11))*pendat(ll,6))/2d0) &
-(pendat(ll,11)*(1+pendat(ll,11))*pendat(ll,7)/2d0 ))*(UU(I,J,K)-pendat(ll,14+cntr))/pendat(ll,2) &
+ (pendat(ll,13) + pendat(ll,12)*Z(K)-((pendat(ll,12)*(1-pendat(ll,12))*pendat(ll,8))/2d0) &
-(pendat(ll,12)*(1+pendat(ll,12))*pendat(ll,9)/2d0 ))*(UU(I,J,K)-pendat(ll,14+cntr))/pendat(ll,2)

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

FVZ(I,J,K) = FVZ(I,J,K) +(pendat(ll,13) + pendat(ll,10)*X(I)-((pendat(ll,10)*(1-pendat(ll,10))*pendat(ll,5))/2d0) &
-(pendat(ll,10)*(1+pendat(ll,10))*pendat(ll,4)/2d0 ))*(WW(I,J,K)-pendat(ll,14+size(penU0)+size(penV0)+cntr))/pendat(ll,2) &
+(pendat(ll,13) + pendat(ll,11)*Y(J)-((pendat(ll,11)*(1-pendat(ll,11))*pendat(ll,6))/2d0) &
-(pendat(ll,11)*(1+pendat(ll,11))*pendat(ll,7)/2d0 ))*(WW(I,J,K)-pendat(ll,14+size(penU0)+size(penV0)+cntr))/pendat(ll,2) &
+ (pendat(ll,13) + pendat(ll,12)*Z(K)-((pendat(ll,12)*(1-pendat(ll,12))*pendat(ll,8))/2d0) &
-(pendat(ll,12)*(1+pendat(ll,12))*pendat(ll,9)/2d0 ))*(WW(I,J,K)-pendat(ll,14+size(penU0)+size(penV0)+cntr))/pendat(ll,2)

endif
endif
endif

endif
ENDDO
ENDDO
ENDDO
endif

endif
endif
enddo

TRUNK_IF: IF (trunks) THEN
do ll=1,ntrunks
DO I=0,IBAR
DO J=0,JBAR
xloc=trnk_loc(1,ll)
yloc=trnk_loc(2,ll)
if ((X(I).eq.xloc)) then
if ((Y(J).eq.yloc)) then

do k=0,KBAR
if ((Z(K).gt.trnk_min).and.(Z(K).lt.trnk_max)) then
FVX(I,J,K)=FVX(I,J,K)+UU(I,J,K)/eta
FVY(I,J,K)=FVY(I,J,K)+VV(I,J,K)/eta
FVZ(I,J,K)=FVZ(I,J,K)+WW(I,J,K)/eta
endif
enddo

endif
endif
ENDDO
ENDDO
enddo
ENDIF TRUNK_IF


IF (WFDS_BNDRYFUEL)            CALL VEGETATION_DRAG             ! Surface vegetation drag
IF (PATCH_VELOCITY)            CALL PATCH_VELOCITY_FLUX(DT,NM)  ! Specified patch velocity
IF (CC_IBM)                    CALL CCIBM_VELOCITY_FLUX(DT,NM)  ! Direct-forcing Immersed Boundary Method
IF (PERIODIC_TEST==7)          CALL MMS_VELOCITY_FLUX(NM,T)     ! Source term in manufactured solution

CONTAINS

SUBROUTINE MOMENTUM_NUDGING

! Add a force vector to the momentum equation that moves the flow field towards the direction of the mean flow.

REAL(EB) :: UBAR,VBAR,WBAR,INTEGRAL,SUM_VOLUME,VC,UMEAN,VMEAN,WMEAN,DU_FORCING,DV_FORCING,DW_FORCING,DT_LOC
INTEGER  :: NSC,I_LO,J_LO,I_HI,J_HI

DT_LOC = MAX(DT,DT_MEAN_FORCING)
NSC    = SPONGE_CELLS

MEAN_FORCING_X: IF (MEAN_FORCING(1)) THEN
SELECT_RAMP_U: SELECT CASE(I_RAMP_U0_Z)
CASE(0) SELECT_RAMP_U
INTEGRAL = 0._EB
```

```
SUM_VOLUME = 0._EB
DO K=1,KBAR
DO J=1,JBAR
DO I=0,IBAR
IC1 = CELL_INDEX(I,J,K)
IC2 = CELL_INDEX(I+1,J,K)
IF (SOLID(IC1)) CYCLE
IF (SOLID(IC2)) CYCLE
IF (.NOT.MEAN_FORCING_CELL(I,J,K)   ) CYCLE
IF (.NOT.MEAN_FORCING_CELL(I+1,J,K)) CYCLE
VC = DXN(I)*DY(J)*DZ(K)
INTEGRAL = INTEGRAL + UU(I,J,K)*VC
SUM_VOLUME = SUM_VOLUME + VC
ENDDO
ENDDO
ENDDO
IF (SUM_VOLUME>TWO_EPSILON_EB) THEN
UMEAN = INTEGRAL/SUM_VOLUME
ELSE
UMEAN = 0._EB
ENDIF
UBAR = U0*EVALUATE_RAMP(T,DUMMY,I_RAMP_U0_T)
DU_FORCING = (UBAR-UMEAN)/DT_LOC
DO K=1,KBAR
DO J=1,JBAR
DO I=0,IBAR
IF (.NOT.MEAN_FORCING_CELL(I,J,K)   ) CYCLE
IF (.NOT.MEAN_FORCING_CELL(I+1,J,K)) CYCLE
FVX(I,J,K) = FVX(I,J,K) - DU_FORCING
ENDDO
ENDDO
ENDDO
CASE(1:) SELECT_RAMP_U
K_LOOP_U: DO K=1,KBAR
INTEGRAL = 0._EB
SUM_VOLUME = 0._EB
DO J=1,JBAR
DO I=0,IBAR
IC1 = CELL_INDEX(I,J,K)
IC2 = CELL_INDEX(I+1,J,K)
IF (SOLID(IC1)) CYCLE
IF (SOLID(IC2)) CYCLE
VC = DXN(I)*DY(J)*DZ(K)
INTEGRAL = INTEGRAL + UU(I,J,K)*VC
SUM_VOLUME = SUM_VOLUME + VC
ENDDO
ENDDO
IF (SUM_VOLUME>TWO_EPSILON_EB) THEN
UMEAN = INTEGRAL/SUM_VOLUME
ELSE
! this can happen if all cells in a given row, k, are solid
UMEAN = 0._EB
ENDIF
UBAR = U0*EVALUATE_RAMP(T,DUMMY,I_RAMP_U0_T)*EVALUATE_RAMP(ZC(K),DUMMY,I_RAMP_U0_Z)
DU_FORCING = (UBAR-UMEAN)/DT_LOC
! Apply the average force term to bulk of domain, and apply more aggressive forcing at boundary
I_LO = 0
I_HI = IBAR
IF (APPLY_SPONGE_LAYER(1)) THEN
FVX(0:NSC-1,:,K)           = FVX(0:NSC-1,:,K)           - (UBAR-UU(0:NSC-1,:,K))/DT_LOC
I_LO = NSC
ENDIF
IF (APPLY_SPONGE_LAYER(-1)) THEN
FVX(IBAR-NSC+1:IBAR,:,K) = FVX(IBAR-NSC+1:IBAR,:,K) - (UBAR-UU(IBAR-NSC+1:IBAR,:,K))/DT_LOC
I_HI = IBAR-NSC
ENDIF
FVX(I_LO:I_HI,:,K) = FVX(I_LO:I_HI,:,K) - DU_FORCING
ENDDO K_LOOP_U
END SELECT SELECT_RAMP_U
ENDIF MEAN_FORCING_X

MEAN_FORCING_Y: IF (MEAN_FORCING(2)) THEN
SELECT_RAMP_V: SELECT CASE(I_RAMP_V0_Z)
CASE(0) SELECT_RAMP_V
INTEGRAL = 0._EB
SUM_VOLUME = 0._EB
DO K=1,KBAR
DO J=0,JBAR
DO I=1,IBAR
IC1 = CELL_INDEX(I,J,K)
IC2 = CELL_INDEX(I,J+1,K)
IF (SOLID(IC1)) CYCLE
IF (SOLID(IC2)) CYCLE
IF (.NOT.MEAN_FORCING_CELL(I,J,K)   ) CYCLE
IF (.NOT.MEAN_FORCING_CELL(I,J+1,K)) CYCLE
VC = DX(I)*DYN(J)*DZ(K)
INTEGRAL = INTEGRAL + VV(I,J,K)*VC
SUM_VOLUME = SUM_VOLUME + VC
```

```
ENDDO
ENDDO
ENDDO
IF (SUM_VOLUME>TWO_EPSILON_EB) THEN
VMEAN = INTEGRAL/SUM_VOLUME
ELSE
VMEAN = 0._EB
ENDIF
VBAR = V0*EVALUATE_RAMP(T,DUMMY,I_RAMP_V0_T)
DV_FORCING = (VBAR-VMEAN)/DT_LOC
DO K=1,KBAR
DO J=0,JBAR
DO I=1,IBAR
IF (.NOT.MEAN_FORCING_CELL(I,J,K)   ) CYCLE
IF (.NOT.MEAN_FORCING_CELL(I,J+1,K)) CYCLE
FVY(I,J,K) = FVY(I,J,K) - DV_FORCING
ENDDO
ENDDO
ENDDO
CASE(1:) SELECT_RAMP_V
K_LOOP_V: DO K=1,KBAR
INTEGRAL = 0._EB
SUM_VOLUME = 0._EB
DO J=0,JBAR
DO I=1,IBAR
IC1 = CELL_INDEX(I,J,K)
IC2 = CELL_INDEX(I,J+1,K)
IF (SOLID(IC1)) CYCLE
IF (SOLID(IC2)) CYCLE
VC = DX(I)*DYN(J)*DZ(K)
INTEGRAL = INTEGRAL + VV(I,J,K)*VC
SUM_VOLUME = SUM_VOLUME + VC
ENDDO
ENDDO
IF (SUM_VOLUME>TWO_EPSILON_EB) THEN
VMEAN = INTEGRAL/SUM_VOLUME
ELSE
VMEAN = 0._EB
ENDIF
VBAR = V0*EVALUATE_RAMP(T,DUMMY,I_RAMP_V0_T)*EVALUATE_RAMP(ZC(K),DUMMY,I_RAMP_V0_Z)
DV_FORCING = (VBAR-VMEAN)/DT_LOC
! Apply the average force term to bulk of domain, and apply more aggressive forcing at boundary
J_LO = 0
J_HI = JBAR
IF (APPLY_SPONGE_LAYER(2)) THEN
FVY(:,0:NSC-1,K)           = FVY(:,0:NSC-1,K)           - (VBAR-VV(:,0:NSC-1,K))/DT_LOC
J_LO = NSC
ENDIF
IF (APPLY_SPONGE_LAYER(-2)) THEN
FVY(:,JBAR-NSC+1:JBAR,K) = FVY(:,JBAR-NSC+1:JBAR,K) - (VBAR-VV(:,JBAR-NSC+1:JBAR,K))/DT_LOC
J_HI = JBAR-NSC
ENDIF
FVY(:,J_LO:J_HI,K) = FVY(:,J_LO:J_HI,K) - DV_FORCING
ENDDO K_LOOP_V
END SELECT SELECT_RAMP_V
ENDIF MEAN_FORCING_Y

MEAN_FORCING_Z: IF (MEAN_FORCING(3)) THEN
SELECT_RAMP_W: SELECT CASE(I_RAMP_W0_Z)
CASE(0) SELECT_RAMP_W
INTEGRAL = 0._EB
SUM_VOLUME = 0._EB
DO K=0,KBAR
DO J=1,JBAR
DO I=1,IBAR
IC1 = CELL_INDEX(I,J,K)
IC2 = CELL_INDEX(I,J,K+1)
IF (SOLID(IC1)) CYCLE
IF (SOLID(IC2)) CYCLE
IF (.NOT.MEAN_FORCING_CELL(I,J,K)   ) CYCLE
IF (.NOT.MEAN_FORCING_CELL(I,J,K+1)) CYCLE
VC = DX(I)*DY(J)*DZN(K)
INTEGRAL = INTEGRAL + WW(I,J,K)*VC
SUM_VOLUME = SUM_VOLUME + VC
ENDDO
ENDDO
ENDDO
IF (SUM_VOLUME>TWO_EPSILON_EB) THEN
WMEAN = INTEGRAL/SUM_VOLUME
ELSE
WMEAN = 0._EB
ENDIF
WBAR = W0*EVALUATE_RAMP(T,DUMMY,I_RAMP_W0_T)
DW_FORCING = (WBAR-WMEAN)/DT_LOC
DO K=0,KBAR
DO J=1,JBAR
DO I=1,IBAR
IF (.NOT.MEAN_FORCING_CELL(I,J,K)   ) CYCLE
```

```
IF (.NOT.MEAN_FORCING_CELL(I,J,K+1)) CYCLE
FVZ(I,J,K) = FVZ(I,J,K) - DW_FORCING
ENDDO
ENDDO
ENDDO
CASE(1:)
K_LOOP_W: DO K=0,KBAR
INTEGRAL = 0._EB
SUM_VOLUME = 0._EB
DO J=1,JBAR
DO I=1,IBAR
IC1 = CELL_INDEX(I,J,K)
IC2 = CELL_INDEX(I,J,K+1)
IF (SOLID(IC1)) CYCLE
IF (SOLID(IC2)) CYCLE
VC = DX(I)*DY(J)*DZN(K)
INTEGRAL = INTEGRAL + WW(I,J,K)*VC
SUM_VOLUME = SUM_VOLUME + VC
ENDDO
ENDDO
IF (SUM_VOLUME>TWO_EPSILON_EB) THEN
WMEAN = INTEGRAL/SUM_VOLUME
ELSE
WMEAN = 0._EB
ENDIF
WBAR = W0*EVALUATE_RAMP(T,DUMMY,I_RAMP_W0_T)*EVALUATE_RAMP(Z(K),DUMMY,I_RAMP_W0_Z)
DW_FORCING = (WBAR-WMEAN)/DT_LOC
! Apply the average force term to bulk of domain, and apply more aggressive forcing at boundary
IF (APPLY_SPONGE_LAYER(-3) .AND. K==KBAR) THEN
DO J=1,JBAR
DO I=1,IBAR
FVZ(I,J,K) = FVZ(I,J,K) - (WBAR-WW(I,J,K))/DT_LOC
ENDDO
ENDDO
ELSE
FVZ(:,:,K) = FVZ(:,:,K) - DW_FORCING
ENDIF
ENDDO K_LOOP_W
END SELECT SELECT_RAMP_W
ENDIF MEAN_FORCING_Z

END SUBROUTINE MOMENTUM_NUDGING


SUBROUTINE DIRECT_FORCE()
REAL(EB) :: TIME_RAMP_FACTOR

TIME_RAMP_FACTOR = EVALUATE_RAMP(T,DUMMY,I_RAMP_FVX_T)
!$OMP PARALLEL DO PRIVATE(RRHO) SCHEDULE(STATIC)
DO K=1,KBAR
DO J=1,JBAR
DO I=0,IBAR
RRHO = 2._EB/(RHOP(I,J,K)+RHOP(I+1,J,K))
FVX(I,J,K) = FVX(I,J,K) - RRHO*FVEC(1)*TIME_RAMP_FACTOR
ENDDO
ENDDO
ENDDO
!$OMP END PARALLEL DO

TIME_RAMP_FACTOR = EVALUATE_RAMP(T,DUMMY,I_RAMP_FVY_T)
!$OMP PARALLEL DO PRIVATE(RRHO) SCHEDULE(STATIC)
DO K=1,KBAR
DO J=0,JBAR
DO I=1,IBAR
RRHO = 2._EB/(RHOP(I,J,K)+RHOP(I,J+1,K))
FVY(I,J,K) = FVY(I,J,K) - RRHO*FVEC(2)*TIME_RAMP_FACTOR
ENDDO
ENDDO
ENDDO
!$OMP END PARALLEL DO

TIME_RAMP_FACTOR = EVALUATE_RAMP(T,DUMMY,I_RAMP_FVZ_T)
!$OMP PARALLEL DO PRIVATE(RRHO) SCHEDULE(STATIC)
DO K=0,KBAR
DO J=1,JBAR
DO I=1,IBAR
RRHO = 2._EB/(RHOP(I,J,K)+RHOP(I,J,K+1))
FVZ(I,J,K) = FVZ(I,J,K) - RRHO*FVEC(3)*TIME_RAMP_FACTOR
ENDDO
ENDDO
ENDDO
!$OMP END PARALLEL DO

END SUBROUTINE DIRECT_FORCE


SUBROUTINE CORIOLIS_FORCE()

```

```
REAL(EB), POINTER, DIMENSION(:,:,:) :: UP=>NULL(),VP=>NULL(),WP=>NULL()
REAL(EB) :: UBAR,VBAR,WBAR
INTEGER :: II,JJ,KK,IW
TYPE(WALL_TYPE), POINTER :: WC=>NULL()

! Velocities relative to the p-cell center (same work done in Deardorff eddy viscosity)

UP => WORK7
VP => WORK8
WP => WORK9
UP=0._EB
VP=0._EB
WP=0._EB

!$OMP PARALLEL DO SCHEDULE(static)
DO K=1,KBAR
DO J=1,JBAR
DO I=1,IBAR
 IF (SOLID(CELL_INDEX(I,J,K))) CYCLE
UP(I,J,K) = 0.5_EB*(UU(I,J,K) + UU(I-1,J,K))
VP(I,J,K) = 0.5_EB*(VV(I,J,K) + VV(I,J-1,K))
WP(I,J,K) = 0.5_EB*(WW(I,J,K) + WW(I,J,K-1))
ENDDO
ENDDO
ENDDO
!$OMP END PARALLEL DO

DO IW=1,N_EXTERNAL_WALL_CELLS
WC=>WALL(IW)
II = WC%ONE_D%II
JJ = WC%ONE_D%JJ
KK = WC%ONE_D%KK
UP(II,JJ,KK) = U_GHOST(IW)
VP(II,JJ,KK) = V_GHOST(IW)
WP(II,JJ,KK) = W_GHOST(IW)
ENDDO

