WRF-Fire User's Guide

July 3, 2010

Note: This document applies to WRF-Fire as distributed directly by the developers at openwfm.org and it is continuously updated along with the software. The version of this document included in the WRF-ARW User's Guide applies to the frozen software version as released with WRF.

Contributors to the WRF-Fire software

Janice Coen and Terry Clark developed the physical fire model in CAWFE
Ned Patton ported the fire code from CAWFE and interfaced it with WRF.
Jan Mandel is currently leading the software development. He wrote the fire component in WRF-Fire using Ned Patton's interface, with advice from Janice Coen and assistance from Jonathan Beezley and Minjeong Kim.
John Michalakes modified WRF to support refined grids (submesh) for the fire code.
Jonathan Beezley has further modified WRF to support the fire software, provided the software engineering infrastructure, and developed the modified version of WPS for WRF with the fire model.
Volodymyr Kondratenko has improved memory handing in the computation of fuel.
Adam Kochanski has contributed variable atmospheric bubble initialization.

Table of Contents

Introduction

A wildland fire module has been added to WRF to allow users to model the growth of a wildland fire and the dynamic feedbacks with the atmosphere. It is implemented as a physics package with two-way coupling between the fire behavior and the atmospheric environment allowing the fire to alter the atmosphere surrounding it, i.e. 'create its own weather'. Other documents describe the derivation of the model – here we address the mechanics, options, parameters, and datasets for using this module.

The wildland fire module is currently a simple two-dimensional model of a surface fire. The user specifies the time, location, and shape of a fire ignition. The evolution of the fireline, the interface enclosing the burning region, is implemented by the level set method. The level set function is advanced by the Runge-Kutta method of order 2, with spatial discretization by the Godunov method. The rate at which this interface expands is calculated at all points along it using a point-based semi-empirical formula for estimating the rate of spread of the surface fire based upon the Rothermel formula, which calculates the fire rate of spread as a function of local fuel conditions, wind, and terrain slope. Importantly, the winds used to drive the fire are interpolated from nearby low-level wind velocities, which are themselves perturbed by the fire. Once the fireline has passed by, the ignited fuel continues to burn - the mass of fuel is assumed to decay exponentially with time after ignition, the rate depending on the size of the fuel particles making up the fuel complex. The fuel burned in each time step is converted to sensible and latent heat source terms for the lowest levels of the WRF atmospheric model state, where the water vapor source arises from the intrinsic moisture in cellulosic fuels and the additional moisture, the fuel moisture content, held by fuels The fire may not progress to locations where the local fuel moisture content is greater than the moisture content of extinction,

Additional parameters and datasets beyond a standard WRF atmospheric simulation are required and are described here, The surface fuel available to be burned at each point is categorized using the Anderson classification system for "fuel models" (3 grass-dominated types, 4 shrub-dominated types, 3 types of forest litter, and 3 levels of logging slash) which we will henceforth refer to as "fuel categories" to limit confusion. Each of these fuel categories is assigned a set of typical properties consisting of the fuel load (the mass per unit area) and numerous physical properties having to do with fuel geometry, arrangement, and physical makeup. The user may make the fuels spatially homogeneous by using one fuel category for the whole domain, alter fuel properties, add custom fuel categories, or (for real data experiments) project a spatially heterogenous map of fuel categories onto the domain from fuel mapping datasets. The user also sets the number of ignitions, their time, location, and shape, and the fuel moisture content, an important factor in fire behavior.

One time step of the fire model is performed for every WRF time step. The fire model runs on a refined grid, which covers the same region as the innermost WRF domain. The fire module supports both distributed and shared memory parallel execution.

Other References

- WRF-Fire documentation, in particular the Technical description, available at http://www.openwfm.org/wiki/WRF-Fire_documentation

- Users may wish to review Anderson's fuel classification system (Anderson, H. E. 1982. *Aids to determining fuel models for estimating fire behavior.* USDA For. Serv. Gen. Tech. Rep. INT-122, 22p. Intermt. For. and Range Exp. Stn., Ogden, Utah 84401) at http://www.fs.fed.us/rm/pubs_int/int_gtr122.pdf .

