

---

# STANDFIRE

*Release*

June 30, 2016



<b>1</b>	<b>STANDFIRE User Guide</b>	<b>3</b>
1.1	Overview of STANDFIRE . . . . .	3
1.2	Installation . . . . .	3
1.3	Basic Usage . . . . .	3
1.4	Advance Topics . . . . .	3
1.5	Tutorial 1: Interfacing with FVS . . . . .	3
<b>2</b>	<b>STANDFIRE API Reference</b>	<b>5</b>
2.1	fuels module . . . . .	5
2.2	intervene module . . . . .	11
<b>3</b>	<b>Indices and tables</b>	<b>13</b>
	<b>Python Module Index</b>	<b>15</b>
	<b>Index</b>	<b>17</b>



Contents:



---

## STANDFIRE User Guide

---

Contents:

### Overview of STANDFIRE

### Installation

### Basic Usage

### Advance Topics

### Tutorial 1: Interfacing with FVS

Use Suppose to generate a keyword file. Or use the following example .key

```
NOSCREEN
RANNSEED          0
!STATS
STDIDENT
STANDFIRE_example
DESIGN            -10          500          5          9
STDINFO           103          140          60.0          0.0          0.0          36.0
INVYEAR           2010
NUMCYCLE           10
Treedata
FMIN
END
STATS
SVS                0                0                0                15
FMIn
Potfire
FuelOut
BurnRept
MortRept
FuelRept
SnagSum
End
```

```
PROCESS
STOP
```

If don't have a FVS tree list file, then copy and paste the following text and save it to the same directory where the keyword file lives, give it the same prefix as the `.key` but with a `.tre` extension.

```
1  95      9PP 105   35              0 0
1  96      OPP 43    17              0 0
1  97      OPP 148   43              2 0
1  98      OPP 49    30              1 0
1  99      9PP 54    30              0 0
1 100      OPP 100   40              3 0
1 101      OPP 42    30              2 0
1 102      OPP 53    34              1 0
1 103      OPP 97    42              3 0
1 104      OPP 61    35              1 0
1 105      OPP 81    40              1 0
1 106      9PP 80    33              0 0
1 107      OPP 41    32              2 0
1 108      9PP 71    40              0 0
1 109      9PP 73    41              0 0
1 110      9PP 94    35              0 0
1 111      9PP 103   32              0 0
```

Once you have a keyword file and a tree list file in the same directory we can start to build a script to do some work.

First we import the `Fvsfuels` class from the `fuels` module.

```
>>> from standfire.fuels import Fvsfuels
```

Next create an instance of the class passes the desired variant as an argument and register the keyword file.

```
>>> stand_1 = Fvsfuel("iec")
>>> stand_1.set_keyword("/Users/standfire/fvs_exp/example.key")
TIMEINT not found in keyword file, default is 10 years
```

We get a message telling us that the `TIMEINT` keyword was not found in the keyword file. No problem, **STANDFIRE** automatically sets this value to 10 years.

```
>>> stand_1.keywords
{'TIMEINT': 10, 'NUMCYCLE': 10, 'INVYEAR': 2010, 'SVS': 15, 'FUELOUT': 1}
```

Notice the keys in the keywords dictionary. `TIMEINT` is the time interval of the FVS simulation in year, `NUMCYCLE` is the number of cycles, `INVYEAR` is the year of the inventory, and `SVS` and `FUELOUT` are there to check if these keywords are in the keyword file. If the `SVS` and `FUELOUT` keywords are not defined the keyword file then FVS will not calculate tree positions or fuel attributes. So be sure you add these to your keyword file before registering the `.key` with FVS. You can use *post processors*\* in *Suppose* to do so. `TIMEINT`, `NUMCYCLE`, and `INVYEAR` can be manually changed by calling setters for each. For instance, if you only want to calculate fuel attributes for trees during the year of the inventory then simply change the `NUMCYCLE` value in the keyword dictionary.

```
>>> stand_1.set_num_cycle(0)
>>> stand_1.keywords
{'TIMEINT': 10, 'NUMCYCLE': 0, 'INVYEAR': 2010, 'SVS': 15, 'FUELOUT': 1}
```