! x momentum

!$OMP PARALLEL DO PRIVATE(VBAR,WBAR) SCHEDULE(STATIC)
DO K=1,KBAR
DO J=1,JBAR
DO I=0,IBAR
VBAR = 0.5_EB*(VP(I,J,K)+VP(I+1,J,K))
WBAR = 0.5_EB*(WP(I,J,K)+WP(I+1,J,K))
FVX(I,J,K) = FVX(I,J,K) + 2._EB*(OVEC(2)*WBAR-OVEC(3)*VBAR)
ENDDO
ENDDO
ENDDO
!$OMP END PARALLEL DO

! y momentum

!$OMP PARALLEL DO PRIVATE(UBAR,WBAR) SCHEDULE(STATIC)
DO K=1,KBAR
DO J=0,JBAR
DO I=1,IBAR
UBAR = 0.5_EB*(UP(I,J,K)+UP(I,J+1,K))
WBAR = 0.5_EB*(WP(I,J,K)+WP(I,J+1,K))
FVY(I,J,K) = FVY(I,J,K) + 2._EB*(OVEC(3)*UBAR - OVEC(1)*WBAR)
ENDDO
ENDDO
ENDDO
!$OMP END PARALLEL DO

! z momentum

!$OMP PARALLEL DO PRIVATE(UBAR,VBAR) SCHEDULE(STATIC)
DO K=0,KBAR
DO J=1,JBAR
DO I=1,IBAR
UBAR = 0.5_EB*(UP(I,J,K)+UP(I,J,K+1))
VBAR = 0.5_EB*(VP(I,J,K)+VP(I,J,K+1))
FVZ(I,J,K) = FVZ(I,J,K) + 2._EB*(OVEC(1)*VBAR - OVEC(2)*UBAR)
ENDDO
ENDDO
ENDDO
!$OMP END PARALLEL DO

END SUBROUTINE CORIOLIS_FORCE


SUBROUTINE VEGETATION_DRAG()

VEG_DRAG(0,:) = VEG_DRAG(1,:)
K=1
DO J=1,JBAR
DO I=0,IBAR
```

```
VEG_UMAG = SQRT(UU(I,J,K)**2 + VV(I,J,K)**2 + WW(I,J,K)**2) ! VEG_UMAG=2._EB*KRES(I,J,K)
FVX(I,J,K) = FVX(I,J,K) + VEG_DRAG(I,J)*VEG_UMAG*UU(I,J,K)
ENDDO
ENDDO

VEG_DRAG(:,0) = VEG_DRAG(:,1)
DO J=0,JBAR
DO I=1,IBAR
VEG_UMAG = SQRT(UU(I,J,K)**2 + VV(I,J,K)**2 + WW(I,J,K)**2)
FVY(I,J,K) = FVY(I,J,K) + VEG_DRAG(I,J)*VEG_UMAG*VV(I,J,K)
ENDDO
ENDDO

DO J=1,JBAR
DO I=1,IBAR
VEG_UMAG = SQRT(UU(I,J,K)**2 + VV(I,J,K)**2 + WW(I,J,K)**2)
FVZ(I,J,K) = FVZ(I,J,K) + VEG_DRAG(I,J)*VEG_UMAG*WW(I,J,K)
ENDDO
ENDDO

END SUBROUTINE VEGETATION_DRAG

END SUBROUTINE VELOCITY_FLUX


SUBROUTINE MMS_VELOCITY_FLUX(NM,T)

! Shunn et al., JCP (2012) prob 3

USE MANUFACTURED_SOLUTIONS, ONLY: VD2D_MMS_U_SRC_3,VD2D_MMS_V_SRC_3
INTEGER, INTENT(IN) :: NM
REAL(EB), INTENT(IN) :: T
INTEGER :: I,J,K
REAL(EB), POINTER, DIMENSION(:,:,:) :: UU=>NULL(),WW=>NULL()

CALL POINT_TO_MESH(NM)

IF (PREDICTOR) THEN
UU=>U
WW=>W
ELSE
UU=>US
WW=>WS
ENDIF

DO K=1,KBAR
DO J=1,JBAR
DO I=0,IBAR
FVX(I,J,K) = FVX(I,J,K) - VD2D_MMS_U_SRC_3(X(I),ZC(K),T)
ENDDO
ENDDO
ENDDO

DO K=0,KBAR
DO J=1,JBAR
DO I=1,IBAR
FVZ(I,J,K) = FVZ(I,J,K) - VD2D_MMS_V_SRC_3(XC(I),Z(K),T)
ENDDO
ENDDO
ENDDO

END SUBROUTINE MMS_VELOCITY_FLUX


SUBROUTINE VELOCITY_FLUX_CYLINDRICAL(T,NM)

! Compute convective and diffusive terms for 2D axisymmetric

USE MATH_FUNCTIONS, ONLY: EVALUATE_RAMP
REAL(EB) :: T,DMUDX
INTEGER :: I0
INTEGER, INTENT(IN) :: NM
REAL(EB) :: MUY,UP,UM,WP,WM,VTRM,DTXZDZ,DTXZDX,DUDX,DWDZ,DUDZ,DWDX,WOMY,UOMY,OMYP,OMYM,TXZP,TXZM, &
AH,RRHO,GX,GZ,TXXP,TXXM,TZZP,TZZM,DTXXDX,DTZZDZ,DUMMY=0._EB
INTEGER :: I,J,K,IEYP,IEYM,IC
REAL(EB), POINTER, DIMENSION(:,:,:) :: TXZ=>NULL(),OMY=>NULL(),UU=>NULL(),WW=>NULL(),RHOP=>NULL(),DP=>NULL()

CALL POINT_TO_MESH(NM)

IF (PREDICTOR) THEN
UU => U
WW => W
DP => D
RHOP => RHO
ELSE
UU => US
WW => WS
DP => DS
```

```fortran
RHOP => RHOS
ENDIF

TXZ => WORK2
OMY => WORK5

! Compute vorticity and stress tensor components

DO K=0,KBAR
DO J=0,JBAR
DO I=0,IBAR
DUDZ = RDZN(K)*(UU(I,J,K+1)-UU(I,J,K))
DWDX = RDXN(I)*(WW(I+1,J,K)-WW(I,J,K))
OMY(I,J,K) = DUDZ - DWDX
MUY = 0.25_EB*(MU(I+1,J,K)+MU(I,J,K)+MU(I,J,K+1)+MU(I+1,J,K+1))
TXZ(I,J,K) = MUY*(DUDZ + DWDX)
ENDDO
ENDDO
ENDDO

! Compute gravity components

GX  = 0._EB
GZ  = EVALUATE_RAMP(T,DUMMY,I_RAMP_GZ)*GVEC(3)

! Compute r-direction flux term FVX

IF (ABS(XS)<=TWO_EPSILON_EB) THEN
I0 = 1
ELSE
I0 = 0
ENDIF

J = 1

DO K= 1,KBAR
DO I=I0,IBAR
WP     = WW(I,J,K)    + WW(I+1,J,K)
WM     = WW(I,J,K-1) + WW(I+1,J,K-1)
OMYP   = OMY(I,J,K)
OMYM   = OMY(I,J,K-1)
TXZP   = TXZ(I,J,K)
TXZM   = TXZ(I,J,K-1)
IC     = CELL_INDEX(I,J,K)
IEYP   = EDGE_INDEX(8,IC)
IEYM   = EDGE_INDEX(6,IC)
IF (OME_E(-1,IEYP)>-1.E5_EB) THEN
OMYP = OME_E(-1,IEYP)
TXZP = TAU_E(-1,IEYP)
ENDIF
IF (OME_E( 1,IEYM)>-1.E5_EB) THEN
OMYM = OME_E( 1,IEYM)
TXZM = TAU_E( 1,IEYM)
ENDIF
WOMY   = WP*OMYP + WM*OMYM
RRHO   = 2._EB/(RHOP(I,J,K)+RHOP(I+1,J,K))
AH     = RHO_0(K)*RRHO - 1._EB
DWDZ   = (WW(I+1,J,K)-WW(I+1,J,K-1))*RDZ(K)
TXXP   = MU(I+1,J,K)*( FOTH*DP(I+1,J,K) - 2._EB*DWDZ )
DWDZ   = (WW(I,J,K)-WW(I,J,K-1))*RDZ(K)
TXXM   = MU(I,J,K)   *( FOTH*DP(I,J,K) -2._EB*DWDZ )
DTXXDX= RDXN(I)*(TXXP-TXXM)
DTXZDZ= RDZ(K)  *(TXZP-TXZM)
DMUDX = (MU(I+1,J,K)-MU(I,J,K))*RDXN(I)
VTRM   = RRHO*( DTXXDX + DTXZDZ - 2._EB*UU(I,J,K)*DMUDX/R(I) )
FVX(I,J,K) = 0.25_EB*WOMY + GX*AH - VTRM
ENDDO
ENDDO

! Compute z-direction flux term FVZ

DO K=0,KBAR
DO I=1,IBAR
UP     = UU(I,J,K)    + UU(I,J,K+1)
UM     = UU(I-1,J,K) + UU(I-1,J,K+1)
OMYP   = OMY(I,J,K)
OMYM   = OMY(I-1,J,K)
TXZP   = TXZ(I,J,K)
TXZM   = TXZ(I-1,J,K)
IC     = CELL_INDEX(I,J,K)
IEYP   = EDGE_INDEX(8,IC)
IEYM   = EDGE_INDEX(7,IC)
IF (OME_E(-2,IEYP)>-1.E5_EB) THEN
OMYP = OME_E(-2,IEYP)
TXZP = TAU_E(-2,IEYP)
ENDIF
IF (OME_E( 2,IEYM)>-1.E5_EB) THEN
OMYM = OME_E( 2,IEYM)
```

```
TXZM = TAU_E( 2,IEYM)
ENDIF
UOMY  = UP*OMYP + UM*OMYM
RRHO  = 2._EB/(RHOP(I,J,K)+RHOP(I,J,K+1))
AH    = 0.5_EB*(RHO_0(K)+RHO_0(K+1))*RRHO - 1._EB
DUDX  = (R(I)*UU(I,J,K+1)-R(I-1)*UU(I-1,J,K+1))*RDX(I)*RRN(I)
TZZP  = MU(I,J,K+1)*( FOTH*DP(I,J,K+1) - 2._EB*DUDX )
DUDX  = (R(I)*UU(I,J,K)-R(I-1)*UU(I-1,J,K))*RDX(I)*RRN(I)
TZZM  = MU(I,J,K)   *( FOTH*DP(I,J,K)   - 2._EB*DUDX )
DTXZDX= RDX(I) *(R(I)*TXZP-R(I-1)*TXZM)*RRN(I)
DTZZDZ= RDZN(K)*(      TZZP         -TZZM)
VTRM  = RRHO*(DTXZDX + DTZZDZ)
FVZ(I,J,K) = -0.25_EB*UOMY + GZ*AH - VTRM
ENDDO
ENDDO

! Adjust FVX and FVZ at solid, internal obstructions for no flux

END SUBROUTINE VELOCITY_FLUX_CYLINDRICAL


SUBROUTINE NO_FLUX(DT,NM)

! Set FVX,FVY,FVZ inside and on the surface of solid obstructions to maintain no flux

USE MATH_FUNCTIONS, ONLY: EVALUATE_RAMP
INTEGER, INTENT(IN) :: NM
REAL(EB), INTENT(IN) :: DT
REAL(EB), POINTER, DIMENSION(:,:,:) :: HP=>NULL(),OM_HP=>NULL()
REAL(EB) :: RFODT,H_OTHER,DUUDT,DVVDT,DWWDT,UN,TNOW,DHFCT
INTEGER  :: IC2,IC1,N,I,J,K,IW,II,JJ,KK,IOR,N_INT_CELLS,IIO,JJO,KKO,NOM
TYPE (OBSTRUCTION_TYPE), POINTER :: OB=>NULL()
TYPE (WALL_TYPE), POINTER :: WC=>NULL()
TYPE (EXTERNAL_WALL_TYPE), POINTER :: EWC=>NULL()

TNOW=SECOND()
CALL POINT_TO_MESH(NM)

RFODT = RELAXATION_FACTOR/DT

IF (PREDICTOR) HP => H
IF (CORRECTOR) HP => HS

! Exchange H at interpolated boundaries

NO_SCARC_IF: IF (PRES_METHOD /= 'SCARC') THEN

DO IW=1,N_EXTERNAL_WALL_CELLS
WC=>WALL(IW)
EWC=>EXTERNAL_WALL(IW)
NOM =EWC%NOM
IF (NOM==0) CYCLE
IF (PREDICTOR) THEN
OM_HP=>OMESH(NOM)%H
ELSE
OM_HP=>OMESH(NOM)%HS
ENDIF
II = WC%ONE_D%II
JJ = WC%ONE_D%JJ
KK = WC%ONE_D%KK
H_OTHER = 0._EB
DO KKO=EWC%KKO_MIN,EWC%KKO_MAX
DO JJO=EWC%JJO_MIN,EWC%JJO_MAX
DO IIO=EWC%IIO_MIN,EWC%IIO_MAX
H_OTHER = H_OTHER + OM_HP(IIO,JJO,KKO)
ENDDO
ENDDO
ENDDO
N_INT_CELLS = (EWC%IIO_MAX-EWC%IIO_MIN+1) * (EWC%JJO_MAX-EWC%JJO_MIN+1) * (EWC%KKO_MAX-EWC%KKO_MIN+1)
HP(II,JJ,KK) = H_OTHER/REAL(N_INT_CELLS,EB)
ENDDO

ENDIF NO_SCARC_IF

! Set FVX, FVY and FVZ to drive velocity components at solid boundaries within obstructions towards zero

OBST_LOOP: DO N=1,N_OBST

OB=>OBSTRUCTION(N)

DO K=OB%K1+1,OB%K2
DO J=OB%J1+1,OB%J2
DO I=OB%I1  ,OB%I2
IC1 = CELL_INDEX(I,J,K)
IC2 = CELL_INDEX(I+1,J,K)
IF (SOLID(IC1) .AND. SOLID(IC2)) THEN
IF (PREDICTOR) THEN
DUUDT = -RFODT*U(I,J,K)
```

```
ELSE
DUUDT = -RFODT*(U(I,J,K)+US(I,J,K))
ENDIF
FVX(I,J,K) = -RDXN(I)*(HP(I+1,J,K)-HP(I,J,K)) - DUUDT
ENDIF
ENDDO
ENDDO
ENDDO

DO K=OB%K1+1,OB%K2
DO J=OB%J1  ,OB%J2
DO I=OB%I1+1,OB%I2
IC1 = CELL_INDEX(I,J,K)
IC2 = CELL_INDEX(I,J+1,K)
IF (SOLID(IC1) .AND. SOLID(IC2)) THEN
IF (PREDICTOR) THEN
DVVDT = -RFODT*V(I,J,K)
ELSE
DVVDT = -RFODT*(V(I,J,K)+VS(I,J,K))
ENDIF
FVY(I,J,K) = -RDYN(J)*(HP(I,J+1,K)-HP(I,J,K)) - DVVDT
ENDIF
ENDDO
ENDDO
ENDDO

DO K=OB%K1  ,OB%K2
DO J=OB%J1+1,OB%J2
DO I=OB%I1+1,OB%I2
IC1 = CELL_INDEX(I,J,K)
IC2 = CELL_INDEX(I,J,K+1)
IF (SOLID(IC1) .AND. SOLID(IC2)) THEN
IF (PREDICTOR) THEN
DWWDT = -RFODT*W(I,J,K)
ELSE
DWWDT = -RFODT*(W(I,J,K)+WS(I,J,K))
ENDIF
FVZ(I,J,K) = -RDZN(K)*(HP(I,J,K+1)-HP(I,J,K)) - DWWDT
ENDIF
ENDDO
ENDDO
ENDDO

ENDDO OBST_LOOP

! Set FVX, FVY and FVZ to drive the normal velocity at solid boundaries towards the specified value (UW or UWS)
DHFCT=1._EB
IF (.NOT. PRES_ON_WHOLE_DOMAIN) DHFCT=0._EB

WALL_LOOP: DO IW=1,N_EXTERNAL_WALL_CELLS+N_INTERNAL_WALL_CELLS

WC => WALL(IW)

IF (WC%BOUNDARY_TYPE==INTERPOLATED_BOUNDARY .OR. WC%BOUNDARY_TYPE==OPEN_BOUNDARY) CYCLE WALL_LOOP

IF (IW<=N_EXTERNAL_WALL_CELLS) THEN
NOM = EXTERNAL_WALL(IW)%NOM
ELSE
NOM = 0
ENDIF

IF (IW>N_EXTERNAL_WALL_CELLS .AND. WC%BOUNDARY_TYPE==NULL_BOUNDARY .AND. NOM==0) CYCLE WALL_LOOP

II  = WC%ONE_D%II
JJ  = WC%ONE_D%JJ
KK  = WC%ONE_D%KK
IOR = WC%ONE_D%IOR

IF (NOM/=0 .OR. WC%BOUNDARY_TYPE==SOLID_BOUNDARY .OR. WC%BOUNDARY_TYPE==NULL_BOUNDARY) THEN
IF (PREDICTOR) THEN
UN = -SIGN(1._EB,REAL(IOR,EB))*WC%ONE_D%UWS
ELSE
UN = -SIGN(1._EB,REAL(IOR,EB))*WC%ONE_D%UW
ENDIF
SELECT CASE(IOR)
CASE( 1)
IF (PREDICTOR) THEN
DUUDT = RFODT*(UN-U(II,JJ,KK))
ELSE
DUUDT = 2._EB*RFODT*(UN-0.5_EB*(U(II,JJ,KK)+US(II,JJ,KK)) )
ENDIF
FVX(II,JJ,KK) = -RDXN(II)*(HP(II+1,JJ,KK)-HP(II,JJ,KK))*DHFCT - DUUDT
CASE(-1)
IF (PREDICTOR) THEN
DUUDT = RFODT*(UN-U(II-1,JJ,KK))
ELSE
DUUDT = 2._EB*RFODT*(UN-0.5_EB*(U(II-1,JJ,KK)+US(II-1,JJ,KK)) )
ENDIF
```