- The original report introducing Rothermel's semi-empirical formulas (Rothermel, R. C. 1972. *A mathematical model for predicting fire spread in wildland fuels.* Res. Pap. INT-115. Ogden, UT: U.S. Department of Agriculture, Intermountain Forest and Range Experiment Station. 40 p.) is available at http://www.treesearch.fs.fed.us/pubs/32533 .

WRF-Fire in idealized cases

To install WRF-Fire, follow the installation instructions in Chapter 5 to configure WRF and set up the environment. For an idealized case, use

```
./compile em_fire
```

to build WRF for one of the several supplied ideal examples. This will create the links `wrf.exe` and `ideal.exe` in the directory `test/em_fire`. The examples are in its subdirectories. The links `wrf.exe` and `ideal.exe` in the subdirectories point to the parent directory.

To run the `small` idealized example, type

```
cd test/em_fire

cp examples/small/* .

./ideal.exe

./wrf.exe
```

Other idealized examples supplied in `test/em_fire/examples` directory are

`hill`, `nested`, and `fireflux`. Each directory contains all files needed to run the example, namely `namelist.input`, `namelist.fire`, and `input_sounding`.

The file `namelist.input` contains an additional section `&fire` with parameters of the fire model and ignition coordinates. The file `namelist.fire` contains an additional namelist used to enter custom fuel properties.

Fire variables in namelist.input

Variable names	Value	Description
`&domains`		Domain definition
`sr_x`	10	Fire mesh is 10 times finer than the innermost atmospheric mesh in the *x* direction. This number must be even.
`sr_y`	10	Fire mesh is 10 times finer than the innermost atmospheric mesh in the *y* direction. This number must be even.
`&fire`		Fire control
`ifire`	0	The fire model skipped
	2	The fire model runs
`fire_fuel_read`	-1	How to set the fuel data
		-1: real data from WPS
		0: set to `fire_fuel_cat` everywhere
		1: vegetation by altitude
`fire_num_ignitions`	3	Number of ignition lines, max. 5 allowed
`fire_ignition_start_x1`	1000.	*x* coordinate of the start point of the ignition line 1. All ignition coordinates are given in m from the lower left corner of the innermost domain
`fire_ignition_start_y1`	500.	*x* coordinate of the start point of the ignition line 1
`fire_ignition_end_x1`	1000.	*y* coordinate of the end point of the ignition line 1. Point ignition (actually a small circle) is obtained by specifying the end point the same as

		the start point.
fire_ignition_end_y1	1900.	*y* coordinate of the end point of the ignition line 1
fire_ignition_radius1	18.	Everything within 18 meters from the ignition location will be ignited.
fire_ignition_start_time1	2.	Time of ignition in *s* since the start of the run, at the start point of the ignition line.
fire_ignition_end_time1	502.	Time of ignition at the end point of the ignition line. The time is linearly interpolated in between.
fire_ignition_start_x2		Up to 5 ignition lines may be given. Ignition parameters with the number higher than fire_num_ignitions are not allowed.
…		
fire_ignition_end_time5		
fire_print_msg	1	0: no messages from the fire scheme 1: progress messages from the fire scheme
fire_print_file	0	0: no files written (leave as is) 1: fire model state written every 10 s into files that can be read in Matlab. See wrf/doc/README_vis.txt in the developers' version.

There are several more variables in the namelist for setting up various experiments, and then variables for developers' use only to further develop and tune the numerical methods. ***Leave the variables for the developers as they are,*** unless directed by the developers.

namelist.fire

This file serves to redefine the fuel categories if the user wishes to alter default fuel properties.

