The BlueSky Framework Manual

Version 3.5.0

User's Guide Prepared for

USDA Forest Service
Seattle, Washington

January 2014

This document contains blank pages to accommodate two-sided printing.

The BlueSky Framework Manual

Version 3.5.0

User's Guide

STI-913028-5717-UG

Prepared by

John Stilley
Ken Craig
Sonoma Technology, Inc.
1455 N. McDowell Blvd., Suite D
Petaluma, CA 94954-6503
Ph 707.665.9900 | F 707.665.9800
sonomatech.com

Prepared for

Dr. Narasimhan K. Larkin
USDA Forest Service
400 North 34th Street #201
Seattle, Washington 98103

January 16, 2014

Table of Contents

Section **Page**

Section **Page**

List of Figures

Figure **Page**

List of Tables

Table **Page**

Abbreviations and Acronyms

Acronym	Definition
ARL	Air Resources Laboratory
BSF	BlueSky Framework
CALPUFF	California Puff model
CMAQ	Community Multiscale Air Quality model (CMAQ)
CWFIS	Canadian Wildland Fire Information System
EPM	Emissions Production Model
FCCS	Fuel Characteristic Classification System
FEER FRE	Fire Energetics and Emissions Research Fire Radiative Energy emissions model
FEPS	Fire Emission Production Simulator
FERA	Fire and Environmental Research Applications Team
FINN	Fire Inventory of NCAR
FLAMBE	Fire Locating and Modeling of Burning Emissions
GPL	General public license
HAPs	hazardous air pollutants
HARDY	Revised NFDRS fuels map for the western U.S. developed by Colin Hardy
HYSPLIT	Hybrid Single-Particle Lagrangian Integrated Trajectory model
MM5	NCAR/PSU Mesoscale Model version 5 (MM5)
INI	Configuration file
NCAR	National Center for Atmospheric Research
NFDRS	National Fire Danger Rating System
SEV Plume Rise	Sofiev, Ermakova, and Venkevich Plume Rise model
SmartFire	Satellite Mapping Automated Reanalysis Tool for Fire Incident Reconciliation
SMOKE	Sparse Matrix Operator Kernel Emissions model
VSMOKE	VSmoke Model
WRAP	Western Regional Air Partnership
WRF	Weather Research and Forecasting
WSU FRP	Washington State University Fire Radiative Power emissions model

1. What is the BlueSky Framework?

The BlueSky Framework (BSF) is a modeling framework designed to facilitate the use of diagnostic and predictive models to simulate cumulative smoke impacts, air quality, and emissions from forest, agricultural, and range fires. The BSF links a variety of independent, state-of-the-science models of fire information, fuel loading, fuel consumption, fire emissions, plume rise, and smoke dispersion. The BSF is not a model; it is a model management system whose architecture allows multiple and varied models to communicate with each other in a modular, user-driven environment.

1.1 About the BlueSky Framework

The BSF was designed to help land managers understand the impacts of fires on air quality. Though many valid, scientific models have existed for some time, users were required to run the models independently and find a way to pass the relevant information between the models. The BSF is designed to facilitate running these models in sequence and in a seamless manner.

The BSF's modeling steps track the following sequence of questions (**Figure 1-1**):

- Where and how big are the fires?
- What is the fuel available to be burned?
- How much fuel is consumed?
- What emissions are produced?
- Where do the emissions go?

One of the difficulties with a system incorporating several models is that many different models can be used to answer each of these questions. By incorporating as many models as possible, the BSF provides the greatest flexibility to the user and also enables direct model-to-model comparisons. When run as a complete system, it needs only meteorological data, fire location, and fire size to produce smoke concentrations and trajectories; however, it will also make use of any additional available information provided as input. BSF's component models are run only to produce additional information not already provided. Newer models are easily incorporated, allowing the BSF to keep pace with the state of the science. The BSF is also open source, making it accessible for developers to add enhancements.

1.2 Structure of the BlueSky Framework

The sequentially linked processing steps used in the BSF are required to simulate smoke emissions, transport, and chemistry (**Figure 1-1**). The BSF can be run piecemeal, but to run through the full framework, the BSF requires two types of information: (1) meteorological information, specifically information on the full 4-dimensional (x, y, z, t) evolution of the atmosphere; and (2) basic fire information, specifically fire size (e.g., hectares) and location (latitude/longitude). Although the BSF does not need any other information to run, additional

information is treated preferentially and will overwrite what the BSF can otherwise calculate itself. For example, any fuel loadings that are provided will be used instead of the fuel loading maps contained in the BSF.

Once the fire information is input, the BSF determines fuel loadings and moisture conditions; calculates consumption and total emissions; and then speciates and diurnally allocates the emissions. These emissions then drive the dispersion and trajectory models.

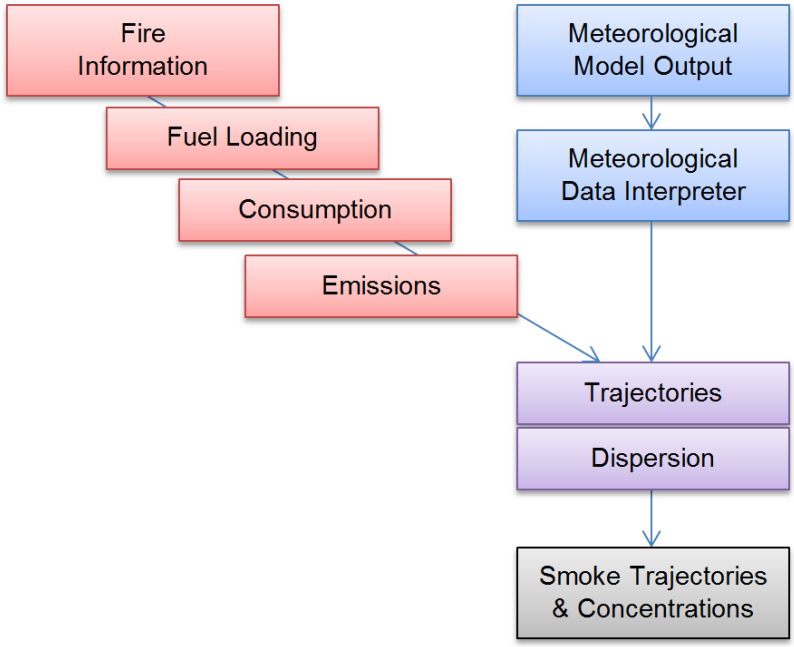

Figure 1-1. BSF model progression. Weather model output and fire information (top) are run through a series of modeling steps to calculate smoke trajectories and concentrations (bottom).

1.2.1 Modularity

The BSF can run any sequential subset of the steps in Figure 1 independently if the necessary input information is provided. This modularity allows for great flexibility. The important components of fire modeling are:

- **Meteorological Model.** The BSF requires spatially and temporally gridded meteorological information in order to run trajectory and dispersion models. The BSF can use meteorological data in the Weather Research and Forecasting (WRF) file format, but a more complete set of functionality is available when using Air Resources Laboratory (ARL) meteorological data files.
- **Fire Information.** The minimum fire information input data required by the BSF are fire location and daily fire growth. Additional fire information may be either supplied by a collection of user-modifiable default values or produced by the input modules. Fire information can be hand-entered into an input file or downloaded from the Satellite

Mapping Automated Reanalysis Tool for Fire Incident Reconciliation (SmartFire) systems. (See Section A.3 for more information about SmartFire, a system that assembles and reconciles fire information from disparate sources, such as NOAA's Hazard Mapping System, the ICS-209 reporting system, and others.)

- **Fuel Loading.** Fuel loadings may be input directly by the user or obtained via fuel reference tables. The default fuel loadings map is the Fuel Characteristic Classification System (FCCS) version 2.
- **Consumption.** After fuel loadings are determined, the BSF determines how much fuel was consumed during each phase of consumption. The default model for these calculations is the CONSUME model, version 3. CONSUME is widely used by U.S. land managers to estimate post-burn consumption for wildfire and prescribed burn reporting emissions. After the amount of fuel consumed is calculated, the consumption is allocated over time using a time profile (which can be considered an ancillary step or sub-step, Time Profile of Consumption). All of the models for this step use relatively simple, idealized profiles or curve fits.
- **Emissions.** The BSF has several models to calculate fire emissions, but the most popular is the FEPS model. FEPS calculates emissions of CO, CO_2, CH_4, and $PM_{2.5}$. The heat released by a fire, necessary for plume rise calculations, is calculated and passed to the following modeling steps.
- **Dispersion and Trajectories.** The default dispersion model in the BSF is HYSPLIT. Trajectories are simulated using the HYSPLIT Lagrangian three-dimensional particle-puff model. Twelve-hour trajectories are computed from each burn location. HYSPLIT is described in great detail in Section B.7.

1.3 List of Available Modules

The BSF facilitates the analysis of different fire science processes. These fire science processes are linked together and used to compute smoke emissions and dispersion. For each process, the BSF has several different models to choose from, each driven by a unique module wrapper. **Table 1-1** shows the modules within the BSF, organized by fire science process. This table also shows the formal module name used to invoke each model in the BSF. Modules in bold text are considered part of the standard BSF modeling pathway, and are described in greater detail in **Appendix B**. However, users are not restricted to using this standard modeling pathway; they are also permitted to develop and distribute new or modified modules in a manner consistent with the terms of the BSF license agreement (see Section 2.1.3).