```
FVX(II-1,JJ,KK) = -RDXN(II-1)*(HP(II,JJ,KK)-HP(II-1,JJ,KK))*DHFCT - DUUDT
CASE( 2)
IF (PREDICTOR) THEN
DVVDT = RFODT*(UN-V(II,JJ,KK))
ELSE
DVVDT = 2._EB*RFODT*(UN-0.5_EB*(V(II,JJ,KK)+VS(II,JJ,KK)) )
ENDIF
FVY(II,JJ,KK) = -RDYN(JJ)*(HP(II,JJ+1,KK)-HP(II,JJ,KK))*DHFCT - DVVDT
CASE(-2)
IF (PREDICTOR) THEN
DVVDT = RFODT*(UN-V(II,JJ-1,KK))
ELSE
DVVDT = 2._EB*RFODT*(UN-0.5_EB*(V(II,JJ-1,KK)+VS(II,JJ-1,KK)) )
ENDIF
FVY(II,JJ-1,KK) = -RDYN(JJ-1)*(HP(II,JJ,KK)-HP(II,JJ-1,KK))*DHFCT - DVVDT
CASE( 3)
IF (PREDICTOR) THEN
DWWDT = RFODT*(UN-W(II,JJ,KK))
ELSE
DWWDT = 2._EB*RFODT*(UN-0.5_EB*(W(II,JJ,KK)+WS(II,JJ,KK)) )
ENDIF
FVZ(II,JJ,KK) = -RDZN(KK)*(HP(II,JJ,KK+1)-HP(II,JJ,KK))*DHFCT - DWWDT
CASE(-3)
IF (PREDICTOR) THEN
DWWDT = RFODT*(UN-W(II,JJ,KK-1))
ELSE
DWWDT = 2._EB*RFODT*(UN-0.5_EB*(W(II,JJ,KK-1)+WS(II,JJ,KK-1)) )
ENDIF
FVZ(II,JJ,KK-1) = -RDZN(KK-1)*(HP(II,JJ,KK)-HP(II,JJ,KK-1))*DHFCT - DWWDT
END SELECT
ENDIF

IF (WC%BOUNDARY_TYPE==MIRROR_BOUNDARY) THEN
SELECT CASE(IOR)
CASE( 1)
FVX(II  ,JJ,KK) = 0._EB
CASE(-1)
FVX(II-1,JJ,KK) = 0._EB
CASE( 2)
FVY(II  ,JJ,KK) = 0._EB
CASE(-2)
FVY(II,JJ-1,KK) = 0._EB
CASE( 3)
FVZ(II  ,JJ,KK) = 0._EB
CASE(-3)
FVZ(II,JJ,KK-1) = 0._EB
END SELECT
ENDIF

ENDDO WALL_LOOP

T_USED(4)=T_USED(4)+SECOND()-TNOW
END SUBROUTINE NO_FLUX


SUBROUTINE VELOCITY_PREDICTOR(T,DT,DT_NEW,NM)

USE TURBULENCE, ONLY: COMPRESSION_WAVE
USE MANUFACTURED_SOLUTIONS, ONLY: UF_MMS,WF_MMS,VD2D_MMS_U,VD2D_MMS_V
USE COMPLEX_GEOMETRY, ONLY : CCIBM_VELOCITY_NO_GRADH

! Estimates the velocity components at the next time step

REAL(EB) :: TNOW,XHAT,ZHAT
INTEGER  :: I,J,K
INTEGER, INTENT(IN) :: NM
REAL(EB), INTENT(IN) :: T,DT
REAL(EB) :: DT_NEW(NMESHES)

IF (SOLID_PHASE_ONLY) RETURN
IF (PERIODIC_TEST==4) THEN
CALL COMPRESSION_WAVE(NM,T,4)
CALL CHECK_STABILITY(DT,DT_NEW,NM)
RETURN
ENDIF

TNOW=SECOND()
CALL POINT_TO_MESH(NM)

FREEZE_VELOCITY_IF: IF (FREEZE_VELOCITY) THEN
US = U
VS = V
WS = W
ELSE FREEZE_VELOCITY_IF

DO K=1,KBAR
DO J=1,JBAR
DO I=0,IBAR
```

```
US(I,J,K) = U(I,J,K) - DT*( FVX(I,J,K) + RDXN(I)*(H(I+1,J,K)-H(I,J,K)) )
ENDDO
ENDDO
ENDDO

DO K=1,KBAR
DO J=0,JBAR
DO I=1,IBAR
VS(I,J,K) = V(I,J,K) - DT*( FVY(I,J,K) + RDYN(J)*(H(I,J+1,K)-H(I,J,K)) )
ENDDO
ENDDO
ENDDO

DO K=0,KBAR
DO J=1,JBAR
DO I=1,IBAR
WS(I,J,K) = W(I,J,K) - DT*( FVZ(I,J,K) + RDZN(K)*(H(I,J,K+1)-H(I,J,K)) )
ENDDO
ENDDO
ENDDO

 IF (PRES_METHOD == 'GLMAT') CALL WALL_VELOCITY_NO_GRADH(DT,.FALSE.)
 IF (CC_IBM)                 CALL CCIBM_VELOCITY_NO_GRADH(DT)

 ENDIF FREEZE_VELOCITY_IF

 ! Manufactured solution (debug)

 IF (PERIODIC_TEST==7 .AND. .FALSE.) THEN
DO K=1,KBAR
DO J=1,JBAR
DO I=0,IBAR
XHAT =  X(I) - UF_MMS*(T)
ZHAT = ZC(K) - WF_MMS*(T)
US(I,J,K) = VD2D_MMS_U(XHAT,ZHAT,T)
ENDDO
ENDDO
ENDDO
DO K=0,KBAR
DO J=1,JBAR
DO I=1,IBAR
XHAT = XC(I) - UF_MMS*(T)
ZHAT =  Z(K) - WF_MMS*(T)
WS(I,J,K) = VD2D_MMS_V(XHAT,ZHAT,T)
ENDDO
ENDDO
ENDDO
 ENDIF

 ! No vertical velocity in Evacuation meshes

 IF (EVACUATION_ONLY(NM)) WS = 0._EB

 ! Check the stability criteria, and if the time step is too small, send back a signal to kill the job

CALL CHECK_STABILITY(DT,DT_NEW,NM)

 IF (DT_NEW(NM)<DT_INITIAL*LIMITING_DT_RATIO .AND. (T+DT_NEW(NM)<(T_END-TWO_EPSILON_EB))) STOP_STATUS = INSTABILITY_STOP

T_USED(4)=T_USED(4)+SECOND()-TNOW
END SUBROUTINE VELOCITY_PREDICTOR


SUBROUTINE VELOCITY_CORRECTOR(T,DT,NM)

USE TURBULENCE, ONLY: COMPRESSION_WAVE
USE MANUFACTURED_SOLUTIONS, ONLY: UF_MMS,WF_MMS,VD2D_MMS_U,VD2D_MMS_V
USE COMPLEX_GEOMETRY, ONLY : CCIBM_VELOCITY_NO_GRADH

 ! Correct the velocity components

REAL(EB) :: TNOW,XHAT,ZHAT
INTEGER  :: I,J,K
INTEGER, INTENT(IN) :: NM
REAL(EB), INTENT(IN) :: T,DT

 IF (SOLID_PHASE_ONLY) RETURN
 IF (PERIODIC_TEST==4) THEN
CALL COMPRESSION_WAVE(NM,T,4)
RETURN
 ENDIF

TNOW=SECOND()
CALL POINT_TO_MESH(NM)

 FREEZE_VELOCITY_IF: IF (FREEZE_VELOCITY) THEN
U = US
```

```
V = VS
W = WS
ELSE FREEZE_VELOCITY_IF

IF (STORE_OLD_VELOCITY) THEN
U_OLD = U
V_OLD = V
W_OLD = W
ENDIF

IF (PRES_METHOD == 'GLMAT') CALL WALL_VELOCITY_NO_GRADH(DT,.TRUE.) ! Store U velocities on OBST surfaces.

DO K=1,KBAR
DO J=1,JBAR
DO I=0,IBAR
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))) )
ENDDO
ENDDO
ENDDO

DO K=1,KBAR
DO J=0,JBAR
DO I=1,IBAR
V(I,J,K) = 0.5_EB*( V(I,J,K) + VS(I,J,K) - DT*(FVY(I,J,K) + RDYN(J)*(HS(I,J+1,K)-HS(I,J,K))) )
ENDDO
ENDDO
ENDDO

DO K=0,KBAR
DO J=1,JBAR
DO I=1,IBAR
W(I,J,K) = 0.5_EB*( W(I,J,K) + WS(I,J,K) - DT*(FVZ(I,J,K) + RDZN(K)*(HS(I,J,K+1)-HS(I,J,K))) )
ENDDO
ENDDO
ENDDO

IF (PRES_METHOD == 'GLMAT') CALL WALL_VELOCITY_NO_GRADH(DT,.FALSE.)
IF (CC_IBM)                  CALL CCIBM_VELOCITY_NO_GRADH(DT)

ENDIF FREEZE_VELOCITY_IF

! Manufactured solution (debug)

IF (PERIODIC_TEST==7 .AND. .FALSE.) THEN
DO K=1,KBAR
DO J=1,JBAR
DO I=0,IBAR
XHAT =  X(I) - UF_MMS*T
ZHAT = ZC(K) - WF_MMS*T
U(I,J,K) = VD2D_MMS_U(XHAT,ZHAT,T)
ENDDO
ENDDO
ENDDO
DO K=0,KBAR
DO J=1,JBAR
DO I=1,IBAR
XHAT = XC(I) - UF_MMS*T
ZHAT =  Z(K) - WF_MMS*T
W(I,J,K) = VD2D_MMS_V(XHAT,ZHAT,T)
ENDDO
ENDDO
ENDDO
ENDIF

! No vertical velocity in Evacuation meshes

IF (EVACUATION_ONLY(NM)) W = 0._EB

T_USED(4)=T_USED(4)+SECOND()-TNOW
END SUBROUTINE VELOCITY_CORRECTOR


SUBROUTINE VELOCITY_BC(T,NM)

! Assert tangential velocity boundary conditions

USE MATH_FUNCTIONS, ONLY: EVALUATE_RAMP
USE TURBULENCE, ONLY: WALL_MODEL,WANNIER_FLOW
REAL(EB), INTENT(IN) :: T
REAL(EB) :: MUA, TSI,WGT,TNOW,RAMP_T,OMW,MU_WALL,RHO_WALL,SLIP_COEF,VEL_T,UBAR,VBAR,WBAR, &
UUP(2),UUM(2),DXX(2),MU_DUIDXJ(-2:2),DUIDXJ(-2:2),PROFILE_FACTOR,VEL_GAS,VEL_GHOST, &
MU_DUIDXJ_USE(2),DUIDXJ_USE(2),VEL_EDDY,U_TAU,Y_PLUS,WT1,WT2,DUMMY
INTEGER :: I,J,K,NOM(2),IIO(2),JJO(2),KKO(2),IE,II,JJ,KK,IEC,IOR,IWM,IWP,ICMM,ICMP,ICPM,ICPP,IC,ICD,ICDO,IVL,
      I_SGN,IS, &
VELOCITY_BC_INDEX,IIGM,JJGM,KKGM,IIGP,JJGP,KKGP,SURF_INDEXM,SURF_INDEXP,ITMP,ICD_SGN,ICDO_SGN, &
BOUNDARY_TYPE_M,BOUNDARY_TYPE_P,IS2,IWPI,IWMI,VENT_INDEX
LOGICAL :: ALTERED_GRADIENT(-2:2),PROCESS_EDGE,SYNTHETIC_EDDY_METHOD,HVAC_TANGENTIAL,INTERPOLATED_EDGE
INTEGER, INTENT(IN) :: NM
```

```
REAL(EB), POINTER, DIMENSION(:,:,:) :: UU=>NULL(),VV=>NULL(),WW=>NULL(),U_Y=>NULL(),U_Z=>NULL(), &
V_X=>NULL(),V_Z=>NULL(),W_X=>NULL(),W_Y=>NULL(),RHOP=>NULL(),VEL_OTHER=>NULL()
TYPE (SURFACE_TYPE), POINTER :: SF=>NULL()
TYPE (OMESH_TYPE), POINTER :: OM=>NULL()
TYPE (VENTS_TYPE), POINTER :: VT
TYPE (WALL_TYPE), POINTER :: WCM,WCP

IF (SOLID_PHASE_ONLY) RETURN

TNOW = SECOND()

! Assign local names to variables

CALL POINT_TO_MESH(NM)

! Point to the appropriate velocity field

IF (PREDICTOR) THEN
UU => US
VV => VS
WW => WS
RHOP => RHOS
ELSE
UU => U
VV => V
WW => W
RHOP => RHO
ENDIF

! Set the boundary velocity place holder to some large negative number

IF (CORRECTOR) THEN
U_Y => WORK1
U_Z => WORK2
V_X => WORK3
V_Z => WORK4
W_X => WORK5
W_Y => WORK6
U_Y = -1.E6_EB
U_Z = -1.E6_EB
V_X = -1.E6_EB
V_Z = -1.E6_EB
W_X = -1.E6_EB
W_Y = -1.E6_EB
UVW_GHOST = -1.E6_EB
ENDIF

! Set OME_E and TAU_E to very negative number

TAU_E = -1.E6_EB
OME_E = -1.E6_EB

! Loop over all cell edges and determine the appropriate velocity BCs

EDGE_LOOP: DO IE=1,N_EDGES

INTERPOLATED_EDGE = .FALSE.

! Throw out edges that are completely surrounded by blockages or the exterior of the domain

PROCESS_EDGE = .FALSE.
DO IS=5,8
IF (.NOT.EXTERIOR(IJKE(IS,IE)) .AND. .NOT.SOLID(IJKE(IS,IE))) THEN
PROCESS_EDGE = .TRUE.
EXIT
ENDIF
ENDDO
IF (.NOT.PROCESS_EDGE) CYCLE EDGE_LOOP

! If the edge is to be "smoothed," set tau and omega to zero and cycle

IF (EVACUATION_ONLY(NM)) THEN
OME_E(:,IE) = 0._EB
TAU_E(:,IE) = 0._EB
CYCLE EDGE_LOOP
ENDIF

! Unpack indices for the edge

II     = IJKE( 1,IE)
JJ     = IJKE( 2,IE)
KK     = IJKE( 3,IE)
IEC    = IJKE( 4,IE)
ICMM   = IJKE( 5,IE)
ICPM   = IJKE( 6,IE)
ICMP   = IJKE( 7,IE)
ICPP   = IJKE( 8,IE)
NOM(1) = IJKE( 9,IE)
```

```
IIO(1) = IJKE(10,IE)
JJO(1) = IJKE(11,IE)
KKO(1) = IJKE(12,IE)
NOM(2) = IJKE(13,IE)
IIO(2) = IJKE(14,IE)
JJO(2) = IJKE(15,IE)
KKO(2) = IJKE(16,IE)

! Get the velocity components at the appropriate cell faces

COMPONENT: SELECT CASE(IEC)
CASE(1) COMPONENT
UUP(1)  = VV(II,JJ,KK+1)
UUM(1)  = VV(II,JJ,KK)
UUP(2)  = WW(II,JJ+1,KK)
UUM(2)  = WW(II,JJ,KK)
DXX(1)  = DY(JJ)
DXX(2)  = DZ(KK)
MUA     = 0.25_EB*(MU(II,JJ,KK) + MU(II,JJ+1,KK) + MU(II,JJ+1,KK+1) + MU(II,JJ,KK+1) )
CASE(2) COMPONENT
UUP(1)  = WW(II+1,JJ,KK)
UUM(1)  = WW(II,JJ,KK)
UUP(2)  = UU(II,JJ,KK+1)
UUM(2)  = UU(II,JJ,KK)
DXX(1)  = DZ(KK)
DXX(2)  = DX(II)
MUA     = 0.25_EB*(MU(II,JJ,KK) + MU(II+1,JJ,KK) + MU(II+1,JJ,KK+1) + MU(II,JJ,KK+1) )
CASE(3) COMPONENT
UUP(1)  = UU(II,JJ+1,KK)
UUM(1)  = UU(II,JJ,KK)
UUP(2)  = VV(II+1,JJ,KK)
UUM(2)  = VV(II,JJ,KK)
DXX(1)  = DX(II)
DXX(2)  = DY(JJ)
MUA     = 0.25_EB*(MU(II,JJ,KK) + MU(II+1,JJ,KK) + MU(II+1,JJ+1,KK) + MU(II,JJ+1,KK) )
END SELECT COMPONENT

! Indicate that the velocity gradients in the two orthogonal directions have not been changed yet

ALTERED_GRADIENT = .FALSE.

! Loop over all possible orientations of edge and reassign velocity gradients if appropriate

SIGN_LOOP: DO I_SGN=-1,1,2
ORIENTATION_LOOP: DO IS=1,3

IF (IS==IEC) CYCLE ORIENTATION_LOOP

! IOR is the orientation of the wall cells adjacent to the edge

IOR = I_SGN*IS

! IS2 is the other coordinate direction besides IOR.

SELECT CASE(IEC)
CASE(1)
IF (IS==2) IS2 = 3
IF (IS==3) IS2 = 2
CASE(2)
IF (IS==1) IS2 = 3
IF (IS==3) IS2 = 1
CASE(3)
IF (IS==1) IS2 = 2
IF (IS==2) IS2 = 1
END SELECT

! Determine Index_Coordinate_Direction
! IEC=1, ICD=1 refers to DWDY; ICD=2 refers to DVDZ
! IEC=2, ICD=1 refers to DUDZ; ICD=2 refers to DWDX
! IEC=3, ICD=1 refers to DVDX; ICD=2 refers to DUDY

IF (IS>IEC) ICD = IS-IEC
IF (IS<IEC) ICD = IS-IEC+3
ICD_SGN = I_SGN * ICD

! IWM and IWP are the wall cell indices of the boundary on either side of the edge.