Variable names	Description
&fuel_scalars	Scalar fuel constants
cmbcnst	The energy released per unit fuel burned for cellulosic fuels (constant, 1.7433e7 J kg^{-1}).
hfgl	The threshold heat flux from a surface fire at which point a canopy fire is ignited above (in W m^{-2}).
fuelmc_g	Surface fuel, fuel moisture content (in percent expressed in

	decimal form, from 0.00 – 1.00).
fuelmc_c	Canopy fuel, fuel moisture content (in percent expressed in decimal form, from 0.00 – 1.00).
nfuelcats	Number of fuel categories defined (default: 13)
no_fuel_cat	The number of the dummy fuel category specified to be used where there is 'no fuel'
&fuel_categories	Domain specifications
fgi	The initial mass loading of surface fuel (in kg m^{-2}) in each fuel category
fueldepthm	Fuel depth (m)
savr	Fuel Surface-area-to-volume-ratio (m^{-1})
fuelmce	Fuel moisture content of extinction (in percent expressed in decimal form, from 0.00 – 1.00).
fueldens	Fuel particle density lb ft^{-3} (32 if solid, 19 if rotten)
st	Fuel particle total mineral content. (kg minerals/kg wood)
se	Fuel particle effective mineral content. (kg minerals – kg silica)/kg wood
weight	Weighting parameter that determines the slope of the mass loss curve. This can range from about 5 (fast burnup) to 1000 (40% decrease in mass over 10 minutes).
fci_d	Initial dry mass loading of canopy fuel (in kg m^{-2})
fct	The burnout time of canopy fuel once ignited (s)
ichap	Is this a chaparral category to be treated differently using an empirical rate of spread relationship that depends only on windspeed? (1: yes, this is a chaparral category and should be treated differently; 0: no, this is not a chaparral category or should not be treated differently). Primarily used for Fuel Category 4.

Running WRF-Fire on real data

Building the code

Running WRF with real data is a complicated process of converting data formats and interpolating to the model grid. This process is simplified by the WRF Preprocessing System (WPS). The purpose of this section is to summarize the use of this system and to highlight the differences in its use between fire and ordinary atmospheric simulations. For more complete documentation of WPS, see Chapter 3 of the WRF-ARW User's Guide.

WPS consists of three utility programs: `geogrid.exe`, `ungrib.exe`, and `metgrid.exe`. Each program is designed to take existing data sets and convert/interpolate them into an intermediate format. The build system for WPS is similar to that of WRF. NetCDF must be installed and the environment variable NETCDF should be set to the installation prefix. Run the configure script in the main WPS directory, pick a configuration option from the list, and then run compile. Note that

WRF itself must be built prior to compiling WPS. In addition, the build process assumes that WRF exists in `../WRFV3/`. WRF should be configured as described in Section 3 and compiled with the command

```
./compile em_real >& compile.log
```

The WPS can be configured from inside the top level directory wrf-fire/WPS with the command

```
./configure
```

and compiled in the same directory with the command

```
./compile >& compile.log
```

Upon successful completion the three binaries listed above should exist in the current directory.
Because the WPS programs are, for the most part, not processor intensive, it is not generally necessary to compile these programs for parallel execution, even if they do support it. Typical usage of WRF with real data involves doing all of the preprocessing work either locally on a workstation or on the head node of a supercomputer. The intermediate files are all architecture independent, so they can be transferred between computers safely. If you intend to use a supercomputer for the main simulation, it is advisable to generate the WPS output locally and transfer the met_em files to the computer you will be using for WRF-Fire. The met_em files are much smaller than the wrfinput and wrfbdy files and can be transported easily. This also eases the process of dealing with the dependencies of the python scripts described below because it may not be easy or even possible to meet these requirements on a shared parallel computer.