Table 1-1. Modules available in the BSF as of January 2014, organized by fire science process. Modules in bold text are considered part of the standard BSF modeling pathway.

Fire Science Process	Model Name	Formal Module Wrapper Name (used to invoke the model)
Fuel Loading	**FCCS**	**FCCS**
	FINN	FINNFuelLoading
	FLAMBE	FLAMBEFuelLoading
	HARDY	Hardy
	NFDRS	NFDRS
Consumption	**Consume**	**Consume**
	FEPS	FEPSConsumption
	FINN	FINNConsumption
	FLAMBE	FLAMBEConsumption
Time Profile	**FEPS**	**FEPSTimeProfile**
	WRAP	WRAPTimeProfile
Emissions	FEER FRE	FEERFRE
	FEPS	**FEPSEmissions**
	FINN	FINNEmissions
	FLAMBE	FLAMBEEmissions
	WSU FRP	WSUFRP
Plume Rise	FEPS	FEPSPlumeRise
	SEV	SEVPlumeRise
	WRAP	**WRAPPlumeRise**
Dispersion	**HYSPLIT**	**HYSPLITDispersion**
	VSMOKE	VSMOKEDispersion
Trajectory	HYSPLIT	HYSPLITTrajectory

Note: All model names are defined in the Abbreviations and Acronyms table on page vi.

Note that some modules are not packaged with the standard BSF distribution, but must be requested and downloaded separately. There are several reasons why a module may come separately from the standard BSF distribution. For instance, the FINN modules include five gigabytes of data—an order of magnitude larger than the standard BSF distribution—and thus, they are offered separately from the core to avoid unnecessary impacts on users' general computing resources.

2. Installing the BlueSky Framework

The BSF runs on a Linux computing platform with modest processing, memory, and disk resources. Software distributions of the BSF are developed for a limited set of platforms and operating systems. This section describes the BSF computing requirements, along with information on how to acquire and install the BSF software distribution.

2.1 Before Starting

Prior to acquiring and installing the BSF, prospective users should ensure they have the appropriate computing resources. Users must also read, understand, and agree to the BSF software terms and conditions (see Section 2.1.3).

2.1.1 System Requirements

The BSF supports the Ubuntu 12.04.2 LTS x86-62 (64-bit) and x86 (32-bit) operating systems. Both operating systems are built using glibc 2.15. Additional system requirements include

- Processor: Pentium 4 class or better
- Memory: 1GB minimum (4GB minimum for dispersion simulations with multiple fires)
- Disk space: 1.5GB for installation, plus 4GB minimum scratch space
- Internet connection for downloading fire information from SmartFire (optional)

The BSF may run on other POSIX-compliant operating systems, but has been tested only on the operating systems listed above. The binary distribution of the BSF contains all of the required libraries and packages.

2.1.2 Suggested System Configuration

Installation and use of the BSF does not require administrative privileges or special user accounts. The BSF can be installed anywhere in the file system with sufficient disk space and where the user has read and write privileges. To promote data sharing, ask a system administrator to establish a shared user group and use the **umask** configuration to control the permissions of new files and directories.

The recent addition of advanced output graphing capabilities means that two standard Python libraries must first be installed before the BSF can be run. These dependencies will be included in the BSF in the near future; meanwhile, the install commands are:

```
sudo apt-get install python-imaging
sudo apt-get install python-matplotlib
```

The BSF distribution does not include one optional package dependency, Graphviz, which BSF uses to draw dependency graphs when the -G command line option is invoked. On an Ubuntu system, use the following command to install the Graphviz package:

```
sudo apt-get install graphviz
```

Note that administrative privileges are required to install Graphviz. The BSF will work without Graphviz, but will display an error message when the –G command line option is invoked.

2.1.3 Terms and Conditions

The BSF is distributed under a GNU general public license (GPL) version 3, which places some requirements on the user, particularly if the user wants to modify the BSF. A full description of the license agreement appears in the main BSF folder ($BLUESKY_DIR/) in the file *LICENSE.TXT*. This file references the GPL, which is available in the main BSF folder (see *GPL.TXT*) or on the GNU website: http://www.gnu.org/licenses/gpl.html. Acquiring and installing the BSF requires the user to agree to the license agreement.

In part, the user of the BSF agrees to the following limitations:

1. The user may not redistribute the BSF without the permission of the copyright holders.
2. Derivative works based in whole or in part on the BSF code must be provided to the BSF development team at no cost.
3. The user is prohibited from misrepresenting the origin of the BSF software.

2.2 Downloading the BSF

The BSF is maintained in a source repository and is generally distributed as a collection of both compiled and uncompiled scripts (see Section 1.2 for more information on the structure of the BSF). Full instructions on where to get the BSF software distribution are located at http://airfire.org/bluesky.

2.3 Installing the BSF

The binary BSF distribution is provided as a compressed (“gzipped”) tarball (*bluesky-XXX.tar.gz*) that can be extracted and installed using the following command:

```
tar –xvzf filename.tar.gz
```

If installation is successful, a directory named **bluesky** appears.

2.4 Verifying the Installation

After installation, the resulting directory (located at $BLUESKY_DIR/) should contain the sub-directories listed in **Table 2-1**.

Table 2-1. BSF subdirectories and their contents.

Directory Name	Contents
base	base modules, scripts, library and data files
cache	data that BSF creates to speed up future simulations that use the same data sets
input	BSF-ready fire and meteorology input files
log	BSF system log files
modules	BSF modules
output	output files
setup	configuration files used to select modules
working	intermediate files created by the BSF science modules

A shell script named **bluesky**, which can be executed to invoke the BSF, also appears. The actual **bluesky** executable resides in $BLUESKY_DIR/base/lib/. The BSF license agreement and the software acknowledgments are in the main BSF directory. The $BLUESKY_DIR/working/ and $BLUESKY_DIR/log/ directories store information on the BSF runs. The fire science modules are in the $BLUESKY_DIR/modules/ directory and additional lower-level BSF code in the $BLUESKY_DIR/base/ directory.

3. Testing the BlueSky Framework

The following test cases will help the user verify that the installation of the BSF works correctly, and become familiar with basic BSF operation. To carry out these tests, the user will need to navigate to their "bluesky" directory ($BLUESKY_DIR/) and execute some simple command line statements.

3.1 What Version of the BlueSky Framework Is Installed?

Determining the version of the BSF is the most basic test of a BSF installation. To see which version of the BSF is installed, the user can execute the following statement:

```
$BLUESKY_DIR/bluesky -V
```

The BSF version number should appear on screen as shown below.

```
BlueSky Framework version 3.3.0 (rev 23321)
To see versions of individual modules/models, use "bluesky -p"
To see copyright information, use "bluesky --copyright"
```

If this test fails, the BSF distribution may not be appropriate for the user's system architecture, or the distribution may have been corrupted. If the result of this command is "Permission denied," the user should ensure the bluesky run script has the appropriate execute permissions.

3.2 What Science Modules Are Available in this Distribution?

A BSF user may want to verify what modules are accessible in the BSF installation. To do so, the user can execute the following command:

```
./bluesky -n
```

This test generates a list of BSF science modules, similar to Table 1-1. It also gives a local path and version number of each Python module. Users can add the -v flag to view a more exhaustive listing of BSF modules, including important non-science modules.

3.3 Simple Test Case

To create a BSF simulation, the user must develop a configuration file (also known as an INI file) by either (1) directly using a default INI file in the $BLUESKY_DIR/setup folder, (2) modifying a default INI file in the $BLUESKY_DIR/setup folder, or (3) developing a new INI file. Additional information on using INI files can be found in subsequent sections of this documentation.

A more extensive test of a BSF installation is to calculate the emissions for all the fires detected or reported in the United States on a single day. To do this, execute one of the following equivalent commands:

```
./bluesky setup/emissionsOnly.ini
```

```
./bluesky emissionsOnly
```

These commands both run the BSF using the *emissionsOnly.ini* configuration file. If the user does not explicitly declare the path of the INI file (as the first of these two commands does), the INI file must reside in the $BLUESKY_DIR/setup/ folder. This default emissionsOnly configuration is provided with the BSF distribution as an example of how to run the standard BSF modeling pathway, which obtains fire information from SmartFire and estimates fire emissions.

When the BSF is invoked with the emissionsOnly configuration, it prompts the user for the SmartFire user name and password (for more information on SmartFire, see Section A.3). The user will need to have these credentials to run this example. After running this command, various statements should print to the screen for up to a minute, and then a new folder should appear inside the BSF output directory. That folder will have a name starting with the date of the model run; for example, a run performed on February 14, 2012, would create a folder called 2012021400.1. BSF will not overwrite existing output, so it saves each run in a different folder, increasing the ".1" extension incrementally. For example, the folder created by the second run performed on the same date would be 2012021400.2.