IF (IOR<0) THEN
IWM  = WALL_INDEX(ICMM,-IOR)
IWMI = WALL_INDEX(ICMM, IS2)
IF (ICD==1) THEN
IWP  = WALL_INDEX(ICMP,-IOR)
IWPI = WALL_INDEX(ICMP,-IS2)
ELSE ! ICD==2
IWP  = WALL_INDEX(ICPM,-IOR)
IWPI = WALL_INDEX(ICPM,-IS2)
ENDIF
ELSE
```

```
IF (ICD==1) THEN
IWM  = WALL_INDEX(ICPM,-IOR)
IWMI = WALL_INDEX(ICPM,IS2)
ELSE ! ICD==2
IWM  = WALL_INDEX(ICMP,-IOR)
IWMI = WALL_INDEX(ICMP,IS2)
ENDIF
IWP  = WALL_INDEX(ICPP,-IOR)
IWPI = WALL_INDEX(ICPP,-IS2)
ENDIF

! If both adjacent wall cells are undefined, cycle out of the loop.

IF (IWM==0 .AND. IWP==0) CYCLE ORIENTATION_LOOP

! If there is a solid wall separating the two adjacent wall cells, cycle out of the loop.

IF (WALL(IWMI)%BOUNDARY_TYPE==SOLID_BOUNDARY .OR. WALL(IWPI)%BOUNDARY_TYPE==SOLID_BOUNDARY) CYCLE ORIENTATION_LOOP

! If only one adjacent wall cell is defined, use its properties.

IF (IWM>0) THEN
WCM => WALL(IWM)
ELSE
WCM => WALL(IWP)
ENDIF

IF (IWP>0) THEN
WCP => WALL(IWP)
ELSE
WCP => WALL(IWM)
ENDIF

! If both adjacent wall cells are NULL, cycle out.

BOUNDARY_TYPE_M = WCM%BOUNDARY_TYPE
BOUNDARY_TYPE_P = WCP%BOUNDARY_TYPE

IF (BOUNDARY_TYPE_M==NULL_BOUNDARY .AND. BOUNDARY_TYPE_P==NULL_BOUNDARY) CYCLE ORIENTATION_LOOP

! OPEN boundary conditions, both varieties, with and without a wind

OPEN_AND_WIND_BC: IF ((IWM==0.OR.WALL(IWM)%BOUNDARY_TYPE==OPEN_BOUNDARY) .AND. &
(IWP==0.OR.WALL(IWP)%BOUNDARY_TYPE==OPEN_BOUNDARY)) THEN

VENT_INDEX = MAX(WCM%VENT_INDEX,WCP%VENT_INDEX)
VT => VENTS(VENT_INDEX)

WIND_NO_WIND_IF: IF (.NOT.ANY(MEAN_FORCING)) THEN ! For regular OPEN boundary, (free-slip) BCs

SELECT CASE(IEC)
CASE(1)
IF (JJ==0    .AND. IOR== 2) WW(II,0,KK)    = WW(II,1,KK)
IF (JJ==JBAR .AND. IOR==-2) WW(II,JBP1,KK) = WW(II,JBAR,KK)
IF (KK==0    .AND. IOR== 3) VV(II,JJ,0)    = VV(II,JJ,1)
IF (KK==KBAR .AND. IOR==-3) VV(II,JJ,KBP1) = VV(II,JJ,KBAR)
CASE(2)
IF (II==0    .AND. IOR== 1) WW(0,JJ,KK)    = WW(1,JJ,KK)
IF (II==IBAR .AND. IOR==-1) WW(IBP1,JJ,KK) = WW(IBAR,JJ,KK)
IF (KK==0    .AND. IOR== 3) UU(II,JJ,0)    = UU(II,JJ,1)
IF (KK==KBAR .AND. IOR==-3) UU(II,JJ,KBP1) = UU(II,JJ,KBAR)
CASE(3)
IF (II==0    .AND. IOR== 1) VV(0,JJ,KK)    = VV(1,JJ,KK)
IF (II==IBAR .AND. IOR==-1) VV(IBP1,JJ,KK) = VV(IBAR,JJ,KK)
IF (JJ==0    .AND. IOR== 2) UU(II,0,KK)    = UU(II,1,KK)
IF (JJ==JBAR .AND. IOR==-2) UU(II,JBP1,KK) = UU(II,JBAR,KK)
END SELECT

ELSE WIND_NO_WIND_IF ! For wind, use prescribed far-field velocity all around

UBAR = U0*EVALUATE_RAMP(T,DUMMY,I_RAMP_U0_T)*EVALUATE_RAMP(ZC(KK),DUMMY,I_RAMP_U0_Z)
VBAR = V0*EVALUATE_RAMP(T,DUMMY,I_RAMP_V0_T)*EVALUATE_RAMP(ZC(KK),DUMMY,I_RAMP_V0_Z)
WBAR = W0*EVALUATE_RAMP(T,DUMMY,I_RAMP_W0_T)*EVALUATE_RAMP(ZC(KK),DUMMY,I_RAMP_W0_Z)

SELECT CASE(IEC)
CASE(1)
IF (JJ==0    .AND. IOR== 2) WW(II,0,KK)    = WBAR
IF (JJ==JBAR .AND. IOR==-2) WW(II,JBP1,KK) = WBAR
IF (KK==0    .AND. IOR== 3) VV(II,JJ,0)    = VBAR
IF (KK==KBAR .AND. IOR==-3) VV(II,JJ,KBP1) = VBAR
CASE(2)
IF (II==0    .AND. IOR== 1) WW(0,JJ,KK)    = WBAR
IF (II==IBAR .AND. IOR==-1) WW(IBP1,JJ,KK) = WBAR
IF (KK==0    .AND. IOR== 3) UU(II,JJ,0)    = UBAR
IF (KK==KBAR .AND. IOR==-3) UU(II,JJ,KBP1) = UBAR
CASE(3)
IF (II==0    .AND. IOR== 1) VV(0,JJ,KK)    = VBAR
```

```
IF (II==IBAR .AND. IOR==-1) VV(IBP1,JJ,KK) = VBAR
IF (JJ==0 .AND. IOR== 2) UU(II,0,KK) = UBAR
IF (JJ==JBAR .AND. IOR==-2) UU(II,JBP1,KK) = UBAR
END SELECT

ENDIF WIND_NO_WIND_IF

IF (IWM/=0 .AND. IWP/=0) THEN
CYCLE EDGE_LOOP ! Do no further processing of this edge if both cell faces are OPEN
ELSE
CYCLE ORIENTATION_LOOP
ENDIF

ENDIF OPEN_AND_WIND_BC

! Define the appropriate gas and ghost velocity

IF (ICD==1) THEN ! Used to pick the appropriate velocity component
IVL=2
ELSE !ICD==2
IVL=1
ENDIF

IF (IOR<0) THEN
VEL_GAS = UUM(IVL)
VEL_GHOST = UUP(IVL)
IIGM = I_CELL(ICMM)
JJGM = J_CELL(ICMM)
KKGM = K_CELL(ICMM)
IF (ICD==1) THEN
IIGP = I_CELL(ICMP)
JJGP = J_CELL(ICMP)
KKGP = K_CELL(ICMP)
ELSE ! ICD==2
IIGP = I_CELL(ICPM)
JJGP = J_CELL(ICPM)
KKGP = K_CELL(ICPM)
ENDIF
ELSE
VEL_GAS = UUP(IVL)
VEL_GHOST = UUM(IVL)
IF (ICD==1) THEN
IIGM = I_CELL(ICPM)
JJGM = J_CELL(ICPM)
KKGM = K_CELL(ICPM)
ELSE ! ICD==2
IIGM = I_CELL(ICMP)
JJGM = J_CELL(ICMP)
KKGM = K_CELL(ICMP)
ENDIF
IIGP = I_CELL(ICPP)
JJGP = J_CELL(ICPP)
KKGP = K_CELL(ICPP)
ENDIF

! Decide whether or not to process edge using data interpolated from another mesh

INTERPOLATION_IF: IF (NOM(ICD)==0 .OR. &
(BOUNDARY_TYPE_M==SOLID_BOUNDARY .OR. BOUNDARY_TYPE_P==SOLID_BOUNDARY) .OR. &
(BOUNDARY_TYPE_M/=INTERPOLATED_BOUNDARY .AND. BOUNDARY_TYPE_P/=INTERPOLATED_BOUNDARY)) THEN

! Determine appropriate velocity BC by assessing each adjacent wall cell. If the BCs are different on each
! side of the edge, choose the one with the specified velocity or velocity gradient, if there is one.
! If not, choose the max value of boundary condition index, simply for consistency.

SURF_INDEXM = WCM%SURF_INDEX
SURF_INDEXP = WCP%SURF_INDEX
IF (SURFACE(SURF_INDEXM)%SPECIFIED_NORMAL_VELOCITY .OR. SURFACE(SURF_INDEXM)%SPECIFIED_NORMAL_GRADIENT) THEN
SF=>SURFACE(SURF_INDEXM)
ELSEIF (SURFACE(SURF_INDEXP)%SPECIFIED_NORMAL_VELOCITY .OR. SURFACE(SURF_INDEXP)%SPECIFIED_NORMAL_GRADIENT) THEN
SF=>SURFACE(SURF_INDEXP)
ELSE
SF=>SURFACE(MAX(SURF_INDEXM,SURF_INDEXP))
ENDIF
VELOCITY_BC_INDEX = SF%VELOCITY_BC_INDEX
IF (WCM%VENT_INDEX==WCP%VENT_INDEX .AND. WCP%VENT_INDEX > 0) THEN
IF(VENTS(WCM%VENT_INDEX)%NODE_INDEX>0 .AND. WCM%ONE_D%UW >= 0._EB) VELOCITY_BC_INDEX=FREE_SLIP_BC
ENDIF

! Compute the viscosity in the two adjacent gas cells

MUA = 0.5_EB*(MU(IIGM,JJGM,KKGM) + MU(IIGP,JJGP,KKGP))

! Set up synthetic eddy method (experimental)

SYNTHETIC_EDDY_METHOD = .FALSE.
HVAC_TANGENTIAL = .FALSE.
IF (IWM>0 .AND. IWP>0) THEN
```

```
IF (WCM%VENT_INDEX==WCP%VENT_INDEX) THEN
IF (WCM%VENT_INDEX>0) THEN
VT=>VENTS(WCM%VENT_INDEX)
IF (VT%N_EDDY>0) SYNTHETIC_EDDY_METHOD=.TRUE.
IF (ALL(VT%UVW > -1.E12_EB) .AND. VT%NODE_INDEX > 0) HVAC_TANGENTIAL = .TRUE.
ENDIF
ENDIF
ENDIF

! Determine if there is a tangential velocity component

VEL_T_IF: IF (.NOT.SF%SPECIFIED_TANGENTIAL_VELOCITY .AND. .NOT.SYNTHETIC_EDDY_METHOD .AND. .NOT. HVAC_TANGENTIAL)
      THEN
VEL_T = 0._EB
ELSE VEL_T_IF
VEL_EDDY = 0._EB
SYNTHETIC_EDDY_IF: IF (SYNTHETIC_EDDY_METHOD) THEN
IS_SELECT: SELECT CASE(IS) ! unsigned vent orientation
CASE(1) ! yz plane
SELECT CASE(IEC) ! edge orientation
CASE(2)
IF (ICD==1) VEL_EDDY = 0.5_EB*(VT%U_EDDY(JJ,KK)+VT%U_EDDY(JJ,KK+1))
IF (ICD==2) VEL_EDDY = 0.5_EB*(VT%W_EDDY(JJ,KK)+VT%W_EDDY(JJ,KK+1))
CASE(3)
IF (ICD==1) VEL_EDDY = 0.5_EB*(VT%V_EDDY(JJ,KK)+VT%V_EDDY(JJ+1,KK))
IF (ICD==2) VEL_EDDY = 0.5_EB*(VT%U_EDDY(JJ,KK)+VT%U_EDDY(JJ+1,KK))
END SELECT
CASE(2) ! zx plane
SELECT CASE(IEC)
CASE(3)
IF (ICD==1) VEL_EDDY = 0.5_EB*(VT%V_EDDY(II,KK)+VT%V_EDDY(II+1,KK))
IF (ICD==2) VEL_EDDY = 0.5_EB*(VT%U_EDDY(II,KK)+VT%U_EDDY(II+1,KK))
CASE(1)
IF (ICD==1) VEL_EDDY = 0.5_EB*(VT%W_EDDY(II,KK)+VT%W_EDDY(II,KK+1))
IF (ICD==2) VEL_EDDY = 0.5_EB*(VT%V_EDDY(II,KK)+VT%V_EDDY(II,KK+1))
END SELECT
CASE(3) ! xy plane
SELECT CASE(IEC)
CASE(1)
IF (ICD==1) VEL_EDDY = 0.5_EB*(VT%W_EDDY(II,JJ)+VT%W_EDDY(II,JJ+1))
IF (ICD==2) VEL_EDDY = 0.5_EB*(VT%V_EDDY(II,JJ)+VT%V_EDDY(II,JJ+1))
CASE(2)
IF (ICD==1) VEL_EDDY = 0.5_EB*(VT%U_EDDY(II,JJ)+VT%U_EDDY(II+1,JJ))
IF (ICD==2) VEL_EDDY = 0.5_EB*(VT%W_EDDY(II,JJ)+VT%W_EDDY(II+1,JJ))
END SELECT
END SELECT IS_SELECT
ENDIF SYNTHETIC_EDDY_IF
IF (ABS(SF%T_IGN-T_BEGIN)<=SPACING(SF%T_IGN) .AND. SF%RAMP_INDEX(TIME_VELO)>=1) THEN
TSI = T
ELSE
TSI=T-SF%T_IGN
ENDIF
PROFILE_FACTOR = 1._EB
IF (HVAC_TANGENTIAL .AND. 0.5_EB*(WCM%ONE_D%UWS+WCP%ONE_D%UWS) > 0._EB) HVAC_TANGENTIAL = .FALSE.
IF (HVAC_TANGENTIAL) THEN
VEL_T = 0._EB
IEC_SELECT: SELECT CASE(IEC) ! edge orientation
CASE (1)
IF (ICD==1) VEL_T = 0.5_EB*ABS((WCM%ONE_D%UWS+WCP%ONE_D%UWS)/VT%UVW(ABS(VT%IOR)))*VT%UVW(3)
IF (ICD==2) VEL_T = 0.5_EB*ABS((WCM%ONE_D%UWS+WCP%ONE_D%UWS)/VT%UVW(ABS(VT%IOR)))*VT%UVW(2)
CASE (2)
IF (ICD==1) VEL_T = 0.5_EB*ABS((WCM%ONE_D%UWS+WCP%ONE_D%UWS)/VT%UVW(ABS(VT%IOR)))*VT%UVW(1)
IF (ICD==2) VEL_T = 0.5_EB*ABS((WCM%ONE_D%UWS+WCP%ONE_D%UWS)/VT%UVW(ABS(VT%IOR)))*VT%UVW(3)
CASE (3)
IF (ICD==1) VEL_T = 0.5_EB*ABS((WCM%ONE_D%UWS+WCP%ONE_D%UWS)/VT%UVW(ABS(VT%IOR)))*VT%UVW(2)
IF (ICD==2) VEL_T = 0.5_EB*ABS((WCM%ONE_D%UWS+WCP%ONE_D%UWS)/VT%UVW(ABS(VT%IOR)))*VT%UVW(1)
END SELECT IEC_SELECT
ELSE
IF (SF%PROFILE/=0 .AND. SF%VEL>TWO_EPSILON_EB) &
PROFILE_FACTOR = ABS(0.5_EB*(WCM%UW0+WCP%UW0)/SF%VEL)
RAMP_T = EVALUATE_RAMP(TSI,SF%TAU(TIME_VELO),SF%RAMP_INDEX(TIME_VELO))
IF (IEC==1 .OR. (IEC==2 .AND. ICD==2)) VEL_T = RAMP_T*(PROFILE_FACTOR*(SF%VEL_T(2) + VEL_EDDY))
IF (IEC==3 .OR. (IEC==2 .AND. ICD==1)) VEL_T = RAMP_T*(PROFILE_FACTOR*(SF%VEL_T(1) + VEL_EDDY))
ENDIF
ENDIF VEL_T_IF

! Choose the appropriate boundary condition to apply

HVAC_IF: IF (HVAC_TANGENTIAL)  THEN

VEL_GHOST = 2._EB*VEL_T - VEL_GAS
DUIDXJ(ICD_SGN) = I_SGN*(VEL_GAS-VEL_GHOST)/DXX(ICD)
MU_DUIDXJ(ICD_SGN) = MUA*DUIDXJ(ICD_SGN)
ALTERED_GRADIENT(ICD_SGN) = .TRUE.

ELSE HVAC_IF

BOUNDARY_CONDITION: SELECT CASE(VELOCITY_BC_INDEX)
```

```

CASE (FREE_SLIP_BC) BOUNDARY_CONDITION

VEL_GHOST = VEL_GAS
DUIDXJ(ICD_SGN) = I_SGN*(VEL_GAS-VEL_GHOST)/DXX(ICD)
MU_DUIDXJ(ICD_SGN) = MUA*DUIDXJ(ICD_SGN)
ALTERED_GRADIENT(ICD_SGN) = .TRUE.

CASE (NO_SLIP_BC) BOUNDARY_CONDITION

WANNIER_BC: IF (PERIODIC_TEST==5) THEN
SELECT CASE(IOR)
CASE( 1)
VEL_T = WANNIER_FLOW(X(II),Z(KK),2)
CASE(-1)
VEL_T = WANNIER_FLOW(X(II),Z(KK),2)
CASE( 3)
VEL_T = WANNIER_FLOW(X(II),Z(KK),1)
CASE(-3)
VEL_T = WANNIER_FLOW(X(II),Z(KK),1)
END SELECT
ENDIF WANNIER_BC

VEL_GHOST = 2._EB*VEL_T - VEL_GAS
DUIDXJ(ICD_SGN) = I_SGN*(VEL_GAS-VEL_GHOST)/DXX(ICD)
MU_DUIDXJ(ICD_SGN) = MUA*DUIDXJ(ICD_SGN)
ALTERED_GRADIENT(ICD_SGN) = .TRUE.