Fire variables in namelist.wps

The simulation domain is described in the file `namelist.wps`. This namelist contains four sections, one for each of the main binaries created in WPS and one shared among them all. This file, as distributed with WRF-Fire, is set up for a test case useful for testing, but in general one needs to modify it for each simulation domain. The following table lists namelist options that can be modified. Other options in this file are generally not necessary to change for WRF-Fire simulations. See the WRF-ARW User's Guide for more information.

Variable names	Description
`&share`	Shared namelist options
`max_dom`	Number of nested domains to use
`start_date/end_date`	Starting/ending date and time to process atmospheric data in the format YYYY-MM-DD_hh:mm:ss. These times should coincide with reanalysis cycles for your atmospheric data (hours 00,03,06,09,12, etc. for 3 hour NARR data). The

	simulation window in which you are interested in running must be inside this interval.
`Subgrid_ratio_[xy]`	The refinement ratio from the atmospheric grid to the fire grid.
`interval_seconds`	Number of seconds between each atmospheric dataset. (10800 for 3 hour NARR data)
`&geogrid`	Domain specifications
`parent_id`	When using nested grids, the parent of the current grid, or 0 if it is the highest level.
`parent_grid_ratio`	The refinement ratio from the parent grid (ignored for top level grid) (only 3 or 5 is supported by WRF)
`[ij]_parent_start`	The indices on the parent grid of the lower left corner of the current grid (ignored for top-level grid)
`E_we/e_sn`	The size of the grid in the x/y axis
`dx/dy`	Resolution of the grid in the x/y axis
`map_proj, true_lat[12], stand_lon`	Projection specifications. Lambert is typically used for central latitudes such as the continental US. For small domains, the projection used does not matter much.
`ref_x/ref_y`	Grid index of a reference point with known geographic location. Defaults to the center of the domain.
`ref_lon/ref_lat`	The location (longitude/latitude) of the reference point.
`geog_data_path`	Absolute or relative path to geogrid data released with WPS (http://www.mmm.ucar.edu/wrf/src/wps_files/geog_v3.1.tar.gz)

Geogrid

The geogrid executable acts exclusively on static datasets (those that don't change from day to day) such as surface elevation and land use. Because these datasets are static, they can be obtained as a single tarball from the main WPS distribution website in resolutions of 10 minutes, 2 minutes, and 30 seconds. The geogrid executable extracts from these global data sets what it needs for the current domain. While resolutions of this magnitude are acceptable for ordinary atmospheric simulations, these datasets are too coarse for a high-resolution fire simulation. In particular, a WRF-Fire simulation will require two additional data sets not present in the standard data tarball.

NFUEL_CAT

The variable NFUEL_CAT contains Anderson 13 fuel category data. This data can be obtained for the US from the USGS seamless data access server at: http://landfire.cr.usgs.gov/viewer/. Using the zooming and panning controls, the user can select the desired region with LANDFIRE 13 Anderson Fire Behavior Fuel Models box selected. This will open a new window where the user can request the data in specific projections and data formats.

ZSF

The variable ZSF contains high resolution terrain height information similar to that in the HGT variable present in atmospheric simulations; however, the standard topographical data set is only available at a maximum resolution of 30 arc seconds (about 900 meters).

For a WRF-Fire simulation, data resolution of at least 1/3 of an arc second is desirable. Such a dataset is available for the US at http://seamless.usgs.gov/. This is another USGS seamless data access server similar to that of LANDFIRE. The desired dataset on this server is listed under elevation and is called 1/3" NED.

Conversion to geogrid format

Once one has collected the necessary data from USGS servers or elsewhere, it is necessary to convert it from the given format (such as geotiff, arcgrid, etc) into geogrid format. The format specification of the geogrid format is given in the WPS section of the WRF users guide. The process of this conversion is somewhat technical; however, work is in progress to automate it. See the openwfm wiki (http://www.openwfm.org/wiki) for the latest information and links to helper scripts and source code.