Now that we have our simulation parameters established, we startup FVS.

```
>>> stand_1.run_fvs()
```



---

## STANDFIRE API Reference

---

Contents:

### fuels module

Contents:

#### Fvsfuels

**class** `fuels.Fvsfuels` (*variant*)

Bases: `object`

A Fvsfuels object is used to calculate component fuels at the individual tree level using the Forest Vegetation Simulator. To create an instance of this class you need two items: a keyword file (.key) and tree list file (.tre) with the same prefix as the keyword file. If you don't already have a tree list file then you can use `'fuels.Inventory'` class to generate one.

**Parameters** `variant` (*string*) – FVS variant to be imported

#### Example:

A basic example to extract live canopy biomass for individual trees during year of inventory

```
>>> from standfire.fuels import Fvsfuels
>>> stand001 = Fvsfuels("iec")
>>> stand001.set_keyword("/Users/standfire/test/example.key")
TIMEINT not found in keyword file, default is 10 years
>>> stand001.keywords
{'TIMEINT': 10, 'NUMCYCLE': 10, 'INVYEAR': 2010, 'SVS': 15, 'FUELOUT': 1}
```

The keyword file is setup to simulate 100 years at a time interval of 10 years. Lets change this to only simulate the inventory year.

```
>>> stand001.set_num_cycles(0)
>>> stand001.keywords
{'TIMEINT': 10, 'NUMCYCLE': 0, 'INVYEAR': 2010, 'SVS': 15, 'FUELOUT': 1}
>>> stand001.run_fvs()
```

Now we can write the trees data frame to disk

```
>>> stand001.save_trees_by_year(2010)
```

---

**Note:** The argument must match one of the available variant in the PyFVS module. Search through `standfire/pyfvs/` to see all variants

---

**get\_simulation\_years()**

Returns a list of the simulated years

**Returns** simulated year

**Return type** list of integers

**get\_snags(*year*)**

Returns pandas data fram of the snags by indexed year

**Parameters** *year* (*int*) – simulation year of the data frame to return

**Returns** data frame of snags at indexed year

**Return type** pandas dataframe

---

**Note:** If a data frame for the specified year does not exist then a message will be printed to the console.

---

**get\_standid()**

Returns stand ID as defined in the keyword file of the class instance

**Returns** stand ID value

**Return type** string

**get\_trees(*year*)**

Returns pandas data fram of the trees by indexed year

**Parameters** *year* (*int*) – simulation year of the data frame to return

**Returns** data frame of trees at indexed year

**Return type** pandas dataframe

---

**Note:** If a data frame for the specified year does not exist then a message will be printed to the console.

---

**run\_fvs()**

Runs the FVS simulation

This method run a FVS simulation using the previously specified keyword file. The simulation will be paused at each time interval and the trees and snag data collected and appended to the fuels attribute of the Fvsfuels object.

**Example:**

```
>>> from standfire.fuels import Fvsfuels
>>> stand010 = Fvsfuels("iec")
>>> stand010.set_keyword("/Users/standfire/example/test.key")
>>> stand010.run_fvs()
>>> stand010.fuels["trees"][2010]
```

xloc	yloc	species	dbh	ht	crd	cratio	crownwt0	crownwt1	...
33.49	108.58	PIPO	19.43	68.31	8.77	25	33.46	4.3	
24.3	90.4	PIPO	11.46	56.6	5.63	15	6.55	2.33	
88.84	162.98	PIPO	18.63	67.76	9.48	45	75.88	6.89	
...									

**save\_all()**

Writes all data frame in the `fuels` attribute of the class to the specified working directory. Output file are .csv.