CASE (WALL_MODEL_BC) BOUNDARY_CONDITION

ITMP = MIN(5000,NINT(0.5_EB*(TMP(IIGM,JJGM,KKGM)+TMP(IIGP,JJGP,KKGP))))
MU_WALL = MU_RSQMW_Z(ITMP,1)/RSQ_MW_Z(1)
RHO_WALL = 0.5_EB*( RHOP(IIGM,JJGM,KKGM) + RHOP(IIGP,JJGP,KKGP) )
CALL WALL_MODEL(SLIP_COEF,U_TAU,Y_PLUS,VEL_GAS-VEL_T,MU_WALL/RHO_WALL,DXX(ICD),SF%ROUGHNESS)
SELECT CASE(SLIP_CONDITION)
CASE(0)
SLIP_COEF = -1._EB
CASE(1)
SLIP_COEF = SLIP_COEF
CASE(2)
SLIP_COEF = 0.5_EB*(SLIP_COEF-1._EB)
CASE(3)
WT1 = MAX(0._EB,MIN(1._EB,(Y_PLUS-Y_WERNER_WENGLE)/(Y_PLUS+TWO_EPSILON_EB)))
WT2 = 1._EB-WT1
SLIP_COEF = WT1*SLIP_COEF-WT2
CASE(4)
IF ( ABS(0.5_EB*(WCM%ONE_D%UWS+WCP%ONE_D%UWS))>ABS(VEL_GAS-VEL_T) ) THEN
SLIP_COEF = -1._EB
ELSE
SLIP_COEF = 0.5_EB*(SLIP_COEF-1._EB)
ENDIF
END SELECT
VEL_GHOST = VEL_T + SLIP_COEF*(VEL_GAS-VEL_T)
DUIDXJ(ICD_SGN) = I_SGN*(VEL_GAS-VEL_GHOST)/DXX(ICD)
MU_DUIDXJ(ICD_SGN) = RHO_WALL*U_TAU**2 * SIGN(1._EB,I_SGN*(VEL_GAS-VEL_T))
ALTERED_GRADIENT(ICD_SGN) = .TRUE.

END SELECT BOUNDARY_CONDITION

ENDIF HVAC_IF

ELSE INTERPOLATION_IF  ! Use data from another mesh

INTERPOLATED_EDGE = .TRUE.
OM => OMESH(ABS(NOM(ICD)))

IF (PREDICTOR) THEN
SELECT CASE(IEC)
CASE(1)
IF (ICD==1) THEN
VEL_OTHER => OM%WS
ELSE ! ICD=2
VEL_OTHER => OM%VS
ENDIF
CASE(2)
IF (ICD==1) THEN
VEL_OTHER => OM%US
ELSE ! ICD=2
VEL_OTHER => OM%WS
ENDIF
CASE(3)
IF (ICD==1) THEN
VEL_OTHER => OM%VS
ELSE ! ICD=2
VEL_OTHER => OM%US
ENDIF
END SELECT
ELSE
```

```
SELECT CASE(IEC)
CASE(1)
IF (ICD==1) THEN
VEL_OTHER => OM%W
ELSE ! ICD=2
VEL_OTHER => OM%V
ENDIF
CASE(2)
IF (ICD==1) THEN
VEL_OTHER => OM%U
ELSE ! ICD=2
VEL_OTHER => OM%W
ENDIF
CASE(3)
IF (ICD==1) THEN
VEL_OTHER => OM%V
ELSE ! ICD=2
VEL_OTHER => OM%U
ENDIF
END SELECT
ENDIF

WGT = EDGE_INTERPOLATION_FACTOR(IE,ICD)
OMW = 1._EB-WGT

SELECT CASE(IEC)
CASE(1)
IF (ICD==1) THEN
VEL_GHOST = WGT*VEL_OTHER(IIO(ICD),JJO(ICD),KKO(ICD)) + OMW*VEL_OTHER(IIO(ICD),JJO(ICD),KKO(ICD)-1)
ELSE ! ICD=2
VEL_GHOST = WGT*VEL_OTHER(IIO(ICD),JJO(ICD),KKO(ICD)) + OMW*VEL_OTHER(IIO(ICD),JJO(ICD)-1,KKO(ICD))
ENDIF
CASE(2)
IF (ICD==1) THEN
VEL_GHOST = WGT*VEL_OTHER(IIO(ICD),JJO(ICD),KKO(ICD)) + OMW*VEL_OTHER(IIO(ICD)-1,JJO(ICD),KKO(ICD))
ELSE ! ICD=2
VEL_GHOST = WGT*VEL_OTHER(IIO(ICD),JJO(ICD),KKO(ICD)) + OMW*VEL_OTHER(IIO(ICD),JJO(ICD),KKO(ICD)-1)
ENDIF
CASE(3)
IF (ICD==1) THEN
VEL_GHOST = WGT*VEL_OTHER(IIO(ICD),JJO(ICD),KKO(ICD)) + OMW*VEL_OTHER(IIO(ICD),JJO(ICD)-1,KKO(ICD))
ELSE ! ICD==2
VEL_GHOST = WGT*VEL_OTHER(IIO(ICD),JJO(ICD),KKO(ICD)) + OMW*VEL_OTHER(IIO(ICD)-1,JJO(ICD),KKO(ICD))
ENDIF
END SELECT

ENDIF INTERPOLATION_IF

! Set ghost cell values at edge of computational domain

SELECT CASE(IEC)
CASE(1)
IF (JJ==0    .AND. IOR== 2) WW(II,JJ,KK)   = VEL_GHOST
IF (JJ==JBAR .AND. IOR==-2) WW(II,JJ+1,KK) = VEL_GHOST
IF (KK==0    .AND. IOR== 3) VV(II,JJ,KK)   = VEL_GHOST
IF (KK==KBAR .AND. IOR==-3) VV(II,JJ,KK+1) = VEL_GHOST
IF (CORRECTOR .AND. JJ>0 .AND. JJ<JBAR .AND. KK>0 .AND. KK<KBAR) THEN
IF (ICD==1) THEN
W_Y(II,JJ,KK) = 0.5_EB*(VEL_GHOST+VEL_GAS)
ELSE ! ICD=2
V_Z(II,JJ,KK) = 0.5_EB*(VEL_GHOST+VEL_GAS)
ENDIF
ENDIF
CASE(2)
IF (II==0    .AND. IOR== 1) WW(II,JJ,KK)   = VEL_GHOST
IF (II==IBAR .AND. IOR==-1) WW(II+1,JJ,KK) = VEL_GHOST
IF (KK==0    .AND. IOR== 3) UU(II,JJ,KK)   = VEL_GHOST
IF (KK==KBAR .AND. IOR==-3) UU(II,JJ,KK+1) = VEL_GHOST
IF (CORRECTOR .AND. II>0 .AND. II<IBAR .AND. KK>0 .AND. KK<KBAR) THEN
IF (ICD==1) THEN
U_Z(II,JJ,KK) = 0.5_EB*(VEL_GHOST+VEL_GAS)
ELSE ! ICD=2
W_X(II,JJ,KK) = 0.5_EB*(VEL_GHOST+VEL_GAS)
ENDIF
ENDIF
CASE(3)
IF (II==0    .AND. IOR== 1) VV(II,JJ,KK)   = VEL_GHOST
IF (II==IBAR .AND. IOR==-1) VV(II+1,JJ,KK) = VEL_GHOST
IF (JJ==0    .AND. IOR== 2) UU(II,JJ,KK)   = VEL_GHOST
IF (JJ==JBAR .AND. IOR==-2) UU(II,JJ+1,KK) = VEL_GHOST
IF (CORRECTOR .AND. II>0 .AND. II<IBAR .AND. JJ>0 .AND. JJ<JBAR) THEN
IF (ICD==1) THEN
V_X(II,JJ,KK) = 0.5_EB*(VEL_GHOST+VEL_GAS)
ELSE ! ICD=2
U_Y(II,JJ,KK) = 0.5_EB*(VEL_GHOST+VEL_GAS)
ENDIF
ENDIF
END SELECT
```

```

ENDDO ORIENTATION_LOOP
ENDDO SIGN_LOOP

! Cycle out of the EDGE_LOOP if no tangential gradients have been altered.

IF (.NOT.ANY(ALTERED_GRADIENT)) CYCLE EDGE_LOOP

! If the edge is on an interpolated boundary, and all cells around it are not solid, cycle

IF (INTERPOLATED_EDGE) THEN
PROCESS_EDGE = .FALSE.
DO IS=5,8
IF (SOLID(IJKE(IS,IE))) PROCESS_EDGE = .TRUE.
ENDDO
IF (.NOT.PROCESS_EDGE) CYCLE EDGE_LOOP
ENDIF

! Loop over all 4 normal directions and compute vorticity and stress tensor components for each

SIGN_LOOP_2: DO I_SGN=-1,1,2
ORIENTATION_LOOP_2: DO ICD=1,2
IF (ICD==1) THEN
ICDO=2
ELSE ! ICD=2
ICDO=1
ENDIF
ICD_SGN = I_SGN*ICD
IF (ALTERED_GRADIENT(ICD_SGN)) THEN
DUIDXJ_USE(ICD) =    DUIDXJ(ICD_SGN)
MU_DUIDXJ_USE(ICD) = MU_DUIDXJ(ICD_SGN)
ELSEIF (ALTERED_GRADIENT(-ICD_SGN)) THEN
DUIDXJ_USE(ICD) =    DUIDXJ(-ICD_SGN)
MU_DUIDXJ_USE(ICD) = MU_DUIDXJ(-ICD_SGN)
ELSE
CYCLE ORIENTATION_LOOP_2
ENDIF
ICDO_SGN = I_SGN*ICDO
IF (ALTERED_GRADIENT(ICDO_SGN)) THEN
DUIDXJ_USE(ICDO) =    DUIDXJ(ICDO_SGN)
MU_DUIDXJ_USE(ICDO) = MU_DUIDXJ(ICDO_SGN)
ELSEIF (ALTERED_GRADIENT(-ICDO_SGN)) THEN
DUIDXJ_USE(ICDO) =    DUIDXJ(-ICDO_SGN)
MU_DUIDXJ_USE(ICDO) = MU_DUIDXJ(-ICDO_SGN)
ELSE
DUIDXJ_USE(ICDO) = 0._EB
MU_DUIDXJ_USE(ICDO) = 0._EB
ENDIF
OME_E(ICD_SGN,IE) =    DUIDXJ_USE(1) -    DUIDXJ_USE(2)
TAU_E(ICD_SGN,IE) = MU_DUIDXJ_USE(1) + MU_DUIDXJ_USE(2)
ENDDO ORIENTATION_LOOP_2
ENDDO SIGN_LOOP_2

ENDDO EDGE_LOOP

! Store cell node averages of the velocity components in UVW_GHOST for use in Smokeview only

IF (CORRECTOR) THEN
DO K=0,KBAR
DO J=0,JBAR
DO I=0,IBAR
IC = CELL_INDEX(I,J,K)
IF (IC==0) CYCLE
IF (U_Y(I,J,K) >-1.E5_EB) UVW_GHOST(IC,1) = U_Y(I,J,K)
IF (U_Z(I,J,K) >-1.E5_EB) UVW_GHOST(IC,1) = U_Z(I,J,K)
IF (V_X(I,J,K) >-1.E5_EB) UVW_GHOST(IC,2) = V_X(I,J,K)
IF (V_Z(I,J,K) >-1.E5_EB) UVW_GHOST(IC,2) = V_Z(I,J,K)
IF (W_X(I,J,K) >-1.E5_EB) UVW_GHOST(IC,3) = W_X(I,J,K)
IF (W_Y(I,J,K) >-1.E5_EB) UVW_GHOST(IC,3) = W_Y(I,J,K)
ENDDO
ENDDO
ENDDO
ENDIF

T_USED(4)=T_USED(4)+SECOND()-TNOW
END SUBROUTINE VELOCITY_BC


SUBROUTINE MATCH_VELOCITY(NM)

! Force normal component of velocity to match at interpolated boundaries

INTEGER  :: NOM,II,JJ,KK,IOR,IW,IIO,JJO,KKO
INTEGER, INTENT(IN) :: NM
REAL(EB) :: UU_AVG,VV_AVG,WW_AVG,TNOW,DA_OTHER,UU_OTHER,VV_OTHER,WW_OTHER,NOM_CELLS
REAL(EB), POINTER, DIMENSION(:,:,:) :: UU=>NULL(),VV=>NULL(),WW=>NULL(),OM_UU=>NULL(),OM_VV=>NULL(),OM_WW=>NULL()
TYPE (OMESH_TYPE), POINTER :: OM=>NULL()
TYPE (MESH_TYPE), POINTER  :: M2=>NULL()
```

```
TYPE (WALL_TYPE), POINTER :: WC=>NULL()
TYPE (EXTERNAL_WALL_TYPE), POINTER :: EWC=>NULL()
IF (SOLID_PHASE_ONLY) RETURN
IF (EVACUATION_ONLY(NM)) RETURN

TNOW = SECOND()

! Assign local variable names

CALL POINT_TO_MESH(NM)

! Point to the appropriate velocity field

IF (PREDICTOR) THEN
UU => US
VV => VS
WW => WS
D_CORR = 0._EB
ELSE
UU => U
VV => V
WW => W
DS_CORR = 0._EB
ENDIF

! Loop over all external wall cells and force adjacent normal components of velocty at interpolated boundaries to
      match.

EXTERNAL_WALL_LOOP: DO IW=1,N_EXTERNAL_WALL_CELLS

WC=>WALL(IW)

IF (WC%BOUNDARY_TYPE/=INTERPOLATED_BOUNDARY) CYCLE EXTERNAL_WALL_LOOP

EWC=>EXTERNAL_WALL(IW)

II  = WC%ONE_D%II
JJ  = WC%ONE_D%JJ
KK  = WC%ONE_D%KK
IOR = WC%ONE_D%IOR
NOM = EWC%NOM
OM => OMESH(NOM)
M2 => MESHES(NOM)

! Determine the area of the interpolated cell face

DA_OTHER = 0._EB

SELECT CASE(ABS(IOR))
CASE(1)
IF (PREDICTOR) OM_UU => OM%US
IF (CORRECTOR) OM_UU => OM%U
DO KKO=EWC%KKO_MIN,EWC%KKO_MAX
DO JJO=EWC%JJO_MIN,EWC%JJO_MAX
DO IIO=EWC%IIO_MIN,EWC%IIO_MAX
DA_OTHER = DA_OTHER + M2%DY(JJO)*M2%DZ(KKO)
ENDDO
ENDDO
ENDDO
CASE(2)
IF (PREDICTOR) OM_VV => OM%VS
IF (CORRECTOR) OM_VV => OM%V
DO KKO=EWC%KKO_MIN,EWC%KKO_MAX
DO JJO=EWC%JJO_MIN,EWC%JJO_MAX
DO IIO=EWC%IIO_MIN,EWC%IIO_MAX
DA_OTHER = DA_OTHER + M2%DX(IIO)*M2%DZ(KKO)
ENDDO
ENDDO
ENDDO
CASE(3)
IF (PREDICTOR) OM_WW => OM%WS
IF (CORRECTOR) OM_WW => OM%W
DO KKO=EWC%KKO_MIN,EWC%KKO_MAX
DO JJO=EWC%JJO_MIN,EWC%JJO_MAX
DO IIO=EWC%IIO_MIN,EWC%IIO_MAX
DA_OTHER = DA_OTHER + M2%DX(IIO)*M2%DY(JJO)
ENDDO
ENDDO
ENDDO
END SELECT

! Determine the normal component of velocity from the other mesh and use it for average

SELECT CASE(IOR)

CASE( 1)

UU_OTHER = 0._EB
```

```
DO KKO=EWC%KKO_MIN,EWC%KKO_MAX
DO JJO=EWC%JJO_MIN,EWC%JJO_MAX
DO IIO=EWC%IIO_MIN,EWC%IIO_MAX
UU_OTHER = UU_OTHER + OM_UU(IIO,JJO,KKO)*M2%DY(JJO)*M2%DZ(KKO)/DA_OTHER
IF (EWC%AREA_RATIO>0.9_EB) OM_UU(IIO,JJO,KKO) = 0.5_EB*(OM_UU(IIO,JJO,KKO)+UU(0,JJ,KK))
ENDDO
ENDDO
ENDDO
UU_AVG = 0.5_EB*(UU(0,JJ,KK) + UU_OTHER)
IF (PREDICTOR) D_CORR(IW) = DS_CORR(IW) + 0.5*(UU_AVG-UU(0,JJ,KK))*R(0)*RDX(1)*RRN(1)
IF (CORRECTOR) DS_CORR(IW) = (UU_AVG-UU(0,JJ,KK))*R(0)*RDX(1)*RRN(1)
UVW_SAVE(IW) = UU(0,JJ,KK)
UU(0,JJ,KK)  = UU_AVG

CASE(-1)

UU_OTHER = 0._EB
DO KKO=EWC%KKO_MIN,EWC%KKO_MAX
DO JJO=EWC%JJO_MIN,EWC%JJO_MAX
DO IIO=EWC%IIO_MIN,EWC%IIO_MAX
UU_OTHER = UU_OTHER + OM_UU(IIO-1,JJO,KKO)*M2%DY(JJO)*M2%DZ(KKO)/DA_OTHER
IF (EWC%AREA_RATIO>0.9_EB) OM_UU(IIO-1,JJO,KKO) = 0.5_EB*(OM_UU(IIO-1,JJO,KKO)+UU(IBAR,JJ,KK))
ENDDO
ENDDO
ENDDO
UU_AVG = 0.5_EB*(UU(IBAR,JJ,KK) + UU_OTHER)
IF (PREDICTOR) D_CORR(IW) = DS_CORR(IW) - 0.5*(UU_AVG-UU(IBAR,JJ,KK))*R(IBAR)*RDX(IBAR)*RRN(IBAR)
IF (CORRECTOR) DS_CORR(IW) = -(UU_AVG-UU(IBAR,JJ,KK))*R(IBAR)*RDX(IBAR)*RRN(IBAR)
UVW_SAVE(IW) = UU(IBAR,JJ,KK)
UU(IBAR,JJ,KK) = UU_AVG