Editing GEOGRID.TBL

In order to include your custom data into the WPS output, you must add a description of it in the GEOGRID.TBL file, which is located, by default, in the geogrid subdirectory of the main WPS distribution. In addition to the standard options described in the WPS users guide, there is one additional option that is necessary for defining data for fire grid variables. For them, there is a subgrid option, which is off by default. For fire grid data, one should add the option subgrid=yes to indicate that the variable should be defined on a refined subgrid with a refinement ratio defined by the subgrid_ratio_[xy] option the wps namelist. For example, typical table entries would appear as follows:

```
==============================
name=NFUEL_CAT
        priority=1
        dest_type=categorical
        dominant_only=NFUEL_CAT
        z_dim_name=fuel_cat
        halt_on_missing=yes

interp_option=default:nearest_neighbor+average_16pt+search
        rel_path=default:landfire/
        subgrid=yes
==============================
name = ZSF
        priority = 1
        dest_type = continuous
        halt_on_missing=yes
        interp_option = default:four_pt
        rel_path=default:highres_elev/
        subgrid=yes
```

This table assumes that the converted data resides as a subdirectory of the standard data directory given in the namelist under the option geog_data_path. The NFUEL_CAT data should reside in landfire/ and ZSF in highres_elev/. In general, the only options that should be modified by the user are the rel_path or abs_path options.
Once the data has been obtained and converted and the geogrid table has been properly set up, the user can run:

./geogrid.exe
which will create files such as geo_em.d01.nc that contain the interpolated static data fields.

Ungrib and Metgrid

The ungrib executable performs initial processing on atmospheric data. There are many different datasets that can be used as input to ungrib. One must obtain this data manually for a given simulation. Because fire simulations will be at a much higher resolution than most atmospheric simulations, it is advisable to get as high resolution data as possible. The 32 km resolution data from the North American Regional Reanalysis (NARR) is likely a good choice. This data is available freely from https://dss.ucar.edu/datazone/dsszone/ds608.0/NARR/3HRLY_TAR/. For real data WRF runs, three individual datasets from this website are required: 3d, flx, and sfc. To use them, download the files for the appropriate date/time and extract them somewhere on your filesystem. The files have the naming convention, NARR3D_200903_0103.tar. NARR indicates it comes from the NARR model, 3D indicates that it is the atmospheric data fields, and 200903_0103 indicates that it contains data from March 1st through 3rd of 2009. Once these files are extracted, they must be linked into the main WPS directory with the command link_grib.csh. It takes as arguments all of the files extracted from the dataset. For example, if you extracted these files to `/home/joe/mydata`, then you would issue the command:

```
./link_grib.csh /home/joe/mydata/*
```

into the top level WPS directory. Each atmospheric dataset requires a descriptor table known as a variable table to be present. WPS comes with several variable tables that work with most major data sources. These files reside in `WPS/ungrib/Variable_Tables/`. The appropriate file must be symlinked into the top level WPS directory as the file Vtable. For NARR data, type:

```
ln –sf ungrib/Variable_Tables/Vtable.NARR Vtable
```

Once this has been done, everything should be set up properly to run the ungrib command:

```
./ungrib.exe
```

Finally, the program metgrid combines the output of ungrib and geogrid to create a series of files, which can be read by WRF's real.exe. This is accomplished by

```
./metgrid.exe
```

Assuming everything completed successfully, you should now have a number of files named something like `met_em.d01.2009-03-01_12:00:00.nc`. These should be copied or linked to your `WRFV3/test/em_real` directory. If any errors occur

during execution of ungrib or metgrid, then make sure you have downloaded all of the necessary atmospheric data and that the variable table and namelist are configured properly. See the WRF-ARW User's Guide and the user's forum at http://forum.wrfforum.com/ for many helpful hints at using various datasets.