Each output folder should have the following contents:

```
archive-2012021400.tar.gz
fire_emissions.csv
fire_events.csv
fire_locations.csv
pthour-2012021400.ems95
ptinv-2012021400.ida
summary.txt
```

The three CSV files are a serialized form of the Fire Information object used by the BSF to represent fire data (see Section A.1.1). These three files are

- **fire_locations.csv** – contains daily outputs; each row provides data for one fire on one day, while each column is an output parameter.
- **fire_emissions.csv** – contains hourly emissions outputs; each row provides data for one fire at a single hour, while each column is a pollutant.
- **fire_events.csv** – contains daily outputs; each row provides data for one fire *event* on one day, while each column is an output parameter.

The EMS95 and IDA files in the output folder are inputs for the Sparse Matrix Operator Kernel Emissions (SMOKE) model, which is not included in the BSF. The summary file is a record of modules invoked during the BSF run. Lastly, the archive file is a compressed archive of intermediate files that were generated by modules during the BSF run. For instance, if the user runs a module that in turn runs an executable with text input and output files, those files will be saved into this archive.

This archive file also contains two important diagnostic files. The first, *run.log*, provides a record of all information, debugging, warning, and error messages reported by the BSF, its modules, or the science models themselves. The second file, *run.ini*, contains all configuration parameters that the BSF used for the current run. These diagnostic files can be extracted from the archive as follows:

```
tar –xvzf archive-2012021400.tar.gz run.log
tar –xvzf archive-2012021400.tar.gz run.ini
```

Additionally, *run.ini* is a valid BSF INI file that can be used directly to invoke BSF again. This feature allows users to easily reproduce a BSF simulation using an identical set of configurations.

3.4 The Testing Module

3.4.1 The Goal

The purpose of the testing module is to run end-to-end tests of the BSF. That is, these tests run the BSF using the same system that a user would to run the BSF. And a user can execute any module chain from start to finish. In the end, these tests will compare the BSF outputs to output files saved within the test. If the files differ by more than 0.1%, then the tests fail.

3.4.2 Use Cases for Testing

There are three major use-cases for these tests:

1. Users can run these tests by hand, just as they would run the BSF normally, to test the system. The tests should run quickly and provide immediate validation that the user's copy of the BSF is working correctly.
2. Users can create their own tests quickly and easily to test a new module they are developing. They can also test to see whether they are changing the scientific output of an existing module.
3. These tests are run automatically during the BSF build process as an end-to-end test to make sure recent changes to the BSF codes have not changed the scientific results generated by the BSF.

3.4.3 Run a Test

The test cases present in the testing module can be run using a normal BSF configuration (INI) file, like any other BSF run. However, the run must be made using the '-t' flag on the command line, e.g.:

```
./bluesky -t testing/ex1_one_fire/ex1_one_fire.ini
```

At the end of the BSF run, the testing module will be run and print out (1) "ok" if the tests pass, or (2) "fail", followed by a stack trace, if the tests fail.

3.4.4 Create a New Test

Building a new test is very simple. First, create a new folder in the Testing directory $BLUESKY_DIR/bluesky/testing/my_test/ and then add two subfolders: /inputs/ and /expected_outputs/.

Next, add a configuration (INI) file:

```
$BLUESKY_DIR/testing/my_test/my_test.ini
```

Run the BSF as you normally would. Set up the INI file and the input files as you normally would. After the BSF has run, copy a select set of output files to your "expected_outputs" folder. Copy your fire locations input file and/or met data to the new "inputs" folder.

Finally, edit the INI file you used to run the BSF in the following ways:

1. Add a new output node to the end of your output nodes list:

```
outputs=StandardFiles KML Testing
```

2. Reassign your input and met areas to your new testing directory:

```
TEST_DIR = ${BS_DIR}/testing/my_test/
INPUT_DIR = ${TEST_DIR}input
MET_DIR = ${TEST_DIR}input
```

3. Finally, add a list of output files you want to compare with your expected outputs:

```
[OutputTesting]
FILES_TO_TEST = fire_locations.csv
```

3.5 What Command Line Options Are Available?

To see all configurations that currently reside in the $BLUESKY_DIR/setup/ folder, invoke the following command:

```
./bluesky -l
```

To see a complete listing of available command line flags in the BSF, execute the following command:

```
./bluesky –h
```

Users can also display additional BSF kernel options.

```
./bluesky –Khelp
```

4. Configuring the BlueSky Framework

There are several layers of configuration (INI) files in the BSF. The structure and content of these INI files is discussed in this section. During runtime, BSF collates the settings from all these INI files into one master configuration *run.ini* file. This master configuration file is saved to the working and output directories for later reference.

4.1 Configuration Files

There are three kinds of configuration files in the BSF, and they are found in different locations within the BSF.

4.1.1 Setup INI Files

Located in $BLUESKY_DIR/setup/, the setup INI files are the configuration files most commonly modified by the user. Each BSF execution requires one properly configured INI file. This is the file provided to the BSF as a command line input, as demonstrated in Section 3. Several INI files are provided as examples to demonstrate how to build custom configurations. The setup INI file defines the date of the run, what inputs and outputs are to be used, which fire science modules will be used, and what settings are to be applied. Importantly, the setup configuration file has precedence in the BSF, meaning that the parameters defined in this INI file will override any default settings defined elsewhere in the BSF. Sections 4.2 and 4.3 explain this INI file in more detail.

4.1.2 Module INI Files

Each module in the BSF includes one INI file that defines the default settings for that module. Located at $BLUESKY_DIR/modules/MODULE_NAME/v#/*module_name.ini*, these INI files contain important scientific constants, flags that define different operational modes of the module, and the location of module-specific input files. Although users can modify these module INI files, we recommend that users instead override these settings through their setup INI configuration file, following the examples provided in subsequent sections.

4.1.3 Base INI Files

Base INI files, located in $BLUESKY_DIR/base/etc/, are rarely touched by the user. The *types.ini* file defines the structure of the Fire Information object and other data structures used by the BSF (see Section A.1.1), while the *defaults.ini* file defines the standard module chain and paths within the BSF, along with other system settings.

4.1.4 Command Line Configuration

Any configuration parameter can be set from the command line. Any parameter set from the command line has precedence over all INI files. To set such a parameter, the *–D* flag is used on the command line.

```
./bluesky -D inputs=StandardFiles
```

4.2 Configuration File Structure

A BSF setup INI file consists of bracketed section headings, followed by the settings applying to that section. All BSF setup INI files must contain the [META] and [DEFAULT] sections, which contain the most general information defining the simulation resources and module chain. All other section headings refer to specific BSF modules. Settings for a particular module can be specified by adding the bracketed module name (from Table 1-1) followed by the settings for that module. Such additions should be made below all other lines in the setup INI file. The example INI section below illustrates how to define sampling grid parameters for a HYSPLIT dispersion simulation. INI sections can be in any order.

```
[HYSPLITDispersion]
USER_DEFINED_GRID = true
CENTER_LATITUDE = 55.0
CENTER_LONGITUDE = -125.0
WIDTH_LONGITUDE = 40.0
HEIGHT_LATITUDE = 20.0
SPACING_LONGITUDE = 0.25
SPACING_LATITUDE = 0.25
```

4.2.1 [META] Section

Several important parameters are set in the [META] section of the INI files. For instance, the file $BLUESKY_DIR/setup/*emissionsOnly.ini* provides the following five configuration statements:

```
[META]
includes=${GRAPH_DIR}/default.graph
inputs=StandardFiles SMARTFIRE
start=FillDefaultData
targets=$PLUME_RISE
outputs=StandardFiles SMOKEReadyFiles KML
```

A description of these five configuration statements follows:

1. **includes**: Sets the graph used by the BSF. The graph is a plain text file that defines the sequence of processes that will be run, and how the processes interconnect (see Section A.1.2 for more details). This setting, and the contents of the *default.graph* file, will rarely be changed.
2. **inputs**: Defines the input data. Users can explicitly provide all of the necessary information for all relevant fires in a CSV file (*StandardFiles*), download all fires in the United States for a specific time period from SmartFire, or both. If a fire is defined in both, the source listed first will take precedence. Dispersion runs require an additional meteorological data input, in one of the standard formats: Fifth-Generation Penn

State/NCAR Mesoscale Model (MM5), Weather Research and Forecasting (WRF), or Air Resources Laboratory (ARL). The available inputs are easily determined from the command line using the following command:

```
./bluesky -i
```

3. **start**: Defines which process BSF should start with. Any process can be listed here. For non-dispersion runs, this parameter will usually be FillDefaultData. However, if the user already has fuel loading and consumption data for their fires, this parameter can be changed to $EMISSIONS to bypass the unnecessary modeling steps. For a dispersion run, change the parameter to $EXTRACT_LOCAL_MET, and set EXTRACT_LOCAL_MET to a process that accepts meteorological data inputs, such as MM5LocalMet, ARLLocalMet, or WRFLocalMet.
4. **targets**: Defines the final process executed by BSF. Again, any process can be listed here. For instance, if the user is only interested in the total emissions, $PLUME_RISE can be replaced here with $EMISSIONS. For a dispersion run, set this parameter to $DISPERSION, and set the DISPERSION parameter to a valid dispersion modeling process.
5. **outputs**: Defines the output file types that BSF will produce. In most cases, the user will want to keep *StandardFiles* in this space-separated list. However, if the user is running a dispersion model, this field may be more complex (see Section B.7 for a discussion of the HYSPLIT dispersion module). The KML output node creates a KML file of all the fire information, viewable in Google Earth. If dispersion was run, this file will include maps of $PM_{2.5}$ emissions. The available outputs are easily determined from the command line using the following command:

```
./bluesky -o
```