**save\_snags\_by\_year(year)**

Writes snag data frame at indexed year to .csv in working directory

**save\_trees\_by\_year(year)**

Writes tree data frame at indexed year to .csv in working directory

**set\_dir(wdir)**

Sets the working directory of a Fvsfuels object

This method is called by `Fvsfuels.set_keyword()`. Thus, the default working directory is the folder containing the specified keyword file. If you wish to store simulation outputs in a different directory then use this methods to do so.

**Parameters** `wdir` (*string*) – path/to/desired\_directory

**Example**

```
>>> from standfire.fuels import Fvsfuels
>>> test = Fvsfuels("emc")
>>> test.set_keyword("/Users/standfire/test/example.key")
```

Whoops, I would like to store simulation outputs elsewhere...

```
>>> test.set_dir("/Users/standfire/outputs/")
```

**set\_inv\_year(inv\_year)**

Sets inventory year for FVS simulation

**Parameters** `inv_year` (*int*) – year of the inventory

**set\_keyword(keyfile)**

Sets the keyword file to be used in the FVS simulation

**Date** 2015-8-12

**Authors** Lucas Wells

This method will initialize a FVS simulation by registering the specified keyword file (.key) with FVS. The working directory of a Fvsfuels object will be set to the folder containing the keyword file. You can manually change the working directory with `Fvsfuels.set_dir()`. This function will also call private methods in this class to extract information from the keyword file and set class fields accordingly for use in other methods.

**Parameters** `keyfile` (*string*) – path/to/keyword\_file. This must have a .key extension

**Example:**

```
>>> from standfire.fuels import Fvsfuels
>>> test = Fvsfuels("iec")
>>> test.set_keyword("/Users/standfire/test/example.key")
```

**set\_num\_cycles(num\_cyc)**

Sets number of cycles for FVS simulation

**Parameters** `num_cyc` (*int*) – number of simulation cycles

**set\_time\_int(time\_int)**

Sets time interval for FVS simulation

**Parameters** `time_int` (*int*) – length of simulation time step

## Inventory

**class** `fuels.Inventory`

Bases: `object`

This class contains methods for converting inventory data to FVS .tre format

This class currently does not read inventory data from an FVS access database. The FVS\_TreeInit database first needs to be exported as comma delimited values. Multiple stands can be exported in the same file, the `formatFvsTreeFile()` function will format a .tre string for each stand. All column headings must be default headings and unaltered during export. You can view the default format by importing this class and typing `FMT`. See the FVS guide <sup>1</sup> for more information regarding the format of .tre files.

### Example:

```
>>> from standfire import fuels
>>> toDotTree = fuels.Inventory()
>>> toDotTree.read_inventory("path/to/FVS_TreeInit.csv")
>>> toDotTree.format_fvs_tree_file()
>>> toDotTree.save()
```

### References

**convert\_sp\_codes** (*method*='2to4')

Converts species codes from 4 letter codes to 2 letter codes or vise versa

**Parameters** *method* (*string*) – must be either “2to4” or “4to2”

**crwratio\_percent\_to\_code** ()

Converts crown ratio from percent to ICR code

ICR code is described in the Essential FVS Guide on pages 58 and 59. This method should only be used if crown ratios values are percentages in the FVS\_TreeInit.csv. If you use this method before calling `formatFvsTreeFile()` then you must set the optional argument `cratioToCode` to `False`.

**filter\_by\_stand** (*stand\_list*)

Filters data by a list of stand IDs

**Parameters** *stand\_list* (*python list*) – List of stand ID to retain in the data. All other stands will be removed

**format\_fvs\_tree\_file** (*cratio\_to\_code*=`True`)

Converts data in FVS\_TreeInit.csv to FVS .tre format

This methods reads entries in the pandas data frame (`self.data`) and writes them to a formatted text string following FVS .tre data formatting standards shown in `FMT`. If multiple stands exist in `self.data` then each stand will written as a (key,value) pair in `self.fvsTreeFile` where the key is the stand ID and the value is the formatted text string.

**Parameters** *cratio\_to\_code* (*boolean*) – default = `True`

---

**Note:** If the `crwratio_percent_to_code()` methods has been called prior to call this methods, then the `cratio_to_code` optional argument must be set to `False` to prevent errors in crown ratio values.