Running real case and WRF-Fire

First copy or link the met_em files generated by metgrid into test/em_real. If the simulation is being done locally, this can be accomplished by running in `wrf-fire/WRFV3/test/em_real`

```
ln –sf ../../../WPS/met_em* .
```

The namelist for WRF in the file `namelist.input` must now be edited to reflect the domain configured in WPS. In addition to the fire-specific settings listed in Section 4.3 regarding the ideal simulation, a number of other settings must be considered as listed below. See Chapter 5 for more details on these settings.

Variable	Description
`&time_control`	
`start_xxx/end_xxx`	These describe the starting and ending date and time of the simulation. They must coincide with the start_date/end_date given in namelist.wps.
`run_xxx`	The amount of time to run the simulation.
`interval_seconds`	Must coincide with interval seconds from namelist.wps.
`restart_interval`	A restart file will be generated every x minutes. The simulation can begin from a restart file rather than wrfinput. This is controlled by the namelist variable 'restart'.
`&domains`	All grid settings must match those given in the geogrid section of namelist.wps.
`num_metgrid_levels`	The number of vertical levels of the atmospheric data being used. This can be determined from the met_em files: `ncdump -h met_em* | grep 'num_metgrid_levels ='`
`sr_x/sr_y`	Fire grid refinement. Must match that given in namelist.wps as subgrid_ratio_x/subgrid_ratio_y.
`p_top_requested`	The default is 5000, but may need to be edited if there is an error executing real. If so, just set this to whatever it tells you in the error message.

Once the namelist is properly configured, run the real executable:

```
./real.exe
```

and then run wrf:

```
./wrf.exe
```

Fire state variables

A number of array variables was added to the registry to the WRF state in order to support the fire model. They are available in the `wrfout*` files created when running WRF. All fire array variables are based at the centers of the fire grid cells. Their values in the strips at the upper end of width `sr_x` in the *x* direction and `sr_y` in the *y* direction are unused and are set to zero by WRF.

The following variables can be used to interpret the fire model output.

`LFN`	level set function. Node (i,j) is on fire if LFN(i,j)<=0
`FXLONG, FXLAT`	longitude and latitude of the nodes
`FGRNHFX`	ground heat flux from the fire (W/m^2), averaged over the cell
`FGRNQFX`	ground heat flux from the fire (W/m^2), averaged over the cell
`ZSF`	terrain elevation above sea level (m)
`UF,VF`	surface wind
`FIRE_AREA`	approximate part of the area of the cell that is on fire, between 0 and 1

WRF-Fire software

This section is intended for programmers who wish to modify or extend the fire module.

WRF-Fire coding conventions

The fire model resides in WRF physics layer and conforms to *WRF Coding Conventions*, which can be found at http://www.mmm.ucar.edu/wrf/WG2/WRF_conventions.html The purpose of the conventions is to produce a transparent, fast, and maintainable code that runs in parallel without any effort on the side of the programmer of the physics layer routines. The fire code maintains the conventions as they apply to on atmospheric grids, adapts them to 2D surface-based computations, and follows analogous conventions on the fire grid. In particular, the fire code may not maintain any variables or arrays that persist between between calls, and may not use common blocks, allocatable variables, or pointer variables. Work arrays with variable bounds may be declared only as automatic; thus, they are freed between on exit from the subroutine where they are declared. All grid-sized arrays that should persist between calls to the fire code must be created in WRF through the registry mechanism, and passed to the fire code as arguments.

In addition, the fire code may not call any WRF routines directly but only through a utility layer. This is so that the fire code can be easily run standalone or coupled with

another weather code. All variables in the fire code are based at grid centers. Grid dimensions are passed in argument lists as

```
ifds,ifde,jfds,jfde, & ! fire domain dims
ifms,ifme,jfms,jfme, & ! fire memory dims
ifps,ifpe,jfps,jfpe, & ! fire patch dims (may be omitted)
ifts,ifte,jfts,jfte, & ! fire tile dims
```

Atmosphere grid 2D variables are declared with `dimension(ims:ime, jms:jme)`. Fire grid variables are declared with `dimension(ifms:ifme, jfms:jfme)`. Loops on the fire grid are always over a tile. The index variable names, the order of the loops, and the bounds are required exactly as in the code fragment below.

```
do j=jfts,jfte
     do i=ifts,ifte
              fire_variable(i,j)=…
```

In loops that need to index more than one grid at the same time time (such as computations on a submesh, or interpolation between atmosphere and fire) the index variable names must always begin with `i j`.