4.2.2 [DEFAULT] Section

The [DEFAULT] section defines the actual fire science modules to be invoked, along with other run-specific options. The example below shows the portion of the INI file that pertains to defining models.

```
[DEFAULT]
# Models
EXTRACT_LOCAL_MET=ARL
FUEL_LOAD=FCCS
WILDFIRE_CONSUMPTION=CONSUME
PRESCRIBED_CONSUMPTION=CONSUME
OTHER_CONSUMPTION=CONSUME
TIME_PROFILE=FEPSTimeProfile
EMISSIONS=FEPSEmissions
```

```
PLUME_RISE=WRAPPlumeRise
WILDFIRE_GROWTH=Persistence
PRESCRIBED_GROWTH=NoGrowth
OTHER_GROWTH=NoGrowth
DISPERSION_MET=NoDispersionMet
DISPERSION=HYSPLITDispersion
```

The options here correspond to the fire science processes detailed in Table 1-1**.** Please refer to Section A.2 for more detailed information about the parameters in the [DEFAULTS] section of the INI file.

4.3 Configuration Examples

This section illustrates a set of example changes a user might make to the various INI files defined in the previous section. The INI files are plain text and can be edited using any text editor.

4.3.1 Changing the Date and Length of a Run

To change the date and length of a run, the user can open /setup/*emissionsOnly.ini*, for example, and look for the start of the [DEFAULT] settings.

```
[DEFAULT]
# Time range
DATE = 2008050100Z
HOURS_TO_RUN = 24
```

The code above indicates that the BSF will simulate 24 hours of fires, starting on May 1, 2008. To change the run to cover 72 hours, starting on December 31, 2012, change these lines to

```
[DEFAULT]
# Time range
DATE = 2012123100Z
HOURS_TO_RUN = 72
```

As a test, the user can change the **DATE** and **HOURS_TO_RUN** parameters above and run the BSF using the updated configuration file.

```
./bluesky setup/emissionsOnly.ini
```

Alternatively, the **DATE** and **HOURS_TO_RUN** parameters can be specified directly through the command line, which overrides the settings in the INI file. The following command line accomplishes this, and uses an assumed path for the setup file:

```
./bluesky –d 2012123100 –H 72 emissionsOnly
```

This approach to defining the simulation date and run length is convenient when attempting to batch BSF runs for many different dates.

4.3.2 Fire Data Sources

The BSF can be set either to download and use fire data via SmartFire or to use fire data supplied by the user (also in the SmartFire format). To run SmartFire, change the **inputs** parameter in the [META] section to

```
inputs=SMARTFIRE
```

If the user can provide fire data, the inputs parameter should change to

```
inputs=StandardFiles
```

If the user doesn't have any fire data to test this case, he or she can create the following file:

```
$BLUESKY_DIR/inputs/fires/fire_locations.csv
```

The user can then add the following text to that file:

```
id,event_id,latitude,longitude,type,area,date_time
SF11,11,37.99550002,-90.27600007,,218.9858793,201202140000-06:00
SF12,12,38.36800002,-95.36850004,,226.2668506,201202140000-06:00
```

This will create a simple test case with two fires in the United States on February 14, 2012. The data fields depicted above represent the minimum required fields for a fire information input file (*fire_locations.csv*). Please refer to Section 4.3.1 for a test case that shows how to match the date in this *fire_locations.csv* file with the date in the setup configuration file. The BSF will check to see if any of the fires in the *fire_location.csv* file match the date in the setup configuration file. If there are no matching dates, the BSF will fail and display this message:

```
ERROR: There are currently zero fire locations to burn; stop.
```

As a test exercise, new BSF users may try running the test case described above.

4.3.3 Choosing a Different Fuel Loading Module

BSF users are free to change the modules they use for each scientific process. The following example shows how to change the Fuel Loading module.

To determine which Fuel Loading modules are available in the BSF distribution, the user can type

```
./bluesky –n
```

This should produce a set of information similar to that described in Table 1-1, including a list of available Fuel Loading modules.

```
FuelLoading (base/modules/fuel_loading.py) v3.3.0
  FCCS (modules/FCCS/v2/fccs.py) v2
  FINNFuelLoading (modules/FINN/v1/finn.py) v1
  Hardy (modules/Hardy/v1/hardy.py) v1
  NFDRS (modules/NFDRS/v1/nfdrs.py) v1
  NoFuelLoading (base/modules/noop.py) v3.3.0
```

As a test, a new user can change the Fuel Loading module in the setup configuration file to something else on this list: for example, NFDRS. To do so, the user should find the following line in the setup configuration file:

```
FUEL_LOAD=FCCS
```

And change it to invoke the NFDRS as follows:

```
FUEL_LOAD=NFDRS
```

The user can then run the BSF, using the updated configuration file.

Optionally, if a user has input data that includes fuel loading information, he or she can choose to set the above parameter to “NoFuelLoading” and the BSF will skip that process. There are similar “no-op” modules for each science process in the BSF.

4.3.4 Changing a Default Scientific Parameter in Consume

The preceding discussion dealt only with changes to the setup configuration files. However, each module within the BSF also has its own configuration settings. These settings define the locations of various input files, different modes the modules might operate in, and the values of scientific constants.

This example focuses on changing a scientific constant within the Consume fuel consumption module. When this module was written, the scientists involved felt it useful to expose a scientific parameter to the user, so that it could be easily changed without re-compiling the source code.

In the Consume module, an exposed parameter defines the percentage of shrub fuels that are burned during a fire. The default value is 50 percent, but when a BSF user has more specific or detailed information, this default value may be changed. To do so, the user can open the following Consume configuration file in a text editor:

```
$BLUESKY_DIR/modules/CONSUME/v5/consume.ini
```

The following line defines the percent of shrubs burned during a fire:

```
SHRUB_PERCENT_BLACKENED = 50.0
```

That parameter can be changed something else; this example indicates that 99.9% of shrubs were burned during the fire in question.

```
SHRUB_PERCENT_BLACKENED = 99.9
```

However, making a change in this manner means that this new default shrub value will be applied to every BSF run in the future, or that the user will have to change this file before every simulation. Setting this parameter in the setup configuration file ($BLUESKY_DIR/setup/*emissionsOnly.ini*) is a better approach.

```
[CONSUME]
SHRUB_PERCENT_BLACKENED = 50.0
```

The setup configuration file has precedence, meaning that any parameter setting there will override the same parameter in the Consume configuration file. In this way, the user can make a run-specific change to the model and not have to worry about resetting that change before the next simulation.

5. Modifying the BlueSky Framework

Users can develop a custom module for the BSF. The modules provided with the BSF core distribution are good examples of how BSF modules are constructed, how to manipulate data in the BSF, and how to use BSF functionality to run fire science models. The following sections provide simplified examples to demonstrate how to make some basic changes to an existing BSF module, and how to add new modules to the BSF. A basic understanding of the Python programming language is needed to develop a custom BSF module. All Python code in the BSF must conform to the PEP8[1] standards.

5.1 Changing an Existing Module

A BSF module consists of a fire science model wrapped by Python code designed to help that model run within the BSF. The model itself may be written in Python, or in another programming language. The following example demonstrates where to find modules, how to edit model wrappers, and how to make use of the module configuration files.

In this example, the user adds a default value for the FCCS fuelbed number for each fire in the FCCS module. On line 129 of the FCCS model wrapper ($BLUESKY_DIR/modules/FCCS/v2/*fccs.py*), an FCCS number is passed into the fuelbed object.

```
fuelInfo["metadata"]["fccs_number"] = fccsNumber
```

To ensure that a specific default FCCS number is always passed into the Fire Information object, the following line can be added:

```
if not fccsNumber: fccsNumber = 0
fuelInfo["metadata"]["fccs_number"] = fccsNumber
```

All changes to BSF Python modules take effect immediately. The next time the BSF executable is run, the FCCS module will be automatically imported and re-evaluated, and an updated bytecode (*.pyc) file will be created on the fly.

In this case, users who want to change this new default FCCS number would have to edit the module wrapper. Exposing this new default value through the module-level configuration file is a better programming paradigm. To do this, add the following line at the end of the FCCS module configuration file ($BLUESKY_DIR/modules/FCCS/v2/*fccs.ini*):

```
DEFAULT_FCCS_NUMBER = 0
```

To access this new configuration variable, edit the FCCS model wrapper to include the following:

[1] http://www.python.org/dev/peps/pep-0008/

```
FCCS_NUM = self.config("DEFAULT_FCCS_NUMBER")
if not fccsNumber: fccsNumber = FCCS_NUM
fuelInfo["metadata"]["fccs_number"] = fccsNumber
```

The **self.config** function accesses the new INI configuration. Users can now set this default FCCS fuelbed number through the configuration file, without editing the module itself.