---

### Example:

---

<sup>1</sup> Gary E. Dixon, Essential FVS: A User's Guide to the Forest Vegetation Simulator Tech. Rep., U.S. Department of Agriculture , Forest Service, Forest Management Service Center, Fort Collins, Colo, USA, 2003.

```

>>> toDotTree.format_fvs_tree_file()
>>> toDotTree.fvsTreeFile['Stand_ID_1']
5   1   5      OPP 189    65      3      0 0
5   2  15      OPP 110    52      2      0 0
5   3   5      OPP 180    64      5      0 0
5   4  14      OPP 112    56      3      0 0
5   5   6      OPP 167    60      4      0 0
5   6   5      OPP 190    60      5      0 0
5   7   7      OPP 161    62      3      0 0
5   8  86      OPP 46     37      1      0 0
5   9  10      OPP 130    50      2      0 0
5  10  5       OPP 182    60      3      0 0
5  11  8       OPP 144    50      3      0 0
6   1  16      OPP 107    42      4      0 0
6   2 109      OPP 41     27      2      0 0
...

```

**get\_fvs\_cols()**

Get list of FVS standard columns

**Returns** FVS standard columns

**Return type** list of strings

**get\_stands()**

Returns unique stand IDs

**Returns** stand IDs

**Return type** list of strings

**Example:**

```

>>> toDotTree.get_stands()
['BR', 'TM', 'SW', HB']

```

**print\_format\_standards()**

Print FVS formatting standards

The FVS formatting standard for .tre files as described in the Essential FVS Guide is stored in FMT as a class attribute. This method is for viewing this format. The keys of the dictionary are the column headings and values are as follows: 0 = variable name, 1 = variable type, 2 = column location, 3 = units, and 4 = implied decimal place.

**Example:**

```

>>> toDotTree.print_format_standards()
{'Plot_ID'      : ['ITRE',      'integer',  [0,3],   None,    None],
 'Tree_ID'      : ['IDTREE2',   'integer',  [4,6],   None,    None],
 'Tree_Count'   : ['PROB',      'integer',  [7,12],  None,    None],
 'History'      : ['ITH',       'integer',  [13,13], 'trees', 0   ],
 'Species'      : ['ISP',       'alphanum', [14,16], None,    None],
 'DBH'          : ['DBH',       'real',     [17,20], 'inches', 1   ],
 'DG'           : ['DG',       'real',     [21,23], 'inches', 1   ],
 'Ht'           : ['HT',       'real',     [24,26], 'feet',  0   ],
 'HtTopK'       : ['THT',      'real',     [27,29], 'feet',  0   ],
 'HTG'          : ['HTG',      'real',     [30,33], 'feet',  1   ],
 'CrRatio'      : ['ICR',      'integer',  [34,34], None,    None],
 'Damage1'      : ['IDCD(1)',   'integer',  [35,36], None,    None],
 'Severity1'    : ['IDCD(2)',   'integer',  [37,38], None,    None],
 'Damage2'      : ['IDCD(3)',   'integer',  [39,40], None,    None],

```

```

'Severity2'      : ['IDCD(4)', 'integer', [41,42], None, None],
'Damage3'       : ['IDCD(5)', 'integer', [43,44], None, None],
'Severity3'      : ['IDCD(6)', 'integer', [45,46], None, None],
'TreeValue'      : ['IMC', 'integer', [47,47], None, None],
'Prescription'   : ['IPRSC', 'integer', [48,48], None, None],
'Slope'          : ['IPVARS(1)', 'integer', [49,50], 'percent', None],
'Aspect'         : ['IPVARS(2)', 'integer', [51,53], 'code', None],
'PV_Code'        : ['IPVARS(3)', 'integer', [54,56], 'code', None],
'TopoCode'       : ['IPVARS(4)', 'integer', [57,59], 'code', None],
'SitePrep'       : ['IPVARS(5)', 'integer', [58,58], 'code', None],
'Age'            : ['ABIRTH', 'real', [59,61], 'years', 0 ]}

```

See page 61 and 62 in the Essential FVS Guide.