Parallel execution

In the fire code, all computational subroutines are called from a thread that services a single tile. There is no code running on a patch. Loops may update only array entries within in the tile but they may read other array entries in adjacent tiles, for example for interpolation or finite differences. The values of arrays that may be read between adjacent tiles are synchronized outside of the computational routines. Consequently, the values of a variable that was just updated may be used from an adjacent tile only in the next call to the computational subroutines, after the required synchronization was done outside. Synchronization within a patch is by exiting the OpenMP loop. Synchronization of the values between patches is by explicit HALO calls on the required variables and with the required width. HALOs are provided by the WRF infrastructure and specified in the registry.

The overall structure of the parallelism is spread over multiple software layers, subroutines and source files. The computation is organized in stages, controlled by the value of `ifun`.

```
! the code executes on a single patch
! if distributed memory, we are one of the MPI processes

do ifun=ifun_start,ifun_end ! what to do

  if(ifun.eq.1)then  ! this HALO needed before stage ifun=1
     #include "SOME_HALO.inc" ! communicate between patches
  endif
```

```
...
!$OMP PARALLEL DO
  do ij=1,num_tiles ! parallel loop over tiles

     if(ifun.eq.1)then   ! one of the initialization stages
          call some_atmosphere_to_fire_interpolation(…)
     endif
     ...
     call sfire_model(…,ifun,…) ! call the actual model
     ! for some values of ifun, sfire_model may do nothing

     if(ifun.eq.6)then   ! fire step done
          call some_fire_to_atmosphere_computation(…)
     endif

  enddo ! end parallel loop over tiles
  ! array variables are synchronized between tiles now

enddo ! end ifun loop
```

Software layers

The fire code is called from WRF file `dyn_em/module_first_rk_step_part1`. The output of the fire code (the heat and moisture tendencies) are stored on exit from the fire code and added to the tendencies in WRF later in a call to `update_phy_ten` from `dyn_em/module_first_rk_step_part2`
The fire code itself consists from the following files in the `phys` directory, each constituting a distinct software layer:

`module_fr_sfire_driver.F` **Fire driver** layer: Subroutines called directly from WRF. All parallelism is contained here. The rest of the code runs is called on a single tile.

`module_fr_sfire_atm.F` **Atmosphere-fire interaction** layer: routines to interface fire and the atmosphere, interpolate between fire and atmosphere.

`module_fr_sfire_model.F` **Fire model** layer: The fire model itself, callable independently of WRF. Calls the core and the physics layers. Formulated in terms of the fire grid only. Intended to be independent of particular mathematical methods used in the core layer.

`module_fr_sfire_core.F` **Core** layer: Numerical algorithms for fire propagation and fuel decay calculation. Dimensionless. Calls the physics layer for the fire spread rate.