5.2 Creating a New Module Using Pure Python

One feature that makes the BSF so flexible is the ability to easily add a new module. While the BSF kernel itself is wrapped up in a compiled executable, everything a user needs to add a new module is exposed in the /modules/ and /setup/ folders within the BSF distribution.

In the following example, the user adds an extremely simple Fuel Loading module to the BSF. Two things should be noted here. First, this module is purely for example and is not scientifically valid. Second, this module is pure Python. That is, no external executable will be run in this module; the fuel loading model will be coded directly into the Python module. Section 5.3 covers the case where a module wraps an external executable.

The first step is to create a new folder for the module using the following command:

```
mkdir $BLUESKY_DIR/modules/examplefuelloading/v1/ -p
```

Next, copy the following code into a file named *examplefuelloading.py* and save it in the new folder.

```
# all modules must declare a BSF version
_bluesky_version_ = "3.3.0"

from fuel_loading import FuelLoading
from kernel.types import construct_type

class ExampleFuelLoading(FuelLoading):
    """ This class extends FuelLoading, which in turn extends Process.
    This allows the BSF to place this module correctly in the graph
    or module chain.
    """

    # all modules must implement this method
    def run(self, context):
        # Get fire information from the last module in the module chain
        fireInfo = self.get_input("fires")
```

```
        # loop over all fires in the simulation
        for fireLoc in fireInfo.locations():
            # construct a new object of FuelsData type, this object
            # is part of the master Fire Information object
            fuelInfo = construct_type("FuelsData")

            # fill fuels data for each fire with dummy values
            fuelInfo["fuel_1hr"] = float(1.0)
            fuelInfo["fuel_10hr"] = float(1.0)
            fuelInfo["fuel_100hr"] = float(1.0)
            fuelInfo["fuel_1khr"] = float(1.0)
            fuelInfo["fuel_10khr"] = float(1.0)
            fuelInfo["fuel_gt10khr"] = float(1.0)
            fuelInfo["shrub"] = float(1.0)
            fuelInfo["grass"] = float(1.0)
            fuelInfo["canopy"] = float(1.0)
            fuelInfo["rot"] = float(1.0)
            fuelInfo["duff"] = float(1.0)
            fuelInfo["litter"] = float(1.0)

            # place these new fuel loadings in the fire object
            fireLoc["fuels"] = fuelInfo

        # Set output plug to updated fireInfo
        self.set_output("fires", fireInfo)
```

Also, create a configuration file for the module. Name the file *examplefuelloading.ini*, and paste the following code into the file:

```
[ExampleFuelLoading]
DUMMY=0
```

Finally, every BSF module needs a version number file. Name the file *version.inc*, and paste the following code into the file:

```
MODVERSION=1
```

The next time the BSF is executed, it will automatically recognize the presence of the new module. To test that the new module was correctly placed in the system, run the BSF executable with the nodes flag.

```
./bluesky –n
```

The new module should appear under the FuelLoading heading. Following the instructions in Section 4.3.3, change the fuel loading module chosen in the setup configuration file. The name here must exactly match the name of the module, as defined by how the Python class is named in the code, and not by the file name.

```
FUEL_LOAD=ExampleFuelLoading
```

The BSF will now run with the new fuel loading module. When the BSF first runs, a bytecode file, *examplefuelloading.pyc*, will be created in the new module folder.

It is instructive to look at the new code in *examplefuelloading.py*. The first thing to notice is that the class ExampleFuelLoading extends the FuelLoading class, which is imported from *fuel_loading*. This class, which can be found in $BLUESKY_DIR/base/modules/*fuel_loading.py*, contains the boilerplate code that establishes the data inputs and outputs for the new module. The BSF has similar classes in $BLUESKY_DIR/base/modules/ for each fire science process to promote code reuse.

Each module must define two methods, **init** and **run**. For this example module, the **init** method was defined in the FuelLoading class that ExampleFuelLoading extends. The run method above is called when the BSF invokes a module, so any actions the module takes must start there. The **run** method alters the data that is passed into and out of a module. For instance, a fuel loading module takes in a Fire Information object, adds fuel information to it, and then passes the Fire Information object back out, to be used in a consumption module. Inside a module, the developer can use the **get_input** method to retrieve the input nodes to a module and the **set_output** method to send information along to the next module in the simulation.

The body of the above example module loops through all of the fires in the Fire Information object, using the Fire Information class iterable **.locations()** method. This syntax is a common pattern in all BSF modules, and allows the developer to treat each fire location separately. The science within a module is typically written (or called) inside this fire loop. In this hypothetical example, various elements of the fuels data object are set within the loop.

The last important thing to notice in the above example module is the **construct_type** call. With this method, the developer can create, on the fly, any data object that might be included in the Fire Information data structure described in Section A.1.1.

5.3 Creating a New Module Using a Non-BSF Executable

The previous example built a new module in pure Python. However, a BSF module can instead be based on a preexisting and entirely independent executable. The Fire Emission Production Simulator Emissions model is a good example.

To run an external executable in a BSF module, use the **context.execute()** method.

```
context.execute(PATH_TO_EXECUTABLE,
                "-f", inputFile,
                "-o", outputFile)
outputData = self.readOutputData(outputFile)
```

This pattern allows the developer to arbitrarily run any executable (that runs correctly on the local system architecture), using relative BSF paths. The best practice is to define the paths to any executables or input/out files in the *examplefuelloading.ini* file.

5.4 A High-Level View of Creating a New Node

A node in the BSF is a generic step in the modeling chain. The consumption node is one example, and includes the Consume and FEPS Consumption modules.

While a complete set of nodes is available in the BSF for standard fire science calculations, a developer may wish to add a new node to support a new scientific approach. Fortunately, the BSF developer can alter the functionality of the BSF. Everything necessary to create a new node is easily accessible in the $BLUESKY_DIR/base/ directory.

To create a new node, a developer will need to create a new subclass of Process in $BLUESKY_DIR/base/modules/—much like several of the subclasses already present there (*consumption.py*, *emissions.py*, *plume_rise.py*, etc.). This is an example of a new node module:

```
# place at: $BLUESKY_DIR/base/modules/emissions_preprocessor.py

_bluesky_version_ = "3.3.0"

from kernel.core import Process

class EmissionsPreprocessor(Process):
    def init(self):
        self.declare_input("fires", "FireInformation")
        self.declare_output("fires", "FireInformation")
```

Before a developer can start using the new node in a module, he or she must create a new graph that includes that module (see Section A.1.2 for a discussion of graphs). In the simple case above, it would be best just to copy the default module as follows:

```
cp setup/graphs/default.graph setup/graphs/my_first.graph
```

Then, the developer may edit these few lines (starting on line 78 in the copied file):

```
[Process($TIME_PROFILE)]
input(fires) = connect(ConsolidateConsumption.fires)

[Process($EMISSIONS_PREPROCESSOR)]
input(fires) = connect($TIME_PROFILE.fires)

[Process($EMISSIONS)]
input(fires) = connect($EMISSIONS_PREPROCESSOR.fires)
```

To create a new module that uses the new node (as in Section 5.2) the developer would need to create a new module that extends the new node.

```
class NewEmissionsPreprocessor(EmissionsPreprocessor):
```

The **run** method will have to be implemented in this class. Once the new module is built, the developer will need to include the module in a setup configuration file, as described in Section 4.2.1. The new graph will also have to be included in the new setup configuration file.

```
[META]
includes=${GRAPH_DIR}/my_first.graph
```

6. For More Information

A user with more questions regarding the BSF can refer to the following resources (current as of May 2013):

- Published literature
  - Larkin N.K., O'Neill S.M., Solomon R., Raffuse S., Strand T.M., Sullivan D.C., Krull C., Rorig M., Peterson J., and Ferguson S.A. (2009) The BlueSky smoke modeling framework. *Int. J. Wildland Fire*, **18** (8), 906-920 (doi:10.1071/WF07086).
  - Strand T.M., Larkin N., Craig K.J., Raffuse S., Sullivan D., Solomon R., Rorig M., Wheeler N., and Pryden D. (2012) Analysis of BlueSky Gateway $PM_{2.5}$ predictions during the 2007 southern and 2008 northern California fires. *J. Geophys. Res,* 117(D17301), (doi:10.1029/2012JD017627).
  - Raffuse S., Craig K., Larkin N., Strand T., Sullivan D., Wheeler N., and Solomon R. (2012) An evaluation of modeled plume injection height with satellite-derived observed plume height. *Atmosphere,* 3, special issue: Biomass Emissions, Dr. Charles Ichoku, ed., 103-123 (STI-908054-3870, doi: 10.3390/atmos3010103).
  - O'Neill S.M., Larkin N.K., Hoadley J., Mills G., Vaughan J.K., Draxler R.R., Rolph G., Ruminski M., Ferguson S.A., (2009) Regional real-time smoke prediction systems. In: S. V. Krupa, editor: Developments in Environmental Science, Vol 8, Wild Land Fires and Air Pollution, A. Bytnerowicz, M.J. Arbaugh, A.R. Riebau and C. Andersen. The Netherlands: Elsevier, 2009, pp. 499–534.
  - O'Neill S.M., Hoadley J.L, Ferguson S.A., Solomon R., Peterson J., Larkin N.K., Peterson R., Wilson R., Matheny D. (2005) Applications of the BlueSkyRAINS smoke modeling system. Journal of the Air and Waste Management Association, Sept 2005, 20-23.