CASE( 2)

VV_OTHER = 0._EB
DO KKO=EWC%KKO_MIN,EWC%KKO_MAX
DO JJO=EWC%JJO_MIN,EWC%JJO_MAX
DO IIO=EWC%IIO_MIN,EWC%IIO_MAX
VV_OTHER = VV_OTHER + OM_VV(IIO,JJO,KKO)*M2%DX(IIO)*M2%DZ(KKO)/DA_OTHER
IF (EWC%AREA_RATIO>0.9_EB) OM_VV(IIO,JJO,KKO) = 0.5_EB*(OM_VV(IIO,JJO,KKO)+VV(II,0,KK))
ENDDO
ENDDO
ENDDO
VV_AVG = 0.5_EB*(VV(II,0,KK) + VV_OTHER)
IF (PREDICTOR) D_CORR(IW) = DS_CORR(IW) + 0.5*(VV_AVG-VV(II,0,KK))*RDY(1)
IF (CORRECTOR) DS_CORR(IW) = (VV_AVG-VV(II,0,KK))*RDY(1)
UVW_SAVE(IW) = VV(II,0,KK)
VV(II,0,KK)  = VV_AVG

CASE(-2)

VV_OTHER = 0._EB
DO KKO=EWC%KKO_MIN,EWC%KKO_MAX
DO JJO=EWC%JJO_MIN,EWC%JJO_MAX
DO IIO=EWC%IIO_MIN,EWC%IIO_MAX
VV_OTHER = VV_OTHER + OM_VV(IIO,JJO-1,KKO)*M2%DX(IIO)*M2%DZ(KKO)/DA_OTHER
IF (EWC%AREA_RATIO>0.9_EB) OM_VV(IIO,JJO-1,KKO) = 0.5_EB*(OM_VV(IIO,JJO-1,KKO)+VV(II,JBAR,KK))
ENDDO
ENDDO
ENDDO
VV_AVG = 0.5_EB*(VV(II,JBAR,KK) + VV_OTHER)
IF (PREDICTOR) D_CORR(IW) = DS_CORR(IW) - 0.5*(VV_AVG-VV(II,JBAR,KK))*RDY(JBAR)
IF (CORRECTOR) DS_CORR(IW) = -(VV_AVG-VV(II,JBAR,KK))*RDY(JBAR)
UVW_SAVE(IW)     = VV(II,JBAR,KK)
VV(II,JBAR,KK) = VV_AVG

CASE( 3)

WW_OTHER = 0._EB
DO KKO=EWC%KKO_MIN,EWC%KKO_MAX
DO JJO=EWC%JJO_MIN,EWC%JJO_MAX
DO IIO=EWC%IIO_MIN,EWC%IIO_MAX
WW_OTHER = WW_OTHER + OM_WW(IIO,JJO,KKO)*M2%DX(IIO)*M2%DY(JJO)/DA_OTHER
IF (EWC%AREA_RATIO>0.9_EB) OM_WW(IIO,JJO,KKO) = 0.5_EB*(OM_WW(IIO,JJO,KKO)+WW(II,JJ,0))
ENDDO
ENDDO
ENDDO
WW_AVG = 0.5_EB*(WW(II,JJ,0) + WW_OTHER)
IF (PREDICTOR) D_CORR(IW) = DS_CORR(IW) + 0.5*(WW_AVG-WW(II,JJ,0))*RDZ(1)
IF (CORRECTOR) DS_CORR(IW) = (WW_AVG-WW(II,JJ,0))*RDZ(1)
UVW_SAVE(IW) = WW(II,JJ,0)
WW(II,JJ,0)  = WW_AVG

CASE(-3)

WW_OTHER = 0._EB
DO KKO=EWC%KKO_MIN,EWC%KKO_MAX
DO JJO=EWC%JJO_MIN,EWC%JJO_MAX
DO IIO=EWC%IIO_MIN,EWC%IIO_MAX
```

```fortran
WW_OTHER = WW_OTHER + OMWW(IIO,JJO,KKO-1)*M2%DX(IIO)*M2%DY(JJO)/DA_OTHER
IF (EWC%AREA_RATIO>0.9_EB) OMWW(IIO,JJO,KKO-1) = 0.5_EB*(OMWW(IIO,JJO,KKO-1)+WW(II,JJ,KBAR))
ENDDO
ENDDO
ENDDO
WW_AVG = 0.5_EB*(WW(II,JJ,KBAR) + WW_OTHER)
IF (PREDICTOR) D_CORR(IW) = DS_CORR(IW) - 0.5*(WW_AVG-WW(II,JJ,KBAR))*RDZ(KBAR)
IF (CORRECTOR) DS_CORR(IW) = -(WW_AVG-WW(II,JJ,KBAR))*RDZ(KBAR)
UVW_SAVE(IW)    = WW(II,JJ,KBAR)
WW(II,JJ,KBAR) = WW_AVG

END SELECT

! Save velocity components at the ghost cell midpoint

U_GHOST(IW) = 0._EB
V_GHOST(IW) = 0._EB
W_GHOST(IW) = 0._EB

IF (PREDICTOR) OMUU => OM%US
IF (CORRECTOR) OMUU => OM%U
IF (PREDICTOR) OMVV => OM%VS
IF (CORRECTOR) OMVV => OM%V
IF (PREDICTOR) OMWW => OM%WS
IF (CORRECTOR) OMWW => OM%W

DO KKO=EWC%KKO_MIN,EWC%KKO_MAX
DO JJO=EWC%JJO_MIN,EWC%JJO_MAX
DO IIO=EWC%IIO_MIN,EWC%IIO_MAX
U_GHOST(IW) = U_GHOST(IW) + 0.5_EB*(OMUU(IIO,JJO,KKO)+OMUU(IIO-1,JJO,KKO))
V_GHOST(IW) = V_GHOST(IW) + 0.5_EB*(OMVV(IIO,JJO,KKO)+OMVV(IIO,JJO-1,KKO))
W_GHOST(IW) = W_GHOST(IW) + 0.5_EB*(OMWW(IIO,JJO,KKO)+OMWW(IIO,JJO,KKO-1))
ENDDO
ENDDO
ENDDO
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)
U_GHOST(IW) = U_GHOST(IW)/NOM_CELLS
V_GHOST(IW) = V_GHOST(IW)/NOM_CELLS
W_GHOST(IW) = W_GHOST(IW)/NOM_CELLS

ENDDO EXTERNAL_WALL_LOOP

T_USED(4)=T_USED(4)+SECOND()-TNOW
END SUBROUTINE MATCH_VELOCITY


SUBROUTINE MATCH_VELOCITY_FLUX(NM)

! Force normal component of velocity flux to match at interpolated boundaries

INTEGER  :: NOM,II,JJ,KK,IOR,IW,IIO,JJO,KKO
INTEGER, INTENT(IN) :: NM
REAL(EB) :: TNOW,DA_OTHER,FVX_OTHER,FVY_OTHER,FVZ_OTHER
TYPE (OMESH_TYPE), POINTER :: OM
TYPE (MESH_TYPE), POINTER :: M2
TYPE (WALL_TYPE), POINTER :: WC
TYPE (EXTERNAL_WALL_TYPE), POINTER :: EWC

IF (NMESHES==1) RETURN
IF (SOLID_PHASE_ONLY) RETURN
IF (EVACUATION_ONLY(NM)) RETURN

TNOW = SECOND()

! Assign local variable names

CALL POINT_TO_MESH(NM)

! Loop over all cell edges and determine the appropriate velocity BCs

EXTERNAL_WALL_LOOP: DO IW=1,N_EXTERNAL_WALL_CELLS

WC=>WALL(IW)
EWC=>EXTERNAL_WALL(IW)
IF (WC%BOUNDARY_TYPE/=INTERPOLATED_BOUNDARY) CYCLE EXTERNAL_WALL_LOOP

II  = WC%ONE_D%II
JJ  = WC%ONE_D%JJ
KK  = WC%ONE_D%KK
IOR = WC%ONE_D%IOR
NOM = EWC%NOM
OM => OMESH(NOM)
M2 => MESHES(NOM)

! Determine the area of the interpolated cell face

DA_OTHER = 0._EB

```

```
SELECT CASE(ABS(IOR))
CASE(1)
DO KKO=EWC%KKO_MIN,EWC%KKO_MAX
DO JJO=EWC%JJO_MIN,EWC%JJO_MAX
DO IIO=EWC%IIO_MIN,EWC%IIO_MAX
DA_OTHER = DA_OTHER + M2%DY(JJO)*M2%DZ(KKO)
ENDDO
ENDDO
ENDDO
CASE(2)
DO KKO=EWC%KKO_MIN,EWC%KKO_MAX
DO JJO=EWC%JJO_MIN,EWC%JJO_MAX
DO IIO=EWC%IIO_MIN,EWC%IIO_MAX
DA_OTHER = DA_OTHER + M2%DX(IIO)*M2%DZ(KKO)
ENDDO
ENDDO
ENDDO
CASE(3)
DO KKO=EWC%KKO_MIN,EWC%KKO_MAX
DO JJO=EWC%JJO_MIN,EWC%JJO_MAX
DO IIO=EWC%IIO_MIN,EWC%IIO_MAX
DA_OTHER = DA_OTHER + M2%DX(IIO)*M2%DY(JJO)
ENDDO
ENDDO
ENDDO
END SELECT

! Determine the normal component of velocity from the other mesh and use it for average

SELECT CASE(IOR)

CASE( 1)

FVX_OTHER = 0._EB
DO KKO=EWC%KKO_MIN,EWC%KKO_MAX
DO JJO=EWC%JJO_MIN,EWC%JJO_MAX
DO IIO=EWC%IIO_MIN,EWC%IIO_MAX
FVX_OTHER = FVX_OTHER + OM%FVX(IIO,JJO,KKO)*M2%DY(JJO)*M2%DZ(KKO)/DA_OTHER
ENDDO
ENDDO
ENDDO
FVX(0,JJ,KK) = 0.5_EB*(FVX(0,JJ,KK) + FVX_OTHER)

CASE(-1)

FVX_OTHER = 0._EB
DO KKO=EWC%KKO_MIN,EWC%KKO_MAX
DO JJO=EWC%JJO_MIN,EWC%JJO_MAX
DO IIO=EWC%IIO_MIN,EWC%IIO_MAX
FVX_OTHER = FVX_OTHER + OM%FVX(IIO-1,JJO,KKO)*M2%DY(JJO)*M2%DZ(KKO)/DA_OTHER
ENDDO
ENDDO
ENDDO
FVX(IBAR,JJ,KK) = 0.5_EB*(FVX(IBAR,JJ,KK) + FVX_OTHER)

CASE( 2)

FVY_OTHER = 0._EB
DO KKO=EWC%KKO_MIN,EWC%KKO_MAX
DO JJO=EWC%JJO_MIN,EWC%JJO_MAX
DO IIO=EWC%IIO_MIN,EWC%IIO_MAX
FVY_OTHER = FVY_OTHER + OM%FVY(IIO,JJO,KKO)*M2%DX(IIO)*M2%DZ(KKO)/DA_OTHER
ENDDO
ENDDO
ENDDO
FVY(II,0,KK) = 0.5_EB*(FVY(II,0,KK) + FVY_OTHER)

CASE(-2)

FVY_OTHER = 0._EB
DO KKO=EWC%KKO_MIN,EWC%KKO_MAX
DO JJO=EWC%JJO_MIN,EWC%JJO_MAX
DO IIO=EWC%IIO_MIN,EWC%IIO_MAX
FVY_OTHER = FVY_OTHER + OM%FVY(IIO,JJO-1,KKO)*M2%DX(IIO)*M2%DZ(KKO)/DA_OTHER
ENDDO
ENDDO
ENDDO
FVY(II,JBAR,KK) = 0.5_EB*(FVY(II,JBAR,KK) + FVY_OTHER)

CASE( 3)

FVZ_OTHER = 0._EB
DO KKO=EWC%KKO_MIN,EWC%KKO_MAX
DO JJO=EWC%JJO_MIN,EWC%JJO_MAX
DO IIO=EWC%IIO_MIN,EWC%IIO_MAX
FVZ_OTHER = FVZ_OTHER + OM%FVZ(IIO,JJO,KKO)*M2%DX(IIO)*M2%DY(JJO)/DA_OTHER
ENDDO
ENDDO
```

```
ENDDO
FVZ(II,JJ,0) = 0.5_EB*(FVZ(II,JJ,0) + FVZ_OTHER)

CASE(-3)

FVZ_OTHER = 0._EB
DO KKO=EWC%KKO_MIN,EWC%KKO_MAX
DO JJO=EWC%JJO_MIN,EWC%JJO_MAX
DO IIO=EWC%IIO_MIN,EWC%IIO_MAX
FVZ_OTHER = FVZ_OTHER + OM%FVZ(IIO,JJO,KKO-1)*M2%DX(IIO)*M2%DY(JJO)/DA_OTHER
ENDDO
ENDDO
ENDDO
FVZ(II,JJ,KBAR) = 0.5_EB*(FVZ(II,JJ,KBAR) + FVZ_OTHER)

END SELECT

ENDDO EXTERNAL_WALL_LOOP

T_USED(4)=T_USED(4)+SECOND()-TNOW
END SUBROUTINE MATCH_VELOCITY_FLUX


SUBROUTINE CHECK_STABILITY(DT,DT_NEW,NM)

! Checks the Courant and Von Neumann stability criteria, and if necessary, reduces the time step accordingly

USE PHYSICAL_FUNCTIONS, ONLY: GET_SPECIFIC_HEAT
INTEGER, INTENT(IN) :: NM
REAL(EB), INTENT(IN) :: DT
REAL(EB) :: UODX,VODY,WODZ,UVW,UVWMAX,R_DX2,MU_MAX,MUTRM,CP,ZZ_GET(1:N_TRACKED_SPECIES),PART_CFL,MU_TMP
REAL(EB) :: DT_NEW(NMESHES)
INTEGER  :: I,J,K,IW,IIG,JJG,KKG
TYPE(WALL_TYPE), POINTER :: WC=>NULL()
REAL(EB), PARAMETER :: DT_EPS = 1.E-10_EB

IF (EVACUATION_ONLY(NM)) RETURN

UVWMAX = 0._EB
VN     = 0._EB
MUTRM  = 1.E-9_EB
R_DX2  = 1.E-9_EB

! Determine max CFL number from all grid cells

DO K=1,KBAR
DO J=1,JBAR
DO I=1,IBAR
IF (SOLID(CELL_INDEX(I,J,K))) CYCLE
UODX = MAXVAL(ABS(US(I-1:I,J,K)))*RDX(I)
VODY = MAXVAL(ABS(VS(I,J-1:J,K)))*RDY(J)
WODZ = MAXVAL(ABS(WS(I,J,K-1:K)))*RDZ(K)
SELECT CASE (CFL_VELOCITY_NORM)
CASE(0) ; UVW = MAX(UODX,VODY,WODZ) + ABS(DS(I,J,K))
CASE(1) ; UVW = UODX + VODY + WODZ  + ABS(DS(I,J,K))
CASE(2) ; UVW = SQRT(UODX**2+VODY**2+WODZ**2) + ABS(DS(I,J,K))
CASE(3) ; UVW = MAX(UODX,VODY,WODZ)
END SELECT
IF (UVW>=UVWMAX) THEN
UVWMAX = UVW
ICFL = I
JCFL = J
KCFL = K
ENDIF
ENDDO
ENDDO
ENDDO

HEAT_TRANSFER_IF: IF (CHECK_HT) THEN
WALL_LOOP: DO IW=1,N_EXTERNAL_WALL_CELLS+N_INTERNAL_WALL_CELLS
WC=>WALL(IW)
IF (WC%BOUNDARY_TYPE/=SOLID_BOUNDARY) CYCLE WALL_LOOP
IIG = WC%ONE_D%IIG
JJG = WC%ONE_D%JJG
KKG = WC%ONE_D%KKG
ZZ_GET(1:N_TRACKED_SPECIES) = ZZS(IIG,JJG,KKG,1:N_TRACKED_SPECIES)
CALL GET_SPECIFIC_HEAT(ZZ_GET,CP,TMP(IIG,JJG,KKG))
UVW = WC%ONE_D%HEAT_TRANS_COEF/(WC%ONE_D%RHO_F*CP) * WC%ONE_D%RDN
IF (UVW>=UVWMAX) THEN
UVWMAX = UVW
ICFL=IIG
JCFL=JJG
KCFL=KKG
ENDIF
ENDDO WALL_LOOP
ENDIF HEAT_TRANSFER_IF

CFL = DT*UVWMAX
```

```
PART_CFL = DT*PART_UVWMAX

! Determine max Von Neumann Number for fine grid calcs

PARABOLIC_IF: IF (CHECK_VN) THEN

MU_MAX = 0._EB
DO K=1,KBAR
DO J=1,JBAR
I_LOOP: DO I=1,IBAR
IF (SOLID(CELL_INDEX(I,J,K))) CYCLE I_LOOP
MU_TMP = MAX(D_Z_MAX(I,J,K),MU(I,J,K)/RHOS(I,J,K))
IF (MU_TMP>=MU_MAX) THEN
MU_MAX = MU_TMP
I_VN=I
J_VN=J
K_VN=K
ENDIF
ENDDO I_LOOP
ENDDO
ENDDO

IF (TWO_D) THEN
R_DX2 = RDX(I_VN)**2 + RDZ(K_VN)**2
ELSE
R_DX2 = RDX(I_VN)**2 + RDY(J_VN)**2 + RDZ(K_VN)**2
ENDIF

MUTRM = MU_MAX
VN = DT*2._EB*R_DX2*MUTRM

ENDIF PARABOLIC_IF

! Adjust time step size if necessary

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
DT_NEW(NM) = DT
IF (CFL<=CFL_MIN .AND. VN<VN_MIN .AND. PART_CFL<PARTICLE_CFL_MIN .AND. .NOT.LOCK_TIME_STEP) THEN
SELECT CASE (RESTRICT_TIME_STEP)
CASE (.TRUE.); DT_NEW(NM) = MIN(1.1_EB*DT,DT_INITIAL)
CASE (.FALSE.); DT_NEW(NM) =     1.1_EB*DT
END SELECT
CHANGE_TIME_STEP_INDEX(NM) = 1
ENDIF
ELSE
DT_NEW(NM) = 0.9_EB*MIN( CFL_MAX/MAX(UVWMAX,DT_EPS)                    , &
VN_MAX/(2._EB*R_DX2*MAX(MUTRM,DT_EPS))     , &
PARTICLE_CFL_MAX/MAX(PART_UVWMAX,DT_EPS)  , &
DT*DRAG_CFL_MAX/MAX(DRAG_CFL,DT_EPS))
DRAG_CFL = 0._EB
CHANGE_TIME_STEP_INDEX(NM) = -1
ENDIF

END SUBROUTINE CHECK_STABILITY


SUBROUTINE BAROCLINIC_CORRECTION(T,NM)

! Add baroclinic term to the momentum equation

USE MATH_FUNCTIONS, ONLY: EVALUATE_RAMP
REAL(EB), INTENT(IN) :: T
INTEGER, INTENT(IN) :: NM
REAL(EB), POINTER, DIMENSION(:,:,:) :: UU=>NULL(),VV=>NULL(),WW=>NULL(),RHOP=>NULL(),HP=>NULL(),RHMK=>NULL(),RRHO
    =>NULL()
INTEGER  :: I,J,K,II,JJ,KK,IIG,JJG,KKG,IOR,IW
REAL(EB) :: P_EXTERNAL,TSI,TIME_RAMP_FACTOR,DUMMY,UN,TNOW
LOGICAL  :: INFLOW
TYPE(VENTS_TYPE), POINTER :: VT=>NULL()
TYPE(WALL_TYPE), POINTER :: WC=>NULL()

TNOW=SECOND()
CALL POINT_TO_MESH(NM)

! If the baroclinic torque term has been added to the momentum equation RHS, subtract it off.