### **read\_inventory** (*fname*)

Reads a .csv file containing tree records.

The csv must be in the correct format as described in FMT. This method check the format of the file by calling a private method `_is_correct_format()` that raises a value error.

**Parameters** *fname* (*string*) – path to and file name of the Fvs\_TreeInit.csv file

### **Example:**

```

>>> from standfire import fuels
>>> toDotTree = fuels.Inventory()
>>> toDotTree.readInventory("path/to/FVS_TreeInit.csv")
>>> np.mean(toDotTree.data['DBH'])
9.0028318584070828

```

The `read_inventory()` method stores the data in a pandas data frame. There are countless operations that can be performed on these objects. For example, we can explore the relationship between diameter and height by fitting a linear model

```

>>> import statsmodels.formula.api as sm
>>> fit = sm.ols(formula="HT ~ DBH", data=test.data).fit()
>>> print fit.params
Intercept    19.688167
DBH           2.161420
dtype: float64
>>> print fit.summary()
OLS Regression Results
=====
Dep. Variable:          Ht      R-squared:                0.738
Model:                  OLS      Adj. R-squared:            0.736
Method:                 Least Squares      F-statistic:          351.8
Date:                  Tue, 07 Jul 2015      Prob (F-statistic):    3.77e-38
Time:                  08:32:02      Log-Likelihood:        -407.10
No. Observations:      127      AIC:                  818.2
Df Residuals:          125      BIC:                  823.9
Df Model:               1
Covariance Type:       nonrobust
=====
coef    std err          t      P>|t|      [95.0% Conf. Int.]
-----
Intercept    19.6882     1.205    16.338     0.000     17.303     22.073
DBH           2.1614     0.115    18.757     0.000     1.933     2.389
=====
Omnibus:                 2.658      Durbin-Watson:           0.995
Prob(Omnibus):           0.265      Jarque-Bera (JB):        2.115

```

Skew:	-0.251	Prob (JB) :	0.347
Kurtosis:	3.385	Cond. No.	23.8
=====			

Read more about pandas at <http://pandas.pydata.org/>

#### **save** (*outputPath*)

Writes formatted fvs tree files to specified location

If multiple stands exist in the FVS\_TreeInit then the same number of files will be created in the specified directory. The file names will be the same as the Stand\_ID with a .tre extension.

**Parameters** **outputPath** (*string*) – directory to store output .tre files

**Note:** This method will throw an error if it is called prior to the `format_fvs_tree_file()` method.

#### **set\_FVS\_variant** (*var*)

Sets FVS variant. This is need if converting from USDA plant symbols (PSME) to alpha codes (DF). If so, then all stand you wish to process must be of the same FVS variant

**Parameters** **var** (*string*) – FVS variant ('iec', 'emc' ...)

## intervene module

The intervene module is a collection of treatment algorithms

Contents:

### SpaceCrowns

**class** `intervene.SpaceCrowns` (*trees*)

Bases: `intervene.BaseSilv`

**Variables** **crown\_space** – instance variable for crown spacing; initial value = 0

**add\_to\_treatment\_collection** (*treatment, ID*)

Adds treatment to static class attribute in `intervene.BaseSilv()`

**clear\_treatment\_collection** ()

Deletes all treatment currently in the treatment collection class attribute

**get\_distance** (*tree\_a, tree\_b*)

Calculate the distance between two trees

Uses Pythagoras' theorem to calculate distance between two tree crowns in units of input data frame

**Parameters**

- **tree\_a** (*int*) – indexed row of tree a in Pandas data frame
- **tree\_b** (*int*) – indexed row of tree b in Pandas data frame

**Returns** distance between two crowns in units of input data frame

**Return type** float

**get\_extent** ()

Returns bounding box of tree coordinates

**Returns** [min\_x, min\_y, max\_x, max\_y]

**Return type** list of floats

**Examples**

```
>>> from standfire.intervene import SpaceCrowns
>>> space = SpaceCrowns("/Users/standfire/test_trees.csv")
>>> bbox = space.get_extent()
>>> bbox
[1.3, 3.5, 63.1, 61.4]
```