- Website
  - http://airfire.org/bluesky

- AirFire contact information
  - Sim Larkin, larkin@fs.fed.us, 206-732-7828

Appendix A: BlueSky Framework Information

A.1. Important BlueSky Framework Concepts

This appendix discusses some of the concepts used to organize the ideas and the source code in the BSF. This appendix is meant as reference, particularly for those who wish to develop in the BSF.

- **The Fire Information Data Structure**

An understanding of the Fire Information object is crucial to developing a general understanding of the BSF. The Fire Information object contains all the information about a fire. Each module in a module chain deals directly with the Fire Information object; the module receives the Fire Information object, adds data to it, and passes it on to the next module in the module chain.

The BSF was designed so that the exact nature of this data structure would be explicitly viewable and easily alterable via a text file editor. The Fire Information data structure is stored in

```
/path/to/bluesky/base/etc/types.ini
```

The data in this text file are used to create nested objects during run time. The most important object defined herein is the Fire Information object.

```
[FireInformation:base.modules.fire_information.FireInformation]
fire_locations = FireLocationData[]
fire_events = FireEventData[]
dispersion = DispersionData
start_date = BSDateTime, time
hours_to_run = int, hours
emissions_offset = int, hours
dispersion_offset = int, hours
emissions_start = BSDateTime, time
emissions_end = BSDateTime, time
dispersion_start = BSDateTime, time
dispersion_end = BSDateTime, time
metadata = dict
```

Notice that the Fire Information object contains a list of FireLocationData objects, which are also defined in this same text file using the following code:

```
[FireLocationData:base.modules.fire_information.FireLocationData]
id = str
```

```
owner = str
latitude = float
longitude = float
fips = str
elevation = float, meters
…
fuels = FuelsData
consumption = ConsumptionData
emissions = EmissionsData
plume_rise = PlumeRise
metadata = dict
```

Each attribute (of some object) defined in the *types.ini* file can be defined in one of three ways:

1. As another object defined in the types.ini file: **fuels = FuelsData**
2. As a primitive Python type: **metadata = dict**
3. As a scientific variable (with a Python type and a physical unit): **elevation = float, meters**

The developer may want to refer to the *types.ini* file for two reasons. First, the developer may need to read and understand an existing module in the BSF to understand the data structures that are in use; these data structures are defined in *types.ini*. Second, if the developer wants to build a new module in the BSF, information can be added to the *types.ini* file and the data structure that the BSF passes around can be changed.

- **The Module Graph**

The BSF organizes a series of scientific models such that they can be strung together and the outputs of one can be used as the inputs of another. Each module in the BSF is defined with input and output "plugs." Section 5.4 explains that each module is designed with an **init** method that includes an input and an output. For example, the Consume module takes in a Fire Information object and returns an updated Fire Information object, which are the input and output plugs of the Consume module. The module chain is a sequence of modules that are linked together by the information they pass in their input/output (I/O) plugs.

The module graph is key to this organization. The graph defines the order of the processes and how their plugs are linked together. The default graph in the BSF is an example and can be found here:

```
$BLUESKY_DIR/setup/graphs/default.graph
```

The graph does not define what modules are used for each process; that information is defined in the setup configuration file. The following set of connections in the default graph illustrates how this information is defined.

```
[Process($TIME_PROFILE)]
input(fires) = connect(ConsolidateConsumption.fires)

[Process($EMISSIONS)]
input(fires) = connect($TIME_PROFILE.fires)
```

The first line of this code identifies the time profile process, and the second line connects the input of the time profile to the output of the consumption process. The third and fourth lines show that the input of the emissions process is connected to the output of the time profile process. Notice that the setup configuration file doesn't specify whether the FEPS time profile or the WRAP time profile will be used, just that a time profile module will be run after an emissions module.

A.1 Configuration

Tables A-1 and A-2 describe the core variables defined in a typical BSF INI file. To streamline configuration changes made by the user, some variables in the INI are shell substitutions (preceded by a dollar sign, $) that are defined elsewhere in the INI. Many of the INI parameters are used as substitutions in the default dependency graph. Note that the INI file for a specific BSF simulation may include additional settings that override some defaults.

- **META**

Table A-1. Input settings for the BSF setup INI files: [META] section.

Setting	Description	Options
includes	The dependency graph that describes how the modules interconnect.	${GRAPH_DIR}/default.graph, or user-defined dependency graph.
inputs	The input tasks that the BSF will run to read input data.	StandardFiles, SmartFire, MM5, ARL
start	The module that accepts input data.	FillDefaultData for emissions runs; $EXTRACT_LOCAL_MET for dispersion runs. All processes are valid here.
targets	The step(s) to be accomplished.	$EMISSIONS, $DISPERSION All processes are valid here.
outputs	The output tasks that the BSF will run once it reaches the stopping point (target). More than one output task can be specified.	StandardFiles SMOKEReadyFiles KMLAnimation MapImages

- **DEFAULT**

Table A-2. Input settings for the BSF setup INI files: [DEFAULT] section.

Setting	Description	Example Settings
DATE	Date for a BSF run.	2009040100Z
HOURS_TO_RUN	Number of hours to run.	24
MET	Meteorological model data source.	MM5 ARL
EXTRACT_LOCAL_MET	Module for processing meteorological inputs.	MM5LocalMet ARLLocalMet
FUEL_LOAD	Fuel loading process module.	FCCSFuelLoad
WILDFIRE_CONSUMPTION	Consumption process module for wildfires.	CONSUME
PRESCRIBED_CONSUMPTION	Consumption process module for prescribed fires.	CONSUME
OTHER_CONSUMPTION	Consumption process module for fires types other than wildfire and prescribed fire.	CONSUME
TIME_PROFILE	Time profile process module. This process temporalizes daily outputs from the consumption process, making daily data hourly/diurnal.	FEPSTimeProfile
EMISSIONS	Emissions process module.	FEPSEmissions
DISPERSION_MET	Meteorological data translation module that prepares data for dispersion modeling.	MM52ARL
DISPERSION	Dispersion process module.	HYSPLITDispersion
TRAJECTORY_MET	Meteorological data translation module that prepares data for trajectory modeling.	MM52ARL
TRAJECTORY	Trajectory process module.	HYSPLITDispersion
WILDFIRE_GROWTH	Fire growth model for wildfires.	Persistence NoGrowth
PRESCRIBED_GROWTH	Fire growth model for prescribed fires.	Persistence NoGrowth
OTHER_GROWTH	Fire growth model for other fire types.	Persistence NoGrowth
WILDFIRE_CANOPY_FRACTION	Consumption canopy fraction for wildfires.	A floating point value (e.g. 0.5) “auto”
PRESCRIBED_CANOPY_FRACTION	Consumption canopy fraction for prescribed fires.	A floating point value (e.g., 0.1) “auto”
OTHER_CANOPY_FRACTION	Consumption canopy fraction for fire types other than wildfire or prescribed fire.	A floating point value (e.g. 0.1) “auto”

Setting	Description	Example Settings
STOP_IF_NO_BURNS	The BSF stops if it finds no valid fires.	True or False
STOP_IF_NO_MET	The BSF stops if it finds no valid meteorological data.	True or False
SPIN_UP_EMISSIONS	The BSF spins up emissions prior to the requested run date.	True or False (true by default)
EMISSIONS_OFFSET	Number of hours to offset the emissions time period. Can be positive or negative.	-24 hours by default
DISPERSION_OFFSET	Number of hours to offset the dispersion time period. Can be positive or negative.	0 hours by default
MET_DIR	The directory in which the BSF expects to find meteorological data files (or symbolic links to those data).	input/met by default

A.2 SmartFire

- **About SmartFire**

The SmartFire fire information system is a framework separate from the BSF that aggregates, associates, and reconciles wildland fire information from disparate sources, including satellite, aircraft, and ground reporting. The current version of SmartFire is Version 2 (SF2), which includes significant advancements to data processing, associating, and reconciliation algorithms. SF2 can use any number of data sources, associating and reconciling their information to avoid double counting of fires; it selectively uses the best pieces of data from each source.

There is no single, complete, best fire information source. Fire occurrence data can come from operational reports from wildfire fighting operations, automated detection of "hot spots" by satellite-based instruments, infrared perimeters derived from helicopter overflights, prescribed burn permitting databases, high-resolution satellite vegetation scar analysis, and other sources. Each data source's coverage is limited by fire size, fire types, jurisdiction, cloud cover, or other features. Some data sets are available in near real time, while others are only available many years after the fire. Each data source has strengths and weaknesses, and information can be redundant among sources. SmartFire reconciles these sources into a unified data set for modeling and analysis.

SmartFire can produce fire information suitable as input for the BSF. The BSF can be set up to download data from a SmartFire instance.