IF (BAROCLINIC_TERMS_ATTACHED) THEN
FVX = FVX - FVX_B
FVY = FVY - FVY_B
FVZ = FVZ - FVZ_B
ENDIF

BAROCLINIC_TERMS_ATTACHED = .TRUE.

RHMK => WORK1 ! p=rho*(H-K)
RRHO => WORK2 ! reciprocal of rho

```

```
IF (PREDICTOR) THEN
UU => U
VV => V
WW => W
RHOP=>RHO
IF (PRESSURE_ITERATIONS>1) THEN
HP => H
ELSE
HP => HS
ENDIF
! Note: this ordering of HP=HS in PREDICTOR is required to achieve 2nd order temporal convergence.
! We should rethink our notation and re-examine whether both H and HS are required.
ELSE
UU => US
VV => VS
WW => WS
RHOP=>RHOS
IF (PRESSURE_ITERATIONS>1) THEN
HP => HS
ELSE
HP => H
ENDIF
ENDIF

! Compute pressure and 1/rho in each grid cell

!$OMP PARALLEL PRIVATE(WC, VT, TSI, TIME_RAMP_FACTOR, P_EXTERNAL, &
!$OMP& II, JJ, KK, IOR, IIG, JJG, KKG, UN, INFLOW)
!$OMP DO SCHEDULE(static)
DO K=0,KBP1
DO J=0,JBP1
DO I=0,IBP1
RHMK(I,J,K) = RHOP(I,J,K)*(HP(I,J,K)-KRES(I,J,K))
RRHO(I,J,K) = 1._EB/RHOP(I,J,K)
ENDDO
ENDDO
ENDDO
!$OMP END DO

! Set baroclinic term to zero at outflow boundaries and P_EXTERNAL at inflow boundaries

!$OMP MASTER
EXTERNAL_WALL_LOOP: DO IW=1,N_EXTERNAL_WALL_CELLS
WC=>WALL(IW)
IF (WC%BOUNDARY_TYPE/=OPEN_BOUNDARY) CYCLE EXTERNAL_WALL_LOOP
IF (WC%VENT_INDEX>0) THEN
VT => VENTS(WC%VENT_INDEX)
IF (ABS(WC%ONE_D%T_IGN-T_BEGIN)<=SPACING(WC%ONE_D%T_IGN) .AND. VT%PRESSURE_RAMP_INDEX>=1) THEN
TSI = T
ELSE
TSI = T - T_BEGIN
ENDIF
TIME_RAMP_FACTOR = EVALUATE_RAMP(TSI,DUMMY,VT%PRESSURE_RAMP_INDEX)
P_EXTERNAL = TIME_RAMP_FACTOR*VT%DYNAMIC_PRESSURE
ENDIF
II  = WC%ONE_D%II
JJ  = WC%ONE_D%JJ
KK  = WC%ONE_D%KK
IOR = WC%ONE_D%IOR
IIG = WC%ONE_D%IIG
JJG = WC%ONE_D%JJG
KKG = WC%ONE_D%KKG
INFLOW = .FALSE.
IOR_SELECT: SELECT CASE(IOR)
CASE( 1); UN = UU(II,JJ,KK)
CASE(-1); UN = UU(II-1,JJ,KK)
CASE( 2); UN = VV(II,JJ,KK)
CASE(-2); UN = VV(II,JJ-1,KK)
CASE( 3); UN = WW(II,JJ,KK)
CASE(-3); UN = WW(II,JJ,KK-1)
END SELECT IOR_SELECT
IF (UN*SIGN(1._EB,REAL(IOR,EB))>TWO_EPSILON_EB) INFLOW=.TRUE.
IF (INFLOW) THEN
RHMK(II,JJ,KK) = 2._EB*P_EXTERNAL - RHMK(IIG,JJG,KKG)  ! Pressure at inflow boundary is P_EXTERNAL
ELSE
RHMK(II,JJ,KK) = -RHMK(IIG,JJG,KKG)                    ! No baroclinic correction for outflow boundary
ENDIF
ENDDO EXTERNAL_WALL_LOOP
!$OMP END MASTER
!$OMP BARRIER

! Compute baroclinic term in the x momentum equation, p*d/dx(1/rho)

!$OMP DO SCHEDULE(static)
DO K=1,KBAR
DO J=1,JBAR
DO I=0,IBAR
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)/ &
```

```
(RHOP(I+1,J,K)+RHOP(I,J,K))
FVX(I,J,K) = FVX(I,J,K) + FVX_B(I,J,K)
ENDDO
ENDDO
ENDDO
!$OMP END DO nowait

! Compute baroclinic term in the y momentum equation, p*d/dy(1/rho)

IF (.NOT.TWO_D) THEN
!$OMP DO SCHEDULE(static)
DO K=1,KBAR
DO J=0,JBAR
DO I=1,IBAR
FVY_B(I,J,K) = -(RHMK(I,J,K)*RHOP(I,J+1,K)+RHMK(I,J+1,K)*RHOP(I,J,K))*(RRHO(I,J+1,K)-RRHO(I,J,K))*RDYN(J)/ &
(RHOP(I,J+1,K)+RHOP(I,J,K))
FVY(I,J,K) = FVY(I,J,K) + FVY_B(I,J,K)
ENDDO
ENDDO
ENDDO
!$OMP END DO nowait
ENDIF

! Compute baroclinic term in the z momentum equation, p*d/dz(1/rho)

!$OMP DO SCHEDULE(static)
DO K=0,KBAR
DO J=1,JBAR
DO I=1,IBAR
FVZ_B(I,J,K) = -(RHMK(I,J,K)*RHOP(I,J,K+1)+RHMK(I,J,K+1)*RHOP(I,J,K))*(RRHO(I,J,K+1)-RRHO(I,J,K))*RDZN(K)/ &
(RHOP(I,J,K+1)+RHOP(I,J,K))
FVZ(I,J,K) = FVZ(I,J,K) + FVZ_B(I,J,K)
ENDDO
ENDDO
ENDDO
!$OMP END DO nowait
!$OMP END PARALLEL

T_USED(4) = T_USED(4) + SECOND() - TNOW
END SUBROUTINE BAROCLINIC_CORRECTION


! ---------------------------------------- PATCH_VELOCITY_FLUX ----------------------------------------

SUBROUTINE PATCH_VELOCITY_FLUX(DT,NM)

! The user may specify a polynomial profile using the PROP and DEVC lines. This routine
! specifies the source term in the momentum equation to drive the local velocity toward
! this user-specified value, in much the same way as the immersed boundary method
! (see IBM_VELOCITY_FLUX).

USE DEVICE_VARIABLES, ONLY: DEVICE_TYPE,PROPERTY_TYPE,N_DEVC,DEVICE,PROPERTY
USE TRAN, ONLY: GINV
REAL(EB), INTENT(IN) :: DT
TYPE(DEVICE_TYPE), POINTER :: DV=>NULL()
TYPE(PROPERTY_TYPE), POINTER :: PY=>NULL()
INTEGER, INTENT(IN) :: NM
INTEGER :: N,I,J,K,IC1,IC2,I1,I2,J1,J2,K1,K2
REAL(EB), POINTER, DIMENSION(:,:,:) :: UU=>NULL(),VV=>NULL(),WW=>NULL(),HP=>NULL()
REAL(EB) :: VELP,DX0,DY0,DZ0

IF (PREDICTOR) THEN
UU => U
VV => V
WW => W
HP => H
ELSE
UU => US
VV => VS
WW => WS
HP => HS
ENDIF

DEVC_LOOP: DO N=1,N_DEVC

DV=>DEVICE(N)
IF (DV%QUANTITY/='VELOCITY PATCH') CYCLE DEVC_LOOP
IF (DV%PROP_INDEX<1)                CYCLE DEVC_LOOP
IF (.NOT.DEVICE(DV%DEVC_INDEX(1))%CURRENT_STATE) CYCLE DEVC_LOOP

IF (DV%X1 > XF .OR. DV%X2 < XS .OR. &
DV%Y1 > YF .OR. DV%Y2 < YS .OR. &
DV%Z1 > ZF .OR. DV%Z2 < ZS) CYCLE DEVC_LOOP

PY=>PROPERTY(DV%PROP_INDEX)

I_VEL_SELECT: SELECT CASE(PY%I_VEL)

```

```
CASE(1) I_VEL_SELECT

I1 = MAX(0,     NINT( GINV(DV%X1-XS,1,NM)*RDXI    )-1)
I2 = MIN(IBAR,NINT( GINV(DV%X2-XS,1,NM)*RDXI    )+1)
J1 = MAX(0,     NINT( GINV(DV%Y1-YS,2,NM)*RDETA  )-1)
J2 = MIN(JBAR,NINT( GINV(DV%Y2-YS,2,NM)*RDETA  )+1)
K1 = MAX(0,     NINT( GINV(DV%Z1-ZS,3,NM)*RDZETA )-1)
K2 = MIN(KBAR,NINT( GINV(DV%Z2-ZS,3,NM)*RDZETA )+1)

DO K=K1,K2
DO J=J1,J2
DO I=I1,I2

IC1 = CELL_INDEX(I,J,K)
IC2 = CELL_INDEX(I+1,J,K)
IF (SOLID(IC1) .OR. SOLID(IC2)) CYCLE

IF ( X(I)<DV%X1 .OR.  X(I)>DV%X2) CYCLE ! Inefficient but simple
IF (YC(J)<DV%Y1 .OR. YC(J)>DV%Y2) CYCLE
IF (ZC(K)<DV%Z1 .OR. ZC(K)>DV%Z2) CYCLE

DX0 =  X(I)-DV%X
DY0 = YC(J)-DV%Y
DZ0 = ZC(K)-DV%Z
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)) &
+ 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)) &
+ 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))

FVX(I,J,K) = -RDXN(I)*(HP(I+1,J,K)-HP(I,J,K)) - (VELP-UU(I,J,K))/DT
ENDDO
ENDDO
ENDDO

CASE(2) I_VEL_SELECT

I1 = MAX(0,     NINT( GINV(DV%X1-XS,1,NM)*RDXI    )-1)
I2 = MIN(IBAR,NINT( GINV(DV%X2-XS,1,NM)*RDXI    )+1)
J1 = MAX(0,     NINT( GINV(DV%Y1-YS,2,NM)*RDETA  )-1)
J2 = MIN(JBAR,NINT( GINV(DV%Y2-YS,2,NM)*RDETA  )+1)
K1 = MAX(0,     NINT( GINV(DV%Z1-ZS,3,NM)*RDZETA )-1)
K2 = MIN(KBAR,NINT( GINV(DV%Z2-ZS,3,NM)*RDZETA )+1)

DO K=K1,K2
DO J=J1,J2
DO I=I1,I2

IC1 = CELL_INDEX(I,J,K)
IC2 = CELL_INDEX(I,J+1,K)

IF (SOLID(IC1) .OR. SOLID(IC2)) CYCLE

IF (XC(I)<DV%X1 .OR. XC(I)>DV%X2) CYCLE
IF ( Y(J)<DV%Y1 .OR.  Y(J)>DV%Y2) CYCLE
IF (ZC(K)<DV%Z1 .OR. ZC(K)>DV%Z2) CYCLE

DX0 = XC(I)-DV%X
DY0 =  Y(J)-DV%Y
DZ0 = ZC(K)-DV%Z
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)) &
+ 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)) &
+ 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))

FVY(I,J,K) = -RDYN(J)*(HP(I,J+1,K)-HP(I,J,K)) - (VELP-VV(I,J,K))/DT
ENDDO
ENDDO
ENDDO

CASE(3) I_VEL_SELECT

I1 = MAX(0,     NINT( GINV(DV%X1-XS,1,NM)*RDXI    )-1)
I2 = MIN(IBAR,NINT( GINV(DV%X2-XS,1,NM)*RDXI    )+1)
J1 = MAX(0,     NINT( GINV(DV%Y1-YS,2,NM)*RDETA  )-1)
J2 = MIN(JBAR,NINT( GINV(DV%Y2-YS,2,NM)*RDETA  )+1)
K1 = MAX(0,     NINT( GINV(DV%Z1-ZS,3,NM)*RDZETA )-1)
K2 = MIN(KBAR,NINT( GINV(DV%Z2-ZS,3,NM)*RDZETA )+1)

DO K=K1,K2
DO J=J1,J2
DO I=I1,I2

IC1 = CELL_INDEX(I,J,K)
IC2 = CELL_INDEX(I,J,K+1)
IF (SOLID(IC1) .OR. SOLID(IC2)) CYCLE

IF (XC(I)<DV%X1 .OR. XC(I)>DV%X2) CYCLE
IF (YC(J)<DV%Y1 .OR. YC(J)>DV%Y2) CYCLE
IF ( Z(K)<DV%Z1 .OR.  Z(K)>DV%Z2) CYCLE

```

```
DX0 = XC(I)-DV%X
DY0 = YC(J)-DV%Y
DZ0 =  Z(K)-DV%Z
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)) &
+ 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)) &
+ 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))

FVZ(I,J,K) = -RDZN(K)*(HP(I,J,K)-HP(I,J,K+1)) - (VELP-WW(I,J,K))/DT
ENDDO
ENDDO
ENDDO

END SELECT I_VEL_SELECT

ENDDO DEVC_LOOP

END SUBROUTINE PATCH_VELOCITY_FLUX


! ---------------------------- WALL_VELOCITY_NO_GRADH ----------------------------

SUBROUTINE WALL_VELOCITY_NO_GRADH(DT,STORE_UN)

! This routine recomputes velocities on wall cells, such that the correct
! normal derivative of H is used on the projection. It is only used when the Poisson equation
! for the pressure is solved .NOT. PRES_ON_WHOLE_DOMAIN (i.e. using the GLMAT solver).

REAL(EB), INTENT(IN) :: DT
LOGICAL, INTENT(IN) :: STORE_UN

! Local variables:
INTEGER :: IIG,JJG,KKG,IOR,IW
REAL(EB) :: DHDN, VEL_N
TYPE (WALL_TYPE), POINTER :: WC
REAL(EB), SAVE, ALLOCATABLE, DIMENSION(:) :: UN_WALLS


IF (PRES_ON_WHOLE_DOMAIN) RETURN

STORE_UN_COND : IF ( STORE_UN ) THEN

! These velocities from the beginning of step are needed for the velocity fix on wall cells at the corrector
! phase (i.e. the loops in VELOCITY_CORRECTOR will change U,V,W to wrong reults using (HP1-HP)/DX gradients,
! when the pressure solver in the GLMAT solver.
IF(ALLOCATED(UN_WALLS)) DEALLOCATE(UN_WALLS)
ALLOCATE( UN_WALLS(1:N_EXTERNAL_WALL_CELLS+N_INTERNAL_WALL_CELLS) )
UN_WALLS(:) = 0._EB

STORE_LOOP : DO IW=1,N_EXTERNAL_WALL_CELLS+N_INTERNAL_WALL_CELLS

WC => WALL(IW)
IIG    = WC%ONE_D%IIG
JJG    = WC%ONE_D%JJG
KKG    = WC%ONE_D%KKG
IOR    = WC%ONE_D%IOR

SELECT CASE(IOR)
CASE( IAXIS)
UN_WALLS(IW) = U(IIG-1,JJG  ,KKG  )
CASE(-IAXIS)
UN_WALLS(IW) = U(IIG  ,JJG  ,KKG  )
CASE( JAXIS)
UN_WALLS(IW) = V(IIG  ,JJG-1,KKG  )
CASE(-JAXIS)
UN_WALLS(IW) = V(IIG  ,JJG  ,KKG  )
CASE( KAXIS)
UN_WALLS(IW) = W(IIG  ,JJG  ,KKG-1)
CASE(-KAXIS)
UN_WALLS(IW) = W(IIG  ,JJG  ,KKG  )
END SELECT

ENDDO STORE_LOOP

RETURN

ENDIF STORE_UN_COND

! Case of not storing, recompute INTERNAL_WALL_CELL velocities, taking into acct that DHDN=0._EB:
PREDICTOR_COND : IF (PREDICTOR) THEN

! Loop internal wall cells -> on OBST surfaces:
WALL_CELL_LOOP_1: DO IW=1,N_EXTERNAL_WALL_CELLS+N_INTERNAL_WALL_CELLS

WC => WALL(IW)

IF (WC%BOUNDARY_TYPE/=SOLID_BOUNDARY .AND. WC%BOUNDARY_TYPE/=NULL_BOUNDARY) CYCLE

IIG    = WC%ONE_D%IIG
```

```
JJG    = WC%ONE_D%JJG
KKG    = WC%ONE_D%KKG
IOR    = WC%ONE_D%IOR

DHDN=0._EB ! Set the normal derivative of H to zero for solids.