**get\_treatment\_options()**

Returns dictionary of treatment options

**Returns** treatment option codes and description

**Return type** dictionary

**get\_trees()**

Returns the trees data frame of the object

**Returns** trees data frame

**Return type** Pandas.DataFrame

**set\_crown\_space(crown\_space)**

Sets spacing between crowns for the treatment

**Parameters** **crown\_space** (*float*) – crown spacing in units of input data frame

**set\_extent(min\_x, min\_y, max\_x, max\_y)**

Sets extent instance variable

**Parameters**

- **min\_x** (*float*) – minimum x coordinate
- **min\_y** (*float*) – minimum y coordinate
- **max\_x** (*float*) – maximum x coordinate
- **max\_y** (*float*) – maximum y coordinate

---

**Note:** `set_extent` is automatically called by `BaseSilv()` constructor

---

**treat()**

Treatment algorithm for removing trees based on input crown spacing

---

**Todo**

Optimize algorithm by incorporating `search_rad`.

---

---

**Todo**

split this function into 3

---

**treatment\_collection = {}**



---

## Indices and tables

---

- `genindex`
- `modindex`
- `search`



**f**

`fuels`, [5](#)

**i**

`intervene`, [11](#)



## A

`add_to_treatment_collection()` (`intervene.SpaceCrowns` method), 11

## C

`clear_treatment_collection()` (`intervene.SpaceCrowns` method), 11

`convert_sp_codes()` (`fuels.Inventory` method), 8

`crwratio_percent_to_code()` (`fuels.Inventory` method), 8

## F

`filter_by_stand()` (`fuels.Inventory` method), 8

`format_fvs_tree_file()` (`fuels.Inventory` method), 8

`fuels` (module), 5

`Fvsfuels` (class in `fuels`), 5

## G

`get_distance()` (`intervene.SpaceCrowns` method), 11

`get_extent()` (`intervene.SpaceCrowns` method), 11

`get_fvs_cols()` (`fuels.Inventory` method), 9

`get_simulation_years()` (`fuels.Fvsfuels` method), 6

`get_snags()` (`fuels.Fvsfuels` method), 6

`get_standid()` (`fuels.Fvsfuels` method), 6

`get_stands()` (`fuels.Inventory` method), 9

`get_treatment_options()` (`intervene.SpaceCrowns` method), 12

`get_trees()` (`fuels.Fvsfuels` method), 6

`get_trees()` (`intervene.SpaceCrowns` method), 12

## I

`intervene` (module), 11

`Inventory` (class in `fuels`), 8

## P

`print_format_standards()` (`fuels.Inventory` method), 9

## R

`read_inventory()` (`fuels.Inventory` method), 10

`run_fvs()` (`fuels.Fvsfuels` method), 6

## S

`save()` (`fuels.Inventory` method), 11

`save_all()` (`fuels.Fvsfuels` method), 6

`save_snags_by_year()` (`fuels.Fvsfuels` method), 7

`save_trees_by_year()` (`fuels.Fvsfuels` method), 7

`set_crown_space()` (`intervene.SpaceCrowns` method), 12

`set_dir()` (`fuels.Fvsfuels` method), 7

`set_extent()` (`intervene.SpaceCrowns` method), 12

`set_FVS_variant()` (`fuels.Inventory` method), 11

`set_inv_year()` (`fuels.Fvsfuels` method), 7

`set_keyword()` (`fuels.Fvsfuels` method), 7

`set_num_cycles()` (`fuels.Fvsfuels` method), 7

`set_time_int()` (`fuels.Fvsfuels` method), 7

`SpaceCrowns` (class in `intervene`), 11

## T

`treat()` (`intervene.SpaceCrowns` method), 12

`treatment_collection` (`intervene.SpaceCrowns` attribute), 12