- **References**
  - The SmartFire website, created by the AirFire team at USFS: http://www.airfire.org/smartfire/

- Raffuse S.M., Sullivan D.C., Chinkin L.R., Gilliland E.K., Larkin N.K., and Solomon R. (2008): Development and sensitivity analysis of wildland fire emission inventories for 2002-2006: *17th International Emissions Inventory Conference, Portland, OR, June 2-5*.
- Raffuse S.S., Pryden D.A., Sullivan D.C., Larkin N.K., Strand T., and Solomon R. (2009): SMARTFIRE Algorithm Description: White Paper, October 2.
- Sullivan D.C., Raffuse S.S., Pryden D.A., Craig K.J., Larkin N.K., Strand T., and Solomon R. (2009): Development and Applications of Systems for Modeling Emissions and Smoke from Fires: The BlueSky Smoke Modeling Framework and SMARTFIRE: White Paper.

Appendix B: Module Information

This appendix contains a brief introduction to the standard BSF modules. It also includes a description of the underlying scientific model, important configuration options in the BSF, and references for more information.

B.1 FCCS

- **About FCCS**

The user has the option of using either the FCCS v1 or FCCS v2 fuel loading module; FCCS v2 is the default. The FCCS fuel loading map provides fuel loading information at the 1-km scale for the continental United States (CONUS) (McKenzie et al., 2007). The FCCS v1 fuel mapping has a crosswalk between 112 fuelbeds to each 1-km grid cell based on an assignment methodology described in McKenzie et al. (2007). The updated FCCS v2 fuel mapping was created using similar methodologies, but with updated information and twice as many fuelbed definitions, as described in the FCCS User's Guide (Prichard, et al., 2011). The standard FCCS fuelbeds and every unique 1-km grid within CONUS were independently assigned a value for Bailey's ecosystem sections (Bailey 1996) and Kuchler's potential natural vegetation classification (Kuchler 1964). The Bailey's ecosystem sections are a vegetation cover type derived from AVHRR data by Schmidt et al. (2002). These classification values were then used to match fuelbed values with map values. The FCCS fuelbed concept is the most comprehensive of the fuel maps and includes downed woody fuels (1, 10, 100, 1000, and 10000+ hour), shrubs, herbs, grasses, canopy fuels, dead standing trees (snags), stumps, litter, moss, lichens, and duff.

- **How to Invoke FCCS**

In the setup configuration file (/setup/*example.ini*), the user will need to set the FUEL_LOAD parameter as follows:

```
FUEL_LOAD=FCCS
```

- **Configuration Options for FCCS**

In the FCCS configuration file (/modules/FCCS/v2/*fccs.ini*), one parameter defines whether the fuel mapping will come from FCCS v1 or FCCS v2.

```
FCCS_VERSION=2
```

The rest of the parameters in the FCCS configuration file define the input files that FCCS will use to look up fuel mapping values. If the user wishes to use different fuel mapping, the paths in *fccs.ini* need to be changed to direct FCCS to the new data. Remember, it is preferable to set parameters in the setup configuration file (see Section 4.3).

- **References**
  - FCCS v1 and FCCS v2 maps were downloaded from the FERA website on 4/23/2010 and 2/14/2013, respectively (these versions may no longer be available for download if a newer version has been released): http://www.fs.fed.us/pnw/fera/fccs/maps.shtml.
  - Bailey, R.G. Ecosystem geography. Springer-Verlag, Inc., New York.
  - Kuchler A.W. 1964. Potential natural vegetation of the coterminous United States. American Geographical Society (with separate map at 1 : 3 168 00). New York. Special Publication 36.
  - McKenzie D., Raymond C.L., Kellogg L.-K.B., Norheim R.A., Andreu A.G., Bayard A.C., Kopper K.E., and Elman E. (2007) Mapping fuels at multiple scales: landscape application of the Fuel Characteristic Classification System. *Canadian Journal of Forest Research* **37**:2421-2437.
  - Prichard S. J., Ottmar R. D., Sandberg D. V., Eagle P. C., Andreu A. G., Swedin K. (2011) *FCCS User's Guide*.
  - Schmidt KM, Menakis JP, Hardy CC, Hann WJ, Bunnell DL (2002) Development of coarse-scale spatial data for wildland fi re and fuel management. USDA Forest Service Rocky Mountain Research Station, Fort Collins, General Technical Report RMRS-GTR-87.

B.2 Consume

- **About Consume**

Consume v3 (Prichard et al., 2006) predicts fuel consumption by strata for each of the following fuels/vegetation: canopy, shrubs, herbaceous, downed woody fuels (1, 10, 100, 1000, 10,000+ hour fuels), litter, moss, lichen, and duff. Consume calculates consumption for duff, litter, and woody fuels using empirically derived algorithms. Simple rule of thumb equations are used to predict canopy, shrub, and herbaceous fuel consumption. Consume partitions fuel consumption into flaming and smoldering consumption using set proportions for each fuels strata (Prichard et al., 2006). Also, Consume has undergone significant development work in recent years, including conversion to a Python-based code various bug fixes. Consume can also compute emissions for pile burns, but that functionality is not in place in the BSF as of July 2013.

- **How to Invoke Consume**

The ***_CONSUMPTION** parameters are set in the setup configuration file (/setup/*example.ini*).

```
# Consumption models (by fire type)
WILDFIRE_CONSUMPTION=CONSUME
PRESCRIBED_CONSUMPTION=CONSUME
```

```
OTHER_CONSUMPTION=CONSUME
```

The three consumption parameters above do not all need to be set to the same module, or to Consume. Other consumption modules can be chosen.

- **Configuration Options for Consume**

The Consume configuration file (/modules/CONSUME/v5/*consume.ini*) is broken into five sections. The first section simply defines the paths to various input files. Section 2 is where all physical constants are kept. In particular, this is where the user will set the default value for the percent of shrubs that are burned during a fire.

```
SHRUB_PERCENT_BLACKENED = 50.0
```

Section 3 of the Consume configuration file defines the measurements that were used to generate the fuel loading inputs for Consume.

```
MEASUREMENT_TYPE = MEAS-Th
# MEASUREMENT_TYPE = ADJ-Th
# MEASUREMENT_TYPE = NFDRS-Th
```

Section 4 defines many parameters that are used as default fuel loading values. These values are not set by the FCCS fuel loading module but are required by Consume. Section 4 can and should be modified if the user is studying fires in an area where local fuel loadings are known. Section 5 deals with Live Fuel Moisture (LFM) inputs and methodologies in Consume. The LFM configurations are not part of the standard Consume logic and should be considered experimental science. Best practice is to change any of the above parameters in the setup configuration file (see Section 4.3).

- **References**

- Prichard, S.J., Ottmar, R.D., and Anderson G.K. (2006). Consume 3.9 User's Guide. Available for download at http://www.fs.fed.us/pnw/fera/research/smoke/consume/consume30_users_guide.pdf.
- http://www.fs.fed.us/pnw/fera/research/smoke/consume/consume_download.shtml.

B.3 FEPS Emissions

- **About FEPS Emissions**

The FEPS emissions model calculates speciated emissions from given consumption values. Emissions of CO_2, CO, CH_4, and $PM_{2.5}$ (in tons per fire) are based on emission factors derived from empirical relationships. The emission factors are calculated from combustion efficiency through a linear relationship derived from field observations. The combustion efficiency corresponds to the fire phase (smoldering or flaming) under which the emissions were released. The combustion efficiency is calculated through the consumption portion of FEPS

and is assumed to reflect the fuel type and moisture. The fuel information is only related to the emission factors algorithm through combustion efficiency. Emissions themselves are computed by multiplying the calculated emission factors (mass of emissions per mass of fuel consumed) by the quantity of fuels consumed.

- **How to Invoke FEPS Emissions**

In the setup configuration file (/setup/*example.ini*), the user will need to set the EMISSIONS parameter.

```
EMISSIONS=FEPSEmissions
```

- **Configuration Options for FEPS Emissions**

In the FEPS configuration file (/modules/FEPS/v2/*feps.ini*), one section for FEPS Emissions is available:

```
[FEPSEmissions]
FEPS_WEATHER_BINARY = ${PACKAGE_DIR}/feps_weather
FEPS_PLUMERISE_BINARY = ${PACKAGE_DIR}/feps_plumerise
FEPS_EMISSIONS_BINARY = ${PACKAGE_DIR}/feps_emissions
FEPS_OUTPUT_BINARY = ${PACKAGE_DIR}/feps_output
FEPS_EMIS_HAP = false
```

The first four parameters are simply paths to the four FEPS executables. The last parameter, which is recommended to be set to **false**, is an optional parameter that was designed separately from FEPS to allow the FEPS model to approximate several species of Hazardous Air Pollutants (HAPs), along with criteria pollutants. This capability is not part of the canonical BSF and should be considered experimental science.

- **References**

  - Anderson, G. K., Sandberg, D. V., and Norheim, R. (2004) Fire Emission Production Simulator user's guide, version 1.0. Produced for the Joint Fire Science Program (98-1-9-05).
  - FEPS User's Guide: http://www.fs.fed.us/pnw/fera/feps/FEPS_users_guide.pdf.
  - Fire Emission Production Simulator (FEPS), version 1.1; Fire and Environmental Research Applications Team: Seattle, WA, USA, 2005.