SELECT CASE(IOR)
CASE( IAXIS)
US(IIG-1,JJG   ,KKG  ) = (U(IIG-1,JJG   ,KKG  ) - DT*( FVX(IIG-1,JJG   ,KKG  ) + DHDN ))
CASE(-IAXIS)
US(IIG   ,JJG   ,KKG  ) = (U(IIG   ,JJG   ,KKG  ) - DT*( FVX(IIG   ,JJG   ,KKG  ) + DHDN ))
CASE( JAXIS)
VS(IIG   ,JJG-1,KKG  ) = (V(IIG   ,JJG-1,KKG  ) - DT*( FVY(IIG   ,JJG-1,KKG  ) + DHDN ))
CASE(-JAXIS)
VS(IIG   ,JJG   ,KKG  ) = (V(IIG   ,JJG   ,KKG  ) - DT*( FVY(IIG   ,JJG   ,KKG  ) + DHDN ))
CASE( KAXIS)
WS(IIG   ,JJG   ,KKG-1) = (W(IIG   ,JJG   ,KKG-1) - DT*( FVZ(IIG   ,JJG   ,KKG-1) + DHDN ))
CASE(-KAXIS)
WS(IIG   ,JJG   ,KKG  ) = (W(IIG   ,JJG   ,KKG  ) - DT*( FVZ(IIG   ,JJG   ,KKG  ) + DHDN ))
END SELECT

ENDDO WALL_CELL_LOOP_1

ELSE ! Corrector

! Loop internal wall cells -> on OBST surfaces:
WALL_CELL_LOOP_2: DO IW=1,N_EXTERNAL_WALL_CELLS+N_INTERNAL_WALL_CELLS

WC => WALL(IW)

IF (WC%BOUNDARY_TYPE/=SOLID_BOUNDARY .AND. WC%BOUNDARY_TYPE/=NULL_BOUNDARY) CYCLE

IIG    = WC%ONE_D%IIG
JJG    = WC%ONE_D%JJG
KKG    = WC%ONE_D%KKG
IOR    = WC%ONE_D%IOR

DHDN=0._EB ! Set the normal derivative of H to zero for solids.

VEL_N = UN_WALLS(IW)

SELECT CASE(IOR)
CASE( IAXIS)                                ! | - Problem with this is it was modified in VELOCITY_CORRECTOR,
! V   => Store the untouched U normal on internal WALLs.
U(IIG-1,JJG   ,KKG  ) = 0.5_EB*(                        VEL_N + US(IIG-1,JJG   ,KKG  ) - &
DT*( FVX(IIG-1,JJG   ,KKG  ) + DHDN ))
CASE(-IAXIS)
U(IIG   ,JJG   ,KKG  ) = 0.5_EB*(                        VEL_N + US(IIG   ,JJG   ,KKG  ) - &
DT*( FVX(IIG   ,JJG   ,KKG  ) + DHDN ))
CASE( JAXIS)
V(IIG   ,JJG-1,KKG  ) = 0.5_EB*(                        VEL_N + VS(IIG   ,JJG-1,KKG  ) - &
DT*( FVY(IIG   ,JJG-1,KKG  ) + DHDN ))
CASE(-JAXIS)
V(IIG   ,JJG   ,KKG  ) = 0.5_EB*(                        VEL_N + VS(IIG   ,JJG   ,KKG  ) - &
DT*( FVY(IIG   ,JJG   ,KKG  ) + DHDN ))
CASE( KAXIS)
W(IIG   ,JJG   ,KKG-1) = 0.5_EB*(                        VEL_N + WS(IIG   ,JJG   ,KKG-1) - &
DT*( FVZ(IIG   ,JJG   ,KKG-1) + DHDN ))
CASE(-KAXIS)
W(IIG   ,JJG   ,KKG  ) = 0.5_EB*(                        VEL_N + WS(IIG   ,JJG   ,KKG  ) - &
DT*( FVZ(IIG   ,JJG   ,KKG  ) + DHDN ))
END SELECT

ENDDO WALL_CELL_LOOP_2

DEALLOCATE(UN_WALLS)

ENDIF PREDICTOR_COND

RETURN
END SUBROUTINE WALL_VELOCITY_NO_GRADH

END MODULE VELO
```

A.4 *mod_canopy.f90*

```
module penalization

REAL :: penalizationParameter, blendingParameter, dampingParameter, &
penXmin, penXmax, penYmin, penYmax, penZmin, penZmax, &
mX, mY, mZ, b, timesttep, newtimestep

INTEGER :: points_Xmin, points_Xmax, points_Ymin, points_Ymax, points_Zmin, points_Zmax, i1, j1, temp,
      cntr=0, &
meshresX, meshresY, meshresZ, pena_I, pena_J, pena_K

character(len=256), dimension(10) :: dataFileName
integer, dimension(1000) :: pendat_size

logical :: trunks
integer :: ntrunks, fileread, npen, penX, penY, penZ, IERROR
double precision :: trnk_min,trnk_max,eta
double precision, dimension(:,:), allocatable :: trnk_loc , pendat
double precision, dimension(:,:), allocatable :: pendat_store
real, dimension(:,:,:), allocatable :: penU0, penV0, penW0
double precision, dimension(:), allocatable :: arraytime



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_{st} Australiasian Fluid Mechanics Conference, Adelaide, Australia $10 - 13$ December, 2018

(c)

21st Australasian Fluid Mechanics Conference
Adelaide, Australia
10-13 December 2018

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

S. Singha Roy[1], **D. Sutherland**[1,2,3], **N.Khan**[1] **and K. Moinuddin**[1,2]

[1]Institute for Sustainable Industries and Livable Cities
Victoria University, Melbourne, Victoria 3030, Australia

[2]Bushfire and Natural Hazards Cooperative Research Centre
Melbourne, Victoria 3002, Australia

[3]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)

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

where $u(z)$ is the wind velocity at height z, u_* is the friction velocity, κ 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).

$$N = max(\frac{V_B}{\sigma^3}) \tag{2}$$

where (σ) 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):

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

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 $\sim 5.5 m/s$ 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.

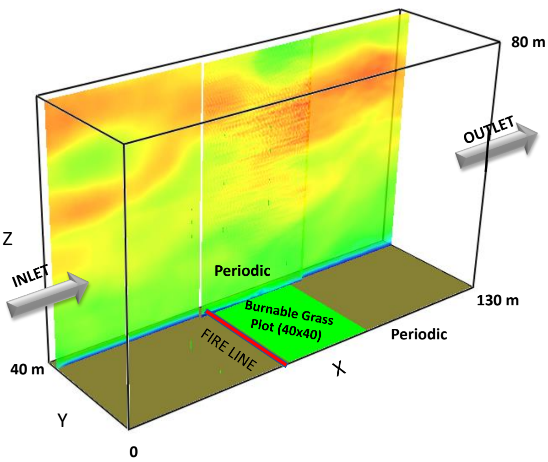


Figure 1: *Domain of simulation showing the dimensions, fire plot, fire line and establishment of ABL.*

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

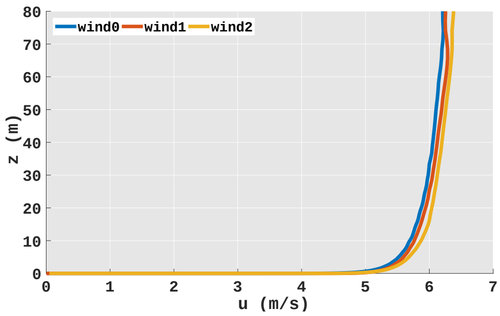


Figure 2: *The mean velocity profile comparison for 3 wind cases over the fire ground.*

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

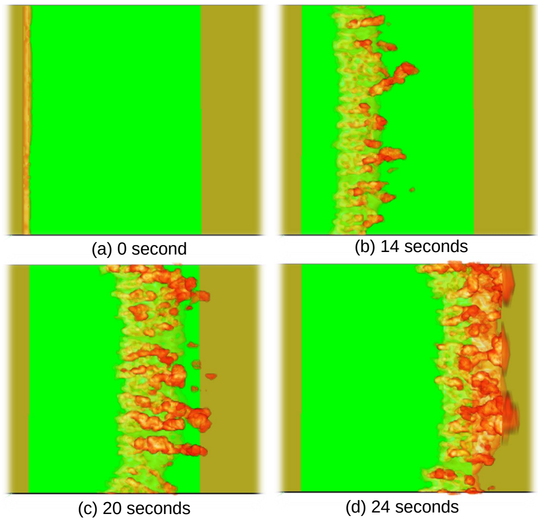


Figure 3: *Fire propagation contour for fire1.*

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.

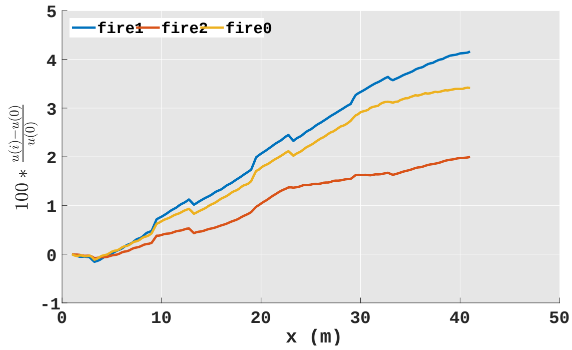


Figure 4: *Wind speed variation over the burnable-grass plot at 2.04 m height.*

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

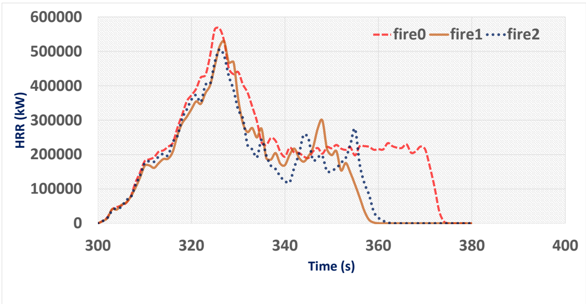


Figure 5: *Heat Release Rate (HRR) as functions of time*

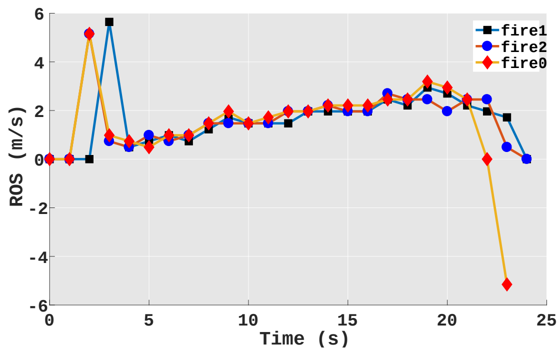


Figure 6: *ROS vs Time comparison over fire plot.*

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 achieves 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 models. *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 multiphase 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_τ 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:

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

where κ 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)):

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

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

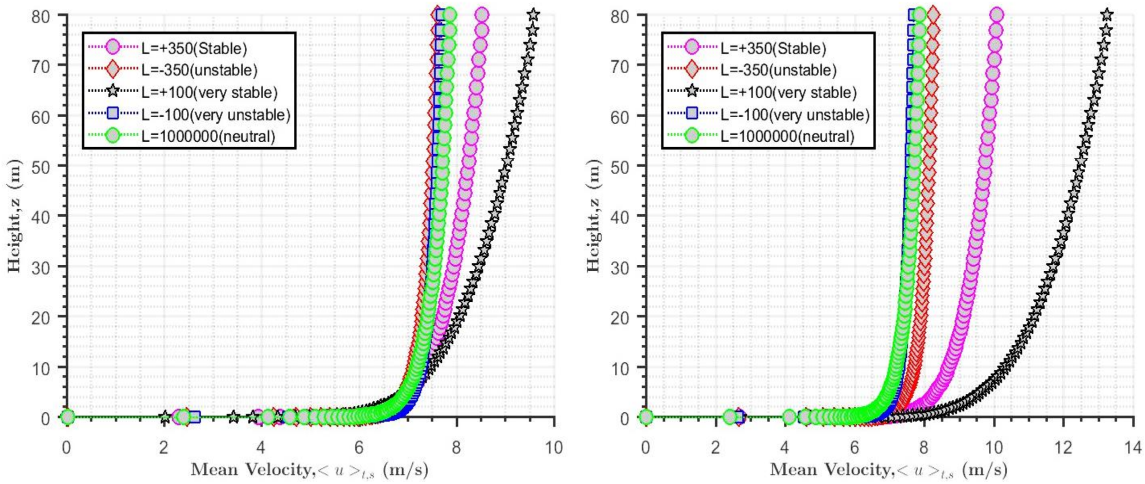
for $z >> z_0$, where z_0 is the aerodynamic roughness, T_* is the scaling potential temperature, T_0 is the ground level potential temperature and ψ_m and ψ_h are are similarity functions which are obtained from measurements. The most common similarity functions used are those proposed by Dyer (1974). 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**
Very Unstable	-200 $<=$ L $<$ 0	-100
Unstable	-500 $<=$ L $<$ -0.02	-350
Neutral	$\|L\| > 500$	1000000
Stable	200 $<$ L $<=$ 500	350
Very stable	0 $<$ L $<=$ 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).

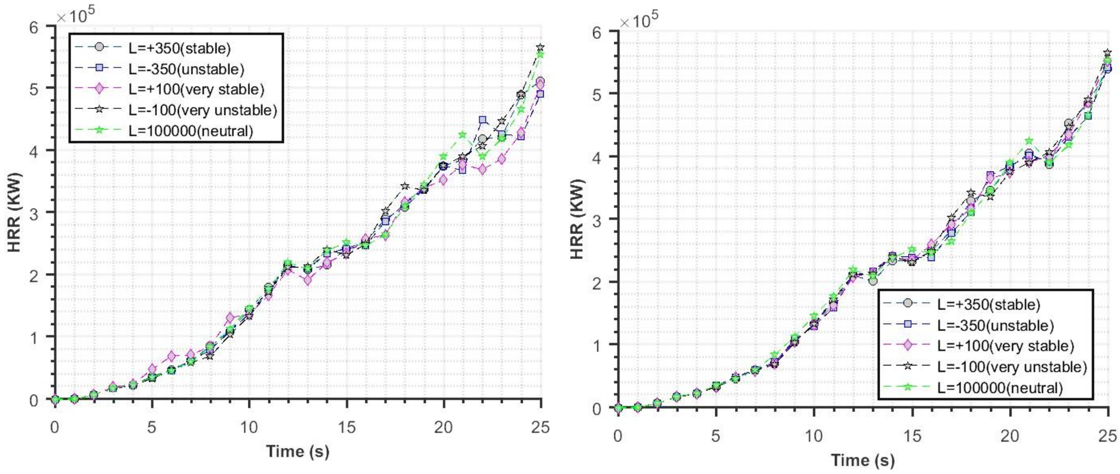


(a) u10 values matched immediately before ignition line to ~7m/s

(b) u10 values matched at inlet to 6.5m/s

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 $\sim$ 7m/s, the inlet velocity may vary ; (b) The u_{10} velocities matched at the inlet to* 6.5*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 supresses 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 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.

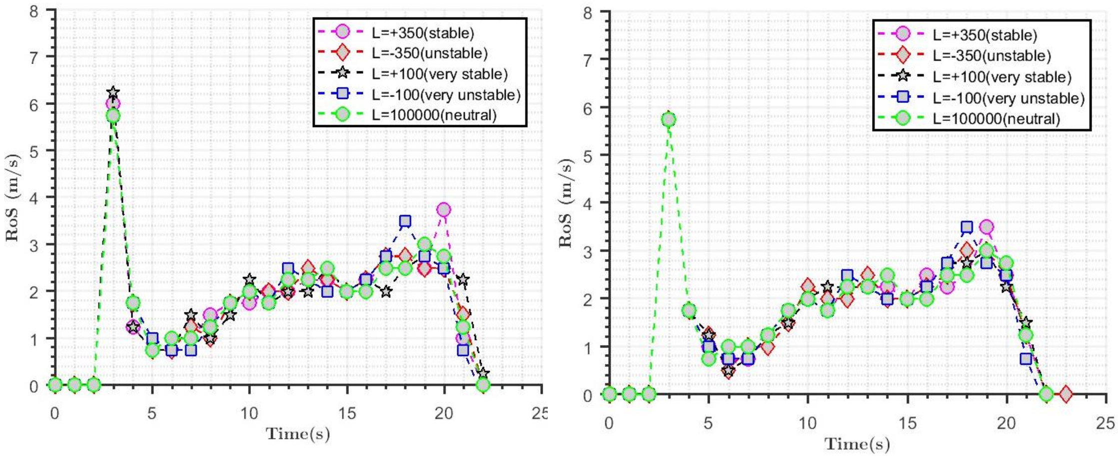


(a) HRR for u10 values matched immediately before ignition line to ~7m/s

(b) HRR for u10 values matched at inlet to 6.5m/s

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 $\sim$ 7m/s for various stabilities; (b) The u_{10} velocity is matched at the inlet to* 6.5*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 suprising. It is 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)).



(a) RoS for u10 values matched immediately before ignition line to ~7m/s

(b) RoS for u10 values matched at inlet to 6.5m/s

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* ~ 7*m/s for various stabilities; (b) The* u_{10} *velocity is matched at the inlet to* 6.5*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.