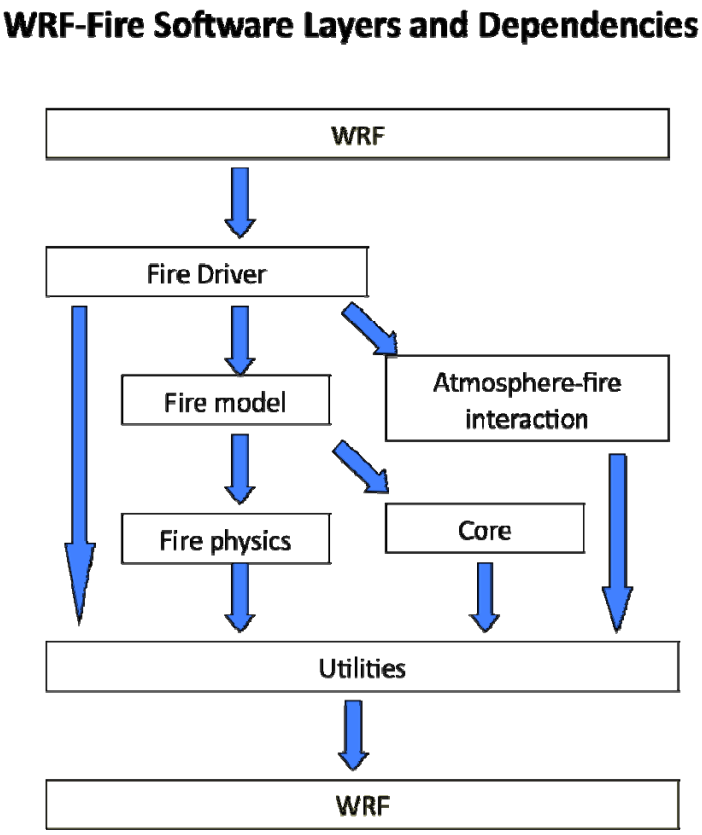
`module_fr_sfire_phys.F` **Fire physics** layer: Physical fire spread model and associated initialization.

`module_fr_sfire_util.F` **Utilities** layer: Used by all other layers. Declares scalar switches and parameters. Contains all interpolation and other service routines that may be general in nature and could be conceivably used for multiple purposes, and interface to WRF routines such as messages and error exits. To maintain independence on WRF, this is the only layer that may call any WRF routines.

`fr_sfire_params_args.h` Include file for subroutine argument lists to pass through all arguments that are needed in the fire spread rate routine in the physics layer. Necessary to write this long argument list only once given the WRF requirement that arrays may be passed as arguments only, and not shared globally, say, as pointers. Also, the include maintains the independence of the core layer on the physics layer and the independence of the fire code on WRF.

`fr_sfire_params_decl.h` Include file with the matching declarations.

The dependencies (allowed direction of subroutine and function calls) between the layers and WRF are in the following graph:



Initialization in idealized case

The initialization of model arrays in the idealized case is done in the file `dyn_em/module_initialize_fire.F`

This file was adapted from other initialization files in the same directory and extended to deal with fire model variables.

a. Vertically stretched grid

Because of the fine meshes used in fire modeling, the user may wish to search for the text `grid%znw(k)` and modify the following loop to assure a desired refinement of the vertical atmospheric grid near the Earth surface:

```
DO k=1, kde
     grid%znw(k) = (exp(-(k-1)/float(kde-1)/z_scale) &
     - exp(-1./z_scale))/(1.-exp(-1./z_scale)
ENDDO
```

b. Topography

The relevant code is found by searching for the text

```
!******* set terrain height
```

The terrain height needs to be set consistently in the atmosphere model in the array `grid%ht` and in the fire model array `grid%zsf` at the finer resolution. In the supplied examples, controlled by namelist.input variables `fire_mountain_type`, `fire_mountain_start_x`, `fire_mountain_start_y`, `fire_mountain_end_x`, `fire_mountain_end_y`, and `fire_mountain_height`, both arrays are set consistently from an algebraic formula (a cosine hill or a cosine ridge).

It is possible, though not recommended, to set only `grid%ht` and have the fire module interpolate the terrain height from the atmosphere mesh by specifying `fire_topo_from_atm=1` in namelist.input. This will result in blocky terrain with discontinuous terrain gradients, which will affect fire spread patterns.

Note that in a real run, the user should leave `fire_topo_from_atm=0` and both terrain height arrays are set consistently at the best available resolution from the WPS.

The user should not modify the code immediately after the setting of the terrain height arrays, which initializes a number of atmosphere variables consistently with the terrain height.