B.4 FEPS Plume Rise

- **About FEPS Plume Rise**

FEPS v2 calculates three plume rise quantities. It uses a modified version of the classical empirical plume rise equations developed by Briggs (1975) to calculate maximum

plume rise. FEPS also provides a second empirical approach for calculating both maximum and minimum plume rise. All FEPS plume rise calculations depend on hourly consumption inputs and generate hourly outputs. Not all emissions in FEPS are subject to plume rise. A fraction of the emissions are assigned no plume rise based on calculated entrainment efficiency. The default BSF implementation of FEPS used for this case uses the modified Briggs method to calculate plume top height and the FEPS empirical approach to calculate plume bottom. The emissions are then evenly distributed throughout the plume bottom and top.

- **How to Invoke FEPS Plume Rise**

In the setup configuration file (/setup/*example.ini*), the user will need to set the **PLUME_RISE** parameter.

```
PLUME_RISE=FEPSPlumeRise
```

- **Configuration Options for FEPS Plume Rise**

In the FEPS configuration file (/modules/FEPS/v2/*feps.ini*), one section for FEPS Plume Rise is available.

```
[FEPSPlumeRise]
FEPS_WEATHER_BINARY = ${PACKAGE_DIR}/feps_weather
FEPS_PLUMERISE_BINARY = ${PACKAGE_DIR}/feps_plumerise
PLUME_TOP_BEHAVIOR = auto
```

The first two parameters are simply paths to the relevant FEPS executables. The **PLUME_TOP_BEHAVIOR** parameter has three options for how the maximum plume height is calculated; it is set to **auto** by default.

- **References**

- Anderson, G. K., Sandberg, D. V., and Norheim, R. 2004. Fire emission production simulator User's Guide, version 1.0. Produced for the Joint Fire Science Program (98-1-9-05).
- Briggs, G.A. Plume Rise Equations. In *Lectures on Air Pollution and Environmental Impact Analysis*; Haugen, D.A., Ed.; AMS: Boston, MA, USA, 1975; pp. 59–111.
- FEPS User's Guide: http://www.fs.fed.us/pnw/fera/feps/FEPS_users_guide.pdf.
- Fire Emission Production Simulator (FEPS), version 1.1; Fire and Environmental Research Applications Team: Seattle, WA, USA, 2005.

B.5 FEPS Time Profile

- **About FEPS Time Profile**

Within the FEPS model (see also Section B.3), the time profile of a prescribed fire is described with two curves: one for the flaming phase of the fire and one for the smoldering phase. The flaming phase of combustion is modeled as directly proportional to area burned for a given hour. The smoldering phase of combustion is modeled to continue after fire growth with exponential decay.

- **How to Invoke FEPS Time Profile**

In the setup configuration file (/setup/*example.ini*), the user will need to set the TIME_PROFILE parameter.

```
TIME_PROFILE=FEPSTimeProfile
```

- **Configuration Options for FEPS Time Profile**

In the FEPS configuration file (/modules/FEPS/v2/*feps.ini*), a section for FEPS Time Profile is available.

```
[FEPSTimeProfile]
FEPS_WEATHER_BINARY = ${PACKAGE_DIR}/feps_weather
FEPS_TIMEPROFILE_BINARY = ${PACKAGE_DIR}/feps_timeprofile
INTERPOLATION_TYPE = 1
NORMALIZE = true
```

The first two parameters are simply the paths to two executables needed by the FEPS time profile. **INTERPOLATION_TYPE** refers to the spline interpolation routine used to break consumption totals into hourly values. The number here represents the order of the spline interpolation function used. The number must be an integer greater than zero; the default value is 1. The **normalize** parameter should always be set to **true**. If set to **false**, the time profile will be output in tons instead of percentage per hour, which will break most downstream modules.

- **References**

- Anderson, G. K., Sandberg, D. V., and Norheim, R. 2004. Fire Emission Production Simulator User's Guide, version 1.0. Produced for the Joint Fire Science Program (98-1-9-05).
- FEPS User's Guide: http://www.fs.fed.us/pnw/fera/feps/FEPS_users_guide.pdf.
- Fire Emission Production Simulator (FEPS), version 1.1; Fire and Environmental Research Applications Team: Seattle, WA, USA, 2005.

B.6 WRAP Time Profile

- **About WRAP Time Profile**

The WRAP time profile is a static lookup table that allocates daily emissions to hourly emissions. The WRAP profile is based on expert judgment (Air Sciences 2005). The hourly allocation is shown in **Table B-1**.

Table B-1. Percentage of total emissions in each hour in the WRAP Time Profile.

Hour	0	1	2	3	4	5	6	7	8	9	10	11
Percent	0.57	0.57	0.57	0.57	0.57	0.57	0.57	0.57	0.57	0.57	2	4
Hour	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**	**20**	**21**	**22**	**23**
Percent	7	10	13	16	17	12	7	4	0.57	0.57	0.57	0.57

- **How to Invoke WRAP Time Profile**

In the setup configuration file (/setup/*example.ini*), the user will need to set the **TIME_PROFILE** parameter.

```
TIME_PROFILE=WRAPTimeProfile
```

- **Configuration Options for WRAP Time Profile**

The WRAP time profile has no configuration options to set because it is simply the lookup table shown in Table B-1.

- **References**

- Air Sciences, Inc. (2005) Integrated assessment update and 2018 emissions inventory for prescribed fire, wildfire, and agricultural burning. Air Sciences, Inc. Denver, CO. http://www.wrapair.org/forums/fejf/documents/emissions/WGA2018report20051123.pdf.
- Air Sciences (2005) Integrated Assessment Update and 2018 Emissions Inventory for Prescribed Fire, Wildfire, and Agricultural Burning Report. Project 178-2. November 2005. Prepared for the Western Governors' Association/WRAP/Fire Emissions Forum. http://www.wrapair.org/forums/fejf/documents/emissions/WGA2018report20051123.pdf.

B.7 HYSPLIT Dispersion

- **About HYSPLIT Dispersion**

The HYSPLIT v5 model is used for computing both simple trajectories and dispersion of plumes. Advection (transport) and diffusion (spread) calculations are made in a Lagrangian framework, while concentrations are calculated on a fixed Eulerian grid. The HYSPLIT model simulates plume dispersion and transport over uneven terrain, and in changing wind regimes. It is standard practice in the meteorological modeling community to simulate smoke plume surface concentrations and plume footprints. HYSPLIT does not include complex chemical transformation algorithms within the model. Modeling of ozone production should be done with a chemical transport model, such as the Community Multiscale Air Quality model (CMAQ).

- **How to Invoke HYSPLIT Dispersion**

In the setup configuration file (setup/*example.ini*), the user will need to set the **DISPERSION** parameter.

```
DISPERSION=HYSPLITDispersion
```

- **Configuration Options for HYSPLIT Dispersion**

In the HYSPLIT configuration file (/modules/HYSPLIT/v5/*hysplit.ini*), a section is available for HYSPLIT Dispersion.

```
[HYSPLITDispersion]
HYSPLIT_BINARY = ${PACKAGE_DIR}/hymodelc
HYSPLIT2NETCDF_BINARY = ${PACKAGE_DIR}/hysplit2netcdf
SMOLDER_HEIGHT = 10.0
VERTICAL_METHOD = DATA
TOP_OF_MODEL_DOMAIN = 30000.0
VERTICAL_LEVELS = 10
VERTICAL_EMISLEVELS_REDUCTION_FACTOR = 1

USER_DEFINED_GRID = false
CENTER_LATITUDE = 55.0
CENTER_LONGITUDE = -125.0
WIDTH_LONGITUDE = 40.0
HEIGHT_LATITUDE = 20.0
SPACING_LONGITUDE = 0.25
SPACING_LATITUDE = 0.25

OPTIMIZE_GRID_RESOLUTION = false
MAX_SPACING_LONGITUDE = 0.50
MAX_SPACING_LATITUDE = 0.50
FIRE_INTERVALS = 0 100 200 500 1000
```

The **SMOLDER_HEIGHT** parameter defines the level at which smoldering emissions are emitted in HYSPLIT. The **VERTICAL_METHOD** parameter defines the vertical motion calculation method. The default method uses vertical velocity data directly from the meteorological model, but other common choices include isobaric (ISOB) and isentropic (ISEN) data. The number of vertical levels to include in the output is set by **VERTICAL_LEVELS**. The **USER_DEFINED_GRID** is used to set a custom HYSPLIT sampling grid, which is defined by a central geographic coordinate, dimensions (in degrees), and resolution (also in degrees). HYSPLIT is a very large and commonly used piece of software; an interested user should refer to the HYSPLIT User's Guide for more information.

- **References**
  - Draxler, R.R.; Hess, G.D. (1998) An overview of the HYSPLIT_4 modeling system for trajectories, dispersion, and deposition. *Aust. Met. Mag*., **47**, 295-308.
  - Draxler, R.R.; Hess, G.D. (1997) Description of the HYSPLIT_4 modeling system. NOAA Technical Memorandum ERL ARL-224, December, revised January 2004.
  - General HYSPLIT Information link: http://www.arl.noaa.gov/hysplit.php.
  - User's Guide: http://www.arl.noaa.gov/documents/reports/hysplit_user_guide.pdf.